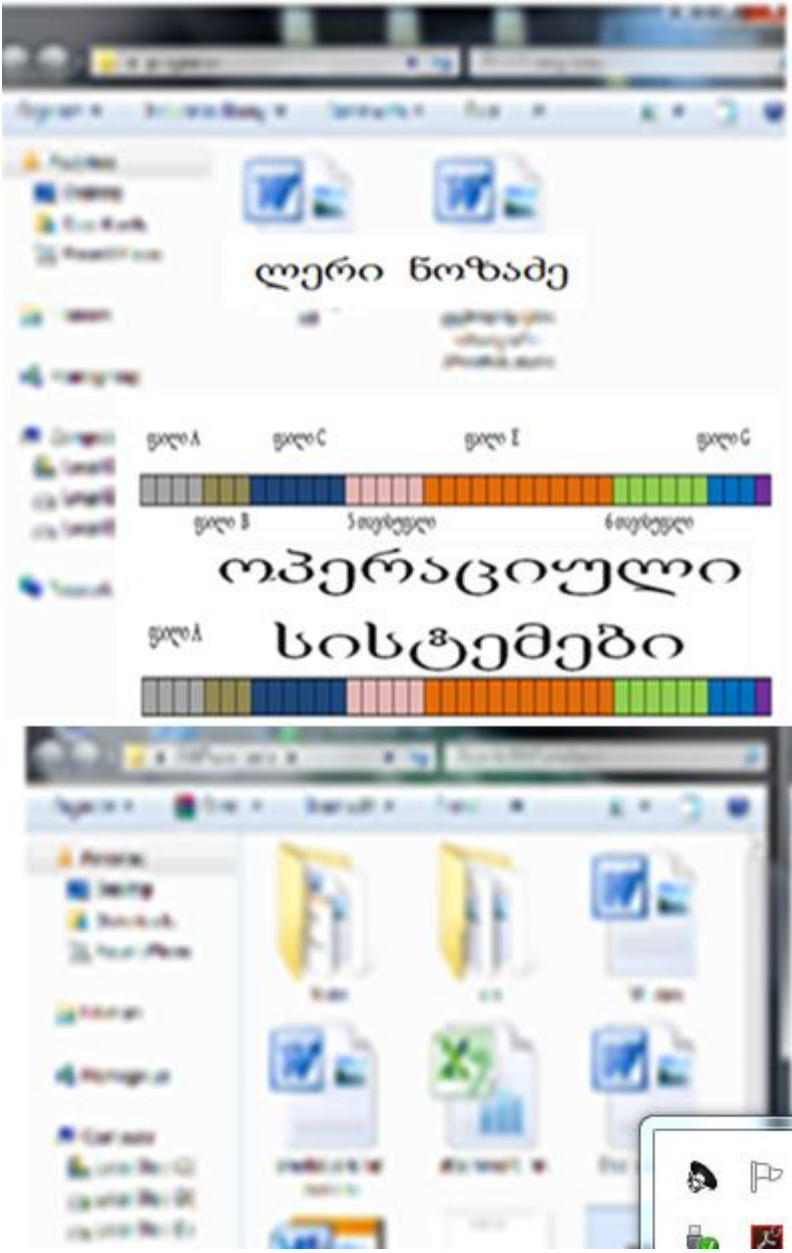

ლერი ნოზაძე
ოპერაციული
სისტემები

ლერი ნოზაძე

ოპერაციული სისტემები

დამხმარე სახელმძღვანელო

თბილისი 2015

დამხმარე სახელმძღვანელო „ოპერაციული სისტემები“ შედგენილია ე.ს. ტანენბაუმის, ვ. ბ. კომიაგინის , ი.ა. რივკინის და სხვა ცნობილი ავტორების ნაშრომების მიხედვით.

დამუშავებულია ისეთი ძირითადი თეორიული და პრაქტიკული საკითხები, როგორიცაა: ოპერაციული სისტემის კონცეფციების ძირითადი ცნებები და არქიტექტურული თავისებურებანი, დაგეგმვის დონეები და ალგორითმებისათვის წაყენებული კრიტერიუმები და მოთხოვნები, დამისამართება და კავშირის მიმართულებები, საკითხი სინქრონიზაციის აუცილებლობის შესახებ და მოთხოვნები სინქრონიზაციის ალგორითმის მიმართ, ურთიერთბლოკირება და დაბლოკვის პირობები, რესურსები და მეხსიერების მართვა, პირდაპირი წვდომა მეხსიერებასთან, წყვეტის შესახებ და ფაილური სისტემის რეალიზაცია, ბუფერის პროგრამული მართვა და გვერდების ცხრილის ინვერტირება, ინფორმაციის დაცულობა თანამედროვე კომპიუტერულ ქსელებში და სხვა, რომელთა ცოდნა აუცილებელია ამ სფეროთი დაინტერესებული სპეციალისტებისა და სტუდენტებისათვის.

უაკ(UDC) 004.451

ნ-811

წარმოდგენილი დამხმარე სახელმძღვანელო განხილულია სამცხე-ჯავახეთის სახელმწიფო უნივერსიტეტის ინჟინერიის, აგრარულ და საბუნებისმეტყველო მეცნიერებათა ფაკულტეტის საბჭოს სხდომაზე ოქმი # 22/04/01 22 აპრილი, 2014 წ. და შეესაბამება ინფორმაციული ტექნოლოგიების სპეციალობის სტუდენტებისთვის ამჟამად მოქმედ პროგრამას.

რედაქტორი:

გიორგი მესხი, ფიზიკის დოქტორი, პროფესორი

რეცენზენტები:

ლელა წითაშვილი, ინჟინერიის დოქტორი, ასოცირებული პროფესორი
ნუგზარ გუაჩიძე, ტექნიკის დოქტორი, ასოცირებული პროფესორი
დავით გოგალაძე, ფიზიკის დოქტორი.

©

ISBN 978-9941-0-6796-9

შესავალი

Windows-ების ისტორია იწყება 1986 წლიდან, როდესაც სისტემის პირველი ვერსია Windows 1.0 მოგვევლინა. ის პროგრამების ერთობლიობას, პროგრამების კომპლექტს წარმოადგენდა, რომელიც არსებული ოპერაციული სისტემის გაფართოების შესაძლებლობას იძლეოდა და მუშაობისთვისაც მოხერხებული იყო. შემდეგ Windows 2.0 ვერსია დამუშავდა, თუმცა მან განსაკუთრებული პოპულარობა ვერ მოიპოვა. 1990 წელს ვერსია - Windows 3.0 გამოვიდა, რომელიც ფართოდ გავრცელდა პერსონალურ კომპიუტერებზე.

შემდგომი ვერსიები სისტემის მაღალ საიმედოობას, მულტიმედიის საშუალებისადმი მხარდაჭერას (Windows 3.1) და კომპიუტერულ ქსელში (Windows 3.11) მუშაობას უზრუნველყოფდნენ.

Windows-ების დამუშავების პარალელურად Microsoft კომპანიამ 1988 წელს მუშაობა დაიწყო ახალ ოპერაციულ სისტემაზე, Windows NT სახელწოდებით. ძირითადი ამოცანა გამოვლინდა ისეთი სისტემის შექმნაში, რომელიც უმაღლეს საიმედოობას და ქსელთან მუშაობის დროს ეფექტურ მხარდაჭერას უზრუნველყოფდა. ამასთან, Windows NT-ს ინტერფეისი არ განსხვავდებოდა Windows 3.0-ის ინტერფეისისგან. 1992 წელს გაჩნდა ახალი ვერსია, Windows NT 3.0 , ხოლო 1994 წელს - Windows NT 3.5.

1995 წელს გამოვიდა Windows95, რაც ახალ ეტაპად იქცა Windows-ების და საერთოდ, პერსონალური კომპიუტერების ისტორიაში. Windows 3.1-თან შედარებით, მნიშვნელოვნად

შეიცვალა ინტერფეისი, ამაღლდა პროგრამების მუშაობის სიჩქარე. ახალმა ოპერაციულმა სისტემამ საშუალება გააჩინა ავტომატურად მომხდარიყო პერსონალური კომპიუტერების დამატებითი მოწყობილობების მომართვა ისე, რომ თავიდან აგვეცილებინა შეუთავსებლობები ურთიერთზემოქმედების დროს(ეს დაბალანსებული სისტემის ნიშანია). გარდა ამისა, Windows95-ში გაკეთდა პირველი ნაბიჯები ინტერნეტთან მხარდაჭერის რეალიზებისათვის. ინტერნეტთან, რომელიც იმ პერიოდის პირმშოა. ის, ის იყო იბადებოდა და 1996 წელს გამოჩნდა სერვერული ოპერაციული სისტემის Windows NT 4.0-ს საოჯახო ვერსია, რომელსაც ისევ Windows 95-ის ინტერფეისი ჰქონდა.

1998 წელს გაჩნდა Windows 98, როგორც მნიშვნელოვნად გადამუშავებული სტრუქტურა, ვიდრე Windows 95 იყო. ამ ახალ ვერსიაში დიდი ყურადღება ეთმობოდა ინტერნეტში მუშაობას, აგრეთვე თანამედროვე ქსელური გარემოსადმი მხარდაჭერას. ასევე გაჩნდა მხარდაჭერა რამდენიმე მონიტორთან მუშაობის მიმართაც.

შემდეგ ეტაპს Windows-ების განვითარებაში Windows 2000-ის გაჩენა წარმოადგენს. ეს სისტემა დამუშავდა Windows NT-ის ბაზაზე და მისგან მემკვიდრეობით მაღალი საიმედოობა და გარეგანი ზემოქმედებისაგან(ჩარევისაგან) ინფორმაციის დაცულობა მიიღო. გამოშვებული იქნა ორი ვერსია: Windows 2000 Server და Windows 2000 Professional - სამუშაო სადგურისათვის, რომელიც მრავლად დაამონტაჟეს(დააყენეს) საოჯახო კომპიუტერებზე.

შემდეგი იყო Windows ME (Millenium Edition) სისტემა. არსებითად, Windows 98-ის გაფართოებული ვერსია, მულტიმედიისათვის მხარდაჭერის გაუმჯობესებული გამოცემა. მიჩნეულია, რომ ეს სისტემა ყველაზე წარუმატებელი ვერსიაა Windows-ებში, მუშაობის დროს არასტაბილურობით გამოირჩეოდა, ხშირად ავარიულად შეჩერებები ახასიათებდა და ასე დაასრულა თავისი არსებობა.

2001 წელს მოგვევლინა eXPerience (გამოცდილება) - ახალი ოპერაციული სისტემა, რომელიც ეფუძვნება Windows NT-ის ბირთვს და ამიტომ მაღალი სტაბილურობით და ეფექტურობით გამოირჩევა მუშაობის დროს, Windows-ების წინა ვერსიებთან შედარებით. მასში ასევე სერიოზულად არის გადამუშავებული გრაფიკული ინტერფეისი. შემოტანილია მხარდაჭერები ახალი ფუნქციებისა და პროგრამების მიმართ.

გასაოცარი არაა, რომ Windows XP ისეთი წარმატებული გამოდგა, რომ 2008 წელსაც კი 70%-ს იკავებდა ოპერაციული სისტემების ბაზარზე. Windows-ისთვის გამოუშვეს განახლების სამი პაკეტი (Service Pack), რომელთაგან მესამე 2008 წლის აპრილში გამოვიდა. თითოეული პაკეტი აფართოვებდა ოპერაციული სისტემის შესაძლებლობებს, ასწორებდა, აღმოფხვრიდა შეცდომებს, უფრო საიმედოსა და დაცულს ხდიდა სისტემას.

2003 წელს ახალი ვერსია Windows Server 2008 გამოვიდა, რომელმაც Windows 2000 შეცვალა. ცოტაოდენი ხნის შემდეგ გამოვიდა მისი განახლებაც, Windows Server 2008 R2 სახელწოდებით. მან ახალი სტანდარტები შემოგვთავაზა

მწარმოებლობისა და საიმედოობის საკითხებში. ის გახდა Microsoft-ის ყველაზე წარმატებული სერვერული სისტემა.

ჯერ კიდევ Windows XP-ს გამოსვლამდე, Microsoft კომპანია აქტიურად ამუშავებდა ოპერაციული სისტემის ახალ ვერსიას, რომელსაც ჰქონდა კოდური სახელწოდება Windows Longhorn, შემდეგ კი ეს სახელწოდება Windows Vista-თი შეიცვალა. ეს ახალი სისტემა გაჩნდა 2007 წელს. საოჯახო კომპიუტერებისათვის მისი ოპერაციული სისტემის მყარი ტრადიცია დაფუძნებული იყო მწარმოებლურ და საიმედო ბირთვზე Windows Server 2003 (ისე, როგორც Windows XP იყო დაფუძნებული Windows NT ბირთვზე), Vista-ში კარდინალურად შეიცვალა სამომხმარებლო ინტერფეისი, სერიოზულად გაუმჯობესდა უსაფრთხოების სისტემა. გაჩნდა ახალი შესაძლებლობების და ფუნქციების დიდი არეალი, მაგრამ ამ ბრწყინვალე მონაცემების მიუხედავად, სისტემას ცივად შეხვდნენ. ბევრმა ის „ჩავარდნადაც“ მიიჩნია.

ვინდოუსების პირველი ვერსიები

მუდმივად უმჯობესდებოდა კომპიუტერების ტექნიკური დონე, რასაც, შეუძლებელია გავლენა არ მოეხდინა მათი პროგრამული უზრუნველყოფის სრულყოფაზე, და პირველ რიგში, ოპერაციული სისტემის სრულყოფაზე, რადგანაც სწორედ ისაა ის სისტემური პროგრამა, რომლის გარეშეც კომპიუტერის მუშაობა შეუძლებელია.

დღეისათვის, ყველაზე პოპულარული ოპერაციული სისტემა ვინდოუსია. თავდაპირველად ის შეიქმნა, როგორც გრაფიკული გარსი MS DOS-ისათვის (Microsoft Disc Operations System), ხოლო შემდგომ სრულყოფილი ოპერაციული სისტემის სახე მიიღო. Windows-ში ჩადებული უმთავრესი იდეა - ინფორმაციის წარმოდგენის რეალურობაა.

1983 წ. 10/X საერთაშორისო კომპიუტერულ გამოფენაზე COMDEX კორპორაცია Microsoft-მა დააანონსა MS DOS-ზე აწყობილი Windows-ის მოდელი, სადაც პირველად იყო გამოყენებული გრაფიკული ფანჯრების გარემოს კონცეფცია, რაც მნიშვნელოვნად აადვილებს მომხმარებლის კომპიუტერთან მუშაობას. უნდა აღინიშნოს, რომ იმ დროისათვის Windows-ი ერთადერთი ამგვარი ჩანაფიქრი არ ყოფილა. უბრალოდ, დანარჩენი მცდელობები იმისა, რომ კომპიუტერები სრულყოფილი გაეხადათ მუშაობის თვალსაჩინოობის თვალსაზრისით, - წარუმატებლად დამთავრდა, რადგან მათი MS DOS-თან მისადაგება ვერ მოხერხდა.

სისტემა Windows 1.01 ფაქტობრივად, 1985 წ. 20/XI. გამოვიდა. კომპლექტს ჰქონდა რამდენიმე სპეციალური

უტილიტი (როგორიცაა საათი, კალკულატორი, MS Notepad და უფრო მეტად სრულყოფილი MS Write). მოცემული სისტემა ითვალისწინებდა რამდენიმე პროგრამასთან მუშაობას და ერთიდან მეორეზე გადართვას ისე, რომ არ ჩაგვეხურა ერთი და გაგვეშვა სხვა პროგრამა, შეეძლო შეესრულებინა DOS-ის განსაზღვრეული ფუნქციები, ეთავსებოდა CGA და EGA - გრაფიკულ ადაპტერებს და ვიდეო ფირტფიტას Hercules Graphic Card - სტანდარტს, იძლეოდა ფანჯრული გარემოს თაგვის მეშვეობით მართვის შესაძლებლობას და ეთავსებოდა რამდენიმე სახის მატრიცულ პრინტერს. მაგრამ ფანჯრების გადაფარვა დაუშვებელი იყო.

ამასთანავე, ამ სისტემას ჰქონდა მრავალი ხარვეზიც: ინტერნეტის არ არსებობა მნიშვნელოვნად ართულებდა განახლების პროცესს. საბოლოო ანგარიშში Windows 1.0-მა დიდად გავრცელება ვერ ჰპოვა ბაზარზე, ძირითადად, საკმარისი ოდენობით გამოყენებების უქონლობის გამო. შავ-თეთრი მონიტორიც არც ისე სახარბიელო იყო, რადგან სისტემის ინტერფეისის სრულყოფილად რეალიზებას ვერ ახდენდა და, ამავე დროს, მაღალი მოთხოვნები გააჩნდა სისტემის რესურსებისადმი.

Windows 2.03. გადამფარავი ფანჯრების სისტემის რეალიზება Windows-ის ახალ ვერსიაში მოხერხდა, რომელიც გამოვიდა 1987წ. 7 დეკემბერს. დადებითი თვისება (ხარისხი)Windows 2.03-თვის არის ისიც, რომ შესაძლებელი გახდა 80286 პროცესორის დამცავი რეჟიმის გამოყენება. ის უფრო სრულყოფილი იყო და შესაძლებელს ხდიდა „გამოვსულიყავით“ ძირითადი მეხსიერების მოცულობის საზღვრებიდან, რომელიც, როგორც ცნობილია, DOS-თვის 640 kb იყო. თავად სისტემის

სტრუქტურა და არქიტექტურა არ იყო ძლიერ შეცვლილი. დამუშავდა ახალი აღჭურვილობის მხარდამჭერი პროგრამა. მაგ: „VGA სტანდარტის ვიდეოკარტა“, დიდი მოცულობის მქონე მყარი დისკები და მოდემები. პროგრამული უზრუნველყოფის კომპლექტში დაემატა ტერმინალი, რომელიც მოდემის მეშვეობით რამდენიმე კომპიუტერთან კავშირის დამყარების შესაძლებლობას იძლეოდა. მაგრამ, როგორც წინა ვერსიაში, აქაც Windows 2.03-მა გამოავლინა მაღალი მოთხოვნები აპარატული ნაწილის მიმართ და იკავებდა დისკური სივრცის 5 მეგაბაიტამდე მოცულობას, რამაც მის ფართოდ გავრცელებაზე მნიშვნელოვნად უარყოფითად იმოქმედა.

Windows 286. 1987 წ. 9 დეკემბერს MS უშვებს Windows 286, რომელიც Windows 2.0-ის რედაქტირებული ვერსია იყო. ახალი ტიპის პროცესორის - SP- Intel-ის ოპტიმირებას ახდენდა. მისი მთავარი თავისებურება იყო ერთდროულად რამდენიმე DOS პროგრამის გაშვების შესაძლებლობა. აღნიშნული სისტემა თავის თავში მოიცავს Windows 3.0-ის ფუნქციების უმრავლესობას.

Windows 3.0 ნამდვილი ნახტომი ოპერაციული სისტემების სფეროში იყო 1990 წ. 22 მაისს Windows 3.0-ის გამოსვლა! რომელიც გაფართოებული რეჟიმის გამოყენების დროს, ფაქტობრივად სრულად აკონტროლებდა DOS-ის მუშაობას. ამ ვერსიაში პროგრამების დისპეტჩერი და პიქტოგრამები იყო დამუშავებული. აგრეთვე გააჩნდა ფაილების დისპეტჩერი და სრულყოფილი მართვის პანელი, რომელიც მრავალი გამოსახულების აგების ოპერირებას ახდენდა. პროგრამისტებზე ორიენტირებულმა ამ სიახლემ Windows 3.0-ის ბაზარზე ლიდერობა განაპირობა. ეს სრულად ფუნქციონალური

16 ბიტიანი ოპერაციული სისტემა გარდამავალ ვარიანტს წარმოადგენდა Windows 2x-დან - Windows 3.1-სკენ. DOS-ის ფაილური გარსი სრულიად შეიცვალა დისპეტჩერული პროგრამიმთ. მნიშვნელოვანი ყურადღება პლატფორმის დამუშავების დროს ინტერფეისს დაეთმო, რის გამოც მისმა მრავალმა ელემენტმა, მათ შორის ფანჯარამ და ფუნქციონალურმა კლავიშებმა შეიძინეს ფსევდო სამგანზომილებიანი გაფორმება VGA-ს გაფართოებული ფერთა პალიტრის გამოყენებით. Windows 3.0 მოიცავდა მოდერნიზებულ დახმარების სისტემას, ჰიპერგზავნილების გამოყენებით, რაც აადვილებდა სხვადასხვა მინიშნებების თემატურ განყოფილებებზე წვდომას. საცნობარო სისტემის ორგანიზებით ეს პრინციპი გამოიყენება უფრო გვიანდელ ვერსიებშიც... თითქმის Windows 98-ის ჩათვლით.

MS-მა მნიშვნელოვნად გააფართოვა პროგრამების პაკეტი, რომელიც მიეწოდებოდა სისტემასთან ერთად. სტრანდარტულ კომპლექტებთან ერთად Windows 3.0 თამაშების პაკეტსაც შეიცავდა („პასიანსი“,“კატა“, „თავშალი“ და სხვა)

Windows 3.1 გამოვიდა 1992 წ. 6 აპრილს. ეს ითვლებოდა ერთ-ერთ ყველაზე პოპულარულ ვერსიად. მასში შესწორდა წინა ვერსიებში შემჩნეული შეცდომები. აქცენტი გაკეთდა სტაბილურობაზე, დაემატა რამდენიმე ახალი შესაძლებლობა, მათ შორის მასშტაბირებადი შრიფტები True Type.

Windows 3.1 – სრულფასოვანი ოპერაციული სისტემაა. უმრავლეს შემთხვევაში გამოიყენება გაფართოვებულ რეჟიმში, თუმცა ასევე არსებობს სტანდარტულიც, მაგრამ მხოლოდ DOS-ის გამოყენებებთან თავსებადობისათვის. შეტანა-გამოტანის

ოპერაციებისათვის ჯერ კიდევ გამოიყენება DOS, მაგრამ უკვე Windows-ის კონტროლის ქვეშ. შეიქმნა კორპორაციული მრავალამოცანიანი რეჟიმი, რაც შესაძლებლობას აძლევს სისტემას ერთდროულად შეასრულოს Windows-ის და DOS-ის ამოცანები. დრაივერების გამოყენება უზრუნველყოფს Windows-ის გამოყენებებთთან სხვადასხვა მისაერთებელი მოწყობილობების თავსებადობას. ამ ვერსიაში Microsoft-მა მოახდინა თავისი ახალი დამუშავების - Drag & Drop ტექნოლოგიის დემონსტრირება. (ეს სიტყვა-სიტყვით ნიშნავს „ამოაძრე და გადააგდე“) და DLE ტექნოლოგიის დემონსტრირება, რაც ობიექტების დანერგვასა და დაკავშირებას ეხებოდა.

სისტემას შეეძლო ემუშავა პრაქტიკულად ყველანაირ მონაცემებზე. პლატფორმის შემადგენლობაში ახლა უკვე შევიდა ისეთი სასარგებლო უტილიტები, როგორიცაა ფონოგრაფი და უნივერსალური მაგნიტოფონი. დაემატა მხარდაჭერა სხვადასხვა ხმოვანი ფირფიტებისა და სხვა პერიფერიული მოწყობილობების მიმართ. Windows 3.1-ის ფართოდ გავრცელებას განაპირობებდა მისი მრავალრიცხოვანი კომპონენტები, რომლებიც გამიზნული იყო სხვადასხვა კვალიფიკაციის მომხმარებლების მოხერხებული სამუშაო გარემოს შესაქმნელად.

Windows 3.1-ის გამოსვლის შემდეგ Microsoft-მა ბაზარი გაყო ორ მსხვილ სეგმენტად: 1. საოჯახო და საოფისე პერსონალური კომპიუტერების ბაზარი, რომელი კატეგორიის საზღვრებშიც MS-მა აქცენტი გააკეთა მაქსიმალურ თავსებადობასა და მწარმოებლურობაზე.2. მაღალმწარმოებლური სამუშაო სადგურებისა და სერვერების ბაზარი. ამ კომპიუტერების მონაცემებისაგან პირველ რიგში მოითხოვება მაქსიმალური საიმედოობა და მდგრადობა (რაც ამავე დროს

არაიშვიათად ამცირებს მწარმოებლობას.) ამ პრინციპით დამუშავდა ვერსია Windows NT – New Technology.

ბაზრის სხვადასხვა სეგმენტისათვის გამოდიოდა სპეციალიზირებული პროდუქტები. აქ უკვე საყურადღებოა ოპერაციული სისტემის განვითარება საოჯახო დანიშნულებისად და საოფისედ.

სამომხმარებლო Windows-ები

Windows 95 პირველი სამომხმარებლო ოპერაციული სისტემა, რომელიც მზის სინათლეზე პროგრამული უზრუნველყოფის ბაზრის გაყოფის შემდეგ გამოვიდა, - Windows 95 იყო. ეს მოხდა 1995 წ. 24 აგვისტოს. და მომდევნო თორმეტ თვეში ამ პლატფორმის 40 მლნ-ზე მეტი ლიცენზირებული ასლი გაიყიდა.

Windows 95-ის ოფიციალური სისტემური მოთხოვნები შემდეგია: პროცესორი - არანაკლები 380DX.(პრაქტიკულად რეკომენდებული 486DX.) ოპერატიული დამმახსოვრებელი მოწყობილობა - არანაკლებ 4 მეგაბაიტი.(პრაქტიკულად რეკომენდებული 12 მეგაბაიტი). ადგილი მყარ დისკზე - არანაკლებ 100 მეგაბაიტი(პრაქტიკულად რეკომენდებული - 200 მეგაბაიტი + ადგილი გადმოტვირთვის ფაილისთვისაც).

ეს გაცილებით მცირეა 3.1 ვერსიასთან შედარებით, რაც შესაძლებელს ხდის გაიშვას MS DOS ნებისმიერი გამოყენება პირდაპირ, უშუალოდ, ან ემულაციის რეჟიმის მეშვეობით.

ვიდეოადაპტერი - VGA - მხარდაჭერა რეჟიმზე 640X480 წერტილი 16 გარდასახვის ფერის დროს. რეკომენდებულია 256

ფერის მხარდაჭერა, ეკრანის გენერაციის სიხშირით არანაკლებ 75 ჰერცი.

დასაწყისში Windows 95-ს აყენებდნენ Windows 3.1-ის „გარეთ“, განახლების სახით, მაგრამ მალე Microsoft-მა დაამუშავა სრულყოფილი ოპერაციული სისტემა, ისეთი, რომლის ჩატვირთვა ავტომატურად ხდებოდა კომპიუტერის ჩართვისთანავე. მასში ძირითადად გამოიყენებოდა 32-თანრიგიანი კოდი, მაგრამ 16-თანრიგიანიც ისევ არსებობდა, თანხვდენილობის მოსაზრებით.

როცა ლაპარაკია დამატებებზე, სიახლეებზე და დამუშავებებზე, რომლებიც გვაქვს Windows 95-თან, პირველ რიგში უნდა აღინიშნოს გრაფიკული ინტერფეისის განახლება. აქ პირველად გამოიყენეს სამუშაო მაგიდის კონცეფცია და მისი თანმდევი ელემენტები. მაგ: ნაგვის ყუთი. საკუთრივ, სწორედ იმაში მდგომარეობს მთავარი და ყველაზე მნიშვნელოვანი გარეგნული განსხვავება, რომ აქ გვაქვს ობიექტ-ორიენტირებული მიდგომა. ახლა პირველ ადგილზე გადმონაცვლდა ობიექტი და არა გამოყენება, რომელიც მის დამუშავებას ემსახურება. სისტემის ასეთნაირად აგება აადვილებს მომხმარებლის მუშაობას, რადგან ფაილებთან მუშაობას ამსგავსებს ობიექტებთან რეალურ სამყაროში მანიპულაციებს. მნიშვნელოვან დამატებად ჩაითვალა ისიც, რომ შესაძლებელი გახდა ფაილის სახელწოდებაში სიმბოლოების რაოდენობის გაზრდა (255-მდე!)

ოპერაციული სისტემის ევოლუცია გულისხმობს ახალი ტექნოლოგიების დანერგვას და უკვე არსებულების დახვეწას, დამუშავებას, გამონაკლისი არც Windows 95 იყო.

შევჩერდეთ ზოგიერთ მათგანზე, რომლებმაც Windows 95 პლატფორმის უპირატესობა და პირველი ადგილის დაკავება განაპირობა მაშინდელ ბაზარზე Windows-ების სხვა ვერსიებთან შედარებით.

აუცილებელია აღინიშნოს მის შემადგენლობაში დამატებითი ბიბლიოთეკის Direct X-ის გაჩენა, რაც აღჭურვილობასთან პირდაპირ წვდომას ემსახურებოდა და ამით მაქსიმალურ სწრაფქმედებას უზრუნველყოფდა. მაგალითად Direct Draw და Direct Sound გამოდგება.

ოპერაციული სისტემის შემადგენლობაში ჩაირთო code-ების ერთობლიობა მონაცემთა სხვადასხვა ფორმებთან სამუშაოდ, მათ შორის მულტიმედიაც.

რადიკალურად შეიცვალა გამოყენებებთან მუშაობის სხვადასხვა ასპექტები. მაგ. გამოყენებების დეინსტალაცია ხორციელდებოდა ოპერაციული სისტემის სპეციალური საშუალებებით და არა პროგრამიდან ფაილის მოცილების გზით. ასეთი სიახლე მოსახერხებელია რამდენიმე მიზეზით. ჯერ ერთი, ის შესაძლებელს ხდის მოვიშოროთ თავად პროგრამა, ამ დროს კი შევინახოთ კომპიუტერის მეხსიერებაში საკონფიგურაციო ფაილები, რომლებსაც დისკზე შედარებით ნაკლები ადგილი უკავიათ, მაგრამ ამ დროს, თავის თავში შეიცავენ ცნობებს სამომხმარებლო ზედნაშენების შესახებ, რაც მნიშვნელოვნად ამარტივებს გამოყენებების განმეორებით დაყენების შესაძლო პროცესს. ამასთანავე ეს წესი ანთავისუფლებს მომხმარებელს პროგრამის ცალკეული ნაწილების ხელით მოშორების აუცილებლობისაგან, რაც განსაკუთრებით აქტუალურია რთული პროგრამებისთვის, რომლებიც რამდენიმე საქაღალდეში ინახება.

კომპაქტ-დისკებთან მუშაობის დროს Windows 95-ში პირველად გამოიყენეს მექანიზმი Spin & Grin - სისტემა, რომელიც ავტომატურად გამოიცნობს დისკის ტიპს და გაუშვებს შესაბამის გამოყენებას მისი კვლავწარმოებისათვის(მაგ. დაფორმატებისათვის)

Windows 95 თავდაპირველად შეიქმნა ქსელში სამუშაოდ. ამისათვის გათვალისწინებული იყო ფაილებისა და მოწყობილობების ერთობლივად გამოყენების შესაძლებლობა, ქსელთან წვდომა ქსელური ადაპტერის მომართვის გარეშე; მას ცვლის მოდემი, განვითარებული პროგრამული საშუალება ისეთ ქსელებთან წვდომისათვის, როგორიცაა Internet, Network და სხვა.

Plug & Play ტექნოლოგიის მხარდაჭერამ „ჩართე და იმუშავე“, შესაძლებლობა მოგვცა ადვილად მივუერთოთ ნებისმიერი მოწყობილობა, რომელიც თავსებადია ოპერაციულ სისტემასთან. ისინი კონფიგურაციულდებიან ავტომატურად დრაივერების მეშვეობით, რომელსაც Windows ავტომატურად არჩევს და მიმართავს. ბეჭდვის მოწყობილობაც არსებითად დაიხვეწა, რის შედეგადაც შესაძლებელი გახდა კომპიუტერთან მუშაობა გაგრძელდეს ბეჭდვის დროსაც.

მრავალამოცანიანობის ადრეულ პლატმორფებში არსებული შემჭიდროვება Windows 95-ში გარდაიქმნა ისე, რომ პროცესორული დროის განაწილებაზე კონტროლი ხორციელდება უკვე არა გამოყენებებში, არამედ სისტემის ბირთვში. ამავდროულად, უზრუნველყოფს ნორმალურ მუშაობას ფონებთან დაკავშირებულ ამოცანებზე.

Windows 95-ის პოპულარულობის კიდევ ერთი მიზეზია ამ ფორმისთვის გამოყენებების დიდი რაოდენობის გამოშვება და მათი დამუშავების სრულიად შეწყვეტა MS DOS-ის და Windows 3X-ის ქვეშ, დაწყებული 1997 წლიდან.

ამავე დროს, ამ უამრავი დადებითი მომენტების გვერდით, Windows 95-ს ჰქონდა თავისი სუსტი მხარეებიც. ეს ოპერაციული სისტემა იყო უკიდურესად „არასაიმედო“. მის დამუშავებისას იყო დაშვებული საკმაოდ ბევრი არსებითი შეცდომა, რის გამოც ერთი გამოყენების არაკორექტულ მუშაობას მწყობრიდან გამოჰყავდა მთელი სისტემა. კიდევ ერთი უარყოფითი იყო ის, რომ Intel პროცესორზე ორიენტირებული Windows 95 ცუდად მუშაობდა სხვა ფორმების პროცესორებზე. ამ ნაკლის გათვალისწინებით კორპორაციამ შეიმუშავა Windows 95 OSR-2 (Operation System release 2). არსებითად, Windows 95 OSR-2 წარმოადგენდა მხოლოდ Windows 95-ს ახალ რედაქციას. კერძოდ, Microsoft იძულებული იყო ის გამოეცა ცალკე ოპერაციული სისტემის სახით, რადგანაც სხვადასხვა ონლაინ სამსახურებში(ინტერნეტის ქსელში) ნაკლებად გავრცელების გამო, თავი იჩინა მნიშვნელოვანმა სირთულეებმა, რომელსაც ადგილი ჰქონდა განახლებებსა და დრაივერებთან დაკავშირებით.

როგორც დაგეგმილი იყო, ახალ ვერსიაში ჩასწორდა ბევრი შეცდომა, რაც დამახასიათებელი იყო Windows 95-თვის. სხვადასხვა მოწყობილობებისათვის განახლდა და გაფართოვდა დრაივერების სპექტრი. კომპლექტში Windows 95 OSR-2 შედის პრაქტიკულად უნივერსალური ბრაუზერი Internet Explorer.

უნდა აღინიშნოს, რომ არის კიდევ ერთი გარემოება. Windows 95 OSR-2 მხარს უჭერს ფაილურ სისტემას FAT 32 (File Allocation Table - ფაილების განთავსების ცხრილი), რაც დისკზე თავისუფალი ადგილების ეკონომიას განაპირობებს. ძირითადად, Windows 95-დან Windows 95 OSR-2-ზე გადასვლა მხოლოდ FAT 32-სთვის განხორციელდა.

რაც შეეხება Windows 95 OSR-2-ის სუსტ მხარეებს: ზემოხსენებული დახვეწილობების მიუხედავად, პრაქტიკულად, სისტემის საიმედოობა თითქმის არ გაუმჯობესებულა და მომხმარებლებს, ვინც დაინტერესებული იყო პირველ რიგში კომპიუტერის გამძლეობაზე, ისევ და ისევ რთულ Windows NT-ზე უხდებოდათ მუშაობა.

Windows 7

2007 წლის 24 ივლისს კომპანია მაიკროსოფტმა ოფიციალურად შეცვალა კოდური სახელწოდება Windows7 Vienna სახელწოდებით Windows7. ამ Windows 7-ში ჩასწორებული იყო Windows Vista-ში არსებული ყველა შეცდომა და ოპერაციული სისტემა Windows 7 გამოვიდა გაყიდვაში 2009 წ. 25/X . ის ნამდვილ ჰიტად იქცა გაყიდვების თვალსაზრისით. ასე, რომ Windows7 - ეს დღევანდელობაა, მაგრამ მალე ისტორიულად იქცევა, რადგან უახლოეს დროში ვარაუდობენ Windows-ის ახალი ვერსიების პრეზენტაციას. დაველოდოთ.

Windows7 დამუშავება დაიწყო OS Vista-ს გამოსვლისთანავე. დაანონსება მოხდა 2008 წ. ოქტომბერში. Microsoft-მა მოდელი #6801 წარმოადგინა კონფერენციაზე, ხოლო 2009 წ. იანვარში გამოუშვა პროდუქტის ბეტა-ვერსია. ეს ვერსია Release Candidate ხელმისაწვდომი გახდა 2009 წ. აპრილიდან -

MSDN და TechNet-ის ხელმომწერებისათვის, ხოლო 5 მაისიდან - ფართო საზოგადოებისთვისაც. ფინალური ანაწყობის გამოსვლაზე განცხადება გაკეთდა 2009 წ. 7 ივლისს.

Blackcomb. კოდური სახელწოდება Blackcomb ეკუთვნოდა Windows NT 6.0 ოპერაციულ სისტემას, რომელიც, როგორც იგეგმებოდა, Windows XP-ის შემცვლელი იქნებოდა. Blackcomb უნდა გამხდარიყო ოპერაციული სისტემის მისაღებად, როგორც სამაგიდო, ასევე სერვერებზე მომუშავე სადგურებისათვის. 2001 წლის ბოლოს ამ Blackcomb-ის გამოსვლა დაიგეგმა 2005 წლისათვის, ხოლო 2002 წ. აგვისტოში გამოაცხადეს, რომ შუალედური ვერსია იქნებოდა Windows Longhorn, რომელიც განახლება იყო Windows NT 5X-ის ბირთვისათვის. Windows Longhorn-ის დამუშავების მსვლელობისას, მას Blackcomb-ის სხვადასხვა ფუნქციები დაუმატეს და მიანიჭეს ნომერი 6.0. Blackcomb-თან დაკავშირებით გაურკვევლობასაც ჰქონდა ადგილი, რადგან სამარკეტინგო გეგმების მიხედვით, ის უნდა ყოფილიყო სერვერული ოპერაციული სისტემა Windows 6.X, მაგრამ გაუმჯობესებული, მაგრამ 2006 წლის იანვარში Microsoft-მა გამოაცხადა, რომ ახალი, კლიენტზე ორიენტირებული ოპერაციული სისტემა იქნებოდა Vienna, რომლის გამოსვლა 2010 წლისათვის დაიგეგმაო.

Windows7-ის პირველი მოდელი იყო ცნობილი სახელწოდებით “Milestone 1 (M1) Code drop” ნომრით 6.1.6519.1. ის დაეგზავნა Microsoft -ის რიგ გამსაღებელ პარტნიორებს 2008 წლის იანვარში 32 და 64 თანრიგიან(ბიტიან) ვერსიებად. ამ აწყობას ჰქონდა Vista-საგან განსხვავებული ამოცანათა პანელი, თუმცა ეს შესაძლებლობა შემდეგ გამორიცხეს. 20 აპრილს 2008 წ. ქსელში გამოჩნდა ვიდეო აწყობა 6.1.6574.1 (-მეორე აწყობა M1-

ისა). მასში ჩართული იყო შეცვლილი Windows Explorer და ახალი მხარდამჭერი ცენტრი.

TG Daily-ის cmambe-ს 2008 წ. 16 იანვრის ცნობის მიხედვით, Milestone 2 Code corp-ის გამოსვლა დაგეგმილი იყო აპრილ-მაისში. დემონსტრირება მოხდა კონფერენციაზე ნომრით D6/6.1.6589.1. მოდელს ჰქონდა Windows Vista-სგან განსხვავებული ამოცანათა პანელი.

Paul Thurrott-ის თანახმად, Milestone 3 დაეგზავნა თანამშრომლებს და Microsoft-ის ახლობელ პარტნიორებს 2008 წ. სექტემბრის I კვირაში. ის ფუნქციონალურად Windows Vista-ს მსგავსი იყო, მაგრამ ზოგიერთ ჩაშენებულ გამოყენებას ახლა ისევე ჰქონდა ინტერფეისი, როგორც Office 2007 „Ribbon". მოხსნილი იყო გამოყენებანი, რომლებიც წინა ვერსიებში გვხვდებოდა.

Pre-beta. კონფერენციაზე(PDC-ზე), Microsoft-მა წამოადგინა Pre-beta ახალი ოპერაციული სისტემა. დემონსტრაცია ჩატარდა მოდელზე #6933, იგივე დაყენდა ყველა პერსონალურ კომპიუტერზე საკონფერენციო დარბაზში მიმოხილვისა და ინტერაქციისთვის. მაგრამ, ანაწყობი მოდელი, რომელიც Microsoft-მა დააგზავნა, როგორც Pre-Beta, - იყო ნომრით 6801 და განსხვავდებოდა 6933-გან ინტერფეისის ცვლილებებით. კერძოდ, 6801-ში გულისხმობის პრინციპით გამორთული იყო ახალი ამოცანათა პანელი (Status bar), რომელიც 6933-ის Status bar-ის მსგავსი იყო.

2008 წ. დეკემბრის ბოლოს ინტერნეტში გაჩნდა შემდგომი, მორიგი ტესტური ვერსია, დანომრილი როგორც build 7000. სახელდობრ, ეს გახდა პირველი ოფიციალური Beta ვერსია

ახალი სისტემისათვის Windows7 beta. ოფიციალურად, ახალი ოპერაციული სისტემის ბეტა-გამოშვება შედგა 2009 წ. 9 იანვარს. ის ხელმისაწვდომი იყო გადმოსაწერად ISO სახედ DVD დისკზე 32 და 64 ბიტურ არქიტექტურაში. Windows7 beta მუშაობდა 2009 წ. 1 აგვისტომდე. გადმოწერა 2009 წ. 10 თებერვლამდე იყო შესაძლებელი.

Windows7-beta-ს გამოშვებისას გაჩნდა პრობლემები. მაგ: ჭრიდა MP3 ფაილების პირველ წამებს, მაგრამ შეცდომა მალე ჩასწორდა დამმუშავებლების მიერ.

Pre-RC. 2009წ. 14 მარტს ინტერნეტის მომხმარებლებს მიეცათ Windows7 build 7057-ზე წვდომის შანსი. 25 მარტს TechNet პარტნიორების შეზღუდულმა ჯგუფმა ეს შანსი build 7068 ნომრით მიიღო. 26 მარტს ანაწყობი ქსელშიც გაჩნდა. 7 აპრილს ქსელში ხელმისაწვდომი გახდა შემდეგი ანაწყობი Windows build 7077.

Release Candidate 1 (RC1) მოდელი ეფუძვნება winmain-win7re შტოს, რომელშიც ის იქნა აწყობილი. ეს ინფორმაცია ოპერატიულად დაადასტურა WZorNet-მა. ოფიციალური ვერსია Windows7 Release Candidate გახდა ანაწყობი 7100.0 (winmain-win7rc, 090421-1700). ამ ვერსიას აქვს ინგლისური, გერმანული, ესპანური, ფრანგული და იაპონური ლოკალიზაცია. 30 აპრილს Microsoft-მა Windows 7 RC ხელმომწერებს - MSDN და Technet-ს წარუდგინა.

ქსელში გაჩნდა ასევე IDX ანაწყობი Windows 7 build 7201. ეს ანალოგიურია Release Candidate -ის მე-2 Release to Manufacturing(RTM) ვარიანტისა. ეს არ იყო დაშვებული მასობრივი მომხმარებლისათვის. ოფიციალურად წარდგინება

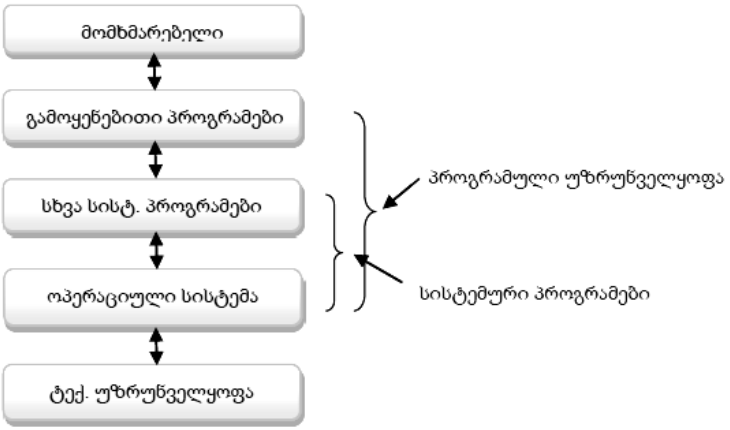
მოხდა 2009წ. 22 ოქტომბერს. თუმცა, 2009 წ. 18 ივლისს გამოვიდა ფინალური RTM ვერსია Windows 7 ე.წ. „ოქროს კოდი“, ხოლო 23 ივლისს 2009 წ. თავის ბლოგზე Microsoft-მა განაცხადა, რომ თავისი ოპერაციული სისტემის დამუშავება დაასრულა.

ოპერაციული სისტემა

უმარტივესი განმარტებით ოპერაციული სისტემა - ეს არის პროგრამა, ეს არის გამოთვლითი სისტემა, რომელიც უზრუნველყოფს მომხმარებლისათვის კომპიუტერის მოწყობილობათა რაციონალური გამოყენების შესაძლებლობას.

ნებისმიერი გამოთვლითი სისტემა შედგება ერთი მხრივ, ტექნიკური უზრუნველყოფისაგან(hardware): პროცესორი, მეხსიერება, მონიტორი, მაუსი (თაგუნა), დისკური მოწყობილობა და ა.შ., რომლებიც გაერთიანებული არიან მაგისტრალური დაკავშირით, რომელსაც სალტე ეწოდება. მეორე მხრივ, გამოთვლითი სისტემა შედგება პროგრამული უზრუნველყოფისგან(software). ყოველი პროგრამული უზრუნველყოფა იყოფა ორ ნაწილად: გამოყენებითი და სისტემური. როგორც წესი, გამოყენებით პროგრამულ უზრუნველყოფას განეკუთვნება სხვადასხვა საყოფაცხოვრებო, საინჟინრო, საბანკო და სხვა ბიზნეს-პროგრამები, თამაშები, ტექსტური პროცესორები, და ა.შ. ჩვეულებრივ, სისტემური პროგრამული უზრუნველყოფის ქვეშ იგულისხმება სისტემის ფუნქციონირებისათვის და გამოყენებითი პროგრამების შექმნისათვის ხელშემწყობი პროგრამები. უნდა აღინიშნოს, რომ პროგრამული უზრუნველყოფის დაყოფა გამოყენებით და სისტემურ ნაწილებად ნაწილობრივ პირობითია და დამოკიდებულია იმაზე, თუ ვინ ანხორციელებს დაყოფას. უფრო ზუსტად, ჩვეულებრივმა მომხმარებლმა, რომელიც გამოუცდელია პროგრამირებაში, შეიძლება ჩათვალოს, რომ

Microsoft Word არის სისტემური პროგრამა, ხოლო პროგრამისტის თვალსაზრისით ის არის დანართი. C ენის კომპილიატორი ჩვეულებრივი პროგრამისტისათვის წარმოადგენს სისტემურ პროგრმას, ხოლო სისტემურისთვის კი ის არის გამოყენებითი პროგრამა. მიუხედავად ამ გაურკვეველი საზღვრისა, მოცემული სიტუაცია შეიძლება წარმოვიდგინოთ დონეების(შრეების) მიმდევრობის სახით (ნახ.1), რომელშიც გამოყოფილი იქნება სისტემური პროგრამული უზრუნველყოფის ყველაზე უფრო ზოგადი ნაწილები - ოპერაციული სისტემა და სხვა სისტემური პროგრამები.



ნახ. **1** კომპიუტერული სისტემის პროგრამული უზრუნველყოფის დონეები

რა არის ოპერაციული სისტემა? კომპიუტერის მომხმარებელთა უმეტესს გააჩნია ოპერაციული სისტემის ექსპლუატაციის გამოცდილობა, მაგრამ მიუხედავად ამისა, მათ მაინც გაუჭირდებათ მისთვის ზუსტი განმარტების მიცემა. მოკლედ მიმოვიხილოთ ძირითადი მოსაზრებანი.

ოპერაციული სისტემა როგორც ვირტუალური მანქანა

ოპერაციული სისტემის შემუშავებისას ფართოდ გამოიყენება აბსტრაგირება, რომელიც წარმოადგენს გამარტივების მთავარ მეთოდს და იძლევა სისტემის მაღალდონიანი კომპონენტების ურთიერთქმედებაზე კონცენტრირების საშუალებას, მათი რეალიზაციის დეტალების იგნორირებით. ამ აზრით, ოპერაციული სისტემა წარმოადგენს ინტერფეისს მომხმარებელსა და კომპიუტერს შორის.

კომპიუტერის არქიტექტურა მანქანურ ბრძანებათა დონეზე გამოყენებითი პროგრამების გამოყენებისათვის ძალიან მოუხერხებელია. მაგალითად, დისკთან მუშაობა გულისხმობს მისი შიდა მოწყობილობების ელექტრული კომპონენტების ცოდნას - დისკის ბრუნვისათვის ბრძანებათა შეყვანის კონტროლერს, ბილიკების მოძებნას და ფორმატირებას, სექტორების წაკითხვასა და წაშლას და ა.შ. ცხადია, რომ საშუალო დონის პროგრამისტს არ შეუძლია გაითვალისწინოს მოწყობილობათა მუშაობის ყველა თავისებურება (თანამედროვე ტერმინოლოგიით - შეუდგეს მოწყობილობათა დრაივერების შემუშავებას), არამედ უნდა გააჩნდეს მარტივი მაღალდონიანი აბსტრაქცია. მაგალითად, წარმოიდგინოს დისკის ინფორმაციული სივრცე, როგორც ფაილების ნაკრები. ფაილები შეიძლება იყოს გახსნილი წასაკითხად ან ჩასაწერად, ანუ, ინფორმაციის მისაღებად. ეს კონცეპტუალურად უფრო მარტივია ვიდრე იმაზე ფიქრი, თუ როგორაა მოწყობილი დიკის ბილიკები და როგორ მუშაობს მისი ბრუნვის მოწყობილობა. ანალოგიურად, მარტივი და ცხადი აბსტრაქციით პროგრამისტისაგან დამალულია ყველა არასასურველი წვრილმანი წყვეტის ორგანიზაციის, ტაიმერის მუშაობის, მეხსიერების მართვის და ა.შ. უფრო მეტიც, თანამედროვე ოპერაციულ სისტემებში შეიძლება შეგვექმნას ილუზია ოპერატიული მეხსიერების და პროცესორების შეუზღუდავი რაოდენობით გამოყენების შესაძლებლობაზე. ყველაფერ ამას განაგებს ოპერაციული სისტემა. მაშასადამე, ოპერაციული

სისტემა მომხმარებელს შეიძლება წარმოუდგეს როგორც ვირტუალური მანქანა, რომელთანაც მუშაობა უფრო ადვილია ვიდრე უშუალოდ კომპიუტერის მოწყობილობებთან.

ოპერაციული სისტემა - რესურსების მენეჯერი

ოპერაციული სისტემა განკუთვნილია კომპიუტერის საკმაოდ რთული არქიტექტურის ყველა ნაწილის სამართავად. მაგალითად, წარმოვიდგინოთ რა მოხდება, თუ ერთ კომპიუტერზე მომუშავე რამდენიმე პროგრამა ერთდროულად შეეცდება პრინტერზე საკუთარი შედეგის ბეჭდვას. ასეთ შემთხვევაში მივიღებდით სხვადასხვა პროგრამების მიერ ქაოტურად დაბეჭდილ სტრიქონებისაგან შედგენილ გვერდებს. ოპერაციული სისტემა იცილებს ამ ტიპის ქაოსს ბეჭდვისათვის განკუთვნილი ინფორმაციის დისკზე ბუფერირების და ბეჭდვის თანმიმდევრობის ორგანიზაციის გზით. მრავალ მომხმარებლიანი კომპიუტერებისათვის მართვის რესურსების და მათი უსაფრთხოების დაცვის აუცილებლობა კიდევ უფრო ცხადია. შესაბამისად, ოპერაციული სისტემა, როგორც რესურსების მენეჯერი, ანხორციელებს პროცესორების, მეხსიერების და სხვა რესურსების სხვადასხვა პროგრამებს შორის დალაგებულ და კონტროლირებად განაწილებას.

ოპერაციული სისტემა - მომხმარებელთა და პროგრამათა დამცველი

თუ გამოთვლითი სისტემა უშვებს რამდენიმე მომხმარებლის ერთობლივ მუშაობას, მაშინ წარმოიშობა მათი მუშაობის უსაფრთხოების ორგანიზაციის პრობლემა. აუცილებელია დისკზე ინფორმაციის დაცულობის უზრუნველყოფა, რათა არავინ შეძლოს სხვისი ფაილის წაშლა ან დაზიანება. არ შეიძლება ცალკეული მომხმარებლის პროგრამამ ზემოქმედება იქონიოს სხვა მომხმარებლების პროგრამაზე. უნდა

აღიკვეთოს გამოთვლითი სისტემის არასანქცირებული გამოყენების მცდელობები. ყველა ამ მოქმედებას ანხორციელებს ოპერაციული სისტემა როგორც მომხმარებლების და მათი პროგრამების მუშაობის უსაფრთხოების დაცვის ორგანიზატორი.

ოპერაციული სისტემა - მუდმივად ფუნქციონირებადი ბირთვი

შეიძლება მოვიყვანოთ ოპერაციული სისტემის ასეთი განმარტებაც: ოპერაციული სისტემა - ეს არის პროგრამა, რომელიც მუდმივად მუშაობს კომპიუტერზე და ურთიერთქმედებს ყველა გამოყენებით პროგრამასთან. შეიძლება გვეჩვენოს, რომ ეს არის აბსოლუტურად სამართლიანი განმარტება, მაგრამ, როგორც ამას შემდგომში ვნახავთ, მრავალი თანამედროვე ოპერაციული სისტემიდან კომპიუტერში მუდმივად მუშაობს ოპერაციული სისტემის მხოლოდ ის ნაწილი, რომელსაც მისი **ბირთვი** ეწოდება.

ჩვენ დავრწმუნდით, რომ არსებობს მრავალი თვალსაზრისი იმაზე, თუ რას წარმოადგენს ოპერაციული სისტემა. შეუძლებელია მისთვის ადეკვატური მკაცრი განმარტების მინიჭება. ჩვენთვის იმის თქმა კი არ არის მარტივი, თუ რა არის ოპერაციული სისტემა, არამედ იმისი, - თუ რისთვისაა ის საჭირო და რისი გაკეთება შეუძლია მას.

ოპერაციული სისტემის კონცეფციების ძირითადი ცნებები

ევოლუციის პროცესში წარმოიშვა რამდენიმე მნიშვნელოვანი კონცეფცია, რომლებიც იქცა ოპერაციული სისტემის თეორიისა და პრაქტიკის განუყოფელ ნაწილად. ესენია:

ა). სისტემური გამოძახება

ნებისმიერ ოპერაციულ სისტემაში მხარდაჭერილია მექანიზმი, რომელიც მომხმარებლის პროგრამებს საშუალებას აძლევს მიმართოს ოპერაციული სისტემის ბირთვის მიერ შემოთავაზებულ მომსახურეობებს. სხვადასხვა ოპერაციულ

სისტემებში მათ სხვადასხვა სახელებს ეძახიან, მაგალითად, IBM-ის ოპერაციული სისტემებისთვის ესაა ექსტრაკოდები, Unix-მსგავსი ოპერაციული სისტემებისთვის კი - სისტემური გამოძახება.

სისტემური გამოძახება (system calls) - ეს არის ოპერაციულ სისტემასა და მომხმარებლის პროგრამებს შორის ინტერფეისი. მას შეუძლია შექმნას, წაშალოს და გამოიყენოს სხვადასხვა ობიექტები, რომელა შორის მნიშვნელოვანია პროცესები და ფაილები. მომხმარებლის პროგრამა გარკვეული სერვისის მისაღებად ოპერაციულ სისტემას მიმართავს სისტემური გამოძახების მეშვეობით. არსებობენ პროცედურათა ბიბლიოთეკები, რომლებიც მანქანის რეგისტრებში ტვირთავენ გარკვეულ პარამეტრებს და ახდენენ პროცესორის წყვეტას, რის შემდეგაც მართვა გადაეცემა ოპერაციული სისტემის ბირთვის შესაბამის ნაწილს, რომელიც ახდენს მიმდინარე გამოძახების დამუშავებას. ასეთი ბიბლიოთეკების არსებობის მიზანს წარმოადგენს ის, რომ სისტემური გამოძახება ქვეპროგრამის ჩ). ბ). **ჩვეულებრივ გამოძახების** მსგავს გამოძახებად იქცეს.

ძირითადი განსხვავება მდგომარეობს იმაში, რომ სისტემური გამოძახებისას ამოცანა გადადის პრივილეგირებულ ანუ ბირთვის (kernel mode) რეჟიმში. ამიტომ სისტემურ გამოძახებას ხშირად პროგრამულ წყვეტასაც უწოდებენ, განსხვავებით აპარატული წყვეტისაგან, რომელსაც შემოკლებით **წყვეტა** ეწოდება.

გ). პრივილეგირებულ რეჟიმში მუშაობს ოპერაციული სისტემის ბირთვის კოდი. ამასთან, ის მისი გამომძახებელი ამოცანის კონტექსტის ფარგლებში მისამართების სივრცეში სრულდება. მაშასადამე, ოპერაციული სისტემის ბირთვს გააჩნია სრული წვდომა მომხმარებლის პროგრამის მეხსიერებაზე. სისტემური გამოძახებისას საკმარისია მეხსიერების ერთი ან რამდენიმე მიდამოს მისამართის მითითება გამოძახების პარამეტრებით და მეხსიერების ერთი ან რამდენიმე მიდამოს მისამართის მითითება გამოძახების შედეგისათვის.

უმეტეს ოპერაციულ სისტემებში სისტემური გამოძახება პროგრამული წყვეტის ბრძანებით (INT) ხორციელდება. პროგრამული წყვეტა ეს არის სინქრონული მოვლენა, რომელიც შეიძლება იქნას განმეორებული ერთიდაიგივე პროგრამული კოდის შესრულებისას.

დ). წყვეტა (hardware interrupt). ეს არის(პროცესორთან მიმართებაში) გარე მოწყობილების მიერ გენერირებული მოვლენა. აპარატურა აპარატული წყვეტის მეშვეობით ან ახდენს ცენტრალური პროცესორის ინფორმირებას იმაზე, რომ მოხდა გარკვეული მოვლენა (მაგალითად, მომხმარებელმა დააჭირა ღილაკზე) და საჭიროა დაუყოვნებელი რეაქცია, ან, აცნობებს შეტანა/გამოტანის ასინქრონული ოპერაციის (მაგალითად, დასრულდა დისკიდან მონაცემების კითხვა ძირითად მეხსიერებაში) დასრულების შესახებ. აპარატული წყვეტის მნიშვნელოვანი ტიპია - *ტაიმერიდან წყვეტა,* რომელიც გენერირდება პერიოდულად გარკვეული დროითი შუალედის - კვოტის, გასვლის შემდეგ. ტაიმერიდან წყვეტა ოპერაციული სისტემის მიერ პროცესების დაგეგმვისას გამოიყენება. აპარატული წყვეტის ყოველ ტიპს გააჩნია საკუთარი ნომერი, რომელიც ცალსახად განსაზღვრავს წყვეტის წყაროს. აპარატული წყვეტა ეს არის ასინქრონული მოვლენა ანუ, ის წარმოიშობა იმისგან დამოუკიდებლად, მოცემულ მომენტში თუ რომელი კოდი სრულდება პროცესორის მიერ.

ე). განსაკუთრებული სიტუაცია (exception) ეს არის პროგრამის მიერ ბრძანების შესრულების მცდელობის შედეგად წარმოშობილი მოვლენა, რომელიც გარკვეული მიზეზების გამო შეუძლებელია შესრულდეს. ასეთი ბრძანების მაგალითი შეიძლება იყოს შეზღუდულ რესურსზე წვდომის მცდელობა არასაკმარისი პრივილეგიის ქონის დროს ან მიმართვა მახსოვრობის არარსებულ გვერდზე. როგორც სისტემური გამოძახებები, ისე განსაკუთრებული სიტუაციები წარმოადგენენ მიმდინარე ამოცანის კონტექსტის ფარგლებში წარმოშობილ სინქრონულ მოვლენას. განსაკუთრებული სიტუაციები შეიძლება

დაიყოს ორ ნაწილად: გამოსწორებადი და გამოუსწორებელი. გამოსწორებადს მიეკუთვნება ისეთი განსაკუთრებული სიტუაცია, როგორიცაა ოპერატიულ მეხსიერებაში საჭირო ინფორმაციის არარსებობა. გამოსწორებადი განსაკუთრებული სიტუაციის გამომწვევი მიზეზის აღმოფხვრის შემდეგ პროგრამას შეუძლია გააგრძელოს შესრულება. ოპერაციული სისტემის მუშაობის პროცესში გამოსწორებადი განსაკუთრებული სიტუაციის წარმოშობა ითვლება ნორმალურ მოვლენად. გამოუწორებელი განსაკუთრებული სიტუაციები ძირითადად წარმოიშობა პროგრამული შეცდომებისას (მაგალითად, ნულზე გაყოფა). ჩვეულებრივ, ასეთ შემთხვევებში ოპერაციული სისტემა რეაგირებს იმ პროგრამის შესრულების შეწყვეტით, რომელმაც ეს გამოუწორებელი განსაკუთრებული სიტუაცია წარმოშვა.

ფაილები

ფაილები განკუთვნილია დისკზე ან სხვა მოწყობილობებზე ინფორმაციის შესანახად. ჩვეულებრივ ფაილის ქვეშ ინფორმაციის მატარებელზე სივრცის სახელდებული ნაწილი იგულისხმება.

ფაილური სისტემის (file system) ძირითადი ამოცანაა დამალოს შეტანა/გამოტანის მოწყობილობის თავისებურებანი და პროგრამისტს წარმოუდგინოს ფაილების მარტივი აბსტრაქტული მოდელი, რომელიც დამოუკიდებელია მოწყობილებებისგან. არსებობს სისტემური გამოძახებების ფართო კატეგორია, რომლებიც გამოიყენება ფაილების შექმნის, გახსნის, დახურვის, წაკითხვის, ჩაწერის და წასაშლელად, და ასევე, არსებობს სისტემური გამოძახებები, რომლებიც გამოიყენება ფაილებთან დაკავშირებული ცნებებით მანიპულირებისათვის(როგორიცაა კატალოგი, მიმდინარე კატალოგი, საბაზო კატალოგი, გზა).

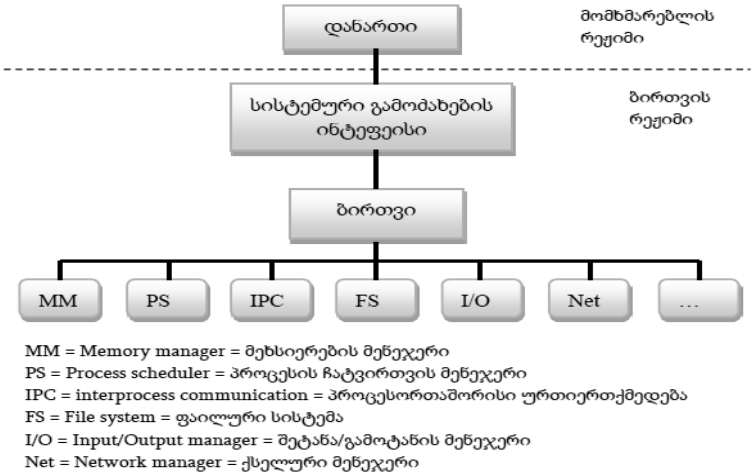
ოპერაციული სისტემის არქიტექტურული თავისებურებანი

თანამედროვე ოპერაციული სისტემები მიისწრაფვიან სირთულეებისკენ, ანუ, იმისკენ, რომ მომხმარებლებს წარმოუდგინონ მომსახურეობათა ფართო სპექტრი, და გააჩნიათ სხვადასხვა აპარატული და პროგრამული რესურსების მხარდაჭერა. ოპერაციული სისტემისათვის შესაფერისი არქიტექტურის სწორად არჩევა, ოპერაციული სისტემის კომპონენტების დალაგების და მათი პრივილეგირებულობის განსაზღვრის გზით, მის შემქმნელებს ეხმარება სირთულეებთან გამკლავებაში. თუ არქიტექტურა მონოლიტურია, მაშინ ოპერაციული სისტემის ყოველი კომპონენტი შედის ბირთვის შემადგენლობაში. თუ სისტემის არქიტექტურა აგებულია მიკრობირთვზე, მაშინ მასში შედის მხოლოდ ყველაზე აუცილებელი კომპონენტები.

მონოლიტური ბირთვი

მონოლიტური ოპერაციული სისტემა **(monolithic operating system)** - ეს ოპერაციული სისტემის ყველაზე ადრეული და გავრცელებული არქიტექტურაა. ასეთი ოპერაციული სისტემის ყოველი კომპონენტი ინტეგრირებულია ბირთვში და სხვა კომპონენტებთან უშუალოდ ურთიერთქმედება შეუძლია. როგორც წესი, ბირთვი რესურსებზე დაშვების განუსაზღვრელი უფლებებით სრულდება (ნახ.2). ოპერაციული სისტემები OS/360, VMS და Linux-ი განეკუთვნებიან მონოლიტურს. კომპონენტებს შორის უშუალო ურთიერთქმედება მონოლიტურ ოპერაციულ სისტემას ხდის მაღალკოეფიციენტიანს. თუმცა იმის შედეგად, რომ მონოლიტური ბირთვი აერთიანებს ყველა კომპონენტს, - შეცდომებისა და შეფერხებების წყაროს დადგენა რთულ ამოცანას წარმოადგენს. უფრო მეტიც, ვინაიდან ბირთვის კოდი სრულდება რესურსებზე დაშვების შეუზღუდავი უფლებებით, ამიტომ ის სისტემები, რომლებსაც გააჩნიათ მონოლიტური ბირთვი, განსაკუთრებით სუსტია

შეცდომის ან ზიანის შემცველი კოდის მიმართ.

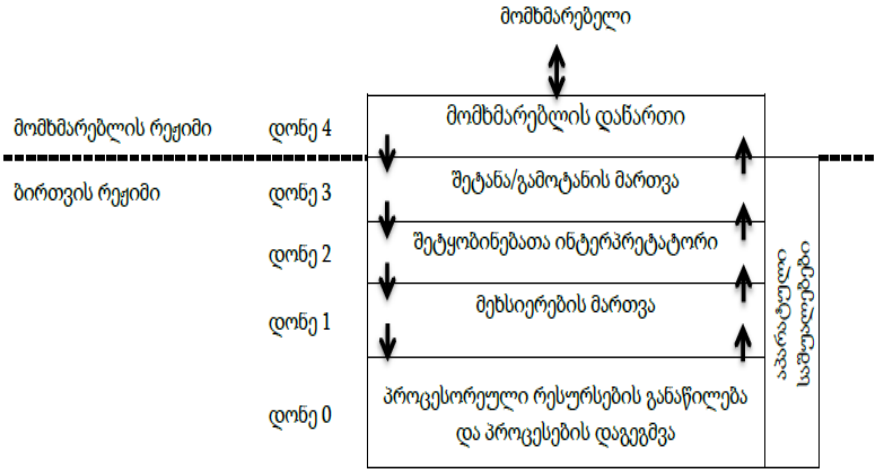


ნახ.2 ბირთვის რეჟიმი

მრავალდონიანი არქიტექტურა. იქიდან გამომდინარე, რომ ოპერაციული სისტემა დიდი და რთული გახდა, მონოლიტურმა გადაწყვეტამ დაკარგა თავის პოპულარობა. ოპერაციული სისტემის აგების მრავალდონიანმა **(Layered)** მიდგომამ წარმოაჩინა ალტერნატიული არქიტექტურული გადაწყვეტა, რომელიც დაფუძვნებული იყო მსგავსი ფუნქციების შემსრულებელი კომპონენტების დონეებად ორგანიზებაზე. სისტემის ყოველი კომპონენტი ურთიერთქმედებდა მხოლოდ მის მეზობელ დონესთან, რომელიც უშუალოდ მის ზემოთ ან ქვემოთ იყო განთავსებული. დაბალი დონეები ემსახურება მაღალ დონეებს, მათი რეალიზაციის დამმალავი ინტერფეისის გამოყენებით.

მრავალდონიან ოპერაციულ სისტემას გააჩნია უფრო მაღალი მოდულობა ვიდრე მონოლიტურს, ვინაიდან ყოველი დონის რეალიზაცია შესაძლებელია მოდიფიცირებული იყოს სხვა დონეებისაგან დამოუკიდებლად. მოდულური სისტემა შედგება ავტონომიური ჩაკეტილი კომპონენტებისგან, რომლებიც შესაძლებელია ხელმეორედ იქნან გამოყენებულნი. ყოველი

კომპონენტი “მალავს”, თუ როგორ წყვეტს ის საკუთარ ამოცანებს, მაგრამ სთავაზობს ინტერფეისს, რომელიც შესაძლებელია გამოყენებული იქნას სხვა კომპონენტების მიერ მოცემული კომპონენტისათვის მომსახურეობაზე მოთხოვნის გასაკეთებლად.



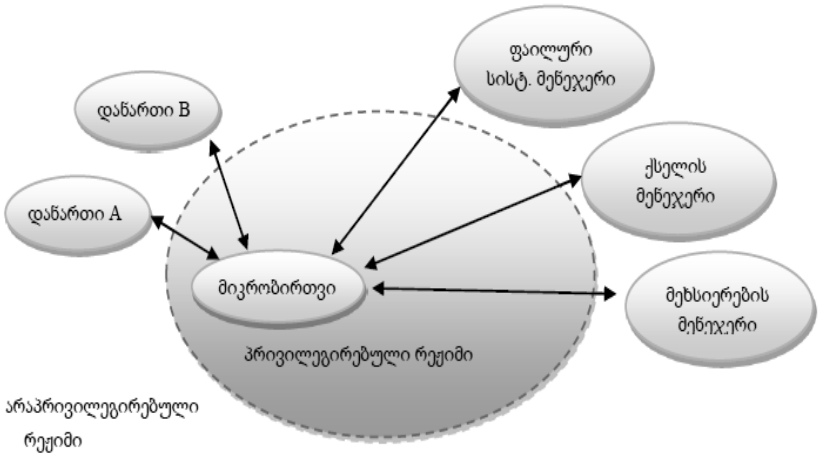
ნახ. 3 THE ოპერაციული სისტემის დონეები

ოპერაციული სისტემის მოდულურობა უზრუნველყოფს სტრუქტურული ორგანიზაციისა და ურთიერთქმედების სიმარტივეს, როგორც წესი, ქმედითუნარიანობის შემოწმების, გადადენის და მოდიფიცირების გამარტივებით. მიუხედავად ამისა, მრავალდონიანი მიდგომისას მომხმარებლის პროცესს შესაძლებელია დასჭირდეს მრავალი დონის გავლა საკუთარი მოთხოვნის დასაკმაყოფილებლად. რადგანაც მონაცემების ერთი დონიდან მეორეზე გადაცემისას გამოყენებული უნდა იყოს შუალედური ელემენტები, სისტემის წარმადობა ეცემა. ეს ხდება მონოლიტური ბირთვისაგან განსხვავებით, რომელსაც შეიძლება დასჭირდეს ერთი გამოძახება ანალოგიური მოთხოვნის მოსამხურებისათვის. გარდა ამისა, ვინაიდან დონეები

აღჭურვილია რესურსებზე განუსაზღვრელი დაშვებით, მრავალდონიანი ბირთვიც სუსტია შეცდომის ან ზიანის შემცველი კოდის მიმართ. THE (Technishe Hogeschool Eindhoven) ოპერაციული სისტემა წარმოადგენს მრავალდონიანი ოპერაციული სისტემის მაგალითს (ნახ.3). მრავალი თანამედროვე ოპერაციული სისტემა, მათ შორის Windows XP და Linux შეიძლება გარკვეული თვალსაზრისით განეკუთვნებოდნენ მრავალდონიან სისტემებს.

მიკრობირთვული არქიტექტურა (microkernel architecture).

ოპერაციული სისტემის თანამედროვე ტენდენციები სისტემური კოდის მნიშვნელოვანი ნაწილის მომხმარებლის დონეზე გადატანასა და იმავდროულად ბირთვის მინიმიზაციაში მდგომარეობს. ამ შემთხვევაში საუბარია ისეთი ბირთვის აგებაზე, რომელსაც მიკრობირთვულ არქიტექტურას უწოდებენ, რომლის შემადგენელი ნაწილებიდან უმეტესი წარმოადგენს დამოუკიდებელ პროგრამას. ასეთ შემთხვევაში, მათ შორის კავშირს უზრუნველყოფს ბირთვის სპეციალური მოდული, რომელსაც **მიკრობირთვი** ეწოდება. მიკრობირთვი მუშაობს პრივილეგირებულ რეჟიმში და უზრუნველყოფს პროგრამებს შორის კავშირს, პროცესორის გამოყენების დაგეგმვას, წყვეტის პირველად დამუშვებას, შეტანა/გამოტანის ოპერაციებს და მეხსიერების საბაზო მართვას.
სისტემის სხვა კომპონენტები ერთმანეთთან ურთიერთქმედებენ მიკრობირთვის საშუალებით შეტყობინებათა გადაცემის გზით. მიკრობირთვული არქიტექტურის ძირითადი უპირატესობა ოპერაციული სისტემის მაღალი ხარისხით მოდულობაში მდგომარეობს. ეს არსებითად ამარტივებს მასში ახალი კომპონენტების დამატების შესაძლებლობას. ოპერაციულ სისტემაში მიკრობირთვის ბაზაზე შესაძლებელია მისი მუშაობის შეუჩერებლად ახალი დრაივერების, ფაილური სისტემების და ა.შ. ჩატვირთვა/ამოტვირთვა.



ნახ.**4** ოპერაციული სისტემის მიკრობირთვული არქიტექტურა

მიკრობირთვული არქიტექტურა ამაღლებს სისტემის საიმედოობას, ვინაიდან არაპრივილეგირებულ რეჟიმში პროგრამის შეცდომა ნაკლებად სახიფათოა ვიდრე შეცდომა ბირთვის რეჟიმში. იმავდროულად მიკრობირთვული არქიტექტურა მოითხოვს დამატებით ზედნადებ დანახარჯებს, რაც დაკავშირებულია მოდულებს შორის შეტყობინებების გადაცემასთან, რომელიც ამცირებს სისტემის წარმადობას. იმისათვის, რომ მიკრობირთვის ბაზაზე აგებული ოპერაციული სისტემა სიჩქარით არ ჩამოუვარდებოდეს მონოლიტური ბირთვის ბაზაზე აგებულ ოპერაციულ სისტემას, საჭიროა სისტემის კომპონენტებად დაყოფისას უფრო მეტი სიფრთხილე და ამასთან, კომპონენტებს შორის კავშირი უნდა იყოს მინიმიზირებული.

შერეული სისტემები. ოპერაციული სისტემის აგების ზემოთ განხილულ ყოველ მიდგომას თავისი უპირატესობა და ნაკლი გააჩნია. უმეტეს შემთხვევებში თანამედროვე ოპერაციული სისტემები იყენებენ ამ მიდგომათა სხვადასხვა კომბინაციებს. მაგალითად, Linux ოპერაციული სისტემის ბირთვი წარმოადგენს მონოლიტურ სისტემას მიკრობირთვული არქიტექტურის ელემენტებით. ბირთვის კომპილაციისას შესაძლებელია ბირთვის მრავალი კომპონენტის ე.წ. მოდულების დინამიური ჩატვირთვა/ამოტვირთვა. მოდულის ჩატვირთვის მომენტში მისი კოდი იტვირთება სისტემის დონეზე და უკავშირდება ბირთვის დანარჩენ ნაწილს. მოდულის შიგნით შესაძლებელია ბირთვის მიერ ექსპორტირებული ნებისმიერი ფუნქციის გამოყენება.

შერეული მიდგომის სხვა მაგალითად შეიძლება გამოდგეს მონოლიტური ბირთვის ოპერაციული სისტემის მიკრობირთვის მართვის ქვეშ გაშვების შესაძლებლობა. ასეთნაირადაა მოწყობილი BSD და MkLinux სისტემები, რომლებიც Mach მიკრობირთვზე არიან დაფუძნებული. მიკრობირთვი უზრუნველყოფს ვირტუალური მეხსიერების მართვას და დაბალდონიანი დრაივერების მუშაობას. სხვა დანარჩენ ფუნქციებს, მათ შორის, გამოყენებით პროგრამებს შორის კავშირი ხორციელდება მონოლიტური ბირთვის მეშვეობით. გამოსავალი მიდგომის ფორმირება მოხდა ისეთი ოპერაციული სისტემის აგების მცდელობისას, რომელშიც სრულად იქნებოდა გამოყენებული მიკრობირთვული არქიტექტურის უპირატესობა და შეძლებისდაგვარად შენარჩუნებული იქნებოდა მონოლიტური ბირთვის „კარგი“ კოდის უდიდესი ნაწილი.
მიკრობირთვული არქიტექტურისა და მონოლიტური ბირთვის ელემეტები ყველაზე მეტადაა შეზავებული Windows NT ბირთვში. თუმცა Windows NT-ს ხშირად უწოდებენ მიკრობირთვულ ოპერაციულ სისტემას, რაც მთლად ასეც არაა. Windows NT-ს კომპონენტები განთავსებულია ცვალებად მეხსიერებაში და ერთმანეთთან კავშირს შეტყობინებათა გადაცემის გზით

ამყარებენ, რაც მონოლიტური ბირთვის ბაზაზე მყოფი ოპერაციული სისტემისათვის არის დამახასიათებელი. მაშასადამე, სრულიად მართებული იქნება Windows NT-ს ვუწოდოთ ჰიბრიდული ოპერაციული სისტემა.

ოპერაციული სისტემების კლასიფიკაცია

არსებობს ოპერაციული სისტემების კლასიფიკაციის რამდენიმე სქემა. ქვემოთ მოყვანილია კლასიფიკაცია მომხმარებლის თვალსაზრისით რამდენიმე ნიშან-თვისებით.

მრავალამოცანიანობის რეალიზაცია

ოპერაციული სისტემა ერთდროულად შესრულების პროცესში მყოფი ამოცანათა რიცხოვნობის მიხედვით შეიძლება დაიყოს ორ კლასად:

მრავალამოცანიანი (Unix, OS/2, Windows);

ერთამოცანიანი (მაგალითად, MS-DOS).

მრავალამოცანიანი ოპერაციული სისტემა ყოველ პროგრამას გამოუყოფს პროცესორული დროის კვოტას, რომლის დასრულების შემდეგ მართვას გადასცემს სხვა პროგრამას.

მრავალმომხმარებლიანი რეჟიმის მხარდაჭერა

ოპერაციული სისტემა ერთდროულად მომუშავე მომხმარებელთა რიცხოვნობით შეიძლება გავყოთ:

ერთმომხმარებლიანი (MS-DOS, Windows 3.x);

მრავალმომხმარებლიანი (Windows NT, Unix).

ამ ოპერაციულ სისტემებს შორის მნიშვნელოვანი განსხვავება მდგომარეობს მრავალმომხმარებლიან სისტემებში ყოველი მომხმარებლის პერსონალური მონაცემების დაცვის მექანიზმის არსებობაში.

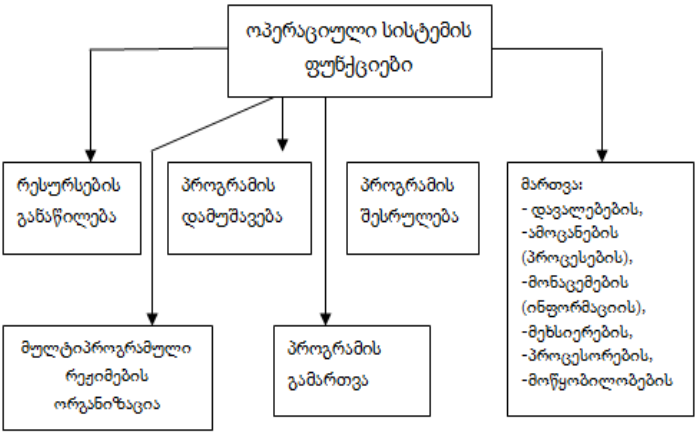
მრავალპროცესორული დამუშავება. თითქმის ბოლო პერიოდამდე გამოთვლით მანქანებს გააჩნდათ ერთი ცენტრალური პროცესორი. წარმადობის ამაღლების გაზრდილი მოთხოვნის და სხვა მრავალი ფაქტორის ზემოქმედების შედეგად გაჩნდა

მრავალპროცესორული სისტემები, რომლებიც შედგებოდნენ საერთო დანიშნულების 2 ან მეტი პროცესორისაგან და შეეძლოთ ერთმანეთის პარალელურად ბრძანებათა შესრულება. მრავალპროცესორულობის მხარდაჭერა წარმოადგენს ოპერაციული სისტემის მნიშვნელოვან თვისებას და მივყევართ რესურსების მართვის ყველა ალგორითმის გართულებამდე. მრავალპროცესორული დამუშვება რეალიზებულია ისეთ ოპერაციულ სისტემებში როგორიცაა Linux, Solaris, Windows NT და სხვა მრავალი.

მრავალპროცესორულ სისტემებს ყოფენ ორ ნაწილად: სიმეტრიული და ასიმეტრიული. სიმეტრიულ ოპერაციულ სისტემაში ყოველ პროცესორზე ფუნქციონირებს ერთიდაიგივე ბირთვი და ამოცანა შეიძლება იყოს შესრულებული ნებისმიერ მათგანზე, ანუ დამუშავება ბოლომდე დეცენტრალიზებულია. ამასთან, ყოველი პროცესორისათვის მთლიანი მეხსიერება ხელმისაწვდომია სრულად.

რეალური დროის სისტემები გამოიყენება სხვადასხვა ტექნიკური ობიექტების ან ტექნოლოგიური პროცესების სამართავად. ასეთი სისტემები ხასიათდება გარე მოვლენებზე უკიდურესად მისაღები დროითი რეაქციით, რომლის განმავლობაშიც შეიძლება შესრულებული იყოს ობიექტების მმართველი პროგრამა. სისტემამ უნდა დაამუშავოს შემომავალი მონაცემები იმაზე სწრაფად, ვიდრე ისინი შემოედინება, ამასთან შესაძლებელია მონაცემების შემოდინება ხდებოდეს ერთდროულად რამდენიმე წყაროდან.

ასეთი მკაცრი შეზღუდვა აისახება რეალური დროის სისტემების არქიტექტურაზე, მაგალითად, მათში შეიძლება არ არსებობდეს ვირტუალური მეხსიერება, რომლის მხარდაჭერამაც პროგრამის შესრულების პროცესში შეიძლება გამოიწვიოს მისი შენელება, რისი წინასწარმეტყველებაც შეუძლებელია.



ნახ. 5 ოპერაციული სისტემის ფუნქციები

პროცესის ცნება

პროგრამის შესასრულებლად ოპერაციულმა სისტემამ უნდა გამოყოს გარკვეული რაოდენობის ოპერატიული მეხსიერება, მათ დაუკავშიროს შეტანა / გამოტანის გარკვეული მოწყობილობები ან ფაილები (საიდანაც უნდა მიიღოს შემომავალი მონაცემები და სადაც უნდა შეინახოს მიღებული შედეგები), ანუ რესურსების საერთო რაოდენობიდან უნდა მოახდინოს გარკვეული რაოდენობის დარეზერვება. მათი რაოდენობა და კონფიგურაცია დროის მიმდინარეობასთან ერთად შეიძლება შეიცვალოს. ასეთი აქტიური ობიექტების აღსაწერად ოპერაციულ სისტემაში

გამოვიყენებთ ტერმინს ‘‘პროცესი’’.
შეიძლება პროცესი გამოვიდეს ,,შესრულების‘‘ მდგომარეობიდან და პროცესის შეჩერება შეიძლება მოხდეს ორი მიზეზით: მისი შემდგომი შესრულებისთვის საჭირო გახდეს გარკვეული მოვლენის დაფიქსირება ან მიმდინარე პროცესისთვის განკუთვნილი დრო ამოიწუროს.
პროცესებთან მიმართებაში განასხვავებენ ერთჯერად და მრავალჯერად ოპერაციებს.
პროცესის წარმოქმნა / დასრულების ოპერაციები წარმოადგენენ ერთჯერად ოპერაციებს, ვინაიდან პროცესთან მიმართებაში მათი გამოყენება ხდება არაუმეტეს ერთისა. პროცესის მდგომარეობის შეცვლასთან დაკავშირებული ოპერაციები, ეს იქნება პროცესის ამუშავება შესასრულებლად თუ მისი ბლოკირება, როგორც წესი წარმოადგენენ მრავალჯერად ოპერაციებს.

პროცესის მართვის ბლოკი, მომხმარებლის კონტექსტი, პროცესის კონტექსტის გადართვა

პროცესის მართვის ბლოკში (Process Control Block, PCB) ინახება ინფორმაცია ოპერაციული სისტემის მიერ პროცესებზე შესრულებული აუცილებელი მოქმედებების განხორციელების შესახებ. ოპერაციული სისტემისთვის პროცესის მართვის ბლოკი წარმოადგენს მოდელს, რომელშიც ოპერაციული სისტემის მიერ პროცესებზე განხორციელებული ნებისმიერი ოპერაცია მასში იწვევს გარკვეულ ცვლილებას. მომხმარებლის თვალსაზრისით, განსხვავებით ოპერაციული სისტემისგან, უდიდესი მნი-შვნელობა აქვს პროცესის მისამართების სივრცის შემცველობას. ამიტომ პროცესის მისამართების სივრცეში ჩატვირთულ კოდს და მონაცემებს ვუწოდებთ მისი

მომხმარებლის კონტექსტს.
სისტემის საქმიანობა შედგება ოპერაციათა ჯაჭვისაგან, რომლებიც სრულდება სხვადასხვა პროცესებზე და თანდართულია პროცესების გადართვით ერთი პროცესიდან მეორეზე. ამისთვის აუცილებელია შესრულებადი პროცესის კონტექსტის შენახვა და იმ პროცესის კონტექსტის აღდგენა რომელზეც იქნა პროცესი გადართული. პროცესის ქმედითუნარიანობის შენახვა / აღდგენის ასეთ პროცედურას კონტექსტის გადართვა ეწოდება.

დაგეგმვის დონეები, ალგორითმებისთვის წაყენებული კრიტერიუმები და მოთხოვნები

პროცესის დაგეგმვა იყოფა დონეებად: გრძელვადიანი დაგეგმვა, მოკლევადიანი დაგეგმვა და შუალედური დონე. გრძელვადიანი დაგეგმვა პასუხისმგებელია სისტემაში ახალი პროცესის წარმოქმნაზე, მისი მულტიპროგრამირების ხარისხის განსაზღვრაზე, ანუ ერთდროულად მასში არსებული პროცესების რაოდენობის განსაზღვრაზე. პროცესის მოკლევადიანი დაგეგმვა ხორციელდება, მაგალითად, შესრულებადი პროცესის მიერ შეტანა / გამოტანის მოწყობილობის მიმართვისას ან უბრალოდ გარკვეული დროითი შუალედის ამოწურვისას. ამიტომ, როგორც წესი, მოკლევადიანი დაგეგმვა ხორციელდება ერთხელ არანაკლებ 100 მწ.-სა. ახალი პროცესის არჩევა ზემოქმედებს სისტემის ფუნქციონირებაზე, შემდგომი ანალოგიური მოვლენის დაგეგმვამდე. ზოგიერთ შემთხვევაში სისტემის წარმადობის ამაღლების მიზნით, მომგებიანია რომელიმე ნაწილობრივ შესრულებული პროცესის ოპერატიული მეხსიერებიდან დისკზე გადატანა, ხოლო მოგვიანებით მისი დაბრუნება შესრულების

შემდგომი გაგრძელებისთვის. თუ როდის და რომელი პროცესის გადატანაა საჭირო დისკზე და უკან დაბრუნება, წყდება პროცესის დაგეგმვის დამატებითი დონით – რომელსაც შუალედური დონე ეწოდება. პროცესების დაგეგმვის ყოველი დონისთვის შეიძლება შეთავაზებული იყოს მრავალი ალგორითმი. თუ რომელი ალგორითმი იქნას არჩეული, ამის განსაზღვრისთვის ყურადღება ექცევა გამოთვლითი სისტემის ამოსახსნელ ამოცანის კლასს და იმ მიზანს, რომლის მიღწევაც გვსურს ამ დაგეგმვის გამოყენებით. ეს მიზნებია: სამართლიანობა(გამოთვლით სისტემაში ყოველი ამოცანის ან პროცესისთვის პროცესორის გამოყენების გარანტირება დროის რაღაც შუალედით, ამასთან არ უნდა შეიქმნას სიტუაცია, რომ რომელიღაც პროცესი მუდმივად სარგებლობდეს პროცესორით როცა სხვა თუნდაც ერთ პროცესს არც კი დაუწყია შესრულება), ეფექტურობა(მუშაობის პროცესში პროცესორის 100%-ით დატვირთვა) შესრულების მთელი დროის შემცირება (მინიმალური დროის უზრუნველყოფა პროცესორის ჩართვასა და გამორთვას შორის), ლოდინის დროის შემცირება, (პროცესების მზადყოფნის მდგომარეობაში ყოფნის და ამოცანების მიმდევრობაში ჩატვირთვის დროის შემცირება) გამოხმაურების დროის შემცირება(პროცესის მომხმარებლის მოთხოვნის საპასუხოდ განკუთვნილი დროის შემცირება). ალგორითმებს ასევე უნდა გააჩნდეთ თვისებები: იყვნენ წინასწარმეტყველირებადნი(ერთი და იმავე ამოცანა ერთი და იმავე დროში სრულდებოდეს), უნდა იყვნენ დაკავშირებული მინიმალურ ზედნადებ დანახარჯებთან(არ იხარჯებოდეს ‘‘უაზროდ’’ პროცესისთვის პროცესორის გამოყენებისთვის განკუთვნილი დრო), გამოთვლითი სისტემის რესურსები

იტვირთებოდნენ თანაბრად, იმ პროცესებისთვის უპირატესობის მინიჭებით, რომლებიც ნაკლებად გამოყენებად რესურსებს იკავებენ. გააჩნდეთ მასშტაბურობა(ანუ დატვირთვის გაზრდის შემთხვევაში არ დაკარგონ ქმედითუნარიანობა).

განასხვავებენ გაძევებად და გაუძევებად დაგეგმვას. თუ ოპერაციულ სისტემაში დაგეგმვა ხორციელდება მხოლოდ იძულებით სიტუაციაში, მაშინ ამბობენ, რომ ადგილი აქვს გაუძევებად დაგეგმვას. თუკი დამგეგმავი ღებულობს, როგორც იძულებით ისე არა იძულებით გადაწყვეტილებებს, მაშინ ამბობენ, რომ ადგილი აქვს გაძევებად დაგეგმვას.

დაგეგმვის ალგორითმები:
FჩFშ, ღღ, შჟF, გარანტირებული დაგეგმვა, პრიორიტეტული დაგეგმვა, მრავალდონიანი მიმდევრობები უკუკავშირით

Fირსტ-ჩომე, Fირსტ-შერვედ - პირველი მოვიდა პირველს მოემსახურნენ. ეს ალგორითმი ანხორციელებს გაუძევებად დაგეგმვას და სრულდება შემდეგნაირად: პროცესორი გადაეცემა მიმდევრობაში პირველ პროცესს და ის პროცესორს იკავებს მიმდინარე ჩPU ბურსტ-ის(მომსახურერბის დროის) ამოწურვამდე. ამის შემდეგ შესასრულებლად ახალი პროცესი აირჩევა მიმდევრობის თავიდან. ამ ალგორითმის უპირატესობას მხოლოდ მისი სიმარტივე წარმოადგენს, ხოლო მისი ნაკლოვანება ისაა, რომ დროის დაყოფის სისტემისთვის (რეჟიმისთვის) პრაქტიკულად გამოუსადეგარია, რადგან მოითხოვს საკმაოდ დიდ საშუალო დროს.
ღოუნდ ღობინ (ღღ). ეს ალგორითმი წარმოადგენს FჩFშ ალგორითმის მოდიფიკაციას. იმ განსხვავებით, რომ ამ

ალგორითმში რეალიზებულია გაძევებადი დაგეგმვის რეჟიმი. შეიძლება ითქვას, რომ პროცესების დაგეგმვა რეალიზებულია ციკლის სახით – პროცესები განთავსებული არიან კარუსელზე. კარუსელი ტრიალებს და ყოველი პროცესი იკავებს პროცესორს გარკვეული დროითი კვოტის განმავლობაში.

შჰორტესტ-ჟობ-Fირსტ (შჟF). ამ ალგორითმში გათვალისწინებულია მზა პროცესების მიმდევრობაში პროცესების განლაგება. თუ პროცესების ჩPU ბურსტ წინასწარ გვეცოდინება, მაშინ შესასრულებლად პროცესებს ავირჩევთ არა მიმდევრობით, არამედ ჩPU ბურსტ-ის ხანგრძლივობის მიხედვით. (ავირჩევთ მინიმალურს). თუ ასეთი პროცესების რაოდენობა აღმოჩნდა ერთზე მეტი, მაშინ მათ შორის ერთის ასარჩევად ვიყენებთ FჩFშ ალგორითმს. ეს ალგორითმი შეიძლება იყოს როგორც გაძევებადი ასევე გაუძევებადი. გაუძევებადი შჟF ალგორითმის შესრულებისას პროცესს გადაეცემა პროცესორი, მიუხედავად გამოთვლით სისტემაში მიმდინარე მოვლენებისა, მისთვის აუცილებელი მთელი დროით. გაძევებადი შჟF ალგორითმის შესრულებისას ხდება მზა პროცესების მიმდევრობაში გამოჩენის გათვალისწინება და თუ ახალი პროცესისთვის საჭირო დრო ნაკლებია რიგში მდგომ პროცესისთვის აუცილებელ დროზე, მაშინ პროცესორი გადაეცემა ახლად გამოჩენილ პროცესს.

გარანტირებული დაგეგმვა.

N მომხმარებელი სისტემაში გადავნომროთ 1-დან N-მდე. ყოველ მომხმარებელისთვის შემოვიღოთ ორი სიდიდე: Ti - სისტემაში მომხმარებლის ყოფნის დრო და τi –სისტემაში ყოფნის განმავლობაში მისი პროცესებისთვის გამოყოფილი ჯამური პროცესორული დრო. სამართლიანი იქნება

მომხმარებლისთვის Ti/NNპროცესორული დროის მიღება(გამოყოფა). თუ $\tau i \ll Ti/N$, მაშინ ამ მომხმარებლისთვის არასწორადაა გამოყოფილი პროცესორული დრო, ხოლო თუ $\tau i \gg Ti/N$, მაშინ პირიქით, სისტემა ამ მომხმარებლის მხარესაა. ყოველი მომხმარებლისთვის გამოითვლება სამართლიანობის კოეფიციენტი: $\tau iN/Ti$ და მზა პროცესს დროით კვანტად შეუსაბამდება უმცირესი.
ამ ალგორითმს გარანტირებულად დაგეგმვა ეწოდება და მისი ნაკლი არის ის, რომ წინასწარ შეუძლებელია მომხმარებლის ყოფაქცევის წინასწარმეტყველება.
პრიორიტეტული დაგეგმვა. შჟF და გარანტირებულად დაგეგმვის ალგორითმები წარმოადგენენ პრიორიტეტულ დაგეგმვის კერძო შემთხვევებს. პრიორიტეტული დაგეგმვისას ყოველ პროცესს ენიჭება რიცხვითი მნიშვნელობა – პრიორიტეტი, რომლის შესაბამისადაც მათ გამოეყოფა პროცესორი. ერთნაირი პრიორიტეტის შემთხვევაში ხორციელდება FჩFშ მიმდევრობით. შჟF ალგორითმისთვის ასეთი პრიორიტეტი იქნება ჩPU ბურსტ დროის ხანგრძლივობა, რადგან რაც უფრო ნაკლებია ჩPU ბურსტ-ის მნიშვნელობა, მით უფრო მაღალია პროცესის პრიორიტეტი. გარანტირებულად დაგეგმვის შემთხვევაში პრიორიტეტის როლში გამოდის სამართლიანობის კოეფიციენტი, რადგან რაც უფრო ნაკლებია სამართლიანობის კოეფიციენტი, მით უფრო მაღალია მისი პრიორიტეტი.
მრავალდონიანი მიმდევრობები უკუკავშირით. სისტემებისთვის, რომლებშიც პროცესები ადვილად შესაძლებელია იყვნენ დახარისხებულნი სხვადასხვა ჯგუფებად, შემუშავებული იყო დაგეგმვის ალგორითმების

სხვა კლასი. პროცესების ყოველ ჯგუფს, რომლისთვისაც იქმნება მზა პროცესების საკუთარი მიმდევრობა, სადაც მიმდევრობებს ენიჭებათ ფიქსირებული პრიორიტეტები და გააჩნიათ უკუკავშირის მექანიზმები, ვუწოდებთ მრავალდონიან მიმდევრობას უკუკავშირით. აქ პროცესი მუდმივად კი არ მიეკუთვნება გარკვეულ მიმდევრობას, არამედ შესაძლებელია საკუთარი ყოფაქცევის მიხედვით მიგრირებული იყოს ნებისმიერ მიმდევრობაში

ინფორმაციის გაცვლის საშუალებები

პროცესები ერთმანეთთან ურთიერთქმედებენ მხოლოდ ინფორმაციის გაცვლის გზით. გაცვლის ასეთი საშუალებები შეიძლება დაიყოს სამ კატეგორიად: სიგნალური _ გადაიცემა ინფორმაციის მინიმალური რაოდენობა ერთი ბიტი ‘‘კი’’ ან ‘‘არა’’, არხული _ პროცესების ‘‘საუბარი’’ ხორციელდება ოპერაციული სისტემის მიერ წარმოქმნილი კავშირის არხებით, ჩანაწერების, წერილების და განცხადებების მეშვეობით, განაწილებადი მეხსიერება _ ორ ან რამდენიმე პროცესს შეუძლია შეთანხმებულად გამოყოს მისამართების სივრცის გარკვეული ნაწილი.

დამისამართება, კავშირის მიმართულობა

განასხვავებენ დამისამართების ორ მეთოდს: პირდაპირს და არაპირდაპირს. პირდაპირი დამისამართებისას ურთიერთმოქმედი პროცესები უშუალოდ ურთიერთობენ

ერთმანეთთან, მონაცემების გაცვლის ყოველი ოპერაციისას იმ პროცესის სახელისა და ნომრის ცხადად მითითებით, რომლისთვისაცაა ინფორმაცია განკუთვნილი ან რომლისგანაც ის უნდა იყოს მიღებული. თუ პროცესი, რომლისგანაც გამოდის მონაცემები, და პროცესი, რომელიც იღებს ამ ინფორმაციას, უთითებს თავისი ურთიერთქმედების პარტნიორს, მაშინ დამისამართების ასეთ სქემას სიმეტრიული პირდაპირი დამისამართება ეწოდება. თუ ურთიერთმოქმედი პროცესებიდან მხოლოდ ერთი, მაგალითად გამგზავნი უთითებს თავისი ურთიერთქმედების პარტნიორის სახელს, ხოლო მეორე პროცესი შესაძლოა პარტნიორად განიხილავს სისტემაში ნებისმიერ პროცესს, მაშინ ასეთ დამისამართებას ასიმეტრიული პირდაპირი დამისამართება ეწოდება. არაპირდაპირი დამისამართებისას გამგზავნი პროცესის მიერ გაგზავნილი მონაცემები შესანახად თავსდება გარკვეულ შუალედურ ობიექტში, რომელსაც გააჩნია მისამართი, საიდანაც ის შეიძლება იყოს აღებული რაიმე სხვა პროცესის მიერ. ერთმიმართულებიანი კავშირის ქვეშ იგულისხმება ისეთი კავშირი, რომლის დროსაც მასთან ასოცირებულ პროცესს კავშირის ეს საშუალება შეუძლია გამოიყენოს მხოლოდ ან ინფორმაციის მისაღებად, ან მის გადასაცემად. ორმიმართულებიან კავშირში ყოველ პროცესს, რომელიც მონაწილეობს ურთიერთქმედებაში, შეუძლია გამოიყენოს კავშირი მონაცემების მისაღებად და მის გადასაცემად. კომინიკაციურ სისტემებში ერთმიმართულებიან კავშირს ეწოდება სიმპლექსური. ორ მიმართულებიანს, სხვადასხვა მიმართულებით თანმიმდევრობით გადაცემის შესაძლებლობით – ნახევრადდუპლექსური, ხოლო მონაცემების ერთდროულად

გადაცემის შესაძლებლობით კი - დუპლექსური.

ამასთანავე, საჭიროა კომუნიკაციური საშუალების საიმედოობის საკითხიც დავაზუსტოთ:
კომუნიკაციის საშუალებას ეწოდება საიმედო, თუ მონაცემთა გაცვლისას სრულდება შემდეგი ოთხი პირობა:

- არ ხდება ინფორმაციის დაკარგვა.
- არ ხდება ინფორმაციის დაზიანება.
- არ ჩნდება ზედმეტი ინფორმაცია.
- არ ირღვევა მონაცემთა თანმიმდევრობა გაცვლის **პროცესში.**

საკითხი სინქრონიზაციის აუცილებლობის შესახებ

მრავალნაკადურ სისტემებში სინქრონიზაციის უგულებელ-ყოფამ შეიძლება ამოცანის არასწორ ამოხსნამდე და თვით ოპერაციული სისტემის კრახამდე მიგვიყვანოს.
განვიხილოთ მაგალითი საწარმოს კლიენტების მონაცემთა ბაზის ფორმირების შესახებ. ყოველ კლიენტს ცალკეული ჩანაწერი გამოეყოფა მონაცემთა ბაზაში, სადაც სხვადასხვა ველებს შორის არსებობს შეკვეთებისა და გადახდის ველი. პროგრამა, რომელიც უძღვება მონაცემთა ბაზას, გაფორმებულია როგორც ერთიანი პროცესი, რომელსაც აქვს რამდენიმე ნაკადი, მათ შორის A ნაკადი, რომელსაც შეაქვს მონაცემთა ბაზაში ინფორმაცია კლიენტებიდან მოსული შეკვეთების შესახებ და ნაკადი B, რომელიც აფიქსირებს მონაცემთა ბაზაში ცნობებს

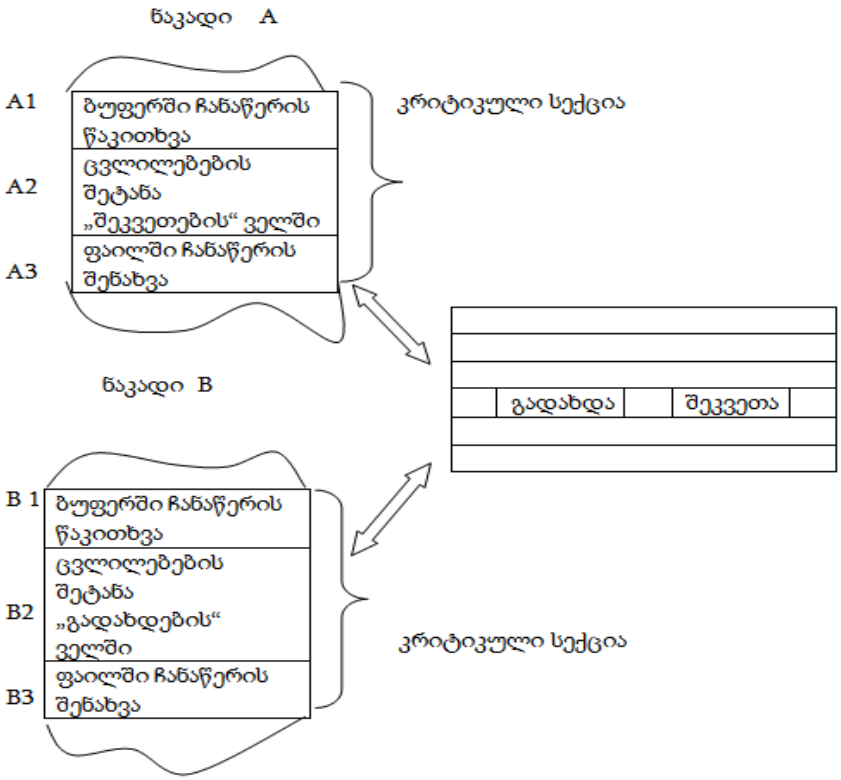
კლიენტებისადმი წაყენებული ანგარიშების განაღდების შესახებ. ორივე ეს ნაკადი ერთობლივად მუშაობს მონაცემთა ბაზის საერთო ფაილზე, რომელიც მოიცავს ერთი ტიპის ალგორითმებს და შედგება სამი ბიჯისაგან.

1. გამოთვალოს მონაცემთა ბაზის ფაილისაგან ჩანაწერი ბუფერში კლიენტის შესახებ მოცემული იდენტიფიკატორის შესაბამისად.
2. შეიტანოს ახალი მნიშვნელობა შკვეთების ველში (A ნაკადისათვის) ან ანაზღაურება (B ნაკადისათვის)
3. დააბრუნოს მოდიფიცირებული ჩანაწერი მონაცემთა ბაზის ფაილში .

აღვნიშნოთ შესაბამისი ბიჯები A ნაკადისათვის A1, A2 და A3, ხოლო B1, B2 და B3 კი B ნაკადისათვის. დავუშვათ, რომ რაღაც მომენტში A ნაკადი N კლიენტის შესახებ შეკვეთას განაახლებს. ამისათვის ის თავის ბუფეში აკეთებს შესაბამის ჩანაწერს. (A1ბიჯი). ხდება შეკვეთების ველის მნიშვნელობის მოდიფიცირება (A2ბიჯი). მაგრამ მონაცემთა ბაზაში ჩანაწერის გაკეთებას ვერ ასწრებს, რადგან მისი შესრულება წყდება მაგ. დროის კვოტის დასრულების გამო.
დავუშვათ აგრეთვე, რომ ნაკადს დავადეთ მოთხოვნილება, შევიტანოთ ცნობები იმავე N კლიენტის გადახდასთან დაკავშირებით. როცა B ნაკადის ჯერი დგება, ის ასწრებს იანგარიშოს ჩანაწერი თავის ბუფერში(B1ბიჯი) და შეასრულოს გადახდის ველის განახლება (B2ბიჯი) და შემდეგ კი წყდება. შევნიშნოთ, რომ B ნაკადის ბუფერში იმყოფება ჩანაწერი N კლიენტის შესახებ, რომელშიც შეკვეთების ველში არის ძველი შეუცვლელი მნიშვნელობა.

როცა მომდევნო ჯერზე მართვა გადაეცემა A ნაკადს, ის აგრძელებს რა, თავის სამუშაოს, ჩაწერს ჩანაწერს N კლიენტის შესახებ, შეკვეთების მოდიფიცირებულ ველში მონაცემთა ბაზაში (A3ბიჯი). A ნაკადის შეწყვეტისა და B ნაკადის აქტივიზაციის შემდეგ, უკანასკნელი ჩაწერს მონაცემთა ბაზაში

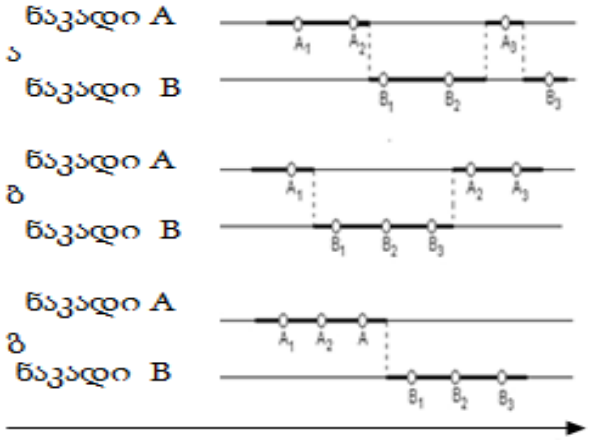
ჩანაწერს ზემოთ ამ წამს განახლებულ ჩანაწერს N კლიენტის შესახებ, ჩანაწერის საკუთარ ვარიანტს, რომელშიც



ნახ.6 შესაძლო ნაკადები განცალკევებული მონაცემების წვდომის დროს

განახლებულია გადახდის ველის მნიშვნელობა. ამგვარად, მონაცემთა ბაზაში დაფიქსირებული იქნება ცნობები იმის შესახებ, რომ N კლიენტმა განახორციელა გადახდა, მაგრამ მის შეკვეთის შესახებ ინფორმაცია დაკარგული აღმოჩნდება. ნახ.6.ა. სინქრონიზაციის პრობლემის სირთულის პოზიტიური გაგება

წარმოქმნილი სიტუაციების რეგულარულობაში მდომარეობს. მაგ. წინა მაგალითში შეიძლება წარმოვიდგინოთ მოვლენათა სულ სხვაგვარი განვითარება: შეიძლება დაკარგულიყო ინფორმაცია არა შეკვეთის, არამედ გადახდის შესახებ (ნახ.6.ბ). ან პირიქით, ყველა ცვლილებები შეტანილ იქნას წარმატებით. (ნახ.6.გ). ყველაფერი განისაზღვრება ნაკადების ერთმანეთის მიმართ (ფარდობითი) სიჩქარეებით და მათი წყვეტის მომენტებით. ამიტომ ურთიერთზემოქმედი ნაკადების გამართვა წარმოადგენს რთულ სისტემას. იმის მსგავს სიტუაციებს, როცა ორი ან მეტი ნაკადი ამუშავებს განცალკევებულ მონაცემებს და საბოლოო შედეგი დამოკიდებულია ნაკადების ფარდობით სიჩქარეებზე, ეწოდება **რბოლა.**



ნახ.7 ნაკადების ფარდობითი სიჩქარეების გავლენა ამოცანის ამოხსნის შედეგზე.

მოთხოვნები სინქრონიზაციის ალგორითმის მიმართ

კრიტიკული მონაკვეთებისათვის (სექციებისათვის) ურთიერთგამორიცხვის ორგანიზაცია, ცხადია, საშუალებას იძლევა თავი ავარიდოთ რბოლის მდგომარეობის(race condition)

წარმოშობას, მაგრამ საკმარისი არ არის კორპორატიული პროცესების სწორი და ეფექტური პარალელური მუშაობისათვის. მოგვყავს ხუთი პირობა რომელიც უნდა შესრულდეს ურთიერთზემოქმედი პროცესების ორგანიზების კარგი პროგრამული ალგორითმისათვის, რომლებსაც აქვთ კრიტიკული მონაკვეთები და შეუძლიათ მათი გავლა თავისუფალი რიგით:

1. ამოცანა უნდა ამოიხსნას წმინდა პროგრამული წესებით, სადაც არ იქნება ურთიერთგამორიცხვის კოდები. ამ დროს იგულისხმება, რომ პროგრამირების ენის ძირითადი ინსტრუქციები წარმოადგენენ ატომალურ ოპერაციებს.
2. არ შეიძლება არანაირი წინასწარი დაშვებები ფარდობითი სიხშირეების შესახებ შესასრულებელი პროცესებისა და /ან პროცესორებისა, რომლებზედაც ისინი სრულდება.
3. თუ პროცესი Pi არის თავის კრიტიკულ მონაკვეთზე, მაშინ დაუშვებელია რაიმე სხვა პროცესის არსებობა, რომელიც სრულდება თავის შესაბამის კრიტიკულ სექციაში. ამ პირობას ეწოდება ურთიერთგამორიცხვის პირობა. (mutual exclusion).
4. პროცესები, რომლებიც თავის კრიტიკულ მონაკვეთებზე არ იმყოფებიან და არც აპირებენ მასში შესვლას, არ უნდა უშლიდეს ხელს სხვა პროცესებს, რომ შედიოდნენ თავის კრიტიკულ მონაკვეთებზე. თუ არ არიან პროცესები თავის კრიტიკულ მონაკვეთებზე და არსებობენ პროცესები, რომლებსაც იქ შესვლა სურთ, მაშინ მხოლოდ იმ პროცესებმა, რომლებიც არიან დარჩენილი სექციაში და არ გამოიყენებიან ამ დარჩენილ სექციაში(remainder section)-უნდა მიიღონ გადაწყვეტილება იმის შესახებ, თუ რომელი პროცესი შევიდეს თავის შესაბამის კრიტიკულ სექციაში. ეს გადაწყვეტილება უსასრულოდ დიდხანს არ უნდა გაგრძელდეს. ამას ეწოდება პროგრესის (progress) პირობა.
5. არ უნდა ჰქონდეს ადგილი პროცესების თავის კრიტიკულ მონაკვეთებზე შესვლისათვის უსასრულო ლოდინს. იმ

მომენტიდან როცა პროცესმა მოითხოვა შესვლა თავის კრიტიკულ მონაკვეთებზე, სექციაში, და იმ მომენტამდე, როცა მან ეს უფლება მოიპოვა, სხვა პროცესებმა შეიძლება გაიარონ თავისი კრიტიკული მონაკვეთები მხოლოდ შეზღუდული რაოდენობით, ისე, რომ ხელი არ შეუსალონ ზემოხსენებულ შესასრულებელ პროცესს. ამას ეწოდება შეზღუდული ლოდინის პირობა (bound waiting).

აღსანიშნავია, რომ შესაბამისი ალგოროთმის აღწერა წარმოადგენს პროლოგისა და ეპილოგის ორგანიზების წესის აღწერას კრიტიკული სექციებისათვის.

სინქრონიზაციის ალგორითმი

1. წყვეტების აკრძალვა

დასმული ამოცანა ყველაზე ადვილად წყდება შემდეგი წესით:

while (პირობა) {

აიკრძალოს ყველა წყვეტა

კრიტიკული სექცია

დასაშვებია ყველა წყვეტა

ნარჩენი კოდი }

რამდენადაც, პროცესის გამოსვლა შესრულების მდგომარეობიდან სრულდება მისი დასრულების გარეშე, წყვეტებით, ამიტომ კრიტიკული სექციის შიგნით არავის არ შეუძლია ჩაერიოს მის მუშაობაში. ამასთან, ასეთ ამოხსნას ექნება(გამოიწვევს) შორს მიმავალი შედეგები, რადგანაც შანსს აძლევს პროცესის მომხმარებელს ნება დაერთოს და დაამთავროს

წყვეტა მთელს გამოთვლით სისტემაში.

დავუშვათ, რომ შეცდომის ან ბოროტი ჩანაფიქრის შედეგად მომხმარებელმა აკრძალა სისტემაში წყვეტა და ჩაციკლა ან დაასრულა თავისი პროცესი. სისტემის გადატვირთვის გარეშე ამ სიტუაციიდან ვერ გამოვალთ.

უფრო მეტიც, წყვეტის აკრძალვა ან გაშვება ხშირად გამოიყენება როგორც პროლოგი და ეპილოგი კრიტიკულ სექციებთან თავად ოპერაციული სისტემის შიგნითაც.

ცვლადი - კლიტე

ამოცანის გადაწყვეტის შემდგომი მცდელობის სახით სამომხმარებლო პროცესებისათვის განვიხილოთ შემდეგი წინადადება: ავიღოთ რაღაც ცვლადი რომელიც ხელმისაწვდომია ყველა პროცესისათვის და დავუშვათ მისი საწყისი მნიშვნელობა ტოლია 0 -ის. პროცესი შეიძლება შემოვიდეს კრიტიკულ სექციაში, როცა ამ ცვლადი-კლიტის მნიშვნელობა გახდება 0, ამავდროულად მისი მნიშვნელობის 1-იანით შეცვლით კლიტე იკეტება. კრიტიკული სექციიდან გამოსვლის დროს პროცესი მნიშვნელობას „ისვრის“,აქცევს მას ნულად და კლიტე გაიღება.

```
shared int lock = 0;
while (პირობა) {
while(კლიტე); lock = 1;
კრიტიკული სექცია
კლიტე = 0;
ნარჩენი კოდი }
```

სამწუხაროდ, ყურადღებით შესწავლა გვიჩვენებს, რომ ასეთი გადაწყვეტა არ აკმაყოფილებს ურთიერთგამორიცხვის პირობას, რადგან მოქმედება **while(კლიტე); lock = 1** არ წარმოადგენს ატომარულს. დავუშვათ, პროცესმა P0 გაატესტირა კლიტის ცვლადის მნიშვნელობა და მიიღო გადაწყვეტილება გაგრძელებაზე. მიიღოს „0“ მნიშვნელობა. ე.ი. გააქტიურდეს. ამ მომენტში, ჯერ კიდევ კლიტის ცვლადისათვის 1 მნიშვნელობის

მინიჭებამდე, დამგეგმავმა მართვა გადასცა P1პროცესორს. ის ასევე შეისწავლის კლიტის ცვლადის შინაარსს და აგრეთვე მიიღებს გადაწყვეტილებას შევიდეს კრიტიკულ მონაკვეთში. ამანაც მიიღოს „0“ მნიშვნელობა.ჩვენ ორ პროცესს ერთდროულად ვღებულობთ, რომლებიც ასრულებენ თავის კრიტიკულ სექციებს.

პერეტსონის ალგორითმი

პრობლემის პირველი გადაწყვეტა, რომელიც ადრე განხილული ალგორითმების ყველა მოთხოვნას და გამოყენებულ იდეებს აკმაყოფილებდა, შემოთავაზებული იყო დანიელი მათემატიკოსის დეკკერის მიერ(Dekker) და 1981 წელს პეტერსონმა (Peterson) უკეთესი გადაწყვეტა შემოგვთავაზა. ვთქვათ, ორივე პროცესი ფლობს მზადყოფნის წვდომას მასივების ალმებთან და რიგითობის ცვლადებთან.

```
shared int ready[2] = {0, 0};

shared int turn;

while (პირობა) {

ready[i] = 1;

turn =1- i;

while(ready[1-i] && turn == 1-i);

კრიტიკული სექცია

ready[i] = 0;

ნარჩენი კოდი

}
```

კრიტიკული სექციის პროლოგის შესრულების დროს პროცესი Pi განაცხადებს თავის მზადყოფნის შესახებ, რომ შეასრულოს კრიტიკული მონაკვეთი და ამავდროულად გადასცემს (სთავაზობს) მეორე პროცესს, შეუდგეს, მის

შესრულებას. თუკი ორივე პროცესი პროლოგთან პრაქტიკულად ერთდროულად მივიდა, ისინი ორივენი გამოაცხადებენ თავიანთ მზადყოფნის შესახებ და ერთმანეთს შესთავაზებენ შესრულებას. ამ შემთხვევაში ერთი შეთავაზება ყოველთვის მიედევნება მეორეს. ამასთანავე, კრიტიკულ მონაკვეთზე მუშაობას განაგრძობს პროცესი, რომელმაც მიიღო ბოლო შეთავაზება.

დავასაბუთოთ ურთიერთგამორიცხვის პირობის შესრულება საწინააღმდეგოს მეთოდით. ვთქვათ, ორივე პროცესი ერთდროულად აღმოჩნდნენ თავიანთი კრიტიკული სექციის შიგნით. შევნიშნოთ, რომ Pi პროცესი შეიძლება შემოვიდეს კრიტიკულ სექციაში, მხოლოდ მაშინ, თუ [1-i] == 0 ანუ turn == i. აღვნიშნოთ ისიც, რომ თუ ორივე პროცესი ერთდროულად ასრულებს თავიანთ კრიტიკულ სექციებს, მაშინ მზადყოფნის ალმების მნიშვნელობები ორივე პროცესისათვის დაემთხვევა ერთმანეთს და =1. შეეძლოთ, კი ორივე პროცესი ერთდროულად აღმოჩენილიყვნენ თავიანთ კრიტიკულ სექციაში იმ მდგომარეობიდან, რაშიც ისინი ერთდროულად იყვნენ while-ციკლის შესრულების პროცესში? არა! რადგან ამ შემთხვევაში turn ცვლადმა ერთდროულად უნდა მიიღოს მნიშვნელობა 0 და 1 (როცა ორივე პროცესი ასრულებს ციკლს, მაშინ ცვლადების მნიშვნელობების შეცვლა არ შეუძლიათ). ვთქვათ, P0 პროცესი პირველი შევიდა კრიტიკულ მონაკვეთში. მაშინ P1 -მა უნდა შეასრულოს while ციკლში შესვლის წინ, უკიდურეს შემთხვევაში, ერთი წინასწარი(turn = 0;) ოპერატორი. ამასთან, ამის შემდეგ, მას არ შეეძლება გამოვიდეს ციკლიდან, ვიდრე არ დამთავრდება კრიტიკული მონაკვეთი P0 პროცესისა, რადგანაც ციკლ ready[0] == 1 და turn == 0, შესვლის წინ, არც ეს მნიშვნელობები არ შეიძლება შეიცვალონ, მანამდე, სანამ P0 პროცესი არ დასტოვებს თავის კრიტიკულ უბანს. მივედით წინააღმდეგობამდე, ამრიგად, ადგილი აქვს ურთიერთგამორიცხვადობას.

ახლა დავამტკიცოთ პროგრესის პირობის შესრულება. ზოგადობის შეუზღუდავად ავიღოთ P0 პროცესი. აღვნიშნოთ, რომ მას არ შეუძლია შევიდეს თავის კრიტიკულ სექციაში

მხოლოდ ერთდროულად შესრულების დროს პირობებისა: ready[1] == 1 და turn == 1. თუ P1 პროცესი არ არის მზად, შეასრულოს თავისი კრიტიკული უბანი, მაშინ ready[1] == 0 და პროცესმა P0 შეიძლება განახორციელოს შესვლა. თუკი P1 პროცესი არის მზად შეასრულოს თავისი კრიტიკული უბანი, მაშინ ready[1] == 1 და turn == 0. ან 1, რაც საშუალებას აძლევს ან P0 ან P1 პროცესებს, დაიწყონ კრიტიკული უბნების შესრულება. თუ P1 დაამთავრა კრიტიკული უბნის შესრულება, ის გადააგდებს თავის მზადყოფნის ალამს ready[1] == 0,. და შესთავაზებს მეორე P0 პროცესს, შეუდგეს, თავის კრიტიკული საქმის შესრულებას. ასე, რომ პროგრესის პირობა სრულდება.

აქედანვე გამომდინარეობს, რომ სრულდება შეზღუდული ლოდინის პირობაც. რადგანაც ნებართვაზე ლოდინის პროცესში P0 პროცესი დასაწყისში არ იცვლის მნიშვნელობების მნიშვნელობებს, ამიტომ მას შეუძლია დაიწყოს თავისი კრიტიკული უბნის შესრულება, იმის შემდეგ როცა გავიდა P1პროცესის არაუმეტეს ერთი კრიტიკული სექცია.

საფუნთუშის ალგორითმი Bakery algorithm

პეტერსონის ალგორითმი გვაძლევს ორი პროცესის ურთიერთქმედების კორექტული ორგანიზაციის ამოცანის ამოხსნას. ახლა განვიხილოთ n ურთიერთმოქმედი პროცესის შესაბამისი ალგორითმი. საფუნთუშის ალგორითმის Bakery algorithm მთავარი იდეა ასეთია: ყოველი ხელახლად გამოჩენილი კლიენტი(იგივე პროცესია) ღებულბს მომსახურების ნომრიან ტალონს. მომდევნო ჯერზე მოემსახურებიან კლიენტს უმცირესი ნომრით. სამწუხაროდ, შემდეგი ნომრის გამოთვლის ოპერაციის არაატომურობის გამო, საფუნთუშის ალგორითმი გარანტიას არ იძლევა რომ ყველა პროცესს ექნება ტალონი განსხვავებული ნომრით. ორ და მეტ კლიენტის ტალონზე ნომრების დამთხვევის შემთხვევაში სახელები შეიძლება შევადაროთ

ლექსიკოგრაფიკული წესით. პირველად მომსახურება ხვდება მოკლე სახელიანს. მონაცემთა დისკრეტული სტრუქტურები ალგორითმისათვის - ეს ორი მასივია.

shared enum {false, true} choosing[n];

shared int number[n];

თავდაპირველად ამ მასივების ელემენტები ინიცირდებიან false და 0 მნიშვნელობებით შესაბამისად. შემოვიტანოთ შემდეგი აღნიშვნები:

(a,b) < (c,d), if a < c else if a == c и b < d

max(a0, a1,...., an) — ეს რიცხვი k ისეთია, რომ k >= ai ყველა i = 0,...,n

Piპროცესის სტრუქტურა საფუნთუშის ალგორითმისათვის ასეთია:

while (პირობა) {

choosing[i] = true;

number[i] = max(რიცხვი[0],..., რიცხვი[n-1]) + 1;

choosing[i] = false;

for(j = 0; j < n; j++){

while(choosing[j]);

while(რიცხვი[j] != 0 && (რიცხვი[[j],j) < (რიცხვი[[i],i));

}

კრიტიკული სექცია

(რიცხვი[[i] = 0;

ნარჩენი კოდი}

ურთიერთ გამორიცხვა შუქნიშნების მაგალითზე

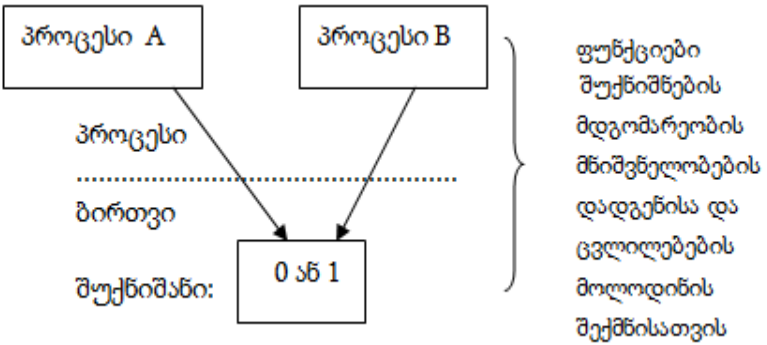
ოპერაციული სისტემების თეორიაში შუქნიშანი წარმოადგენს არაუარყოფთ მთელ ცვლადს, რომელზედაც შესაძლებელია ორი ოპერაცია: P და V.

P ოპერაცია შუქნიშანზე ნიშნავს მისი მნიშვნელობის ერთი ერთეულით შემცირებას. თუკი P ოპერაციის შესრულების წინ შუქნიშნის მნიშვნელობა იყო 0- ზე მეტი, მაშინ P ოპერაციის შესრულება დაუბრკოლებელია, თუკი P ოპერაციის შესრულების წინ შუქნიშნის მნიშვნელობა იყო 0, მაშინ P ოპერაციით შესარულებელი პროცესი გადადის ლოდინის მდგომარეობაში მანამ, სანამ შუქნიშნის მნიშვნელობა 0- ზე მეტი არ გახდება. V-ოპერაცია შუქნიშანზე ნიშნავს მისი მნიშვნელობის ერთი 1 ერთეულით გადიდებას. თუ ამ დროს არსებობენ P-ოპერაციის შესრულების გამო, ამ შუქნიშანზე შეჩერებული პროცესები, მაშინ ამ პროცესებდან ერთი გამოდის ლოდინის მდგომარეობიდან და შეუძლია შესარულოს თავისი P ოპერაცია. შუქნიშნებს, რომლებიც იღებენ მნიშვნელობებს 0 და 1, უწოდებენ ორნიშნებს. მთვლელი შუქნიშნები(რომლებსაც აქვთ მრიცხველები) ღებულობენ მთელ, არაუარყოფით მნიშვნელობებს, 2-ზე მეტს.

P და V ოპერაციები განუყოფლები არიან. ეს ნიშნავს, რომ დროის ყოველ მომენტში მხოლოდ ერთ პროცესს შეუძლია შეასრულოს P ან V შუქნიშნის მონაცემებზე. ოპერაციების განუყოფლობა იმასაც ნიშნავს, რომ თუკი რამდენიმე პროცესი არის დაკავებული Pოპერციით, მაშინ მათ შორის მხოლოდ ერთს შეუძლია წარმატებით დაასრულოს ეს თავისი ოპერაცია, თუკი შუქნიშნის მნიშვნელობა გახდა 0-ზე მეტი. ამ დროს არავითარი დაშვებები არ კეთდება იმის შესახებ, რომ, როგორი იქნება ეს პროცესი.

ყოველ შუქნიშანს უკავშირდება სია(პროცესების რიგი),

რომლებიც ელოდებიან ნებართვას შუქნიშანზე გავლის თაობაზე. ოპერაციულ სისტემას შეუძლია პროცესებზე შეასრულოს სამი ოპერაცია: 1. მან შეიძლება დანიშნოს შესასრულებლად მზა პროცესი. 2. შეუძლია დაბლოკოს შესრულებადი პროცესი და მოათავსოს ის სიაში, რომელიც დაკავშირებულია კონკრეტულ შუქნიშანთან. 3. მას შეუძლია პროცესის დებლოკირება და ამასთანავე, გადაიყვანოს იგი შესასრულებლად მზა მდგომარეობაში და ნება დართოს მას, ოდესმე განაახლოს შესრულებადობა. იმყოფება, რა დაბლოკილთა სიაში, მომლოდინე პროცესს არ ამოწმებს შუქნიშანი განუწყვეტლივ, ისე როგორც იქცევა აქტიური ლოდინის დროს, (ე.ი. ამოწმებს აწტიურად მომლოდინეებს). ამის ნაცვლად, ამ შემთხვევის გამო პროცესორში შეიძლება შესრულდეს სრულიად სხვა პროცესი.



ნახ.8 პროცესების მიმართვა შუქნიშანზე

Pdo V ოპერაციები ოპერაციული სისტემის მიერ სრულდება მოთხოვნის საპასუხოდ, რომელიც გაცემულია რომელიმე პროცესით და შეიცავს შუქნიშნის სახელს პარამეტრის სახით. **Pdo** V ოპერაციებით სრულდება შემდეგი მოქმედებები:

P(S): if S=1

then S=S-1 /*დაიხუროს შუქნიშანი*/

else *დაიბლოკოს მიმართული პროცესი* S *-ის მიხედვით*

V(S): if პროცესების სია, რომელიც ელოდება S -ს , ცარიელი არაა,

then განიბლოკოს პროცესი, რომელიც ელოდება S -ს

else S=1 /*/*გაიხსნას შუქნიშანი*/*/

ამ დროს ვერ განისაზღვრება, რამდენიმე მომლოდინე პროცესიდან რომლის დებლოკირება მოხდება. ურთიერთგამორიცხვის სარეალიზაციოდ, მაგალითად, ორი ან მეტი პროცესის საერთო მონაცემების ერთდროულად შეცვლის შესაძლებლობის თავიდან აშორების მიზნით, იქმნება ორმაგი(ორადი)შუქნიშანი S. ამ შუქნიშნის საწყისი მნიშვნელობა დგინდება 1 -ის ტოლად. კოდის კრიტიკული სექციები (სექციები, რომლებიც შეიძლება შესრულდეს მხოლოდ ერთი პროცესორით.), ჩარჩოვდება обрамляются ოპერაციებით P(S) სექციის დასაწყისში და V(S) სექციის დასასრულში.

P(S)

კრიტიკული სიტუაცია

V(S)

კრიტიკულ სექციაში შემავალი პროცესი ასრულებს P(S) ოპერაციას და გადაყავს შუქნიშანი 0 -ში. თუკი კრიტიკულ სექციაში უკვე იმყოფება სხვა პროცესი, მაშინ შუქნიშნის მნიშვნელობა უკვე 0-ის ტოლია. მაშინ მეორე პროცესი, რომელსაც სურს კრიტიკულ სექციაში შესვლა, ბლოკირდება თავისი P

ოპერაციით, მანამ, სანამ პროცესი, რომელიც კრიტიკულ სექციაში ამჟამად იმყოფება, არ გამოვა მისგან, შეასრულებს, რა გამოსვლისას V(S) ოპერაციას.

თუკი შუქნიშნის საწყისი მნიშვნელობა უკვე 1-ის ტოლია, მაშინ ურთიერთგამორიცხვა ნამდვილად გარანტირებულია, რადგან პროცესს შეუძლია შეასრულოს P ოპერაცია მანამდე, სანამ მეორე შეასრულებს Vოპერაციას. გარდა ამისა, პროცესი, თუ აუცილებელი არ არის, არ ჩაკეტავს შესასვლელებს თავის კრიტიკულ სექციაში. შესვლა ფერხდება მხოლოდ მაშინ, როცა რომელიღაც სხვა პროცესი უკვე იმყოფება თავის საკუთარ კრიტიკულ სექციაში. პროცესი გადაცვლის შესასვლელს მხოლოდ მაშინ, როცა შუქნიშნის მნიშვნელობა იქნება 0. (ჩაკეტავს).

ამგვარად, დროის ყოველ მომენტში, პროცესი, რომელსაც სურს წვდომა განცალკევებულ ცვლადებზე ან რესურსებზე, ამას უნდა აკეთებდეს კრიტიკული სექციის მეშვეობით, რომელსაც შუქნიშანი იცავს.

მაგალითის სახით განვიხილოთ და დავუშვათ, რომ ოპერაციული სისტემა ასრულებს ორ პროცესს **A** და **B** და ერთი მათგანი (პროცესი**A**) ამატებს ელემენტს რიგში, ხოლო მეორე კი(პროცესი **B**) აკლებს ელემენტს რიგიდან. რომ რიგების მიმთითებლებმა ერთმანეთში არეულობა არ გამოიწვიონ, აუცილებელია შევზღუდოთ წვდომა რიგში ერთ ერთეულამდე, ერთდროულად. ამ გზით ელემენტების დამატება და გამოკლება კოდირებულ იქნება კრიტიკული სექციების სახით.

პროგრამა A პროცესისათვის **პროგრამა B პროცესისათვს**
პროგრამის კოდი ***პროგრამის კოდი***

P(S)
რიგში ელემენტის დამატება
V(S)
პროგრამის კოდი

P(S)
რიგში ელემენტის დამატება
V(S)
პროგრამის კოდი

ესაა მარტივი ამოხსნა, რომელშიც გამოყენებულია ორადი შუქნიშანი, ზუსტად ისევე, როგორც პროცესების დიდი რიცხვის დროს. სწორედ ესაა მთავარი ღირსება შუქნიშნის გამოყენების მიდგომისა.

ვინდოუსის შუქნიშნები

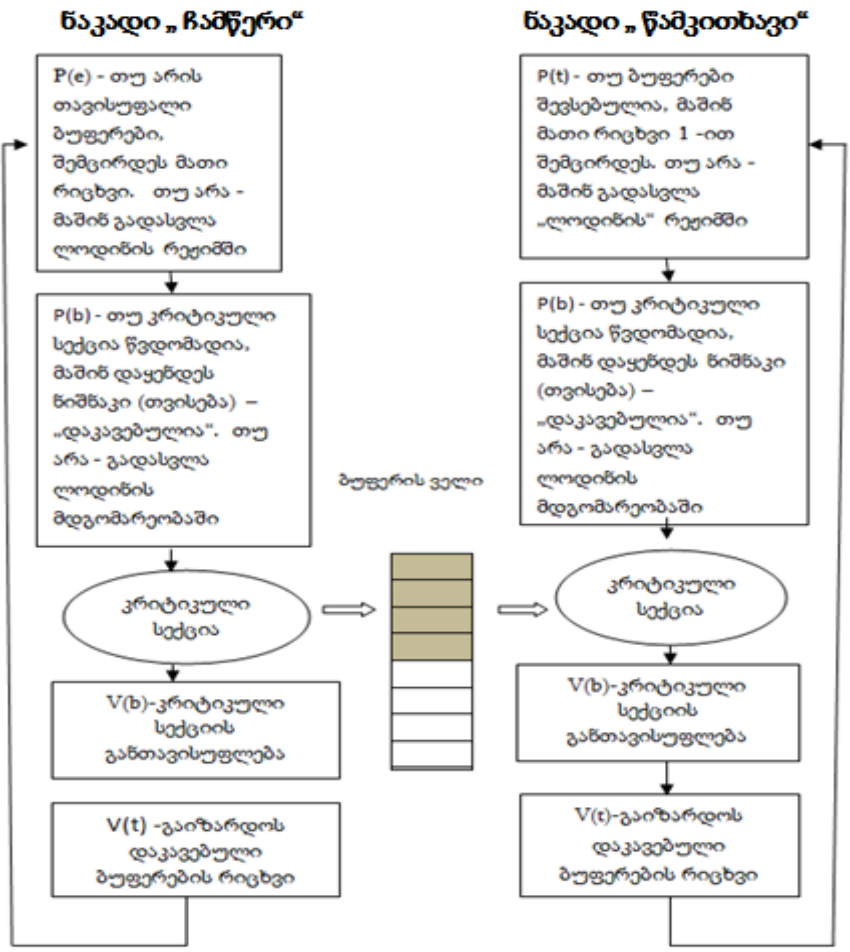
შუქნიშნები და მათზე ოპერაციები როგორც წესი, რეალიზდება ოპერაციული სისტემის ბირთვში. სადაც პროცესების მდგომარეობის მონაცვლეობის მართვა ხორციელდება .

შუქნიშნის შექმნისათვის დანართმა უნდა გამოიძახოს ფუნქცია **CreateSemaphore**, რომლის სინტაქსიც ასეთია:

HANDLE CreateSemaphore(*lpsa*: LPSECURITY_ATTRIBUTES; *cSemInitial*: LONG; *cSemMax*: LONG; *lpszSemName*: LPTSTR);

პარამეტრი *lpsa* იძლევა დაცვის დესკრიპტორს, რომელიც განსხვავდება სტანდარტული დესკრიპტორისაგან. *lpsa* პარამეტრის NULL მნიშვნელობა მიუთითებს დაცვის სტანდარტულ ატრიბუტებზე.

პარამეტრების *cSemInitial* და *cSemMax* მეშვეობით გადაეცემა მთვლელის საწყისი და მაქსიმალური მნიშვნელობა, რაც დაკავშირებულია შექმნილ შუქნიშანთან.



ნახ.9 ორადი შუქნიშნის გამოყენება

მთვლელის საწყისი მნიშვნელობა უნდა >= 0 - ზე. და არ უნდა გადააჭარბოს მთვლელის მაქსიმალურ მნიშვნელობას, რომელიც გადაიცემა *cSemMax* პარამეტრით.

პარამეტრი *lpszSemName* ანიჭებს სახელს სტრიქონის სახით.

შემდეგ ეს სახელი გამოიყენება სხვა პროცესებიდან შუქნიშნის აღმწერის მისაღებად. თუ შუქნიშანი გამოიყენება მხოლოდ ამოცანის სინქრონიზაციისათვის, რომელიც შექმნილია ერთი გამოყენების ფარგლებში, შეიძლება შეიქმნას უსახელო შუქნიშანი, პარამეტრის სახით *lpszSemName* მიეთითოს ფუნქციის **CreateSemaphore** მნიშვნელობით NULL. ხოლო როცა საჭიროა მოვახდინოთ სხვადასხვა პროცესების ამოცანების სინქრონიზაცია, შუქნიშნის სახელის განსაზღვრა აუცილებელია. შუქნიშნის წარმატებით შექმნის შემთხვევაში **CreateSemaphore** ფუნქცია აბრუნებს მის იდენტიფიკატორს. შეცდომის წარმოქმნისას ბრუნდება მნიშვნელობა NULL. ამ დროს შეცდომის კოდი შეიძლება გავიგოთ **GetLastError** ფუნქციის დახმარებით. რადგან შუქნიშნების სახელები წვდომადია სისტემაში ყველა გამოყენებისათვის, შეიძლება ისეთი სიტუაცია შეიქმნას, როდესაც გამოყენება ეცდება შექმნას შუქნიშანი უკვე გამოყენებული სახელით. ამ დროს ახალი შუქნიშანი არ იქმნება. ხოლო გამოყენება ღებულობს იდენტიფიკატორს უკვე არსებული შუქნიშანისათვის. თუ ასეთი სიტუაცია შეიქმნება, მაშინ **GetLastError** ფუნქცია, რომელიც სწრაფადვე გამოიძახება **CreateSemaphore** ფუნქციით, აბრუნებს მნიშვნელობას ERROR_ALREADY_EXISTS .

შუქნიშანის წაშლა(განადგურება)

შუქნიშანის განადგურებისათვის აუცილებელია მისი იდენტიფიკატორი გადავცეთ **CloseHandle** ფუნქციას. შევნიშნოთ, რომ პროცესის დასრულების შემდეგ მისგან შექმნილი ყველა შუქნიშანი ნადგურდება ავტომატურად.

შუქნიშანის მთვლელის მნიშვნელობის გაზრდა

აქ საჭიროა გამოყენებამ შეასრულოს ფუნქცია **ReleaseSemaphore** . ამ ფუნქციის გამოძახების სინტაქსი ასეთია:

BOOL ReleaseSemaphore(*Semaphore*: HANDLE; *cRelease*: LONG; *lplPrevious* LPLONG);

ფუნქცია **ReleaseSemaphore** ზრდის შუქნიშნის მთვლელის მნიშვნელობებს, რომლის იდენტიფიკატორი გადაეცემა მას *Semaphore* პარამეტრის მეშვეობით, მნიშვნელობაზე, რომელიც მითითებულია პარამეტრში *cRelease*. რესურსების მთვლელი **ReleaseSemaphore** ფუნქციის მეშვეობით, რომელიც მიეკუთვნება შუქნიშანს, შეიძლება გაიზარდოს ერთეულზე მეტად, ერთდროულად. *cRelease* პარამეტრი განსაზღვრავს თუ როგორი „ულუფებით“ გამონთავისუფლდეს შუქნიშანი.

შევნიშნოთ, რომ *cRelease* პარამეტრის მეშვეობით შეიძლება გადავცეთ მხოლოდ დადებითი მნიშვნელობები, მეტი ნულზე. ამასთან, თუ გაზრდის შედეგად, მთვლელის ახალმა მნიშვნელობამ გადააჭარბა მაქსიმალურ მნიშვნელობას, რომელიც მოიცემა შუქნიშნის შექმნის დროს, მაშინ ფუნქცია **ReleaseSemaphore** აბრუნებს შეცდომის ნიშანს და არ ცვლის მთვლელის მნიშვნელობას.

მთვლელის წინარე მნიშვნელობა, რომელიც იყო **ReleaseSemaphore** ფუნქციის გამოყენებამდე, ჩაიწერება LONG ტიპის ცვლადში.

თუ ფუნქცია **ReleaseSemaphore** დასრულდა წარმატებით, მაშინ ის აბრუნებს TRUE მნიშვნელობას, შეცდომის დროს აბრუნებს FALSE მნიშვნელობას. შეცდომის კოდი ამ შემთხვევაში შეიძლება განვსაზღვროთ, როგორც წესი, ფუნქციის **GetLastError** დახმარებით.

ReleaseSemaphore ფუნქცია გამოიყენება, ჩვეულებრივ, ორი ამოცანის ამოსახსნელად. ჯერ ერთი, ამ ფუნქციის მეშვეობით ამოცანები ანთავისუფლებენ რესურსს, რომელზე წვდომასაც არეგულირებს შუქნიშანი. მათ ამის გაკეთება შეუძლიათ რესურსის გამოყენების შემდეგ, ან საკუთრივ, როცა თვითონ მთავრდება. მეორეც, ეს ფუნქცია შეიძლება გამოყენებულ იქნას

მულტიამოცანური გამოყენებების ინიციალიზაციის ეტაპზე. ქმნის, რა შუქნიშანს, მთვლელის ნულის ტოლი საწყისი მნიშვნელობით, მთავარი ამოცანა ბლოკავს ამოცანას, რომელიც ასრულებს ამ შუქნიშნის ლოდინს. ე.ი. იმ ამოცანას ბლოკავს, რომელიც შუქნიშნის ლოდინის მდგომარეობაშია, და რომლითაც მთავრდება შუქნიშნის ლოდინის მდგომარეობაში ყოფნა. ინიციალიზაციის დამთავრების შემდეგ მთავარმა ამოცანამ **ReleaseSemaphore** ფუნქციის მეშვეობით შეიძლება გაზარდოს შუქნიშნის მთვლელის მნიშვნელობა მაქსიმუმამდე, რის შედეგადაც მომლოდინე ამოცანების ცნობილი რაოდენობა გააქტიურდება.

შუქნიშანის მთვლელის მნიშვნელობის შემცირება

ვინდოუსის ოპერაციული სისტემის პროგრამულ ინტერფეისში არ არის ფუნქცია რომელიც სპეციალურად დანიშნულია შუქნიშნის მთვლელის მნიშვნელობის შემცირებისათვის. ეს მთვლელი მცირდება, როცა ამოცანა იძახებს ლოდინის ფუნქციას **WaitForSingleObject**, რომლის სინტაქსიც ასეთია:

DWORD WaitForSingleObject(*Semaphore*: HANDLE;

Milliseconds: DWORD);

ფუნქცია **WaitForSingleObject** ამცირებს შუქნიშნის მთვლელის მნიშვნელობას, რომლის იდენტიფიკატორი გადაეცემა *Semaphore* პარამეტრის მნიშვნელობით. პარამეტრი *Milliseconds* მიუთითებს ლოდინის ინტერვალს მილიწამებში. თუ პარამეტრს *Milliseconds* მინიჭებული აქვს მნიშვნელობა *infinite*, მაშინ ლოდინის ინტერვალი თავისუფლდება. მიიჩნევა უსასრულობის ტოლად. თუკი ამოცანა რამდენჯერმე იძახებს ლოდინის ფუნქციას, ერთი და იმავე შუქნიშანისათვის, მისი მთვლელის შინაარსი ყოველ ჯერზე შემცირდება. ფუნქცია **WaitForSingleObject** სრულდება,

რათა დააკავოს რესურსი.

შუქნიშანის მთვლელის მიმდინარე მნიშვნელობის განსაზღვრა

შუქნიშანის მთვლელის მიმდინარე მნიშვნელობის განსაზღვრის ერთადერთი შესაძლებლობა მდგომარეობს **ReleaseSemaphore** ფუნქციის ამ მნიშვნელობის გაზრდაში. გაზრდამდე არსებული მთვლელის მნიშვნელობა ჩაიწერება ცვლადში, რომლის მისამართი გადაეცემა **ReleaseSemaphore** ფუნქციით *lplPrevious* პარამეტრის გამოყენებით. შევნიშნოთ, რომ ვინდოუსის ოპერაციულ სისტემაში არაა გათვალისწინებული საშუალებები, რომლის მეშვეობითაც შეიძლება განისაზღვროს შუქნიშნის მიმდინარე მნიშვნელობა მისი შეცვლის გარეშე. კერძოდ, შეუძლებელია ავიღოთ, ანუ მივცეთ **ReleaseSemaphore** ფუნქციას ინკრემენტ - *lplPrevious* - ის ნულოვანი მნიშვნელობა, რადგან ამ შემთხვევაში მითითებული ფუნქცია უბრალოდ, პროგრამულად აბრუნებს შეცდომის შესაბამის კოდს.

წინასწარი განმარტებები ურთიერთბლოკირებასთან დაკავშირებით

ურთიერთბლოკირებები, ანუ ჩიხური სიტუაციები შეიძლება აღიწეროს ასე: პროცესების ჯგუფი იმყოფება ჩიხურ სიტუაციაში, თუკი ჯგუფის თითოეული წევრი ელოდება მოვლენას, რომელსაც შეუძლია გამოიწვიოს მხოლოდ მეორე, რაიმე პროცესი და მხოლოდ და მხოლოდ იმავე ჯგუფიდან.

რადგან ყველა პროცესი იმყოფება ლოდინის რეჟიმში, არც ერთი მათგანი არ შეიძლება გახდეს რაიმე მოვლენის მიზეზი, რომელსაც

შეუძლია გააქტიუროს ჯგუფის ნებისმიერი სხვა პროცესი. და ყველა პროცესი აგრძელებს ლოდინს უსასრულობამდე.

ამ მოდელში ჩვენ ვუშვებთ, რომ პროცესებს აქვთ მხოლოდ ერთი ნაკადი და არ აქვს ადგილი წყვეტებს, რომლებსაც შეუძლიათ გააქტიურონ დაბლოკილი პროცესი. წყვეტის არ ყოფნის მოთხოვნა აუცილებელია, რომ თავიდან ავიცილოთ სიტუაცია, როცა ესა თუ ის დაბლოკილი პროცესი აქტიურდება, ვთქვათ, განგაშის სიგნალის მეშვეობით და შემდეგ მოასწავებს სიტუაციას, რომელიც ანთავისუფლებს ჯგუფის სხვა პროცესებს.

უმრავლეს შემთხვევაში, მოვლენები, რომლებსაც ელოდება ყოველი პროცესი, წარმოადგენენ მობრუნებას ამათუ იმ რესურსისას, რომელიც მოცემულ მომენტში დაკავებულია ჯგუფის სხვა მონაწილის მიერ. სხვა სიტყვებით, პროცესების ჯგუფის ყოველი მონაწილე, რომელიც შევა ჩიხში, -ელოდება იმ რესურსზე წვდომას, რომელიც მიეკუთვნება დაბლოკილ პროცესს. არც ერთ პროცესს არ შეუძლია მუშაობა, არცერთ პროცესს არ შეუძლია გამოანთავისუფლოს რომელიმე რესურსი და არცერთ მათგანს არ შეუძლია განახლება. პროცესების რაოდენობა და რესურსების სახეები და რაოდენობა, სახეზე არსებულები და მოთხოვნილები, აქ უკვე გამოუსადეგარია, არ არიან საჭირონი. შედეგი რჩება ისევ იგივე ყოველი სახის რესურსისათვის, აპარატურულისათვის და პროგრამულისათვის.

ურთიერთბლოკირება

კომპიუტერულ სისტემებში რესურსების დიდი რაოდენობა არსებობს, რომელთაგან თითოეული დროის კონკრეტულ მომენტში შეიძლება გამოიყენებოდეს მხოლოდ ერთ პროცესში. ასეთი მაგალითის ნიმუშად შეიძლება მოვიტანოთ პრინტერი, მაგნიტურ დისკებზე დამგროვებლები, სისტემის შინაგანი ცხრილის ელემენტები და სხვა. ორი პროცესის გაჩენა, რომლებიც ერთდროულად გადასცემენ მონაცემებს პრინტერს, იწვევს სიმბოლოების გაუგებარი ჩანაწერების ბეჭდვას. ერთდროულად ორი პროცესის არსებობა, რომლებიც ფაილური სისტემის ცხრილის ერთსა და იმავე ელემენტს იყენებს, - აუცილებლად გახდება ფაილური სისტემის მოშლის მიზეზი. ამიტომ ყველა ოპერაციული სისტემა ფლობს შესაძლებლობას, მიანიჭოს პროცესს განსაზღვრულ რესურსებზე ექსკლუზიური წვდომის შანსი(უკიდურეს შემთხვევაში დროებით).

გამოყენებითი ამოცანების შესასრულებლად პროცესს ხშირად სჭირდება უკიდურესად აუცილებლად წვდომა არა ერთ, არამედ რამდენიმე რესურსზე. დავუშვათ, რომ ყოველს, ორი პროცესიდან, სურს ჩაწეროს დასკანირებული დოკუმენტი კომპაქტდისკზე. პროცესი A შეუკვეთავს ნებართვას სკანერის გამოყენებაზე და მიიღებს მას. პროცესი B დაპროგრამებულია სხვანაირად. ამიტომ ჯერ შეუკვეთავს მოწყობილობას რომ მოხდეს კომპაქტდისკზე ჩაწერა და ისიც აგრეთვე დაკმაყოფილდება. შემდეგ A მიმართავს მოწყობილობას კომპაქტდისკზე ჩასაწერად, მაგრამ მოთხოვნა გადაიდება მანამ, სანამ ეს მოწყობილობა დაკავებულია B პროცესით. სამწუხაროდ, იმის ნაცვლად რომ გამონთავისუფლდეს მოწყობილობა კომპაქტდისკზე ჩასაწერად, B მოითხოვს სკანერს. ამ მომენტში პროცესები დაიბლოკებიან და

მუდმივად დარჩებიან ამ მდგომარეობაში. ასეთ სიტუაციას ვუწოდებთ ჩიხებს. ჩიხური სიტუაცია ანუ ურთიერთბლოკირება.

ურთიერთბლოკირება შეიძლება გაჩნდეს სხვადასხვა კომპიუტერებს შორის. ბევრ ოფისში არის ლოკალური ქსელი, რომელიც რამდენიმე კომპიუტერს მოიცავს. ხშირად პრინტერი, მაგნიტურ დისკებზე ჩამწერი, სკანერი და სხვა შეერთებულნი არიან ქსელში როგორც რესურსები ერთობლივი წვდომისათვის, ანუ წვდომადი არიან ნებისმიერი მომხმარებლისათვის ნებისმიერ მანქანაზე. თუკი ეს რესურსები მოშორებით (ან მომხმარებლის საოჯახო კომპიუტერთან) რეზერვირების შესაძლებლობას იძლევიან, მაშინ შეიძლება გაჩნდეს ზემოთ აღწერილის ანალოგიური სიტუაცია. უფრო რთული სიტუაციები შეიძლება გაჩნდეს ჩიხების მიზეზები, რომლებიც მოიცავენ სამ, ოთხ და მეტ მოწყობილობასა და მომხმარებელს.

ურთიერთბლოკირება შეიძლება ბევრ სხვა სიტუაციაშიც, იმ მოთხოვნებთან დაკავშირებით, რომლებიც წარმოიშობა შეტანა-გამოტანის მოწყობილობებისათვის. მონაცემთა ბაზების სისტემებში პროგრამა შეიძლება აღმოჩნდეს იძულებული, დაბლოკოს რამდენიმე ჩანაწერი, რათა აერიდოს კონკურენციის მდგომარეობას. თუკი A პროცესი ბლოკავს R1 ჩანაწერს, B პროცესი ბლოკავს R2 ჩანაწერს, და შემდეგ ყოველი პროცესი შეეცდება დაბლოკოს უცხო ჩანაწერიც.

დაბლოკვის პირობა

ვ. კოფმანი(Coffman)და სხვა მკვლევარები ამტკიცებენ, რომ ურთიერთბლოკირების სიტუაციის წარმოქმნისათვის უნდა შესრულდეს 4 პირობა:

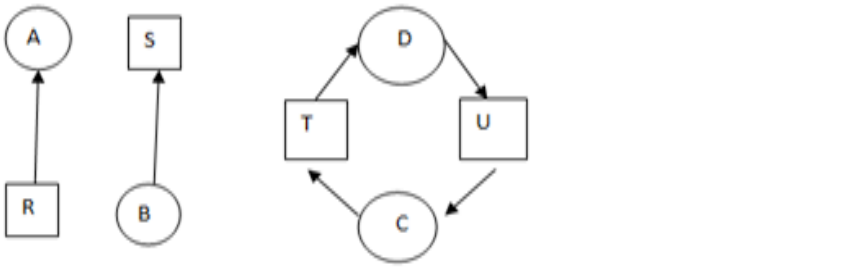
1. ურთიერთგამორიცხვის პირობა. ყოველი რესურსი მოცემულ მომენტში ან მოცემულია ზუსტად ერთი პროცესისთვის, ან ჩამოშორებული.
2. შეკავების და ლოდინის პირობა. პროცესებს, რომლებსაც მოცემულ მომენტში განკარგულებაში აქვთ(ფლობენ) ადრე მიღებული რესურსები(ს), შუძლიათ გამოითხოვონ ახალი რესურსები.
3. რესურსის იძულებით განტვირთვის გამორიცხვის(არ არსებობის) პირობა. პროცესისაგან არ შეიძლება იძულების წესით გავიტანოთ ადრე მიღებული რესურსი. პროცესმა, რომელიც ფლობს ამ რესურსს, თავად უნდა გამოანთავისუფლოს იგი.
4. ციკლურად ლოდინის პირობა. უნდა არსებობდეს ორი ან მეტი პროცესის წრიული თანმიმდევრობა რომელთაგან თითოეული ელოდება წვდომას რესურსზე, რომელიც მიმდევრობის მომდევნო წევრის დაქვემდებარებაში იმყოფება.

იმისათვის, რომ ურთიერთბლოკირება მოხდეს, უნდა შესრულდეს ეს ოთხივე პირობა. თუკი თუნდაც ერთი მათგანი აკლია, - ჩიხური სიტუაცია შეუძლებელია. უნდა აღინიშნოს, რომ ყოველი პირობა მიეკუთვნება იმის განსაზღვრის სტრატეგიას, თუ რა არის დასაშვები სისტემაში და რა - არა? შეიძლება თუ არა მოცემული რესურსი მივაკუთვნოთ ერთზე მეტ პროცესს?

შეიძლება თუ არა პროცესი ფლობდეს ერთ რესურსს და თხოულობდეს მეორეს? შეიძლება თუ არა პროცესს რესურსი წავართვათ? შეიძლება თუ არა არსებობდეს ციკლური ლოდინი? ჩვენ გავეცნობით, თუ როგორ შეიძლება მოვშალოთ ურთიერთბლოკირება ამ პირობებიდან რომელიმეს „არ ყოფნის“ ცნობის საფუძველზე.

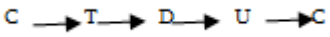
ურთიერთბლოკირების მოდელირება

B. Holt (ჰოლტმა) აჩვენა, როგორ შეიძლება ჩიხების წარმოშობის ოთხი პირობის მოდელირება მოვახდინოთ, თუკი გამოვიყენებთ მიმართულ გრაფებს. გრაფებს აქვთ ორი სახის კვანძი. პროცესები, რომლებიც ნაჩვენებია წრეებით და რესურსები, ნაჩვენები კვადრატებით. წიბო(ისარი), მიმართული რესურსის კვანძიდან პროცესის კვანძისაკენ, ვიჩვენებს, რომ რესურსი ადრე იყო მოთხოვნილი პროცესის მიერ. მიღებულ იქნა და ამ მომენტში გამოიყენება ამ პროცესის მიერ. (ნახ. 10. ა). რესურსი R ამ მომენტში მიეწოდება A პროცესს. კვადრატებით. წიბო(ისარი), მიმართული პროცესიდან რესურსისაკენ, გვიჩვენებს, რომ მოცემულ მომენტში პროცესი ბლოკირებულია და იმყოფება ლოდინის მდგომარეობაში ამ რესურსზე წვდომისათვის. (ნახ.10. ბ). B პროცესი ელოდება S რესურსს. (ნახ.10.გ). -ზე ჩვენ ვხედავთ ურთიერთბლოკირებას: C პროცესი ელოდება T რესურსს, რომელიც (ეს T) ამ მომენტში იმყოფება D- ს განკარგულებაში. D სულაც არ გეგმავს გამოანთავისუფლოს T რესურსი. რადგან ის ელოდება U რესურსს რომელსაც ამჟამად C იყენებს . ორივე პროცესი შეიძლება იყვნენ უსასრულო ლოდინის მდგომარეობაში.



ნახ.10 რესურსების განაწილების გრაფიკი

გრაფაში ციკლი ნიშნავს ბლოკირების არსებობას, ციკლურად ჩართულ პროცესებსა და რესურსებს. (იგულისხმება, რომ სისტემაში არის ყოველი სახის თითო-თითო რესურსი). ამ მაგალითში ციკლი წარმოადგენს თანმიმდევრობას:



ახლა განვიხილოთ მაგალითი იმისა, თუ როგორ შეგვიძლია გამოვიყენოთ გრაფის რესურსები. წარმოვიდგინოთ, რომ გვაქვს სამი პროცესი A,B, C. და სამი რესურსი R, S, T . რესურსების მოთხოვნებისა და დაბრუნების თანმიმდევრობა სამივე პროცესისათვის წარმოდგენილია ნახ.10.აბგ. ოპერაციულმა სისტემამ შეიძლება გაუშვას ნებისმიერი დაუბლოკავი პროცესი დროის ნებისმიერ მომენტში. ანუ, მან შეიძლება გადაწყვიტოს, რომ ჯერ გაუშვას A პროცესი, რომელიც შესრულდება მანამდე, სანამ არ დაასრულებს თავის მოქმედებას. შემდეგ გაიშვება პროცესი B დასრულებამდე და ბოლოს კი პროცესი C.

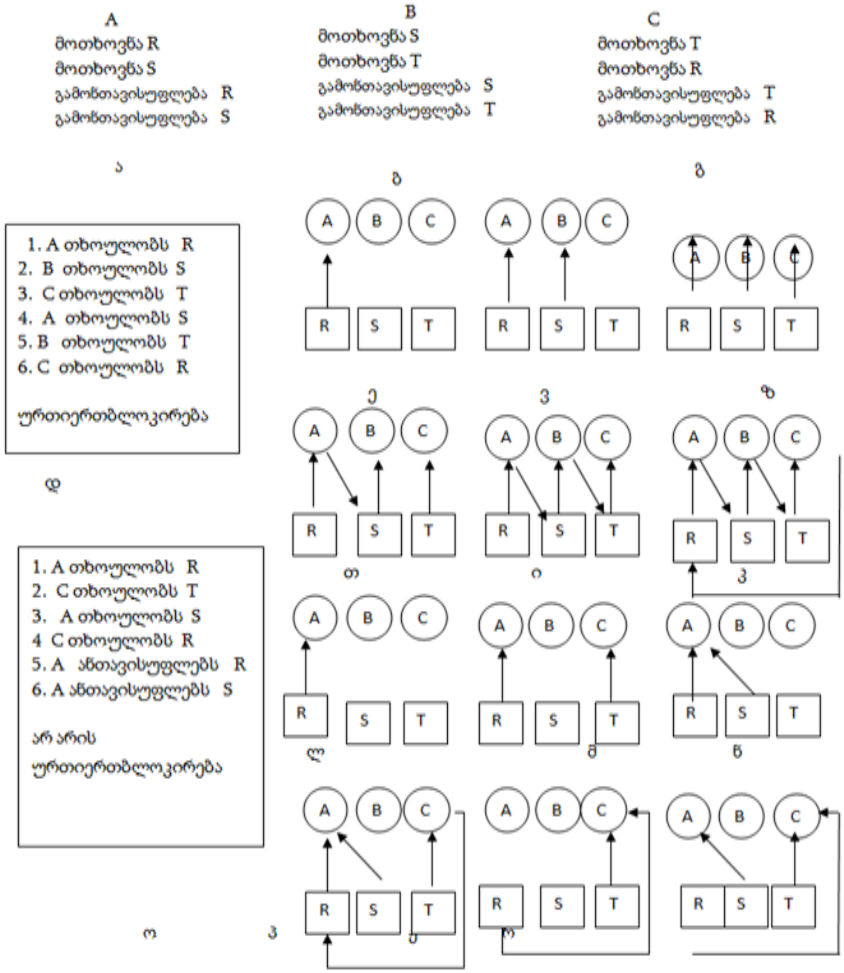
ასეთი თანმიმდევრობა არ იწვევს ურთიერთ დაბლოკვას. რადგან მასში არ წარმოიქმნება წინააღმდეგობრიობა რესურსების გამოყენების თვალსაზრისით. მაგრამ ამის დროს ასევე არ არის პარალელური სამუშაოები. რესურსების მოთხოვნებისა და დაბრუნების გარდა პროცესები ასრულებენ გამოთვლებსა და მონაცემების შეტანა-გამოტანას. როცა პროცესები მუშაობენ თანმიმდევრობით, შეუძლებელია სიტუაცია, რომლის დროსაც ერთი პროცესი იყენებს პროცესორს, იმ დროს, როცა მეორე ელოდება შეტანა-გამოტანის პროცესის დასრულებას. ამგვარად, პროცესების მკაცრად განსაზღვრული მუშაობა არ შეიძლება იყოს ოპტიმალური. მეორე მხრივ, თუკი არცერთი პროცესი არ ასრულებს მონაცემების შეტანა-გამოტანის ოპერაციას, მაშინ ალგორითმი - „უმოკლესი ამოცანა - პირველად“ - მუშაობს უკეთ, ვიდრე ციკლური. ამიტომ, ზოგიერთ გარემოებაში ყველა პროცესის თანმიმდევრული გაშვება შეიძლება საუკეთესო იყოს.

ახლა დავუშვათ, პროცესები ასრულებენ გამოთვლებსა და მონაცემების შეტანა-გამოტანას. ისე, რომ დაგეგმვის ციკლური ალგორითმი წარმოადგენს რაციონალურს. რესურსებზე მოთხოვნილებები შეიძლება მოხდეს იმ თანმიმდევრობით, როგორც ეს 11.გ -ზეა. თუკი ეს ექვსი მოთხოვნა განხორციელდება ამ თანმიმდევრობით, მაშინ, ჩვენ, შედეგად მივიღებთ 6 გრაფას, როგორც ეს ნაჩვენებია 11. დ-კ -ზე. მოთხოვნის შემდეგ 4 პროცესი A ბლოკირდება S რესურსის მოლოდინში. ნახ. 11 ზ. მომდევნო ორ ბიჯზე ასევე ბლოკირდება B და C პროცესები. საბოლოო ანგარიშში მიდის ციკლამდე და ურთიერთბლოკირებამდე. ნახ.11. კ. მაგრამ, ჩვენ ადრევე აღვნიშნეთ, ოპ. სისტემა ვალდებული არაა პროცესები გაუშვას რაღაც განსაკუთრებული წესით. კერძოდ, თუკი ცალკეული მოთხოვნის შესრულებას ჩიხამდე მივყავართ,

მაშინ ოპ. სისტემას უბრალოდ, შეუძლია პროცესი შეაჩეროს პროცესი მოთხოვნის დაუკმაყოფილებლად. (ანუ, პროცესის გეგმის შესრულების გარეშე). სანამ ეს უსაფრთხოა. 11.-ზე ოპერაციულ სისტემას შეეძლო შეეჩერებინა B პროცესი, იმის ნაცვლად, რომ არ მიეცა მისთვის S რესურსი. თუკი მას ეცოდინებოდა, რომ ჩიხი გარდაუვალი იყო. იმუშავებს რა, მხოლოდ A და C პროცესები, მაშინ ჩვენ შეგვიძლია მივიღოთ რესურსების მოთხოვნებისა და მათი მობრუნების თანმიმდევრობა, როგორც ეს 11 H ნახაზზეა . იმის ნაცვლად, რაც 11.გ-ზე იყო წარმოდგენილი. მოქმედებების ასეთი თანმიმდევრობა გამოსახულია 11 H- C - ზე და იგი არ გამოიწვევს ურთიერთბლოკირებას.

C ბიჯის შემდეგ B პროცესმა შეიძლება მიიღოს S რესურსი. იმიტომ, რომ A პროცესმა უკვე დაამთავრა თავისი სამუშაო, ხოლო C - ს აქვს თავის განკარგულებაში მისთვის საჭირო ყველა რესურსი. უფრო მეტიც, თუკი შემდეგ B პროცესი , რომელიც როცა მოითხოვს T რესურსს, იქნება დაბლოკილი, - მაინც არ ჩავარდება სისტემა ჩიხში, B პროცესი მხოლოდ და მხოლოდ იქნება C პროცესის მუშაობის დასრულების მოლოდინში.

ქვემოთ დაწვრილებით განვიხილავთ ალგორითმს რესურსების განაწილების შესახებ გადაწყვეტილების მისაღებად, რომელსაც არ მივყავართ ბლოკირებამდე. ჯერ კი მნიშვნელოვანია აღვნიშნოთ, რომ რესურსების გრაფები წარმოადგენენ ინსტრუმენტებს, რომლებიც შესაძლებლობას გვაძლევენ, დავინახოთ, გახდება თუ არა რესურსების მოთხოვნა /დაბრუნების მოცემული თანმიმდევრობა ურთიერთ დაბლოკვის მიზეზი.



ნახ. 11. ურთიერთდაბლოკვის მაგალითები და მათი თავიდან აშორების წესები

ჩვენ მხოლოდ ნაბიჯ-ნაბიჯ ვანხორციელებთ რესურსების მოთხოვნა / დაბრუნებას და ყოველი ბიჯის შემდეგ ვამოწმებთ გრაფებს ციკლების გაჩენის თვალსაზრისით. თუკი ის გაჩნდა, -

მაშინ ციკლში შევსულვართ, თუ არა, - ნიშნავს რომ არც ურთიერთბლოკირებასთან გვქონია საქმე. მიუხედავად იმისა, რომ ჩვენ რესურსების გრაფები იმ შემთხვევისათვის განვიხილეთ, როცა სისტემაში არსებობს თითოეული სახის თითო რესურსი, გრაფები აგრეთვე შეიძლება ავაგოთ იმ სიტუაციის დასამუშავებლადაც, როცა გვექნებოდა რამდენიმე ერთგვაროვანი რესურსიც. ახლა კი მოკლედ შეიძლება ჩამოვაყალიბოთ ურთიერთბლოკირებისათვის არსებული 4 სტრატეგია:

1. პრობლემის მთლიანად უგულვებელყოფა. თუკი ჩვენ იგნორირებას ვუხდენთ პრობლემას, შესაძლოა, შემდეგ მან მოახდინოს ჩვენი იგნორირება.
2. აღმოჩენა და აღდგენა. ვაცადოთ, მოხდეს ურთიერთ ბლოკირება. აღმოვაჩინოთ ის და მოვახდინოთ მასზე რაიმე რეაგირება.
3. დინამიურად ავარიდოთ თავი ჩიხურ სიტუაციებს, თუკი ზედმიწევნით გავანაწილებთ რესურსებს.
4. სტრუქტურული მტკიცების უარყოფის დახმარებით თავიდან ავიშოროთ ერთი მაინც ამ ოთხი პირობიდან, რაც აუცილებელია ურთიერთბლოკვისათვის.

ურთიერთდაბლოკვის აღმოჩენა და მოსპობა, მოშორება

მეორე ტექნიკური მიდგომა წარმოადგენს აღმოჩენასა და აღდგენას . ამ მეთოდის გამოყენების დროს სისტემა არ ცდილობს აირიდოს ჩიხურ სიტუაციაში მოხვედრა. ამის ნაცვლად ის შესაძლებლობას აძლევს განხორციელდეს ბლოკირება. ცდილობს გაარკვიოს, თუ როდის მოხდა ეს, შემდეგ კი განახორციელოს რაღაც მოქმედებები, რათა დააბრუნოს სისტემა იმ მდგომარეობაში, რაც ჩიხში მოქცევამდე ჰქონდა.

ქვემოთ ჩვენ ასეთი სიტუაციისათვის საჭირო წესები უნდა განვიხილოთ.

ა). **ურთიერთდაბლოკვის აღმოჩენა თითოეული სახის თითო რესურსის არსებობის შემთხვევაში**

ყველაზე მარტივი ვარიანტია: სისტემაში არსებობს მხოლოდ ერთი რესურსი თითოეული ტიპისა. ასეთ სისტემას შეიძლება ჰქონდეს ერთი სკანერი, კომპაქტდისკზე ჩამწერი ერთი მოწყობილობა, ერთი პლოტერი, ერთი დამაგროვებელი მაგნიტურ დისკზე, ე. ი. თითო-თითო თითოეული კლასის. ანუ, ჩვენ გამოვრიცხავთ განხილვიდან სისტემას, სადაც ერთდროულად „ორი პრინტერია ჩართული“ ამ საკითხს მოგვიანებით სხვა მეთოდითაც განვიხილავთ.

ორფაზოვანი ბლოკირება

თუმცა ზოგად შემთხვევაში არც აღმოჩენა და არც ბლოკირების არიდება არ წარმოადგენენ დიდად იმედის მომცემ მოვლენებს, მაგრამ მაინც ცალკეული გამოყენებითი ამოცანებისათვის არსებობს მრავალი სპეციალური დანიშნულების ბრწყინვალე ალგორითმი. მაგ. მონაცემთა მრავალ სისტემაში ძალიან ხშირად სრულდება ოპერაცია, რომელიც თხოულობს დაიბლოკოს რამდენიმე ჩანაწერი და შემდეგ კი განახლდეს ყველა დაბლოკილი ჩანაწერი ერთად.

როცა ერთდროულად რამდენიმე პროცესი მუშაობს, მაშინ დაბლოკვის რეალური საშიშროება ჩნდება. ხშირად გამოყენებულ მიდგომას ორფაზოვანი ბლოკირება ეწოდება. პირველ ფაზაში, ანუ პირველ ეტაპზე, პროცესი ცდილობს დაბლოკოს ყველა მოთხოვნილი ჩანაწერი ერთ ჯერზე. თუკი ოპერაცია

წარმატებულია, პროცესი გადადის მეორე ეტაპზე, ასრულებს რა, ბლოკირების განახლებას და განთავისუფლებას. არავითარი ნამდვილი სამუშაო პირველ ეტაპზე არ სრულდება.

თუკი პირველი ფაზის დროს ესა თუ ის აუცილებელი ჩანაწერი აღმოჩნდება უკვე დაბლოკილი, მაშინ პროცესი, უბრალოდ, ანთავისუფლებს ყველა თავის ბლოკირებებს და იწყებს თავის პირველ ფაზას ხელახლა. გარკვეული აზრით, ეს მეთოდი მსგავსია სქემისა, რომელშიც ყველა აუცილებელი ჩანაწერის მოთხოვნა ხდება წინასწარ, ან უკიდურეს შემთხვევაში იმის წინ, ვიდრე მოხდებოდეს რაიმე შეუქცევადი. ორფაზოვანი ბლოკირების ზოგიერთ ვერსიაში, თუკი ბლოკირება შეგვხვდა პირველი ფაზის დროს, არ ხდება რესურსების მობრუნება და პროცესის განახლება. ასეთ ვერსიებში შეიძლება გაჩნდეს ჩიხური სიტუაცია.

მაგრამ ეს სტრატეგია არ შეიძლება განვაზოგადოთ. რეალური დროის სისტემებში და პროცესების კონტროლის სისტემებში დაუშვებელია ნაწილობრივ დასრულდეს პროცესი იმის გამო, რომ რესურსი არ მიეწოდა და შემდეგ კი ხელახლა დაიწყოს ყველაფერი. ასევე დაუშვებელია გადავტვირთოთ პროცესი, თუკი მან წაიკითხა ან ჩაწერა შეტყობინება ქსელიდან, განაახლა ფაილები და კიდევ რაღაც გააკეთა, რაც არ შეიძლება უსაფრთხო იყოს განმეორების დროს.

ალგორითმი მუშაობს მხოლოდ იმ სიტუაციებში, როცა პროგრამისტი ძალიან დეტალურად ამზადებს ყველაფერს იმგვარად, რომ პროგრამა შეიძლება შევაჩეროთ პირველი ფაზის ნებისმიერ წერტილში და გავუშვათ ხელახლაც. ბევრი პროგრამა არ არის სტრუქტურირებული ასეთი სახით.

ჩიხები რესურსების გარეშე

აქამდე ჩვენი ყველა განხილვა, მსჯელობა კონცენტრირებული იყო რესურსების ურთიერთბლოკირებაზე. ერთ პროცესს სურს მიიღოს რაიმე, რომელიც გააჩნია მეორე რესურსს და უნდა ელოდოს, სანამ ის (მეორე) არ მისცემს ამ რესურსს(თავისი ნებით). უფრო მეტიც. ურთიერთბლოკირება შეიძლება მოხდეს აგრეთვე სხვა სიტუაციებშიც, იმის ჩათვლით, რომ შეიძლება რესურსები მათში საერთოდ არ მონაწილეობდნენ. მაგ. შეიძლება მოხდეს ისე, რომ ორმა პროცესმა გადაბლოკოს ერთმანეთი. თითოეული ელოდება, სანამ მეორე არ შეასრულებს რაიმე მოქმედებას. ასეთი რამე ხშირად ემართებათ შუქნიშნებს.

ვთქვათ, მაგალითად, პროცესმა უნდა შეასრულოს სისტემური გამოძახება down ორ შუქნიშანზე. როგორც წესი, ერთ matex შუქნიშანზე და კიდევ რომელიმე ერთზე. თუკი ამ ოპერაციას შევასრულებთ არასწორი თანმიმდევრიბით, მაშინ შედეგად ურთიერთდაბლოკვას მივიღებთ.

შიმშილი წარმოადგენს პრობლემას, რომელიც მჭიდროდ არის დაკავშირებული ურთიერთბლოკირებასთან. დინამიკურ სისტემებში მუდმივად სწარმოებს რესურსებზე მოთხოვნები. აუცილებელია რაღაც პოლიტიკა, გადაწტვეტილების მისაღებად იმაზე, რომ თუ როდის, ვინ და რა სახის რესურსი უნდა მიიღოს. ამ პოლიტიკამ, თუმცა კი გვეჩვენება გონივრულად, მაგრამ შეიძლება მოგვიტანოს ის, რომ რაიმე პროცესებმა არასდროს არ მიიღონ მოთხოვნილი რესურსი, მიუხედავად იმისა, რომ არ არიან დაბლოკილები.

მაგალითად, განვიხილოთ პრინტერზე მიმართვის პროცესი. წარმოვიდგინოთ, რომ სისტემა იყენებს რაიმე სახის ალგორითმს,

რომელიც გარანტიას იძლევა, რომ პრინტერზე მიმართული პროცესი არ მიგვიყვანს ბლოკირებამდე. ახლა დავუშვათ, რომ რამდენიმე პროცესს ერთდროულად სურს ისარგებლოს პრინტერით. რომელმა მათგანმა უნდა მიიღოს ეს რესურსი?

ერთი შესაძლო ალგორითმი რესურსების მიწოდებისა, აძლევს პრინტერს იმ პროცესს, რომელიც შეიცავს საბეჭდი ფაილების უმცირეს ზომას(დავუშვათ, ასეთი ინფორმაცია მიღწევადია). ასეთი მიდგომა მადლიერი კლიენტების მაქსიმალურ რიცხვს იძლევა და მომსახურების თვალსაზრისით ბრწყინვალედ გვეჩვენება. ახლა განვიხილოთ, რა მოხდება ძლიერ დატვირთულ სისტემაში, როცა ერთერთმა პროცესმა უნდა გამობეჭდოს უზარმაზარი ფაილი. ყოველ ჯერზე, როცა პრინტერი განთავისუფლდება, სისტემა ამოარჩევს პროცესს ყველაზე მოკლე ფაილის მიხედვით. თუკი სისტემაში არსებობს მუდმივი ნაკადი მოკლე-მოკლე ფაილიანი პროცესებისა, მაშინ პრინტერი არასოდეს „მოიცლის“ იმ პროცესისათვის, რომელსაც დიდი ფაილი აქვს. ეს პროცესი უბრალოდ, „კვდება შიმშილით“. (გადაიდება განუსაზღვრელი ვადით, მიუხედავად იმისა, რომ დაბლოკილი არ ყოფილა.)

„შიმშილობა“ შეიძლება ავირიდოთ თუკი გამოვიყენებთ რესურსების განაწილების სტრატეგიას პრინციპით: „პირველი მოვიდა - პირველს მოემსახურენ“. ამგვარი მიდგომით პროცესი, რომელიც ყველაზე დიდი ხანია ელოდება, ექვემდებარება მომსახურებას ზღურბლზე. ჯამში, ნებისმიერი პროცესი რაღაც მომენტში აღმოჩნდება ყველაზე უფროსი რიგში და ამგვარად, - მიიღებს საჭირო რესურსს - მომსახურებას.

სივრცეში ურთიერთბლოკირების გამოკვლევები

თუკი ოდესმე არსებობდა საგანი, რომლის გამოკვლევაზეც არ ენანებოდათ ძალისხმევა ოპ. სისტემის შექმნის ეპოქის გარიჟრაჟზე, - ეს სწორედ ჩიხური სიტუაციები იყო. ეს კი ხდებოდა შემდეგი მიზეზების გამო: ჩიხის აღმოჩენა წარმოადგენს სასიამოვნო მცირე პრობლემას გრაფთა თეორიაში, რომლითაც მათემატიკაში გაწვრთნილი სტუდენტები დაინტერესდებიან და არ მოისვენებენ. თუნდაც ამას 3-4 წელიწადი დასჭირდეს. გამოგონილია სრულიად სხვადასხვა სახის ალგორითმი, რომელთაგან ყოველი მომდევნო უფრო მეტად ეგზოტიკური და არაპრაქტიკულია წინასთან შედარებით. ბოლოს და ბოლოს გამოკვლევები ამ სფეროშიც შეწყდა. მაგრამ ხანდახან მაინც გამოჩნდება ხოლმე ამ თემისადმი მიძღვნილი ახალი სტატია, როცა ოპ. სისტემებს სურთ აღმიაჩინონ და გააუვნებელყონ ურთიერთბლოკირება, სადაც ზოგიერთი მათგანი ვერ შესრულდება, -ისინი იყენებენ ერთერთს იმ მეთოდიდან, რომელიც ამ თავში განვიხილეთ.

რესურსები

სისტემა შეიძლება შევიდეს ჩიხში, როცა პროცესებს მიეცემათ უკიდურესი უფლება ფაილებთან და მოწყობილობებთან წვდომასთან დაკავშირებით. ურთიერთბლოკირების შესახებ საუბრის მაქსიმალურად განზოგადებისათვის ჩვენ მოვიხსენიებთ ობიექტებს, რომლებსაც წვდომის უფლება ეძლევათ. რესურსი შეიძლება იყოს აპარატურული მოწყობილობა(დამგროვებელი მაგნიტურ დისკზე) ან ინფორმაციის ნაწილი(მონაცემთა ბაზაში დახურული ჩანაწერი).

კომპ-ში არის მასა სხვადასხვა რესურსისა, რომელზედაც შეიძლება მოხდეს მიმართვა. გარდა ამისა, სისტემაში შეიძლება აღმოჩნდეს რამდენიმე იდენტური ეგზემპლარი ამა თუ იმ რესურსისა, მაგ. 3 დამგროვებელი მაგნიტურ დისკზე და ა.შ. თუკი ეს ასეა, მაშინ მასზე მიმართვის საპასუხოდ, შეიძლება წარმოვადგინოთ ნებისმიერი, - ნებისმიერი შესაძლო ასლებიდან. მოკლედ რომ ვთქვათ, რესურსი - ესაა ყოველივე ის, რაც შეიძლება გამოყენებულ იქნას მხოლოდ ერთი პროცესის მიერ დროის ნებისმიერ მომენტში.

რესურსები ორი ტიპისაა. **ჩატვირთვადი და ჩაუტვირთვადი.** ჩატვირთვადი შეიძლება უპრობლემოდ იყოს აღებული მისი მფლობელი პროცესის მიერ. ასეთი რესურსის მაგალითად გამოდგება მეხსიერება. განვიხილოთ მაგ. სისტემა სამომხმარებლო მეხსიერებით 32გბ. ერთი პრინტერით და ორი პროცესორით 32-32 გბ. თითოეულს სურს რაღაც დაბეჭდოს. A პროცესი მოითხოვს და ღებულობს პრინტერს, შემდეგ დაიწყებს გამოთვლებს დაბეჭდვის მიზნით. ჯერ კიდევ გამოთვლების დასრულებამდე, ის ამოწურავს თავისი დროის კვოტას და გადმოიტვირთვება დისკზე გადმოქაჩვის სივრცეში(არეში).

ახლა ამუშავდება B პროცესი და წარუმატებლად ცდილობს მიმართვას პრინტერზე. ამ მომენტში მიიღება პოტენციური ჩიხური სიტუაცია, იმიტომ რომ A იყენებს პრინტერს, ხოლო B პროცესი იკავებს მეხსიერებას და არცერთ მათგანს არ შეუძლია გააგრძელოს მუშაობა რესურსების გარეშე, რომელიც იმყოფება მეორის განკარგულებაში. საბედნიეროდ, შეიძლება განვტვირთოთ(ავიღოთ მისგან) მეხსიერება B პროცესისგან, თუკი მას მოვათავსებთ დისკზე გადმოქაჩვის არეში და გადმოვქაჩავთ დისკიდან მეხსიერებაში A პროცესს. ახლა A - ს შეეძლება

გამოთვლების დამთავრება, შესარულებს ბეჭდვას და გაანთავისუფლებს პრინტერს. ურთიერთბლოკირება აღარ მოხდება.

სირაქლემას ალგორითმი

ეს ალგორითმი ყველაზე მარტივი მიდგომით ხასიათდება. მათემატიკოსებს ეს სრულიად მიუღებლად მიაჩნიათ და ამბობენ,ამბობენ, რომ ურთიერთბლოკირება ნებისმიერ ფასად უნდა მოვიშოროთ თავიდან. ინჟინრები ეკითხებიან: რამდენად ხშირად დგება ასეთი სიტუაცია? პრობლემა? რამდენად ხშირად ვარდება სისტემა ავარიულ მდგომარეობაში(სიტუაციაში) სხვა მიზეზების გამო და რამდენად სერიოზულია ურთიერთ ბლოკირების შედეგები. თუკი ურთიერთბლოკირება ხდება 5 წელიწადში საშუალოდ ერთხელ, ხოლო ოპერაციული სისტემის ჩაშლა, კომპილატორის შეცდომები და კომ-პის გათიშვები აპარატურის უწესრიგობის გამო კვირაში ერთხელ, მაშინ ინჟინრების უმრავლესობას არ მოესურვება პოზიცია ნებაყოფლობით დათმოს მწარმოებლურობისა და მოხერხებულობის გაგებით, იმისათვის, რომ ლიკვიდირებულ იქნას ურთიერთბლოკირების შესაძლებლობა.

ამ მიდგომებს შორის კონტრასტის გამძაფრებისათვის, აღსანიშნავია, რომ ოპერაციულ სისტემათა უმრავლესობა პოტენციურად განიცდის ურთიერთბლოკირებას, რომლებიც ზოგჯერ შეიძლება არც კი გამოვლინდებიან ხოლმე, რომ აღარაფერი ვთქვათ ჩიხიდან ავტომატურად გამოსვლაზე, - პროცესების სისტემაში პროცესების ჯამური რაოდენობა ცხრილის ჩანაწერების რაოდენობის მიხედვით განისაზღვრება. თუკი სისტემური გამოძახება fork ღებულობს უარს, იმიტომ, რომ

ცხრილი მთლიანად შევსებულია, გონივრული იქნება, რომ პროგრამა, რომელმაც გამოიწვია fork-ი დაელოდება გარკვეული დროით და გაიმეორებს მცდელობას.

ახლა დავუშვათ, რომ სისტემა UNIX- ს აქვს პროცესის 100 უჯრა. მუშაობს 10 პროგრამა. თითოეულს ესაჭიროვება შექმნას 12 პროცესი. (ან ქვეპროცესი). თითოეული პროცესის გამოსახვის შემდეგ, 10 საწყისი პროცესიდან 9 პროცესი და 90 ახალი პროცესი მთლიანად შეავსებს ცხრილს. ახლა ყოველი 10 საწყისი პროცესიდან ჩავარდება უსასრულო ციკლში, რომელიც შედგება მოსინჯვებისაგან და უარებისაგან, ანუ წარმოშობს ურთიერთბლოკირებას. ალბათობა იმისა, რომ მსგავსი რამ მოხდება, - მინიმალურია. მაგრამ, ის მაინც შეიძლება მოხდეს. შეგვიძლია თუ არა ჩვენ უარი ვთქვათ პროცესებზე და fork - ის გამოწვევაზე, რომ თავიდან ავიშოროთ ეს პრობლემა?

ღია ფაილების მაქსიმალური რაოდენობა აგრეთვე შეზღუდულია i-ური კვანძების ცხრილის ზომებით. ცხადია, როცა ცხრილი მთლიანად ივსება, იგივე პრობლემა წარმოიშობა. დისკზე ფაილების გადმოქაჩვისათვის სივრცე არის კიდევ ერთი შეზღუდვა რესურსებზე. ფაქტობრივად, ოპ. სისტემაში თითქმის ყველა ცხრილი წარმოადგენს რესურსს, რომელსაც აქვს საზღვრები. ჩვენ ვართ თუ არა ვალდებული, გავამართლოთ ისინი იმით, მხოლოდ იმიტომ, რომ შეიძლება მოხდეს სიტუაცია, როცა ჯგუფში n პროცესიდან ყოველმა შეიძლება მოითხოვოს მთელის 1/n ნაწილი და კიდევ ეცადოს დამატებით რაიმე ნაწილის მიღებას?

ოპერაციული სისტემების უმრავლესობა და მათ შორის UNIX და windows -ც იგნორირებენ ამ პრობლემას. ისინი გამოდიან იმ

დაშვებიდან, რომ მომხმარებელთა უმრავლესობას ურჩევნია საქმე ჰქონდეს დროდადრო მოხდენად ურთიერთბლოკირებასთან, ვიდრე იმ წესთან, რომლის მიხედვითაც ყველა მომხმარებელს ნება მიეცემა ჰქონდეს მხოლოდ ერთი პროცესი, ერთი ფაილი, ერთი ... და ა.შ.

თუკი შესაძლებელი იქნებოდა, ადვილად მოვიშოროთ ურთიერთბლოკირება, მაშინ აღარ გაჩნდებოდა ამდენი სალაპარაკო ამ თემაზე. სირთულე იმაში მდგომარეობს, რომ ფასები საკმაოდ მაღალია და ძირითადად, აითვლება პროცესებზე დადებული მოუხერხებელი შეზღუდვების დადების გამო.

ამგვარად, ჩვენ აღმოვჩნდით არასასიამოვნო არჩევანის წინაშე. მოხერხებულ გარემოსა და კორექტულობასა და მრავალ დისკუსიას შორის იმის შესახებ, რომ თუ რა არის უფრო მნიშვნელოვანი და საჭირო. ყველა ამ პირობებიდან ძნელია გამოიტანო ჭეშმარიტი დასკვნა.

მეხსიერების მართვა

მეხსიერება უმნიშვნელოვანესი რესურსია და ძალიან ყურადღებით მართვას საჭიროებს. მიუხედავად იმისა, რომ თანამედროვე კომპიუტერების მეხსიერება ათასობით და მილიონობით სჭარბობს გასული საუკუნის კომპიუტერების მეხსიერებას, - პროგრამული უზრუნველყოფა მაინც გაცილებით უფრო სწრაფად იზრდება, ვიდრე კომპიუტერული მეხსიერება. იდეაში, ყოველი პროგრამისტი ისურვებდა კომპიუტერში ჰქონდეს შეუზღუდავი სიდიდისა და სიჩქარის მეხსიერება, ამასთან - ენერგოდამოუკიდებელიც. ანუ მთელი შინაარსი ინახებოდეს ელექტრული დენის გათიშვის შემდეგაც და, რაკი ამ

თემას ვეხებით, ამავე დროს, რატომ არ უნდა ვისურვოთ, მეხსიერების სიიაფე? სამწუხაროდ, ტექნოლოგია ვერ უზრუნველყოფს ასეთ მეხსიერებას. ამის შედეგად, კომპ-ებში მეხსიერებას აქვს იერარქიული სტრუქტურა. მათი გარკვეული ნაწილი(არც ისე დიდი) წარმოადგენს ძალზე სწრაფ, ძვირ და ენერგოდამოკიდებულს. ქეშ-მეხსიერებას. გარდა ამისა, კომპ-ები ფლობენ ათობით მეგაბაიტ საშუალო სისწრაფეს და საშუალო ფასს. აგრეთვე ენერგოდამოკიდებულ RAM(озу) ოპერატიულ მეხსიერებას და ათობით და ათასობით გიგაბაიტ ნელ, იაფ, ენერგოდამოუკიდებელ სივრცეს მყარ დისკზე, ოპ. სისტემის ერთ ერთ მთავარ ამოცანას სწორედ რომ ყველა ამგვარი მეხსიერების გამოყენების კოორდინაცია წარმოადგენს.

ოპერაციული სისტემის ნაწილს, რომელიც პასუხისმგებელია მეხსიერების მართვაზე, - ეწოდება მეხსიერების მართვის მოდული, ანუ მეხსიერების მენეჯერი. ის თვალყურს ადევნებს „თუ მეხსიერების რა ნაწილი გამოიყენება მოცემულ მომენტში. - რამდენია თავისუფალი“. აუცილებლობის შემთხვევაში გამოუყოფს მეხსიერებას პროცესებს და მათი დასრულების შემდეგ ანთავისუფლებს რესურსებს. მეხსიერების მენეჯერი აგრეთვე მართავს მონაცემთა გაცვლას ოპერატიულ მეხსიერებასა და დისკს შორის. (swapping). ესაა მონაცემთა გადაქაჩვა თუკი მეხსიერება ძალზე მცირეა იმისათვის, რომ მოითავსოს მთელი პროცესი. (დიდი ინფორმაციის გადმოქაჩვის ორგანიზებას ახდენს გამონთავისუფლებული რესურსებისა და სახეზე დარჩენილი მეხსიერების გამოყენების გზით).

როგორც ყველგან, კომპიუტერულ სამყაროშიც ისტორიას აქვს განმეორებითობის ტენდენცია. და მიუხედავად იმისა, რომ მეხსიერების მართვის უმარტივესი სქემები უკვე აღარც კი

გამოიყენება თანამედროვე კომპ-ებში, - ისინი ჯერ კიდევ მოქმედებენ ჯიბის კომპ-ებში, ჩაშენებულ სისტემებსა და სმარტ-ქარდებში.

მეხსიერების მართვის ძირითადი სისტემები

მეხსიერების მართვის სისტემები შეიძლება ორ კლასად დაიყოს: ოპერაციულ სისტემასა და დისკებს შორის გადატანითი პროცესები მათი შესრულების დროს(ანუ გადაქაჩვის განხორციელება პროცესისა მთლიანად(swapping). და ფურცელ-ფურცელ გადაქაჩვის გამოყენება(paging). (მესამე ვარიანტად ამ მოქმედების რაიმე კომბინაციით შესრულება შეიძლება ვივარაუდოთ).

მეორე ვარიანტი უფრო მარტივია. უნდა აღინიშნოს, რომ ჩვეულებრივი და ფურცვლოვანი(ფურცლებრივი) ვარიანტები საკმარისად მნიშვნელოვანი ხარისხით წარმოადგენენ ხელოვნურ პროცესებს, რომლებიც გამოწვეულია ოპერატიული მეხსიერების არასაკმარისობით, რათა ერთდროულად მოხდეს ყველა პროგრამის შენახვა. თუკი ოდესმე ოპერაციული სისტემა იმდენად გაიზრდება ზომებში, რომ ეს ზემოთ აღნიშნულს შესაძლოა ეყოს, - მაშინ არგუმენტები მეხსიერების მართვის ამა თუ იმ სქემის სასარგებლოდ - მოძველებულად ჩაითვლება.

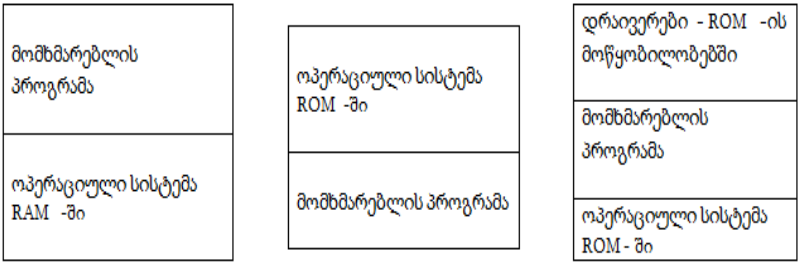
მეორე მხრივ, უკვე აღნიშნული იყო, რომ პროგრამული უზრუნველყოფა უფრო სწრაფად იზრდება, ვიდრე კომპ-ის მეხსიერება, ამიტომ, სრულიად შესაძლებელია, რომ მეხსიერების რაციონალურად და ეფექტურად მართვისათვის მოთხოვნილება ყოველთვის იარსებებს. გასული საუკუნის 80-იან წლებში უნივერსიტეტები იყენებდნენ დროის დაყოფვის სისტემას ათობით მომხმარებლისათვის, რადგან მაშინ მეხსიერების

მოცულობა 4 მეგაბაიტი იყო. !! ახლა კი ეს პრობლემა მოხსნილია, თუმცა „გარეგნულად“ მაინც ჩანს, რომ კომპიუტერების მეხსიერება მაინც არ ჰყოფნით. ასე, რომ სრულიად მაღალალბათურია, რომ კომპ-ების ამ ნაწილის მართვის ხარისხი აქტუალური იქნება უკიდურეს შემთხვევაში რამდენიმე ათეული წელი კიდევ.

ერთამოცანიანი სისტემა დისკზე გადაჭრის გარეშე

მეხსიერების მართვის სქემებიდან შესაძლო ყველაზე მარტივი იმაში მდგომარეობს, რომ ყოველ კონკრეტულ მომენტში მუშაობს მხოლოდ ერთი პროგრამა და ამ დროს მეხსიერება ნაწილდება პროგრამასა და ოპერაციულ სისტემას შორის. (ნახ.12) ნაჩვენებია ასეთი სქემის სამი ვარიანტი. ოპ. სისტემა შეიძლება მდებარეობდეს (იმყოფებოდეს) მეხსიერების ქვედა ნაწილში, RAM (Random Access Memory თავისუფალი წვდომის მეხსიერება) ოპერატიულ დამმახსოვრებელ მოწყობილობაში ნახ. 12. ა. ანდა ოპ. სისტემა შეიძლება მდებარეობდეს (იმყოფებოდეს) მეხსიერების ყველაზე მაღლა ნაწილში, მუდმივ დამმახსოვრებელ მოწყობილობაში, ROM (Read Only memory მეხსიერება მხოლოდ წასაკითხავად) . ნახ. 12. ბ. მესამე წესი იქნება: მოწყობილობათა დრაივერები იმყოფებოდნენ ზედა ნაწილში, დანარჩენი ნაწილი სისტემისა RaM -ში, რომელიც ქვემოთ მდებარეობს. ნახ. 19.2. პირველი მოდელი ადრე გამოიყენებოდა მინი კომპ-ებში და დღეისათვის პრაქტიკულად აღარ გამოიყენება. მე-2 - ჯიბის კომპიუტერებსა და ჩაშენებულ სისტემებში. მე-3 - ადრეულ პერსონალურ კომპ-ებში, MS DOS - თაობაში. ამასთან, სისტემურ ნაწილს ПЗУ-ში ეწოდებოდა ***BIOS*** საბაზო სისტემა შეტანა-გამოტანისათვის.

როცა სისტემა ასეა ორგანიზებული, დროის ყოველ კონკრეტულ მომენტში შეიძლება მუშაობდეს მხოლოდ ერთი პროცესი. როგორც კი მომხმარებელი იღებს ბრძანებას, - ოპერაციული სისტემა აკოპირებს მოთხოვნილ პროგრამას დისკიდან მეხსიერებაში, ასრულებს მას და პროცესის დასრულების შემდეგ ეკრანზე გამოდის მოწვევის სიმბოლო და ელოდება ახალ ბრძანებას. მიიღებს რა, ბრძანებას, - ის ჩატვირთავს ახალ პროგრამას მეხსიერებაში და ამას დააწერს თავზე ზემოთ, წინა არსებულს. შესაძლებელია სხვა ვარიანტების არსებობაც.



ნახ.12 ერთამოცანიანი სისტემის სქემა

მრავალამოცანიანობა ფიქსირებული განყოფილებებით (ნაწილებით)

თანამედროვე სისტემების უმრავლესობა იყენებს რამდენიმე პროცესის ერთდროულად გაშვების პრინციპს. რამდენიმე პროცესის არსებობა, რომლებიც მუშაობენ დროის მოცემულ მომენტში, ნიშნავს, რომ როცა ერთი პროცესი შეჩერებულია შეტანა-გამოტანის ოპერაციის დასრულების მოლოდინში, მაშინ მეორეს შეუძლია გამოიყენოს ცენტრალური პროცესორი. ასე, რომ მრავალამოცანიანობა ზრდის პროცესორის დატვირთვას. ქსელურ სერვერებს ყოველთვის აქვთ შესაძლებლობა, ერთდროულად

იმუშაონ რამდენიმე პროცესზე(სხვადასხვა კლიენტებთან). მაგრამ კლიენტების კომპ-ების უმრავლესობაც კი (ანუ მაგიდის) ჩვენს პირობებში უკვე ფლობენ ამ შესაძლებლობებს.

მრავალამოცანიანობის მისაღწევად ყველაზე მარტივი წესი წარმოადგენს დახარისხებას, მეხსიერების დაყოფას n ნაწილად, თუნდაც არათანაბარ ნაწილებად. ეს დაყოფა ხელითაც შეიძლება პროგრამული სისტემის გაშვების დროს. როცა ამოცანა(დავალება) ხვდება მეხსიერებაში, ის შეიძლება მოთავსდეს შესავალ რიგში, უმცირეს განყოფილებაში, და ისიც კი შეიძლება საკმაოდ დიდი გამოდგეს, რომ მოითავსოს ეს ამოცანა. რადგან ამ სქემაში ნაწილების (განყოფილებების) ზომები უცვლელია(ნახ.20.ა), მთელი სივრცე განყოფილებაში, რომელსაც არ იყენებს მიმდინარე მუშა პროცესი, - იკარგება. აქ ნაჩვენებია, როგორ გამოიყურება სისტემა ფიქსირებული განყოფილებებით და საწყისი დავალებების ცალკეული რიგებით.

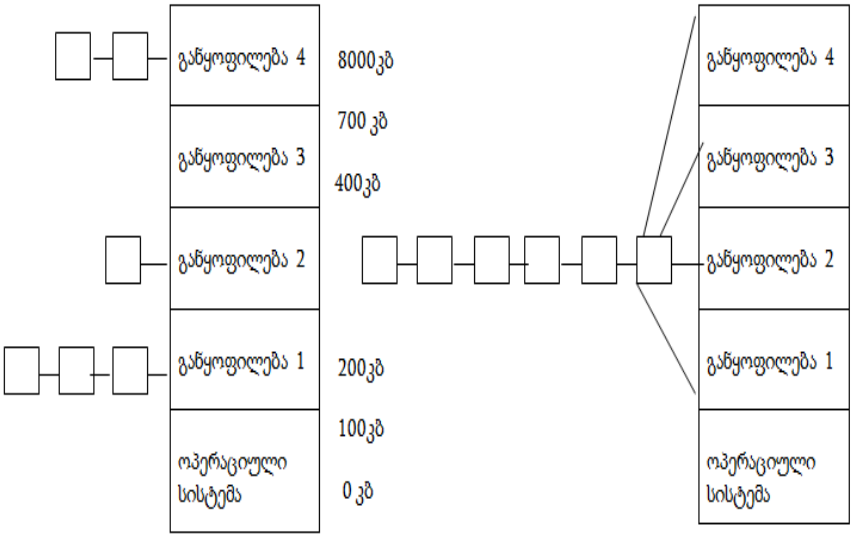
შემავალი სამუშაოების დახარისხების ნაკლი ცალკეული რიგების მიხედვით თვალსაჩინო ხდება, როცა დიდ განყოფილებას არ აქვს რიგები. იმ დროს, როცა მცირეს მიეწერა საკმაოდ ბევრი ამოცანა(1და 4 განყოფილებები 13.ა - ზე).

საშუალო სიდიდის ამოცანები შეიძლება დაელოდნენ თავიანთ რიგებს, რომ მოხვდნენ მეხსიერებში. და ეს ყველაფერი იმის მიუხედავად, არის თუ არა თავისუფალი მეხსიერების ძირითადი ნაწილი. ალტერნატიული სქემა მდგომარეობს ერთიანი საერთო რიგის შემუშავებაში ყველა განყოფილებისათვის, როგორც ეს ნახ.13.ბ არის ნაჩვენები. როგორც კი განყოფილება განთავისუფლდება, მაშინვე, ამოცანა, რომელიც ყველაზე ახლოს არის რიგის დასაწყისთან, და გადის შესასრულებლად ამ

განყოფილ;ებაში, - საჭიროა და შეიძლება კიდეც მისი გაშვება და მისი დამუშავების დაწყება. რადგანაც არაა სასურველი დავხარჯოთ(დავაკავოთ) დიდი განყოფილებები მცირე ამოცანებისთვის, - ამიტომ არსებობს სხვა სტრატეგიაც. ის იმაში მდგომარეობს, რომ ყოველთვის, განყოფილების განთავისუფლების შემდეგ, ხდება რიგში ყველაზე დიდი ამოცანის მოძიება, რომელიც ამოცანათა ამ განყოფილებაშია, და სწორედ ეს ამოცანა შეირჩევა დასამუშავებლად. შევნიშნოთ, რომ ეს უკანასკნელი ალგორითმი ახდენს მცირე ამოცანების დისკრიმინაციას, რომლებისთვისაც მიზანშეუწონელია მთლიანი დიდი განყოფილების გამოყოფა, თუმცა, როგორც წესი, უკიდეურესად აუცილებელია და სასურველი, წარედგინოს უმცირეს ამოცანებსაც(ხშირად ინტერაქტიულებს), ცუდი კი არა, - უკეთესი მომსახურეობა.

ამ მდგომარეობიდან გამოსავალი შეიძლება, თუკი შვქმნით ერთს მაინც მეხსიერების მცირე განყოფილებას, რომელიც საშუალებას მოგვცემს, შევასრულოთ მცირე დავალებები, ისე, რომ ხანგრძლივად აღარ ველოდოთ დიდი განყოფილების გამონთავისუფლებას.

სხვა მიდგომის შემთხვევაში შემდეგი წესი ყალიბდება: ამოცანა, რომელიც მოიპოვებს დამუშავების „უფლებას“, შეიძლება გაშვებულ იქნას არაუმეტეს K-ჯერ. ყოველ ჯერზე, როდესაც მას არ „გადაახტებიან“, - მთვლელს ემატება ერთიანი.



ნახ.13. წარმოდგენილია ფიქსირებული მეხსიერების განყოფილებები ცალკეული შესასვლელი რიგებით ყოველი განყოფილებისათვის(ა) და ფიქსირებული მეხსიერების განყოფილებები შესასვლელში ერთი რიგით(ბ)

როცა მთვლელი მიიღებს მნიშვნელობას = K უკვე ამოცანა იგნორირდება, აღარ შეიძლება. მსგავსი სქემაა, როცა დილით, ადმინისტრატორი იძლევა ფიქსირებულ განყოფილებებს და ამის შემდეგ უკვე ისინი აღარ იცვლებიან. - მრავალი წლის განმავლობაში გამოიყენებოდა OS/360 და IBM - ში. ამას ეწოდება MFT – Multiprogramming with a Fixed numbering of Tasks - მულტიპროგრამირება ამოცანათა ფიქსირებული რიცხვით. ის

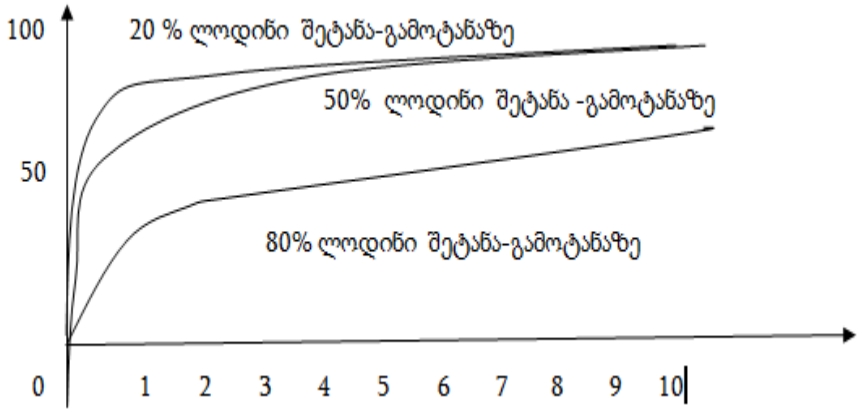
მარტივია გასაგებად და ასევე ადვილია შესასრულებლად: შესასვლელი ამოცანა დგას რიგში მანამ, სანამ წვდომადი არ გახდება შესაბამისი განყოფილება. შემდეგ ის იტვირთება მეხსიერების ამ განყოფილებაში და იქ მუშაობს პროცესის დასასრულამდე. დღეისათვის ოპერაციული სისტემები უჭერენ მხარს ამ მოდელს.

მრავალამოცანიანობის მოდელირება

მრავალამოცანიანობის გამოყენების დროს იზრდება ცენტრალური პროცესორის(ცპ) დატვირთვის ეფექტურობა. უხეშად რომ ვთქვათ, თუკი ერთი საშუალო პროცესი აქტიურ სამუშაოს ასრულებს იმ დროის 20 % - ში, რომლის განმავლობაშიც ისა იმყოფება მეხსიერებაში, - მაშინ მეხსიერებაში ერთდროულად 5 პროცესის ყოფნის შემთხვევაში, ცპ - მ უნდა დაიკავოს მთელი დრო 5*20 = 100 . ეს სქემა ძალზე ოპტიმისტურია რეალური სიტუაციისაგან განსხვავებით. რადგანაც ის უშვებს, რომ ყველა 5 პროცესი არასდროს არ ელოდება ოპერაციის დასრულებას და შეტანა-გამოტანას ერთდროულად.

უფრო სრულყოფილი მოდელი განიხილავს ცენტრალური პროცესორის ექსპლოატაციას ალბათობის თეორიის თვალსაზრისით. დავუშვათ, რომ პროცესი ატარებს თავისი დროის P ნაწილს შეტანა-გამოტანის ოპერაციის დასრულების მოლოდინში. თუკი მეხსიერებაში იმყოფებოდა ერთდროულად n პროცესი, მაშინ ალბათობა იმისა, რომ ყველა n პროცესი ელოდება შეტანა-გამოტანას, (ამ შემთხვევაში ც.პ. უმოქმედო იქნება.) ტოლია P^n. მაშინ ც.პ - ს დატვირთვის(გამოყენების) ხარისხი გამოითვლება ფორმულით : ცპგხ = 1- P^n .

ნახც.14-ზე ნაჩვენებია დამოკიდებულება ც.პ-ს გამოყენებების ხარისხისა n რიცხვზე(n - არის მრავალ ამოცანიანობის ე.წ. ხარისხი.) ანუ, ცპდხ დამოკიდებულება მეხსიერებაში პროცესების რაოდენობაზე.



ნახ.14. ცენტრალური პროცესორის დატვირთვის გრაფიკი

ნახაზიდან ჩანს, რომ თუკი პროცესები შეტანა-გამოტანის პროცესის დასრულებისათვის ლოდინში გადიან თავისი დროის 80 % - ს, მაშინ იმისათვის, რომ 10 % -ზე ნაკლები იყოს დროის დანაკარგი, (80:10 = 8 ≅ 10). მეხსიერებაში ერთდროულად უნდა იმყოფებოდნენ სულ მცირე 10 პროცესი. როცა მომხმარებელი ბეჭდავს რაღაცას ტერმინალზე, მაშინ ლოდინის ინტერაქტიული პროცესი, იქნება პროცესი, რომელიც იმყოფება შეტანა-გამოტანის ლოდინის მდგომარეობაში. ნათელი უნდა იყოს, რომ შეტანა-გამოტანის ლოდინის დრო ტოლია ან მეტია 80 %-ზე და ეს არ აღიქმება არაბუნებრივად!

პროცესების პაკეტური დამუშავების სისტემებშიც კი რომლებიც შეტანა-გამოტანას ძირითადად დისკიდან აწარმოებდნენ , - ხშირად გვაქვს ასეთივე და/ან მეტი პროცენტი.

უნდა აღინიშნოს, რომ ზემოთ აღწერილი ალბათური მოდელი წარმოადგენს საკმარისად უხეშ მიახლოებას... ის არა ცხადად გულისხმობს, რომ ყველა n პროცესი დამოუკიდებელია. ანუ დასაშვებია შემდეგი სიტუაცია: მეხსიერებაში იმყოფება 5 პროცესი. აქედან 3 მუშაობს, 2 ელოდება. მაგრამ, როცა სისტემაში არის მხოლოდ ერთი ც.პ. მას არ შეუძლია ერთდროულად დაამუშავოს 3 პროცესი. ამიტომ უკვე სამუშაოდ გამზადებული პროცესი ვალდებულია დაელოდოს პროცესორის გამონთავისუფლებას. ამრიგად, რეალურად, პროცესები არ არიან დამოუკიდებლები. უფრო დახვეწილი მოდელი შეიძლება ავაგოთ ლოდინის რიგების ორგანიზაციის თეორიის საფუძველზე. მაგრამ მთავარი იდეა, - რომელზედაც ჩვენ ყურადღებას ვამახვილებთ, - მრავალ ამოცანიანობაა - საშუალებას აძლევს პროცესებს, რათა გამოიყენონ ც.პ. მაშინ, როცა სხვა გარემოებებში ის უმოქმედო იქნებოდა. ცხადია, ძალაში რჩება, თუკი 14 - ზე წირები ცოტათიც კი რომ შეიცვალონ. თუმცა 14-ე მოდელი ძალიან მარტივია, მაინც მიუხედავად ამისა, ის მაინც საშუალებას იძლევა მწარმოებლობასთან მიმართებაში ც.პ - ს განსაზღვრული, თუნდაც მიახლოვებითი პროგნოზი გავაკეთოთ.

მაგ. დავუშვათ, რომ კომპ-ს აქვს 32 მბ. მეხსიერება. 16 მბ. განკუთვნილია ოპ. სისტემისათვის და 4 მბ მომხმარებლების თითოეული პროგრამისათვის. ასეთად მოცემულ განყოფილებებში ერთდროულად შეიძლება ჩაიტვირთოს 4 სამომხმარებლო პროგრამა მეხსიერებაში.. შეტანა-გამოტანაზე

ლოდინის დროის 80% შემთხვევაში, საშუალოდ, ჩვენ მივიღებთ პროცესორის დატვირთვას(ოპერაციული სისტემის დანახარჯების იგნორირებით) 10,81 - ის ტოლს, ანუ 60%. კიდევ 16 მბ. დამატებით საშუალებას მისცემს სისტემას აამაღლოს მრავალამოცანიანობის ხარისხი 4-დან 8 -მდე, და ამგვარად, აამაღლოს პროცესორის დატვირთვა 83 % -მდე. სხვა სიტყვებით, დამატებით 16 მბ. ზრდის მწარმოებლობას 38 % - ით. (60 + 38 ≅ 100). თუკი 16 მბ- ს შეუძლია პროცესორის დატვირთვა აამაღლოს 83 – 93%, მაშინ, ამგვარად, თუ მწარმოებლობას გავზრდით მხოლოდ 12% -ით, მაშინ ამ მოდელის მეშვეობით კომპ-ის მფლობელს შეუძლია გადაწყვიტოს, რომ ოპერატიული მეხსიერების პირველი 16 მბ - ეს კარგად დაბანდებული კაპიტალია. ხოლო მეორე ნახევარი - არა.

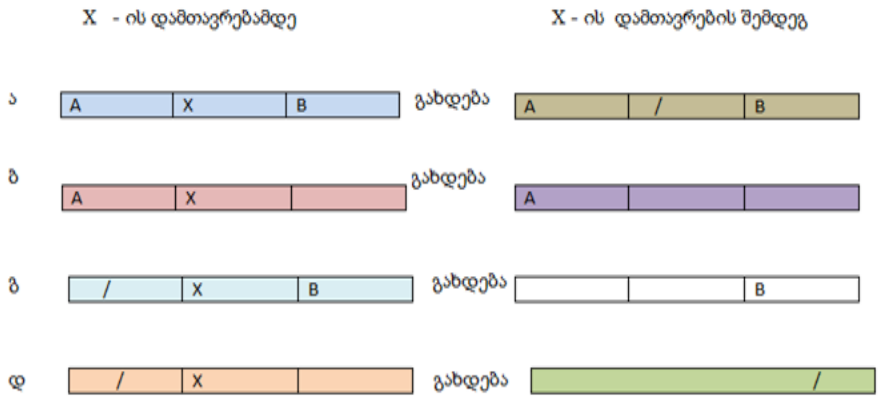
მეხსიერების მართვა დაკავშირებული სიებით

მეხსიერების მდგომარეობის გადახედვის მეორე წესი საშუალებას გვაძლევს მხარი დავუჭიროთ დაკავშირებულ სიებს და მეხსიერების თავისუფალ ფრაგმენტებს, სადაც სეგმენტებად გვევლინებიან ან პროცესები ან მონაკვეთი ორ პროცესს შორის.

სიაში ყოველი ჩანაწერი მიუთითებს არის თუ არა მეხსიერების არე თავისუფალი ანდა დაკავებული პროცესისაგან. H(hole - სიცარიელე) ; P (process - პროცესი). მისამართი, საიდანაც იწყება ეს არე. მისი სიგრძე; შეიცავს მაჩვენებელს, მიმთითებელს შემდეგ ჩანაწერზე.

მაგალითისათვის, სია შეიძლება დახარისხდეს მისამართების მიხედვით. ასეთ დახარისხებას აქვს ის უპირატესობა, რომ, როცა პროცესი მთავრდება ანდა გადაიქაჩება დისკზე, მაშინ სიის შეცვლა არ წარმოადგენს რთულ ოპერაციას. დასრულებად

პროცესს ჰყავს, ჩვეულებრივ ორი მეზობელი. (იმ შემთხვევის გარდა, როცა ის იმყოფება ყველაზე ზევით ან ყველაზე ძირს, მეხსიერების ფსკერზე.) მეზობლად შეიძლება მოგვევლინოს პროცესები ან თავისუფალი ფრაგმენტები, რაც გვაძლევს 4 კომბინაციას, წარმოდგენილს ნახ. 15. ა- ზე სიის კორექტირება თხოულობს P - შეცვლას H -ით.(ე.ი. მეხსიერების გამონთავისუფლებას). 15 ბ,გ ორი ჩანაწერი ერთიანდება ერთში, ხოლო სია მოკლდება 1 ერთეულით. 15.დ -ზე ერთიანდება სამი ჩანაწერი და სია გრძელდება 2 პუნქტით. რადგან პროცესების ცხრილის უჯრედმა დასრულებადი პროცესისათვის დაუყოვნებლივ უნდა მიუთითოს სიაში ჩაწერაზე ამ პროცესისათვის, ამიტომ, შესაძლებელია, უფრო მოსახერხებელი იყოს გვქონდეს სია არა ერთი, არამედ ორი კავშირით. ასეთი სრტუქტურა ამარტივებს წინა ჩანაწერის ძიების პროცესს შეერთების შესაძლებლობის შეფასებისას.



ნახ.15 პროცესის დასრულებისათვის მეზობლების ოთხი კომბინაცია

თუკი პროცესები და თავისუფალი მონაკვეთები ინახება სიაში, რომელიც დახარისხებულია მისამართებით, მაშინ არსებობს სხვადასხვა ალგორითმი ხელახლა შექმნილი პროცესისათვის მეხსიერების წარსადგენად, (ან არსებული პროცესისათვის, რომლებიც გადმოიქაჩა დისკიდან). დავუშვათ, მეხსიერების მენეჯერმა „იცის“, რამდენი მეხსიერება უნდა წარადგინოს, რა მოცულობის. უმარტივესი ალგორითმი არის **პირველი შესაბამისი** მონაკვეთის შერჩევა. მეხსიერების მენეჯერი შეამოწმებს არეების სიებს მანამდე, სანამ არ იპოვნის ყველაზე დიდ თავისუფალ მონაკვეთს. შემდეგ ეს მონაკვეთი იყოფა ორ ნაწილად. ერთი მიეწოდება პროცესორს, მეორე კი რჩება გამოუყენებელი. ასე ხდება ყოველთვის, გარდა სტატისტიკურად ნორმალური შემთხვევისა, როდესაც ზუსტად შეესაბამებიან ერთმანეთს თავისუფალი მონაკვეთი და პროცესი. ესაა სწრაფი ალგორითმი, რადგან ძიება შემცირებულია იმდენად, რამდენადაც კი შესაძლებელია.

ალგორითმი **„შემდეგი შესაფერისი მონაკვეთი“** მოქმედებს „პირველი შესაფერისისაგან“ მცირეოდენი განსხვავებით. ის მუშაობს ისევე როგორც პირველი ალგორითმი, მაგრამ ყველა შემთხვევაში, როცა შესაბამის თავისუფალ ფრაგმენტს პოულობს, ის იმახსოვრებს მის მისამართს. და როცა ალგორითმი მომდევნოჯერ გამოიძახება ძიებისათვის, ის სტარტს იღებს იმ ადგილიდან, სადაც შეჩერდა წინა შემთხვევაში. იმის ნაცვლად, რომ ძიებას შეუდგეს ხოლმე ყოველთვის ისევ სიის დასაწყისიდანვე, როგორც ამას ახდენდა ადრე განხილული „პირველი შესაფერისი ალგორითმი“. **ბეისიმ** ამ ალგორითმის მუშაობა შეისწავლა და აჩვენა, რომ „შემდეგი შესაფერისის“ სქემის მწარმოებლობა უფრო დაბალია „პირველი შესაფერისისთან“ შედარებით.

ვირტუალური მეხსიერება

უკვე დიდი ხანია მას შემდეგ, რაც თავი იჩინა პრობლემამ პროგრამის განთავსების თაობაზე, რომელიც საკმაოდ დიდი მოცულობისა აღმოჩნდა და ამიტომ არ ეტეოდა ხელმისაწვდომ ფიზიკურ მეხსიერებაში. ჩვეულებრივ მიიღებოდა გადაწყვეტილება პროგრამის შედგენისა ნაწილებად დაყოფის პრინციპით. რომელთაც უწოდებენ „ოვერლეიმებს“. (Overlays). ოვერლეი-O, პირველად ჩვეულებრივ გაიშვება. თავისი შესრულების დამთავრების შემდეგ ის იძახებს მომდევნო ოვერლეის. ზოგიერთი ოვერლეიების სისტემა ძალზე რთული იყო. რომლებიც რამდენიმე ოვერლეის მეხსიერებაში ყოფნის შესაძლებლობას იძლეოდა. ისინი ინახებოდა დისკზე და აუცილებლობის შემთხვევაში დინამიკურად მეხსიერებასა და დისკებს შორის გადაადგილდებოდნენ ოპერაციული სისტემის საშუალებების მეშვეობით.

იმის მიუხედავად, რომ ფაქტობრივი მუშაობა ოვერლეის დისკიდან ჩატვირთვისა და დისკზე გადმოტვირთვისა, სრულდებოდა სისტემის მეშვეობით, მაინც პროგრამის დაყოფა შემადგენელ ნაწილებად პროგრამისტს უხდებოდა.

დიდი პროგრამების დაშლა მცირე მოდულებად გვართმევდა ბევრ დროს და არც ისე საინტერესო საქმიანობას წარმოადგენდა, ამასთან, ასეთი სიტუაცია გრძელდებოდა ცოტა ხანს, რადგან მალე მოიგონეს წესი, რომ ეს სამუშაო შეესრულებინა კომპიუტერს.

შემუშავებული მეთოდი ცნობილია „ვირტუალური მეხსიერების“ სახელწოდებით. ძირითადი იდეა მდგომარეობს იმაში, რომ პროგრამების, მონაცემებისა და შეთავსების(სტიკ)

გაერთიანებული ზომა შეიძლება მეტი აღმოჩნდეს ფიზიკური მეხსიერების დასაშვებ მოცულობაზე. ოპ. სისტემა პროგრამის ნაწილს, რომელიც გამოყენებაშია მოცემული მომენტისათვის, ინახავს ოპერატიულ მეხსიერებაში, ხოლო დანარჩენს კი დისკზე. მაგ. 16გბ. პროგრამამ შეიძლება იმუშაოს კომპ-ზე, რომელსაც აქვს 4 გბ. მეხსიერება, თუკი დეტალურად გავიაზრებთ, რომელი 4 გბ. უნდა შევინახოთ მეხსიერებაში დროის ყოველ მომენტში. ამ დროს პროგრამის ნაწილი, რომელიც იმყოფება დისკზე, მეხსიერებაში, ადგილს მოინაცვლებენ საჭიროებისდა მიხედვით. 4-4-4 -ად.

ვირტუალურმა მეხსიერებამ შეიძლება აგრეთვე იმუშაოს მრავალ ამოცანიან სისტემაშიც, თუკი ბევრი პროგრამების ნაწილები ერთდროულად განთავსდება მეხსიერებაში. როცა პროგრამა ელოდება ოპერატიულ მეხსიერებაში რიგით მომდევნო მისი ნაწილის გადაადგილებას, მაშინ ის იმყოფება შეტანა-გამოტანის ლოდინის მდგომარეობაში და არ შეეძლება მუშაობა, ამიტომ ც.პ შეიძლება მიეცეს სხვა პროცესს, იგივე წესით, როგორც ნებისმიერ მრავალამოცანიან სისტემაში ხდება ხოლმე.

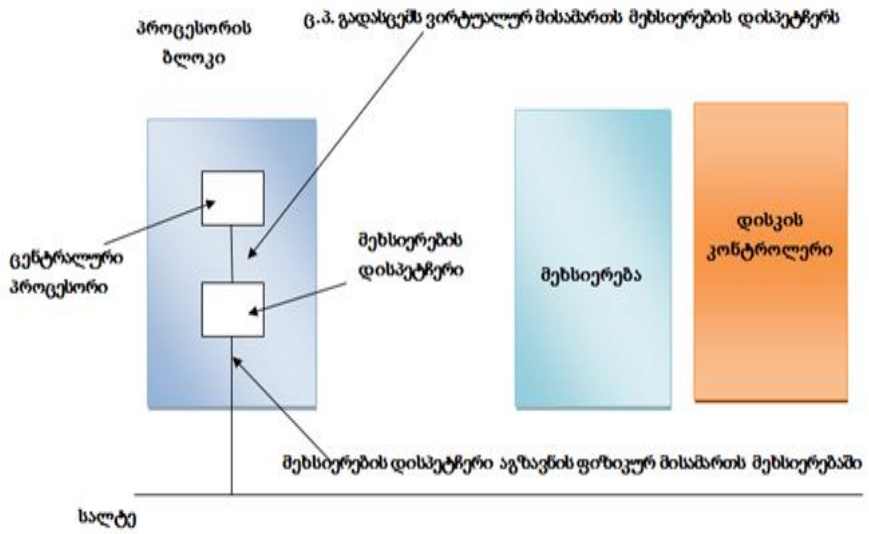
მეხსიერების ორგანიზება ფურცლებად

ვირტუალური მეხსიერების მრავალი სისტემა იყენებს ტექნიკას, რომელსაც ეწოდება ფეიჯინგი “paging“. ნებისმიერ კომპ-ში არსებობს უამრავი მისამართი მეხსიერებაში, რომელსაც შეიძლება მივმართოთ პროგრამით. როცა პროგრამა იყენებს შემდეგ ინსტრუქციას: MOV REG. 1000 ამას ის აკეთებს იმისათვის, რომ დააკოპიროს 1000 მისამართზე არსებული შინაარსი რეგისტრში REG. (ან პირიქით, კომპ-ისდა მიხედვით). მისამართები შეიძლება

დაფორმირდეს ინდექსაციის გამოყენებით, საბაზო რეგისტრებით, სეგმენტური რეგისტრებით და სხვა გზებით.

ეს პროგრამულად ფორმირებული მისამართები რომლებსაც ვირტუალურ მისამართებს უწოდებენ, აფორმირებენ ვირტუალურ სამისამართო სივრცეს. კომპ-ებში, რომლებსაც არა აქვთ ვირტუალური მეხსიერება, - ვირტუალური მისამართების გადაცემა ხდება უშუალოდ მეხსიერების სალტეზე და სიტყვას გამოიძახებენ წასაკითხად ან ჩასაწერად ფიზიკურ მეხსიერებაში იმავე მისამართზე. როცა გამოიყენება ვირტუალური მეხსიერება, მაშინ ვირტუალური მისამართი ამ წესით არ გადაიცემა. ამის ნაცვლად ისინი გადაეცემა მეხსიერების დისპეტჩერს (MMU – Memory Management Unit)რომელიც გადასახავს ვირტუალურ მისამართებს მეხსიერების ფიზიკურ მისამართებად როგორც ეს ნახ.16 არის ნაჩვენები.

ძალიან მარტივი მაგალითი, თუ როგორ მუშაობს გარდასახვა, მოყვანილია მე-16 ნახაზზე. ჩვენ განვიხილავთ კომპიუტერს, რომელიც შეიძლება დაფორმირდეს 16 თანრიგიანი მისამართებით, 0-დან 64 კბ. ესაა ვირტუალური მისამართი. მაგრამ თუ ამ კომპ-ს აქვს მხოლოდ 32კბ. ფიზიკური მისამართი, მიუხედავად იმისა, რომ პროგრამები ზომით 64კბ. შეიძლება დაწერონ, ისინი მაინც შეუძლებელია მთლიანად ჩაიტვირთოს მეხსიერებაში. მეხსიერების სახის სრული ასლი, 64კბ-მდე სიდიდის პროგრამისათვის, შეიძლება არსებობდეს დისკზე, მაგრამ ისეთი სახით, რომ საჭიროების შემთხვევაში მისი მეხსიერებაში ნაწილ-ნაწილ გადატანა შეიძლებოდეს.



ნახ.16. MMU - ს განაწილება და ფუნქციები. (აქ მეხსიერების დისპეტჩერი ნაჩვენებია როგორც პროცესორის მიკროსქემა. იმიტომ რომ დღეისათვის ეს მართლაც ასე არის. მაგრამ ლოგიკურად არ შეიძლება იყოს ის ცალკე მიკროქემად).

ვირტუალური მისამართების სივრცე დაყოფილია ერთეულებად რომლებსაც ეწოდება ფურცელი. შესაბამის ერთეულებს ფიზ. მეხსიერებაში ეწოდება ფურცვლოვანი ბლოკები. (page frame). ფურცლები და მათი გვერდები ყოველთვის ერთნაირგანზომილებიანია. ამ მაგალითში ისინი ტოლია 4კბ. მაგრამ რეალურ სისტემაში გამოიყენება 512 დან 64 კბ. თუკი გვაქვს 64 კბ ვირტ. სამისამართო სივრცე, და 32 კბ ფიზ.მეხსიერება, ჩვენ ვღებულობთ 16 ვირტუალურ გვერდს და 8 ფურცვლოვან ბლოკს. მონაცემების გადაცემა RAM-ის ფურცელსა და დისკს შორის ყოველთვის ხდება ფურცლებში.

პირდაპირი წვდომა მეხსიერებათან

იმის მიუხედავად, გარდაისახებიან თუ არა შეტანა-გამოტანის რეგისტრები ან ბუფერები მეხსიერებაზე, ცენტრალური პროცესორები(ცპ) მაინც უნდა როგორმე გადამისამართდეს მოწყობილობათა კონტროლერებთან, რომ მოხდეს მასთან მონაცემების გაცვლა. ცპ-მ შეტანა-გამოტანის კონტროლერებისაგან შეიძლება მოითხოვოს მონაცემები თითო ბაიტობით, მაგრამ მონაცემთა გაცვლის მსგავსი ორგანიზაცია უკიდურესად არეფექტურია, რადგან პროფესიული დროის დიდი ოდენობა სჭირდება. ამიტომ, პრაქტიკულად, ხშირად სხვა სქემა გამოიყენება, რომელსაც მეხსიერებათან პირდაპირი წვდომა ეწოდება. DMA(Direct Memory Access)

ოპერაციულმა სისტემამ შეიძლება ისარგებლოს მეხსიერებათან პირდაპირი წვდომით მხოლოდ მაშინ, თუკი სახეზეა აპარატურული DMA -კონტროლერი, რომელიც მართლაც აქვს სისტემების უმრავლესობას. ზოგჯერ DMA -კონტროლერები ინტეგრირებულია სხვა კონტროლერებთან, მაგ. დისკურ კონტროლერებში. ასეთი დიზაინი საჭიროებს DMA - კონტროლერებით მოხდეს ყველა პერიფერიული მოწყობილობის დაფუძვნება. როგორც წესი, დედა პლატაზე დაყენებული DMA ემსახურება სხვადასხვა, ზოგჯერ რამდენიმე შეტანა-გამოტანის მოწყობილობას, ხშირად კონკურენტულ საფუძველზე.

ფიზიკურად სადაც არ უნდა მდებარეობდეს, DMA - კონტროლერს შეუძლია მიიღოს წვდომა სისტემურ სალტეზე ც.პ-საგან დამოუკიდებლად, როგორც ეს ნაჩვენებია მე-18 ნახაზზე. ის შეიცავს რამდენიმე რეგისტრს, რომელიც წვდომადია ცპ-სათვის წასაკითხად და ჩასაწერად. მას მიეკუთვნება მეხსიერების მისამართის რეგისტრი, ბაიტების მთვლელი, ერთი ან რამდენიმე მმართველი რეგისტრი. მმართველი რეგისტრები იძლევიან ინფორმაციას, თუ რომელი შეტანა-გამოტანის პორტი შეიძლება იქნას გამოყენებული, განსაზღვრავენ მონაცემების გადატანის მიმართულებას(შეტანა-გამოტანის მოწყობილობიდან წაკითხვას

ან ჩაწერას). ასევე, გადატანის ერთეულს(მონაცემების გადატანა განხორციელდეს ბაიტაბად თუ სიტყვებით), აგრეთვე, ბაიტების რიცხოვნობას, რომლებიც შეიძლება გადატანილ იქნას ერთი ოპერაციით.

DMA - მუშაობის პრინციპის ასახსნელად ჯერ გავეცნოთ იმას, თუ როგორ ხდება დისკიდან წაკითხვა, როდესაც არა გვაქვს DMA . თავდაპირველად, ჩვენს მიერ განხილულ მოდელებში, რომლებსაც ზოგჯერ გამჭოლი რეჟიმიც ეწოდებათ, კონტროლერი DMA „ნებას რთავს“ მოწყობილობათა კონტროლერს გადააგზავნოს მონაცემები პირდაპირ ოპერატიულ მეხსიერებაში. ზოგიერთ DMA -ში გამოიყენება ისეთი რეჟიმები, რომელთა დროსაც მოწყობილობათა კონტროლერი გზავნის მონაცემების სიტყვას DMA კონტროლერთან, რომელიც შემდეგ დასვამს სალტეზე კიდევ ერთ მოთხოვნას ამ სიტყვის იქ გადასაცემად, სადაც საჭიროება მოითხოვს. ამგვარი სქემის დროს საჭირო ხდება სალტის ზედმეტი ციკლი, ყოველი სიტყვის გადასაცემად. მაგრამ ასეთი სქემა გამოირჩევა დიდი მოქნილობით, რადგან ასევე შესაძლებელს ხდის შესრულდეს კოპირება ერთი მოწყობილობიდან მეორეზე, მეხსიერების ჩათვლით. და მეხსიერებიდან - მეხსიერებაშიც კი. (ამისათვის საჭიროა თავდაპირველად გაიცეს მეხსიერებაზე წაკითხვის ბრძანება, შემდეგ კი ბრძანება მეხსიერებაში ჩაწერაზე, მაგრამ უკვე სხვა მისამართით).

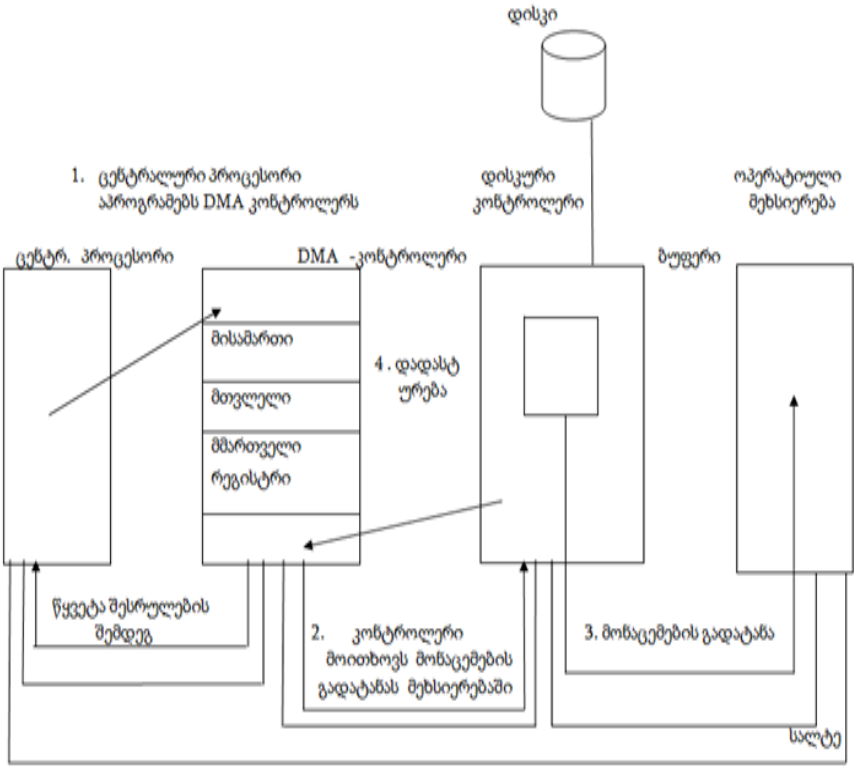
კონტროლერების უმრავლესობა მონაცემების გადასაცემად იყენებს მეხსიერების ფიზიკურ მისამართებს. ფიზიკური მისამართის გამოსაყენებლად ოპერაციულმა სისტემამ უნდა გარდასახოს მეხსიერების ბუფერის ვირტუალური მისამართი

ფიზიკურში და ჩაწეროს ეს ფიზიკური მისამართი DMA კონტროლერების მისამართების რეგისტრში. ზოგიერთ DMA კონტროლერებში გამოიყენება ალტერნატიული სქემა, რომლის დროსაც DMA კონტროლერში მაშინვე იწერება ვირტუალური მისამართი. ამ შემთხვევაში კონტროლერმა უნდა გამოიყენოს მეხსიერების მენეჯერი MMU მისამართის გარდასახვისათვის. ვირტუალური მისამართი შეიძლება დაყენდეს მისამართების სალტეზე მხოლოდ იმ შემთხვევაში, როცა MMU წარმოადგენს მეხსიერების რაღაც ნაწილს(რაც, შესაძლოა, რომ იშვიათობა არა არის) და არა ნაწილს ცენტრალური პროცესორისა.

როგორც უკვე აღვნიშნეთ, DMA -ს ოპერაციის დაწყებამდე, დისკი პირველად კითხულობს მონაცემებს თავის შინაგან ბუფერში, შესაძლოა თქვენ გაგიჩნდეთ კითხვა: -რატომ კონტროლერი არ გადაადგილებს მონაცემებს პირდაპირ ოპერატიულ მეხსიერებაში, მათი დისკიდან მიღებამდე? სხვა სიტყვებით რომ ვთქვათ, რაში სჭირდება შინაგანი ბუფერი? აქ ორი მიზეზია. ჯერ ერთი შინაგანი ბუფერიზაციის წყალობით დისკის კონტროლერს შეუძლია შეამოწმოს საკონტროლო ჯამი, მეხსიერებაში მონაცემების გადატანის დაწყებამდე. თუკი საკონტროლო ჯამები არ დაემთხვა, - ფორმირდება სიგნალი შეცდომის შესახებ და მონაცემთა გადატანა არ ხდება. მეორეც, საქმე იმაშია, რომ როგორც კი დისკიდან წაკითხვის ოპერაცია დაიწყება, - ბიტები იწყებენ მუდმივი სიჩქარით შემოსვლას, იმისგან დამოუკიდებლად, - დისკის კონტროლერი არის თუ არ არის მზად მათ მისაღებად. თუკი დისკის კონტროლერი შეეცდება ჩაწეროს ეს მონაცემები პირდაპირ მეხსიერებაში, მაშინ მას მოუწევს ეს გააკეთოს სისტემურ სალტეზე. თუკი მომდევნო სიტყვის გადაცემის დროს სალტე დაკავებული აღმოჩნდა ამა თუ იმ მოწყობილობით, (მაგ. იმით, რომელიც მას იყენებს პაკეტურ რეჟიმში) მაშინ დისკის კონტროლერს მოცდა მოუწევს. თუკი შემდეგი სიტყვა დისკიდან მოვა ადრე, ვიდრე კონტროლერი

მოასწრებს წინაშემოსულის შენახვას, მაშინ კონტროლერი ან დაკარგავს წინა სიტყვას, ან მას მოუწევს ის შეინახოს კიდევ სხვაგან სადმე. ამგვარად, შინაგანი ბუფერირების აუცილებლობა თვალნათლივი ხდება. ვირტუალური ბუფერის არსებობის შემთხვევაში დისკის კონტროლერს სალტე არ სჭირდება მანამ, სანამ არ დაიწყება DMA -ს ოპერაცა. ამის შედეგად დისკის კონტროლერის მოწყობილობა აღმოჩნდება უმარტივესი, რადგან მონაცემების გადაგზავნების DMA -ს ოპერაციის დროს დროის პარამეტრი არ წარმოადგენს კრიტიკულს.(ზოგიერთი ძველი კონტროლერები მართლაც პირდაპირ მიმართავდნენ მეხსიერებაზე, რადგან ჰქონდათ მცირე ზომის შინაგანი ბუფერი, რაც ხშირად გადატვირთვის დროს შეცდომებს იწვევდა სალტეების დატვირთვის დროს).

DMA -ყველა კომპ-ში არ გამოიყენება. მისი გამოუყენებლობის მთავარი არგუმენტია ის რომ, ცპ, როგორც წესი, მნიშვნელოვნად სჭარბობს DMA კონტროლერს სიჩქარით და შეუძლია შეასრულოს იგივე ოპერაცია მნიშვნელოვნად სწრაფად. (თუკი სიჩქარე მხოლოდ შეტანა-გამოტანის მოწყობილობების სწრაფქმედებით არ შემოიფარგლება). სხვა სამუშაოების არ არსებობის შემთხვევაში, სწრაფ ცპ-ს რომ აიძულო ლოდინი, სანამ ნელი DMA კონტროლერი შეასრულებს თავის სამუშაოს, - არის უაზრობა. გარდა ამისა, კომ-პი, DMA კონტროლერის გარეშე, ცპ -ით, რომელიც ყველა საქმეს ასრულებს პროგრამულად, - აღმოჩნდება იაფი, რაც უკიდურესად აუცილებელია დაბალი ფასის კატეგორიის(იაფი) კომ-პების წარმოების დროს. (მაგ. ჩაშენებული კომპიუტერები).



ნახ.17 წყვეტის სტრუქტურა

კიდევ ერთხელ წყვეტის შესახებ

ტიპიური პერსონალური კომპიუტერული სისტემისათვის წყვეტის სტრუქტურა წარმოდგენილია ნახ.18-ზე. აპარატურულ დონეზე წყვეტები შემდეგნაირად მოქმედებენ: როცა შეტანა-გამოტანის მოწყობილობა ამთავრებს თავის მუშაობას, ის ინიცირებს წყვეტას(ვიგულისხმოთ, რომ წყვეტა ნებადართულია

ოპ. სისტემის მიერ). ამისათვის მოწყობილობა დასვამს სიგნალს გამოყოფილ მოწყობილობაზე, სპეციალური სალტის ხაზზე, ეს სიგნალი აღიქმება წყვეტის კონტროლერის მიკროსქემის მეშვეობით, რომელიც განთავსებული არის დედა-პლატაზე. წყვეტის კონტროლერი ღებულობს გადაწყვეტილებას შემდგომი მუშაობის შესახებ.

წყვეტის სხვა დაუმუშავებელი მოთხოვნების არ არსებობის შემთხვევაში წყვეტის კონტროლერი ამუშავებს მას დაუყოვნებლივ. თუკი წყვეტა უკვე მუშავდება და ამავ დროს, სხვა მოწყობილობებიდან მოდის მოთხოვნა იმ ხაზით, რომელიც უფრო დაბალი პრიორიტეტით სარგებლობს, მაშინ ახალი მოთხოვნა უბრალოდ იგნორირდება. ამ შემთხვევაში მოწყობილობა განაგრძობს შეიკავოს წყვეტის სიგნალი სალტეზე მანამ, სანამ ის არ დაექვემდებარება ცპ - ს მომსახურებას.

წყვეტის დასამუშავებლად კონტროლერი დასვამს სამისამართო სალტეზე იმ მოწყობილობის ნომერს, რომელიც მოითხოვს ყურადღებას. და დააშენებს წყვეტის სიგნალს პროცესორის შესაბამის კონტაქტზე. ის სიგნალი აიძულებს პროცესორს შეაჩეროს მიმდინარე სამუშაო და დაიწყოს წყვეტის დამუშავების საქმე. ნომერიც, რომელიც დაისმება სამისამართო სალტეზე, გამოიყენება ცხრილში ინდექსის სახით, რომელსაც ეწოდება წყვეტის ვექტორი. საიდანაც ბრძანებათა მრიცხველის ახალი მნიშვნელობა აიღება. ბრძანებათა ახალი მრიცხველი მიუთითებს წყვეტის დამუშავების შესაბამის პროცედურის დასაწყისზე. როგორც წესი, ამ ადგილიდან აპარატურული და ემულირებული (ვირტუალური) წყვეტები იყენებენ ერთსა და იმავე მექანიზმს. და ხშირად სარგებლობენ ერთი და იმავე ვექტორით. ვექტორის მდებარეობა შეიძლება მყარად იყოს ჩაშენებული(ჩაკერებული)

აპარატურულ დონეზე და შეიძლება პირიქით, შეიძლება განთავსდეს მეხსიერების ნებისმიერ ადგილზე, რომელსაც მიუთითებს ცპ-ს სპეციალური რეგისტრი, რომელიც ჩაიტვირთვება ოპერაციული სისტემის მეშვეობით.

მალევე, თავისი საქმის დაწყებისთანავე, წყვეტის დამუშავების პროცედურა ადასტურებს წყვეტის მიღებას იმით, რომ, წყვეტის კონტროლერის პორტში ჩაწერს განსაზღვრულ მნიშვნელობას. ეს დადასტურება ნებას აძლევს კონტროლერს, გამოუშვას ახალი წყვეტები. იმის მეშვეობით, რომ ცპ-ს კიდევ შეუძლია დადასტურების გადადება იმ დრომდე, სანამ ის არ იქნება მზად ახალი წყვეტის დასამუშავებლად, - შესაძლებლობა არსებობს თავი ავარიდოთ იმ სიტუაციას, რომ ერთდროულად გაჩნდეს წყვეტის მოთხოვნილებები რამდენიმე მოწყობილობიდან. საჭიროა გავითვალისწინოთ, რომ ზოგიერთ კომპიუტერის მიკროსქემებზე არ არის წყვეტის ცენტრალიზებული კონტროლერები. ამიტომ ყოველი მოწყობილობის კონტროლერი სვამს თავის საკუთარ წყვეტას.

აპარატურა ყოველთვის, სანამ დაიწყებდეს წყვეტის დამუშავების პროცედურას, ინახავს განსაზღვრულ ინფორმაციას. ის და მისი შენახვის ადგილი ფართო ვარირებას ექვემდებარება ცპ-თან დამოკიდებულებისდა მიხედვით. როგორც მინიმუმ, შენარჩუნდება ბრძანებათა მთვლელი, რაც საშუალებას იძლევა შეწყვეტილი პროცესის შესრულება გაგრძელდეს. მეორე უკიდურესობა წარმოადგენს ყველა პროგრამული წვდომადი რეგისრტის შენახვას და ცპ - ის დიდი რაოდენობით ვირტუალური რეგისტრების შენახვას. ამ ინფორმაციის შენახვის ადგილიც ასევე პრობლემას წარმოადგენს. ერთი ვარიანტი იმაში მდგომარეობს, რომ შევინახოთ ეს მონაცემები რაიმე შიგა

რეგისტრში, რომელიც წვდომადია ოპ. სისტემისათვის. ამ მიდგომის ნაკლი ისაა, რომ სანამ მთლიანი შენახული ინფორმაცია არ იქნება წაკითხული, წყვეტის დამმუშავებლის მიერ, - შეუძლებელია ახალი წყვეტის დამუშავება. წინააღმდეგ შემთხვევაში ახალი წყვეტა უბრალოდ, წაშლიდა ამგვარად შენახულ მთელ ინფორმაციას, როცა თავზე გადააწერდა მათ ახალ მონაცემებს. წყვეტის შედეგად აღმოჩნდებოდა მოთხოვნილებები ხანგრძლივი დროის წინა ინტერვალებში დაფიქსირებულ ინფორმაციაზე, რაც ზოგიერთი სიგნალების შესაძლო იგნორირებამდე მიგვიყვანდა წყვეტის ამ მოწყობილობებიდან და, შესაბამისად, მონაცემების შესაძლო დაკარგვასაც ექნებოდა ადგილი.

ამიტომ, ცპ -ს უმრავლესობა ინფორმაციას შეთავსებულად ინახავს. მაგრამ ამ მიდგომასაც აქვს ნაკლი. ჯერ ერთი, ვის(რის) საფუძველზე უნდა შევინახოთ მონაცემები: თუკი გამოვიყენებთ მიმდინარეს, - ის შეიძლება აღმოჩნდეს მომხმარებლის პროცესის შეტავსება. ამ დროს შეიძლება ისიც გახდეს ცნობილი, რომ მომხმარებელი იყენებს თანხვდენის მიმნიშნებელს, თავის პროგრამაში, სრულიად არასტანდარტულად. ანუ, მან შეიძლება მიუთითოს მეხსეირების არე, რომელშიც არ შეიძლება მონაცემების შენახვა. სტეკში რამდენიმე სიტყვის ჩაწერის მცდელობამ ამ შემთხვევაში შეიძლება გამოუსწორებელ შცდომამდე მიგვიყვანოს. ასევე, ამ მაჩვენებელმა შეიძლება მიგვანიშნოს მეხსიერების ფურცლის გვერდის ბოლოსკენაც!!!!

თავდაპირველად კონტროლერი დისკიდან ბლოკს - ერთ ან რამდენიმე სექტორს თანმიმდევრობით, ბიტებად კითხულობს, მანამ მთელი ბლოკი არ აღმოჩნდება კონტროლერის შიგა ბუფერზე. შემდეგ კონტროლერი ამოწმებს საკონტროლო ჯამს,

რომ დარწმუნდეს, კითხვის დროს ხომ არ მოხდა რაიმე შეცდომა. შემდეგ კონტროლერი ახდენს წყვეტის ინიცირებას. როცა ოპერაციული სისტემა იწყებს მუშაობას, მან შეიძლება წაიკითხოს ბლოკი დისკიდან ბაიტებად ან სიტყვებად. ოპერატიულ მეხსიერებაში, ციკლში შეინახავს სააღრიცხვო სიტყვას ან ბაიტს.

DMA -ს გამოყენების დროს პროცედურა სრულიად განსხვავებულია. თავდაპირველად ც.პ. აპროგრამებს DMA - კონტროლერს, დააყენებს რა, მის რეგისტრს და ამგვარად მიუთითებს, თუ რომელი მონაცემი და სად უნდა მოათავსოს. (1 ბიჯი). შემდეგ პროცესორი ბრძანებას აძლევს დისკის კონტროლერს წაიკითხოს მონაცემები შიგა ბუფერიდან და შეამოწმოს საკონტროლო ჯამი. როცა მონაცემები მიღებული და შემოწმებულია დისკის კონტროლერით, DMA -მ შეიძლება დაიწყოს მუშაობა. DMA-კონტროლერი იწყებს მონაცემების გადატანას. ესაა დისკის კონტროლერისათვის სალტის მეშვეობით მოთხოვნა წაკითხვაზე. (ბიჯი2). ეს ჩვეულებრივი წაკითხვის ბრძანებასავით გამოიყურება. ასე, რომ დისკის კონტროლერმა არც კი იცის, ის ცპ-დან მოვიდა თუ DMA-კონტროლერიდან. როგორც წესი, მეხსიერების მისამართი უკვე იმყოფება მისამართის სალტეზე, ასე, რომ დისკის კონტროლერმა უკვე იცის, ყოველთვის სად გააგზავნოს მომდევნო სიტყვა თავისი შიგა ბუფერიდან. მეხსიერებაში ჩაწერა სალტის კიდევ ერთ სტანდარტულ ციკლს წარმოადგენს.(ბიჯი3). როცა ჩაწერა დამთავრებულია, დისკის კონტროლერი აგრეთვე სალტით აგზავნის DMA-თი დამოწმების სიგნალს(ბიჯი 4). შემდეგ DMA-კონტროლერი ზრდის მეხსიერების გამოყენებულ მისამართს და ამცირებს ბაიტების მთვლელის მნიშვნელობას. ამის შემდეგ მე-2 - მე-4 ბიჯები მეორდება, მანამ, სანამ მთვლელის მნიშვნელობა არ გახდება 0. ციკლის დასრულების შემდეგ, კოპირებული DMA-კონტროლერი ინიცირებს პროცესორის წყვეტას, ატყობინებს რა,

მას, ისე, რომ მონაცემთა გადატანა დასრულებულია. ოპერაციულ სისტემას არ სჭირდება დაკოპირების დისკის ბლოკი მეხსიერებაში. ის უკვე ისედაც იქ იმყოფება.

DMA-კონტროლერები მნიშვნელოვნად განსხვავდებიან თავისი სირთულის და ხარისხის მიხედვით. მათ შორის ყველაზე მარტივები ერთ ჯერზე ასრულებენ ერთ ოპერაციას-მონაცემების გადატანას, ისე, როგორც ზემოთ აღიწერა. უფრო რთულ კონტროლერებს შეუძლიათ ერთბაშად შეასრულონ რამდენიმე მსგავსი ოპერაცია. ასეთ კონტროლერებს აქვთ რამდენიმე არხი, რომელთაგან თითოეული შიგა რეგისტრების თავისი ერთობლიობის მეშვეობით იმართება.

ცენტრალური პროცესორი იწყებს იქიდან, რომ ამ რეგისტრებში ტვირთავს შესაბამის პარამეტრებს. მონაცემების გადატანის ყველა ოპერაცია, უნდა შესრულდეს შეტანა-გამოტანის სხვადასხვა მოწყობილობების მეშვეობით. მონაცემების ყოველი სიტყვის შეტანის შემდეგ(ბიჯი 2-4) DMA-კონტროლერი გადაწყვეტს, თუ რომელ მოწყობილობას უნდა მოემსახუროს მომდევნო ჯერზე. ეს შერჩევა შეიძლება განხორციელდეს ციკლურად, ან რაიმე პრიორიტეტული სქემის მეშვეობით, რომელიც რომელიმე ერთ მოწყობილობას რაღაც უპირატესობას ანიჭებს სხვეთან შედარებით. შეიძლება ერთდროულად რამდენიმე მოთხოვნა ელოდებოდეს შესრულების რიგს, იმ პირობით, რომ არსებობს წესი, რომელიც სხვადასხვა მოწყობილობების დადასტურებებს ერთმანეთისაგან ცალსახად განასხვავებს. ხშირად ამ მიზნით DMA-ს ყოველი არხისათვის დადასტურებებს სხვადასხვა ხაზები გამოიყენება(დადასტურების ხაზებია).

მრავალ სალტეს შეუძლია ორ რეჟიმში მუშაობა: სიტყვების ან ბლოკების. ზოგიერთი DMA-კონტროლერი შეიძლება ორივე რეჟიმში ფუნქციონირებდეს. სიტყვების რეჟიმში პროცედურა ისე გამოიყენება, როგორც ზემოთ იყო აღწერილი. DMA-კონტროლერი დასვამს მოთხოვნას ერთი

სიტყვის გადატანაზე და ღებულობს მას. თუკი ცპ-ს ასევე ესაჭიროვება იგივე სალტე, მაშინ მას მოცდა მოუწევს. ამ მექანიზმს ეწოდება „ციკლის დაპყრობა“(Cycle Stealing). იმიტომ, რომ მოწყობილობის კონტროლერი პერიოდულად აიკრეფს სალტის შემთხვევით ციკლებს ცპ- დან.(ადვილად აყოვნებს მას და იმიტომ). ბლოკურ რეჟიმში DMA-კონტროლერი ნებას აძლევს მოწყობილობას, რომ სალტის დაკავება შეძლოს. შეასრულოს გადაგზავნის სერია და სალტე გაუშვას. მოქმედებათა ასეთ წესს ეწოდება ***პაკეტური რეჟიმი.*** ის უფრო ეფექტურია, ვიდრე ციკლის დაპყრობა, რადგანაც სალტის დაკავება დროს საჭიროებს. ხოლო პაკეტურ რეჟიმში ეს პროცედურა მხოლოდ ერთხელ სრულდება, რომ მთელ ბლოკს გადასცეს მონაცემები. ამ მეთოდის ნაკლი ისაა, რომ მონაცემთა დიდი ბლოკის გადაცემის დროს მან შეიძლება დაბლოკოს ცპ და სხვა მოწყობილობები დროის მნიშვნელოვან შუალედებში.

ფაილური სისტემის რეალიზაცია

ახლა საჭიროა მომხმარებლის თვალსაზრისით განვიხილოთ ფაილური სისტემა, რომელიც დამმუშავებლის თვალსაზრისით არის შექმნილი. მომხმარებელს აინტერესებს ის, თუ რას ეწოდება ფაილი, რა ოპერაციები შეიძლება მასზე შესრულდეს, როგორ გამოიყენება ფაილური ხე და ა.შ. მსგავსი ინტერფეისული საკითხები.

ფაილური სისტემის დამპროექტებლები ინტერესდებიან იმით, თუ როგორ ინახება ფაილები კატალოგში, როგორ ხორციელდება დისკური სივრცის მართვა და როგორ მივაღწიოთ საიმედო და ეფექტურ მუშაობას ფაილურ სისტემაში.

ფაილური სისტემის სტრუქტურა. ფაილური სისტემა ინახება

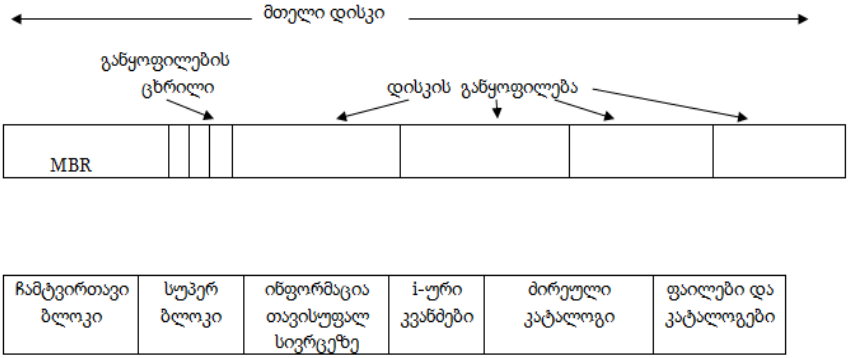
დისკზე. დისკების უმრავლესობა იყოფა რამდენიმე განყოფილებად, თითოეულ მათგანში ფაილური სისტემისაგან დამოუკიდებლად. სექტორი O დისკისა იწოდება მთავარ ჩამტვირთავ ჩანაწერად. MBR – Master Boot Record და გამოიყენება კომპიუტერის ჩასატვირთავად. მთავარი ჩამტვირთავი ჩანაწერის ბოლოში შენარჩუნებულია განყოფილების ცხრილი. ამ ცხრილში ინახება თითოეული განყოფილების საწყისი და საბოლოო მისამართები.(ბლოკების ნომრები). განყოფილებიდან ერთერთი დაჭდევებულია, როგორც ცხრილში აქტიური განყოფილება. კომპის ჩატვირთვის დროს BIOS ითვლის და ასრულებს MBR - ჩანაწერს, რის შემდეგაც MBR – ში ჩამტვირთავი განსაზღვრავს დისკის აქტიურ ნაწილს. ანგარიშობს მის პირველ ბლოკს, რომელსაც ეწოდება ჩამტვირთავი და ასრულებს მას. პროგრამა, რომელიც მოთავსებულია ჩამტვირთავ ბლოკში, - ტვირთავს ოპ. სისტემას, რომელიც იმყოფება ამ განყოფილებაში.

ერთიანობისათვის, ყოველი დისკური განყოფილება იწყება ჩამტვირთავი ბლოკებისაგან, იმის მიუხედავად, რომ შეიძლება მასში არც კი იყოს ჩამტვირთავი ოპერაციული სისტემა. ამავე განყოფილებაში შეიძლება შემდგომში დაყენდეს ოპერაციული სისტემაც, ამიტომ დარეგისტრირებული ჩამტვირთავი ბლოკი აღმოჩნდება სასარგებლო. დისკის სხვა დანარჩენი განყოფილებები იცვლებიან სისტემიდან -სისტემამდე.

ხშირად ფაილური სისტემები შეიცავენ სხვადასხვა ელემენტებს, რომლებიც წარმოდგენილია ნახ.18-ზე. ერთერთი ასეთი ელემენტი, ე.წ. სუპერბლოკი, რომელიც შეიცავს ფაილური სისტემის გასაღებ პარამეტრებს, და გამოითვლებიან მეხსიერებაში კომპიუტერის ჩატვირთვის დროს ან ფაილურ სისტემაზე პირველი მიმართვის დროს. ტიპიური ინფორმაცია,

რომელიც ინახება სუპერბლოკში, შეიცავს „მაგიურ რიცხვს“, რომელიც საშუალებას გვაძლევს, განვასხვავოთ სისტემური ფაილები, ბლოკების რაოდენობა ფაილურ სისტემაში და აგრეთვე სხვა გამხსნელი ინფორმაცია.

გზადაგზა(ნაკვალევზე) განლაგდება ინფორმაცია ფაილური სისტემის თავისუფალი ბლოკების შესახებ. მაგ. ბიტური მასივებისა და და/ან მაჩვენებლების სიის სახით. ამ მონაცემებს შეიძლება მოჰყვებოდეს ინფორმაცია i-რი კვანძების შესახებ, რომლებიც წარმოადგენენ მონაცემთა სტრუქტურების მასივს ერთ სტრუქტურის სახით იმ ფაილზე, რომელიც შეიცავს მთელ ინფორმაციას ფაილების შესახებ. ამის შედეგად შეიძლება განთავსდეს ძირეული კატალოგი, რომელსაც ხის სათავეში აქვს ფაილური სისტემა. და ბოლოს, დისკური განყოფილების დანარჩენი ადგილები განკუთვნილია სხვა დანარჩენი კატალოგებისა და ფაილებისათვის.



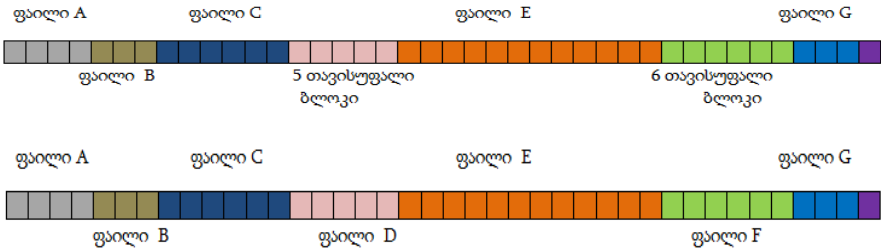
ნახ.18. ფაილური სისტემის შესაძლო სტრუქტურა

ფაილების რეალიზაცია

ფაილების შენახვის რეალიზაციაში ყველაზე მოწშვნელოვანი მომენტი არის დისკის ბლოკების ფაილებთან შესატყვისობის აღრიცხვა. იმის განსასაზღვრავად, თუ რომელი ბლოკი რომელ ფაილს მიეკუთვნება, სხვადასხვა ოპ.სისტემაში სხვადასხვა მეთოდები გამოიყენება. ერთერთს წარმოადგენს უწყვეტი ფაილების მეთოდი.

უწყვეტი ფაილების მეთოდი

ფაილებისათვის დისკზე განსაზღვრული ბლოკების გამოყოფის უმარტივეს სქემას წარმოადგენენ სისტემა, რომელშიც ფაილები მეზობელი ბლოკების უწყვეტი ერთიონბაა. მაშინ დისკზე, რომელიც შედგება ერთი კბ. მოცულობის ბლოკებისაგან, ფაილებისაგან - ზომით 50კბ, - შეიძლება დაიკავოს 50 თანმიმდევრული ბლოკი. 2კბ ბლოკების შემთხვევაში ასეთი ფაილი დაიკავებს 25 მეზობელ ბლოკს. უწყვეტი ფაილების მაგალითი ნაჩვენებია ნახ.19 - ზე. აქ ნაჩვენებია დისკის პირველი -40 ბლოკი. დაწყებული მარცხნიდან 0 - ით. თავიდან დისკი ცარიელი იყო. შემდეგ დისკზე, დაწყებული 0 ბლოკიდან, ჩაიწერა ფაილი A სიგრძის 4 ბლოკით. შემდეგ ჩაიწერა B ფაილი 3 ბლოკიანი. დაბრუნება A ფაილთან. მივაქციოთ ყურადღება რომ ყოველი ფაილი იწყება ახალი ბლოკიდან. ასე, რომ თუკი Aფაილის სიგრძეა 3,5 ბლოკი, მაშინ რაღაც ადგილი ბოლო ბლოკის ბოლოში იკარგება. სურათზე სულ 7 ფაილია. თითოეული ფაილი იწყება ბლოკით, რომელიც მოსდევს წინა ფაილის ბოლო ბლოკს. გაფერადება გამოყენებულია მხოლოდ იმ მიზნით, რომ ადვილი იყოს ცალკეული ფაილების გარჩევა.



ნახ.19. შვიდი უწყვეტი ფაილი ერთ დისკზე (ა) მდგომარეობა დისკიდან 2 ფაილის წაშლის შემდეგ (ბ)

უწყვეტ ფაილებს აქვთ ორი მნიშვნელოვანი უპირატესობა. ჯერ ერთი, ასეთი სისტემა ადვილად რეალიზებადია, რადგან, სისტემას, რომ განისაზღვროს, რომელი ბლოკები მიეკუთვნება ამა თუ იმ ფაილს, უნდა თვალი ვადევნოთ, მხოლოდ ორი რიცხვით: ფაილის პირველი ბლოკის ნომრის მიხედვით და ფაილში ბლოკების რიცხვით. ვიცით, რა ფაილის პირველი ბლოკი, - მისი ნებისმიერი სხვა ბლოკი ადვილად შეიძლება მივიღოთ დამატების მარტივი ოპერაციით.

მეორეც, უწყვეტი ფაილების მუშაობის დროს მწარმოებლობა საუკეთესოა, რადგან მთლიანად ფაილი შეიძლება წაკითხულ იქნას დისკიდან ერთი ოპერაციით. საჭიროა მხოლოდ ძიების ერთი ოპერაცია. (I ბლოკისათვის). ამის შემდეგ მეტად აღარ არის საჭირო ვეძებოთ ცილინდრები და დრო დავკარგოთ დისკთან დამშვიდობებისათვის ლოდინში. ამიტომ მონაცემები შეიძლება გამოითვალოს მაქსიმალური სიჩქარით. რისი უნარიც დისკს გააჩნია. ამგვარად, უწყვეტი ფაილები ადვილად რეალიზდება და მაღალმწარმოებლურია.

სამწუხაროდ, დისკური სივრცის განაწილების ასეთ წესს აქვს სერიოზული ხარვეზებიც. დისკი დროდადრო ხდება ფრაგმენტირებული. რომ გავიგოთ ეს როგორ ხდება, განვიხილოთ ნახ.19(ბ) . ორი ფაილი D და F მოშორებულია. ამ დროს მათი ბლოკები თავისუფლდება და რჩება შორის მდებარე ადგილები. თავისუფალი ბლოკების მეშვეობით დისკზე. დისკიდან ფაილების წაშლის კვალობაზე სულ უფრო მეტად ხდება დისკი „ნახვრეტებიანი“. დასაწყისში ეს ფრაგმენტაცია არ წარმოადგენს პრობლემურს. რადგან ყოველი ახალი ფაილი შეიძლება ჩაიწეროს დისკის ბოლოში, წინა ფაილის შემდეგ. მაგრამ, ბოლოს და ბოლოს დისკი ხომ ივსება და/ან საჭირო გახდება სპეციალური ოპერაცია დისკის გამოყენებული სივრცის შეკვეცისათვის, შემჭიდროვებისათვის, ანდა საჭირო გახდება გამოიძებნოს წესი რომ გამოვიყენოთ თავისუფალი არე დაშორებულ ფაილებისაგან დატოვებულ ადგილებზე. გამონთავისუფლებული სივრცის განმეორებითი გამოყენებისათვის საჭიროა გვქონდეს დისკზე თავისუფალი მონაკვეთების სია, რაც, პრინციპში, - შესრულებადია. მაგრამ, ახალი ფაილის შექმნის დროს, საჭირო იქნება ვიცოდეთ მისი საბოლოო ზომა, რომ შევარჩიოთ მისთვის შესაფერისი ადგილი ამ „თავისუფლებს“ შორის.

წარმოვიდგინოთ ასეთი სტრუქტურის შედეგი. მომხმარებელი უშვებს ტექსტურ რედაქტორს ან ტექსტურ პროცესორს დოკუმენტის შესაქმნელად. პირველი, რითაც პროგრამა ინტერესდება, არის - რამდენი ბაიტი იქნება დოკუმენტში. ამ კითხვაზე პასუხი უნდა გაეცეს. წინააღმდეგ შემთხვევაში პროგრამა არ იმუშავებს. თუ მომხმარებელი მიუთითებს ძალზე მცირე რიცხვს, პროგრამა შეიძლება ავარიულად დასრულდეს, რადგან დისკის თავისუფალი არე შეივსება და ფაილის

დარჩენილი ნაწილი სადღა ჩაიწეროს, - ეს გახდება პრობლემა. თუკი მომხმარებელი შეეცდება თავი აარიდოს ამ პრობლემას და მიუთითებს ძალზე დიდ საბოლოო ზომას, მაგ. 100მბ. მაშინ შეიძლება მოხდეს, რომ რედაქტორმა ვერ შეძლოს ასეთი დიდი თავისუფალი არეს მოძებნა და შეგვატყობინოს, რომ არ შეუძლია ასეთ პირობებში ფაილის შექმნა. ცხადია, მომხმარებელი დაიწყებს „ შევაჭრებას“ და ჩამოვა 50მბ. -მდე. და ა.შ. სანამ არ მოიძებნება თავისუფალი ადგილი, მაგრამ, ცხადია, ასეთი სქემა მაინცდა მაინც „დიდ სიამოვნებას“ არ აძლევს მომხმარებელს.

მიუხედავად ყველაფრისა, არის სიტუაციები, სადაც უწყვეტი ფაილები შეიძლება გამოვიყენოთ და და მართლაც ფართოდ გამოიყენება კომპაქტ-დისკებზე, აქ ფაილების ყველა ზომა წინასწარ ცნობილია და არ მოხდება ცვლილებები CD –ROM გამოყენების შედეგად.

ახალი ტექნოლოგიების გაჩენის კვალობაზე კიბერნეტიკაში ისტორია ხშირად მეორდება. ფაილური სისტემები, რომლებიც შედგებიან უწყვეტი ფაილებისაგან, გამოიყენებოდა ბევრ მაგნიტურ დისკზე დიდი ხნის წინათ მათი მუშაობის სიმარტივისა და მაღალი წარმადობის წყალობით. (მომხმარებლისთვის მოხერხებული სამუშაო გარემოს ცნება მაშინ საერთოდაც კი არ განიხილებოდა). შემდეგ ეს იდეა მიივიწყეს, იმიტომ, რომ cd და dvd ზუსტი ზომების მითითებით გვარდებოდა პრობლემა. ძველი სისტემების და იდეების შესწავლას ის სიკეთე მოაქვს, რომ მაშინდელი ბევრი ნათელი და მარტივი კონცეფცია - ახალ სისტემებში უაღრესად მისაღები სახისაც კი აღმოჩნდა დღეს.

მისამართის სწრაფად გარდასახვის ბუფერები
Translation Lookaside Buffer

ფურცლებად ორგანიზების მეხსიერების უმრავლეს სქემებში ფურცლების ცხრილები ინახება მეხსიერებაში მათი მნიშვნელოვანი სიდიდეების გამო. პოტენციალურად, ასეთი მოწყობილობა უძლიერეს გავლენას ახდენს მწარმოებლებზე. განვიხილოთ პროცესორის ბრძანება, რომელიც ახდენს ერთი რეგისტრის შინაარსის კოპირებას მეორეში. მეხსიერების ფურცვლოვანი ორგანიზაციის არქონის შემთხვევაში ეს ბრძანება ახდენს მხოლოდ ერთ მიმართვას მეხსიერებასთან თავად ბრძანების შესარჩევად. თუკი მეხსიერება ფურცლებად ორგანიზებულია, მაშინ აუცილებელი იქნება დამატებითი გზავნილებები ფურცლის ცხრილის წვდომისათვის. რადგან ბრძანებების შესრულების სიჩქარე ძირითადად დამოკიდებულია და შეზღუდულია სიჩქარით, რომლითაც ცპ. შეარჩევს ბრძანებებსა და მონაცემებს მეხსიერებიდან, ამიტომ ფურცლის ცხრილზე ორი მიმართვის აუცილებლობა, რაც ერთ გზავნილებაზე მოდის, - ამცირებს მწარმოებლობას 2/3 -ით. ასეთ პირობებში არავინ ისურვებდა ამ მეთოდის გამოყენებას.

კომპიუტერების დამმუშავებლები წლების განმავლობაში ფიქრობდნენ ამ პრობლემებზე და გადაწყვეტილებასაც მიაგნეს. ის ეფუძვნება იმ დაკვირვებას, რომ პროგრამათა უმრავლესობა, როგორც წესი, ახდენს მიმართვათა დიდ რაოდენობას ფურცლების მცირე რაოდენობასთან და არა პირიქით. ამრიგად, ფუ რცლების ცხრილში მხოლოდ ჩანაწერების მცირედი წილი იკითხება ინტენსიურად, დანარჩენები კი... საკითხავია, - გამოიყენება კი , საერთოდ?

მიღებული გადაწყვეტილების შედეგად, კომპ-ის აღჭურვა ხდება აპარატურული მოწყობილობით, რომელიც ემსახურება ვირტუალური მისამართების ფიზიკურში გარდასახვის საკითხს, ისე, რომ არ გაიარონ ფურცლების ცხრილებზე. ეს მოწყობილობა არის მეხსიერების ბუფერი მისამართების სწრაფი გარდასახვისთვის. ანუ, მას ასოციაციურობის მეხსიერებასაც უწოდებენ. ეს ნახ.20 -ზე არის ნაჩვენები. ის, როგორც წესი, იმყოფება მეხსიერების დისპეტჩერის შიგნით და შედგება რამდენიმე ჩანაწერისაგან. ამ ჩანაწერში(მაგალითისათვის ის არის 8). ფაქტობრივად, ჩანაწერი იშვიათად აჭარბებს 64-ს. ყოველი ჩანაწერი შეიცავს ინფორმაციას ერთი ფურცლის შესახებ. სახელდობრ: ვირტუალური გვერდის ნომერი, ბიტი, რომელიც ყენდება გვერდის ცვლილების დროს, დაცვის კოდი, (ნებართვა, წაკითხვა /ჩაწერა/ შესრულებაზე). და ფიზიკური ფურცვლოვანი ბლოკის ნომერი, რომელშიც მოთავსებულია ეს მოცემული გვერდი. ეს ველები ერთმნიშვნელოვნად შეესაბამებიან ველებს ფურცლის ცხრილში. კიდევ ერთი ბიტი ემსახურება ნიშნაკს იმისას, რომ ნამდვილად ხდება თუ არა ჩაწერა(ანუ გამოიყენება თუ არა ის მოცემულ მომენტში).

მაგალითი, რომელიც შეიძლება ფორმირებულ იქნას TLB-ბუფერით, წარმოდგენილია მე-20 ნახაზზე. ესაა ციკლური პროცესი, რომელიც განთავსებულია ვირტუალურ გვერდებზე 19,20, 21. ამიტომ, ამ ჩანაწერებს ბუფერში აქვთ დაცვითი კოდები. წაკითხვისა და შესრულებისათვის. ძირითადი მონაცემები, რომლებიც გამოიყენება დღევანდელ მოცემულ პირობებში, (დასამუშავებელი მასივი) იმყოფება 129, 130 გვერდზე. გვ.40 შეიცავს ინდექსებს, რომელიც საჭიროა მასივის გამოსათვლელად. და ბოლოს, გვ.860 და 861 ეთმობა შერწყმას, ზედდებას. ახლა

განვიხილოთ, როგორ ფუნქციონირებს ბუფერი TLB. როდესაც ვირტუალური მისამართი წარედგინება მეხსიერების

რეალური	ვირტუალური გვერდი	ცვლილება	დაცვა	ფურცლოვანი ფრეიმი
1	140	1	RW	31
1	20	0	RX	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	RX	50
1	21	0	RX	45
1	860	1	RW	14
1	861	1	RW	75

ცხრ.20 მეხსიერების სწრაფი გარდასახვის ბუფერი ფურცლოვანი გარდასახვის სიჩქარის გასაზრდელად

დისპეტჩერის მიერ გარდასახვისათვის, აპარატურა პირველ რიგში, რწმუნდება იმაში, რომ მისი ვირტუალური გვერდის ნომერი იმყოფება TLB ბუფერში. ეს ხდება მისამართის შედარებით ყველა ჩანაწერთან ერთდროულად ანუ, პარალელურად. ე.ი. ყველაფერი რიგზეა. თუკი შესაძლებელია დამთხვევა და მიმართვა არ არღვევს დაცვის კოდს, მაშინ ფურცვლოვანი ბლოკი აიღება პირდაპირ ბუფერიდან ისე, რომ არ გაიაროს ფურცლის ცხრილი. თუკი ვირტუალური გვერდის ნომერი ბუფერში არსებობს, მაგრამ ინსტრუქცია ცდილობს ჩაწეროს იმ სხვა რაიმე გვერდზე, რომელიც წვდომადია მხოლოდ წასაკითხად, - მაშინ ფორმირდება დაცვის შეცდომა ზუსტად ისე, როგორც ეს ხდებოდა

თავად ფურცლის ცხრილის გამო.

საინტერესო სიტუაცია მიიღება, როცა ვირტუალური ფურცლის ნომერი არ იმყოფება TLB ბუფერში. მეხსიერების დისპეტჩერი აღმოაჩენს ფურცლის არ არსებობას და ასრულებს ბუნებრივ ძიებას ფურცლის ცხრილში. შემდეგ ის მოიცილებს ერთერთ ჩანაწერთაგანს ბუფერიდან და შეცვლის მას ეს-ესაა ნაპოვნი ჩანაწერით. (ფურცლის ცხრილში რომ იპოვნა). ამგვარად, თუკი ეს ფურცელი ისევ მალე გახდა მოთხოვნის ობიექტი, მაშინ მეორე ჯერზე ძებნა აღმოჩნდება უფრო წარმატებული და არა წარუმატებელი. როცა ჩანაწერი მოცილდება ბუფერიდან, მაშინ ცვლილების ბიტი კოპირდება ფურცლის ცხრილის ჩანაწერში. მეხსიერებაში. სხვა სიდიდეები უკვე ისედაც იქ იმყოფებოდნენ. როცა TLB ბუფერი ჩაიტვირთება ფურცლის ცხრილიდან, მაშინ ყველა ველი აიღება მეხსიერებიდან.

TLB ბუფერის პროგრამული მართვა

აქამდე ჩვენ მივიჩნევდით, რომ ყოველ გამოთვლით მანქანას ფურცვლოვანი ვირტუალური მეხსიერებით, აქვს ფურცლების ცხრილი, რომელიც აღიქმება აპარატურული უზრუნველყოფის მიერ და TLB ბუფერის მიერ. ამგვარ მოწყობილობაში ბუფერის მართვა და მისი შეცდომების დამუშავება სრულდება მთლიანად აპარატურული დისპეტჩერული მეხსიერების მიერ. (MMU). მართვის გადაცემა ოპერაციულ სისტემაზე ხდება მხოლოდ მაშინ, როცა მეხსიერებაში გვერდი არ მოიძებნება.

ადრე ეს დაშვება სამართლიანად მიიჩნეოდა მაგრამ ბევრი თანამედროვე RISC კომპანია, SPARC, MIPS, ALPHA, HP PA ჩათვლით, თითქმის ყველა ფურცვლოვან მართვას ასრულებენ პროგრამულად. ამ მანქანებზე TLB ბუფერის ჩანაწერები ცხადად,

აშკარად იტვირთებიან ოპერაციული სისტემებით. როცა ხდება წარუმატებელი ძიება, მაშინ მეხსიერების დისპეტჩერი, იმის ნაცვლად, რომ გადავიდეს ფურცლის ცხრილზე, საჭირო გვერდის ძიებისათვის და არჩევისათვის, - ის აფორმირებს ბუფერის შეცდომას და პრობლემას გადაიყვანს ოპერაციული სისტემის ხელში. სისტემამ უნდა იპოვნოს გვერდი, მოაცილოს ჩანაწერი TLB ბუფერიდან, შეიტანოს ახალი ჩანაწერი და გადაგზავნოს(გაუშვას) შეწყვეტილი ინსტრუქცია. და, ცხადია, ყოველივე ეს უნდა გააკეთოს ბრძანებათა მცირე რიცხვის მეშვეობით. რადგანაც ძიების წარუმატებლობა TLB ბუფერში შედარებით უფრო ხშირადაა, ვიდრე შეცდომები ფურცლის არქონის გამო.

საკმაოდ საინტერესოა ის, რომ თუკი TLB ბუფერს აქვს შედარებით მცირე მოცულობა, მაგ. 64 ჩანაწერი, მაშინ წარუმატებელი გაშვებების სიხშირის შესამცირებლად ბუფერის პროგრამული მართვა, როგორც ჩანს, ყველაზე შედეგიანია. მთავარი მოგება აქ ის არის, რომ გაცილებით მარტივად არის მოწყობილი მეხსიერების დისპეტჩერი, რაც ანთავისუფლებს სივრცის საკმარის რაოდენობას პროცესორის მიკროსქემაში. ქეშისათვის და სხვა მოწყობილობისათვის, რომლებსაც შეუძლიათ აამაღლონ მწარმოებლობა.

TLB ბუფერების პროგრამული მართვის მანქანების მწარმოებლობის ამაღლებისათვის დამუშავებულია მოქმედების სხვადასხვა სტრატეგიები. ერთერთი მიდგომაა იმის მცდელობა, რომ შემცირდეს როგორც წარუმატებელი ძიებების სიხშირე ბუფერში და ასევე მისი ღირებულება. (თუკი ის მაინც და მაინც უნდა არსებობდეს. შემცირდეს მაინც!). იმისათვის, რომ შემცირდეს წარუმატებელი ძიებების ალბათობა TLB ბუფერში, ზოგჯერ ოპერაციულ სისტემაში შეიძლება ინტუიციურად

გამოვითვალოთ, თუ რომელი გვერდები შეიძლება იქნეს გამოყენებული შემდეგ ჯერზე, და წინასწარ ჩაიტვირთოს მათთვის ჩანაწერები TLB ბუფერში. მაგ. როცა კლიენტის პროცესი აგზავნის შეტყობინებას სერვერის პროცესთან იმავე მანქანაზე, ძალზე საიმედოა, რომ სერვერმა მალევე დაიწყოს მუშაობა. იცის , რა, „ ეს ამბავი სისტემამ“, მან შეიძლება გადაამოწმოს სად იმყოფება გვერდები სერვერის კოდის, მონაცემების, შერწყმის, ვიდრე წყვეტა დამუშავდებოდეს, მანამდე. ეს საჭიროა იმიტომ, რომ განხორციელდეს Send გამოძახება და გამოისახოს მათი მისამართები ვირტუალურიდან ფიზიკურებში, მანამდე, სანამ ისინი შეიძლება გახდნენ შეცდომის მიზეზები TLB ბუფერისათვის.

TLB ბუფერის წარუმატებელი ძიების დამუშავების ჩვეულებრივი გზა აპარატურულად ან პროგრამულად, ესაა, - გადასვლა გვერდების ცხრილში და ინდექსაციის ოპერაციის შესრულება, რომ განისაზღვროს გვერდების მდებარეობა, რომელზედაც ხდება მიმართვა. ამ ძიებების განსახორციელებლად პროგრამულად ჩნდება პრობლემა, რომელიც იმაში მდგომარეობს, რომ გვერდები, რომელიც ფურცლის შინაარსს შეიცავენ, შეიძლება არ აღმოჩნდნენ ბუფერში, რაც გამოიწვევს დამატებით შეცდომებს TLB ბუფერში დამუშავების დროს. მათი რაოდენობა შეიძლება შევამციროთ, თუკი დავიჭერთ დიდ(მაგ.4გბ) პროგრამულ ქეშს, TLB ჩანაწერისათვის მეხსიერებაში ფიქსირებულად განთავსებისათვის, რომლის გვერდები ყოველთვის ინახება ბუფერში. თუკი თავიდანვე შევამცირებთ პროგრამულ ქეშს, ოპერაციულმა სისტემამ შეიძლება მნიშვნელოვნად შეამციროს TLB ბუფერში წარუმატებელი ძიებების რაოდენობა.

გვერდების ცხრილის ინვერტირება

ტრადიციული გვერდების ცხრილი, რომელთა ტიპები ჩვენ აქამდე აღვწერეთ, თხოულობენ თითო-თითო ჩანაწერს ყოველ ვირტუალურ გვერდზე. რადგან ისინი ამ გვერდის ნომრის მიხედვით ინდექსირდებიან. თუკი მისამართების სივრცე შედგება 2^{32} ბაიტისაგან, და გვერდების მოცულობისაგან 2^{12} =4096 ბაიტი, მაშინ გვერდების ცხრილში უნდა იყოს მილიონზე მეტი ჩანაწერი. ამის გამო გვერდების ცხრილმა უნდა დაიკავოს მინიმუმ 4 გბ. საკმარისად დიდ სისტემებში ეს, ცხადია, შესაძლებელია. მაგრამ , რადგან 64 თანრიგიანი კომპიუტერები ყველაზე უფრო ხშირად გვხვდება, ამიტომ სიტუაცია რადიკალურად იცვლება. თუკი ახლა მისამართების სივრცე გაიზრდება 2^{64} ბაიტამდე, ცხრილის ზომით 4 მბ, მაშინ ჩვენ დაგვჭირდება ცხრილი 2^{52} ჩანაწერით. თუკი ყველა ჩანაწერი ტოლია 8 ბაიტისა, მაშინ ცხრილი დაიკავებს 30 ტერაბაიტამდე და მეტ მოცულობას. 30 ტბ გამოიყოფა მხოლოდ გვერდების ცხრილებისათვის. დღეისათვის არარეალურია და არ იქნება რეალური არც ოდესმე, ამგვარად, 64 თანრიგიანი ფურცვლოვანი ვირტუალური სივრცისათვის საჭიროა სხვაგვარი გადაწყვეტა.

ერთერთი ასეთი გადაწყვეტაა გვერდების ინვერტირებული ცხრილი. ამ მოდელში ცხრილი შეიცავს თითო ჩანაწერს რეალურ მეხსიერებაში ფურცვლოვან ბლოკში და არა გვერდზე, - ვირტუალური მისამართების სივრცეში. მაგ. 64 თანრიგიან ვირტუალურ მისამართებში გვერდის სიდიდისათვის 4კბ 256 მბ ოპ. მეხსიერების მქონე ინვერტირებული ცხრილი მოითხოვს მხოლოდ 65536 ჩანაწერს. თითოეული ჩანაწერი „აკვირდება“ - არის თუ არა, პროცესი(ვირტუალური გვერდი), მოთავსებული მოცემული გვერდების ბლოკში.

თუმცა ინვერტირებული ცხრილები ახდენენ ადგილების მნიშვნელოვან ეკონომიას, უკიდურეს შემთხვევაში, როცა ვირტუალური სამისამართო სივრცე მნიშვნელოვნად მეტია, ვიდრე ფიზიკური მეხსიერება, - მათ აქვთ მაინც სერიოზული ხარვეზი, უკმარისობა: ვირტუალური მისამართის გადაყვანა ფიზიკურში ხდება მნიშვნელოვნად რთული.

როდესაც პროცესი n მიმართავს ვირტუალურ გვერდს w , მაშინ აპარატურულ უზრუნველყოფას უკვე აღარ შეუძლია იპოვოს ფიზიკური გვერდი, რომელიც ინდექსის სახით გვერდების ცხრილში იყენებს ნომერ p - ს. ამის ნაცვლად მან უნდა მოახდინოს (n×p) ჩანაწერის ძებნა მთელს გვერდების ინვერტირებულ ცხრილში. უფრო მეტიც, ეს ძებნა უნდა შესრულდეს მეხსიერებასთან ყოველი მიმართვის დროს. და არა მარტო ფურცვლოვანი წყვეტების დროს. ძებნის ოპერაცია ცხრილში 64კბ მოცულობაში, ყოველი გზავნილების დროს, ზრდის კომპიუტერის სიჩქარეს.

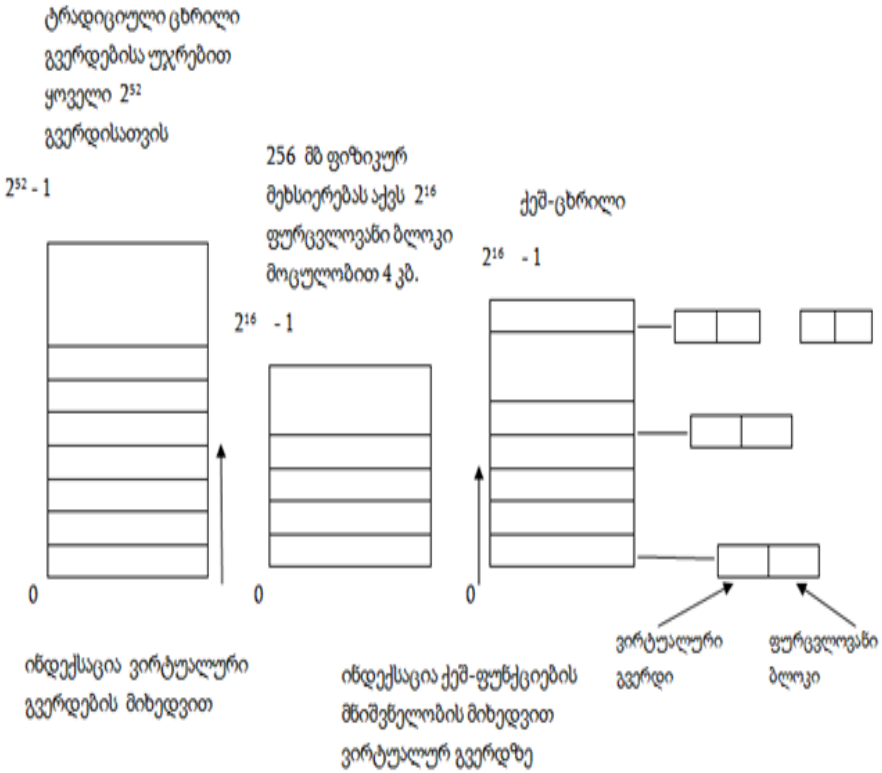
ამ გართულებული მდგომარეობიდან გამოსავალი არსებობს TLB ბუფერის გამოყენების გზით. თუკი ამ ბუფერს შეუძლია შეიცავდეს ყველა ხშირად გამოყენებად გვერდს, მაშინ მისამართების ტრანსლაცია მოხდება ისევე სწრაფად, როგორც ჩვეულებრივ გვერდების ცხრილებში. მაგრამ წარუმატებელი ძებნის დროს, TLB -ში გვერდების ინვერტირებულ ცხრილში ძებნა უნდა განხორციელდეს პროგრამულად. წესებიდან ერთერთი შესაძლო, ეს რომ სრულყოფილი გახდეს, არის ის, რომ დავუჭიროთ მხარი ვირტუალური მისამართების ქეშ-ცხრილს. ყველა ვირტუალური მისამართი, რომელიც მოც. მომენტში მეხსიერებაში იმყოფება, და აქვთ ქეშ-ფუნქციების ერთნაირი მნიშვნელობები, ეჯაჭვებიან ერთი-მეორეს, როგორც ეს

ნაჩვენებია ნახ.21 -ზე. თუკი ქეშ-ცხრილი შედგება მხოლოდ იგივე რაოდენობის უჯრებისაგან, როგორც ფიზიკური მისამართების კომპიუტერშია, მაშინ საშუალო ჯაჭვი იქნება გრძელი მხოლოდ ერთ ჩანაწერში. რაც მნიშვნელოვნად გაზრდის მისამართების გარდასახვას, როგორც კი მოიძებნება გვერდების ბლოკის ნომერი, მაშინვე ახალი წყვილი(ვირტუალური, ფიზიკური) მოთავსდება ბუფერში.

ინვერტირებული ცხრილებიანი გვერდები დღეისათვის გამოიყენება IBM კომპანიის Newletf Packard სხვადასხვა სამუშაო სადგურებში და გვხვდება სულ უფრო და უფრო ხშირად, რადგან 64-თანრიგიანი მანქანები პოულობენ ამჟამად ყველაზე მეტ გავრცელებას.

გვერდების განთავსების ალგორითმი

როცა ფურცვლოვანი წყვეტა ხდება, მაშინ ოპერაციულმა სისტემამ უნდა შეარჩიოს ფურცელი(გვერდი) მეხსიერებიდან მოსაცილებლად, რათა გაანთავისუფლოს ადგილი იმ გვერდისათვის, რომელიც უნდა გადატანილ იქნას მეხსიერებაში. თუკი მოსაცილებელი გვერდი შეიცვალა მისი მეხსიერებაში ყოფნის დროს, მაშინ ის აუცილებელია გადაიწეროს დისკზე, რომ განაახლოს ასლი, რომელიც იქ ინახება. მაგრამ, თუკი გვერდი არ მოდიფიცირებულა, (მაგ. ის შეადგენს პროგრამის ტექსტს), მაშინ ასლი დისკზე უკვე არსებობს სულ მთლად ახალი და მისი გადაწერა საჭირო აღარაა. მაშინ გვერდი, რომელიც უნდა წავიკითხოთ, უბრალოდ მიითვლება ჩატვირთული გვერდის ზემოთ.



ნახ. 21. ინვერტირებულთან ტრადიციული გვერდების ცხრილის შედარება

თუმცა, პრინციპში, შეიძლება ყოველი ფურცლოვანი წყვეტის დროს შევარჩიოთ შემთხვევითი გვერდი, რომლის მეხსიერებიდან მოცილება შეიძლება, მაგრამ, სისტემის მწარმოებლობა მნიშვნელოვნად მაღლდება როცა უპირატესობას მივანიჭებთ

იშვიათად გამოყენებულ გვერდს. თუკი განიტვირთება გვერდი, რომელზე მიმართვაც ხშირად ხდება, მაშინ დიდია ალბათობა იმისა, რომ მალე ისევ მოითხოვება მისი მობრუნება მეხსიერებაში, რაც შედეგად იძლევა დამატებით დანახარჯებს.

ფურცლების ცვლილების ასეთად დამუშავებული ალგორითმები მოითხოვენ დიდ შრომას როგორც თეორიული ასევე ექსპერიმენტალური თვალსაზრისით. ქვემოთ სხვა მნიშვნელოვან ალგორითმებსაც განვიხილავთ.

უნდა აღინიშნოს, რომ „ფურცვლოვანი შენაცვლების“ პრობლემა ასევე დღის წესრიგში დგას კომპიუტერების კონსტრუირების სხვა ასპექტებშიც. მაგ. მათ უმრავლესობას აქვთ ერთი ან რამდენიმე ქეშ-მეხსიერება, რომელიც შედგება ბოლო დროს გამოყენებული 32 ან 64 ბაიტიანი მეხსიერების ბლოკებისაგან. როცა ქეშები შევსებულია, მაშინ აუცილებელია რამდენიმე ბლოკი შეირჩეს მოსაცილებლად. ეს პრობლემა პრაქტიკულად გვერდების შენაცვლების ანალოგიურია. მხოლოდ იმ ერთი განსხვავებით, რომელიც დროის მცირე მასშტაბში მდგომარეობს. (ოპერაცია უნდა შესრულდეს ნანოწამებში, ან მილიწამებში, როგორც ფურცლების ცვლილების დროს). დროის ინტერვალის უფრო ხანმოკლეობის მიზეზი არის ის, რომ წარუმატებლობა იქნება ბლოკის ქეშში, - დამუშავდება ძირითადი მეხსიერებიდან, რომელშიც არ იხარჯება დრო დისკის საჭირო ცილინდრის მოსაძებნად და ადგილი არ აქვს შეფერხებებს მისი ბრუნვის გამო.

მეორე მაგალითი გვხვდება ვებ-სერვერზე. სერვერი უნდა ინახავდეს განსაზღვრული რაოდენობის ხშირად გამოყენებად ვებ-გვერდებს თავის ქეშ-მეხსიერებაში. მაგრამ როცა ქეშ-მეხსიერება შევსებულია მთლიანად, და ხდება მიმართვა ახალ

გვერდზე, მაშინ აუცილებელია გადაწყვეტილება მივიღოთ იმისათვის, თუ რომელი გვერდი განვტვირთოთ. აქ მისაღებია იგივე მსჯელობები, რაც გვერდებისათვის ვირტუალურ მეხსიერებაში. იმ განსხვავებით, რომ ვებ-გვერდები ამ დროს არ იცვლებიან ქეშში, ამიტომ მათთვის ყოველთვის არის ახალი ადგილი დისკზე. ვირტუალური მეხსიერების სისტემაში ოპერატიული მეხსიერების გვერდები შეიძლება იყოს „სუფთაც“ და „ჭუჭყიანიც“.

ოპტიმალური ალგორითმი

გვერდების შენაცვლების შესაძლო ალგორითმებს შორის ყველაზე კარგი ალგორითმი ადვილად აღიწერება, მაგრამ შეუძლებელია განხორციელება. ის ასე მოქმედებს: იმ მომენტში, როცა ფურცვლოვანი წყვეტა ხდება, მაშინ მეხსიერებაში იმყოფება გვერდების რაღაც ერთობლიობა. ამ გვერდებიდან ერთერთზე მოხდება პროცესორის შემდგომი ბრძანების მიმართვა(გვერდზე, რომელიც შეიცავს საჭირო ბრძანებას). სხვა გვერდებზე, შესაძლოა არ იყოს მიმართვები მომდევნო 10, 100 1000 ბრძანების განმავლობაშიც კი. თითოეული გვერდი შეიძლება მონიშნული იყოს ბრძანებების რაოდენობით, რომლებიც შესრულდებიან ამ გვერდზე პირველი მიმართვის წინ.

ოპტიმალური ფურცვლოვანი ალგორითმი უბრალოდ იუწყება, რომ უნდა განიტვირთოს უდიდესი ჭდეს მქონე გვერდი. თუკი ერთი გვერდი არ იქნება გამოყენებული 8 მლნ ბრძანების განმავლობაში, ხოლო მეორე - 6 მლნ ბრძანების (ინსტრუქციის) განმავლობაში, მაშინ პირველის მოცილება გააადგილებს მომავლისათვის შესაძლო ფურცვლოვანი წყვეტის მაქსიმალურ

ვადას, რომელიც დააბრუნებს მას უკან. კომპიუტერები, ადამიანების მსგავსად, ცდილობენ, არასასურველი მოვლენები „გადადონ“ რაც შეიძლება შორს და შორს.

ამ ალგორითმებთან დაკავშირებულია მხოლოდ ერთი პრობლემა. ის შეუსრულებადია. ფურცვლოვანი წყვეტის მომენტში ოპერაციულ სისტემას არ აქვს შესაძლებლობა, გაიგოს როდის მოხდება შემდგომი მიმართვა თითოეულ გვერდზე. (ჩვენ განვიხილეთ ანალოგიური სიტუაცია ადრე, როცა განვაალიზებდით დაგეგმვის ალგორითმს - „უმოკლესი გზა - პირველად“. როგორ შეუძლია სისტემას თქვას, რომელია ამოცანებს შორის ყველაზე მოკლე? მით უმეტეს, ვასრულებთ რა, პროგრამას მოდელზე და თვალს ვადევნებთ გვერდებისადმი ყველა მიმართვას, ოპტიმალური შეცვლა შეიძლება განვახორციელოთ, მეორე გაშვებაზე, თუკი გამოვიყენებთ ინფორმაციას მიმართვის შესახებ, რომელიც მოვიპოვეთ ფურცელზე პირველი მიმართვის დროს.

ამ შემთხვევაში შეიძლება შევადაროთ რეალიზებადი ალგორითმების მწარმოებლობა საუკეთესოს. თუკი ოპერაციული სისტემა აღწევს მწარმოებლობას, ვთქვათ, მხოლოდ 1%-ით დაბალს, ვიდრე ოპტიმალური ალგორითმით მუშაობის დროს, მაშინ ძალისხმევა, რაც იხარჯება კარგი ალგორითმის ძიებაზე, აამაღლებს სქემის ნაყოფიერებას მაქსიმუმ 1% -ით.

შესაძლო გაუგებრობებს რომ თავი გავართვათ, საჭიროა მკაფიოდ ვაჩვენოთ, რომ გვერდებზე მიმართვის პროტოკოლი მიეკუთვნება მხოლოდ ერთს კარგად დაგეგმილი პროგრამებიდან და, გარდა ამისა, აგრეთვე, განსაზღვრულ საწყის მონაცემებს.

ამგვარად, აქედან გამომდინარე, გვერდების შენაცვლების

ალგორითმი, იქნება დამახასიათებელი მხოლოდ ამ პროგრამისათვის და სახელდობრ ამ საწყისი მონაცემებით. თუმცა ეს მეთოდი სასარგებლოა გვერდების შენაცვლების ალგორითმის შესაფასებლად, ის მაინც ნაკლებად გამოიყენება პრაქტიკულ სისტემეში. ქვემოთ ჩვენ განვიხილავთ ალგორითმებს, რომლებიც მისაღებია რეალური სისტემებისათვის.

ალგორითმი FIFO -პირველი მოვიდა, პირველს მოემსახურენ. Firrs -In, First – Out. - რომ გავაანალიზოთ ეს სამუშაო, განვიხილოთ უნივერსალური მაღაზია, რომლის თაროებზეც შეგვიძლია განვათავსოთ ზუსტად k სხვადასხვა პროდუქტი. ის გვთავაზობს ახალ მოხერხებულ საკვებს, - ხსნად, კარგად გაყინულ, ეკოლოგიურად სუფთა იოგურტს, რომელიც შეიძლება წამიერად მომზადდეს მიკროტალღოვან ღუმელში. მომხმარებლები მაშინვე აქცევენ ამ პროდუქტს ყურადღებას. ამიტომ, ჩვენი, სავიტრინო ფართით შეზღუდული სუპერმარკეტი, იმისათვის რომ გაეყიდოს იოგურტი, იძულებულია უარი თქვას ერთერთ დაძველებულ (კი არა, დიდი ხანია რომ ჰყიდის, იმ) საქონელზე.

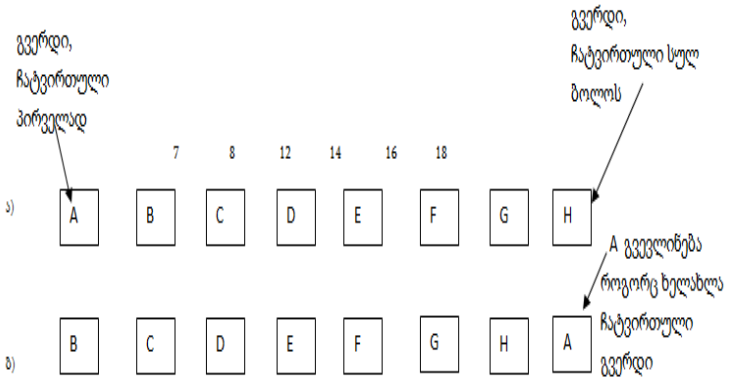
ერთერთი ვარიანტი ისაა, რომ იპოვოს პროდუქტი, რომელსაც მარკეტი ყიდის ძალიან დიდი ხნის განმავლობაში. ვთქვათ, 120 წელია, რაც იყიდება. და გაანთავისუფლოს მაღაზია იმის საფუძველზე, რომ ამ საქონლით უკვე აღარავინ ინტერესდება. რეალურად, სუპერმარკეტი ინახავს ყველა იმ საქონლის ჩამონათვალს, რაც დღეს აქვს სახეზე. - მისი გაყიდვაში გამოჩენის დღიდან. ყოველი ახალი პროდუქტი თავსდება ჩამონათვალის ბოლოში, და სიის თავიდან მოშორდება ერთერთი ძველი

დასახელება.

იგივე იდეა დევს გვერდების შენაცვლების ალგორითმში. ოპერაციული სისტემა იჭერს ყველა გვერდის სიას, რომლებიც იმყოფებიან მოცემულ მომენტში მეხსიერებაში, რომელშიც პირველი გვერდი წარმოადგენს უფროს გვერდს, ხოლო სიის ბოლოში წერია გვერდები, რომლებიც სულ ახლახანს მოხვდა. როცა ხდება ფურცვლოვანი წყვეტა, მაშინ მეხსიერებიდან წაიშლება ის გვერდი, რომელიც სიის თავშია, ხოლო ახალი გვერდი ემატება მას მხოლოდ ბოლოში. თუკი ალგორითმს FiFo ვიყენებთ მაღაზიაში, მაშინ მან შეიძლება მოიცილოს ფქვილიც, მარილიც, კარაქიც. ზუსტად ასევე, კომპ-ებსაც უჩნდებათ ასეთი პრობლემა. ამიტომ, ამ მიზეზით FiFo ალგორითმი იშვიათად გამოიყენება თავისი საწყისი ფორმით.

ალგორითმი „მეორე ცდა“

მარტივ ვარიანტში FiFo, რომელიც საშუალებას იძლევა, გამოვდევნოთ მეხსიერებიდან ყველაზე ხშირად გამოყენებული გვერდი, აქ ყველაზე ძველ გვერდად შეისწავლება ბიტი R თუკი ის ტოლია 0- ის, მაშინ გვერდი არა თუ იმყოფება მეხსიერებაში დიდხანს, ის, მითუმეტეს, არც კი გამოიყენება, ამიტომ დაუყოვნებლივ უნდა შეიცვალოს ახლით. თუკი ბიტი R=1 , მაშინ მას მიენიჭება მნიშვნელობა 0, გვერდი გადაიტანება სიის ბოლოში, ხოლო მისი დატვირთვის დრო განახლდება, ანუ ითვლება რომ, გვერდი ეს-ესაა მოხვდა მეხსიერებაში. შემდეგ პროცედურა განახლდება. ამ ალგორითმის მუშაობას ეწოდება „მეორე ცდა“ (ნახ.22 ა).



ნახ.22 „მეორე ცდა“

აქ გამოსახულია გვერდები A - H. რომლებიც ინახება დაკავშირებულ სიებში. და დახარისხებულია მათი მეხსიერებაში მოხვედრის დროის მიხედვით. გვერდებზე წარწერილი რიცხვები აღნიშნავენ მათი მეხსიერებაში მოხვედრის დროს.

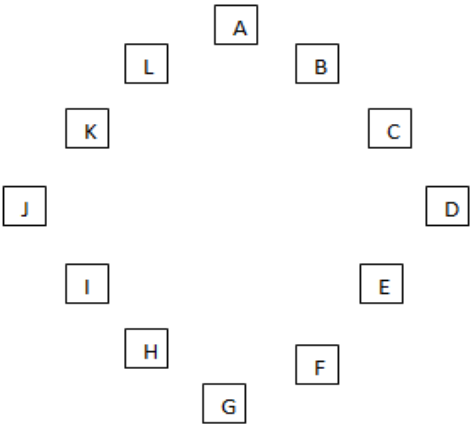
დავუშვათ, რომ დროის მომენტში „20“ ხდება ფურცვლოვანი წყვეტა. ყველაზე ძველი ფურცელია A. ის ჩაიტვირთება მეხსიერებაში 0 - ის დროს. როცა დაიწყო პროცესი. თუკი R ბიტი A გვერდისათვის 0 -ის ტოლია, მაშინ ის განიტვირთება, მეხსიერებიდან, ან დისკზე გადაწერის გზით. (თუკი გვერდი „ჭუჭყიანია“) ანდა უბრალოდ, მოცილდება, (თუკი ის „სუფთაა“). მეორე შემთხვევაში, თუკი R ბიტი =1 , მაშინ A გვერდი გადაადგილდება სიის ბოლოში, ხოლო „ჩატვირთვის დრო“ ღებულობს მიმდინარე მნიშვნელობას. („20“). ამ დროს R სუფთავდება ანუ დაყენდება ნულზე. შესაფერისი გვერდის ძიება გრძელდება. შემდეგ შემოწმდება B გვერდი.

ალგორითმი **„მეორე ცდა“** ეძებს სიაში ყველაზე ძველ გვერდს, რომელზეც არ იყო მიმართვა განხორციელებული დროის წინა ინტერვალში. თუკი განხორციელდა მიმართვები ყველა გვერდზე, მაშინ „მეორე ცდა“ გარდაიქმნება იგივე ძირითად „FiFo“ ალგორითმად. წარმოვიდგინოთ, რომ ყველა გვერდს ნახ.22 ა -ზე R ბიტი =1 . ერთი მეორის მიყოლებით გადაადგილებს ოპერაციული სისტემა გვერდებს სიის ბოლოში. და ამავე დროს, ასუფთავებს R ყოველ ჯერზე, როცა გვერდებს კიდისკენ აადგილებს. ბოლოს ის უბრუნდება A გვერდს. მაგრამ უკვე მის R ბიტს მინიჭებული აქვს 0-ანი მნიშვნელობა. ამ მომენტში A გვერდი გამოიტვირთება მეხსიერებიდან. ამგვარად, ალგორითმი ყოველთვის წარმატებით ასრულებს თავის სამუშაოს.

ალგორითმი საათი

თუმცა „მეორე ცდა“ ალგორითმი კორექტულია, მაგრამ ის ძალიან არაეფექტურია, რადგან გამუდმებით გადააადგილებს გვერდებს სიის მიხედვით. ამიტომ უკეთესია შენახული გვქონდეს ყველა გვერდების ბლოკები წრიულ სიაში საათის სახით. როგორც ეს ნაჩვენებია 23ე ნახაზზე. ისარი მიანიშნებს ყველაზე ძველ გვერდს. როცა ხდება ეს ფურცვლოვნი წყვეტა, შემოწმდება ის გვერდი, რომელზედაც მიმართულია ისარი, თუ მისი R= 0, მაშინ გვერდი გამოიტვირთება. მის ადგილზე საათის წრეში დადგება ახალი გვერდი, ხოლო ისარი გადაადგილდება წინ 1 პოზიციაზე. თუკი R= 1, მაშინ ეცემა, ხოლო ისარი გადაადგილდება ახალ გვერდზე. ეს პროცესი მეორდება მანამ, სანამ არ მოიძებნება ის გვერდი, რომლის R= 0 . მართლაც საათივითაა. მეორე ცდის ალგორითმისაგან მხოლოდ რეალიზაციის ფორმით განსხვავდება.

როცა ხდება ფურცვლოვანი წყვეტა, მოწმდება გვერდი, რომელსაც მიუთითებს ისარი. წინასწარი მოქმედებები დამოკიდებულია R ბიტზე. R= 0 გვერდი გამოტვირთულია, R= 1 ისარი წინ მიიწევს.



ნახ.23. ალგორითმი რომელიც გვერდებს საათისებურად აადგილებს

ალგორითმი LRU - Least Recently Used გვერდი, რომელიც ყველაზე დიდი ხნის გამოუყენებელია

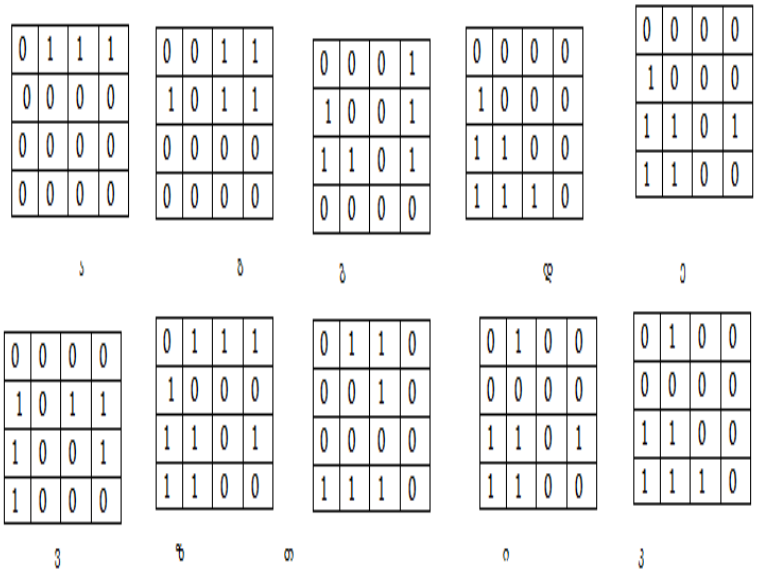
ამ ოპტიმალური ალგორითმის აპროქსიმაციისადმი მიდგომაში დევს დაკვირვება იმის შესახებ, რომ გვერდები, რომელზედაც განხორციელდა მრავალჯერადი მიმართვები, რამდენიმე ბოლო ბრძანებაში, სავარაუდოა, რომ ასევე ხშირად გამოყენება მოუწევთ მომდევნო ინსტრუქციებშიც. და პირიქით.

შეიძლება ვივარაუდოთ, რომ გვერდი, რომელზეც ადრე არ აღიძვრა გამოძახება, ასევე არ იქნება გამოყენებული ხანგრძლივი დროის განმავლობაში. მომავალშიც. ამ იდეამ მიგვიყვანა შემდეგი ალგორითმის რეალიზაციამდე. როცა ხდება გვერდების წყვეტა, - მეხსიერებიდან გამოიტვირთება ის გვერდი, რომელიც ყველაზე დიდი ხნის გამოუყენებელია. ამას ეწოდება LRU - Least Recently Used. ნაკლებად ახლახანს. ანუ დიდი ხნის წინ. თუმცა ალგორითმი LRUთეორიულად რეალიზებადია, მაგრამ არ წარმოადგენს იაფს. მისი სრული განხორციელებისათვის საჭიროა გვქონდეს კავშირების სია მეხსიერებაში არსებული ყველა გვერდისა, სადაც უკანასკნელ ხანს გამოყენებული გვერდი იმყოფება სიის თავში, ხოლო დიდი ხნის გამოუყენებელი - ბოლოში. სირთულე იმაში მდგომარეობს, რომ სია უნდა განახლდეს მეხსიერებაზე ყოველი მიმართვის დროს. გვერდების ძიება, მათი მოშორება, შემდეგ ჩასმა სიის თავში - ეს ოპერაციებია, რომლებიც დიდ დროს შთანთქავს, თუნდაც აპარატურულად სრულდებოდნენ. (იგულისხმება, რომ საჭირო მოწყობილობების კონსტრუქციები შესაძლებელია).

მაგრამ, არსებობენ LRU რეალიზაციის სხვა ალგორითმებიც, სპეციალური აღჭურვილობის მეშვეობით. პირველი მეთოდისათვის საჭიროა კომპ-ი დაფუძვნებული იყოს 64 თანრიგიან აპარატურულ C მთვლელზე, რომელიც ავტომატურად იზრდება ყოველი ბრძანების შემდეგ. გარდა ამისა, გვერდების ცხრილში ყველა ჩანაწერს უნდა ჰქონდეს საკმარისი მოცულობის ველი, რომელიც შეინახავს მთვლელის მნიშვნელობებს, მეხსიერებაზე ყოველი მიმართვის დროს, C მთვლელის მიმდინარე მნიშვნელობა, დაიმახსოვრება ცხრილის ჩანაწერში, რომელიც შეესაბამება იმ გვერდს, რომელზეც

განხორციელდა მიმართვა. ხოლო თუკი ხდება გვერდების წყვეტა, მაშინ ოპერაციული სისტემა ამოწმებს მთვლელის ყველა მნიშვნელობას გვერდების ცხრილში და ეძებს უმცირესს, - სწორედ ესაა ყველაზე გამოუყენებელი.

ახლა განვიხილოთ LRU აპარატურული რეალიზაციის მეორე ვარიანტი. კომპიუტერზე, რომელსაც აქვს n გვერდების ბლოკები, LRU – s აღჭურვილობამ შეიძლება დაიკავოს n*n ბიტი, რომლებიც თავიდან იქნებიან ნულები. ყოველ ჯერზე, როცა ხდება მიმართვა, გვერდების K-ურ ბლოკზე, აპარატურა პირველად ანიჭებს K სტრიქონის ყველა ბიტს „1“ მნიშვნელობას. შემდეგ უტოლებს ნულს ყველა ბიტს K სვეტისას.



ნახ.24. გვერდოვანი ბლოკის სხემა

დროის ნებისმიერ მომენტში, სტრიქონი, რომლის ოროზითი მნიშვნელობა უმცირესია, ყველაზე მეტად გამოუყენებელს წარმოადგენს. ნახ.24 განიხილება 4გვერდოვანი ბლოკი და შემდგომი რიგითობა მიემართება გვერდზე ასე: 0123210323

როდესაც მიმართვა მოხდება 0 გვერდზე, იმის შემდეგ ჩვენ მივიღებთ სისტემას, სიტუაციას, 24ა - ს. 1 გვერდზე მიმართვის შემდეგ 24.ბ. და ა.შ. LRU მატრიცას იყენებს სადაც გვერდებზე მიმართვა ხდება თანმიმდევრობით.

ინფორმაციის დაცულობა თანამედროვე კომპიუტერულ ქსელებში

კომპიუტერულ ტექნიკაში უსაფრთხოება ძალზე ფართო ცნებაა. ის კომპიუტერის მუშაობის საიმედობასაც გულისხმობს, ფასეული მონაცემების შენახვასაც, ინფორმაციის დაცვასაც გარეშე პირების მიერ ცვლილებების შეტანისაგან, მიმოწერის საიდუმლოების შენახვასაც ელექტრონული კავშირების დროს და ა.შ. საგულისხმოა, რომ ყველა ცივილიზებულ ქვეყანაში მოსახლეობის უსაფრთხოებას იცავს კანონები, მაგრამ კომპიუტერიზაციის სფეროში სამართალ შეფარდებითი პრაქტიკა ჯერ კიდევ სრულყოფილად განვითარებული არაა, ხოლო კანონშემოქმედებითი პროცესი წინ ვერ უსწრებს კომპიუტერული სისტემების განვითარებას და ბევრ შემთხვევაში თვითდაცვას ეყრდნობა. ყოველთვის არსებობს არჩევანის პრობლემა დაცულობის აუცილებელ დონესა და

ქსელის მუშაობის ეფექტურობას შორის. ზოგჯერ, უსაფრთხოების უზრუნველსაყოფად, მწარმოებლის ან მომხმარებლის ქმედებანი შეიძლება შეღწევაზე შეზღუდვისა და ეფექტურობის ზომებად შეფასდეს. მაგრამ, ისეთი საშუალებები, როგორიცაა მაგალითად, კრიპტოგრაფია, შესაძლებელს ხდის გაძლიერდეს დაცვის ხარისხი, ისე რომ არ შეიზღუდოს მომხმარებლის შესვლა მონაცემებზე. ინფორმაციის დამუშავებისა და მართვის ავტომატიზებულ სისტემებში კომპიუტერული ტექნოლოგიების ფართო გამოყენებამ გაამძაფრა პრობლემები. კომპიუტერულ სისტემებში ინფორმაციის დაცვა ხასიათდება რიგი სპეციფიკური თავისებურებებით, რომლებსაც აკავშირებთ ის, რომ ინფორმაცია არ წარმოადგენს მის მატარებელთან მყარად დაკავშირებულ სუბსტანციას. შეიძლება მისი ასლის სწრაფად გადაღება და კავშირის არხებით გადაცემა. ცნობილია ინფორმაციისათვის საფრთხეთა დიდი რიცხვი, რომელიც შეიძლება განხორციელდეს როგორც გარეგანი, ასევე შინაგანი დარღვევების გზით.
ელექტრონული ინფორმაციის დაცვის პრობლემის რადიკალური გადაწყვეტა შეიძლება მივიღოთ მხოლოდ კრიპტოგრაფიკული მეთოდების გამოყენების საფუძველზე, რომლებიც დაცული ავტომატიზებული დამუშავებისა და მონაცემთა გადაცემის მნიშვნელოვანი პრობლემების გადაწყვეტის საშუალებას გვაძლევს. ამასთან კრიპტოგრაფიკული გარდაქმნების თანამედროვე ჩქაროსნული მეთოდები საშუალებას გვაძლევს შევინარჩუნოთ ავტომატიზებული სისტემების საწყისი მწარმოებლობა. ისინი წარმოადგენენ მონაცემთა კონფიდენციალურობის, მათი მიზნობრივობისა და უტყუარობის ყველაზე ეფექტურ საშუალებას. აუცილებელ

ტექნიკურ და ორგანიზაციულ ღონისძიებებთან ერთად მხოლოდ მათმა გამოყენებამ შეიძლება უზრუნველყოს პოტენციალური საშიშროებების ფართო სპექტრისაგან დაცვა.
დაშიფვრის სისტემა იმდენივე წლისაა, რამდენისაც ინფორმაციის წერილობითი გადაცემა.
‘‘კრიპტოგრაფია’’ ბერძნულიდან გადმოთარგმნით ნიშნავს ‘‘ფარულობას’’, რაც სრულად ასახავს მის პირველსაწყის დანიშნულებას. პრიმიტიული(თანამედროვე თვალსაზრისით) კრიპტოგრაფიული მეთოდები ცნობილი იყო უძველესი დროიდან და ხანგრძლივი პერიოდის მანძილზე განიხილებოდა როგორც ერთგვარი ეშმაკობა, და არა მკაცრი მეცნიერული დისციპლინა. კრიპტოგრაფიის კლასიკურ ამოცანას წარმოადგენს მოცემული ცხადი, გასაგები საწყისი ტექსტის შექცეული ასახვა რაღაც ნიშნების შემთხვევითი, მოჩვენებითი მიმდევრობის სახით, რომელსაც შიფრტექსტს ანუ კრიპტოგრამას უწოდებენ. აუცილებელ მოთხოვნას წარმოადგენს ის, რომ შიფრტექსტში რაღაც სიმბოლოების ლოგიკური ცვლილებების გამოყენებით უნდა ცალსახად და სრული მოცულობით მოხდეს საწყისი ტექსტის აღდგენა. ინფორმაციის შენახვის საიმედობა უძველეს დროში ფარულად განისაზღვრებოდა იმით რომ, საიდუმლოდ იყო დაცული თავად გარდასახვის მეთოდი. ცხადია, დაშიფრული გზავნილების ინტერპრეტაცია მხოლოდ გასაღების დახმარებით შეიძლება.
კომპიუტერულ ქსელებში მუშაობისას ინფორმაციის გადაცემის უსაფრთხოების თვალსაზრისით თავს იჩენს პრობლემები, რომლებიც შეიძლება სამ ჯგუფად დავყოთ: ინფორმაციის ხელში ჩაგდება, მოტაცება – ამ დროს

ინფორმციის მთლიანობა შენარჩუნებულია, მაგრამ მისი კონფიდენციალურობა ირღვევა;
ინფორმაციის მოდიფიცირება – საწყისი ინფორმაცია (შეტყობინება) იცვლება ან სრულიად შენაცვლდება სხვა ინფორმაციით და ასე გადაეგზავნება ადრესატს;

ავტორობის ცვლილებები. ამ პრობლემას შეიძლება სერიოზული გართულებები მოჰყვეს. მაგალითად, ვინმემ შეიძლება თქვენი სახელით გაგზავნოს წერილი (სპუფინგი) ანდა ვებ-სერვერი იქცეს ელექტრონულ მაღაზიად, მიიღოს შეკვეთები, საკრედიტო ბარათების ნომრები, მაგრამ ნაცვლად არავითარი საქონელი არ გაგზავნოს. თანამედროვე პრქატიკული ინფორმატიკის მოთხოვნილებებმა მიგვიყვანა ელექტრონული ინფორმაციის დაცვის არატრადიციული ამოცანების გაჩენამდე. რომელთა შორის აღსანიშნავია ელექტრონული ინფორმაციის აუტენტიფიკაცია, რომლის დროსაც ინფორმაციის გამცვლელი მხარეები არ ენდობიან ერთმანეთს. ეს პრობლემა დაკავშირებულია ელექტრონული ციფრული ხელმოწერის სისტემის შექმნასთან. რომლის თეორიული ბაზა ორგასაღებიან კრიპტოგრაფიის სახელწოდებით დამუშავებულია ამერიკელი დიფფისა და ხემიმანის მიერ და რევოლუციურ იდეადაა მიჩნეული. რომელმაც განვითარების ახალი გზები და შესაძლებლობები დასახა და უნიკალურია მისი მეთოდების მნიშვნელობა ელექტრონული ინფორმაციული ტექნოლოგიებისათვის მასობრივი გამოყენების თანამედროვე პირობებში. აუტენტიფიკაცია ქსელში ინფორმაციის დაცვის ორგანიზაციის ერთერთ ყველაზე მნიშვნელოვან კომპონენტს წარმოადგენს. ვიდრე მომხმარებელი ამა თუ იმ რესურსის მოპოვების უფლება მიეცემა, აუცილებელია

დარწმუნდეს,რომ ის სწორედ ისაა, რომელსაც ეძიებდა. ამა თუ იმ მომხმარებლის სახელზე რესურსის გამოყენების შესახებ მოთხოვნის მიღებისთანავე, სერვერი, რომელსაც მიეწოდება ეს რესურსი, მართვას გადასცემს აუტენტიფიკაციის სერვერს. ამ სერვერისაგან დადებითი პასუხის მიღების შემთხვევაში მომხმარებელს მიეწოდება შეკვეთილი რესურსი.

აუტენტიფიკაციის დროს, როგორც წესი, გამოიყენება წესი, რომელსაც ეწოდება – ‘‘რა იცის მან’’ – მომხმარებელმა იცის რომელიღაც საიდუმლო სიტყვა, რომელსაც იგი უგზავნის აუტენტიფიკაციის სერვერს მისი შეკითხვის პასუხად. აუტენტიფიკაციის ერთერთ სქემას წარმოადგენს სტანდარტული პაროლების გამოყენება. პაროლი, როგორც მოსალოდნელია, არის სიმბოლოების ერთობლიობა რომელიც ცნობილია ქსელში ჩართული აბონენტისათვის, რომელიც შეაქვს ქსელში სეანსის დაწყებამდე, ზოგჯერ კი სეანსის ბოლოს(განსაკუთრებით საპასუხისმგებლო შემთხვევებში ქსელიდან ნონმალურად გამოსვლის პაროლს განასხვავებენ საწყისისაგან). ეს სქემა უსაფრთხოების თვალსაზრისით ყველაზე მეტად მოწყვლადია (გულსატკენია) – შეიძლება პაროლი წაგართვას და სხვა პირმა გამოიყენოს. ამ შემთხვევაში ყველაზე ხშირად გამოიყენება ერთჯერადი სქემის პაროლები, რომლებიც შემდგომი რეგისტრაციისთვის გამოუსადეგარობის ნიშნით ხასიათდებიან. ხოლო ძველი პაროლიდან ახლის მოღება არც ისე ადვილი ამოცანაა. ერთჯერადი პაროლების გენერაციისათვის გამოიყენება როგორც პროგრამული, ასევე კომპიუტერში ჩაშენებული აპარატურული გენერატორები. ამ მოწყობილობების ასამოქმედებლად მომხმარებლისთვის საიდუმლო სიტყვის ცოდნა აუცილებელია.

ერთერთი ყველაზე მარტივი სისტემა, რომელიც არ საჭიროებს დამატებით დანახარჯებს აღჭურვილობაზე და ამავე დროს დაცვის მაღალ დონეს უზრუნველყოფს, არის შ/Kეყ სისტემა, რომლის მიხედვითაც შესაძლებელია ერთჯერადი პაროლების წარდგინების თანმიმდევრობა ვაჩვენოთ. ამ პროცესში ორი მხარე მონაწილეობს: კლიენტი და სერვერი. სისტემაში რეგისტრაციის დროს სერვერი აგზავნის კლიენტის კომპიუტერზე მიწვევას, რომელიც შეიცავს მთავარ მარცვალს, რომლის გადაცემა შეიძლება ქსელში ღია ფორმით; იტერაციის მთვლელის მიმდინარე მნიშვნელობას და ერთჯერად პაროლზე მოთხოვნას, რომელიც უნდა შეესაბამებოდეს იტერაციის მთვლელის მიმდინარე მნიშვნელობას.
პასუხის მიღებისას სერვერი გადაამოწმებს მას და მართვას გადასცემს მომხმარებლისათვის საჭირო სერვერს.

ინფორმაციულ საზოგადოებაზე გადასვლის ტექნიკურ საფუძველს წარმოადგენს თანამედროვე მიკროელექტრონული ტექნოლოგიები, რომლებიც გამოთვლითი ტექნიკის საშუალებების ხარისხის უწყვეტ ზრდას უზრუნველყოფენ და მისი განვითარების იმ ძირითადი ტენდენციების შენარჩუნების ბაზის როლში გვევლინებიან, როგორიცაა მინიატურიზაცია, ელექტროენერგიაზე მოთხოვნილების შემცირება, ოპერატიული მეხსიერებისა და ჩაშენებული და გადასაღები დამგროვებლების მოცულობის მატება, მწარმოებლობისა და საიმედობის გაზრდა, გამოყენების სფეროსა და მასშტაბების გაფართოება. ამიტომ, თანამედროვე პირობებში კომპიუტერული სისტემების არასანქცირებული შეღწევისაგან დაცვა ხასიათდება პროგრამული და კრიპტოგრაფიკული მექანიზმების როლის ზრდით. რაც

იმაში გამოიხატება, რომ დაცვის სფეროში ახალი პრობლემების გაჩენა თხოულობს შედარებით მაღალი სირთულის გამოთვლითი მექანიზმების გამოყენებას, რაც ეფექტურად წყდება ეგმ –ის რესურსების გამოყენებით.

ერთერთ ყველაზე მნიშვნელოვან სოციალურ – ეთიკურ პრობლემას, რომელიც ჩნდება ინფორმაციის დაცვის კრიპტოგრაფიკული მეთოდების სულ უფრო ფართოდ გამოყენების გამო, წარმოადგენს წინააღმდეგობრიობა მომხმარებლის საკუთარი ინფორმაციის დაცვისა და შეტყობინების გადაცემის ნება-სურვილსა და სპეციალურ სახელმწიფოებრივ სამსახურების სურვილს შორის, - ჰქონდეს შეღწევის შესაძლებლობა ზოგიერთი სხვა ორგანიზაციისა და ცალკეული პირების ინფორმაციებზე, არაკანონიერი ქმედებების გამოაშკარავების მიზნით. მრავალ განვითარებულ ქვეყანაში თვალსაზრისთა ფართო სპექტრი შეინიშნება დაშიფრვის ალგორითმის გამოყენების რეგლამენტირების საკთხის მიდგომებისა. ყალიბდება წინადადებები კრიპტოგრაფიკული მეთოდების ფართო გამოყენების სრული აკრძალვიდან მათი გამოყენების სრულ თავისუფლებამდე. ზოგი წინადადება მიეკუთვნება ნებართვას მხოლოდ შესუსტებული, გამარტივებული ალგორითმის გამოყენებას ანდა დაშიფრვის გასაღებების აუცილებელი რეგისტრაციის რიგის დადგენას. ძალზე ძნელია ამ პრობლემის ოპტიმალური გადაწყვეტის პოვნა; როგორ შეფასდეს მათი ინფორმაციის კანონმორჩილად და არაკანონიერად გამომყენებელი მოქალაქეებისა და ორგანიზაციების დანაკარგების თანაფარდობანი.

გლობალური საინფორმაციო ქსელის ინტერნეტის გაჩენა, რაც თანამედროვე ინფორმაციული ტექნოლოგიების

მნიშვნელოვანი მიღწევაა, უამრავ კომპიუტერულ დანაშაულებებთანაც არის დაკავშირებული. ინტერნეტის ქსელის გამოყენების გამოცდილება გვიჩვენებს თუ რაოდენ სუსტია ინფორმაციის დაცვის ტრადიციული მეთოდები და რაოდენ ჩამორჩება თანამედროვე მეთოდებს. კრიპტოგრაფია გვთავაზობს ინტერნეტში ინფორმაციის უსაფრთხოებას და დღეისათვის აქტიურად მიმდინარეობს ამ ქსელში მისი აუცილებელი მექანიზმების დანერგვის სამუშაოები. არა უარი ინფორმატიზაციის პროგრესისადმი, არამედ კრიპტოზაციის თანამედროვე მიღწევების გამოყენება, - აი ესაა სტრატეგიულად სწორი გადაწყვეტილება. გლობალური ინფორმაციული ქსელების ფართოდ გამოყენების შესაძლებლობა და კრიპტოგრაფია დემოკრატიული საზოგადოების მიღწევაც და იმავდროულად მისი ნიშანთვისებაც არის. კრიპტოგრაფიის საფუძვლების ფლობა ინფორმაციულ საზოგადოებაში შეუძლებელია ცალკეული საზოგადოებრივი სამსახურების ობიექტური აუცილებლობა იყოს. არამედ, ის წარმოადგენს უკიდურეს აუცილებლობას სამეცნიერო-ტექნიკური მუშაკების ყველაზე ფართო ფენებისათვის, რომლებიც მონაცემთა კომპიუტერულ დამუშავებას იყენებენ ანდა თავად ამუშავებენ ინფორმაციულ სისტემებს. უსაფრთხოების სამსახურის თანამშრომლები და ორგანიზაციებისა და დაწესებულებების ხელმძღვანელობა. მხოლოდ ესენი შეიძლება მივიჩნიოთ ინფორმაციული უსაფრთხოების საშუალებების ექსპლუატაციისა და დანერგვის ეფექტურ ბაზად.

ერთ ცალკე აღებულ ორგანიზაციას ინფორმაციულ ნაკადებზე მთელი საზოგადოების მასშტაბით საკმაოდ

სრული და ეფექტური კონტროლის უზრუნველყოფა არ შეუძლია. კერძოდ, ცალკეულ სახელმწიფო ორგანოებს შეუძლიათ შექმნან პირობები დაცვის ხარისხოვანი საშუალებების ბაზრის ფორმირებისათვის. საკმარისი რაოდენობის სპეციალისტების მომზადებისათვის, კრიპტოგრაფიის საფუძვლების ათვისებისათვის და ინფორმაციის დაცვისათვის მასიური მომხმარებლების მხრიდან. მონაცემთა დაცვის სისტემების განვითარებას მნიშვნელოვნად უსწრებს წინ ინფორმაციული სისტემების გამოყენების არისა და მასშტაბების გაფართოების ტენდენცია. ასეთი სიტუაცია გარკვეული ხარისხით ტიპიური იყო და არის განვითარებული კაპიტალისტური ქვეყნებისათვის და ეს კანონზომიერიცაა: თავდაპირველად უნდა წარმოიშვას პრაქტიკული პრობლემა და შემდეგ უნდა მოიძებნოს მისი გადაწყვეტა. განვითარებული ქვეყნების მაგალითმა, - სისტემური პროგრამული უზრუნველყოფვის და კომპიუტერული ტექნიკის შეძენის შესაძლებლობამ, - სურვილი გაუჩინა სამამულო მომხმარებლებსაც. მასობრივი მომხმარებლების ჩათვლით, რომლებიც დაინტერესებული არიან მონაცემების ოპერატიულად დამუშავებითა და თანამედროვე საინფორმაციო-კომპიუტერული სისტემების ღირსებებით(სიკეთეებით), კომპიუტერიზაციის პრობლემების გადაწყვეტამ ამ დარგის განვითარების მაღალი ტემპები გამოავლინა. მაგრამ არსებითად, ინფორმაციის ავტომატიზებული დამუშავების საშუალებებისა ინფორმაციის დაცვის საშუალებების ერთობლივად განვითარება მნიშვნელოვნად დაირღვა, რაც მასობრივი კომპიუტერული დანაშაულობების მიზეზი გახდა. აღარავისთვისაა საიდუმლო, რომ დღეისათვის ეს ერთერთ აქტუალურ პრობლემად იქცა. უცხოური წარმოების

მონაცემთა დაცვის საშუალებების სხვა ქვეყანაში გამოყენებას გამართლება არ აქვს. რადგან ეს პროგრამული პროდუქტები არ აკმაყოფილებენ არსებულ საექსპორტო შეზღუდვებს; მეორე ასპექტს, რომელსაც არსებითი, პირველხარისხოვანი მნიშვნელობა აქვს, წარმოადგენს ის, რომ ამგვარმა პროდუქციამ უნდა გაიაროს სერთიფიცირების პროცედურა ასეთ სამუშაოებს დაქვემდებარებულ ორგანიზაციებში.

უცხოური ფირმებისა და ორგანიზაციების სერთიფიკატებს სამამულოს შეცვლა არანაირად არ შეუძლია. უცხოური სისტემური და გამოყენებითი პროგრამული უზრუნველყოფვის გამოყენების ფაქტიც კი, პოტენციალურად ინფორმაციული რესურსებისადმი მომატებულ საფრთხეს წარმოადგენს. ინფორმაციის დაცულობის საშუალებების სათანადო შესწავლის გარეშე გამოყენებამ, თუ ის არე შეესაბამება შესასრულებელ ფუნქციასა და დონეს, შეიძლება სიტუაციის მრავალჯერ გართულება გამოიწვიოს.

შიგა ბაზარზე ინფორმაციის დაცვის საშუალებების უკმარობა ხანგრძლივი პერიოდის განმავლობაში არ იძლევა მონაცემთა დაცვის ღონისძიებების მასშტაბურად განხორციელების შესაძლებლობას. მდგომარეობას ამ სფეროში სპეციალისტების სიმცირეც ართულებს, რადგან მათი მომზადება მხოლოდ სპეციალურ ორგანიზაციებს შეუძლიათ. ჩვენს ქვეყანაში ინფორმაციული ტექნოლოგიების მასობრივად გამოყენების ერთერთი მნიშვნელოვნი თავისებურება ის არის, რომ ინფორმაციული რესურსის სახელმწიფოებრივი დაცვის პრობლემის ეფექტურად გადაწყვეტისათვის აუცილებელია განმასაშუალებელი ღონისძიებების გატარება მასიურ

მომხმარებლებში. ინფორმაცია პირველ რიგში მისი შექმნის, შეგროვების და გადამუშავების ადგილზე უნდა იყოს დაცული და იმ ორგანიზაციების მიერ, რომლებიც უშუალოდ ზარალდებიან მონაცემებზე არასანქცირებული შეღწევებისა გამო. ეს პრინციპი რაციონალურიცაა და ეექტურიც: ცალკეული ორგანიზაციების ინტერესების დაცვა – ეს მთლიანად სახელმწიფოს ინტერესების დაცვის შემადგენელი ნაწილია.

ქსელებში ინფორმაციის დაცვის უზრუნველყოფა

გამოთვლით სისტემებში თავს იყრის ინფორმაცია, რომელთა გამოყენების განსაკუთრებული უფლება ენიჭებათ განსაზღვრულ პირებსა და ჯგუფებს, რომლებიც მოქმედებენ პირადი ინიციატივის ან სამსახურეობრივი საჭიროების შესაბამისად. ასეთი ინფორმაცია დაცული უნდა იყოს გარეშე პირთა ყოველგვარი ჩარევისაგან. ამისათვის კი საჭიროა წინასწარ იქნას მიღებული სათანადო ზომები. გამოირიცხოს ქსელში შეღწევა პირებისა ვისაც არა აქვთ სათანადო უფლება. სისტემისა და მონაცემების ფიზიკური დაცვა შეიძლება განხორციელდეს მხოლოდ მუშა ეგმ –ებსა და კავშირის არხებთან მიმართებაში და შეუძლებელი აღმოჩნდეს გადაცემის საშუალებებისათვის, რომლებსაც დიდი განფენილობა აქვთ. ამ პრინციპით უნდა ვიყენებდეთ საშუალებებს, რომლებიც გამორიცხავენ მონაცემებზე არასანქცირებულ შეღწევას და უზრუნველყოფენ საიდუმლოების დაცვას.
მონაცემთა დამუშავებისა და გამოთვლითი სისტემების ფუნქციონირების პრაქტიკის გამოკვლევა გვიჩვენებს, რომ ინფორმაციის გადინების(გაჟონვის) და ქსელებსა და სისტემებში არასანქცირებული შეღწევის საკმაოდ მრავალი

მიმართულება არსებობს. მათ შორისაა:

- Yსისტემის მეხსიერებაში დარჩენილი ინფორმაციის წაკითხვა სანქცირებული მოთხოვნის შესრულების შემდეგ;
- ინფორმაციის მატარებლებისა და ფაილების ინფორმაციების კოპირება დაცვის ზომების გადალახვით;
- დარეგისტრირებული მომხმარებლის ნიღბით მოქმედება;
- სისტემის მოთხოვნის ნიღბით მოქმედება;
- პროგრამული მახეების დაგება და გამოყენება;
- ოპერაციული სისტემების სუსტი მხარეებით სარგებლობა;
- უკანონოდ ჩართვა აპარატურასა და კავშირის არხებში;
- დაცვის მექანიზმების დაზიანება ბოროტი განზრახვით;
- კომპიუტერული ვირუსების დანერგვა და გამოყენება.

ინფორმაციის უსაფრთხოების უზრუნველყოფვა გამოთვლით ქსელებსა და ავტონომიურად მომუშავე პერსონალურ კომპიუტერებში მიიღწევა კომპლექსური ორგანიზაციული, ორგანიზაციულ-ტექნიკური, ტექნიკური და პროგრამული საშუალებებით.

ინფორმაციის დაცვის ორგანიზაციულ საშუალებებს მიეკუთვნება:

- იმ სათავსში შეღწევის შეზღუდვა, სადაც ხდება ინფორმაციის მომზადებისა და დამუშავება;
- კონფიდენციალური ინფორმაციის დამუშავებისა და გადაცემის დროს მხოლოდ შემოწმებაგავლილ თანამდებობის პირთა დაშვება;
- ინფორმაციის მატარებლებისა და სარეგისტრაციო

ჟურნალების უცხო პირებისაგან დაცულ სეიფებში განთავსება;

- უცხო პირთათვის დასამუშავებელი მასალების გადახედვის აკრძალვა დისპლეის, პრინტერის და სხვათა მეშვეობით;
- კრიპტოგრაფიკული კოდების გამოყენება მეტად რირებული ინფორმაციის კავშირის არხებიტ გადაცემის დროს;
- შესაღები ლენტების, ქაღალდების და სხვა მასალების განადგურება, რომლებიც შეიძლება შეიცავდნენ ღირებული ინფორმაციის ფრაგმენტებს.

ინფორმაციის დაცვის ორგანიზაციულ-ტექნიკური საშუალებებს მიეკუთვნება:

- ღირებული ინფორმაციის დამამუშავებელი აღჭურვილობის კვების განხორციელება კვების დამოუკიდებელი წყაროს მეშვეობით და სპეციალური ქსელური ფილტრებით;
- სათავსის კარებებზე კოდური საკეტების დაყენება;
- შეტანა-გამოტანის დროს ინფორმაციის ასახვისას თხევადკრისტალური ან პლაზმური დისპლეის გამოყენება; ხოლო მყარი ასლების მისაღებად – თერმოპრინტერების გამოყენება, რადგან დისპლეი იძლევა ისეთ მაღალხარისხოვან ელექტრომაგნიტურ გამოსხივებას, რომ მისი ეკრანიდან გამოსახულება შეიძლება რამდენიმე ასეული კილომეტრის დაშორებითაც იქნას გამოყენებული;
- კომპიუტერის გასარემონტებლად გადაცემის წინ მეხსიერებიდან ინფორმაციის წაშლა;
- კლავიატურისა და პრინტერების დაყენება(მონტაჟი) რბილ საფენებზე, რომ შემცირდეს ინფორმაციის გადაღების შესაძლებლობა აკუსტიკური წესით;

- ელექტრომაგნიტური გამოსხივების შეზღუდვა ინფორმაციის დამუშავების სათავსის ლითონის ფურცლების ან სპეციალური პლასტმასების საფარის ქვეშ მოქცევის გზით.

ინფორმაციის დაცვის ტექნიკური საშუალებებია – ტერიტორიისა და სათავსის დაცვის სისტემა მანქანათა დარბაზების ეკრანიზების მეშვეობით და საკონტროლო-გამშვები სისტემის მოწყობით. ინფორმაციის დაცვა ქსელებსა და გამოთვლით დაწესებულებებში ტექნიკური საშუალებების გამოყენებით ხორციელდება მეხსიერებაში შეღწევის ორგანიზების საფუძველზე:

- მონაცემების ბლოკირებისა და გასაღებების შემოტანის გზით;
- კომპიუტერის მეხსიერების სხვადასხვა დონეზე შეღწევის კონტროლის გზით;
- საკონტროლო ბიტების გამოყოფით ჩანაწერებისათვის იდენტიფიკაციის მიზნით და ა.შ.

ინფორმაციის დაცვის პროგრამული საშუალებების არქიტექტურა მოიცავს:

- უსაფრთხოების კონტროლი, მათ შორის რეგისტრაციის კონტროლი სისტემაში შესასვლელად, ფიქსირება სისტემურ ჟურნალში, მომხმარებლის მოქმედებების კონტროლი;
- რეაქცია(მათ შორის ხმოვანი) ქსელის რესურსებზე შეღწევის კონტროლის დაცვის სისტემის დარღვევაზე;
- შეღწევის მანდატების კონტროლი;
- ოპერაციული სისტემების(ბაზური ზოგადსისტემური და ქსელური) დაცულობის ფორმალური კონტროლი;
- დაცვის ალგორითმის კონტროლი;
- ტექნიკური და პროგრამული უზრუნველყოფვის

სწორი ფუნქციონირების შემოწმება და დადასტურება.

სისტემის საიმედო დაცვისა და უნებართვო მოქმედებების შემთხვევების გამოსავლინებლად ტარდება სისტემის მუშაობის რეგისტრაცია: იქმნება სპეციალური დღიურები და ‘‘პროტოკოლები’’, რომლებშიც სისტემაში ინფორმაციის დაცვასთან დაკავშირებული ყველა მოქმედება ფიქსირდება. ფიქსირდება განაცხადის შემოსვლის დრო, მისი ტიპი, მომხმარებლისა და ტერმინალის სახელი, რიმლიდანაც განაცხადი ინიციალიზდება. მოვლენათა შერჩევის დროხ, რომლებიც რეგისტრაციას ექვემდებარება, მხედველობაში უნდა ვიქონიოთ ის, რომ დარეგისტრირებული მოვლენების რაოდენობის ზრდასთან ერთად, რთულდება დღიურის გადახედვა და დაცვის გადალახვის მცდელობათა აღმოჩენა. ამ შემთხვევაში უნდა მოვიშველიოთ პროგრამული ანალიზი და დავაფიქსიროთ საეჭვო მოვლენები. აქ გამოიყენება სპეციალური პროგრამები დაცვის სისტემის ტესტირებისათვის. პერიოდულად, ანდა შემთხვევით შერჩეული დროის მომენტისათვის ისინი ამოწმებენ აპარატურული და პროგრამული საშუალებების დაცვის ქმედითუნარიანობას.
ინფორმაციის დაცულობის უზრუნველყოფვის და არასანქციონირებული მოთხოვნების გამოვლენის ზომების განსაკუთრებულ ჯგუფს მიეკუთვნება დროის რეალურ რეჟიმში დარღვევათა არმოჩენის პროგრამები, რომლებიც სპეციალურ სიგნალს გამოიმუშავებენ ისეთი მოქმედებების რეგისტრაციასა, რომლებსაც შეუძლიათ ინფორმაციის დაცულობა შეაფერხოს. სიგნალი შეიძლება შეიცავდეს ინფორმაციას დარღვევის ხასიათის, მისი წარმოშობის ადგილის და სხვათა შესახებ.

დაცვის ერთერთი გავრცელებული საშუალებაა – გამოტანილი ინფორმაციის შესახებ საიდუმლოების ცხადად მითითება. სისტემებში, რომლებიც გასაიდულოების რამდენიმე დონეს ინარჩუნებენ, ეკრანზე ან პრინტერზე ნებისმიერი ინფორმაციის ერთეულის გამოტანას (ფაილი, ჩანაწერი, ცხრილი), ახლავს სპეციალური გრიფი, საიდუმლოების დონის მითითებით. ამ მოთხოვნის რეალიზებას შესაბამისი პროგრამული საშუალებები ახდენენ. ცალკე ჯგუფად გამოყოფენ დაცვის საშუალებებს პროგრამული უზრუნველყოფვის არასანქცირებული გამოყენების შემთხვევებისათვის. მათ განსაკუთრებული მნიშვნელობა შეიძინეს თანამედროვე პერსონალურ კომპიუტერებში.

მოთხოვნები ინფორმაციის დაცვის თანამედროვე საშუალებების მიმართ

არასანქცირებული შეღწევისაგან ინფორმაციის დაცვის საშუალებები, რომლებიც მათდამი წაყენებულ მაღალ მოთხოვნებს პასუხობენ, უნდა უზრუნველყოფდნენ შემდეგ მოთხოვნებს:

- შეღწევის დისკრეპციული და მანდატური შეღწევის პრინციპი;
- მეხსიერების გასუფთავება;
- მოდულების იზოლაცია;
- დოკუმენტების მარკირება;
- შეტანა-გამოტანის დაცვა ინფორმაციის განმხოლოებულ ფიზიკურ მატარებლებზე;
- მომხმარებლისა და მოწყობილობების შეპირისპირება;

- იდენტიფიკაცია და აუტენთიფიკაცია;
- რეგისტრაცია;
- მომხმარებლის ურთიერთმოქმედება დაცვის საშუალებების კომპლექსთან;
- საიმედო გამართვა(აღდგენა);
- დაცვის საშუალებების კომპლექსის მთლიანობა;
- მოდიფიკაციის კონტროლი;დისტრიბუციის კონტროლი;
- არქიტექტურის გარანტიები.

არასანქცირებული შეღწევისაგან ინფორმაციის დაცვის კომპლექსურ საშუალებებს თან უნდა ახლდეს შემდეგი საბუთები:

- ინფორმაციის დაცვის საშუალებები გამოყენების ინსტრუქცია;BA
- მომხმარებლის ინსტრუქცია;
- ტესტური დოკუმენტაცია;
- საკონსტრუქტორო(საპროექტო) დოკუმენტაცია.

ასე, რომ კომპლექსური სისტემა ქვესისტემების გაერთიანებას წარმოადგენს, რომლის კონკრეტული შესაძლებლობები განსაზღვრავენ გამოთვლითი ტექნიკის საშუალებების დაცულობის დონეს. რეალური ეფექტურობა განისაზღვრება არა მარტო საბაზო, არამედ დამატებითი ქვესისტემების ფუნქციონალური შესაძლებლობებით და მათი რეალიზების ხარისხით.

კომპიუტერულ სისტემებსა და ქსელებს მიდრეკილება აქვთ ფართო სპექტრის პოტენციალური საშიშროებების მიმართ, რაც დაცვის ქვესისტემებისა ფუნქციების აუცილებლად გათვალისწინებას განაპირობებს. მიზანშეწონილია პირველ რიგში უზრუნველვყოთ ინფორმაციის ყველაზე ინფორმატიული არხების

დაცულობა, რომლებსაც ჩვენის აზრით მიეკუთვნება:

- მანქანური მატარებლებიდან მონაცემების კოპირების შესაძლებლობა;
- ინფორმაციის გადაცემის არხები;
- კომპიუტერის ან ჩაშენებული დამგროვებლების გატაცება.

ამ არხების ჩაკეტვის პრობლემას ართულებს ის, რომ მონაცემთა დაცვის პროცედურები არ უნდა იწვევდნენ გამოთვლითი სისტემების მწარმოებლობის მნიშვნელოვან შემცირებას. ეს საკითხი შეიძლება ეფექტურად გადაწყდეს ინფორმაციის დაშიფრვის გლობალური ტექნოლოგიის საფუძველზე.

დაცვის თანამედროვე მასობრივი სისტემა უნდა იყოს ერგონომიული და ხასიათდებოდეს ისეთი კეთილისმყოფელი თვისებებით, რომლებიც განაპირობებს მათ ფართოდ გამოყენებას. მათ მიეკუთვნება:

- კომპლექსურობა – მონაცემთა დაცულად დამუშავების სხვადასხვა რეჟიმების დაყენების შესაძლებლობა მომხმარებელთა სპეციფიკური მოთხოვნილებების ჩათვლით, რომელიც ითვალისწინებს დამრღვევთა მიერ შემოთავაზებულ ქმედებაღა საკმაოდ ვრცელ ჩამონათვალს.
- თავსევადობა – სისტემა თავსებადი უნდა იყოს ამ ოპერაციული სისტემისათვის დაწერილ ყველა პროგრამასთან და უნდა უზრუნველყოფდეს კომპიუტერის მუშაობის დაცულ რეჟიმს გამოთვლით ქსელში;
- მობილობა – სისტემის სხვადასხვა კომპიუტერულ სისტემებში, მათ შორის პორტატულებში, დაყენების შესაძლებლობა;
- მოხერხებულობა მუშაობის დროს – სისტემა უნდა

იყოს საექსპლუატაციოდ მარტივი რომ არ სჭირდებოდეს მომხმარებელს მუშაობის შეჩვეული ტექნოლოგიის არსებითად შეცვლა;

- მუშაობა რეალური დროის მასშტაბში – ინფორმაციის გარდაქმნის პროცესები, მათ შორის ციფრული, უნდა სრულდებოდეს დიდი სიჩქარით;
- ინფორმაციის დაცვის მაღალი დონე
- სისტემის მინიმალური ღირებულება.

თანამედროვე ინფორმაციული ტექნოლოგიების მასობრივად გამოყენების კვალდაკვალ ინფორმაციის დაცულობის საკითხიც ექცევა ადამიანის ყურადღების სფეროში. ინფორმაციის დაცულობის მეთოდებზე დაფუძვნებულია ელექტრონული გადარიცხვების სისტემა, საიდუმლო ინფორმაციების კავშირის ღია არხებით გადაცემის შესაძლებლობა და ბევრი სხვა ამოცანების გადაწყვეტა, რაც კომპიუტერულ სისტემებსა და ქსელებში XXI საუკუნეში პირველი რიგის ნიშნით ხასიათდება. პრაქტიკულმა მოთხოვნილებებმა დასვა დღის წესრიგში ეს საკითხები და აქედან გამომდინარე, ამ სფეროში კვლევებისა და დამუშავებების ღიად გაშლის აუცილებლობა. და, შესაბამისად ინფორმაციის დაცულობის საფუძვლების ცოდნა მეცნიერ-მკვლევართა და პრაქტიკოსთა და აგრეთვე ინფორმაციულ და სატელეკომუნიკაციო სისტემების პროექტირებასა და ექსპლუატაციაში დასაქმებულთა ცოდნის რეალურ სეგმენტად იქცა.
თანამედროვე გამოყენებითი კრიპტოგრაფიისა და ინფორმაციის დაცულობის ერთერთ აქტუალურ პრობლემას ბლოკური ტიპის ჩქაროსნული პროგრამული შიფრების და შესაბამისი მოწყობილობების დამუშავება წარმოადგენს.

დღეისათვის შეიძლება ვილაპარაკოთ დაშიფრვის რიგ წესებზე, რომლებიც გამოყენების სხვადასხვა იდეას ეფუძვნება:

- დაშიფრვის ალგორითმის გენერირება საიდუმლო გასაღებით;
- ჩასმები, რომლებიც დამოკიდებულია გარდასახვად მონაცემებზე;
- გადამრთველების თავისუფალი განრიგის გამოყენება და სხვა.

ვინდოუს 7-ის ვერსიები

ჭინდოწს7 საწყისი საშუალებას გვაძლევს ვიმუშავოთ ნებისმიერი რაოდენობის პროგრამასთან და გამოყენებებთან. ამავე დროს, იგი ეკონომიურად იყენებს კომპიუტერის რესურსებს (ნეთბუქებსაც და სამაგიდო კომპიუტერებსაც)). სიმარტივე, მოხერხებულობა, ეკონომიურობა - არიან ჭინდოწს 7-ის მთავარი უპირატესობები.
საწყის ვერსიას ჭინდოწს 7-ის ყველაზე „დაჭრილ“ ვერსიად მიიჩნევენ. მასში არ არის არც Aერო ინტერფეისი, არც დაშორებული სამუშაოს შესრულების შესაძლებლობა, არც ვირტუალური სამუშაო მაგიდა და XP Mოდე, არც ვინდოუს მედია ცენტრი. ხოლო ბიზნეს-ფუნქციის შესახებ შეგიძლიათ არც გაიფიქროთ. ყველაზე საწყენი რაცაა - აქ დაბლოკილია ისეთი მარტივი რამ, როგორიცაა ფონური სურათების ცვლილებები (წალლპაპერ)
ჭინდოწს7 საოჯახო, საბაზისო საშუალებას გვაძლევს სწრაფად და ადვილად შევასრულოთ ყოველდღიური ამოცანები. ჭინდოწს 7 საოჯახო, საბაზო, უზრუნველყოფს უფრო სწრაფ და ადვილ შეღწევას პროგრამებთან და

დოკუმენტებთან, რომლებიც ყველაზე ხშირად გამოიყენება. ამ შემთხვევაში, ახლა, დრო, რომელსაც თქვენ ხარჯავდით ძებნაზე, შეგიძლიათ გამოიყენოთ აუცილებელი ამოცანებისათვის. გაუმჯობესებული სანავიგაციო სისტემის წყალობით შესაძლებელი გახდა ადვილად და სწრაფად შეასრულოთ ყოველდღიური ამოცანები. ჭინდოწს7 საოჯახო შედარებით „მდიდარია“. აქ უკვე აღარაა პრობლემა „ფონის შეცვლაზე“, Aერო-მხარდაჭერა ჯერ ისევ ნაწილობრივ განვითარებულია (ნახევრად გამჭოლი ფანჯრებია). უფრო „მოზრდილი“ ფუნქციები ჯერ კიდევ აკლია. საბაზო ვერსია 2 ვარიანტადაა - 32 და 64 ბიტიანი. უფრო უფროსი ვერსიები წარმოდგენილია დისკით (32-იც და 64 ბიტიანი ოპერაციული სისტემით).
არსებობს ჭინდოწს7 საოჯახო, გაფართოებული ვერსია, რომელიც შესაძლებლობას გვაძლევს ადვილად შევქმნათ საოჯახო ქსელი და გავუცვალოთ ერთმანეთს ფოტოსურათები, მუსიკა, ვიდეო ჩანაწერები. შეიძლება აგრეთვე გადავათვალიეროთ, შევაჩეროთ, უკან გადავახვიოთ თV გადაცემები და კიდევაც ჩავიწეროთ ისინი. ეს ჭინდოწს 7 ჰომე Pრემიუმ შეიცავს ორ დისკს: 32 და 64 ბიტიანს. აქტივაციის გასაღები, ცხადია მხოლოდ ერთია, ამიტომ არ შეიძლება ერთ კომპიუტერზე დავაყენოთ 64-ბიტიანი ვერსია და მეორეზე - 32 ბიტიანი.
გარდა ამისა, ჭინდოწს7 ჰომე Pრემიუმ-ს აქვს ორი არსებითი განსხვავება Hომე Bასიც-დან. ესაა მედია ცენტრის არსებობა და Mულტი თოუცჰ ტექნოლოგია. ცხადია, დღეისათვის ყოვლისშემძლე სენსორული მოწყობილობები უფრო და უფრო მეტად ვრცელდება. (ცხადია არა მარტო მობილურ ტელეფონებზე, არამედ სრულყოფილ, სრულფასოვან მონიტორებზე და კომპიუტერ-მონობლოკებზეც (Hჰ

თოუცჰშმარტ) ნათელია ტენდენცია, რომ Mიცროსოფტ-მა გადაწყვიტა გამოუშვას ოპერაციული სისტემა, რომელიც ოპტიმალური იქნება სათითე ტექნოლოგიით. ჭინდოწს7-ს ესმის მრავალთითიანი ბრძანებები. ეს საშუალებას გვაძლევს მოხერხებულად ვიმუშავოთ ფოტოსურათებზე, სამოგზაურო რუკებზე, სურათებზე და სხვა გრაფიკულ ობიექტებზე. ის, რაც რამდენიმე წლის წინ იყო ფანტასტიკა, მაგრამ გახმაურებული - Mიცროსოფტ შურფაცე მაგიდის გამოშვების ემდეგ - რაც იყო ექსპერიმენტული და საკმაოდ შორს ჩვეულებრივი სამომხმარებლო დამუშავებისაგან, დღეს გახდა დასაშვები პრაქტიკულად ყველასათვის. და ეს პირველ რიგში ჭინდოწს 7-ის წყალობით არის.
კიდევ ერთი მცირე დეტალი, რაც Hომე Eდიტიონ-ს განასხვავებს Hომე Bასიც-გან, ესაა მხარდჭერა: 16 გიგაბაიტიანი ოპერატიული მეხსიერება 8 გბ-ს ნაცვლად. ცხადია, ეს ეხება მხოლოდ 64 ბიტიან ვერსიას (32 ბიტიანში - მაქსიმუმ 4 გბ. არის ოპერატიული მეხსიერება). მაგრამ, თუკი ეს მაინც არ გაკმაყოფილებთ, მაშნ მიმართეთ ჭინდოწს7 პროფესიონალს (Pროფესსიონალ). აქ მიღწეულია 192 გბ. ოპერატიული მეხსიერება. ეს ვერსია საშუალებას გვაძლევს ყველა ბარიერი გადავლახოთ წარმატების მიღწევის გზაზე. ის უზრუნველყოფს მრავალი პროგრამის გაშვებას ჭინდოწს XP რეჟიმში. და სწრაფად აღადგენს მონაცემებს ავტომატური არქივაციიის მეშვეობით საშინაო და კორპორატიულ ქსელებში. გარდა ამისა, ადვილი და უსაფრთხოა კორპორატიულ ქსელებში ჩართვა დომენებთან შეერთებით ფუნქციის მეშვეობით. რადგან ჭინდოწს7 Pროფესსიონალ განკუთვნილია არა მარტო ბიზნესისათვის, ის შეიცავს ბრწყინვალე შესაძლებლობებს, რაც Pრემიუმ-ს ჰქონდა გართობების თვალსაზისით.

ჭინდოწს7 - მაქსიმალური. ესაა ყველაზე უფროსი ვერსია, რაც მომხმარებელს აძლევს მაქსიმალურ შესაძლებლობებს. ესაა Uლტიმატე - ვერსია, მასში ჩართულია Mულტილინგუალ Uსერ Iნტერფაცე Pაცკ, რომელიც შესაძლებლობას გვაძლევს შევცვალოთ სისტემის ინტერფეისის ენა. ამგვარად, Uლტიმატე-ში არის შესაძლებლობა ჩატვირთვები ვაწარმოოთ ვირტუალური მყარი დისკიდან და რამდენიმე ტექნოლოგია, რომელიც ორიენტირებულია კორპორაციულ მომხმარებელზე. უპირველეს ყოვლისა, ფედერატიული ძიება (Fედერატედ შეარცჰ), Bიტლოცკერ და Bიტლოცკერ თო Gო. ეს Fედერატედ შეარცჰ უზეუნვწლყოფს კორპორატიული მონაცემების ძიებისადმი ჩაშენებულ მხარდაჭერას, რომელიც ინახება პერსონალურ კომპიუტერის მომხმარებლის გარეთ. ანუ, თქვენ შეგიძლიათ ეძიოთ მონაცემები კორპორატიულ ქსელებში ანდა საიტებზე, შჰარეPოინტ-ზე ისევე, როგორც ლოკალურ კომპიუტერზე ახდენდით. Bიტლოცკერ - ესაა დაშიფრულობა დისკისა კომპიუტერში, ხოლო Bიტლოცკერ თო Gო - დაშიფვრა მონაცემებისა გადამღებ მატარებელზე, მაგალითად ფლეშკაზე.
ზემოხსენებულის გარდა, ჭინდოწს7 Uლტიმატე-ს შეუძლია შეაფასოს მომხმარებლის და Iთ სპეციალისტების სხვა შესაძლებლობებიც, მაგრამ მათი გამოყენებისათვის მათ დასჭირდებათ გააფართოონ, გამოიყენონ ჭინდოწს შერვერ 2008 ღ2. დანარჩენებს კი მისცემს Mიცროსოფტ Dესკტოპ Oპტიმიზატიონ Pაცკ-ი.

ამოცანათა ახალი პანელი

პირველი ცვლილება, რომელსაც ჭინდოწს7 ოპერაციული

სისტემის ჩატვირთვისთანავე შევამჩნევთ – ესაა სრულიად გადამუშავებული ამოცანათა პანელი(თასკბარ) რომელიც გულისხმობის პრინციპით განთავსებულია ეკრანის ქვედა ნაწილში(ნახ.25) ყურადღებას იქცევს ნიშნების სიდიდე – დანართების გაშვების ღილაკები დიდი ზომისაა. თუმცა მომხმარებელს შეუძლია ოპერაციული სისტემის მომართვის დროს ისინი სურვილის მიხედვით შეარჩიოს. ამოცანათა პანელი ვინდოუსების ვერსიებში პრაქტიკულად ჭინდოწს 95-ის შემდეგ არ გადამუშავებულა. ამ ახალ ვერსიაში კი საფუძვლიანად მოდერნიზებულია.პირველ რიგში დამმუშავებლებმა სწრაფი გაშვების პანელი მოხსნეს(Qუიცკ Lაუნცჰ). ეს პანელი სრულიად ზედმეტი აღმოჩნდა, რადგან გაჩნდა მისი ნებისმიერ დანართი ამოცანის პანელზე მიმაგრების შესაძლებლობა. ასევე მოცილებულია ჭMP(ჭინდოწს Mედია Pლაყერ) პანელი.(ნახ.26) მის ნაცვლად გაჩნდა პანელი რომლის მეშვეობით ყველა დანართის მართვა შეიძლება. ამ პანელის გასააქტიურებლად საჭიროა გავუშვათ სასურველი დანართი, შემდეგ გავუშვათ აუდიო/ვიდეო ფაილი და დანართის ფანჯარა ჩავკეცოთ. შემდეგ როცა მომხმარებელი მოისურვებს, ფაილის მუშაობა შეაჩეროს, საჭიროა თავგის მაჩვენებელი მივიტანოთ გაშვებული პანელი დანართის ნიშანზე და გამოჩენილ პანელზე. დავაწვეთ საჭირო ღილაკს. ჩამოშლად პანელში დამატებით რეალიზებულია ჩამოშლადი სიები. ამ ფუნქციის დანიშნულებაა ამოცანათა პანელის მენიუს შეცვლა ხშირად გამოყენებული ფაილებისა და შესასრულებელი მოქმედებების სიით. (ნახ.26) ახლა ამ ჩამოშლილი სიით შეგვიძლია შეღწევა ნებისმიერ ფაილთან/მოქმედებასთან თაგვის ერთი დაკლიკებით. ჩამოშლადი სიის გამოსაძახებლად საკმარისია დანართის

ნიშანზე (ნახ.27) თაგვის მარჯვენა დაკლიკება.
აღსანიშნავია ფუნქცია Aერო Pეეკ. თუ გაშვებული გვაქვს რამდენიმე დანართი, ან ერრთ-ერთ განართში გახსნილია რამდენიმე ჩანართი, მაშინ სასურველი ჩანართის გამოსახვისათვის საკმარისია თაგვის მაჩვენებლის მასთან მიტანა. და ეკრანზე გამოჩნდება ყველა ღია ჩანართის მინიატურა. შესარჩევად საკმარისია თაგვის მაჩვენებლის მხოლოდ შეხება. Yთუნდაც ყველა ჩანართის ან გაშვებული დამატების ფანჯრები არ იყოს ჩაკეცილი, მათი გამოსახვისათვის საჭიროა მათზე თაგვის მაჩვენებლის მიტანა შერჩეულ მინიატურაზე. დანარჩენები ისევ ჩაკეცილი დარჩება.
ამოცანათა პანელის მარჯვენა მხარეში, შეტყობინებების უკიდურესად მარჯვნივ, გაჩნდა ახალი ღილაკი სახელწოდებით შჰოწ Dესკტოპ , ნახ28 რომლის დანიშნულებაა სამუშაო მაგიდა თაგვის ერთი დაკლიკებით გაასუფთავოს. მასზე განმეორებით დაკლიკება ყველა გახსნილ დანართს ძველ ადგილზე დააბრუნებს. მაგრამ თუ წამიერად გვსურს დავხედოთ სამუშაო მაგიდას, საკმარისია ამ ღილაკთან შევაჩეროთ თაგვის მაჩვენებელი. ეკრანზე დარჩება სამუშაო მაგიდა, ხოლო ყველა გახსნილი და ჩაკეცილი დანართები ნახევრად გამჭვირვალე გახდება და არანაირად არ შეუშლის ხელს სამუშაო მაგიდის ნახვას.
არის კიდევ ერთი საინტეწრესო სიახლე ამოცანათა პანელზე. თუ ჩვენ გვსურს ფაილების გადატანა ან კოპირება ერთი საქაღალდიდან მეორეში, მაშინ ამ შესასრულებელი ოპერაციის შესასრულებლად მდგომარეობის შესამოწმებლად საკმარისია შევხედოთ ღილაკს სახელწოდებით Eხპლორერ ამოცანათა პანელის მარცხენა ნაწილში. ეს ნიშანი ოპერაციის შესრულებისას ღებულობს

მწვანე ფონს. მწვანე ზოლის საზღვარი, ისევე როგორც კოპირების ფანჯრის ინდიკატორი, გადაადგილდება მარცხნიდან მარჯვნივ. ამასთანავე, თუ გახსნილია ერთი დამატების რამდენიმე ჩანართი, მაშინ ნიშანი მრავალზოლიანი ხდება. (ნახ.29)

შეტყობინებების ახალი არე

ჭინდოწს7 ოპერაციულ სისტემაში ასევე არსებითად გადამუშავდა შეტყობინებების ასრე ამოცანათა პანელის მარჯვენა მხარეში. ახლა მომხმარებელს თავად შეუძლია შეარჩიოს რომელი დანართების ნიშნები აჩვენოს მოცემულ არეში და რომელი დამალოს. ასევე ეძლევა მას შესაძლებლობა, შეარჩიოს ამათუიმ შეტყობინების გამოსახვის რეჟიმი. შეტყობინებების არეს მოსამართად საკმარისია თაგვის მარჯვენა ღილაკი დავაწკაპუნოთ ამოცანათა პანელზე, გამოჩენილი მენიუდანშევარჩიოთ ბრძანება Рrорerტიესდა ჩანართშითასკბარდიალოგის თასკბარ ანდ შტარტ Мენუ Рrоპერტიეს ელემენტების მართვის ჯგუფშიNოტიპიცატიონ არეა დავაწვეთ ღილაკსჩუსტომიზე. ეკრანზე გამოჩნდება დიალოგი Nოტიპიცატიონ Аrეა Iცონს, (ნახ.30) რომელშიც შეირჩევა სწორედ საჭირო პარამეტრები ნიშნის გამოსახვისთვის ყოველი დამატებისათვის.

საყურადღებოა ახალი ნიშანი რომელიც ჩნდება შეტყობინებების არეში. ესაა მხარდაჭერის სამსახურის ნიშანი ,,Аცტიონ ჩენტერ‘‘ რომელიც ყველა სისტემური შეტყობინების ცენტრს წარმოადგენს. როგორც გვახსოვს და მომხმარებლებს, მრავალი სისტემური შეტყობინება (მეტ-ნაკლებად მნიშვნელო¬ვანი) ჩნდებოდა ეკრანზე

კომპიუტერის ჩატვირთვის თანავე და მუშაობის პროცესში და დისკომფორტს ქმნიდა მუშაობისას. ახლა ყველა ეს შეტყობინება თავმოყრილია მხარდაჭერის ერთ ცენტრში და ერთადერთი, რასაც კომპიუტერი ატყობინებს მომხმარებელს, - არის შეტყობინება ამ სისტემის შესახებ - ესაა ჯვარი წითელ ფონზე ამ ნიშანზე რომ გავარკვიოთ, რას გვატყობინებს სისტემა, საკმარისია ამ ნიშანს შევეხოთ თაგვის მაჩვენებლით.(ნახ.31). უნდა აღინიშნოს, რომ მომხმარებელს თასკ ბარდიალოგის ჩანართის მეშვეობით თასკბარ ანდ შტარტ Мენუ Рროპერტიეს ყოველთვის შეუძლიადამოუკიდებლად გააწყოს ამოცანათა პანელი საკუთარი სურვილისა და გემოვნების მიხედვით.

ფანჯრებთან მუშაობის ახალი წესები

კომპიუტერის ყველა მომხმარებელს, ვისაც აქვს სისტემა გრაფიკული ინტერფეისით, დროდადრო უჩნდება მოთხოვნილება შეცვალოს გაშვებული ფანჯრის ზომები ანდა გაშალოს ფანჯარა ეკრანის მთელ არეზე სასურველი რიგით. დანართების ფანჯარა შეიძლება სამუშაო მაგიდაზე დავდოთ და ამგვარი განლაგების დროს ეკრანის არე განაწილდება გააქტიურებულ დამატებებს შორის ჰორიზონტალურად. ფანჯრების კასკადურად განთავსების დროს შეგვიძლია ფანჯრები ერთიმეორის გვერდით ისე მოვაქციოთ, რომ ზედა დანართის ფანჯრიდან ქვედას ნაწილი ჩანდეს. თუ ფანჯრებს ერთმანეთის გვერდი-გვერდ განვათავსებთ, მაშინ ეკრანის სასარგებლო ფართობი განაწილდება ამ დამატებებს შორის ვერტიკალზე. ყველა ეს ვარიანტები შეგვიძლია შევარჩიოთ მენიუდან ამოცანათა პანელზე მაჯვენა დაკლიკებით. პროგრამებისა და დიალოგების ფანჯრების განლაგების ეს წესები

გამოიყენებოდა ვინდოუსების ადრინდელ ვერსიებშიც, მაგრამ ამ ვერსიაში პროგრამისტებმა ფანჯრებისა და მასშტაბირების ტრადიციულ მიდგომებს დაუმატეს ფანჯრებთან მუშაობის რამდენიმე ახალი და საოცრად მოხერხებული მეთოდი, რომელიც ემყარება ფუნქციასAეროშნაპ, რომელიც საშუალებას გვაძლევს სწრაფად მარტივად დავდოთ ფანჯარა სამუშაო მაგიდაზე ერთერთი სტანდარტული წესით.
დავუშვათ, გვსურს დავაკოპიროთ ფაილი, მაგ. სურათი ერთი საქაღალდიდან მეორეში, იმისგან დამოუკიდებლად თუ რომელ მატარებლებზე არიან ისინი მოთავსებული. ამ ფაილური ოპერაციის ჩატარება შეიძლება ორი გზით: გავხსნათ საწყისი საქაღალდე, გამოვყოთ დასაკოპირებელი ფაილი, ჩავაკოპიროთ ის ოპერაციული სისტემის გაცვლის ბუფერში, დავხუროთ საწყისი საქაღალდე. შემდეგ გავხსნათ დანიშნულების საქაღალდე და ჩავსვათ მასში ფაილი ბუფერიდან. მეორე გზა:ერთდროულად გავხსნათ ორივე საქაღალდე, განვათავსოთ ისინი სამუშაო მაგიდაზე ისე რომ მათი ფანჯრები არ ფარავდნენ ერთმანეთს და თაგვით ‘‘გადავათრიოთ’’ სასურველი დოკუმენტები. და, აი, Aეროშნაპ არსებითად ამარტივებს ფანჯრების განთავსების საქმეს კოპირების მეორე წესისათვის. მარჯვენა და მარცხენა საქაღალდეებს ეკრანი გავუნაწილოთ თანაბრად. ფანჯრის მარცხნივ ან მარჯვნივ გადაადგილებისას ვერტიკალზე ეკრანის თანაბრად გაყოფა ავტომატურად ხდება როცა თაგვის მაჩვენებელი მიაღწევს ეკრანის კიდეს. ეკრანზე ფანჯრების საწყისი მდგომარეობის აღსადგენად საჭიროა თაგვის ორმაგი დაკლიკება ფანჯრის სათაურზე. კოპირების შემდეგ საქაღალდეების ფანჯარა შეგვიძლია დავხუროთ. თუ გვსურს პროგრამის ფანჯრის მთელ ეკრანზე გაშლა,

საკმარისია თაგვის მაჩვენებელი მივიტანოთ პროგრამის სათაურის სტრიქონზე, დავაწვეთ მარცხენა ღილაკს და მივიტანოთ შერჩეული ფანჯარა ეკრანის მარცხენა კოდეში, ავუშვათ თითი მარცხენა ღილაკს. საწყის ზომაზე დასაბრუნებლად საკმარისია სათაურის სტრიქონში ორმაგი დაკლიკება. მეორე გზით, - საკმარისია ორჯერ დაკლიკება დამატების სათაურის სტრიქონში. განმეორებითი ორმაგი დაკლიკება ფანჯარას საწყის მდგომარეობაში დააბრუნებს. რომ გავჭიმოთ დანართის ფანჯარა ვერტიკალზე ეკრანის მთელ სიგრძეზე, საკმარისიაორჯერ დაკიკება მასშტაბირებადი დანართის ქვედა კიდეზე. საწყის მდგომარეობასაც ანალოგიური წესით ვუბრუნდებით. საინტერესოა ისიც, რომ თუ ეკრანზე ერთდროულად რამდენიმე ფანჯარა გვაქვს გახსნილი და გვსურს ჩავკეცოთ ყველა, გარდა ერთისა, საჭიროა: თაგვის მაჩვენებელი მივიტანოთ დასატოვებელი ფანჯრის სათაურზე, დავაწვეთ მარცხენა ღილაკს, თითის აუღებლად ვამოძრავოთ ფანჯარა მანამ, სანამ ეკრანი არ განთავისუფლდება დანარცენი ფანჯრებისაგან. და მხოლოდ ამის შემდეგ გავანთავისუფლოთ ღილაკი. ნიშანზე დავაკლიკოთ.ნახ.32

ახალი სტანდარტული დამატებანი

აგრძელებს რა, წინა ოპერაციული სისტემების შემქმნელთა კარგ ტრადიციებს, კომპანია Microsoft- მა ეს ვერსიაც აღჭურვა ახალი სტანდარტული დამატებებით. რომლებიც საშუალებას გვაძლევენ სრულყოფილად ვიმუშაოთ კომპიუტერზე ისე რომ არ დაგვჭირდეს დამატებითი პროგრამების დაყენება. ცხადია, ჭორდPად დამატება ვერ შეცვლის Microsoft ჭორდ პროგრამას, მაგრამ

დოკუმენტის შექმნა შეიძლება. უნდა ითქვას, რომ ოპერაციული სისტემის წინა ვერსიებში ჭორდPად და Pაინტ - მნიშვნელოვანი ცვლილებები არ განუცდია. ჭინდოწს7 – ში კი შეიცვალა ინტერფეისი. ეს ლენტური ინტერფეისია. გაზრდილია ფუნქციათა რაოდენობაც ნახ.33
სხვა დამატებებში უნდა აღინიშნოს მათემატიკური შეტანის პანელი. ეს შესაძლებლობას გვძლევს ამოვიცნოთ ხელით შეტანილი ფორმულები. ნახ.34.
ოფისში შესვლისას ხშირად ვხედავთ პასუხისმგებელი პირების მონიტორებს რომელთა ეკრანი გადაფარულია წარწერებით. ახლა შეგვიძლია მსგავსი წარწერები მოვხსნათ და ისინი მივამაგროთ სამუშაო მაგიდაზე შტიცკყ Nოტეს დამატების დახმარებით. ჩვენ ნამდვილად აღარ დაგვავიწყდება დანიშნული შეხვედრის ჩატარება, დოკუმენტის დროულად წარდგენა და ა.შ.
დანართი გადახედვის საშუალებები XPშ Vიეწერ საშუალებას გვაძლევს საბუთებს გადავხედოთ ფორმატში. XPშ – დოკუმენტი მსგავსია ქაღალდის ელექტრონული ვერსიისა, რომლის შინაარსის შეცვლა შეუძლებელია. ამ სისტემაში ასეთი დოკუმენტის შექმნა შეიძლება ნებისმიერ პროგრამაში რომელშიც არის ბეჭდვის შესაძლებლობა.
კიდევ ერთი ახალი სტანდარტული დამატებაა მაკრატელი, რომელიც შესაძლებელს ხდის დავაკოპიროთ გაცვლის ბუფერში ეკრანის გამოკვეთილი(შერჩეული) ნაწილი, რომ ის შემდგომ ჩავსვათ რედაქტირებად ტექსტში. ასევე გაჩნდა შესაძლებლობა რომ კომპიუტერი მივუერთოთ ქსელურ პროექტორს და დაშორებულ სამუშაო მაგიდასაც.
უფრო მოხერხებული პერსონალიზაცია

ყოველ ადამიანს სურს ოპერაციული სისტემა საკუთარი

მოთხოვნილებების მიხედვით და მხატვრულად გემოვნებით მოაწყოს. ამისათვის ჩვენ შეგვიძლია შევარჩიოთ სამუშაო მაგიდის სასურველი ფონი დათოთხმეტამდე ხმოვანი სქემა(მუსიკალური გაფორმება). ჭინდოწს7 – ში ეს სერვისიც გათვალისწინებულია განსხვავებით ჭინდოწს Vისტა-საგან, რომელშიც მაგიდაზე მხოლოდ ცალკეული ფოტოგრაფიების განთავსება შეიძლებოდა. ჭინდოწს7– ს გააჩნია სლაიდ-შოუს მოწყობის საშუალებებიც რომლების შერჩევა შეიძლება თემებიდან პერსონალიზაციის ფანჯარაში(Pერსონალიზე). ამასთან, თემების გადმოწერა ინტერნეტიდანაც შეიძლება.ამ ფანჯრის გამოძახება შეიძლება სამუშაო მაგიდაზე მარჯვენა დაკლიკებით და წარმოდგენილი მენიუდან სასურველი სტრიქონის შერჩევით.
ოპერაციულ სისტემას ათზე მეტი ენა გააჩნია რომლებიც შეიძლება გამოვიყენოთ დიალოგებში, ბრძანებებში, კონკრეტულ მენიუში, ტექსტში, ხელნაწერის ამოსაცნობად და ა. შ. რომ დავაყენოთ დამატებითი ენა, საჭიროა შტარტ – იდან შევარჩიოთ ჩონტროლ Pანელ ბრძანება და შემდეგ ელემენტების მართვის ჯგუფში ჩლოცკ, Lანგუაგე ანდ ღეგიონ შევარჩიოთ მიმართვაჩჰანგე Kეყბოარდს ორ ოთჰერ ინპუტ მეთჰოდს და გამოვლენილ დიალოგში ღეგიონ ანდ Lანგუაგე გავხსნათ ჩანართი Kეყბოარდს ანდ Lანგუაგე. ამის შემდეგ უნდა დავაყენოთ ინტერნეტთან შეერთება; დავაწვეთ ღილაკსIნსტალლ/უნინსტალლ Lანგუაგეს და დიალოგში Iნსტალლ ორ უნინსტალლ Dისპლაყ Lანგუაგეს შევარჩიოთ მიმართვა სემდეგ შევარჩიოთ მიმართვა Lანცჰ ჭინდოწს Uპდატე და ფირმის საიტის გახსნის შემდეგ შევასრულოთ დაყენების ოსტატის მითითებები.

მეგზურის ფანჯრის დამატებითი ფუნქციები

ჭინდოწს7 ოპერაციული სისტემის მეგზურიც დაექვემდებარა ცვლილებებს. დამპროექტებლები მიუბრუნდნენ ვირტუალური საქაღალდეების სქემებს, რომელთა რეალიზებას ჯერ კიდევ Vისტა - ს დროს ცდილობდნენ. ამჯერად ყველა ფიზიკური საქაღალდე დაჯგუფებულია თემებად ვირტუალურ საქაღალდეში სახელწოდებით Lიბრარყ. მეგზურის ჩარჩო რომ აისახოს ეკრანზე, საკმარისია ამოცანათა პანელზე შევეხოთ ნიშანს. ეკრანზე გულისხმობის პრინციპით გამოჩნდება ბიბლიოთეკის საქაღალდე. ნახ.35ნახ.36 მეგზურის მარჯვენა მხარეში შესაძლებელია Aშერჩეული ფაილის შინაარსის გაცნობა.
ასევე გადამუშავებულია ინსტრუმენტების პანელი, რომელზედაც თავმოყრილ;ია მეგზურის მუშაობის მართვის ბრძანებები. დაფარული ბრძანებების გამოსახვისათვის საკმარისია მარჯვნივ არსებულ ორმაგისრიან ღილაკს დავაწვეთ. რომ გამოვსახოთ მენიუს ძირითადი სტრიქონი, ვაწვებით Aლტ ღილაკზე.
მეგზური უზრუნველყოფს შერჩეული ფაილის ბეჭდვას დანართის/გამოყენების გაშვების გარეშე, რომელშიც ეს ფაილია შექმნილი. შესაძლებელია ჩაწერა ოპტიკურ დისკზეც. სისტემას შეუძლია IშO-სახით ჩაწერა DVD დისკზედაც.

ახალი ფანჯარა ‘‘მოწყობილობა და პრინტერები

განახლებული დამატება ‘‘მოწყობილობა და პრინტერები’’(Dევიცეს ანდ Pრინტერს) ერთ ფანჯარაში აერთიანებს კომპიუტერთან მიერთებულ ყველა მოწყობილობას. ამჯერად ამ ფანჯარაში აისახება აგრეთვე ყველა მონიტორი, UშB –მეხსიერება, ფოტოაპარატი, პროექტორი, გასართობი მინადგარი და სხვა. ეს საშუალებას იძლევა არა მარტო მუშაობა დავიწყოთ ამ მოწყობილო¬ბებზე, არამედ დიაგნოსტი¬რებისა და უზუსტო¬ბებზე რეაგირების შესაძლებ-ლობაც გვაქვს. ამ ფანჯრის გასახსნელად საჭიროა დავაწვეთ შტარტ ღილაკს და ამოცანათა პანელის მარჯვენა მხარეში გამოსახულ მთავარ მენიუში დავაწკაპუნოთ თაგვის ღილაკი Dევიცეს ანდ Pრინტერს სტრიქონში.

სააღრიცხვო ჩანაწერების კონტროლი

მომხმარებელთა სააღრიცხვო ჩანაწერების კონტროლიUAჩ(უსერ Aცცოუნტ ჩონტროლ) პირველად ოპერაციულ სისტემა Vისტა-ში იყო გამოყენებული. მაგრამ ეს ფუნქცია მოქმედებდა ძალზე პრიმიტიულად. ოპერაციული სისტემა წყვეტდა მუშაობას და ელოდებოდა მომხმარებლის დათანხმებას ყოველი მოქმედების შემდეგ. ითიშებოდა სისტემის დაცვა მავნე პროგრამებისაგან.(ფაილის ან კომპიუტერის პარამეტრის შეცვლა და სხვ.) ჭინდოწს 7 –ში ეს ხარვეზი გამოსწორებულია და ახლა მომხმარებელს თავად შეუძლია საკუთარი კომპიუტერის დაცვის ხარისხი შეარჩიოს. პარამეტრიების დასაყენებლად დიალოგი ხდება ასე: ვხსნით მართვის პანელის ფანჯარას,)რომლის სახე შეგვიძლია შევცვალოთ თუ ჩამოხსნილი Vიეჭ ბყ ჩამონათვალიდან

შევარჩევთ სასურველ პუნქტს, მაგ. Lარგე Iცონს.) შემდეგ საჭიროა დავაწვეთ ღილაკსUსერ Aცცოუნტს და წარმოდგენილ იმავე სახელწოდების დიალოგში დავაკლიკოთ ჩჰანგე Uსერ Aცცოუნტ ჩონტროლ შეტტინგ სტრიქონზე. მხოლოდ ამის შემდეგ გამოჩნდება ეკრანზე დიალოგი Uსერ Aცცოუნტ ჩონტროლ შეტტინგ. შემდეგ კომპიუტერში შეტანილი ცვლილებების შესახებ შეტყობინებების ჩვენება შეიძლება მცოცავი ვერტიკალური რეგულირატორის მეშვეობით.

ლიტერატურა:

1.Э. Таненбаум. Современные Орерационные Системы. 2008 3-е изд., Питер

2. Острейковский В.А. Информатика: Учеб. пособие для студентов сред. проф. учеб. заведений. – М.: Высш. шк., 2001. – 319с.:ил.

3. Экономическая информатика / под ред. П.В. Конюховского и Д.Н. Колесова. – СПб.: Питер, 2007. – 560с.:ил.

4. Дейтел Г. Введение в Операционных Систем М. 2007

5 . Информатика: Базовый курс / С.В. Симонович и др. – СПб.: Питер, 2005. – 640с.:ил.

6. Молдовян А.А., Молдовян Н.А., Советов Б.Я. Криптография. – СПб.: Издательство “Лань”, 2007. – 224с.,ил. – (Учебники для вузов. Специальная литература).

7. Дейтел Х.М., Дейтел П.Дж., Чофнес Д.Р., Операционные системы. Основы и принципы., 2009.

8. В.Б. Комягин И.Ю.Савельев И.В. Ривкин . Windows 7 М . 2011

9. Windows 7 ს. რაზმაძე თბ. 2011
10. გ. მაჭარაშვილი - ოპერაციულ სისტემათა საფუძვლები თბ. სტუ. 2009
11. Олифер И.Г. Сетевие Операционные Системы М. 2001
12. Кейпингерт П. Элемеиты Операционных Систем. М. 1997
13. www.allbest.ru

სარჩევი

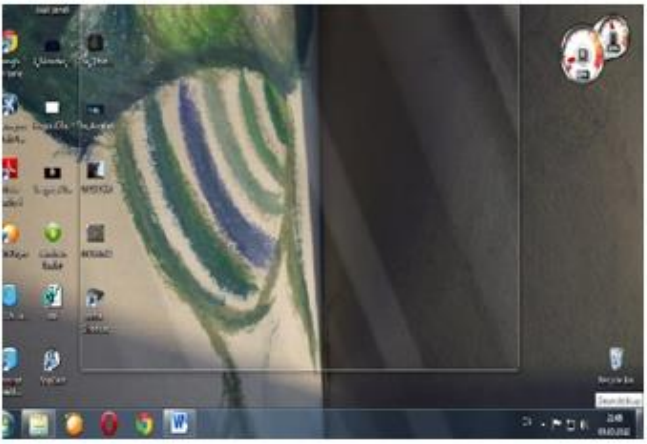
ნახ.25

ნახ.26

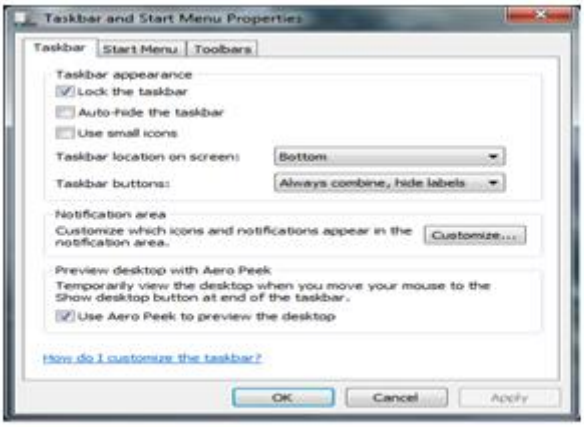
ნახ.47


amocanaTa auli paneli - Micro...
shakespeare[1] [Compatibility ...
maiakovskis leqsebi - Microsoft ...

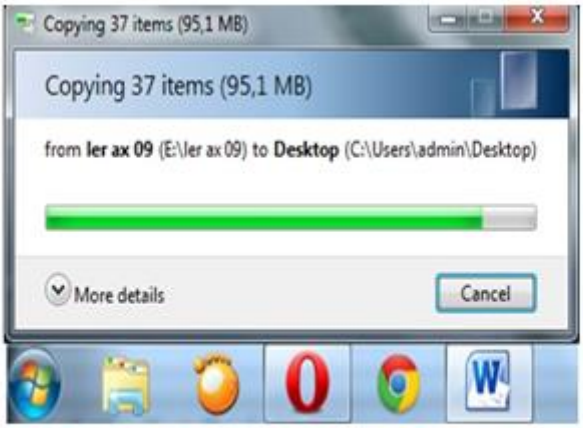
ნახ.28



ნახ.29


Taskbar and Start Menu Properties
Taskbar
Start Menu
Toolbars
Taskbar appearance
Lock the taskbar
Auto-hide the taskbar
Use small icons
Taskbar location on screen:
Bottom
Taskbar buttons:
Always combine, hide labels
Notification area
Customize which icons and notifications appear in the notification area.
Customize...
Preview desktop with Aero Peek
Temporarily view the desktop when you move your mouse to the Show desktop button at end of the taskbar.
Use Aero Peek to preview the desktop
How do I customize the taskbar?
OK
Cancel
Apply

ნახ.30


Copying 37 items (95,1 MB)
Copying 37 items (95,1 MB)
from ler ax 09 (E:\ler ax 09) to Desktop (C:\Users\admin\Desktop)
More details
Cancel

ნახ.31

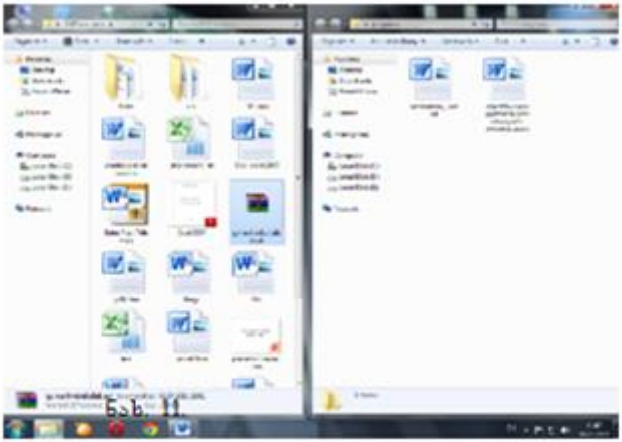
ნახ.32 

ნახ. 33 

ნახ.34 


wereteli
Toolbars
Cascade windows
Show windows stacked
Show windows side by side
Show the desktop
Start Task Manager
Lock the taskbar
Properties

ნახ.35

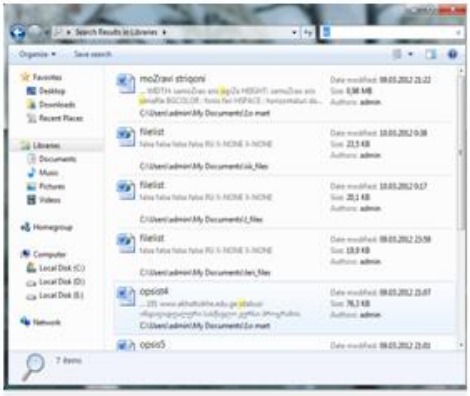


ნახ.36

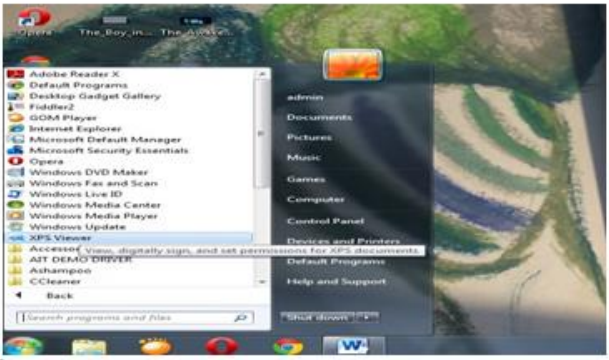
ნახ37 

ნახ.38 

ნახ. 39 



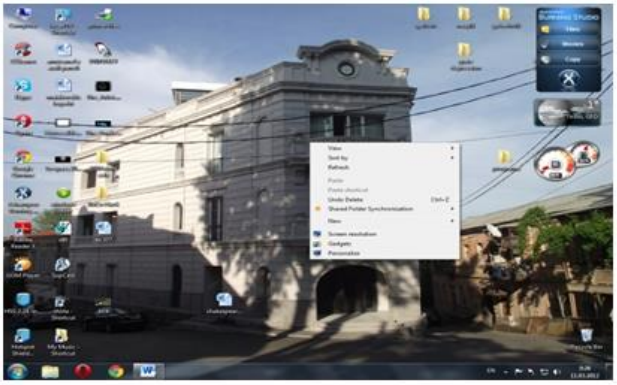
ნახ.41

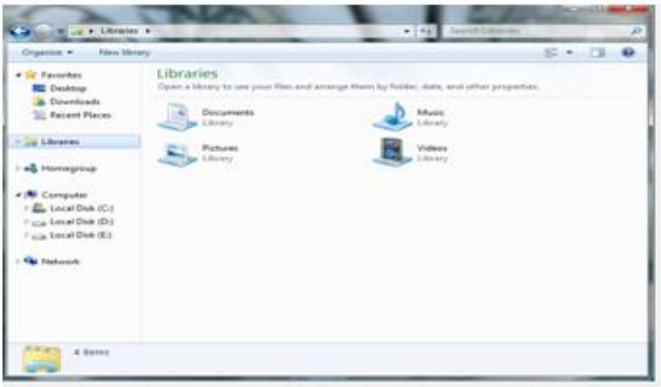

Adobe Reader X
Default Programs
Desktop Gadget Gallery
Fiddler2
GOM Player
Internet Explorer
Microsoft Default Manager
Microsoft Security Essentials
Opera
Windows DVD Maker
Windows Fax and Scan
Windows Live ID
Windows Media Center
Windows Media Player
Windows Update
XPS Viewer
Accessories
AIT DEMO DRIVER
Ashampoo
CCleaner
Back
admin
Documents
Pictures
Music
Games
Computer
Control Panel
Devices and Printers
Default Programs
Help and Support

ნახ.42

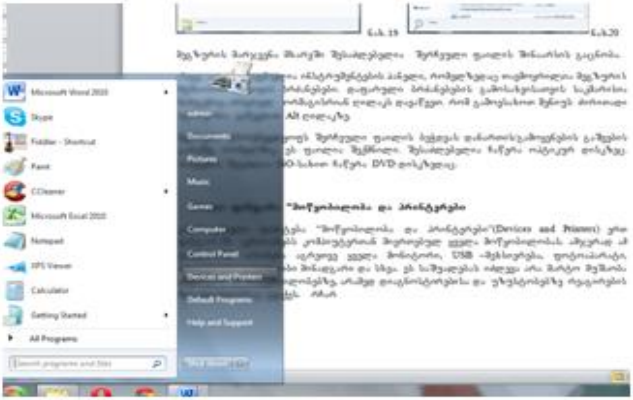
ნახ.40

ნახ.43

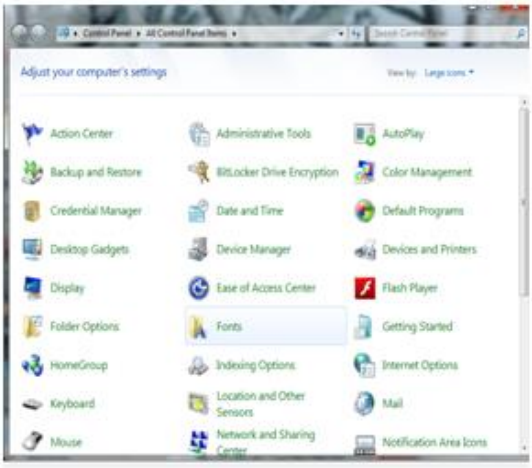



Libraries
Organize
New library
Favorites
Desktop
Downloads
Recent Places
Libraries
Homegroup
Computer
Local Disk (C:)
Local Disk (D:)
Local Disk (E:)
Network
Libraries
Open a library to see your files and arrange them by folder, date, and other properties.
Documents
Library
Music
Library
Pictures
Library
Videos
Library
4 items

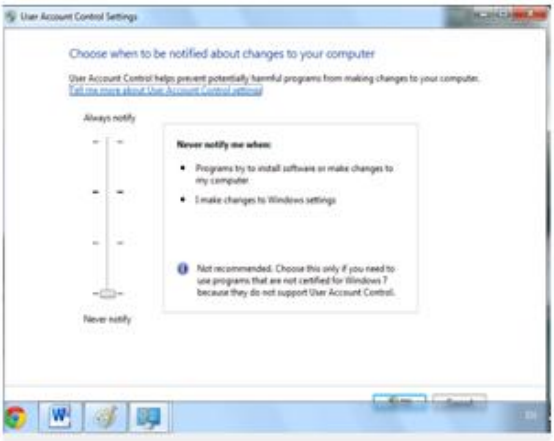
ნახ.44


Microsoft Word 2010
Skype
Folder - Shortcut
Paint
CCleaner
Microsoft Excel 2010
Notepad
IrfanView
Calculator
Getting Started
All Programs
Search programs and files
Pictures
Music
Games
Computer
Control Panel
Devices and Printers
Default Programs
Help and Support

ნახ.45


Adjust your computer's settings
View by: Large icons
Action Center
Administrative Tools
AutoPlay
Backup and Restore
BitLocker Drive Encryption
Color Management
Credential Manager
Date and Time
Default Programs
Desktop Gadgets
Device Manager
Devices and Printers
Display
Ease of Access Center
Flash Player
Folder Options
Fonts
Getting Started
HomeGroup
Indexing Options
Internet Options
Keyboard
Location and Other Sensors
Mail
Mouse
Network and Sharing Center
Notification Area Icons

ნახ.46


User Account Control Settings
Choose when to be notified about changes to your computer
User Account Control helps prevent potentially harmful programs from making changes to your computer.
Tell me more about User Account Control settings
Always notify
Never notify
Never notify me when:
Programs try to install software or make changes to my computer
I make changes to Windows settings
Not recommended. Choose this only if you need to use programs that are not certified for Windows 7 because they do not support User Account Control.

ნახ.47

განმარტებები

№	ხალი ცნება	შინაარსი
1	2	3
1	გამოთვლითი სისტემა	გამოთვლითი მანქანის აპარატურულ და პროგრამულ საშუალებათა ერთობლიობა, რომლებიც ურთიერთქმედებენ ინფორმაციის დამუშავების ამოცანის გადაწყვეტის პროცესში.
2	მეინფრეიმი	დიდი სიმძლავრის უნივერსალური გამომთვლელი მანქანა. ჩვეულებრივ გამოიყენება ერთდროულად რამოდენიმე მომხმარებლის მიერ, რომლებიც მუშაობენ მანქანასთან მიერთებულ ტერმინალებზე.
3	პროტოკოლი	წესების ერთობლიობა, რომელიც განსაზღვრავს მოწყობილობების, პროგრამების, მონაცემთა დამუშავების სისტემების, პროცესების ან მომხმარებლის ურთიერთქმედების ალგორითმს.
4	ინტერფეისი	მოწყობილობებისა და პროგრამების მომხმარებელთან ან ერთმანეთს შორის ურთიერთქმედების წესების ერთობლიობა და საშუალებები, რომლებიც ახორციელებენ ამ ურთიერთქმედებას. ინტერფეისის ცნება შეიცავს როგორც აპარატურულ და პროგრამულ საშუალებებს, რომელიც აკავშირებს სხვადასხვა პროგრამებს ან მოწყობილობებს ერთმენეთს შორის, ასევე იმ წესებსა და ალგორითმებს, რომელთა საფუძველზეც ეს საშუალებებია შექმნილი.

5	ოპერაციული სისტემა	პროგრამების კომპლექსი, რომელიც ორგანიზებას უკეთებს გამოთვლით პროცესს გამოთვლით სისტემებში.
6	შესრულებადი ფაილი	ფაილი, რომელიც მზადაა ოპერაციული სისტემის მიერ შესასრულებლად.
7	რესურსი	დრო, აპარატურული, პროგრამული და სხვა საშუალებანი, რომელიც შეიძლება წარმოუდგინოს პროცესს ან მომხმარებელს გამოთვლითმა სისტემამ ან მისმა ცალკეულმა კომპონენტმა.
8	რესურსების დაგეგმვა	განსაზღვრა იმისა, თუ რომელ პროცესს, როდის ან რა რაოდენობით (თუ რესურსი შეიძლება დაიყოს ნაწილებად) უნდა გამოეყოს მოცემული რესურსი.
9	რესურსების გამოყენების აღრიცხვა	რესურსების დაკავებულობასა და რესურსების წილის განაწილებაზე ოპერატიული ინფორმაციის მხარდაჭერა.
10	ოპერაციული სისტემის ქვესისტემა	(1)ს – ის მოდულები, რომლებიც დაჯგუფებულნი არიან ან იმ ლოკალური რესურსების ტიპებთან შესაბამისობაში, რომლებსაც (1)ს მართავს, ან იმ სპეციფიკური ამოცანების შესაბამისობაში, რომლებიც გამოყენებულნი არიან სისტემის ყველა რესურსების მიერ.
11	პროცესი	პროგრამის ან მისი ნაწილის შესრულებისას ოპერაციათა თანმიმდევრობა გამოყენებულ რესურსებთან ერთად.
12	სამომხმარებლო პროცესი	პროცესი, რომელიც აღიძვრება მომხმარებლის ინიციატივით.

13	სისტემური პროცესი	პროცესი, რომელიც ალიძერება ოპერაციული სისტემის ინიციატივით საკუთარი ფუნქციის შესრულებისას.
14	პროცესებსშორისი ურთიერთქმედების საშუალება	(ᵓ)ს – ის შემადგენლობაში შემავალი საშუალება პროცესების ურთიერთქმედების ოპერატიული შესაძლებლობის განხორციელებისათვის.
15	მეხსიერების დაცვა	შესაძლებლობა, დაიცვას შესრულებადი ამოცანა სხვა ამოცანისათვის განკუთვნილი მეხსიერების ჩაწერა – წაკითხვის ოპერაციისაგან.
16	მოწყობილობის დრაივერი	პროგრამა, რომელიც მართავს კონკრეტული მოდელის გარე მოწყობილობას და ითვალისწინებს ყველა მის თავისებურებებს.
17	ოპერატიული სისტემის ბრძანება	მანქანური ენის ელემენტარული წინადადება, რომელიც შეიცავს კომპიუტერის მიმართ მითითებას, შეასრულოს რომელიმე მანქანური ოპერაცია ან მოქმედება.
18	ბრძანებითი სტრიქონი	მონიტორის ეკრანის სტრიქონი, რომელიც განკუთვნილია მომხმარებლის მიერ შესატანი (ᵓ)ს – ის ბრძანების ჩაწერისათვის.
19	ბრძანებითი ფაილი	ფაილი, რომელიც შეიცავს ოპერაციული სისტემის მართვის ბრძანებათა თანმიმდევრობას.
20	ბრძანებითი ინტერპრეტატორი	(ᵓ)ს – ის პროგრამული მოდული, რომელიც პასუხისმგებელია ცალკეული ბრძანების ან ბრძანებათა თანმიმდევრობის წაკითხვაზე ბრძანებითი ფაილიდან.
21	სისტემის რეაქციის დრო	პროგრამის გაშვებასა და რეზულტატის მიღებას შორის დროის ინტერვალი.

22	გაფართოებადი ოპერაციული სისტემა	(ა)ს, რომელშიც შეიძლება შევიტანოთ დამატება ან ცვლილება მისი მთლიანობის დარღვევის გარეშე.
23	ოპერაციული სისტემის არქიტექტურა	(ა)ს – ის სტრუქტურული ორგანიზაცია სხვადასხვა პროგრამული მოდულების საფუძველზე.
24	ოპერაციული სისტემის ბირთვი	მოდულები, რომლებიც ასრულებენ (ა)ს – ის ძირითად ფუნქციებს.
25	სისტემური გამოძახება	(ა)ს – ის ბირთვისადმი დანართის მოთხოვნა.
26	გამოყენებითი პროგრამების ინტერფეისი	ბირთვის ფუნქციები, რომლებიც შეიძლება გამოძახებულნი იქნან დანართების მიერ.
27	ოპერაციული სისტემის რეზიდენტული მოდული	(ა)ს – ის მოდული, რომელიც მუდმივად იმყოფება მეხსიერებაში.
28	უტილიტა	პროგრამა, რომელიც წყვეტს გამოთვლითი სისტემის მართვისა და თანხლების ცალკეულ ამოცანებს.
29	ოპერაციული სისტემის ტრანზიტული მოდული	(ა)ს – ის მოდული, რომელიც იტვირთება ოპერატიულ მეხსიერებაში მხოლოდ საკუთარი ფუნქციის შესრულების დროს.
30	მუშაობის სამომხმარებლო რეჟიმი	კომპიუტერის აპარატურის მუშაობის რეჟიმი, რომლის დროსაც აკრძალულია რამოდენიმე კრიტიკული ბრძანების შესრულება.
31	პრივილეგირებული რეჟიმი	აპარატურის მუშაობის რეჟიმი, რომლის დროსაც ნებადართულია ნებისმიერი ბრძანების შესრულება.

32	შრეებსშორისი ინტერფეისი	ფუნქციათა ნაკრები, რომელსაც წარმოუდგენს (ა)ს – ის შრეების იერარქიაში ქვემოთმდებარე შრე ზემოთმდგომს.
33	ოპერაციული სისტემის მანქანურ – დამოკიდებული კომპონენტები	პროგრამული მოდულები, რომლებიც ასახავენ კომპიუტერის აპარატურული პლათფორმის სპეციფიკას.
34	რესურსების მენეჯერი (დისპეჩერი)	(ა)ს – ის რთული ფუნქციონალური მოდული, რომელიც აწარმოებს განსაზღვრული ტიპის რესურსების დაგეგმვას და განაწილების აღრიცხვას.
35	ოპერაციული სისტემის სერვერი	სამომხმარებლო რეჟიმში მომუშავე რესურსების მენეჯერი, რომელიც მომსახურებას უწევს ლოკალური დანართებისა და (ა)ს – ის სხვა მოდულების მოთხოვნებს.
36	Windows NT executive	Windows NT ოპერაციული სისტემის ნაწილი, რომელიც მუშაობს ბირთვის რეჟიმში.
37	Windows NT – ს დაცული ქვესისტემები	Windows NT ოპერაციული სისტემის სერვერები, რომლებიც მუშაობენ სამომხმარებლო რეჟიმში.
38	წყვეტა	პროგრამის ბრძანების შესრულების დროებითი შეწყვეტა, მის მიმდინარე მდგომარეობაზე ინფორმაციის შენახვით და მართვის გადაცემა სპეციალურ პროგრამაზე – წყვეტის დამუშავება.
39	BIOS (Basic Input – Output System)	კომპიუტერის შეტანა – გამოტანის ბაზური სისტემა, რომელიც შეიცავს დრაივერებს ბაზურ კონფიგურაციაში შემავალი ყველა მოწყობილობებისათვის.

40	გადატანითი ოპერაციული სისტემა	ოს, რომლის კოდი შეიძლება შედარებით იოლად იქნას გადატანილი ერთი ტიპის აპარატურული პლათფორმიდან სხვა ტიპის აპარატურულ პლათფორმაზე.
41	ოპერაციული სისტემის თავსებადობა	ოს – ის თვისება, რომლითაც იგი უზრუნველყოფს სხვა ოს – ისათვის დაწერილი დანართის შესრულებას.
42	ემულიატორი	სპეციალური პროგრამული უზრუნველყოფა, რომელიც კომპიუტერს საშუალებას აძლევს ინტერპრეტაცია გაუკეთოს იმ მანქანურ ინსტრუქციებს, რომლებიც დაწერილნი არიან განსხვავებული პროცესორული არქიტექტურის მანქანებისათვის.