

- ჩამოთვლილთაგან რომელი შეესაბამება სისტემაში არსებული (data.in) ფაილის კითხვის უფლებით გახსნის მიზნით open სისტემური გამოძახების გამოყენების სწორ ფორმას
`int fd = open("data.in", O_RDONLY);`
- რა ეწოდება აბსტრაქციას, რომელიც გამოთვლით სისტემაში ნაკლები მოცულობის ფიზიკური მეხსიერების არსებობის შემთხვევაში ხელს უწყობს მეტი ფიზიკური მეხსიერების საჭიროების მქონე პროგრამის ამუშავებას
ვირტუალური მეხსიერება
- აღნერეთ რაში მდგომარეობს ურთიერთგამორიცხვის ალგორითმის, „ცვლადი-ბოქსლომი“ არსი და მისი გამოყენების უარყოფითი მხარეები.
- ეს ხელით საწერი იყო. **PDF-ში შეიძლება ნახვა**
- ჩამოთვლილთაგან რომელი წარმოადგენს ენერგო-დამოუკიდებელ მეხსიერებას?
Solid State Disk (SSD)
- რა ეწოდება კომპონენტს, რომელიც ფიზიკურ დონეზე ორ ფიზიკურ კომპონენტს შორის უზრუნველყოფს მონაცემების გადაცემას
სალტე

ვთქვათ ოპერაციულ სისტემაში რესურსები წარმოდგენილია მრავალი ეგზემპლარის სახით. განვიხილოთ სურათზე ნაჩვენები შემთხვევა, სადაც

A - შეესაბამება პროცესებისთვის უკვე გამოყოფილ რესურსებს

R - შეესაბამება მიერ დამატებით მოთხოვნილ რესურსებს

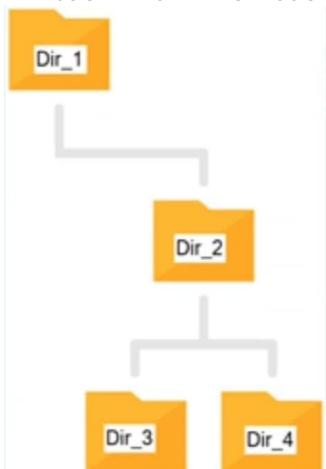
F - შეესაბამება მიმდინარე მომენტში თავისუფალი რესურსების საერთო რაოდენობას

$$A = (1 \ 0 \ 1 \ 2 \ 1 \ 0 \ 0 \ 1 \ 0 \quad 0 \ 1 \ 1 \ 0 \ 1 \ 0) \quad R = (2 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \quad 1 \ 0 \ 1 \ 1 \ 0 \ 0) \quad F = (1 \ 0 \ 1 \ 0 \ 0)$$

იძლევა თუ არა რესურსებზე მოთხოვნათა მიმდევრობა საიმედო მდგომარეობას?

False

- ჩამოთვლილთაგან კოდის რომელი ფრაგმენტი იძლევა სურათზე ნაჩვენები იერარქიის შესაბამისი დირექტორიების სტრუქტურის მიღების საშუალებას



`mkdir Dir_1 Dir_1/Dir_2 Dir_1/Dir_2/Dir_4 Dir_1/Dir_2/Dir_3`

- რა ეწოდება პროგრამულ უზრუნველყოფას, რომელიც ოპერაციულ სისტემაში ამა თუ იმ დონეზე ხელს უწყობს ფიზიკური ან აბსტრაქტული კომპონენტის ნორმალურ ფუნქციონირებას
დრაივები
- ვთქვათ კომპიუტერის მეხსიერება დაყოფილია ფიქსირებული ზომის ბლოკებად. დავუშვათ, რომ კომპიუტერში გვაქს მხოლოდ ისეთი პროგრამები, რომლის კოდისა და მონაცემების განთავსება არის შესაძლებელი მეხსიერების ერთ ბლოკში და მხოლოდ ერთი პროგრამა შეიძლება განთავსებულ იქნას ერთ ბლოკში. ასეთ შემთხვევაში ჩამოთვლილთაგან რომელი ტიპის პრობლემას შეიძლება ჰქონდეს ადგილი?
- შიდა ფრაგმენტაციის პრობლემა

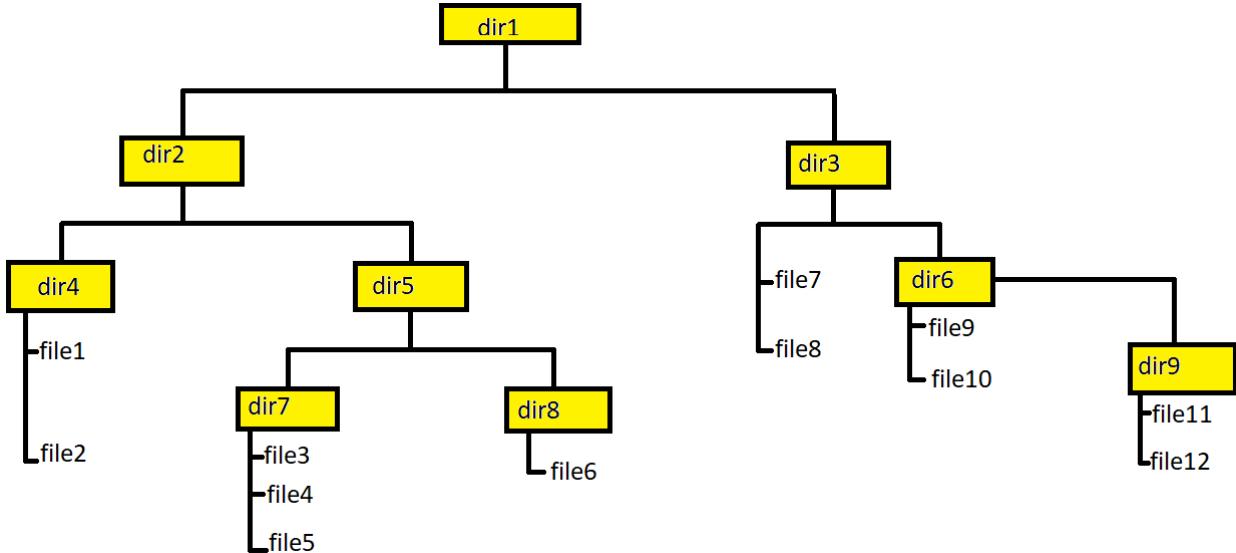
9. როგორც ცნობილია მეცნიერებმა დაადგინეს ურთიერთბლოკირების წარმოქმნის აუცილებელი და საკმარისი პირობები. ჩამოთვლილთაგან რომელი შესაბამება პირობას „ურთიერთგამორიცხვა“ დროის ნებისმიერ მომენტში რესურსი ან გამოყოფილია მხოლოდ ერთი პროცესისთვის ან თავისუფალია
10. თუ მომხმარებელმა გადაწყვიტა ოპერაციულ სისტემის სწრაფებულების ამაღლების მიზნით პროცესების გარკვეული ჯგუფი იძულებით გააჩეროს („დააპაუზოს“), მაშინ შესაბამისი პროცესები აღმოჩნდებიან მდგომარეობაში
- შეჩერებული მზად**
11. ვთქვათ ოპერაციულ სისტემაში პროგრამისთვის მეხსიერების გამოყოფა ხდება მისამართების უწყვეტი მიმდევრობით.
- თუ ცნობილია, რომ პროგრამა მოცულობა არის 19,3 MB (შეიძლება 19.3 ეტეროს). რამდენი ფურცელი იქნება საჭირო პროგრამის მონაცემების შესანახად, თუ ფურცლის ზომა არის 24 კბ (24576 ბაიტი). (შენიშვნა. პროგრამისთვის გამოყოფილ ყველა ფურცელზე გარდა ბოლო ფურცლისა მონაცემების განსათავსებლად გამოყენებული იქნება ყველა ბაიტი.)
- 824**
12. როგორ პროცესს ეწოდება ზომბი
- პროცესს, თუ ის დასრულების შემდეგ გარკვეული დროით დარჩა სისტემაში**
13. აღნერეთ რაში მდგომარეობს ურთიერთგამორიცხვის ალგორითმის „წყვეტის აკრძალვა“ არსი და მისი გამოყენების უარყოფითი მხარეები.
- ხელით საწერია, PDF-ში შეიძლება ნახვა**
14. - r - xrw - - wx დაშვების უფლებების სიმბოლური ჩანაწერის შესაბამის რვაობით ჩანაწერს ექნება სახე: **0563**
15. ვთქვათ, სისტემაში პროცესები წარმოდგენილია ქვემოთ მოყვანილი ცხრილის მეშვეობით. გაუძევებელი SJF ალგორითმის გამოყენებით იპოვეთ P_4 და P_{10} პროცესების შესრულების საერთო დროის საშუალო არითმეტიკული

პროცესები	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
შესრულების დრო	1	7	3	2	4	3	2	1	6	3

მე-4 პროცესის შესრულების დროა 2

მე-10 პროცესის შესრულების დროა 15

16. ჩამოთვლილთაგან რომელ მდგომარეობაში მოხდება პროცესის გადაყვანა თუ მისი შესრულების მომენტში წარმოშობილი წყვეტის გამო პროცესორი გადაერთო წყვეტის დამუშავებაზე და ამ პერიოდში პროცესს ამონტურა დროითი კვანტი
- მზადყოფნის**
17. განსაკუთრებული შემთხვევის რომელ ტიპს მიეკუთვნება პროგრამის შესრულების მიმდინარე მომენტში ისეთ ფაილზე მიმართა, რომელიც ფაილურ სისტემაში არსებობს, მაგრამ სხვა დირექტორიაში არის განთავსებული
- გამოუსწორებელი განსაკუთრებული შემთხვევა**
18. ჩამოთვლილთაგან რომელ დონეს მიეკუთვნება ამოცანა, რომელიც ელოდება შესრულებას **ზედა დონე**



19.

ვთქვათ მოცემულია სურათზე ნაჩვენები იერარქია. დაწერეთ ტერმინალში შესასრულებელი ბრძანება, რომელიც (dir8 დირექტორიის მიმართ) მიმართებითი სახელის გამოყენებით file4-ს შეუცვლის მომხმარებლის ჯგუფს root-იდან user-ზე chfrp group ..//dir4 ??

20. პროგრამული კოდის შემუშავებისას პროგრამისტს დასჭირდა 100 მთელმნიშვნელობიანი მასივის განთავსება განაწილებად მეხსიერებაში. ქვემოთ ჩამოთვლილთაგან პროგრამული კოდის რომელი ფრაგმენტითაა შესაძლებელი (key გასაღებისთვის) არსებული განაწილებადი მეხსიერების გამოყენება? int shmid = shmget(key, 100*sizeof(int), 0);
21. ვთქვათ ოპერაციულ სისტემაში პროგრამისთვის მეხსიერების გამოყოფა ხდება მისამართების უწყვეტი მიმდევრობით. დავუშვათ, ოპერაციულ სისტემაში გამოყენებული ფურცლის ზომა არის 8 K (8*1024 ბაიტი).
- რისი ტოლი იქნება პროგრამისთვის გამოყოფილ ბოლო ფურცელზე ბაიტების რაოდენობა, თუ ცნობილია პროგრამის მოცულობა არის 493621 ბაიტი და ის სრულად იკავებს 60 ფურცელს.

1. ჩამოთვლილთაგან არცერთი

2. 2101

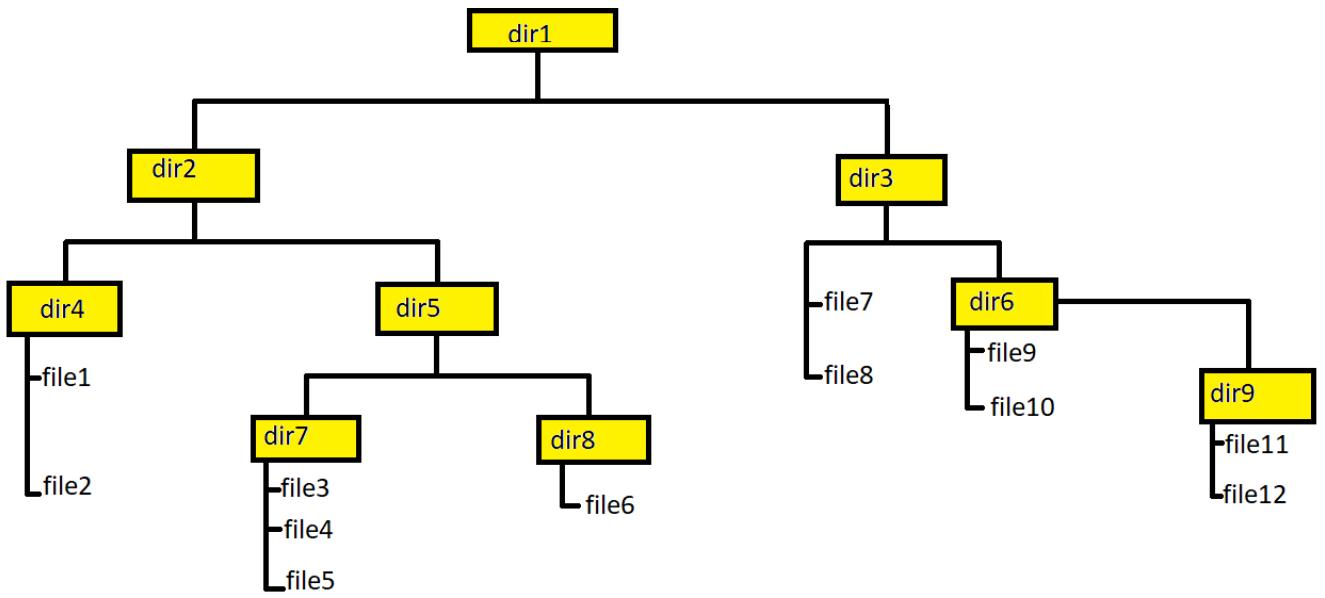
3. 1157

4. 2300

5. 1560

22. ჩამოთვლილთაგან რომელი საქმიანობითაა დაკავებული ბრძანებათა მთვლელი.
1. ითვლის პროცესების მიერ მიმდინარე მომენტის შემდეგ შესასრულებელი ბრძანებების რაოდენობას
 2. ითვლის პროცესების მიერ შესასრულებელი ბრძანებების საერთო რაოდენობას
 - 3. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი**
 4. ითვლის პროცესის მიერ მიმდინარე მომენტამდე შესრულებული ბრძანებების რაოდენობას
23. წყვეტის რომელ ტიპს მიეკუთვნება კოდში დაშვებული შეცდომის გამო წარმოქმნილი წყვეტა **სინქრონულ წყვეტას**
24. ვთქვათ ოპერაციულ სისტემაში პროგრამისთვის მეხსიერების გამოყოფა ხდება მისამართების უწყვეტი მიმდევრობით. დავუშვათ, ოპერაციულ სისტემაში გამოყენებული ფურცლის ზომა არის 24 K, ხოლო ფურცლების ბლოკი შეიცავს 24 K (24*1024) ფურცელს.
- თუ ცნობილია, რომ პროგრამა მოცულობა არის 8.9 GB, ხოლო მისთვის გამოყოფილი ფურცლების რაოდენობა არის 388847, მაშინ რამდენი ბლოკი გამოყოფა მოუწევს ოპერაციულ სისტემას პროგრამისთვის.
- (შენიშვნა. იგულისხმება პროგრამის მიერ დაკავებული ყველა ბლოკი, მიუხედავად იმისა ყველა ფურცელი ბლოკში არის თუ არა გამოყოფილი ერთიდამავე პროგრამისთვის)

25. ვთქვათ ოპერაციულ სისტემაში პროგრამისთვის მეხსიერების გამოყოფა ხდება მისამართების უწყვეტი მიმდევრობით.
- თუ ცნობილია, რომ პროგრამა მოცულობა არის 27.8 MB. რამდენი ფურცელი იქნება საჭირო პროგრამის მონაცემების შესანახად, თუ ფურცლის ზომა არის 32 კბ (32768 ბაიტი).
- (შენიშვნა. პროგრამისთვის გამოყოფილ ყველა ფურცელზე გარდა ბოლო ფურცლისა მონაცემების განსათავსებლად გამოყენებული იქნება ყველა ბაიტი.)
- 890**
26. ჩამოთვლილთაგან რომელი ფუნქცია გამოიყენება ნაკადის (thread-ის) იდენტიფიკატორის მნიშვნელობის მისაღებად?
- pthread_self()**
27. ვთქვათ ოპერაციულ სისტემაში პროგრამისთვის მეხსიერების გამოყოფა ხდება მისამართების უწყვეტი მიმდევრობით. დავუშვათ, ოპერაციულ სისტემაში გამოყენებული ფურცლების ზომა არის 8 K ($8 * 1024$ ბაიტი).
- რისი ტოლი იქნება პროგრამისთვის გამოყოფილ ბოლო ფურცელზე ბაიტების რაოდენობა, თუ ცნობილია პროგრამის მოცულობა არის 574695 ბაიტი და ის სრულად იკავებს 70 ფურცელს.
- 1255**
28. ვთქვათ ოპერაციულ სისტემაში პროგრამისთვის მეხსიერების გამოყოფა ხდება მისამართების უწყვეტი მიმდევრობით.
- დავუშვათ, ოპერაციულ სისტემაში გამოყენებული ფურცლების ზომა არის 24 K, ხოლო ფურცლების ბლოკი შეიცავს 4 K ($4 * 1024$) ფურცელს.
- თუ ცნობილია, რომ პროგრამა მოცულობა არის 2.53 GB, ხოლო მისთვის გამოყოფილი ფურცლების რაოდენობა არის 110538, მაშინ რამდენი ბლოკი გამოყოფა მოუწევს ოპერაციულ სისტემას პროგრამისთვის.
- (შენიშვნა. იგულისხმება პროგრამის მიერ დაკავებული ყველა ბლოკი, მიუხედავად იმისა ყველა ფურცელი ბლოკში არის თუ არა გამოყოფილი ერთიდამავე პროგრამისთვის)
- 27**
29. კავშირის არხის დასრულების პროცესირება რა შემთხვევაში არ არის აუცილებელი
- თუ პროცესებს შორის კავშირის არხის გახსნა არ იყო პროცესირებული სპეციალური მოქმედებით**
30. ვთქვათ ოპერაციულ სისტემაში პროგრამისთვის მეხსიერების გამოყოფა ხდება მისამართების უწყვეტი მიმდევრობით. დავუშვათ, ოპერაციულ სისტემაში გამოყენებული ფურცლების ზომა არის 4 K ($4 * 1024$ ბაიტი).
- რისი ტოლი იქნება პროგრამის მოცულობა, თუ ცნობილია, რომ პროგრამის განთავსება იწყება პირველივე მისამართიდან (0) და პროგრამის მიერ დაკავებული ბოლო ბაიტის მისამართია (14, 1005), სადაც წყვილის პირველი მნიშვნელობა (14) შეესაბამება ბოლო ფურცლის ნომერს, ხოლო მეორე მნიშვნელობა (1005) კი - ამავე ფურცელზე პროგრამის მიერ გამოყენებულ ბოლო ბაიტის მისამართს.
- (შენიშვნა. როგორც ფურცლების ისე ფურცლებში ბაიტების მისამართები ფურცლებში იწყება 0-დან.)
1. 52804
 2. ჩამოთვლილთაგან არცერთი
 3. 60735
 4. 55837
 5. 50590
31. ჩამოთვლილთაგან რომელ მონაცემს არ შეიცავს სუპერბლოკი?
- დაკავებულ (მონაცემების შემცველ) ბლოკებზე ინფორმაციას**



32.

ვთქვათ `dir2` არის სამუშაო დირექტორია. `dir4` დირექტორიის `dir7` დირექტორიაში გადაადგილებისთვის (cut-paste) საჭიროა ტერმინალში ბრძანება შესრულდეს შემდეგი სახით
`mv dir4 dir5/dir7`

33. ჩამოთვლილთაგან რომელი ბრძანებით არის შესაძლებელი ფაილზე მომხმარებლის ჯგუფის შეცვლა

1. ჩამოთვლილთაგან არცერთი

2. chown()
3. mkfifo()
4. chmod()
5. mknod()

34. ჩამოთვლილთაგან რომელი შეესაბამება fifo არხის შექმნის სწორ ფორმას

`char name[] = "a fifo";`

`mknod(name, S_IFIFO | 0664, 0);`

35. ვთქვათ ოპერაციულ სისტემაში პროგრამისთვის მეხსიერების გამოყოფა ხდება მისამართების უწყვეტი მიმდევრობით.

თუ ცნობილია, რომ პროგრამა მოცულობა არის 6.57 MB. რამდენი ფურცელი იქნება საჭირო პროგრამის მონაცემების შესანახად, თუ ფურცლის ზომა არის 12 კბ (12288 ბაიტი).

(შენიშვნა. პროგრამისთვის გამოყოფილ ყველა ფურცელზე გარდა ბოლო ფურცლისა მონაცემების განსათავსებლად გამოყენებული იქნება ყველა ბაიტი.)

1. 525
2. 600
3. ჩამოთვლილთაგან არცერთი
4. 561
5. 610

36. რა ეწოდება კომპონენტს, რომელიც შედგება მიკროსქემის ან მიკროსქემების ნაკრებისაგან და ფიზიკურ დონეზე მართავს შესაბამის მოწყობილობას

კონტროლერი

37. რა ეწოდება მექანიზმს, რომელსაც მიმართავს სამომხმარებლო პროგრამა პრივილეგირებული

მოქმედების განხორციელების მიზნით

1. პროცესი
2. ვირტუალური მეხსიერება
3. ბირთვი
- 4. სისტემური გამოძხევა**
5. წყვეტა

38. თქვათ ოპერაციულ სისტემაში პროგრამებისთვის მეხსიერების გამოყოფა ხდება მისამართების უწყვეტი მიმდევრობით.

დავუშვათ, ოპერაციულ სისტემაში გამოყენებული ფურცლის ზომა არის $10 K (10 * 1024)$ ბაიტი). რისი ტოლი იქნება პროგრამისთვის გამოყოფილ ბოლო ფურცელზე ბაიტების რაოდენობა, თუ ცნობილია პროგრამის მოცულობა არის 566142 ბაიტი და ის სრულად იკავებს 55 ფურცელს.

1. 2942

2. 3027

3. 2505

4. 3200

5. ჩამოთვლილთაგან არცერთი

39. ჩამოთვლილთაგან რომელი ფუნქცია გამოიყენება ნაკადის (thread-ის) შესაქმნელად?

pthread_create()

40. რა ენდება პროცესს, რომელიც საქმიანობის დასრულების შემდეგ გარკვეული დროით დარჩა სისტემაში.

ზომბი

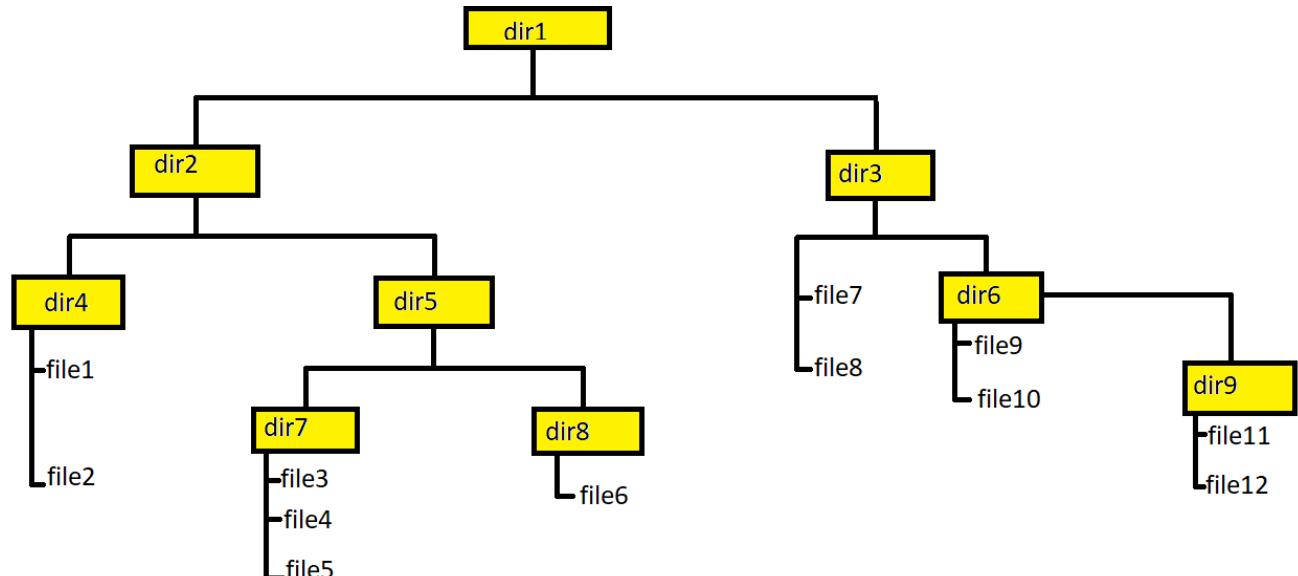
41. როგორ პროცესს ეწოდება დემონი.

პროცესს, თუ ის მუშაობს ფონურ რეჟიმში

42. ვთქვათ ოპერაციულ სისტემაში პროგრამისთვის მეხსიერების გამოყოფა ხდება მისამართების უწყვეტი მიმდევრობით. დავუშვათ, ოპერაციულ სისტემაში გამოყენებული ფურცლების ზომა არის $8 K (8 * 1024)$ ბაიტი).

რისი ტოლი იქნება პროგრამის მოცულობა, თუ ცნობილია, რომ პროგრამის განთავსება იწყება პირველივე მისამართიდან (0) და პროგრამის მიერ დაკავებული ბოლო ბაიტის მისამართია ($7, 5907$), სადაც წყვილის პირველი მნიშვნელობა (7) შეესაბამება ბოლო ფურცლის ნომერს, ხოლო მეორე მნიშვნელობა (5907) კი - ამავე ფურცელზე პროგრამის მიერ გამოყენებულ ბოლო ბაიტის მისამართს. (მენიშვნა. როგორც ფურცლების ისე ფურცლებში ბაიტების მისამართები ფურცლებში იწყება 0-დან.)

63251



43.

ვთქვათ $dir2$ არის სამუშაო დირექტორია. $dir4$ დირექტორიის $dir7$ დირექტორიაში გადაკოპირებისთვის (copy-paste) საჭიროა ტერმინალში ბრძანება შესრულდეს შემდეგი სახით

cp -r dir4 dir5/dir7

44. ვთქვათ ოპერაციულ სისტემაში პროგრამისთვის მეხსიერების გამოყოფა ხდება მისამართების უწყვეტი მიმდევრობით.

თუ ცნობილია, რომ პროგრამა მოცულობა არის $15.45 MB$. რამდენი ფურცელი იქნება საჭირო პროგრამის მონაცემების შესანახად, თუ ფურცლის ზომა არის $20 \text{ } \mu\text{B}$ (20480 ბაიტი).

(მენიშვნა. პროგრამისთვის გამოყოფილ ყველა ფურცელზე გარდა ბოლო ფურცლისა მონაცემების

განსათავსებლად გამოყენებული იქნება ყველა ბაიტი.)

792

45. ჩამოთვლილთაგან რომელი წინადადებაა ჭეშმარიტი pipe კავშირის არხთან მიმართებით.
1. pipe არხის გამოყენება შეუძლია სისტემაში წარმოქმნილ ყველა იმ პროცესს, რომელთა მეშვეობითაც წარმოქმნა pipe არხის წარმოქმნებული პროცესი
 2. pipe არხის გამოყენება შეუძლია სისტემაში წარმოქმნილ ყველა პროცესს მიუხედავად იმისა რომელი პროცესის მიერ მოხდა pipe არხის შექმნა
 - 3. ჩამოთვლილთაგან არცერთი**
 4. pipe არხის გამოყენება შეუძლია სისტემაში წარმოქმნილ ყველა იმ პროცესს, რომელთაც არ გააჩნიათ საერთო წარმოქმნელი
46. ვთქვათ ოპერაციულ სისტემაში პროგრამისთვის მეხსიერების გამოყოფა ხდება მისამართების უწყვეტი მიმდევრობით. დავუშვათ, ოპერაციულ სისტემაში გამოყენებული ფურცლის ზომა არის 24 K, ხოლო ფურცლების ბლოკი შეიცავს 8 K ($8 * 1024$) ფურცელს.
თუ ცნობილია, რომ პროგრამა მოცულობა არის 3.17 GB, ხოლო მისთვის გამოყოფილი ფურცლების რაოდენობა არის 138500, მაშინ რამდენი ბლოკი გამოყოფა მოუწევს ოპერაციულ სისტემას პროგრამისთვის.
(შენიშვნა. იგულისხმება პროგრამის მიერ დაკავებული ყველა ბლოკი, მიუხედავად იმისა ყველა ფურცელი ბლოკში არის თუ არა გამოყოფილი ერთიდაიმავე პროგრამისთვის)
1. 30
 2. ჩამოთვლილთაგან არცერთი
 3. 22
 4. 25
 - 5. 17**

ვთქვათ ოპერაციულ სისტემაში რესურსები წარმოდგენილია მრავალი ეგზემპლარის სახით. განვიხილოთ სურათზე
ნაჩვენები შემთხვევა, სადაც

A - შეესაბამება პროცესებისთვის უკვე გამოყოფილ რესურსებს

R - შეესაბამება პროცესის მიერ დამატებით მოთხოვნილ რესურსებს

F - შეესაბამება მიმდინარე მომენტში თავისუფალი რესურსების საერთო რაოდენობას

$$A = (1 \ 0 \ 1 \ 2 \ 1 \ 0 \ 0 \ 1 \ 0 \quad 0 \ 1 \ 1 \ 0 \ 1 \ 1) \quad R = (2 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \quad 1 \ 0 \ 1 \ 1 \ 0 \ 0) \quad F = (1 \ 0 \ 1 \ 0 \ 0)$$

იძლევა თუ არა რესურსებზე მოთხოვნათა მიმდევრობა საიმედო მდგომარეობას?

True

47. ფაილის ორგანიზაციის მიმდევრობითი მეთოდის გამოყენების შემთხვევაში შესანახ მოწყობილობაზე ფაილის მონაცემების განთავსება ხდება
მონაცემების ფიზიკური მიმდევრობის გათვალისწინებით
48. თუ პროცესების დაგეგმვა ხდება სტატიკური პრიორიტეტის გამოყენებით, მაშინ ოპერაციულ სისტემაში მზადყოფნის მდგომარეობაში მყოფი პროცესების დალაგება მოხდება
პროპრიტეტების ზრდადი მნიშვნელობების მიხედვით
49. ვთქვათ ოპერაციულ სისტემაში პროგრამისთვის მეხსიერების გამოყოფა ხდება მისამართების უწყვეტი მიმდევრობით. დავუშვათ, ოპერაციულ სისტემაში გამოყენებული ფურცლის ზომა არის 16 K, ხოლო ფურცლების ბლოკი შეიცავს 2 K ($2 * 1024$) ფურცელს.
თუ ცნობილია, რომ პროგრამა მოცულობა არის 1.28 GB, ხოლო მისთვის გამოყოფილი ფურცლების რაოდენობა არის 83887, მაშინ რამდენი ბლოკი გამოყოფა მოუწევს ოპერაციულ სისტემას პროგრამისთვის.
(შენიშვნა. იგულისხმება პროგრამის მიერ დაკავებული ყველა ბლოკი, მიუხედავად იმისა ყველა ფურცელი ბლოკში არის თუ არა გამოყოფილი ერთიდაიმავე პროგრამისთვის)
- 41**
50. ჩამოთვლილთაგან რომელ შემთხვევაში შეიძლება აღმოჩნდეს პროცესი მზადყოფნის მდგომარეობაში
1. ჩამოთვლილთაგან არცერთი
 2. თუ პროცესს დაბადების (სისტემაში გამოჩენის) შემდეგ მოთხოვნის შესაბამისად გამოეყო

პროცესორის გარდა შესასრულებლად საჭირო ყველა რესურსი

3. ჩამოთვლილთაგან ყველა სწორია (უარყოფითი პასუხის გარდა)

4. ბლოკირების მდგომარეობაში მყოფ პროცესს გამოყოფილი მისთვის საჭირო ყველა რესურსი

5. დროითი კვანტის ამონურვის გამო პროცესს ჩამოერთვა პროცესორი

51. `mknod` სისტემური გამოძახების გამოყენებით იქმნება
კავშირის არხი

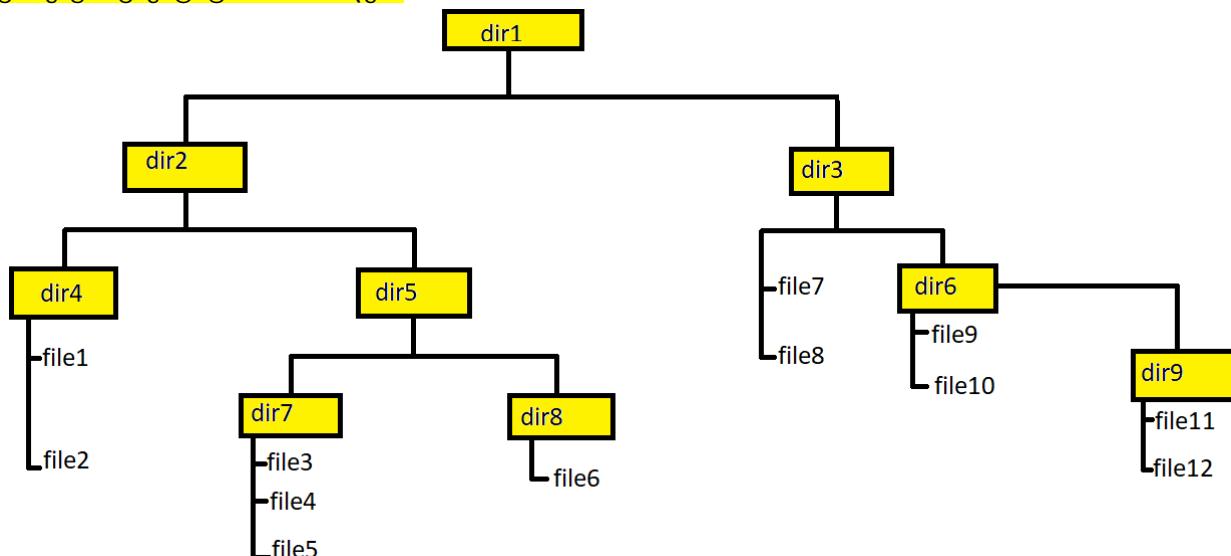
52. ჩამოთვლილთაგან რომელი შესაბამება `write` სისტემური გამოძახების გამოყენების სწორ ფორმას, თუ N
მასივში ელემენტების რაოდენობაა, ხოლო `fd` კი - შესაბამისი ნაკადის სახელი

`char buf[N];`

`write(fd, buf, N);`

53. ვთქვათ გამოთვლით სისტემაში პროგრამებისთვის მეხსიერების გამოყოფა ხორციელდება მისი
მოთხოვნილების შესაბამისა. თუ რომელიმე პროგრამა დასრულდა მისთვის ადრე გამოყოფილი
მისამართების სივრცე საკმარისი ერთი ან მეტი პროგრამის განთავსებისთვის, მაშინ აღნიშნული სივრცე
იყოფა პროგრამების მოთხოვნის შესაბამისად და ა.შ. მეხსიერების ამ ფორმით დანაწილების შემთხვევაში
ჩამოთვლილთაგან რომელი ტიპის პრობლემას შეიძლება ჰქონდეს ადგილი?

გარე ფრაგმენტაციის პრობლემა



54. ვთქვათ, მოცემულია სურათზე ნაჩვენები იერარქია.

დაწერეთ ტერმინალში შესასრულებელი ბრძანება, რომელიც (`dir4` დირექტორიის მიმართ) მიმართებითი
სახელის გამოყენებით `dir7` დირექტორიის შიგთავსზე ინფორმაციას ნაცვლად ტერმინალისა გამოიტანს
`dir4` დირექტორიის განთავსებულ `info.txt` ფაილში.

შენიშვნა. ბრძანება იწერება ერთ სტრიქონად, ხოლო `info.txt` ფაილი შეიქმნება ბრძანების
შესრულებასთან ერთად.

ხელითა საწერი

55. ვთქვათ სისტემაში გვაქს პროგრამა, რომლის კომპილაციის შედეგად კომპილატორი აგენტირებს
გადატანად კოდს. ასეთ შემთხვევაში ჩამოთვლილთაგან რომელ ეტაპზე იქნება შესაძლებელი იზიკური
და ლოგიკური მეხსიერების მისამართების დაკავშირება?

ჩატვირთვის ეტაპი

56. ჩამოთვლილთაგან რომელი წარმოადგენს გამოთვლით სისტემაში გამოყენებულ ყველაზე ნელ
ენერგოდამოკიდებულ მეხსიერებას?

ქეშ-მეხსიერება

57. ჩამოთვლილთაგან რომელი შესაბამება საქმიანობას რომლითაც დაკავებული ბრძანებათა მთვლელი
ინახავს კონკრეტული ერთი პროგრამის ამუშავების შემდეგ ამავე პროგრამის წარმატებული საქმიანობის
გაგრძელებისთვის რიგით შემდეგი შესასრულებელი ბრძანების მიმთითებელს

58. ჩამოთვლილთაგან რომელი სისტემური გამოძახებით იქმნება კავშირის არხი FIFO
`mknod()`

59. -rwxr-x-w- დაშვების უფლებების სიმბოლური ჩანაწერის შესაბამის რეალით ჩანაწერს ექნება სახე:

0752

60. ჩამოთვლილთაგან რომელი ბრძანების გამოყენებით არის შესაძლებელი ფაილზე დაშვების უფლებების შეცვლა.

1. mknod()

2. chown()

3. ჩამოთვლილთაგან არცერთი

4. umask()

5. chgrp()

61. განსაკუთრებული შემთხვევის რომელ ტიპს მიეკუთვნება პროგრამის შესრულების მიმდინარე მომენტში მეხსიერების ფურცლის არარსებობა

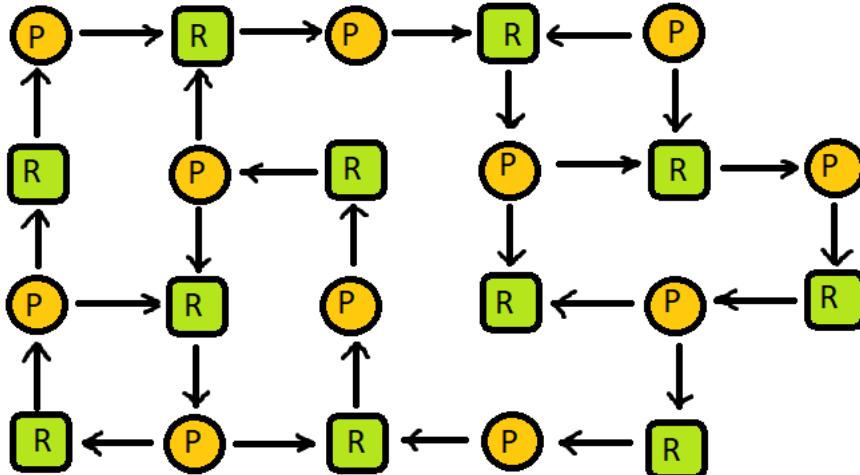
გამოსწორებადი განსაკუთრებული შემთხვევა

62. ჩამოთვლილთაგან რომელი ბრძანების გამოყენებით არის შესაძლებელი განაწილებადი მეხსიერებისთვის შემთხვევითი რიცხვითი მნიშვნელობის დაგენერირება

ftok()

63. სურათზე ნაჩვენები დიაგრამა შეესაბამება პროცესების მიერ რესურსებზე გაკეთებული მოთხოვნათა მიმდევრობის არასაიმედო მდგომარეობას. რამდენი ციკლი იკვრება ამ დიაგრამაზე.

(მენიშვნა. ერთიდაიგივე რესურსი ან პროცესი შეიძლება მონაწილეობდეს სხვა ციკლშიც. წრე აღნიშნავს პროცესს, ხოლო მართულთხედი კი რესურსს).



1. ჩამოთვლილთაგან არცერთი

2. 4 (ეს არაა სწორიო :/)

3. 3

4. 5

5. 2

64. ვთქვათ ოპერაციულ სისტემაში პროგრამისთვის მეხსიერების გამოყოფა ხდება მისამართების უწყვეტი მიმდევრობით.

თუ ცნობილია, რომ პროგრამა მოცულობა არის 17.6 MB. რამდენი ფურცელი იქნება საჭირო პროგრამის მონაცემების შესანახად, თუ ფურცლის ზომა არის 20 კბ (20480 ბაიტი).

(მენიშვნა. პროგრამისთვის გამოყოფილ ყველა ფურცელზე გარდა ბოლო ფურცლისა მონაცემების განსათავსებლად გამოყენებული იქნება ყველა ბაიტი.)

1. ჩამოთვლილთაგან არცერთი

2. 855

3. 820

4. 902

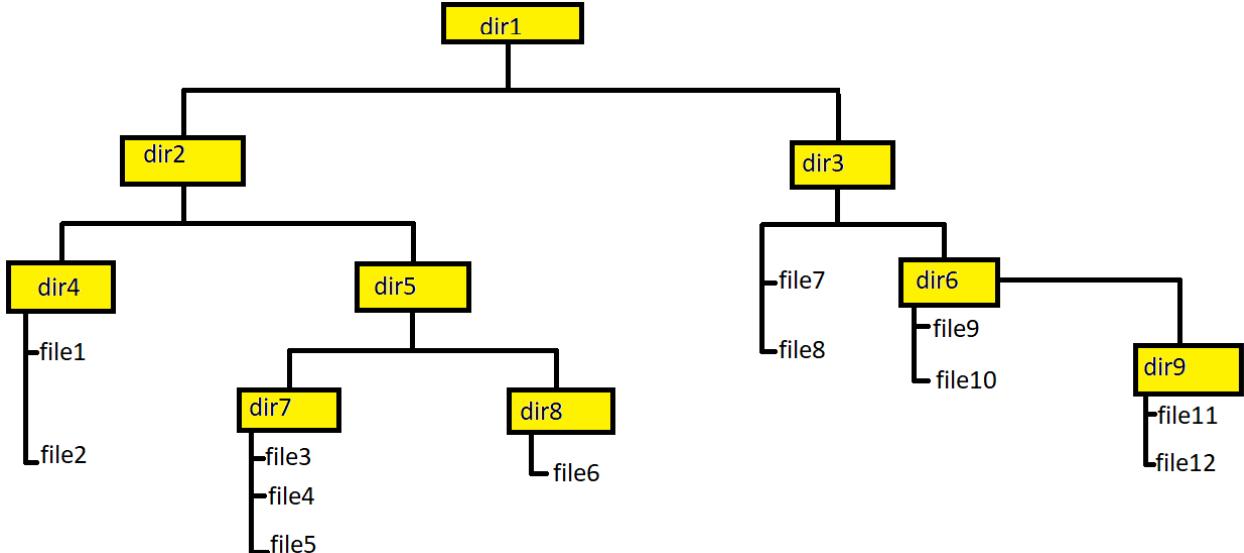
5. 890

65. ვთქვათ ერთპროცესორულ სისტემაში პროცესი იმყოფება შესრულების მდგომარეობაში.
ჩამოთვლილთაგან რომელი წინადადებაა ჭეშმარიტი.
შესრულების მდგომარეობაში მყოფი პროცესის საქმიანობის ბლოკირება შეუძლია მიმდინარე (იმავე) პროცესს
66. სისტემაში წარმოქმნილ და წარმატებით დასრულებულ ყოველ პროცესს მდგომარეობათა დიაგრამაზე გადაადგილება შეუძლია შემდეგი მიმდევრობით>
დაბადება □ მზადყოფნა □ შესრულება □ მზადყოფნა □ შესრულება □ დასრულება
67. ვთქვათ ოპერაციულ სისტემაში პროგრამისთვის მეხსიერების გამოყოფა ხდება მისამართების უწყვეტი მიმდევრობით. დავუშვათ, ოპერაციულ სისტემაში გამოყენებული ფურცლის ზომა არის 12 K (12×1024 ბაიტი).
რისი ტოლი იქნება პროგრამისთვის გამოყოფილ ბოლო ფურცელზე ბაიტების რაოდენობა, თუ ცნობილია პროგრამის მოცულობა არის 684695 ბაიტი და ის სრულად იკავებს 55 ფურცელს.
8855
68. ვთქვათ, სისტემაში პროცესები წარმოდგენილია ქვემოთ მოყვანილი ცხრილის მეშვეობით. გაძევებადი SJF ალგორითმის გამოყენებით იპოვეთ P_4 და P_7 პროცესების ლოდინის დროის მეოთხედი

პროცესები	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
შესრულების დრო	5	1	3	2	2	3	2	6	1	4

$$(3+7)/4=2.5$$

69. ვთქვათ ოპერაციულ სისტემაში პროგრამისთვის მეხსიერების გამოყოფა ხდება მისამართების უწყვეტი მიმდევრობით.
თუ ცნობილია, რომ პროგრამა მოცულობა არის 9.27 MB. რამდენი ფურცელი იქნება საჭირო პროგრამის მონაცემების შესანახად, თუ ფურცლის ზომა არის 12 კბ (12288 ბაიტი).
(შენიშვნა. პროგრამისთვის გამოყოფილ ყველა ფურცელზე გარდა ბოლო ფურცლისა მონაცემების განსათავსებლად გამოყენებული იქნება ყველა ბაიტი.)
1. 780
 2. 820
 - 3. ჩამოთვლილთაგან არცერთი**
 4. 800
 5. 750
70. ჩამოთვლილთაგან რომელი წარმოადგენს მონოლიტური არქიტექტურის ნაკლოვანებას?
1. ყველა პროცედურა ერთმანეთთან ურთიერთქმედებს მთავარი პროცედურის გამოყენებით
 - 2. ჩამოთვლილთაგან არცერთი არაა სწორი**
 3. პროცედურათა ნაკრების სახით შემუშავებული ოპერაციული სისტემის კოდი ნელა მუშაობს
 4. პროცედურებს არ შეუძლიათ ზეგავლენა იქონიონ სხვა პროცედურების საქმიანობაზე
71. რას ნიშნავს რომ რესურსი არაა განაწილებადი?
რესურსის ჩამორთმევით პროცესის ვერ შეძლებს საქმიანობის გავრძელებას
72. ჩამოთვლილთაგან რომელი ბრძანებით არის შესაძლებელი ფაილის შიგთავსის დათვალიერება მისი გახსნის გარეშე.
- cat**



73.

ვთქვათ მოცემულია სურათზე ნაჩვენები იერარქია.

დაწერეთ ტერმინალში შესასრულებელი პრანება, რომელიც (dir4 დირექტორიის მიმართ) მიმართებითი სახელის გამოყენებით მოგვცემს dir7 დირექტორიის შიგთავსის დათვალიერების საშუალებას.

`cd ./dir5/dir7 ???`

74. აღნერეთ რაში მდგომარეობს ურთიერთგამორიცხვის ალგორითმის, „მკაცრი მიმდევრობა“ არსი და მისი გამოყენების უარყოფითი მხარეები.

ხელით საწერია.

75. ჩამოთვლილთაგან რომელი სისტემური გამოძახება გამოიყენება მომხმარებლის ჯგუფის იდენტიფიკატორის მნიშვნელობის მისაღებად.

getgid()

76. ჩამოთვლილთაგან რომელი წარმოადგენს SSD დისკის ნაკლოვანებას HDD დისკთან მიმართებით **SSD დისკს გააჩინა სიცოცხლის ნაკლები ხანგრძლივობა ვიდრე HDD დისკს**

77. ვთქვათ ოპერაციულ სისტემაში პროგრამისთვის მეხსიერების გამოყოფა ხდება მისამართების უწყვეტი მიმდევრობით.

თუ ცნობილია, რომ პროგრამა მოცულობა არის 8.7 MB. რამდენი ფურცელი იქნება საჭირო პროგრამის მონაცემების შესანახად, თუ ფურცლის ზომა არის 4 კბ (16384 ბაიტი).

(შენიშვნა. პროგრამისთვის გამოყოფილ ყველა ფურცელზე გარდა ბოლო ფურცლისა მონაცემების განსათავსებლად გამოყენებული იქნება ყველა ბაიტი.)

1. 557

2. 525

3. 610

4. 593

5. ჩამოთვლილთაგან არცერთი

78. ქვემოთ ჩამოთვლილთაგან რომელი სისტემური გამოძახება აპრუნებს ფაილურ დესკრიპტორს? **open()**

79. ჩამოთვლილთაგან რომელი ბრძანების გამოყენებით არის შესაძლებელი ფაილზე მომხმარებლის შეცვლა

1. mknod()

2. chmod()

3. chgrp()

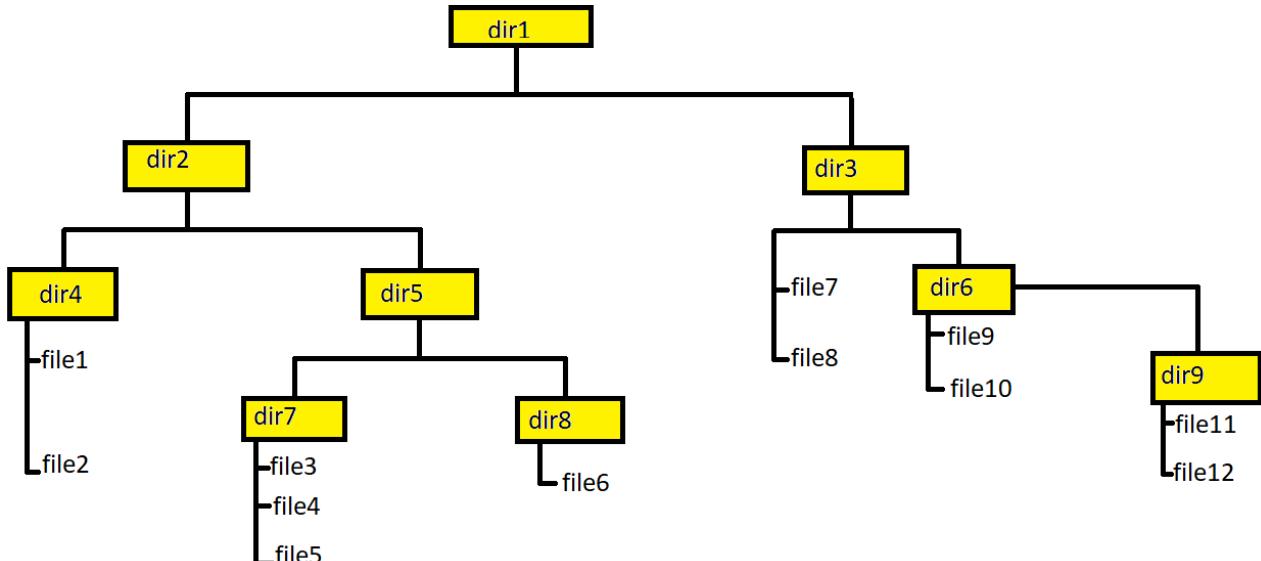
4. ჩამოთვლილთაგან არცერთი

5. chown()

80. ვთქვათ, სისტემაში პროცესები წარმოდგენილია ქვემოთ მოყვანილი ცხრილის მეშვეობით. FCFS ალგორითმის გამოყენებით იპოვეთ P_5 და P_9 პროცესების შესრულების საერთო დროის ნახევარი

პროცესები	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
შესრულების დრო	2	5	3	7	2	6	2	4	1	5

81. სტანდარტულად რამდენი კომპონენტისგან შედგება შეტანა/გამოტანის მოწყობილობა
2
82. ვთქვათ ოპერაციულ სისტემაში პროგრამისთვის მეხსიერების გამოყოფა ხდება მისამართების უწყვეტი მიმდევრობით. დავუშვათ, ოპერაციულ სისტემაში გამოყენებული ფურცლების ზომა არის 4 K ($4 * 1024$ ბაიტი).
- რისი ტოლი იქნება პროგრამის მოცულობა, თუ ცნობილია, რომ პროგრამის განთავსება იწყება პირველივე მისამართიდან (0) და პროგრამის მიერ დაკავებული ბოლო ბაიტის მისამართია (12, 3651), სადაც წყვილის პირველი მნიშვნელობა (12) შეესაბამება ბოლო ფურცლის ნომერს, ხოლო მეორე მნიშვნელობა (3651) კი - ამავე ფურცელზე პროგრამის მიერ გამოყენებულ ბოლო ბაიტის მისამართს. (მენიშვნა. როგორც ფურცლების ისე ფურცლებში ბაიტების მისამართები ფურცლებში იწყება 0-დან.)
1. 45837
 2. 50493
 3. 55207
 4. 52804
 5. ჩამოთვლილთაგან არცერთი
83. წარმატებულად დასრულების შემდეგ რას აბრუნებს open სისტემური გამოძახება.
ფაილურ დესკრიპტორს
84. როგორც ვიცით პროცესის მიერ მოთხოვნილი რესურსის გამოყოფისა და გამოთავისუფლების მოვლენათა ჯაჭვი შედგება რამდენიმე ეტაპისაგან. კერძოდ რამდენი ეტაპისაგან შედგება ის?
3
85. ჩამოთვლილთაგან რომელი მიეკუთვნება აპარატული წყვეტის მნიშვნელოვან ტიპს
ტაიმერიდან წყვეტა
86. როგორც ცნობილია, მეცნიერებმა დაადგინეს ურთიერთბლოკირების წარმოქმნის აუცილებელი და საკმარისი პირობები. ჩამოთვლილთაგან რომელი შეესაბამება პირობას „რესურსის ლოდინი“?
პროცესს, რომელსაც დაკავებული აქვს გარკვეული რესურსი შეუძლია მოითხოვოს ახალი რესურსი



87. ვთქვათ dir2 არის სამუშაო დირექტორია. dir7 დირექტორის dir3 დირექტორიაში გადაკოპირებისთვის (copy-paste) საჭიროა ტერმინალში ბრძანება შესრულდეს შემდეგი სახით
cp -r dir5/dir7 /dir1/dir7
88. 0425 დაშვების უფლებების რვაობითი ჩანაწერის შესაბამის სიმბოლურ ჩანაწერს ექნება სახე:
-r --- w-r-x

89. როგორ იშიფრება აპრევატურა RAM

Random Access Memory

90. ჩამოთვლილთაგან რომელი შეესაბამება ასინქრონული წყვეტის განმარტებას.

ეს არის წყვეტა, რომელიც წარმოიშვა კოდში დაშვებული შეცდომისგან დამოუკიდებლად

ვთქვათ ოპერაციულ სისტემაში რესურსები წარმოდგენილია მრავალი ეგზემპლარის სახით. განვიხილოთ სურათზე
ნაჩვენები შემთხვევა, სადაც

A - შეესაბამება პროცესურისთვის უკვე გამოყოფილ რესურსებს

R - შეესაბამება პროცესის მიერ დამატებით მოთხოვნილ რესურსებს

F - შეესაბამება მიმდინარე მომენტში თავისუფალი რესურსების საერთო რაოდენობას

$$A = (1 \ 0 \ 1 \ 2 \ 1 \ 0 \ 0 \ 0 \ 1 \quad 0 \ 1 \ 1 \ 0 \ 0 \ 1) \quad R = (1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \quad 1 \ 0 \ 1 \ 0 \ 0 \ 0) \quad F = (1 \ 0 \ 1 \ 0 \ 0)$$

იძლევა თუ არა რესურსებზე მოთხოვნათა მიმდევრობა საიმედო მდგომარეობას?

False

91. ჩამოთვლილთაგან რომელი წარმოადგენს გამოთვლით მანქანაში გამოყენებულ ყველაზე სწრაფ
მეხსიერებას.

რეგისტრი

92. 0372 დაშვების უფლებების რვაობითი ჩანაწერის შესაბამის სიმბოლურ ჩანაწერს ექნება სახე:

- - wxrwx-w-

93. ვთქვათ ოპერაციულ სისტემაში პროგრამისთვის მეხსიერების გამოყოფა ხდება მისამართების უწყვეტი
მიმდევრობით. დავუშვათ, ოპერაციულ სისტემაში გამოყენებული ფურცლების ზომა არის 12 K ($12 * 1024$
ბაიტი).

რისი ტოლი იქნება პროგრამის მოცულობა, თუ ცნობილია, რომ პროგრამის განთავსება იწყება
პირველივე მისამართიდან (0) და პროგრამის მიერ დაკავებული ბოლო ბაიტის მისამართია (4, 9303),
სადაც წყვილის პირველი მნიშვნელობა (4) შეესაბამება ბოლო ფურცლის ნომერს, ხოლო მეორე
მნიშვნელობა (9303) კი - ამავე ფურცელზე პროგრამის მიერ გამოყენებულ ბოლო ბაიტის მისამართს.
(შენიშვნა. როგორც ფურცლების ისე ფურცლებში ბაიტების მისამართები ფურცლებში იწყება 0-დან.)

1. 63251

2. ჩამოთვლილთაგან არცერთი

3. 70493

4. 60735

5. 55837

94. ვთქვათ ოპერაციულ სისტემაში პროგრამისთვის მეხსიერების გამოყოფა ხდება მისამართების უწყვეტი
მიმდევრობით. დავუშვათ, ოპერაციულ სისტემაში გამოყენებული ფურცლის ზომა არის 10 K ($10 * 1024$
ბაიტი).

რისი ტოლი იქნება პროგრამისთვის გამოყოფილ ბოლო ფურცელზე ბაიტების რაოდენობა, თუ
ცნობილია პროგრამის მოცულობა არის 663502 ბაიტი და ის სრულად იკავებს 64 ფურცელს.

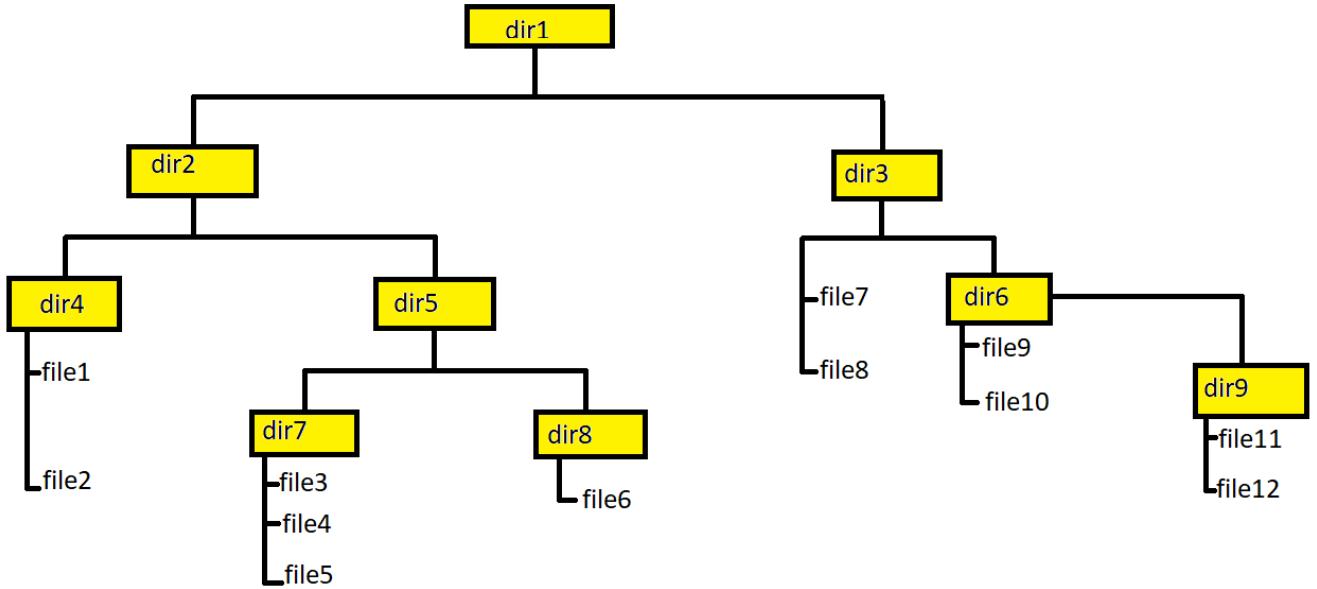
1. 8050

2. 9027

3. 8250

4. 7505

5. ჩამოთვლილთაგან არცერთი



95.

ვთქვათ `dir8` არის სამუშაო დირექტორია. `file4` ფაილის `dir4` დირექტორიაში `file5` სახელით გადაკოპირებისთვის (`copy-paste`) საჭიროა ტერმინალში ბრძანება შესრულდეს შემდეგი სახით
`cp ./dir7/file4 ../../dir4/file5`

განმარტებები

1. მოიყვანეთ პროცესის კონტექსტის გადართვის განმარტება
2. როგორ პროცესებს ეწოდებათ ურთიერთქმედი პროცესები?
3. ჩამოთვალეთ მონაცემთა გაცვლის საშუალებები
4. ახსენით რატომ უნდა ეწოდონ პროცესები ერთობლივ საქმიანობას

ვთქვათ ოპერაციულ სისტემაში 4 პროცესი და 6 რესურსი. ამასთან, რესურსები წარმოდგენილია მრავალი ეზემპლარის სახით. პროცესების მიერ რესურსებზე გაკეთებული მოთხოვნების შესაბამისი მატრიცის მეშვეობით გაარკვიეთ

1. (0.5 ქულა) მოთხოვნათა მიმდევრობა იძლევა თუ არა საიმედო მდგომარეობას (0.5 ქულა).
2. (0.5 ქულა) დადებითი პასუხის შემთხვევაში ცხადი სახით ამონტერეთ პროცესების მიმდევრობა, თუ როგორ უნდა მოხდეს მათი დაკმაყოფილება (0.5 ქულა).
3. (1 ქულა) უნდა ჩანდეს მთლიანი პროცესი
(შენიშვნა. მაქსიმალური ქულის მისაღებად უნდა მოხდეს მთლიანი პროცესის ამონტერა. აუცილებლობის შემთხვევაში შესაძლებელია 2 ფაილის მიმაგრება ან ფურცელზე დავალების გაკეთება.)

$$A = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} C = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} R = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

მოთხოვნათა მიმდევრობა იძლევა საიმედო მდგომარეობას.

პროცესების მიმდევრობა შემდეგია: P4 □ P1 □ P3 □ P2

$$A = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

$$A = \begin{pmatrix} 2 & 0 & 0 & 2 & 1 & 1 \end{pmatrix}$$

$$A = \begin{pmatrix} 3 & 1 & 0 & 3 & 2 & 1 \end{pmatrix}$$

$$A = \begin{pmatrix} 3 & 1 & 1 & 4 & 3 & 1 \end{pmatrix}$$

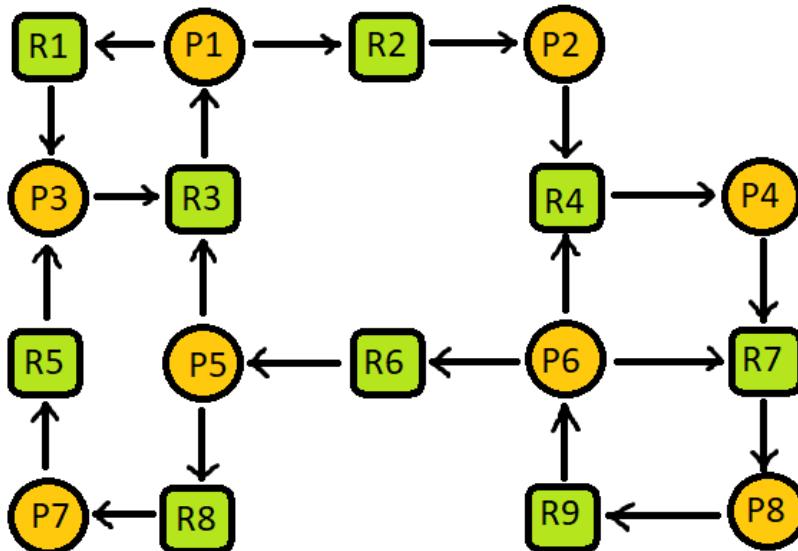
$$A = \begin{pmatrix} 4 & 1 & 2 & 4 & 4 & 2 \end{pmatrix}$$

სხვა ვარიანტის

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} C = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

5. ვთქვათ, ცხრილის სახით მოცემულია პროცესების მიერ რესურსებზე გაკეთებული მოთხოვნათა მიმდევრობა.
- 1) (0.5 ქულა) იძლევა თუ არა მოთხოვნათა მიმდევრობა საიმედო მდგომარეობას (მიუთითეთ ცხადად);
 - 2) (0.5 ქულა) ააგეთ შესაბამისი გრაფი;
 - 3) (2 ქულა) თუ მდგომარეობა არასაიმედოა შეგიძლიათ მიუთითოთ ციკლების რაოდენობა და რესურსები, რომლებიც მონაწილეობენ შესაბამის ციკლებში (რესურსები შეგიძლიათ ამოწეროთ მხოლოდ ერთხელ) ან ამოწერეთ ყველა ციკლი (ყველა პროცესითა და რესურსით).
(შენიშვნა. შეგიძლიათ დიაგრამა ფურცელზე გააკეთოთ და ჩატაროთ ამ ფორმით.)

	იკავებს	ითხოვს
P_1	R_3	$R_1 \ R_2$
P_2	R_2	R_4
P_3	$R_1 \ R_5$	R_3
P_4	R_4	R_7
P_5	R_6	$R_3 \ R_8$
P_6	R_9	$R_4 \ R_6 \ R_7$
P_7	R_8	R_5
P_8	R_7	R_9



მდგომარეობა არის არასაიმედო

ციკლები შემდეგია:

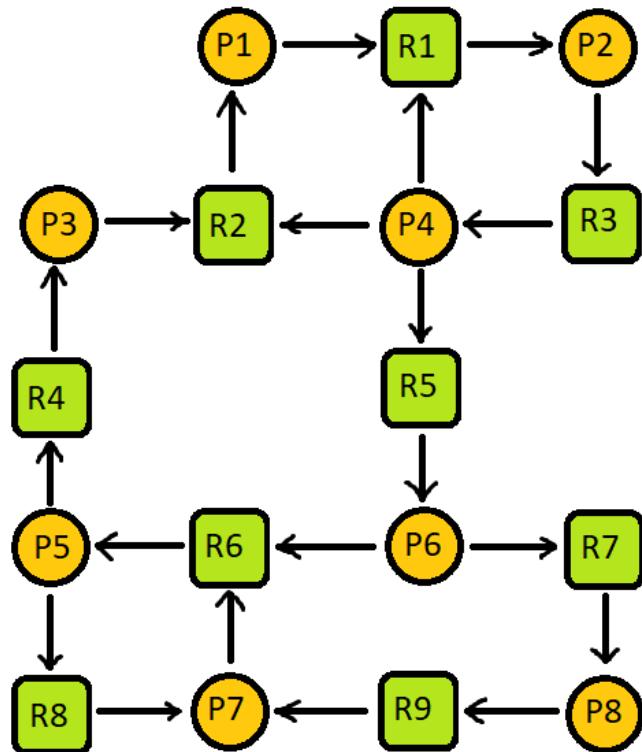
1. $R_4 \square P_4 \square R_7 \square P_8 \square R_9 \square P_6 \square R_4$
2. $P_6 \square R_7 \square P_8 \square R_9 \square P_6$
3. $P_1 \square R_2 \square P_2 \square R_4 \square P_4 \square R_7 \square P_8 \square R_9 \square P_6 \square R_6 \square P_5 \square R_3 \square P_1$
4. $P_1 \square R_2 \square P_2 \square R_4 \square P_4 \square R_7 \square P_8 \square R_9 \square P_6 \square R_6 \square P_5 \square R_8 \square P_7 \square R_5 \square P_3 \square R_3 \square P_1$

5. $P_1 \rightarrow P_3 \rightarrow R_3 \rightarrow P_1 \rightarrow R_1$

სხვა ვარიანტის

	იკავებს	ითხოვს
P_1	R_2	R_1
P_2	R_1	R_3
P_3	R_4	R_2
P_4	R_3	R_1 R_2 R_5
P_5	R_6	R_4 R_8
P_6	R_5	R_6 R_7
P_7	R_8 R_9	R_6
P_8	R_7	R_8

ბოლო სტრიქონში, სავარაუდოდ, R_8 -ს მაგივრად R_9 უნდა ეწეროს



მდგომარეობა არის არასაიმედო

ციკლები შემდეგა:

1. $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_4 \rightarrow R_2 \rightarrow P_1$

2. R1 P2 R3 P4 R1

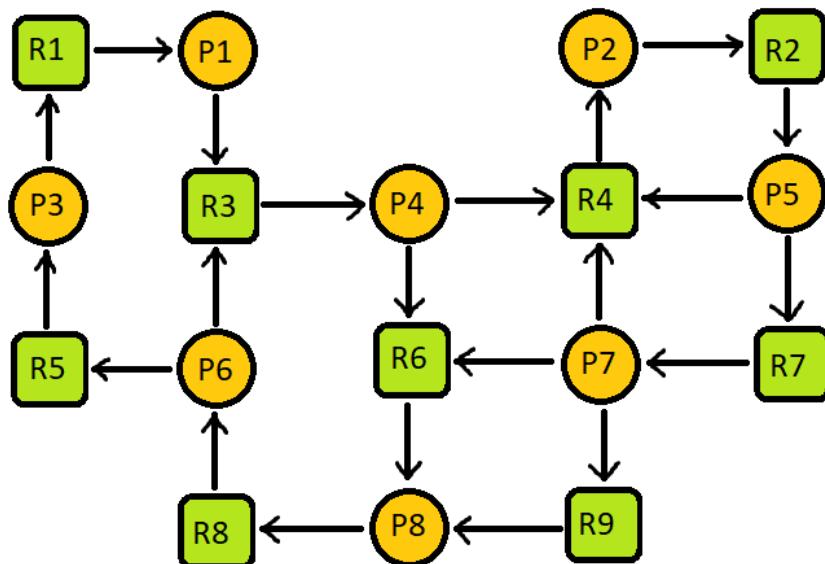
3. P1 □ R1 □ P2 □ R3 □ P4 □ R5 □ P6 □ R6 □ P5 □ R4 □ P3 □ R2 □ P1

4. P5 R8 P7 R6 P5

5. P1 R1 P2 R3 P4 R5 P6 R7 P8 R9 P7 R6 P5 R4 P3 R2 P1

სხვა ვარიანტის

	იკავებს	ითხოვს
P_1	R_1	R_3
P_2	R_4	R_2
P_3	R_5	R_1
P_4	R_3	$R_4 \quad R_6$
P_5	R_2	$R_4 \quad R_7$
P_6	R_8	$R_3 \quad R_5$
P_7	R_7	$R_4 \quad R_6 \quad R_9$
P_8	$R_6 \quad R_9$	R_8



მდგომარეობა არის არასაიმურო

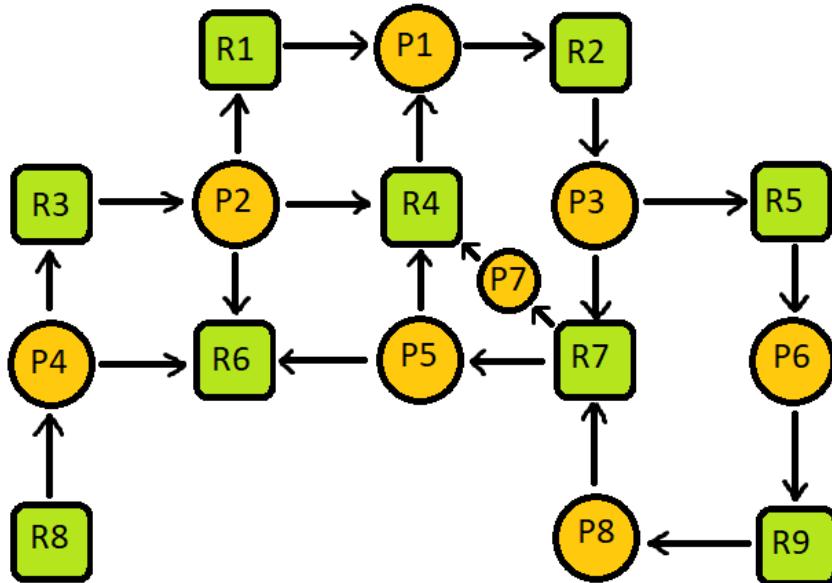
(କୁଳାଙ୍ଗପାଇ ଶ୍ରୀମଦ୍ଭଗବତ:

1. P2 R2 P5 R4 P2

2. P2 □ R2 □ P5 □ R7 □ P7 □ R4 □ P2
 3. R3 □ P4 □ R6 □ P8 □ R8 □ P6 □ R3
4. R1 □ P1 □ R3 □ P4 □ R6 □ P8 □ R8 □ P6 □ R5 □ P3 □ R1
5. R1 □ P1 □ R3 □ P4 □ R4 □ P2 □ R2 □ P5 □ R7 □ P7 □ R9 □ P8 □ R8 □ P6 □ R5 □ P3 □ R1

სხვა ვარიანტის

	იკავებს	ითხოვს
P_1	$R_1 \quad R_4$	R_2
P_2	R_3	$R_1 \quad R_4 \quad R_6$
P_3	R_2	$R_5 \quad R_7$
P_4	R_8	$R_3 \quad R_6$
P_5	R_7	$R_4 \quad R_6$
P_6	R_5	R_9
P_7	R_7	R_4
P_8	R_9	R_7



აქ, აშკარად, P7 ცუდად ჩასვა ცხრილში პაპუნამ
 მდგომარეობა არის არასაიმედო

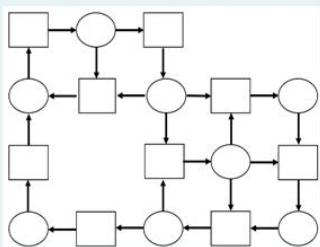
ციკლური შემდეგია:

- 1. P1 □ R2 □ P3 □ R7 □ P5 □ R4 □ P1**
2. P1 □ R2 □ P3 □ R7 □ P7 □ R4 □ P1

3. P1 □ R2 □ P3 □ R5 □ P6 □ R9 □ P8 □ R7 □ P5 □ R4 □ P1

4. P1 □ R2 □ P3 □ R5 □ P6 □ R9 □ P8 □ R7 □ P7 □ R4 □ P1

სურათზე ნაჩვენები დიაგრამა შეესაბამება პროცესების მიერ რესურსებზე გაკეთებული მოთხოვნათა მიმდევრობის არასაიმედო მდგომარეობას. რამდენი ციკლი იკვრება ამ დიაგრამაზე. (შენიშვნა. ერთიდაიგივე რესურსი ან პროცესი შეიძლება მონაწილეობდეს სხვა ციკლშიც. წრე აღნიშნავს პროცესს, ხოლო მართკუთხედი კი რესურსს.)



Select one:

- a. 4
- b. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- c. 3
- d. 5
- e. 2

ცნობილია პროგრამაში გამოყენებული ლოგიკური მისამართების დაკავშირება ფიზიკურ მისამართებთან შესაძლებელია რამდენიმე ეტაპზე. რა ეწოდება ეტაპს, რომელიც თუ წინასწარ უცნობია პროგრამის ადგილმდებარეობა აგენერირებს გადატანად კოდს?

Select one:

- a.
- b. შესრულების ეტაპი
- c. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- d. კომპილაციის ეტაპი
- e. ჩატვირთვის ეტაპი

[Clear my choice](#)

Next

ჩამოთვლილთაგან რომელი წარმოადგენს ენერგო-დამოუკიდებელ მექანიზმას?

Select one:

- a. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი

- c. 3
- d. 5
- e. 2

ნაკადის (thread) მიერ `pthread_join` ფუნქციის გამოყენების შემთხვევაში

elect one:

- a. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- b. იქმნება აზალი ნაკადი
- c. მიმდინარე ნაკადი ბრძანებათა მთვლელში უთითებს გამოძახებული ნაკადის მისამართს, რომელიც მისი დასრულების შემდეგ უნდა შესრულდეს
- d. მიმდინარე ნაკადი იძახებს აზალ ნაკადს, რომელიც იწყებს შესრულებას
- e. ამუშავდება სისტემური ნაკადი

ჩამოთვლილთაგან რომელი წარმოადგენს ენერგო-დამოუკიდებელ მეხსიერებას?

Select one:

- a. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- b. რეგისტრი
- c. Solid State Disk (SSD)
- d. ქაშ-მეხსიერება
- e. ოპერატორული მეხსიერება

[Clear my choice](#)

Question 4

Not yet
answered

Marked out of
0.75

 Flag question

ვთქვათ, ცნობილია, რომ ოპერაციულ სისტემაში პროგრამებისთვის მეხსიერების გამოყოფა ხდება უწყვეტი მიმდევრობით და გამოყენებული ფურცლების ზომა არის 16 KB (კილობაიტი).

გამოთვალით პროგრამისთვის მეშვიდე ფურცელზე გამოყოფილი 10607 -ე (ვირტუალური) მისამართის რეალური (ფიზიკური) მისამართი ოპერატიულ მეხსიერებაში, თუ ცნობილია, რომ პროგრამისთვის მისამართების გამოყოფა დაიწყო მისამართიდან 9075.

შენიშვნა. კალკულატორის გამოყენება წებადართულია. რა შედეგსაც მიიღებთ ის ააწყვეთ პირდაპირ. თუ ჩასაწერი ციფრების რაოდენობა მეტია ვიდრე თქვნის მიერ მიღებული მიმდევრობა, მაშინ ჩაწერისას შესაბამის მნიშვნელობას თავში დაუწერეთ შესაბამისი რაოდენობის 0 -ები.

პასუხი ააწყვეთ მოცემული ციფრებისგან: 0 0 1 7 8 7 8

7

Question 5

Not yet
answered

Marked out of
0.50

 Flag question

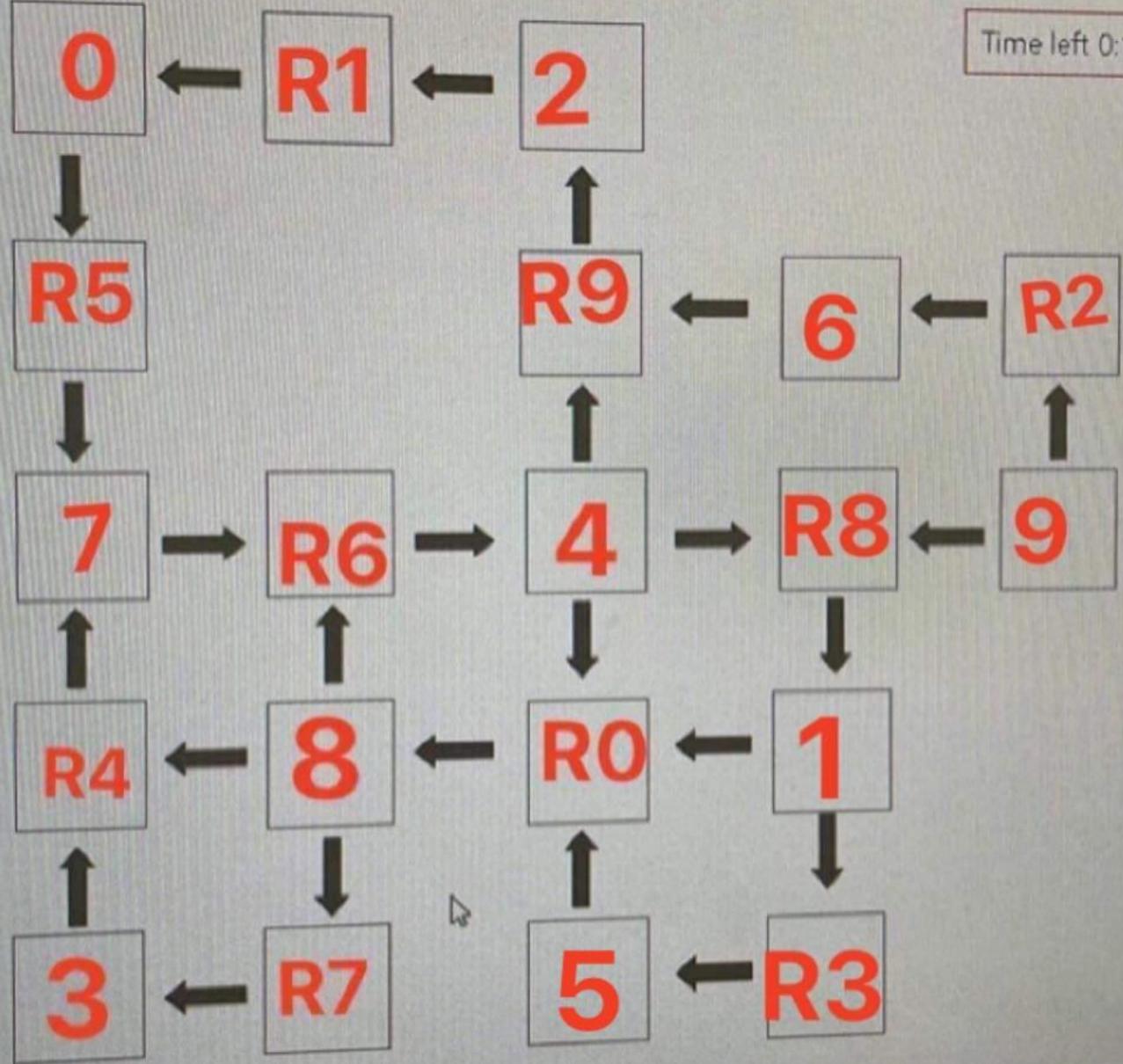
ჩამოთვლილთაგან რომელი ფუნქცია გამოიყენება ნაკადის (thread-ის) იდენტიფიკატორის მნიშვნელობის მისაღებად?

Select one:

- a. ***pthread_self()***
- b. ***pthread_id()***
- c. ***pthread_identifier()***
- d. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- e. ***posix_thread_id()***

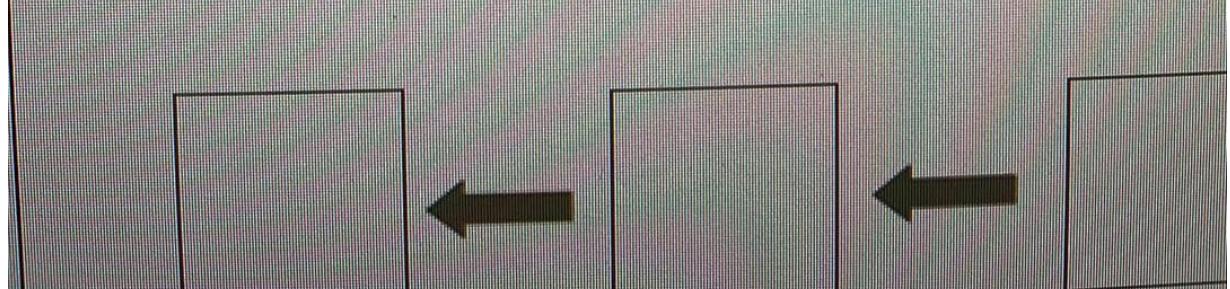
[Clear my choice](#)

Time left 0:13:18



როცესები გამოსახულია უშენებელი მიზნებით).

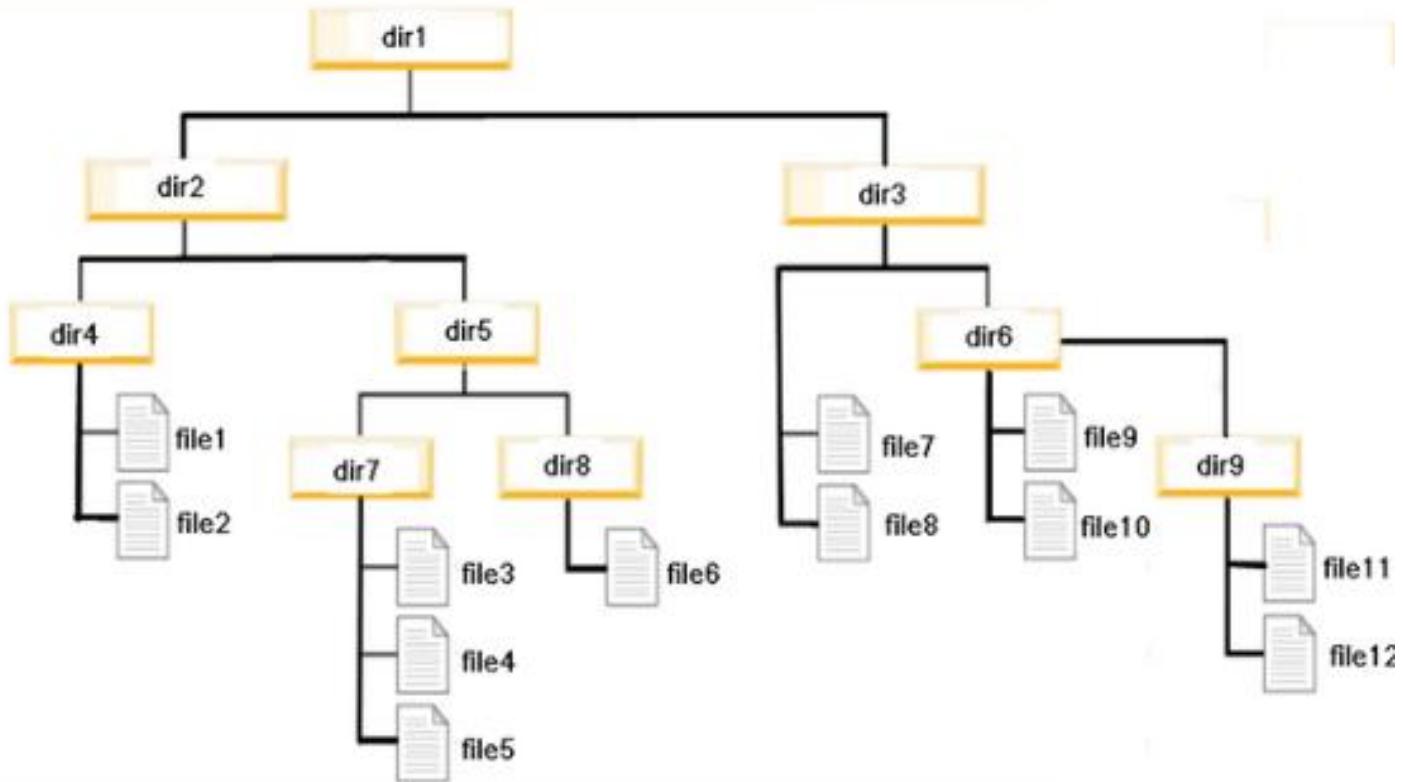
პროცესი	იკავებს	ითხოვს
0	R_1	R_5
1	R_8	R_0, R_3
2	R_9	R_1
3	R_7	R_4
4	R_6	R_0, R_8, R_9
5	R_3	R_0
6	R_2	R_9
7	R_4, R_5	R_6
8	R_0	R_4, R_6, R_7
9		R_2, R_8



ჩამოთვლითაგან რომელი ბრძანების გამოყენებით არის შესაძლებელი ფაილზე დაშვების უფლებების შეცვლა.

Select one:

- a. ჩამოთვლითაგან არცერთი პასუხი არაა სწორი
- b. **mknod**
- c. **chown**
- d. **umask**
- e. **chgrp**



თუ dir5 არის სამუშაო დირექტორია, მაშინ ტერმინალზე
შემდეგი ბრძანების შესრულების შედეგად

rm ..//dir4/file1 dir8/file6

Select one:

- a. ..//dir4/file1 ფუაილი გადაადგილებული იქნება არალ დირექტორიაში აწალი სახელით dir8/file6
- b. ..//dir4/file1 ფუაილი გადაადგილებული იქნება არალ დირექტორიაში სახელის შენარჩუნებით dir8/file1
- c. ფუაილური სისტემიდან წაიშლება ..//dir4/file1 და dir8/file6 ფუაილები
- d. ჩამოთვლილთაგან არცერთი პასუხი არაა სჭირო
- e. ..//dir4/file1 და dir8/file6 ფუაილი გადაადგილებული იქნება სამუშაო დირექტორიაში

**ჩამოთვლითაგან რომელი ბრძანების გამოყენებით არის
შესაძლებელი ფაილზე მომზადებლის ჯგუფის შეცვლა**

Select one:

- a. chmod
- b. mkfifo
- c. chown
- d. mknod
- e. ჩამოთვლითაგან არცერთი პასუხი არაა სწორი

ჩამოთვლილთაგან რომელი წინადადებაა ჰეშმარიტი pipe კავშირის არჩთან მიმართებით.

Select one:

- a. pipe არზის გამოყენება შეუძლია სისტემაში წარმოქმნილ ყველა იმ პროცესს, რომელთაც არ გააჩნიათ საერთო წარმოქმნელი
- b. pipe არზის გამოყენება შეუძლია სისტემაში წარმოქმნილ ყველა პროცესს მიუწედავად იმისა რომელი პროცესის მიერ მოხდა pipe არზის შექმნა
- c. pipe არზის გამოყენება შეუძლია სისტემაში წარმოქმნილ ყველა იმ პროცესს, რომელთა მეშვეობითაც წარმოიქმნა pipe არზის წარმოქმნელი პროცესი
- d. ჩამოთვლილ Copy Image ნი არაა სჭორი

Question 3
Not yet answered
Marked out of 1.00
Flag question

რას ნიშნავს რომ რესურსი არაა განატილებადი?

Select one:

- a. ასეთი რესურსი შეიძლება დასჭირდეს მიმდინარე პროცესს მოვალეობით
- b. რესურსის ჩამორთმევით დაკარგება პროცესის მიერ გაპეთებული მიზიშემლოვანი საქმიანობა
- c. ასეთი რესურსის გაზიარება შესაძლებელია რამდენიმე პროცესს შორის
- d. რესურსის ჩამორთმევით პროცესის ვერ შეძლებს საქმიანობის გაგრძელებას

Clear my choice

Question 5
Not yet answered
Marked out of 1.00
Flag question

რა ეწოდება ასეთაქციას, რომელიც გამოიყოლით სისტემაში ნაკლები მოცულობის ფიზიკური შესივრების არსებობის შემთხვევაში ხელს უწყობს მატი ფიზიკური შესივრების საჭიროების შეორენების მქონე პროგრამის ამუშავებას

Select one:

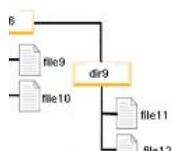
- a. thread-ები
- b. ფაილური სისტემა
- c. ვირტუალური მემკინება
- d. ვირტუალიზაცია

სტანდარტულად რამდენი კომპონენტისგან შედგება შეტანა/გამოტანის მოწყობილობა

Select one:

- a. 4
- b. 2
- c. 3
- d. 5

15



ჩამოთვლილთაგან, რომელი შეესაბამება სიმპლექსური კავშირის არჩის განმარტებას.

Select one:

- a. კავშირის არჩს, რომლის მეშვეობითაც შესაძლებელია ორივე მიმართულებით მონაცემების გადაცემა ერთდროულად
- b. კავშირის არჩს, რომლის მეშვეობითაც შესაძლებელია მემკილე პროცესებს შორის მონაცემების გაცვლა
- c. კავშირის არჩს, რომლის მეშვეობითაც შესაძლებელია ორივე მიმართულებით მონაცემების გადაცემა მაგრამ რიგითობის დაცვით
- d. კავშირის არჩს, რომლის მეშვეობითაც შესაძლებელია როგორც მემკილე ისე არამემკილე პროცესებს შორის მონაცემების გაცვლა
- e. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი

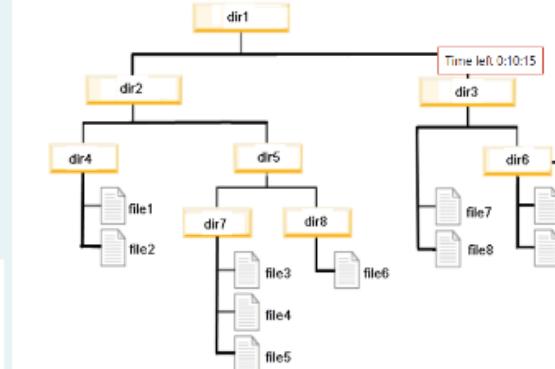
Clear my choice

ჩამოთვლილთაგან რომელი შეესაბამება დაუპლექსური კავშირის არჩის განმარტებას.

Select one:

- a. chgrp
- b. umask
- c. chusr
- d. modes
- e. chmod

Clear my choice



თუ dir3 არის სამუშაო დოკუმენტი, მაშინ ტერმინალში შემდეგი ბრძანების შესრულების შედეგად

`cat dir6/file9 dir6/dir9/file12 > file7`

Select one:

- a. dir3 დოკუმენტისაში მონაცემა dir9/file9 და dir6/dir9/file12 ფაილების გადადგინდება და მათში შემავალი მონაცემების გაურთმისა ახალი სახელით file7
- b. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- c. file7 ფაილში მიმდევრობის შეკრუნებით ჩაიტანების შემცირება
- d. ტერმინალის უკანზე გამოტანილი იქნება file7, dir6/file9 და dir6/dir9/file12 ფაილების შეგთავსი
- e. ფაილური სისტემიდან წიიშლება file7, dir6/file9 და dir6/dir9/file12 ფაილები

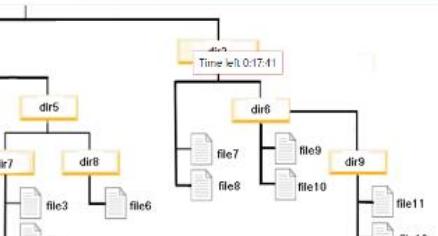
Question 1
Not yet answered
Marked out of 1.00
Flag question

ჩამოთვლილთაგან, რომელი შეესაბამება დაუპლექსური კავშირის არჩის განმარტებას.

Select one:

- a. კავშირის არჩს, რომლის მეშვეობითაც შესაძლებელია ორივე მიმართულებით მონაცემების გადაცემა ერთდროულად
- b. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- c. კავშირის არჩს, რომლის მეშვეობითაც შესაძლებელია მემკილე პრიცესებს შორის მონაცემების გაცვლა
- d. კავშირის არჩს, რომლის მეშვეობითაც შესაძლებელია ორგორც მემკილე ისე არამემკილე პრიცესებს შორის მონაცემების გაცვლა
- e. კავშირის არჩს, რომლის მეშვეობითაც შესაძლებელია ორივე მიმართულებით მონაცემების გადაცემა მაგრამ რიგითობის დაცვით

Clear my choice



Q 6. 5

[Clear my choice](#)

შეჯიბრის მდგომარეობის აცილების მიზნით კრიტიკულ სექტორები აღმოჩინითის "მაცრი მიმდევრობა" გამოყენების შემთხვევაში ჩამოთვლილობაგან რომელი წინადადება ჰქონის.

Select one:

- a. პროცესს შეუძლია შევიდეს საკუთარ კრიტიკულ სექტორიში თუ იქიდან გამოვიდა მიმდევრობაში მის წინ მდგომი პროცესი
- b. პროცესს შეუძლია იმყოფებოდეს საკუთარ კრიტიკულ სექტორიში სამუშაოს დასრულებამდე
- c. ჩამოთვლილობაგან არცერთი პასუხი არაა სწორი
- d. ერთხე მეტ პროცესს ერთდროულად შეუძლია იმყოფებოდეს საკუთარ კრიტიკულ სექტორიში

[Clear my choice](#)

პროგრამული კოდის შემუშავებისას პროგრამისტების ჯგუფს დასჭირდა რამდენიმე განაწილებადი მემსეირების გამოყენება. განაწილებადი მემსეირებისთვის უწინააღმდეგი გასაღების მნიშვნელობის მისაღებად პროგრამისტებმა გამოყენენ სისტემაში არსებული ფაილი (key.txt). ჩამოთვლილობაგან რომელი ფუნქციის გამოყენებით არის შესაღებელი უწინააღმდეგი გასაღების მნიშვნელობის მიღება?

Select one:

- a. `fkey()`
- b. `ftok()`
- c. ჩამოთვლილობაგან არცერთი პასუხი არაა სწორი
- d. `getkey()`
- e. `shmget()`

[Clear my choice](#)

ჩამოთვლილობაგან რომელი შეესაბამება საქმიანობას რომლითაც დაკავებული ბრძანებათა მთვლელი?

Select one:

- a. ინახავს კონკრეტული ერთი პროგრამის ამუშავების შემდეგ ამავე პროგრამის წარმატებული საქმიანობის გაგრძელებისთვის რიგით შემდეგი შესრულებული ბრძანების ბრძანების მიმთხოვებელს
- b. ინახავს კონკრეტული ერთი პროგრამის ამუშავების შემდეგ ამავე პროგრამის წარმატებული საქმიანობის გაგრძელებისთვის რიგით შემდეგი შესრულებული ბრძანების მიმთხოვებელს
- c. ინახავს ოპერაციული სისტემის ამუშავების მომენტიდან მიმდინარე მომენტმდე სხვადასხვა პროგრამების წარმატებული საქმიანობისთვის შესრულებული ბრძანებების საერთო რაოდენობას
- d. ჩამოთვლილობაგან არცერთი პასუხი არაა სწორი
- e. ინახავს კონკრეტული ერთი პროგრამის ამუშავების შემდეგ მიმდინარე მომენტმდე შესრულებული ბრძანებების რაოდენობას

[Clear my choice](#)

ჩამოთვლილობაგან რომელ მდგომარეობაში მოწდება პრაცესის გადაყენების შემდეგ მიმდინარე დროისგან ამავე პროგრამის წარმატებული საქმიანობისთვის შესრულებული ბრძანებების რაოდენობას

Select one:

- a. მზადდების
- b. ჩამოთვლილობაგან არცერთი პასუხი არაა სწორი
- c. ბლოკირებული
- d. შესრულების

[Clear my choice](#)

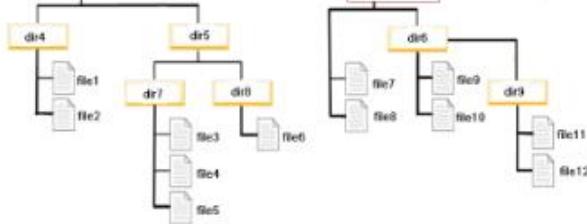
a. chgrp
 b. umask
 c. chusr
 d. modes
 e. chmod
[Clear my choice](#)

რეკვეტის რომელ ტიპს მიეკუთვნება კოდში დაშვებული შეცდომის გამო წარმოქმნილი წყვეტა

Select one:

- a. ასინქრონული წყვეტას
- b. პროცესთაშორის წყვეტას
- c. სინქრონული წყვეტას
- d. ტაიმერილან წყვეტას

[Clear my choice](#)



თუ dirs არის სამუშაო ფორმეტირის, მაშინ ტკრმინალში ცემდევი ბრძანების შესრულების შედეგად
rm .. /dir4/file1 dir8/file6

Select one:

- a. ./dir4/file1 ფაილი გადაადგილებული იქნება ანაღ დორექტორიაზი საწლიას შენაჩირებით dir8/file1
- b. სამოთავლალაგან არცერთი პასუხი არაა სწორი
- c. ./dir4/file1 და dir8/file6 ფაილი გადაადგილებული იქნება სამუშაო ფორმეტირიაზი
- d. ფაილები სისტემით წაიმლანა .. /dir4/file1 და dir8/file6 ფაილები
- e. ./dir4/file1 ფაილი გადაადგილებული იქნება ანაღ დორექტორიაზი ანაღ საწლიას dir8/file6

სიმოთხლილათაგან რომელი სისტემები გამოიჩინება გამოიყენება მშობელი მრიულის იღვნეულიდატორის მინერველიმის მისაღებად.

Select one:

- a. სამოთხლილათაგან არცერთი პასუხი არაა სწორი
- b. getpid
- c. getgid
- d. getppid
- e. getuid

[Clear my choice](#)

ვთქვათ, "Dir 6" არის სამუშაო დოკუმენტირია, რომელიც ამზორებული მიმღიმარე საქმიანობას. "Dir 6" დოკუმენტის სამართლებრივ სამოთხლილათაგან რომელი იქნება "File 6". ფაილის მიმართებით საჭირო

Select one:

- a. .. / Dir 1 / Dir 5 / File 6
- b. . / Dir 1 / Dir 5 / File 6
- c. .. / .. / Dir 5 / File 6
- d. . / Dir 5 / File 6
- e. სამოთხლილათაგან არცერთი პასუხი არაა სწორი

პროგრამული კოდის შემუშავებისას პროგრამისტს დასჭირდა 1000 char ტიპის მნიშვნელობიანი მასივის განთავსება

Time left 0:16:54

პროგრამული კოდის შემუშავებისას პროგრამისტს
დასჭირდა 1000 char ტიპის მნიშვნელობებიანი მასივის
განთავსება განაწილებად მეხსიერებაში. ქვემოთ
ჩამოთვლილთან პროგრამული კოდის რომელი
ფრაგმენტითაა შესაძლებელი (key გასაღებისთვის)
განაწილებადი მეხსიერების შექმნა და გამოყენება?

Select one:

- a. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- b. int shmId = shmget(key, IPC_CREAT | IPC_EXCL,
- c. int shmId = shmget(key , 1000*sizeof(char) | I
- d. int shmId = shmget(key, 250*sizeof(int), 0);
- e. int shmId = shmget(key, 1000 * char, 0);

d

ჩამოთვლილთაგან რომელი შეესაბამება კრიტიკული სექციის
განმარტებას.

Time left 0:13:05

ჩამოთვლილთაგან რომელი შეესაბამება
კრიტიკული სექციის განმარტებას

Select one:

- a. მეხსიერების არე, რომელსაც პროცესებს
შორის მონაცემების გაცვლის მიზნით
იყენებს FIFO არჩი
- b. მეხსიერების არე, რომელშიც ინახება
პროცესის მთლიანი მონაცემები
(იდენტიფიკატორი, სისტემური ცვლადები,
ინსტრუქციები, მონაცემები, და ა.შ.)
- c. ჩამოთვლილთაგან არცერთი პასუხი არაა
სწორი
- d. მეხსიერების არე, რომელსაც პროცესებს
შორის მონაცემების გაცვლის მიზნით
იყენებს rpipe არჩი
- e. მეხსიერების არე, რომელსაც იზიარებს ორი

ჩამოთვლილთაგან რომელი საქმიანობითაა დაკავებული
ბრძანებათა მთვლელი

Time left 0:16:12

ჩამოთვლილთაგან რომელი საქმიანობითაა დაკავებული
ბრძანებათა მთვლელი.

Select one:

1. ითვლის პროცესის მიერ მიმდინარე მომენტამდე შესრულებელი ბრძანებების რაოდენობას

2. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი

3. ითვლის პროცესის მიერ შესასრულებელი ბრძანებების საერთო რაოდენობას

4. ითვლის პროცესის მიერ მიმდინარე მომენტის შემდეგ შესასრულებელი ბრძანებების რაოდენობას

რომელი შეესაბამება პროცესის კონტექსტის გადართვის განმარტებას.

Time left 0:15:42

ჩამოთვლილთაგან რომელი შეესაბამება პროცესის კონტექსტის გადართვის განმარტებას.

Select one:

a. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი

b. მდგომარეობას, რომლის დროსაც აქტიურ მდგომარეობაში მყოფ პროცესი საკუთარი საქმიანობის შემსუბუქების მიზნით ქმნის ანალ პროცესს

c. ჩამოთვლილთაგან ყველა პასუხი სწორია (იგულისხმება უარყოფითი პასუხის გარდა ყველა)

d. მდგომარეობას, რომლის დროსაც პროცესი გადადის ბლოკირების მდგომარეობიდან გზადყოფნის მდგომარეობაში

e. მდგომარეობა, რომლის დროსაც ინახება მიმდინარე პროცესის მიერ შესრულებული საქმიანობა და შესასრულებლად იტვირთება ანალი პროცესის მონაცემები

Next pa

რას ნიშნავს რომ რესურსი არაა განაწილებადი?

Question 3

Not yet
answered

Marked out of
1:00

View question

რას ნიშნავს რომ რესურსი არაა განაწილებადი?

Select one:

- a. ასეთი რესურსი შეიძლება დასჭირდეს მიმდინარე პროცესს მოგვიანებით
- b. რესურსის ჩამორთმევით დაიკარგება პროცესის მიერ გაკეთებული მნიშვნელოვანი საქმიანობა
- c. ასეთი რესურსის გაზიარება შესაძლებელია რამდენიმე პროცესს შორის
- d. რესურსის ჩამორთმევით პროცესის ვერ შეძლებს საქმიანობის გაგრძელებას

ENTER YOUR ANSWER

1) ჩამოთვლილთაგან რომელი სისტემური გამოძახება გამოიყენება

ჩამოთვლილთაგან რომელი სისტემური გამოძახება გამოიყენება მომხმარებლის ჯგუფის იდენტიფიკატორის მნიშვნელობის მისაღებად.

აირჩიეთ ერთი:

- a.
getgroup
- b.
getpid
- c.
getuid
- d.
getgid
- e.
getppid

[Clear my choice](#)

2) ვთქვათ ერთპროცესორულ სისტემაში ...

ვთქვათ ერთპროცესორულ სისტემაში პროცესი იმყოფება შესრულების მდგომარეობაში. ჩამოთვლილთაგან რომელი წინადადებაა ჭეშმარიტი.

აირჩიეთ ერთი:

- a.
შესრულების მდგომარეობაში მყოფი პროცესის საქმიანობის ბლოკირება შეუძლია მიმდინარე (იმავე) პროცესს
- b.
შესრულების მდგომარეობაში მყოფი პროცესის საქმიანობის ბლოკირება შეუძლია მზადყოფნის მდგომარეობაში მყოფ პროცესს
- c.
ყველა ჩამოთვლილი შემთხვევა ჭეშმარიტია (იგულისხმება უარყოფის გარდა ყველა შემთხვევა)
- d.
ჩამოთვლილთაგან არცერთი პასუხი არა სწორი
- e.
შესრულების მდგომარეობაში მყოფი პროცესის საქმიანობის ბლოკირება შეუძლია ბლოკირების მდგომარეობაში მყოფ პროცესს

[Clear my choice](#)

3) ჩამოთვლილთაგან რომელი შეესაბამება ასინქრონულ წყვეტის განმარტებას.

ჩამოთვლილთაგან რომელი შეესაბამება ასინქრონული წყვეტის განმარტებას.

აირჩიეთ ერთი:

- a.
ეს არის წყვეტა, რომელიც წარმოიშვა კოდში დაშვებული შეცდომის გამო.
- b.
ეს არის წყვეტა, რომელიც წარმოიშვა კოდში დაშვებული შეცდომისგან დამოუკიდებლად
- c.
ეს არის წყვეტა, რომელიც წარმოიშვა კლავიატურაზე დაწვაპუნების შედეგად
- d.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი

4) ჩამოთვლილთაგან რომელ შემთხვევაში შეიძლება პროცესი მზადყოფნის მდგომარეობაში აღმოჩნდეს

ჩამოთვლილთაგან რომელ შემთხვევაში შეიძლება აღმოჩნდეს პროცესი მზადყოფნის მდგომარეობაში

აირჩიეთ ერთი:

- a.
თუ პროცესს დაბადების (სისტემაში გამოჩენის) შემდეგ მოთხოვნის შესაბამისად გამოეყო პროცესორის გარდა შესასრულებლად საჭირო ყველა რესურსი
- b.
ბლოკირების მდგომარეობაში მყოფ პროცესს გამოეყო მისთვის საჭირო ყველა რესურსი
- c.
დროითი კვანტის ამოწურვის გამო პროცესს ჩამოერთვა პროცესორი
- d.
ჩამოთვლილთაგან ყველა პასუხი სწორია (იგულისხმება უარყოფითი პასუხის გარდა ყველა)
- e.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი

5) რომელი შესაბამება write სისტემური ...

ჩამოთვლილთაგან რომელი შესაბამება write სისტემური გამოძახების გამოყენების სწორ ფორმას, თუ N მასივში ელემენტების რაოდენობაა, ხოლო fd კი - შესაბამისი ნაკადის სახელი

აირჩიეთ ერთი:

- a. **char buf[N];
write(buf, fd, N);**
- b. **char buf[N];
write(N, fd, buf);**
- c. **char buf[N];
write(N, buf, fd);**
- d. **char buf[N];
write(fd, buf, N);**

6) 0532 დაშვების

0532 დაშვების უფლებების რვაობითი ჩანაწერის შესაბამის სიმბოლურ ჩანაწერს ექნება სახე:

აირჩიეთ ერთი:

- a.
-IW-I-X-WX
- b.
-I-XIW--WX
- c.
-I-XI---W-
- d.
--WXI--IW-
- e.
-I-X-WX-W-

8) რა ეწოდება აბსტრაქციას ,რომელიც გამოთვლით სისტემაში ..

რა ეწოდება აბსტრაქციას ,რომელიც გამოთვლით სისტემაში ნაკლები მოცულობის ფიზიკური მეხსიერების არსებობის შემთხვევაში ხელს უწყობს მეტი ფიზიკური მეხსიერების საჭიროების მქონე პროგრამის ამუშავებას

აირჩიეთ ერთი:

- a.
thread-ები
- b.
ფაილური სისტემა
- c.
ვირტუალიზაცია
- d.
ვირტუალური მეხსიერება

9) კავშირის არხის დასრულების პროცესირება ...

კავშირის არხის დასრულების პროცესირება რა შემთხვევაში არაა აუცილებელი

აირჩიეთ ერთი:

- a.
წამოთვლილთაგან ყველა პასუხი სწორია (იგულისხმება უარყოფითი პასუხის გარდა ყველა)
- b.
წამოთვლილთაგან არცერთი პასუხი არა სწორი
- c.
პროცესებს შორის კომუნიკაციის დასრულების შემდეგ
- d.
თუ პროცესებს შორის კავშირის არხის გახსნა პროცესირებული იყო სპეციალური მოქმედებებით
- e.
თუ პროცესებს შორის კავშირის არხის გახსნა არ იყო პროცესირებული სპეციალური მოქმედებებით

Clear my choice

10) განსაკუთრებული შემთხვევის რომელ ტიპს მიეკუთვნება ..

განსაკუთრებული შემთხვევის რომელ ტიპს მიეკუთვნება პროგრამის შესრულების მიმდინარე მომენტში ისეთ ფაილზე მიმართვა, რომელიც ფაილურ სისტემაში არსებობს, მაგრამ სხვა დირექტორიში არის განთავსებული

აირჩიეთ ერთი:

- a.
მართვადი განსაკუთრებული შემთხვევა
- b.
დამოკიდებული განსაკუთრებული შემთხვევა
- c.
გამოუსწორებელი განსაკუთრებული შემთხვევა
- d.
გამოსწორებადი განსაკუთრებული შემთხვევა

[Clear my choice](#)

11) ჩამოთვლილთაგან რომელი ბრძანებითაა შესაძლებელი ფაილის

ჩამოთვლილთაგან რომელი ბრძანებითაა შესაძლებელი ფაილის შიგთავსის დათვალიერება მისი გახსნის გარეშე.

აირჩიეთ ერთი:

- a.
pwd
- b.
mkdir
- c.
cd
- d.
ls
- e.
cat

[Clear my choice](#)

12) თუ მომხმარებელმა გადაწყვიტა ოპერაციულ სისტემის

თუ მომხმარებელმა გადაწყვიტა ოპერაციულ სისტემის სწრაფებების ამაღლების მიზნით პროცესების გარკვეული ჯგუფი იძულებით გააჩეროს („დააპაუზოს“), მაშინ შესაბამისი პროცესები აღმოჩნდებიან მდგომარეობაში

აირჩიეთ ერთი:

- a.
შეჩერებული ბლოკირებული
- b.
ბლოკირებული
- c.
შეჩერებული მზად
- d.
დაპაუზებული

13) ჩამოთვლილთაგან რომელი შეესაბამება საქმიანობას ..

ჩამოთვლილთაგან რომელი შექსაბამება საქმიანობას რომლითაც დაკავშირდეთ ბრძანებათა მთვლელი?

აღნიშვნელები:

- a. ინახავს კონკრეტული ერთი პროგრამის ამუშავების შემდეგ ამავე პროგრამის წარმატებული საქმიანობის გაგრძელებისთვის რიგით შემდეგი შესრულებული ბრძანების მიმთათებელს
- b. ინახავს კონკრეტული ერთი პროგრამის ამუშავების შემდეგ მიმდინარე დროში ამავე პროგრამის წარმატებული საქმიანობისთვის შესრულებული ბრძანებების რაოდენობას
- c. ინახავს კონკრეტული ერთი პროგრამის ამუშავების შემდეგ ამავე პროგრამის წარმატებული საქმიანობის გაგრძელებისთვის რიგით შემდეგი შესრულებული რამდენიმე ბრძანების მიმთათებელს
- d. ინახავს ოპერაციული სისტემის ამუშავების მომენტიდან მიმდინარე მომენტამდე სხვადასხვა პროგრამების წარმატებული საქმიანობისთვის შესრულებული ბრძანებების საერთო რაოდენობას
- e. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი

14) ჩამოთვლილთაგან რომელი წარმოადგენს მონოლიტური ..

ჩამოთვლილთაგან რომელი წარმოადგენს მონოლიტური არქიტექტურის ნაკლოვანებას ?

აღნიშვნელები:

- a. ყველა პროცედურა ერთმანეთთან ურთიერთქმედებს მთავარი პროცედურის გამოყენებით
- b. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- c. პროცედურებს არ შეუძლიათ ზეგავლენა იქნიონ სხვა პროცედურების საქმიანობაზე
- d. პროცედურათა ნაკრების საბოთ შემუშავდეთ თანამდებობა სისტემის კოდი წელა მუშაობს

[Clear my choice](#)

15) ჩამოთვლილთაგან რომელი ბრძანების გამოყენებით არის შესაძლებელი ფაილზე

..

ჩამოთვლილთაგან რომელი ბრძანების გამოყენებით არის შესაძლებელია ფაილზე დაშვების უფლებების შეცვლა

არჩიეთ ერთი:

- a.
chmod
- b.
chgrp
- c.
chusr
- d.
modes
- e.
umask

[Clear my choice](#)

16) კავშირის არხის დასრულების

The screenshot shows a Windows desktop with a course interface overlaid. The question asks which command is used to change file permissions. The correct answer is 'chmod'. The question text is in Georgian.

პატიორის არხის დასრულების პროცესისა თუ შემთხვევაში არა აუცილებელი
არჩიეთ ერთი:

- a. ჩამოთვლილთაგან ყველ პასუხი სწორია (იგულისხმება ურთყოფით პასუხის გარდა ყველა)
- b. ჩამოთვლილთაგან არცერთი პასუხი არა სწორი
- c. პროცესში შეიძლება კომუნიკაციის დასრულების შემდეგ
- d. თუ პროცესში შეიძლება კავშირის არხის გასწავა პროცესირებული იყო საედისლო მოქმედებებით
- e. თუ პროცესში შეიძლება კავშირის არხის გასწავა არ იყო პროცესირებული საედისლო მოქმედებებით

[Clear my choice](#) [Next Page](#)

PREVIOUS ACTIVITY
დაგენერიკული 1 - მუსიკა: 10 ჭრა

NEXT ACTIVITY
სიმულაცია 1. UNIX ინსტრუმენტები

REDMI NOTE 8T
AI QUAD CAMERA

17) წყვეტის რომელ ტიპს

წყვეტის რომელ ტიპს მიეკუთვნება კოდში დაშვერული შეცდომის გამო წარმოქმნილი წყვეტა

აირჩიეთ ერთი:

- a.
ასინქრონულ წყვეტას
- b.
პროცესთაშორის წყვეტას
- c.
სინქრონულ წყვეტას
- d.
ტაიმერიდან წყვეტას



18) ჩამოთვლილთაგან რომელი წინადადებაა ჭეშმარიტი FIFO კავშირის არხთან
მიმართებით

ჩამოთვლილთაგან რომელი წინადადებაა ჭეშმარიტი FIFO კავშირის არხთან მიმართებით.

აირჩიეთ ერთი:

- a.
FIFO კავშირის არხის გამოყენება შეუძლია სისტემაში წარმოქმნილ ყველა პროცესს მიუხედავად იმისა როდის ან რომელი პროცესის მიერ მოხდა FIFO კავშირის არხის წარმოქმნა
- b.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- c.
FIFO კავშირის არხის გამოყენება შეუძლია სისტემაში წარმოქმნილ ყველა იმ პროცესს, რომელთა მეშვეობითაც წარმოიქმნა FIFO კავშირის არხის წარმოქმნელი პროცესი
- d.
FIFO კავშირის არხის გამოყენება შეუძლია სისტემაში წარმოქმნილ ყველა იმ პროცესს, რომელსაც ყავს FIFO კავშირის არხის წარმოქმნელი საერთო წინაპარი

19) ვთქვათ ერთპროცესორულ სისტემაში პროცესი იმყოფება შესრულების
მდგომარეობაში ..

ვთქვათ ერთპროცესორულ სისტემაში პროცესი იმყოფება შესრულების მდგომარეობაში. ჩამოთვლილთაგან რომელი წინადადებაა ჭეშმარიტი.

აირჩიეთ ერთი:

- a.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- b.
შესრულების მდგომარეობაში მყოფი პროცესის საქმიანობის ბლოკირება შეუძლია ბლოკირების მდგომარეობაში მყოფ პროცესს
- c.
ყველა ჩამოთვლილი შემთხვევა ჭეშმარიტია (იგულისხმება უარყოფის გარდა ყველა შემთხვევა)
- d.
შესრულების მდგომარეობაში მყოფი პროცესის საქმიანობის ბლოკირება შეუძლია მიმდინარე (იმავე) პროცესს
- e.
შესრულების მდგომარეობაში მყოფი პროცესის საქმიანობის ბლოკირება შეუძლია მზადყოფნის მდგომარეობაში მყოფ პროცესს

[Clear my choice](#)

20) ჩამოთვლილთაგან რომელი მიეკუთვნება აპარატურული წყვეტის
მნიშვნელოვან ტიპს

ჩამოთვლილთაგან რომელი მიეკუთვნება აპარატურული წყვეტის მნიშვნელოვან ტიპს

აირჩიეთ ერთი:

- a.
შეტანა/გამოტანა წყვეტა
- b.
ტაიმერიდან წყვეტა
- c.
პროცესორთაშორისი წყვეტა
- d.
კლავიატურიდან წყვეტა

22) ვთქვათ აპარატურულ სისტემის შექმნისას პროგრამისტმა გამოიყენა
ფიქსირებული ზომის სტრუქტურა ..

ვთქვათ ოპერაციული სისტემის შექმნისას პროგრამისტმა გამოიყენა ფიქსირებული ზომის სტრუქტურა გარკვეული ტიპის
მონაცემების განსათავსებლად, რომელიც დინამიურად იცხება ელემენტებით და თავისუფლდება (ძლიერად) ელემენტებისგან.
როგორი ტიპის განსაკუთრებულ შემთხვევას მიეკუთვნება სიტუაცია, რომლის დროსაც სტრუქტურაში მონაცემების ჩაწერისას
მასში შეიძლება არ იყოს ადგილი ახალი მონაცემის განსათავსებლად.

აირჩიეთ ერთი:

- a.
გამოუსწორებელს
- b.
გამოსწორებადს

[Clear my choice](#)

23)ჩამოთვლილთაგან რომელი წარმოადგენს მონოლიტური არქიტექტურის ..

სამოთვლილთაგან რომელი წარმოადგენს მონოლიტური არქიტექტურის ნაკლოვანებას ?

ირჩევის ერთი:

- a. პროცედურა ერთმანეთთან ურთიერთქმედებს მთავარი პროცედურის გამოყენებით
- b. სამოთვლილთაგან არცერთი პასუხი არაა სწორი
- c. პროცედურების არ შეუძლიათ ზეგავლენა იქნიონ სხვა პროცედურების საქმიანობაზე
- d. პროცედურათა ნაკრების სახით შემუშავებული ოპერაციული სისტემის კოდი ნეღა მუშაობს

[Clear my choice](#)

PREVIOUS ACTIVITY

NEXT ACTIVITY

REDMI NOTE 8T
AI QUAD CAMERA

24) ვთქვათ ოპერაციული სისტემის შექმნისას პროგრამისტმა

The screenshot shows a Moodle-based e-learning platform. At the top, there's a green header bar with the title "Kursi 1 - მუსიკა" and a URL "e-learning.tsu.ge/mod/quiz/attempt.php?attempt=371514&cmid=192198&page=2". The main content area has a dark blue header with the university logo and the text "მარცხნიდან ვაკების მიერთების სახელმწიფო უნივერსიტეტი სახელმწიფო უნივერსიტეტი". Below this is a navigation bar with links for "Home", "My Courses", "Events", and "This course". On the right side of the header, there are icons for notifications, messages, and user settings, along with a search bar labeled "Search Courses".

The main content area displays a course page for "Kursi 1 - მუსიკა". The page includes a breadcrumb navigation: "Home > Kursi 1 - მუსიკა > პუნქტუალური სისტემა (2022) > პუნქტუალური დანალექა & ჰერიტაჟი > Kursi 1 - მუსიკა: 5 ქულა". The page features a question section with a question about musical instruments and a choice between two options (a and b). A "Clear my choice" button is available. To the right, there's a sidebar titled "ტესტის შემსრულებელი" with a grid of numbered boxes (1-10) and a progress bar indicating 14:57 minutes completed.

At the bottom, there are "PREVIOUS ACTIVITY" and "NEXT ACTIVITY" buttons, and a search bar with the placeholder "გაძლიერებული შეძენები...". The footer contains a "Activate Windows" link and a copyright notice "© 2022 TSU".

25) რა ეწოდება კომპონენტს, რომელიც ფიზიკურ დონეზე ..

in თბილისის სისტემური უნივერსიტეტი (20) in ქუთა 1 - შეჯაცვალა 5 ქულა (+

e-learning.tsu.ge/mod/quiz/attempt.php?attempt=371613&cmid=192198&page=3

Search Courses

მთავრობის სახელმწიფო უნივერსიტეტი

Home Events My Courses This course Hide blocks Standard view

ჩემი კურსები > ოპერაციული სისტემები (2022) > პრეტრული დავლება & ჰეიზი > ქუთა 1 - შეჯაცვალა 5 ქულა

კურსი 4
მასტერი კურსი
აუტომატური განვითარებული სისტემები
1-ს კურსის მიზანები

რა ეწოდება კომპიუტერზე, რომელიც ფიზიკურ დონეზე არ ფიზიკურ კომპიუტერულ შერის უზრუნველყოფს მომატების გადაღების
ამართებულ ერთობა:

a. კონტროლერი

b. სალტე

c. მაგისტრალი

d. ბიფი

[Clear my choice](#)

Next page

26) როგორ პროცესს ეწოდება დემონი

როგორ პროცესს ეწოდება დემონი.

აირჩიეთ ერთი:

- a.
პროცესს, თუ ის მუშაობს ფონურ რეჟიმში
- b.
პროცესს, თუ ის წარმოიქნა სისტემის ჩატვირთვასთან ერთად
- c.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- d.
პროცესს, თუ ის დასრულების შემდეგ გარკვეული დროით დარჩა სისტემაში

27) სისტემაში წარმოქმნილ და წარმატებით დასრულებულ ყოველ პროცესს

სისტემაში წარმოქმნილ და წარმატებით დასრულებულ ყოველ პროცესს მდგრამარეობათა დიაგრამაზე გადააღვიდება შეუძლია შემდეგი მიმდევრობით

- აირჩიეთ ერთი:
- a.
დაბადება --> ბლოკირება --> მზადყოფნა --> შესრულება --> ბლოკირება --> დასრულება
 - b.
დაბადება --> მზადყოფნა --> შესრულება --> მზადყოფნა --> შესრულება --> დასრულება
 - c.
დაბადება --> ბლოკირება --> შესრულება --> მზადყოფნა --> დასრულება
 - d.
დაბადება --> მზადყოფნა --> შესრულება --> ბლოკირება --> მზადყოფნა --> დასრულება
 - e.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი



Clear my choice

28) რა ეწოდება პროგრამულ უზრუნველყოფას, რომელიც ოპერაციულ სისტემაში ამა თუ იმ დონეზე ხელს უწყობს ფიზიკური ან აბსტრაქტული კომპიუტერის ნორმალურ ფუნქციონირებას

აირჩიეთ ერთი:

- a.
სისტემური პროგრამა
- b.
ბირთვი
- c.
დრაივერი
- d.
ასემბლერი

[Clear my choice](#)

29) სტანდარტულად რამდენი კომპონენტისგან შედგება შეტანა/გამოტანის მოწყობილობა

სტანდარტულად რამდენი კომპონენტისგან შედგება შეტანა/გამოტანის მოწყობილობა

აირჩიეთ ერთი:

a.

2

b.

3

c.

5

d.

4

30) განსაკუთრებული შემთხვევის რომელ ტიპს მიეკუთვნება პროგრამის ..

განსაკუთრებული შემთხვევის რომელ ტიპს მიეკუთვნება პროგრამის შესრულების მიმდინარე მომენტში მებსიერების ფურცლის არარსებობა

აირჩიეთ ერთი:

a.

გამოუსწორებელი განსაკუთრებული შემთხვევა

b.

დამოკიდებული განსაკუთრებული შემთხვევა

c.

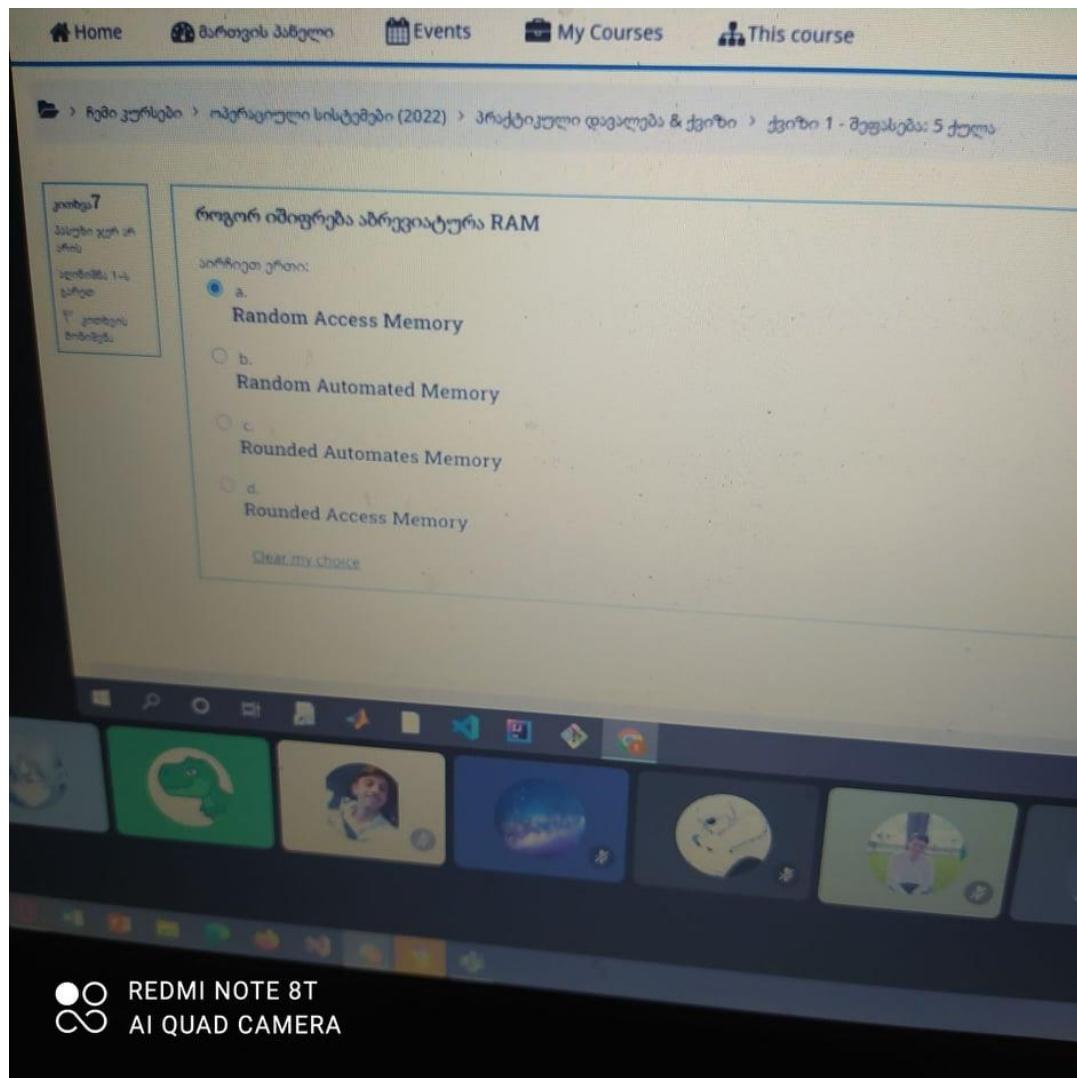
მართვადი განსაკუთრებული შემთხვევა

d.

გამოსწორებადი განსაკუთრებული შემთხვევა

[Clear my choice](#)

31) როგორ იშიფრება აბრევიატურა RAM



32) ჩამოთვლილთაგან რომელ შემთხვევაში შეიძლება აღმოჩნდეს პროცესი მზადყოფნის მდგომარეობაში

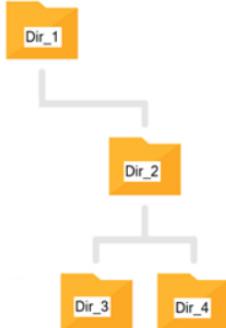
ჩამოთვლილთაგან რომელ შემთხვევაში შეიძლება აღმოჩნდეს პროცესი მზადყოფნის მდგომარეობაში

არჩივება ერთა:

- a.
თუ პროცესს დაბადების (სისტემაში გამოჩენის) შემდეგ მოთხოვთის შესაბამისად გამოყენ პროცესორის გარდა შესაბრულებლად საჭირო ყველა რესურსი
- b.
ბლოკირების მდგომარეობაში მყოფ პროცესს გამოყენ მისთვის საჭირო ყველა რესურსი
- c.
დროითი კვანტუს ამოწურვის გამო პროცესს ჩამოერთვა პროცესორი
- d.
ჩამოთვლილთაგან ყველა პასუხი სწორია (იგულისხმება უარყოფითი პასუხის გარდა ყველა)
- e.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი

33) ჩამოთვლილთაგან კოდის რომელი ფრაგმენტი იძლევა

ჩამოთვლილთაგან კოდის რომელი ფრაგმენტი იძლევა სურათზე ნაჩვენები იერარქიის შესაბამისი დირექტორიების სტრუქტურის მიღების საშუალებას



აირჩიეთ ერთი:

- a. `mkdir Dir_1 Dir_1/Dir_2/Dir_3 Dir_1/Dir_2 Dir_1/Dir_2/Dir_4`
- b. `mkdir Dir_1 Dir_1/Dir_2/Dir_3 Dir_1/Dir_2/Dir_4 Dir_1/Dir_2`
- c. `mkdir Dir_1/Dir_2 Dir_1/Dir_2/Dir_3 Dir_1/Dir_2/Dir_4`
- d. `mkdir Dir_1 Dir_1/Dir_2 Dir_1/Dir_2/Dir_4 Dir_1/Dir_2/Dir_3`

34) ჩამოთვლილთაგან რომელ დონეს მიეკუთვნება ამოცანა, რომელიც ელოდება შესრულებას

ჩამოთვლილთაგან რომელ დონეს მიეკუთვნება ამოცანა, რომელიც ელოდება შესრულებას

აირჩიეთ ერთი:

- a.
ზედა დონე
- b.
ქვედა დონე
- c.
შუალედური დონე

[Clear my choice](#)

35) ჩამოთვლილთაგან რომელიც წარმოადგენს SSD დისკის

ჩამოთვლილთაგან რომელიც წარმოადგენს SSD დისკის ნაკლოვანებას HDD დისკთან მიმართებით

აირჩიეთ ერთი:

- a.
SSD დისკი მონაცემები განთავსებულია წრიულად მზრუნვად დისკზე
- b.
SSD დისკი გააჩნია სიცოცხლის ნაკლები ხანგრძლივობა ვიდრე HDD დისკს
- c.
SSD დისკიდან მონაცემების კითხვა უფრო სწრაფია ვიდრე HDD დისკიდან
- d.
SSD დისკის სწრაფები საკმარისად ეცემა მონაცემების ფრაგმენტირებული განთავსებისას ვიდრე HDD დისკის შემთხვევაში

[Clear my choice](#)

36) წარმატებულად დასრულების შემთხვევაში რას აბრუნებს

წარმატებულად დასრულების შემთხვევაში რას აბრუნებს open სისტემური გამოძახება.

აირჩიეთ ერთი:

- a.
ფაილურ დესკრიპტორს
- b.
ფაილის გამსანელი პროცესის იდენტიფიკატორის მნიშვნელობას
- c.
ფაილში ჩაწერილი ბაიტების რაოდენობას
- d.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი

37) თუ პროცესორის დაგეგმვა ხდება სტატიკური პრიორიტეტის გამოყენებით

თუ პროცესორის დაგეგმვა ხდება სტატიკური პრიორიტეტის გამოყენებით, მაშინ ოპერაციულ სისტემაში მზადყოფნის მდგომარეობაში მყოფი პროცესების დალაგება მოხდება

აირჩიეთ ერთი:

- a.
პრიორიტეტების ზრდადი მნიშვნელობების მიხედვით
- b.
სისტემაში მათი გამოჩენის დროის მიხედვით
- c.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- d.
მათი შესრულებისათვის საჭირო დროის მიხედვით

38) ჩამოთვლილთაგან რომელი შეესაბამება სისტემაში არსებული ...

ჩამოთვლილთაგან რომელი შეესაბამება სისტემაში არსებული (data.in) ფაილის კითხვის უფლებით გახსნის მიზნით open სისტემური გამოძახების გამოყენების სწორ ფორმას

აირჩიეთ ერთი:

- a. `int fd = open("data.in", O_RDONLY);`
- b. `int fd = open(O_RDONLY, "data.in");`
- c. `int fd = open("data.in", O_RDONLY, O_EXCL | 0664);`
- d. `int fd = open(O_EXCL | 0664, "data.in", O_RDONLY);`

39) ჩამოთვლილთაგან რომელი წარმოადგენს გამოთვლით მანქანაში

ჩამოთვლილთაგან რომელი წარმოადგენს გამოთვლით მანქანაში გამოყენებულ ჯველაზე სწრაფ მუხსიერებას.

აირჩიეთ ერთიან:

- a.
M2 SSD დისკი
- b.
ოპერატორული მემორიება
- c.
SSD დისკი
- d.
რეგისტრი
- e.
ჭრი-მემორიება

[Clear my choice](#)

40) თუ პროცესორის დაგეგმვა ხდება სტატიკული პრიორიტეტის გამოყენებით ..

თუ პროცესორის დაგეგმვა ხდება სტატიკული პრიორიტეტის გამოყენებით, მაშინ ოპერაციულ სისტემაში მზადყოფნის მდგომარეობაში მყოფი პროცესების დალაგება მოხდება

აირჩიეთ ერთიან:

- a.
სისტემაში მათი გამოჩენის დროის მიხედვით
- b.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- c.
მათი შესრულებისათვის საჭირო დროის მიხედვით
- d.
პრიორიტეტების ზრდადი მნიშვნელობების მიხედვით

[Clear my choice](#)

41) ჩამოთვლილთაგან რომელი შეესაბამება ასინქრონული წყვეტის განმარტებას .

The screenshot shows a smartphone interface. At the top, there is a blue header bar with the text "Beeline GE" and a signal icon on the left, "10:56 AM" in the center, and "56%" with a battery icon on the right. Below the header is a white search bar with a magnifying glass icon and the text "კითხვის მონიშვნა". The main content area is a white box containing the following text and options:

**ჩამოთვლილთაგან რომელი
შეესაბამება ასინქრონული წყვეტის
განმარტებას.**

აირჩიეთ ერთი:

- a.
ეს არის წყვეტა, რომელიც
ნარმოიშვა კოდში დაშვებული
შეცდომის გამო.
- b.
ჩამოთვლილთაგან არცერთი
პასუხი არაა სწორი
- c.
ეს არის წყვეტა, რომელიც
ნარმოიშვა კლავიატურაზე
დაწკაპუნების შედეგად
- d.
ეს არის წყვეტა, რომელიც
ნარმოიშვა კოდში დაშვებული
შეცდომისგან დამოუკიდებლად

42) როგორ პროცესს ეწოდება დემონი

The screenshot shows a web browser window with the URL e-learning.tsu.ge/mod/quiz/attempt.php?attempt=371612&cmid=192198&page=1. The page title is "კვიზი 1 - შეფასება 5 ქულა". The navigation bar includes "Home", "My Courses", "Events", "This course", "Hide blocks", and "Standard view". Below the navigation is a breadcrumb trail: "ჩიტო კურსები > ოპერაციული სისტემი (2022) > ასაქტუალი დაცულება & ჰარიზ. > კვიზი 1 - შეფასება 5 ქულა".

The main content area contains a question: "როგორ პროცესს ეწოდება დემონი." Below the question is a list of four options (a, b, c, d) with radio buttons:

- a. პროცესი, თუ ის მუშაობს ფონურ რეჟიმში
- b. პროცესი, თუ ის წარმოიქმნა სისტემის ჩატვირთვასთან ერთად
- c. ჩამოთვლილთავან არყენთი პასუხი არაა სწორი
- d. პროცესი, თუ ის დასრულების შემდეგ გარკვეული დროით დარჩა სისტემაში

[Clear my choice](#)

On the right side, there is a grid of numbered boxes from 1 to 10, with boxes 9 and 10 highlighted. Below the grid is the text "დასახულეთ მცდელობა" and "დარწევილობის დრო: 0:19:32".

At the bottom, there are navigation links: "PREVIOUS ACTIVITY" (დაცულება 1 - შეფასება 10 ქულა) and "NEXT ACTIVITY" (სემინარი 1. UNIX იმპლემენტაციები). A search bar "გადახტი შემდეგი... " is also present at the bottom.

1) ჩამოთვლილთაგან რომელი სისტემური გამოძახება გამოიყენება

ჩამოთვლილთაგან რომელი სისტემური გამოძახება გამოიყენება მომხმარებლის ჯგუფის იდენტიფიკაციორის მნიშვნელობის მისაღებად.

აირჩიეთ ერთი:

- a.
getgroup
- b.
getpid
- c.
getuid
- d.
getgid
- e.
getppid

[Clear my choice](#)

2) ვთქვათ ერთპროცესორულ სისტემაში ...

ვთქვათ ერთპროცესორულ სისტემაში პროცესი იმყოფება შესრულების მდგომარეობაში. ჩამოთვლილთაგან რომელი წინადადებაა ჭეშმარიტი.

აირჩიეთ ერთი:

- a.
შესრულების მდგომარეობაში მყოფი პროცესის საქმიანობის ბლოკირება შეუძლია მიმდინარე (იმავე) პროცესს
- b.
შესრულების მდგომარეობაში მყოფი პროცესის საქმიანობის ბლოკირება შეუძლია მზადყოფნის მდგომარეობაში მყოფ პროცესს
- c.
ყველა ჩამოთვლილი შემთხვევა ჭეშმარიტია (იგულისხმება უარყოფის გარდა ყველა შემთხვევა)
- d.
ჩამოთვლილთაგან არცერთი პასუხი არა სწორი
- e.
შესრულების მდგომარეობაში მყოფი პროცესის საქმიანობის ბლოკირება შეუძლია ბლოკირების მდგომარეობაში მყოფ პროცესს

[Clear my choice](#)

3) ჩამოთვლილთაგან რომელი შეესაბამება ასინქრონულ წყვეტის განმარტებას.

ჩამოთვლილთაგან რომელი შეესაბამება ასინქრონული წყვეტის განმარტებას.

აირჩიეთ ერთი:

- a.
ეს არის წყვეტა, რომელიც წარმოიშვა კოდში დაშვებული შეცდომის გამო.
- b.
ეს არის წყვეტა, რომელიც წარმოიშვა კოდში დაშვებული შეცდომისგან დამოუკიდებლად
- c.
ეს არის წყვეტა, რომელიც წარმოიშვა კლავიატურაზე დაწვაპუნების შედეგად
- d.
ჩამოთვლილთაგან არცერთი პასუხი არა სწორი

4) ჩამოთვლილთაგან რომელ შემთხვევაში შეიძლება პროცესი მზადყოფნის მდგომარეობაში აღმოჩნდეს

ჩამოთვლილთაგან რომელ შემთხვევაში შეიძლება აღმოჩნდეს პროცესი მზადყოფნის მდგომარეობაში

აირჩიეთ ერთი:

- a.
თუ პროცესს დაბადების (სისტემაში გამოჩენის) შემდეგ მოთხოვნის შესაბამისად გამოეყო პროცესორის გარდა შესასრულებლად საჭირო ყველა რესურსი
- b.
ბლოკირების მდგომარეობაში მყოფ პროცესს გამოეყო მისთვის საჭირო ყველა რესურსი
- c.
დროითი კვანტის ამოწურვის გამო პროცესს ჩამოერთვა პროცესორი
- d.
ჩამოთვლილთაგან ყველა პასუხი სწორია (იგულისხმება უარყოფითი პასუხის გარდა ყველა)
- e.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი

5) რომელი შესაბამება write სისტემური ...

ჩამოთვლილთაგან რომელი შესაბამება write სისტემური გამოძახების გამოყენების სწორ ფორმას, თუ N მასივში ელემენტების რაოდენობაა, ხოლო fd კი - შესაბამისი ნაკადის სახელი

აირჩიეთ ერთი:

- a. **char buf[N];
write(buf, fd, N);**
- b. **char buf[N];
write(N, fd, buf);**
- c. **char buf[N];
write(N, buf, fd);**
- d. **char buf[N];
write(fd, buf, N);**

6) 0532 დაშვების

0532 დაშვების უფლებების რვაობითი ჩანაწერის შესაბამის სიმბოლურ ჩანაწერს ექნება სახე:

აირჩიეთ ერთი:

- a.
-IW-I-X-WX
- b.
-I-XIW--WX
- c.
-I-XI---W-
- d.
--WXI--IW-
- e.
-I-X-WX-W-

8) რა ეწოდება აბსტრაქციას ,რომელიც გამოთვლით სისტემაში ..

რა ეწოდება აბსტრაქციას ,რომელიც გამოთვლით სისტემაში ნაკლები მოცულობის ფიზიკური მეხსიერების არსებობის შემთხვევაში ხელს უწყობს მეტი ფიზიკური მეხსიერების საჭიროების მქონე პროგრამის ამუშავებას

აირჩიეთ ერთი:

- a.
thread-ები
- b.
ფაილური სისტემა
- c.
ვირტუალიზაცია
- d.
ვირტუალური მეხსიერება

9) კავშირის არხის დასრულების პროცესი ...

კავშირის არხის დასრულების პროცესი რა შემთხვევაში არაა აუცილებელი

აირჩიეთ ერთი:

- a.
წამოთვლილთაგან ყველა პასუხი სწორია (იგულისხმება უარყოფითი პასუხის გარდა ყველა)
- b.
წამოთვლილთაგან არცერთი პასუხი არა სწორი
- c.
პროცესებს შორის კომუნიკაციის დასრულების შემდეგ
- d.
თუ პროცესებს შორის კავშირის არხის გახსნა პროცესი მოქმედებებით
- e.
თუ პროცესებს შორის კავშირის არხის გახსნა არ იყო პროცესი მოქმედებებით

Please answer choices

10) განსაკუთრებული შემთხვევის რომელ ტიპს მიეკუთვნება ..

განსაკუთრებული შემთხვევის რომელ ტიპს მიეკუთვნება პროგრამის შესრულების მიმდინარე მომენტში ისეთ ფაილზე მიმართვა, რომელიც ფაილურ სისტემაში არსებობს, მაგრამ სხვა დირექტორიაში არის განთავსებული

არჩიეთ ერთი:

- a.
მართვადი განსაკუთრებული შემთხვევა
- b.
დამოკიდებული განსაკუთრებული შემთხვევა
- c.
გამოუსწორებელი განსაკუთრებული შემთხვევა
- d.
გამოსწორებადი განსაკუთრებული შემთხვევა

[Clear my choice](#)

11) ჩამოთვლილთაგან რომელი ბრძანებითაა შესაძლებელი ფაილის

ჩამოთვლილთაგან რომელი ბრძანებითაა შესაძლებელი ფაილის შიგთავსის დათვალიერება მისი გახსნის გარეშე.

არჩიეთ ერთი:

- a.
pwd
- b.
mkdir
- c.
cd
- d.
ls
- e.
cat

[Clear my choice](#)

12) თუ მომხმარებელმა გადაწყვიტა ოპერაციულ სისტემის

თუ მომხმარებელმა გადაწყვიტა ოპერაციულ სისტემის სწრაფებების ამაღლების მიზნით პროცესების გარკვეული ჯგუფი იძულებით გააჩეროს („დააპაუზოს“), მაშინ შესაბამისი პროცესები აღმოჩნდებიან მდგომარეობაში

არჩიეთ ერთი:

- a.
შეჩერებული ბლოკირებული
- b.
ბლოკირებული
- c.
შეჩერებული მზად
- d.
დაპაუზებული

13) ჩამოთვლილთაგან რომელი შეესაბამება საქმიანობას ..

ჩამოთვლილთაგან რომელი შექსაბამება საქმიანობას რომლითაც დაკავებული ბრძანებათა მთვლელი?

აირჩიეთ ერთი:

- a. ინახავს კონკრეტული ერთი პროგრამის ამუშავების შემდეგ ამავე პროგრამის წარმატებული საქმიანობის გაგრძელებისთვის რიგით შემდეგი შესრულებული ბრძანების მიმთითებელს
- b. ინახავს კონკრეტული ერთი პროგრამის ამუშავების შემდეგ მიმდინარე დრომდე ამავე პროგრამის წარმატებული საქმიანობისთვის შესრულებული ბრძანებების რაოდენობას
- c. ინახავს კონკრეტული ერთი პროგრამის ამუშავების შემდეგ ამავე პროგრამის წარმატებული საქმიანობის გაგრძელებისთვის რიგით შემდეგი შესრულებული რამდენიმე ბრძანების მიმთითებელს
- d. ინახავს ოპერაციული სისტემის ამუშავების მომენტიდან მიმდინარე მომენტამდე სხვადასხვა პროგრამების წარმატებული საქმიანობისთვის შესრულებული ბრძანებების საერთო რაოდენობას
- e. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი

14) ჩამოთვლილთაგან რომელი წარმოადგენს მონოლიტური ..

ჩამოთვლილთაგან რომელი წარმოადგენს მონოლიტური არქიტექტურის ნაკლოვანებას ?

აირჩიეთ ერთი:

- a. ყველა პროცედურა ერთმანეთთან ურთიერთქმედებს მთავარი პროცედურის გამოყენებით
- b. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- c. პროცედურებს არ შეუძლიათ ზეგავლენა იქნიონ სხვა პროცედურების საქმიანობაზე
- d. პროცედურათა ნაკრების სახით შემუშავებული ოპერაციული სისტემის კოდი წელა მუშაობს

[Clear my choice](#)

15) ჩამოთვლილთაგან რომელი ბრძანების გამოყენებით არის შესაძლებელი ფაილზე

..

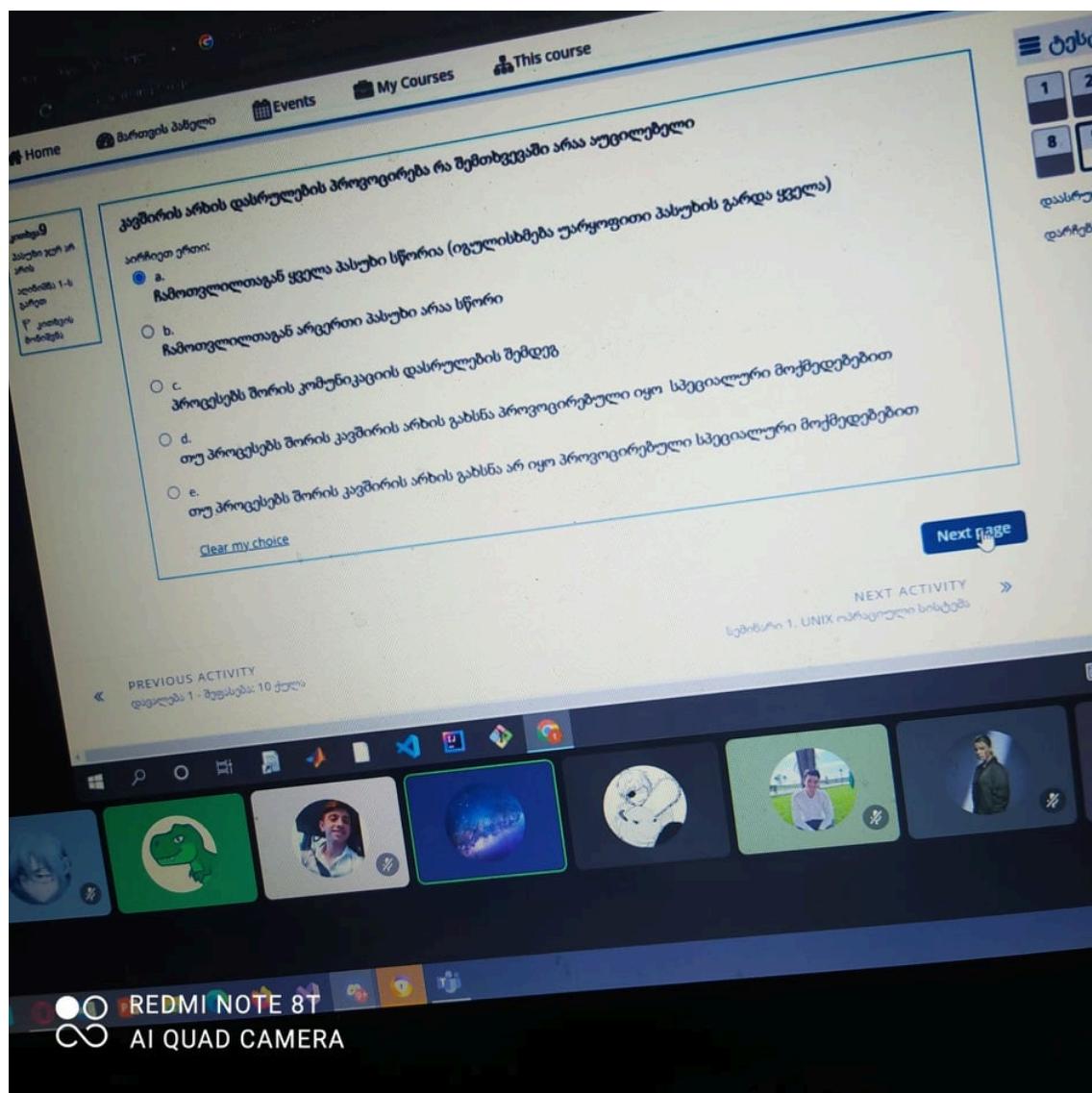
ჩამოთვლილთაგან რომელი ბრძანების გამოყენებით არის შესაძლებელია ფაილზე დაშვების უფლებების შეცვლა

არჩიეთ ერთი:

- a.
chmod
- b.
chgrp
- c.
chusr
- d.
modes
- e.
umask

[Clear my choice](#)

16) კავშირის არხის დასრულების



17) წყვეტის რომელ ტიპს

წყვეტის რომელ ტიპს მიეკუთვნება კოდში დაშვებული შეცდომის გამო წარმოქმნილი წყვეტა

აირჩიეთ ერთი:

- a. ასინქრონულ წყვეტას
- b. პროცესთაშორის წყვეტას
- c. სინქრონულ წყვეტას
- d. ტაიმერიდან წყვეტას

18) ჩამოთვლილთაგან რომელი წინადადებაა ჭეშმარიტი FIFO კავშირის არხთან
მიმართებით

ჩამოთვლილთაგან რომელი წინადადებაა ჭეშმარიტი FIFO კავშირის არხთან მიმართებით.

აირჩიეთ ერთი:

- a. FIFO კავშირის არხის გამოყენება შეუძლია სისტემაში წარმოქმნილ ყველა პროცესს მიუხედავად იმისა როდის ან რომელი პროცესის მიერ მოხდა FIFO კავშირის არხის წარმოქმნა
- b. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- c. FIFO კავშირის არხის გამოყენება შეუძლია სისტემაში წარმოქმნილ ყველა იმ პროცესს, რომელთა მეშვეობითაც წარმოიქმნა FIFO კავშირის არხის წარმომექმნელი პროცესი
- d. FIFO კავშირის არხის გამოყენება შეუძლია სისტემაში წარმოქმნილ ყველა იმ პროცესს, რომელსაც ყავს FIFO კავშირის არხის წარმომექმნელი საერთო წინაპარი

19) ვთქვათ ერთპროცესორულ სისტემაში პროცესი იმყოფება შესრულების
მდგომარეობაში ..

ვთქვათ ერთპროცესორულ სისტემაში პროცესი იმყოფება შესრულების მდგომარეობაში. ჩამოთვლილთაგან რომელი წინადადებაა ჭეშმარიტი.

აირჩიეთ ერთი:

- a. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- b. შესრულების მდგომარეობაში მყოფი პროცესის საქმიანობის ბლოკირება შეუძლია ბლოკირების მდგომარეობაში მყოფ პროცესს
- c. ყველა ჩამოთვლილი შემთხვევა ჭეშმარიტია (იგულისხმება უარყოფის გარდა ყველა შემთხვევა)
- d. შესრულების მდგომარეობაში მყოფი პროცესის საქმიანობის ბლოკირება შეუძლია მიმდინარე (იმავე) პროცესს
- e. შესრულების მდგომარეობაში მყოფი პროცესის საქმიანობის ბლოკირება შეუძლია მზადყოფნის მდგომარეობაში მყოფ პროცესს

[Clear my choice](#)

22) ვთქვათ აპარატურულ სისტემის შექმნისას პროგრამისტმა გამოიყენა ფიქსირებული ზომის სტრუქტურა გარკვეული ტიპის მონაცემების განსათავსებლად, როგორიც დინამიურად იცხება ელემენტებით და თავისუფლდება (იშლება) ელემენტებისგან. როგორი ტიპის განსაკუთრებულ შემთხვევას მიეკუთვნება სიტუაცია, რომლის დროსაც სტრუქტურაში მონაცემების ჩაწერისას მასში შეიძლება არ იყოს ადგილი ახალი მონაცემის განსათავსებლად.

აირჩიეთ ერთი:

- a.
გამოუსწორებელს
- b.
გამოსწორებადს

[Clear my choice](#)

23) ჩამოთვლილთაგან რომელი წარმოადგენს მონოლიტური არქიტექტურის ..

The screenshot shows a web browser window with several tabs open at the top. The active tab displays a question from an e-learning platform. The question asks which of four options represents a monolithic architecture. The options are:

- a. ფელა პროცესურა ერთმანეთთან ურთიერთებული მთავარი პროცედურის გამოყენებით
- b. ჩამოთვლილთაგან არცერთი პასუხი არა სწორი
- c. პროცედურებს არ შეუძლიათ ზუგადურა იქანონობის სხვა პროცედურების საქმიანობაზე
- d. პროცედურათა ნაკრების სახით შემუშავებული იპერაციელი სისტემის კოდი წელა მუშაობს

Below the question, there are navigation links: 'PREVIOUS ACTIVITY' and 'NEXT ACTIVITY'. At the bottom of the screen, the phone status bar is visible, showing the brand 'REDMI NOTE 8T' and 'AI QUAD CAMERA'.

24) ვთქვათ ოპერაციული სისტემის შექმნისას პროგრამისტმა

The screenshot shows a Moodle-based e-learning platform. At the top, there's a green header bar with the title "კურსი 1 - მუდული 5 ქულა" and a URL "e-learning.tsu.ge/mod/quiz/attempt.php?attempt=371514&cmid=192198&page=2". The main content area has a dark blue header with the text "მართვის სამსახურის სახელმწიფო უნივერსიტეტი ენგურის განმარტებული ცენტრი" and a search bar "Search Courses". Below the header, there are navigation links: "Home", "My Institutions", "Events", "My Courses", and "This course". On the right, there are buttons for "Hide blocks" and "Standard view".

The main content area displays a quiz question:

რეზიუმე მომსახურებულ სისტემის შემსრულებელი როლის სტრუქტურა გრადუსის განვითარებული ტენის მომსახურების გამსასიქრებად, რომელიც დამტკიცებული იქნება კულტურული და თავისუფლების (მღლის) ულტრანინებების. როგორი ტენის გამსასიქრებული შემსრულებელი მომკურნეობა სტრუქტურა, რომელს დასაცავ სტრუქტურული მომსახურების ჩატარების მასში შეიძლება არ იყოს აღვრით ასაღი მომსახურებელისად.

არჩევის გზა:

a. გამოიყენონ მურავის
 b. გამოიწყონ მარცალს

[Clear my choice](#)

Below the question, there are "Next page" and "Previous activity" buttons. The "Previous activity" button points to "დაგლობი 1 - შუალედი 5 ქულა". The "Next activity" button points to "სტრუქტური 1. UNIX მუდული 5 ქულა".

At the bottom, there's a message about activating Windows: "Активация Windows" and "Чтобы активировать Windows, перейдите в раздел 'Параметры'".

25) რა ეწოდება კომპონენტს, რომელიც ფიზიკურ დონეზე ...

ოპერაციული სისტემები (202) > ქვიზი 1 - შეფასება: 5 ქულა () +

e-learning.tsu.ge/mod/quiz/attempt.php?attempt=371613&cmid=192198&page=3

თბილისის სახელმწიფო უნივერსიტეტი

 Home Events My Courses This course

ჩემი კურსები > ოპერაციული სისტემები (2022) > პრაქტიკული დავალება & ქვიზი > ქვიზი 1 - შეფასება: 5 ქულა

კვიზი 4
მასშტაბით არა არ
არის
აღნიშვნა 1-ს
გარეთ
„კითხვის
მონიტორი“

რა ეწოდება კომპონენტს, რომელიც ფიზიკურ დონეზე ორ ფიზიკურ კომპონენტებს შორის უზრუნველყოფს მონაცემების გადაცემას

არჩივეთ ერთია:

- a.
კონტროლერი
- b.
სალტე
- c.
მაგისტრალი
- d.
ბიდი

[Clear my choice](#)

1 2
9 10
დასრულე დარჩენილი



26) როგორ პროცესს ეწოდება დემონი

როგორ პროცესს ეწოდება დემონი.

აირჩიეთ ერთი:

- a.
პროცესს, თუ ის მუშაობს ფონურ რეჟიმში
- b.
პროცესს, თუ ის წარმოიქნა სისტემის ჩატვირთვასთან ერთად
- c.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- d.
პროცესს, თუ ის დასრულების შემდეგ გარკვეული დროით დარჩა სისტემაში

27) სისტემაში წარმოქმნილ და წარმატებით დასრულებულ ყოველ პროცესს

სისტემაში წარმოქმნილ და წარმატებით დასრულებულ ყოველ პროცესს მდგომარეობათა დიაგრამაზე გადაადგილება შეუძლია შემდეგი მიმდევრობით

აირჩიეთ ერთი:

- a.
დაბადება --> ბლოკირება --> მზადყოფნა --> შესრულება --> ბლოკირება --> დასრულება
- b.
დაბადება --> მზადყოფნა --> შესრულება --> მზადყოფნა --> შესრულება --> დასრულება
- c.
დაბადება --> ბლოკირება --> მზადყოფნა --> შესრულება --> მზადყოფნა --> დასრულება
- d.
დაბადება --> მზადყოფნა --> შესრულება --> ბლოკირება --> მზადყოფნა --> დასრულება
- e.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი



28) რა ეწოდება პროგრამულ უზრუნველყოფას, რომელიც ოპერაციულ სისტემაში ამა თუ იმ დონეზე ხელს უწყობს ფიზიკური ან აბსტრაქტული კომპონენტის ნორმალურ ფუნქციონირებას

აირჩიეთ ერთი:

- a.
სისტემური პროგრამა
- b.
ბირთვი
- c.
დრაივერი
- d.
ასემბლერი

[Clear my choice](#)

29) სტანდარტულად რამდენი კომპონენტისგან შედგება შეტანა/გამოტანის მოწყობილობა

სტანდარტულად რამდენი კომპონენტისგან შედგება შეტანა/გამოტანის მოწყობილობა

აირჩიეთ ერთი:

- a.
2
- b.
3
- c.
5
- d.
4

30) განსაკუთრებული შემთხვევის რომელ ტიპს მიეკუთვნება პროგრამის ..

განსაკუთრებული შემთხვევის რომელ ტიპს მიეკუთვნება პროგრამის შესრულების მიმდინარე მომენტში მეხსიერების ფურცლის არარსებობა

აირჩიეთ ერთი:

- a.
გამოუსწორებელი განსაკუთრებული შემთხვევა
- b.
დამოკიდებული განსაკუთრებული შემთხვევა
- c.
მართვადი განსაკუთრებული შემთხვევა
- d.
გამოსწორებადი განსაკუთრებული შემთხვევა

[Clear my choice](#)

31) როგორ იშიფრება აბრევიატურა RAM

The screenshot shows a mobile application interface. At the top, there is a navigation bar with icons for Home, Events, My Courses, and This course. Below the navigation bar, the path is displayed: ჩემი კურსები > ინგლისური სიტყვაში (2022) > პრაქტიკული დავალება & ტესტი > ტესტი 1 - შეჯასება: 5 ქულა.

The main content area displays a question:

როგორ იშიფრება აბრევიატურა RAM

პირდისით ერთია:

- a. Random Access Memory
- b. Random Automated Memory
- c. Rounded Automates Memory
- d. Rounded Access Memory

[Clear my choice](#)

At the bottom of the screen, there is a status bar showing the phone model: REDMI NOTE 8T AI QUAD CAMERA.

32) ჩამოთვლილთაგან რომელ შემთხვევაში შეიძლება აღმოჩნდეს პროცესი მზადყოფნის მდგომარეობაში

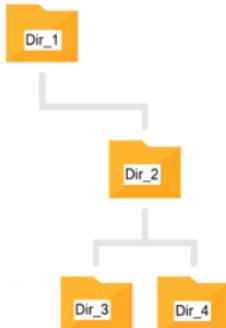
ჩამოთვლილთაგან რომელ შემთხვევაში შეიძლება აღმოჩნდეს პროცესი მზადყოფნის მდგომარეობაში

აირჩიეთ ერთი:

- a. თუ პროცესს დაბადების (სისტემაში გამოჩენის) შემდეგ მოთხოვნის შესაბამისად გამოიყო პროცესორის გარდა შესასრულებლად საჭირო ყველა რესურსი
- b. ბლოკურების მდგომარეობაში შემდეგ მოთხოვნის შესაბამისად გამოიყო პროცესორის გარდა შესასრულებლად საჭირო ყველა რესურსი
- c. დროითი კვანტის ამოწურვის გამო პროცესს ჩამოერთვა პროცესორი
- d. ჩამოთვლილთაგან ყველა პასუხი სწორია (იგულისხმება უარყოფითი პასუხის გარდა ყველა)
- e. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი

33) ჩამოთვლილთაგან კოდის რომელი ფრაგმენტი იძლევა

ჩამოთვლილთაგან კოდის რომელი ფრაგმენტი იძლევა სურათზე ნაჩვენები იერარქიის შესაბამისი დირექტორიების სტრუქტურის მიღების საშუალებას



აირჩიეთ ერთი:

- a. `mkdir Dir_1 Dir_1/Dir_2/Dir_3 Dir_1/Dir_2 Dir_1/Dir_2/Dir_4`
- b. `mkdir Dir_1 Dir_1/Dir_2/Dir_3 Dir_1/Dir_2/Dir_4 Dir_1/Dir_2`
- c. `mkdir Dir_1/Dir_2 Dir_1/Dir_2/Dir_3 Dir_1/Dir_2/Dir_4`
- d. `mkdir Dir_1 Dir_1/Dir_2 Dir_1/Dir_2/Dir_4 Dir_1/Dir_2/Dir_3`

34) ჩამოთვლილთაგან რომელ დონეს მიეკუთვნება ამოცანა, რომელიც ელოდება შესრულებას

ჩამოთვლილთაგან რომელ დონეს მიეკუთვნება ამოცანა, რომელიც ელოდება შესრულებას

აირჩიეთ ერთი:

- a. ზედა დონე
- b. ქვედა დონე
- c. შუალედური დონე

[Clear my choice](#)

როგორ პროცესს ეწოდება ზომბი

როგორ პროცესს ეწოდება ზომბი.

აირჩიეთ ერთი:

- a.
პროცესს, თუ ის დასრულების შემდეგ გარკვეული დროით დარჩა სისტემაში
- b.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- c.
პროცესს, თუ ის წარმოიქნა სისტემის ჩატვირთვასთან ერთად
- d.
პროცესს, თუ ის მუშაობს ფონურ რეჟიმში

35) ჩამოთვლილთაგან რომელიც წარმოადგენს SSD დისკის

ჩამოთვლილთაგან რომელიც წარმოადგენს SSD დისკის ნაკლოვანებას HDD დისკთან მიმართებით

აირჩიეთ ერთი:

- a.
SSD დისკი მონაცემები განთავსებულია წრიულად მხრუნავ დისკოზე
- b.
SSD დისკს გააჩნია სიცოცხლის ნაკლები ხანგრძლივობა ვიდრე HDD დისკს
- c.
SSD დისკიდან მონაცემების კითხვა უფრო სწრაფია ვიდრე HDD დისკიდან
- d.
SSD დისკის სწრაფებულება საკმარისად ეცემა მონაცემების ფრაგმენტირებული განთავსებისას ვიდრე HDD დისკის შემთხვევაში

[Clear my choice](#)

36) წარმატებულად დასრულების შემთხვევაში რას აბრუნებს

წარმატებულად დასრულების შემთხვევაში რას აბრუნებს open სისტემური გამოძრევა.

აირჩიეთ ერთი:

- a.
ფაილურ დესკრიპტორს
- b.
ფაილის გამხსნელი პროცესის იდენტიფიკატორის მნიშვნელობას
- c.
ფაილში ჩაწერილი ბაიტების რაოდენობას
- d.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი

37) თუ პროცესორის დაგეგმვა ხდება სტატიკური პრიორიტეტის გამოყენებით

თუ პროცესორის დაგეგმვა ხდება სტატიკური პრიორიტეტის გამოყენებით, მაშინ ოპერაციულ სისტემაში მზადყოფნის მდგომარეობაში მყოფი პროცესების დალაგება მოხდება

აირჩიეთ ერთი:

- a.
პრიორიტეტების ზრდადი მნიშვნელობების მიხედვით
- b.
სისტემაში მათი გამოჩენის დროის მიხედვით
- c.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- d.
მათი შესრულებისათვის საჭირო დროის მიხედვით

38) ჩამოთვლილთაგან რომელი შეესაბამება სისტემაში არსებული ...

ჩამოთვლილთაგან რომელი შეესაბამება სისტემაში არსებული (data.in) ფაილის კითხვის უფლებით გახსნით open სისტემური გამოძახების გამოყენების სწორ ფორმას

აირჩიეთ ერთი:

- a. `int fd = open("data.in", O_RDONLY);`
- b. `int fd = open(O_RDONLY, "data.in");`
- c. `int fd = open("data.in", O_RDONLY, O_EXCL | 0664);`
- d. `int fd = open(O_EXCL | 0664, "data.in", O_RDONLY);`

39) ჩამოთვლილთაგან რომელი წარმოადგენს გამოთვლით მანქანაში

ჩამოთვლილთაგან რომელი წარმოადგენს გამოთვლით მანქანაში გამოყენებულ ფერაზე სწორ შესიტუაცია.

- აირჩიეთ ერთი:
- a.
M2 SSD დისკი
 - b.
ოპტიკული მემორიები
 - c.
SSD დისკი
 - d.
რეგისტრი
 - e.
ტექ-მეხისიერება

[Clear my choice](#)

40) თუ პროცესორის დაგეგმვა ხდება სტატიკური პრიორიტეტის გამოყენებით ..

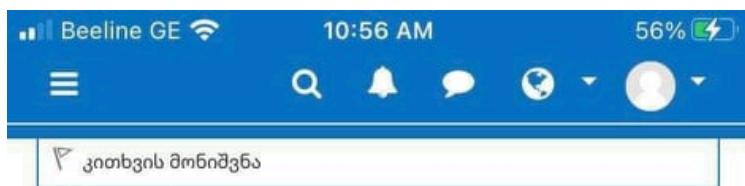
თუ პროცესორის დაგეგმვა ხდება სტატიკური პრიორიტეტის გამოყენებით, მაშინ ოპერაციულ სისტემაში მზადყოფნის მდგრადი მოძრავი მოხდება

აირჩიეთ ერთი:

- a.
სისტემაში მათი გამოჩენის დროის მიხედვით
- b.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- c.
მათი შესრულებისათვის საჭირო დროის მიხედვით
- d.
პრიორიტეტის ზრდადი მნიშვნელობების მიხედვით

[Clear my choice](#)

41) ჩამოთვლილთაგან რომელი შეესაბამება ასინქრონული წყვეტის განმარტებას .



ჩამოთვლილთაგან რომელი შეესაბამება ასინქრონული წყვეტის განმარტებას.

აირჩიეთ ერთი:

- a.
ეს არის წყვეტა, რომელიც
ნარმოიშვა კოდში დაშვებული
შეცდომის გამო.
- b.
ჩამოთვლილთაგან არცერთი
პასუხი არაა სწორი
- c.
ეს არის წყვეტა, რომელიც
ნარმოიშვა კლავიატურაზე
დაწკაპუნების შედეგად
- d.
ეს არის წყვეტა, რომელიც
ნარმოიშვა კოდში დაშვებული
შეცდომისგან დამოუკიდებლად

42) როგორ პროცესს ეწოდება დემონი

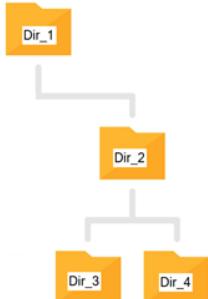
The screenshot shows a web browser window with a quiz interface. The URL is e-learning.tsu.ge/mod/quiz/attempt.php?attempt=371612&cmid=192198&page=1. The page title is "კვიზი 1 - უცასება 5 ჭულა". The navigation bar includes "Home", "My Courses", and "This course". On the right, there are buttons for "Hide blocks" and "Standard view".
The main content area displays a question: "როგორ პროცესს ეწოდება დემონი?". Below it is a list of four options:

- a. პროცესს, თუ ის მუშაობს ფონურ რეჟიმში
- b. პროცესს, თუ ის წირმოიქან სისტემის ჩატვირთვასთან ერთად
- c. ჩამოთვლილთაგან არღერთი პასუხი არაა სწორი
- d. პროცესს, თუ ის დასრულების შემდეგ გარკვეული დროით დარჩა სისტემაში

Below the options are two buttons: "Clear my choice" and "Next page".
At the bottom, there are navigation links for "PREVIOUS ACTIVITY" (დადალა 1 - უცასება: 10 ჭულა) and "NEXT ACTIVITY" (სუმინარი 1. UNIX იმპლემენტაციები). A search bar and a taskbar are visible at the very bottom.

ჩამოთვლილთაგან კოდის რომელი ფრაგმენტი იძლევა სურათზე ..

ჩამოთვლილთაგან კოდის რომელი ფრაგმენტი იძლევა სურათზე ნაჩვენები იერარქიის შესაბამისი დირექტორიების სტრუქტურის მიღების საშუალებას



აირჩიეთ ერთი:

- a. `mkdir Dir_1/Dir_2 Dir_1/Dir_2/Dir_3 Dir_1/Dir_2/Dir_4`
- b. `mkdir Dir_1 Dir_1/Dir_2 Dir_1/Dir_2/Dir_4 Dir_1/Dir_2/Dir_3`
- c. `mkdir Dir_1 Dir_1/Dir_2/Dir_3 Dir_1/Dir_2/Dir_4 Dir_1/Dir_2`
- d. `mkdir Dir_1 Dir_1/Dir_2/Dir_3 Dir_1/Dir_2 Dir_1/Dir_2/Dir_4`

The screenshot shows a web browser window with several tabs open. The main content area displays a programming quiz. A sidebar on the left contains navigation links for Home, My courses, Events, and This course, along with a search bar and a 'Hide blocks' button.

Question 1: მართვა დამუშავება

აღნიშვნა 0 1 - ს გარეთ წარადგინება

ა. დამოუსწორებული შემთხვევის რომელ ტაქს მიეკუთვნება პროგრამის შესრულების მიმდინარე მომენტში ისეთ ფაილზე მიმართვა, რომელიც ფაილზე სისტემში არსებობს, მაგამ სხვა დირექტორიაში არის განთავსებული

არჩიეთ ერთი:

- a. დამოკიდებული განსაკუთრებული შემთხვევა
- b. მართვადი განსაკუთრებული შემთხვევა
- c. გამოუსწორებული განსაკუთრებული შემთხვევა
- d. გამოსწორებადი განსაკუთრებული შემთხვევა

Question 2: გამოუსწორებული განსაკუთრებული შემთხვევა

აღნიშვნა 1 1 - ს გარეთ წარადგინება

ა. გამოთვლილან რომელი წარმოადგის გამოთვლით მანქანაში გამოყენებულ ყველაზე სწრაფ მეხსიერებას.

არჩიეთ ერთი:

- a. გამოთვლილან მეხსიერება

EN 05.05.2022 05:33

განსაკუთრებული შემთხვევის რომელ ტიპს მიეკუთვნება პროგრამის შესრულების მიმდინარე მომენტში ისეთ ფაილზე მიმართვა, რომელიც ფაილურ სისტემაში არსებობს, მაგრამ სხვა დირექტორიაში არის განთავსებული

განსაკუთრებული შემთხვევის რომელ ტიპს მიეკუთვნება პროგრამის შესრულების მიმდინარე მომენტში ისეთ ფაილზე მიმართვა, რომელიც ფაილურ სისტემაში არსებობს, მაგრამ სხვა დირექტორიაში არის განთავსებული

აირჩიეთ ერთი:

- a. დამოკიდებული განსაკუთრებული შემთხვევა
- b. მართვადი განსაკუთრებული შემთხვევა
- c. გამოუსწორებული განსაკუთრებული შემთხვევა
- d. გამოსწორებადი განსაკუთრებული შემთხვევა

სწორი პასუხი:

გამოუსწორებული განსაკუთრებული შემთხვევა

ნიშანი ნაკვეთი

ჩამოთვლილთაგან რომელი წარმოადგენს გამოთვლით მანქანაში გამოყენებულ ყველაზე სწრაფ მეხსიერებას.

ჩამოთვლილთაგან რომელი წარმოადგენს გამოთვლით მანქანაში გამოყენებულ ყველაზე სწრაფ მეხსიერებას.

აირჩიეთ ერთი:

- a.
ოპერატორული მეხსიერება
- b.
SSD დისკი
- c.
M2 SSD დისკი
- d.
ქეშ-მეხსიერება
- e.
რეგისტრი

განსაკუთრებული შემთხვევის რომელ ტიპს მიეკუთვნება პროგრამის შესრულების მიმდინარე მომენტში მეხსიერების ფურცლის არარსებობა

განსაკუთრებული შემთხვევის რომელ ტიპს მიეკუთვნება პროგრამის შესრულების მიმდინარე მომენტში მეხსიერების ფურცლის არარსებობა

აირჩიეთ ერთი:

- a.
გამოსწორებადი განსაკუთრებული შემთხვევა
- b.
დამოკიდებული განსაკუთრებული შემთხვევა
- c.
მართვადი განსაკუთრებული შემთხვევა
- d.
გამოუსწორებელი განსაკუთრებული შემთხვევა

წყვეტის რომელ ტიპს მიეკუთვნება კოდში დაშვებული შეცდომის გამო წარმოქმნილი წყვეტა

წყვეტის რომელ ტიპს მიეკუთვნება კოდში დაშვებული შეცდომის გამო წარმოქმნილი წყვეტა

აირჩიეთ ერთი:

- a.
პროცესთაშორის წყვეტას
- b.
ტაიმერიდან წყვეტას
- c.
სინქრონულ წყვეტას
- d.
ასინქრონულ წყვეტას

-rwxr-x-w- დაშვების უფლებების ...

**-rwxr-x-w- დაშვების უფლებების
სიმბოლური ჩანაწერის შესაბამის
რვაობით ჩანაწერს ექნება სახე :**

აირჩიეთ ერთი:

- a.
0732
- b.
0741
- c.
0662
- d.
0653
- e.
0752

სწორი პასუხია:

0752

ჩამოთვლილთაგან რომელი წინადადებაა ჭეშმარიტი FIFO

ჩამოთვლილთაგან რომელი
წინადადებაა ჭეშმარიტი FIFO კავშირის
არხთან მიმართებით.

აირჩიეთ ერთი:

a.

FIFO კავშირის არხის გამოყენება
შეუძლია სისტემაში წარმოქმნილ
ყველა პროცესს მიუხედავად იმისა
როდის ან რომელი პროცესის მიერ
მოხდა FIFO კავშირის არხის
წარმოქმნა

b.

FIFO კავშირის არხის გამოყენება
შეუძლია სისტემაში წარმოქმნილ
ყველა იმ პროცესს, რომელთა
მეშვეობითაც წარმოიქმნა FIFO
კავშირის არხის წარმომქმნელი
პროცესი

c.

FIFO კავშირის არხის გამოყენება
შეუძლია სისტემაში წარმოქმნილ
ყველა იმ პროცესს, რომელსაც ყავს
FIFO კავშირის არხის წარმომქმნელი
საერთო წინაპარი

d.

ჩამოთვლილთაგან არცერთი პასუხი
არაა სწორი

ნიმუში:

FIFO კავშირის არხის გამოყენება
შეუძლია სისტემაში წარმოქმნილ ყველა
პროცესს მიუხედავად იმისა როდის ან
რომელი პროცესის მიერ მოხდა FIFO
კავშირის არხის წარმოქმნა

**კავშირის არხის დასრულების
პროცესირება რა შემთხვევაში არაა
აუცილებელი**

აირჩიეთ ერთი:

- a.
ჩამოთვლილთაგან ყველა პასუხი
სწორია (იგულისხმება უარყოფითი
პასუხის გარდა ყველა)
- b.
ჩამოთვლილთაგან არცერთი პასუხი
არაა სწორი
- c.
თუ პროცესებს შორის კავშირის არხის
გახსნა არ იყო პროცესირებული
სპეციალური მოქმედებებით
- d.
თუ პროცესებს შორის კავშირის არხის
გახსნა პროცესირებული იყო
სპეციალური მოქმედებებით
- e.
პროცესებს შორის კომუნიკაციის
დასრულების შემდეგ

სიონი პასუხია:

თუ პროცესებს შორის კავშირის არხის
გახსნა არ იყო პროცესირებული
სპეციალური მოქმედებებით

ჩამოთვლილთაგან რომელ შემთხვევაში შეიძლება აღმოჩნდეს პროცესი მზადყოფნის მდგომარეობაში

ჩამოთვლილთაგან რომელ შემთხვევაში
შეიძლება აღმოჩნდეს პროცესი
მზადყოფნის მდგომარეობაში

აირჩიეთ ერთი:

- a.
დროითი კვანტის ამონურვის გამო
პროცესს ჩამოერთვა პროცესორი
- b.
ჩამოთვლილთაგან არცერთი პასუხი
არაა სწორი
- c.
ჩამოთვლილთაგან ყველა პასუხი
სწორია (იგულისხმება უარყოფითი
პასუხის გარდა ყველა)
- d.
ბლოკირების მდგომარეობაში მყოფ
პროცესს გამოეყო მისთვის საჭირო
ყველა რესურსი
- e.
თუ პროცესს დაბადების (სისტემაში
გამოჩენის) შემდეგ მოთხოვნის
შესაბამისად გამოეყო პროცესორის
გარდა შესასრულებლად საჭირო
ყველა რესურსი



სწორი პასუხია:

ჩამოთვლილთაგან ყველა პასუხი სწორია
(იგულისხმება უარყოფითი პასუხის
გარდა ყველა)

**0532 დაშვების უფლებების რვაობითი
ჩანაწერის შესაბამის სიმბოლურ
ჩანაწერს ექნება სახე:**

აირჩიეთ ერთი:



a.

-r-X-WX-W-



b.

-r-XrW--WX



c.

-r-Xr---W-



d.

--WXr--rW-



e.

-rW-r-X-WX

სწორი პასუხია:

-r-X-WX-W-

ჩამოთვლილთაგან რომელი წარმოადგენს მონოლიტური არქიტექტურის

ჩამოთვლილთაგან რომელი წარმოადგენს მონოლიტური არქიტექტურის წაკლოვანებას

აირჩიეთ ერთი:

- a.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- b.
ყველა პროცედურა ერთმანეთთან ურთიერთქმედებს მთავარი პროცედურის გამოყენებით
- c.
პროცედურებს არ შეუძლიათ ზეგავლენა იქონიონ სხვა პროცედურების საქმიანობაზე
- d.
პროცედურათა წაკრების სახით შემუშავებული ოპერაციული სისტემის კოდი წელა მუშაობს

ჩამოთვლილთაგან რომელი მიეკუთვნება აპარატურული წყვეტის ..

ჩამოთვლილთაგან რომელი მიეკუთვნება აპარატურული წყვეტის მნიშვნელოვან ტიპს

აირჩიეთ ერთი:

- a.
პროცესორთაშორისი წყვეტა
- b.
შეტანა/გამოტანა წყვეტა
- c.
ტაიმერიდან წყვეტა
- d.
კლავიატურიდან წყვეტა

[Clear my choice](#)

ჩამოთვლილთაგან გამოთვლითი მანქანის რომელი ოპერაციული სისტემა გამოიყენება ტექნიკური და ტექნოლოგიური პროცესების სამართავად

ჩამოთვლილთაგან გამოთვლითი მანქანის რომელი ოპერაციული სისტემა გამოიყენება ტექნიკური და ტექნოლოგიური პროცესების სამართავად.

აირჩიეთ ერთი:



a. **რეალური დროის**



b. **მიკრო ბირთვული**



c. **მეინფრეიმის**



d. **კლიენტ-სერვერული**

[Clear my choice](#)

ჩამოთვალეთ და აღწერეთ პროცესებს შორის მონაცემების გაცვლის საშუალებები

ჩამოთვალეთ და აღწერეთ პროცესებს შორის მონაცემების გაცვლის საშუალებები.

A standard Microsoft Word-style toolbar with icons for bold, italic, underline, list styles, and various symbols.

1) სიგნალური - ის გადასცემს ერთ ბაიტს, ის პროცესის ინფორმირებას აზდენს როდესაც რაღაც კონკრეტული მოვლენა ხდება და ასევე რა არის საჭირო ამ მოვლენაზე რეაგირებისთვის. ამ დროს მთავარია პროცესის ჰქონდეს ამ სიგნალის ტიპის ინფორმაციის მიღების და რეაგირების საშუალება

2) არხული - მონაცემების გავლა ხდება სპეციალური კავშირის არხებით, მონაცემების ზომა დამოკიდებულია ამ არხების გამტარობაზე და ასევე მონაცემების გაზრდით სწვა პროცესებზე ზემოქმედების საშუალებაც იზრდება

3) განაწილებადი მეხსიერება - რამდენიმე პროცესს შეუძლია გამოიყენოს მეხსიერების მისამართის გარკვეული ნაწილი, მონაცემების ზომა კი ამ მისამართის ზომაზეა დამოკიდებული. ასევე, თუკი დიდ მონაცემებს გადავცემთ ამ გზით, საფრთხილოა, რადგან შეიძლება ამ მისამართის დაკავებით სწვა პროცესების მეხსიერება დავიკავოთ, რამაც შესაძლოა ეს სწვა პროცესები გათიშოს

ჩამოთვლილთაგან რომელი შეესაბამება სიმპლექსური კავშირის არხის განმარტებას

ჩამოთვლილთაგან, რომელი შეესაბამება სიმპლექსური კავშირის არხის განმარტებას.

აირჩიეთ ერთი:

a.

კავშირის არხს, რომლის მეშვეობითაც შესაძლებელია ორივე მიმართულებით მონაცემების გადაცემა მაგრამ რიგითობის დაცვით

b.

კავშირის არხს, რომლის მეშვეობითაც შესაძლებელია როგორც მემკვიდრე ისე არამემკვიდრე პროცესებს შორის მონაცემების გაცვლა

c.

კავშირის არხს, რომლის მეშვეობითაც შესაძლებელია ორივე მიმართულებით მონაცემების გადაცემა ერთდროულად

d.

კავშირის არხს, რომლის მეშვეობითაც შესაძლებელია მემკვიდრე პროცესებს შორის მონაცემების გაცვლა

e.

ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი

[Clear my choice](#)

როგორც ცნობილია მეცნიერებმა დაადგინეს ურთიერთბლოკირების წარმოქმნის აუცილებელი და საკმარისი პირობები.

როგორც ცნობილია მეცნიერებმა დაადგინეს ურთიერთბლოკირების წარმოქმნის აუცილებელი და საკმარისი პირობები. ჩამოთვლილთაგან რომელი შეესაბამება პირობას "რესურსის ღოდინი"?

აირჩიეთ ერთი:

a.

უნდა არსებობდეს ორი ან მეტი პროცესისგან შემდგარი წრიული მიმდევრობა, რომელშიც მიმდევრობის ყოველი წევრი ელოდება მის შემდეგ მდგომი წევრის მიერ დაკავებული რესურსის გამონთავისუფლებას

b.

დროის ნებისმიერ მომენტში რესურსი ან გამოყოფილია მხოლოდ ერთი პროცესისთვის ან თავისუფალია

c.

პროცესს, რომელსაც დაკავებული აქვს გარკვეული რესურსი შეუძლია მოითხოვოს ახალი რესურსი

d.

ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი

e.

პროცესისათვის მის მიერ დაკავებული რესურსის იმულებითი ჩამორთმება შეუძლებელია. პროცესმა დაკავებული რესურსი უნდა გამოათავისუფლოს საკუთარი სურვილით

[Clear my choice](#)

თუ ფაილის მონაცემების განთავსება შესანახ მოწყობილობაზე ხდება იმ მიმდევრობით, რომელიც ოპერაციული სისტემისთვის ან პროგრამისთვის არის ხელსაყრელი, მაშინ ფაილის თრგანიზაციის ასეთ მეთოდს ეწოდება?

..

თუ ფაილის მონაცემების განთავსება შესანახ მოწყობილობაზე ხდება იმ მიმდევრობით, რომელიც ოპერაციული სისტემისთვის ან პროგრამისთვის არის ხელსაყრელი, მაშინ ფაილის თრგანიზაციის ასეთ მეთოდს ეწოდება?

აირჩიეთ ერთი:

- a.
პირდაპირი
- b.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- c.
მოდულური
- d.
ინდექსირებული მიმევრობა
- e.
მიმდევრობითი

ახსენით რას ნიშნავს კავშირის არხი არის **დუპლექსური**

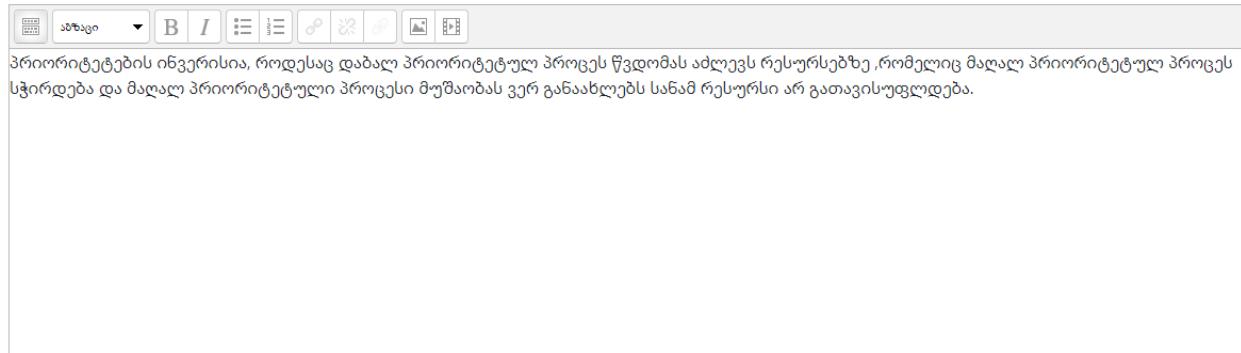
ახსენით რას ნიშნავს კავშირის არხი არის **დუპლექსური**.

კავშირის არხი არის დუპლექსური, როცა მასთან ასოცირებულ პროცესს აქვს ორმიმართულებიანი კავშირის შესაძლებლობა ერთდროულად. ანუ ინფორმაციის მსიაღებად და მის გადასაცემად ერთდროულად შეუძლია მოქმედება კავშირს.

დუპლექსური რეჟიმის დროს მონაცემთა გადაცემა და მიღება სრულდება ერთდროულად. ყველაზე მარტივ შემთხვევაში დუპლექსური რეჟიმისათვის გამოიყენება კავშირის ორი ხაზი. ერთი ხაზი გამოიყენება მონაცემთა გადაცემისათვის, ხოლო მეორე – მონაცემთა მიღებისათვის. თუმცა არსებობს სხვადასხვაგვარი გადაწყვეტები, რომელთა მეშვეობითაც შესაძლებელია დუპლექსური რეჟიმის განხორციელება ერთი კავშირის არხის გამოყენებით. მაგალითად, ორივე კვანძს შეუძლია მონაცემთა გადაცემა ერთდროულად, ხოლო მიღებული მონაცემებიდან თთოეული კვანძი გამოარჩევს მხოლოდ მისთვის განკუთვნილ მონაცემებს.

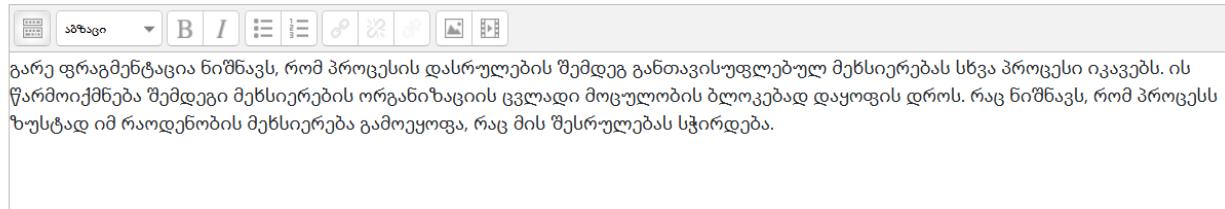
რა ეწოდება პროცესის პრიორიტეტების ინვერსია

რას ეწოდება პროცესის პრიორიტეტების ინვერსია.



მოიყვანეთ გარე ფრაგმენტაციის განმარტება

(2 ქულა) მოიყვანეთ გარე ფრაგმენტაციის განმარტება



რაში მდგომარეობს წყვეტის აკრძალვის ალგორითმის დადებითი ...

(2 ქულა) რაში მდგომარეობს წყვეტის აკრძალვის ალგორითმის დადებითი და უარყოფითი მხარეები (მოკლედ.)

წყვეტის აკრძალვის შემთხვევაში შეუძლებელი იქნება ტაიმერიდან წყვეტის განხორციელება და შესაბამისად პროცესორიც ვერ შეძლებს სწვა პროცესზე გადართვას. ამ შემთხვევაში არ არსებობს საფრთხე რომელიმე პროცესმა განახორციელოს საკუთარ კრიტიკულ სექციაში შესვლა.

მრავალპროცესორული სისტემის შემთხვევაში წყვეტის აკრძალვა მოქმედებს მხოლოდ იმ პროცესორზე, რომელსაც შეეხება ბლოკირების ინსტრუქცია. დანარჩენი პროცესორები ჩვეულ რეჟიმში გააგრძელებენ მუშაობას.

შემდეგი ინსტრუქციის წარმატებით დასრულების შემთხვევაში ...

შემდეგი ინსტრუქციის წარმატებით დასრულების შემთხვევაში

```
int *memory = (int *) shmat(shared_memory, NULL, 0);
```

အဂ်ဂိုလ်

- a.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
 - b.
შეიქმნება განაწილებადი მეხსიერება
 - c.
დაგნერირდება უნიკალური გასაღები განაწილებადი მეხსიერების შესაქმნელად
 - d.
განაწილებადი მეხსიერება მიუერთდება პროცესის მისამართების სივრცეს

[Clear my choice](#)

ვთქვათ სისტემაში გვაქვს პროგრამა, რომლის კომპილაციის შედეგად კომპილიატორი აგენტერირებს გადატანად კოდს. ასეთ შემთხვევაში ჩამოთვლილთაგან რომელ ...

კოქით სიხტემაში გვაქვს პროგრამა, რომელის კომპიუტერის შედეგად კომპიუტერული აგენტერიგნს გადატანა კოდს. ასეთ შემთხვევაში ჩამოთვლილთაგან რომელ ტექნიკაზე იქნა შესაძლებელი ფიზიკური და ლოგიკური მექანიზმების მისამრთების დაკვირვება?

აირჩივთ ერთი:

- a. შესრულების ეტაპი
 - b. კომპილაციის ეტაპი
 - c. ჩატვირთვის ეტაპი
 - d. განვითარების ეტაპი

[Clear my choice](#)

აღწერეთ დაგეგმვის SJF ალგორითმის მუშაობის პრინციპი, მისი დადებითი და უარყოფითი ...

აღწერეთ და გეგმვის SJF ალგორითმის მუშაობის პრინციპი, მისი დადგენითი და უარყოფითი მხარეები. მოიყვანეთ მაგალითი.

sjf ishifreba rogorc shortest job first an shortes job next . es aris algoritmi romelic cxriluri saxit naxulobs romeli samushao shesruldeba yvelaze swrafad rom shesrulos shemdegi,

1. rigshi SJFs schirdeba yvelaze patara dro sxva "cxrilur" algoritmetban shedarebit
2. is aris greedy algoritmi
3. man sheidzleba gamoiwwios starvation tu mokle procesebi gagrdzeldeba . es sheidzleba gadawydes daberebis kocnepciis gamoyenebit sjf sheidzleba gamoyenbul iqnas specialur garemoshi sadac gashvebuli drois zusti shefasebebia xelmisawvdomi

3თქვათ DIR8 არის სამუშაო დირექტორია.

3თქვათ dir8 არის სამუშაო დირექტორია. file4 ფაილის dir4 დირექტორიაში file5 სახელით გადაკოპირებისთვის (copy-paste) საჭიროა ტერმინალში ბრძანება შესრულდეს შემდეგი სახით აირჩიეთ ერთი:

- a. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- b. `cp dir5/dir7/file4 ../../dir4/file5`
- c. `mv dir5/dir7/file4 ../../dir4/file5`
- d. `mv ../../dir7/file4 ../../dir4/file5`
- e. `cp ../../dir7/file4 ../../dir4/file5`

[Clear my choice](#)

როგორ პროცესს ეწოდება დემონი

როგორ პროცესს ეწოდება დემონი.

აირჩიეთ ერთი:

- a. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- b. პროცესს, თუ ის წარმოიქნა სისტემის ჩატვირთვასთან ერთად
- c. პროცესს, თუ ის მუშაობს ფონურ რეჟიმში
- d. პროცესს, თუ ის დასრულების შემდეგ გარკვეული დროით დარჩა სისტემაში

[Clear my choice](#)

74		P1															
75		P2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
76		P3		-	-	-	-	-	-	-	-	-	-	-	-	-	-
77		P4							+	-	-	-	-	-	-	-	-
78		P5							+	+	+	+	+	+	-	-	-
79																	
80																	
81																	
82		RR - quantum 3															
83		პროცესი	P1	P2	P3	P4	P5										
84		შესრულების დრო	5	7	3	6	4										
85																	
86			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
87			P1	+	+	+	-	-	-	-	-	-	-	-	-	-	+
88			P2	-	-	-	+	+	+	-	-	-	-	-	-	-	-
89			P3	-	-	-	-	-	-	+	+	+	-	-	-	-	-
90																	

საშუალო
მნიშვნელობა

19.60

ლოდინის

14.6

მოიყვანეთ შეჯიბრის მდგომარეობის განმარტება

მოიყვანეთ შეჯიბრის მდგომარეობის განმარტება.



შეჯიბრის მდგომარეობა ეწოდება სიტუაციას როდესაც ორი ან რამოდენიმე პროცესი იყენებს საერთო რესურსს და მათი შედეგი დამოკიდებულია ბოლოს შესრულებულლ პროცესზე.

რას ეწოდება წყვეტა ?

ჩამოთვლილთაგან რომელი შექსაბამება ფაილური დესკრიპტორის განმარტებას ?

Home მართვის პანელი Events My Courses This course

> ჩემი კურსები > რენტაციული სისტემა (2020) > შეაღებური გამოცდა > შეაღებური გამოცდა - ჯუფი 7

ჩამოთვლილთაგან რომელი შექსაბამება ფაილური დესკრიპტორის განმარტებას.

აღნიშვნელი ერთი:

a. რიცხვითი მნიშვნელობა, რომელიც გამოსახავს ფაილში ჩაწერილი ბაიტების რაოდენობას

b. ჩამოთვლილთაგან არცერთი პასუხი არა სწორი

c. რიცხვითი მნიშვნელობა, რომელიც გამოსახავს პროცესის მიერ შესრულების (ამუშავების დროიდან მოყოლებული) მთელს პერიოდში გახსნილი ფაილების რაოდენობას

d. რიცხვითი მნიშვნელობა, რომელიც გამოსახავს ფაილიდან წაკითხული ბაიტების რაოდენობას

e. რიცხვითი მნიშვნელობა, რომელიც გამოსახავს პროცესის მიერ მიმღინტში გახსნილი ფაილების რაოდენობას

[Clear my choice](#)

კუდი - 8.11.2019.rar

ჩამოთვლილთაგან რომელი შეესაბამება სიმპლექსური კავშირის

ჩამოთვლილთაგან, რომელი შეესაბამება სიმპლექსური კავშირის არხის განმარტებას.

აირჩიეთ ერთი:

- a.
კავშირის არხს, რომლის მეშვეობითაც შესაძლებელია მემკვიდრე პროცესებს შორის მოწაცემების გაცვლა
- b.
კავშირის არხს, რომლის მეშვეობითაც შესაძლებელია როგორც მემკვიდრე ისე არამეტყვიდრე პროცესებს შორის მოწაცემების გაცვლა
- c.
კავშირის არხს, რომლის მეშვეობითაც შესაძლებელია ორივე მიმართულებით მოწაცემების გადაცემა მაგრამ რიგითობის დაცვით
- d.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- e.
კავშირის არხს, რომლის მეშვეობითაც შესაძლებელია ორივე მიმართულებით მოწაცემების გადაცემა ერთდროულად

3თქვათ ერთპროცესორულ სისტემაში პროცესი იმყოფება

ულუდური გამოცდა - ჯგუფი 7

3თქვათ ერთპროცესორულ სისტემაში პროცესი იმყოფება შესრულების მდგომარეობაში. ჩამოთვლილთაგან რომელი წინადადებაა ჭეშმარიტი.

აირჩიეთ ერთი:

- a.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- b.
შესრულების მდგომარეობაში მყოფი პროცესის საქმიანობის ბლოკირება შეუძლია მიმდინარე (იმავე) პროცესს
- c.
შესრულების მდგომარეობაში მყოფი პროცესის საქმიანობის ბლოკირება შეუძლია მიმდინარე (იმავე) მდგომარეობაში მყოფ პროცესს
- d.
შესრულების მდგომარეობაში მყოფი პროცესის საქმიანობის ბლოკირება შეუძლია მზადყოფნის მდგომარეობაში მყოფ პროცესს

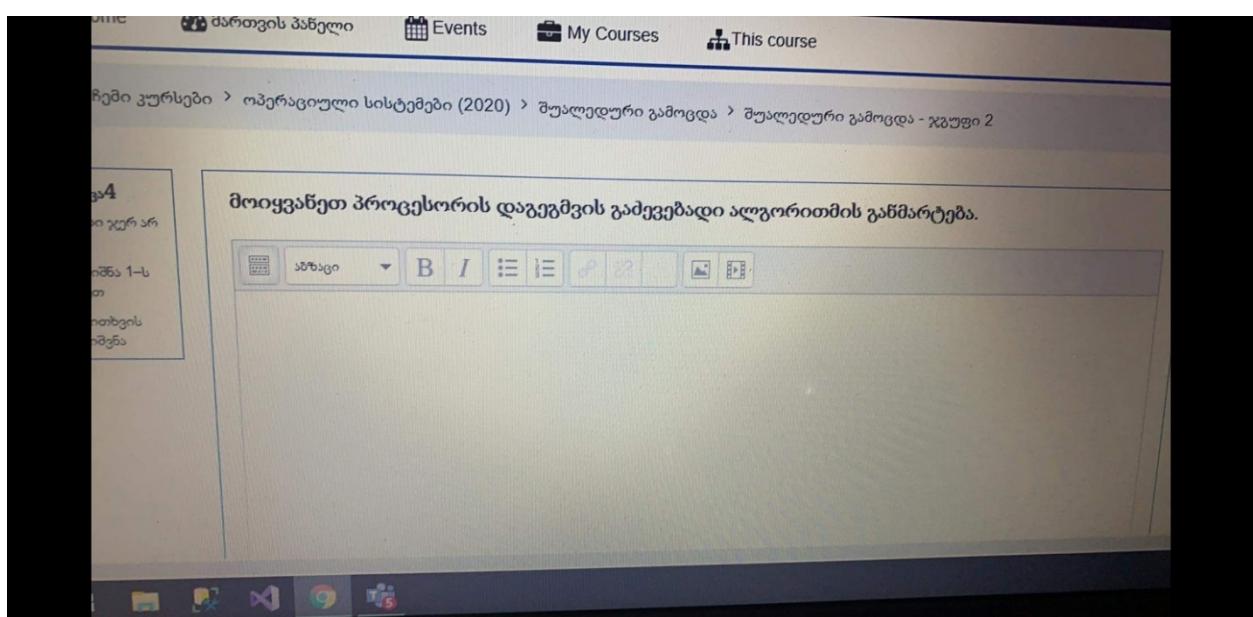
წარმატებულად დასრულების შემთხვევაში რას აბრუნებს

წარმატებულად დასრულების შემთხვევაში რას აბრუნებს open სისტემური გამოძახება.

აირჩიეთ ერთი:

- a.
ფაილურ დესკრიპტორს
- b.
ფაილის გამხსნელი პროცესის იდენტიფიკატორის მნიშვნელობას
- c.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- d.
ფაილში ჩაწერილი ბაიტების რაოდენობას

მოიყვანეთ პროცესორის დაგეგმვის გამევებადი ალგორითმის განმარტება



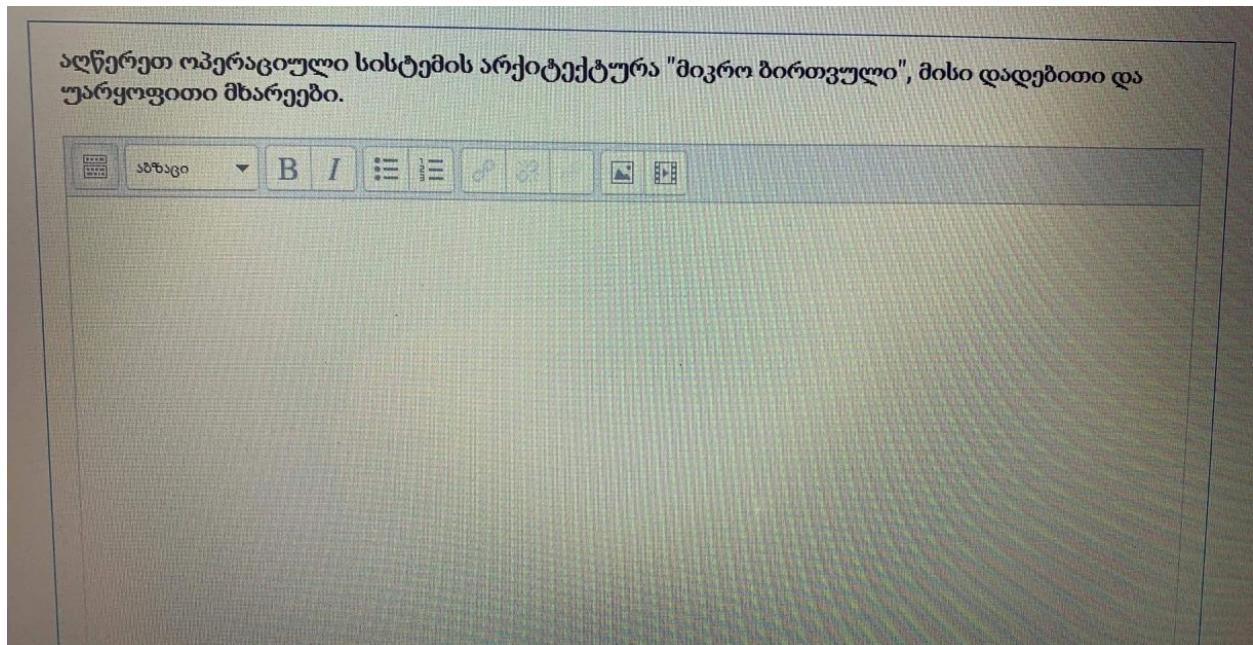
ჩამოთვლილთაგან რომელი შეესაბამება ასინქრონული წყვეტის განმარტებას

ჩამოთვლილთაგან რომელი შეესაბამება ასინქრონული წყვეტის განმარტებას.

აირჩიეთ ერთი:

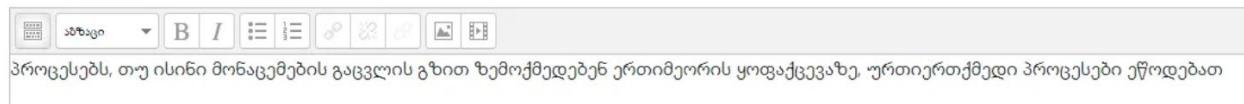
- a.
ეს არის წყვეტა, რომელიც წარმოიშვა კოდში დაშვებული შეცდომისგან დამოუკიდებლად
- b.
ეს არის წყვეტა, რომელიც წარმოიშვა კლავიატურაზე დაწვაპუნების შედეგად
- c.
ეს არის წყვეტა, რომელიც წარმოიშვა კოდში დაშვებული შეცდომის გამო.
- d.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი

აღწერეთ ოპერაციული სისტემის არქიტექტურა „მიკრო ბირთვული“



როგორ პროცესებს ეწოდებათ ურთიერთქმედი პროცესები?

როგორ პროცესებს ეწოდებათ ურთიერთქმედი პროცესები?



ვთქვათ DIR2 არის სამუშაო დირექტორია



ვთქვათ dir2 არის სამუშაო დირექტორია. dir4 დირექტორის dir7 დირექტორიაში გადაკოპირებისთვის (copy-paste) საჭიროა ტერმინალში ბრძანება შესრულდეს შემდეგი სახით

აირჩიეთ ერთი:

- a.
`mv -a dir4 dir5/dir7`
- b.
`mv dir4 dir5/dir7`
- c.
`cp -r dir4 dir5/dir7`
- d.
`cp dir4 dir5/dir7`
- e.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი

3თქვათ ერთპროცესორულ სისტემაში პროცესი იმყოფება შესრულების

3თქვათ ერთპროცესორულ სისტემაში პროცესი იმყოფება შესრულების მდგომარეობაში. ჩამოთვლილთაგან რომელი წინადაღება ჰქონდა.

აირჩიეთ ერთი:

- a. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- b. შესრულების მდგომარეობაში მყოფი პროცესის საქმიანობის ბლოკირება შეუძლია ბლოკირების მდგომარეობაში მყოფ პროცესს
- c. შესრულების მდგომარეობაში მყოფი პროცესის საქმიანობის ბლოკირება შეუძლია მზადყოფნის მდგომარეობაში მყოფ პროცესს
- d. შესრულების მდგომარეობაში მყოფი პროცესის საქმიანობის ბლოკირება შეუძლია მიმდინარე (იმავე) პროცესს
- e. ყველა ჩამოთვლილი შემთხვევა ჰქონდა (იგულისხმება უარყოფის გარდა ყველა შემთხვევა)

3თქვათ სისტემაში პროცესები წარმოდგენილია ქვემოთ მოყვანილი ცხრილის მეშვეობით

The screenshot shows a computer interface with a sidebar on the left containing navigation links like 'my courses', 'This course', and 'Hide blocks'. The main area displays text and a table related to process scheduling.

Text in the main area:

3თქვათ, სისტემაში პროცესები წარმოდგენილია ქვემოთ მოყვანილი ცხრილის მეშვეობით.

RR ალგორითმის გამოყენებით იპოვეთ P_4 , P_6 და P_{10} პროცესების შესრულების საერთო დროის საშუალო არითმეტიკული, თუ დროითი კვანტის მიზნებისა არის 2.

Table:

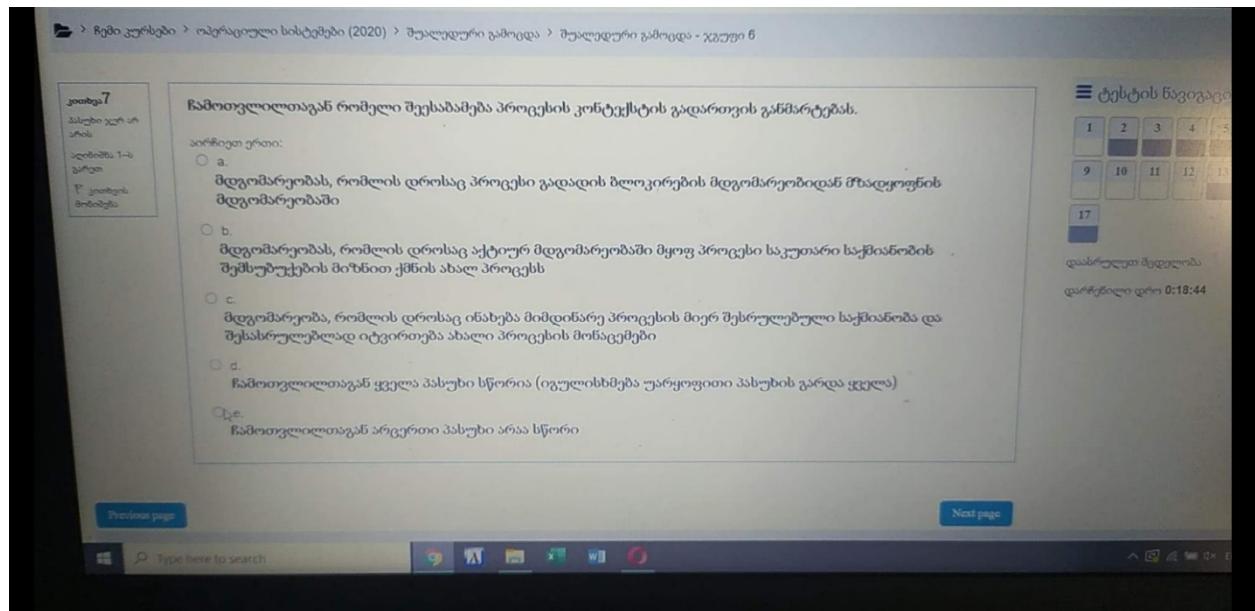
პროცესები	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
შესრულების დრო	1	7	3	2	4	3	2	1	6	3

შემთხვევა. შესაბამისი გამოთვლების ჩატარება შესაძლებელია აქვთ.

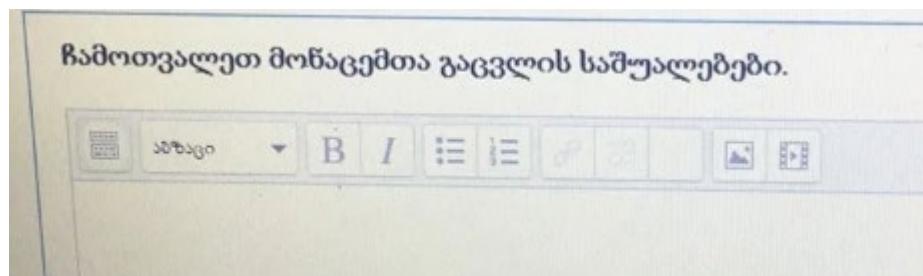
Text below the table:

საერთო დროის საშუალო არის
 $(8+25+28)/3 = 61/3 = 20.3$

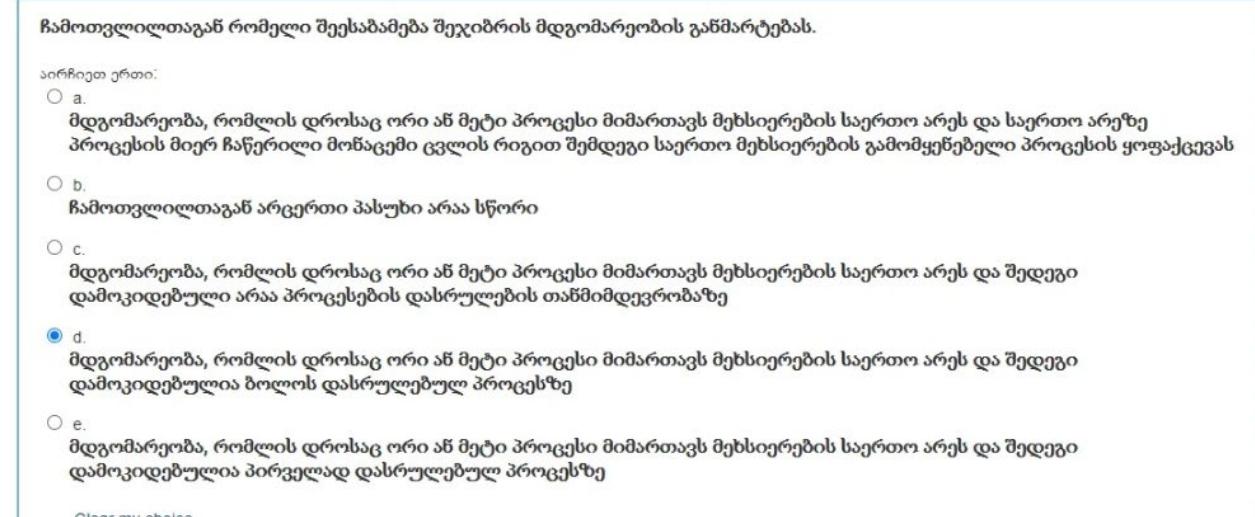
ჩამოთვლილთაგან რომელი შეესაბამება პროცესის კონტექსტის გადართვის განმარტებას



ჩამოთვალეთ მონაცემთა გაცვლის საშუალებები



ჩამოთვლილთაგან რომელი შეესაბამება შეჯიბრის მდგომარეობის განმარტებას



დაგეგმვის RR ალგორითმის გამოყენებით დაითვალიერებული პროცესების შესრულების

დაგეგმვის RR ალგორითმის გამოყენებით დაითვალიერებული პროცესების შესრულების და ლოდინის საერთო დროების საშუალო არითმეტიკული, თუ პროცესები მოცემულია შემდეგი ცხრილით და დროითი კვანტი არის 5.

პროცესები	P_1	P_2	P_3	P_4	P_5	P_6
შესრულების დრო	7	10	5	8	13	9
გამოჩენის დრო	28	18	2	0	13	7

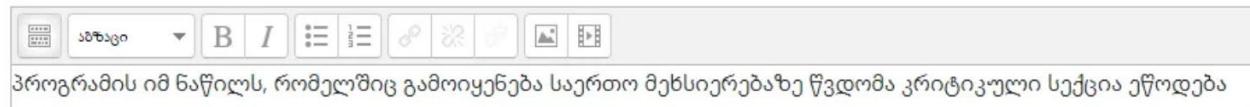
შენიშვნა. არ იზღუდება ამოცანის გაკეთება excel-ის ან word-ის ფაილში.

შესაბამისი ფაილი აუცილებლად უნდა ატვირთოთ.



მოიყვანეთ კრიტიკული სექციის განმარტება

მოიყვანეთ კრიტიკული სექციის განმარტება.



დაშვების უფლებების სიმბოლური ჩანაწერის

-rw-r-x-w- დაშვების უფლებების სიმბოლური ჩანაწერის შესაბამის რვაობით ჩანაწერს ექნება სახე:

აირჩიეთ ერთი:

a.
0642

b.
0752

c.
0652

d.
0641

e.
0732

შეჯიბრის მდგომარეობის აცილების მიზნით კრიტიკულ სექციაზე აღგორითმის წყვეტის აკრძალვა

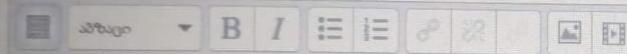
შეჯიბრის მდგომარეობის აცილების მიზნით კრიტიკულ სექციაზე აღგორითმის "წყვეტის აკრძალვა" გამოყენების შემთხვევაში ჩამოთვლილთაგან რომელი წინადადებაა ჭეშმარიტი

აირჩიეთ ერთი:

- a. პროცესს შეუძლია იმყოფებოდეს საკუთარ კრიტიკულ სექციაში სამუშაოს დაასრულებამდე
- b. პროცესს შეუძლია იმყოფებოდეს საკუთარ კრიტიკულ სექციაში სანამ სხვა პროცესი არ დააპირებს საკუთარ კრიტიკულ სექციაში შესვლას
- c. ერთზე მეტ პროცესს ერთდროულად შეუძლია იმყოფებოდეს საკუთარ კრიტიკულ სექციაში
- d. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი

რას ეწოდება ფაილური დესკრიპტორი

რას ეწოდება ფაილური დესკრიპტორი?



მასივის ელემენტის ინდექსს, რომელიც შეესაბამება შეტანა/გამოტანის გარკვეულ ნაკადს, თმ ნაკადისთვის ფაილური დესკრიპტორი ეწოდება

ჩამოთვლილთაგან რომელი სისტემური გამოძახებით იქნება

ჩამოთვლილთაგან რომელი სისტემური გამოძახებით იქმნება კავშირის არხი FIFO.

აირჩიეთ ერთი:

- a. **makefifo**
- b. **fifofile**
- c. **createfifo**
- d. **openfifo**
- e. **mknod**

თუ DIR3 არის სამუშაო დირექტორია

The screenshot shows a terminal window with the following content:

```
my Courses This course
file4
file5
file12

თუ dir3 არის სამუშაო დირექტორია, მაშინ შემდეგი ბრძანების შესრულების
mkmdir dir6/dir1 dir6/dir1/dir2

აირჩიეთ ერთი:
a. ფაილური სისტემიდან წაიშლება dir1 და dir2 დირექტორიები
b. dir6 დირექტორიაში შეიქმნება dir1 და dir2 დირექტორიები
c. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
d. dir6 დირექტორიაში გადაადგილებული იქნება dir1 და dir2 დირექტორიები
e. dir6-ში შეიქმნება dir1 ქვედირექტორია, რომელიც თავისმხრივ შეიქმნება dir2
```

ქვემოთ ჩამოთვლილთაგან რომელ მონაცემს არ შეიცავს სუპერბლოკს

ქვემოთ ჩამოთვლილთაგან რომელ მონაცემს არ შეიცავს სუპერბლოკი?

აირჩიეთ ერთი:

- a. ფაილურ სისტემაში ბლოკების რაოდენობაზე ინფორმაციას
- b. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- c. თავისუფალი ბლოკების ადგილმდებარეობაზე ინფორმაციას
- d. დაკავებულ (მონაცემების შემცველ) ბლოკებზე ინფორმაციას
- e. საბაზო დირექტორიის ადგილმდებარეობაზე ინფორმაციას

თუ პროცესორის დაგეგმვა ხდება სტატიკული პრიორიტეტის გამოყენებით, მაშინ ოპერაციულ სისტემაში მზადებელის მდგომარეობაში მყოფი პროცესების დალაგება მოხდება

აირჩიეთ ერთი:

- a.
პრიორიტეტის ზრდადი მნიშვნელობების მიხედვით
- b.
სისტემაში მათი გამოჩენის დროის მიხედვით
- c.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- d.
მათი შესრულებისათვის საჭირო დროის მიხედვით

[Clear my choice](#)

თუ მეხსიერება დაყოფილია ცვლადი მოცულობის ბლოკებად

თუ მეხსიერება დაყოფილია ცვლადი მოცულობის ბლოკებად, მაშინ ასეთ დანაწილებას ეწოდება?

აირჩიეთ ერთი:

- a.
სეგმენტური
- b.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- c.
ბადისებრი
- d.
ფურცლისებრი
- e.
სეგმენტურ-ფურცლისებრი
- f.
ფურცლისებრ-სეგმენტური

თუ ფაილის მონაცემების განთავსება შესანახ მოწყობილობაზე ხდება იმ მიმდევრობით, რომელიც თქმება

თუ ფაილის მონაცემების განთავსება შესანახ მოწყობილობაზე ხდება იმ მიმდევრობით, რომელიც თქმება

აირჩიეთ ერთი:

- a.
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- b.
მიმდევრობითი
- c.
მოდულური
- d.
პირდაპირი
- e.
ინდექსირებული მიმდევრობა

[Clear my choice](#)

3თქვათ DIR2 არის სამუშაო დირექტორია.

The screenshot shows a Linux terminal window titled "TFO - Named pipes: mififo, m". The terminal displays a file structure with files file3, file6, file11, file12, file4, and file5. A question is displayed in the terminal:

3თქვათ dir2 არის სამუშაო დირექტორია. dir4 დირექტორის dir7 დირექტორიაში გადაღვილებისას (cut-paste) საჭიროა ტერმინალში ბრძანება შესრულდეს შემდეგი სახით

აირჩიეთ ერთი:

- a. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- b. mv dir4 dir5/dir7
- c. cp -a dir4 dir5/dir7
- d. cp -r dir4 dir5/dir7
- e. mv -a dir4 dir5/dir7

[Clear my choice](#)

რას გულისხმობს პროცესის საქმიანობის დაგეგმვის პოლიტიკა „ სამართლიანობა „

რას გულისხმობს პროცესორის საქმიანობის დაგეგმვის პოლიტიკა „ სამართლიანობა “

აირჩიეთ ერთი:

- a. პროცესების რაოდენობის ზრდამ არ უნდა შეარყიოს სისტემის ქმედურარიანობა
- b. თანაბარი დატვირთულობის პირობებში პროცესი სხვადასხვა დროს ამუშავებისას შესასრულებლად არ უნდა საჭიროა დროს
- c. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- d. პრიორიტეტის გამოყენებით უნდა ხდებოდეს პროცესორის გამოყენება

[Clear my choice](#)

თუ პროცესორის დაგეგმვა ხდება სტატიკური პრიორიტეტის გამოყენებით

თუ პროცესორის დაგეგმვა ხდება სტატიკური პრიორიტეტის გამოყენებით, მაშინ ოპერაციულ სისტემაში მზადყოფნის მდგრამარტობაში მყოფი პროცესების დალაგება მოხდება

აირჩიეთ ერთი:

- a. სისტემაში მათი გამოჩენის დროის მიხედვით
- b. მათი შესრულებისათვის საჭირო დროის მიხედვით
- c. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- d. პრიორიტეტების ზრდადი მნიშვნელობების მიხედვით

[Clear my choice](#)

პროგრამული კოდის შემუშავებისას პროგრამისტების ჯგუფს დასჭირდა განაწილებადი

პროგრამული კოდის შემუშავებისას პროგრამისტების ჯგუფს დასჭირდა რამდენიმე განაწილებადი მეხსეირების გამოყენება. განაწილებადი მეხსიერებისთვის უნიკალური გასაღების მნიშვნელობის მისაღებად პროგრამისტებმა გამოიყენეს სისტემაში არსებული ფაილი. ჩამოთვლილთაგან რომელი წინადაღება უზრუნველყოფს განაწილებადი მეხსიერებების წარმატებულ შექმნას?

აირჩიეთ ერთი:

- a. თუ ერთი უნიკალური გასაღებით მოხდება სხვადასხვა ზომის განაწილებადი მეხსიერების შექმნა
- b. თუ უნიკალური გასაღებების მნიშვნელობები მიღებული იყო ფაილისა და ასლის ერთიდაიმავე წყვილისთვის
- c. თუ უნიკალური გასაღებების მნიშვნელობები მიღებული იყო ფაილისა და ასლის განსხვავებული წყვილისთვის
- d. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი

[Clear my choice](#)

რა ეწოდება პროგრამულ უზრუნველყოფას, რომელიც ოპერაციულ სისტემაში ამა თუ იმ დონეზე ხელს უწყობს

რა ეწოდება პროგრამულ უზრუნველყოფას, რომელიც ოპერაციულ სისტემაში ამა თუ იმ დონეზე ხელს უწყობს ფიზიკური ან აბსტრაქტული კომპონენტის ნორმალურ ფუნქციონირებას

აირჩიეთ ერთი:

- a. ასემბლერი
- b. ბირთვი
- c. სისტემური პროგრამა
- d. დრაივერი

ქვემოთ ჩამოთვლილთაგან რომელი სისტემური გამოძახება

ქვემოთ ჩამოთვლილთაგან რომელი სისტემური გამოძახება აბრუნებს ფაილურ დესკრიპტორს?

აირჩიეთ ერთი:

- a. `FDCreate()`
- b. `open()`
- c. `FileDescriptor()`
- d. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
- e. `CreateFD()`

პროცესი თავისი არსით წარმოადგენს პროგრამას შესრულების მომენტში. ყოველ პროცესთან დაკავშირებულია მისი მისამართების სივრცე (მეხსიერების უჯრედების მისამართების სიმრავლე, ერთობლიობა), რომლიდანაც ის კითხულობს და რომელშიც წერს მონაცემებს. მრავალ ოპერაციულ სისტემაში ყველა პროცესზე ინფორმაცია, პროცესის მისამართების სირვცეში არსებული მონაცემების გამოკლებით, ინახება ე.წ. **პროცესების ცხრილში** და ოპერაციულ სისტემაში არსებული ყოველი პროცესისათვის ის წარმოადგენს სტრუქტურათა მასივს.

ფაილი - მყარ დისკზე განთავსებული მონაცემების გარკვეული ერთობლიობა.

სისტემური გამოძახება - მექანიზმი, რომელიც გამოყენებით პროგრამებს აძლევს საშუალებას მიმართოს ოპერაციული სისტემის ბირთვის მიერ შემოთავაზებულ მომსახურეობებს. ინტერფეისი ოპერაციულ სისტემასა და გამოყენებით პროგრამას შორის.

წყვეტა - ეს არის (პროცესორთან მიმართებაში) გარე მოწყობილობის მიერ გენერირებული მოვლენა. აპარატურა აპარატული წყვეტის მეშვეობით ახდენს ცენტრალური პროცესორის ინფორმირებას, რომ მოხდა გარკვეული მოვლენა და საჭიროა დაუყოვნებელი რეაგირება.

განსაკუთრებული სიტუაცია ეს არის პროგრამის მიერ ბრძანების შესრულების მცდელობის შედეგად წარმოშობილი მოვლენა, რომელიც გარკვეული მიზეზების გამო შეუძლებელია შესრულდეს.

მონოლიტური არქიტექტურით ორგანიზებული ოპერაციული სისტემის შემთხვევაში მთლიანი სისტემა მუშაობს როგორც ერთი პროგრამა. ამ ტექნოლოგიით წევისმიერ პროცედურას შეუძლია გამოიძახოს სხვა პროცედურა.

მრავალდონიანი არქიტექტურის შემთხვევაში თითოეულ დონეზე თავმოყრილია სისტემაში მსგავსი ფუნქციების განმახორციელებელი კომპონენტები. დონეები ერთიმეორისგან მაღავენ მათ მიერ განსახორციელებელი ამოცანის გადაწყვეტის მექანიზმებს, მაგრამ მეზობელ დონეებს სთავაზობენ მომსახურეობებს.

მიკრობირთვული არქიტექტურის მიხედვით ოპერაციული სისტემა იყოფა ცალკეულ მოდულებად. ამ მოდულებიდან მხოლოდ ერთი - მიკრობირთვი, მუშაობს ბირთვის რეჟიმში. ყველა მოდული ერთმანეთთან ურთიერთქმედებს მიკრობირთვის მეშვეობით.

კლიენტ-სერვერული მოდელის შემთხვევაში კლიენტსა და სერვერს შორის კავშირი ხორციელდება შეტყობინებათა გადაცემის გზით. კლენტი გარკვეული მომსახურეობით სარგებლობისათვის ადგენს შეტყობინებას საჭირო მომსახურეობაზე ინფორმაციით და გადასცემს მას შესაბამის სამსახურს. შესაბამისი სამსახური შემოსულ შეტყობინებას ამუშავებს და პასუხს უბრუნებს გამომგზავნ პროცესს.

პროცესი ეს არის შესრულებადი პროგრამა მასთან ასოცირებული ბრძანებათა მთვლელის მიმდინარე მდგომარეობით, რეგისტრებითა და ცვლადებით.

პროცესის წარმოქმნა შესაძლებელია გამოწვეული იყოს რამდენიმე მიზეზით: 1) სისტემის ინიციალიზაცია, 2) შესრულებადი პროცესის მიერ ახალი პროცესის წარმოსაქმნელი სისტემური გამოძახება, 3) მომხმარებლის მოთხოვნა ახალი პროცესის წარმოქმნაზე, 4) პაკეტური ინიციალიზაცია.

ფონურ რეჟიმში მომუშავე პროცესებს, რომლებიც წარმოქმნილია გარკვეული აქტიური საქმიანობის წარმოებისათვის, მაგალითად, როგორიცაა, ელექტრონული ფოსტის შემოწმება, პრინტერზე ფაილის ბეჭდვა და ა.შ., დემონები ეწოდებათ.

პროცესის დასრულება შეიძლება გამოწვეული იყოს ერთ-ერთი შემდეგი მიზეზებიდან: 1) ნორმალური დასრულება, 2) შეცდომით გამოწვეული დასრულება, 3) ფატალური შეცდომით გამოწვეული დასრულება (იძულებითი), 4) სხვა პროცესის მიერ შესრულებული სისტემური გამოძახებით გამოწვეული დასრულება (იძულებითი).

ვიტყვით, რომ პროცესი იმყოფება მდგომარეობაში „შესრულება“, თუ მისთვის გამოყოფილია ცენტრალური პროცესორი. ვიტყვით, რომ პროცესი იმყოფება მდგომარეობაში „მზადყოფნა“, თუ ცენტრალური პროცესორის გამოყოფის შემთხვევაში მას შეუძლია გააგრძელოს შესრულება. ვიტყვით, რომ პროცესი იმყოფება მდგომარეობაში „ბლოკირება“, თუ მას არ შეუძლია შესრულების გაგრძელება გარკვეული მომენტის დადგომამდე.

პროცესის მოდელის რეალიზაციისათვის ოპერაციულ სისტემაში არსებობს ე.წ. პროცესების ცხრილი, რომელშიც არსებული ყოველი ჩანაწერი შეესაბამება სისტემაში მიმდინარე მომენტში არსებულ რომელიმე პროცესს.

ოპერაციული სისტემისთვის პროცესის მართვის ბლოკი წარმოადგენს პროცესის მოდელს.

პროცესის რეგისტრული, სისტემური და მომხმარებლის კონტექსტების ერთობლიობას სიმოკლისთვის უწოდებენ პროცესის კონტექსტს.

პროცესორის ერთი პროცესიდან მეორეზე კორექტულად გადასართველად აუცილებელია შესრულებადი პროცესის კონტექსტის შენახვა და იმ პროცესის კონტექსტის აღდგენა, რომელზეც იქნება პროცესორი გადართული. პროცესის ქმედუნარიანობის შენახვა/აღდგენის ასეთ პროცედურას კონტექსტის გადართვა ეწოდება.

ოპერაციული სისტემის იმ ნაწილს, რომელიც განსაზღვრავს თუ რომელ პროცესს უნდა გამოეყოს პროცესორი დამგეგმვი ეწოდება, ხოლო მის მიერ გამოყენებულ ალგორითმს კი - დაგეგმვის ალგორითმი.

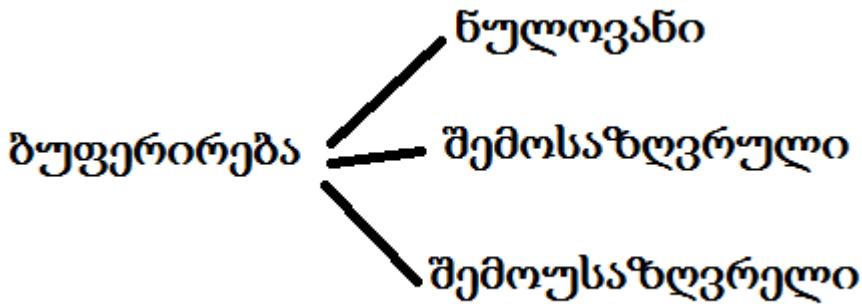
FCFS , SJF, პრიორიტეტული SJF, ციკლური დაგეგმვა RR , პრიორიტეტული დაგეგმვა, მრავალდონიანი მიმდევრობა . // ალგორითმების ჩამოყალიბება

ურთიერთქმედ პროცესებს შორის გადასაცემი მონაცემების მოცულობის და პროცესის ყოფაქცევაზე შესაძლო ზემოქმედების მიხედვით გაცვლის საშუალებები იყოფა სამ ნაწილად: 1) **სიგნალური:** გადასაცემი ინფორმაციის მოცულობა მინიმალურია - ერთი ბიტი. სიგნალური საშუალებით ხდება პროცესორის ინფორმირება გარკვეული მოვლენისდადგომის და მასზე შესაბამისი რეაგირების საჭიროების თაობაზე. 2) **არხული:** პროცესებს შორის მონაცემების გაცვლა ხორციელდება ოპერაციულისისტემის მიერ ამ მიზნისათვის სპეციალურად გამოყოფილი კავშირის არხებით. გადასაცემი მონაცემების მოცულობა დამოკიდებულია კავშირის არხისგამტარუნარიანობაზე. 3) **განაწილებადი მეხსიერება:** ორ ან რამდენიმე პროცესს შეუძლია შეთანხმებულადგამოიყენოს მისამართების სივრცის გარკვეული ნაწილი. გადასაცემი მონაცემებისმოცულობა დამოკიდებულია მეხსიერების გამოყოფილ ნაწილზე.

კავშირის დამყარებასთან მჭიდრო კავშირშია კავშირის საშუალებების გამოყენებისას დამისამართების მეთოდები. განასხვავებენ დამისამართების ორ მეთოდს: **პირდაპირი და არაპირდაპირი.** პირდაპირი დამისამართებისას ურთიერთქმედი პროცესები პირდაპირ ურთიერთქმედებენ ერთიმეორესთან. გაცვლის ყოველი ოპერაციისას ცხადად ეთითება იმ პროცესის სახელი ან ნომერი, რომელსაც ეგზავნება მონაცემები, ან რომელმაც უნდა გამოაგზავონს ისინი. თუ გამგზავნი და მიმღები პროცესი ცხადად უთითებს ურთიერთქმედების პარტნიორს, მაშინ დამისამართების ასეთ სქემას **სიმეტრიული** პირდაპირი დამისამართება ეწოდება. წინააღმდეგ შემთხვევაში დამისამართების სქემას ეწოდება ასიმეტრიული პირდაპირი დამისამართება. **არაპირდაპირი დამისამართებისას** გამგზავნი პროცესის მიერ გაგზავნილი მონაცემები თავსდება გარკვეულ შუალედურ ობიექტში, საიდანაც მისი აღება შეუძლია მსგავსი მონაცემების საჭიროების მქონე რომელიმე პროცესს. ამასთან, არცერთი მხარე, რომელმაც განათავსა მონაცემები შუალედურ ობიექტში და რომელმაც აიღო ისინი იქიდან, არ საჭიროებს მეორის იდენტიფიცირებას.

კავშირს, რომელიც არის ერთმიმართულებიანი, ეწოდება **სიმპლექსური**, ხოლო ორმიმართულებიანს მონაცემების სხვადასხვა მიმართულებით მიმდევრობით გადაცემის შესაძლებლობებით - **ნახევრად დუპლექსური**, ხოლო მონაცემების ერთდროული გადაცემის შესაძლებლობით - **დუპლექსური**.

კავშირის არხული საშუალებების ლოგიკური რეალიზება :



კავშირის არხსი გამოყენებით მონაცემთა გადაცემის ორი მოდელი არსებობს - **შეტანა/გამოტანის** ნაკადი და **შეტყობინების** ნაკადი.

pipe ახდენს შეტანა გამოტანის მოდელის რეალიზებას. ინფორმაციას ფლობს ახალი პროცესის წარმომქმნელი პროცესი, რომელსაც მხოლოდ შთამომავალს უზიარებს.

FIFO არხის საშუალებით შეგვიძლია ნებისმიერ პროცესებს შორის კავშირის დამყარება.

კომუნიკაციის საშუალებას ეწოდება **საიმედო**, თუ : 1) მონაცემები არ იკარგება, 2) არ ზიანდება, 3) არ ჩნდება მათში ზედმეტი ინფორმაცია, 4) არ ირღვევა გაცვლის თანმიმდევრობა.

ნაკადი(Thread) ეს არის პროცესის შიგნით არსებული მსგავსი მინიპროცესი. მომხმარებლის რეჟიმში thread-ები რეალიზებულია ერთიანი სტრუქტურით. ისინი სრულდებიან პროგრამების შესრულების სისტემაში (run time system). **ბირთვის რეჟიმში** thread-ების რეაზლიზებისას run time system-ის ან მსგავსი პროგრამების გამოყენების აუცილებლობა არ არის. ასევე ყოველ thread-ში არ არის პროცესების ცხრილი. ნაცვლად ამისა ბირთვში არის thread-ების ცხრილი, რომელშიც მოთავსებულია სისტემაში არსებული ყველა thread-ი.

სიტუაცია, რომლის დროსაც ორი ან რამდენიმე პროცესი იყენებს საერთო რესურსს და მიღებული შედეგი დამოკიდებულია ბოლოს შესრულებული პროცესზე, ეწოდება **შეჯიბრის** მდგომარეობა.

პროგრამის იმ ნაწილს, რომელშიც გამოიყენება საერთო მეხსიერებაზე წვდომა, კრიტიკული სექცია ეწოდება.

ურთიერთგამორიცხვა აქტიური ლოდინით შეიძლება რეალიზებული იყოს რამდენიმე მეთოდით. **წყვეტის აკრძალვა** ერთპროცესორული სისტემის შემთხვევაში გულისხმობს ტაიმერიდან წყვეტის აკრძალვის განხორციელებას და შესაბამისად ამ დროს პროცესორი ვერ შეძლებს სხვა პროცესზე გადართვას. მრავალპროცესორული სისტემის შემთხვევაში წყვეტის აკრძალვა მოქმედებს მხოლოდ იმ პროცესორზე, რომელსაც შეეხება ბლოკირების ინსტრუქცია. ცვლადი-ბოქსლობის მეთოდის გამოყენება გულისხმობს სისტემაში პროცესებისათვის საერთო

ცვლადის შემოღებას, რომელიც გააკონტროლებს პროცესების შესვლას საკუთარ კრიტიკულ სექციაში. ცვლადი-ბოქლომის საწყისი ინიციალიზაცია ხდება ნულოვანი მნიშვნელობით, რაც გულისხმობს რომ არცერთი პროცესი არ იმყოფება საკუთარ კრიტიკულ სექციაში და ნებისმიერ მათგანს სურვილის შემთხვევაში შეუძლია შევიდეს იქ. პროცესი საკუთარ კრიტიკულ სექციაში შესვლის შემდეგ ცვლის ცვლადი-ბოქლომის მნიშვნელობას 1-ით, რაც იმის მანიშნებელია, რომ ურთიერთქმედი პროცესებიდან ერთ-ერთი იმყოფება საკუთარ კრიტიკულ სექციაში. კრიტიკული სექციის პრობლემის გადაწყვეტა მკაცრი მიმდევრობის მეთოდით გულისხმობს პროცესების წარმოდგენას მკაცრი მიმდევრობის სახიდ, პროცესები მიმდევრობით შედიან საკუთარ კრიტიკულ სექციაში და იქიდან გამოსვლის შემდეგ მიმდევრობის შემდეგ წევრს ეძლევა საშუალება შევიდეს საკუთარ კრიტიკულ სექციაში.

როცა პროცესი იბრძვის გაკრვეული რესურსის დასაკავებლად, რომელიც დაკავებულია ბლოკირებული პროცესის მიერ, იგი ელოდება ისეთ მოვლენას, რომელიც არასდროს მოხდება, ანუ **ჩიხშია**. სისტემის ჩიხის, ან სისტემის ჩაციკვლის მდგომარეობა წარმოადგენს იმის შედეგს, რომ ერთი ან რამდენიმე პროცესი იმყოფება ჩიხში, ამას კი ურთიერთბლოკირება ეწოდება.

განაწილებადია რესურსი, რომლის ჩამორთმევაც პროცესისთვის უმტკივნეულოა, ხოლო გაუნაწილებელია რესურსი, რომლის ჩამორთმევამ შეიძლება გამოიწვიოს მნიშვნელოვანი სამუშაოს დაკარგვა.

ურთიერთბლოკირების წარმოქმნისათვის აუცილებელია და საკმარისი შემდეგი ოთხი პირობის შესრულება:

- 1) ურთიერთგამორიცხვის პირობა (დროის ნებისმიერ მომენტში რესურსი ან გამოყოფილია მხოლოდ ერთი პროცესისთვის ან თავისუფალია)
- 2) რესურსის ლოდინის პირობა (პროცესს, რომელსაც დაკავებული აქვს გარკვეული რესურსი, შეუძლია მოითხოვოს ახალი რესურსი)
- 3) გაუნაწილებლობის პირობა (რესურსის იძულებითი ჩამორთმევა შეუძლებელია)
- 4) ციკლური ლოდინის პირობა (უნდა არსებობდეს ორი ან მეტი პროცესისგან შემდგარი წრიული მიმდევრობა, რომელშიც ყოველი წევრი ელოდება მისი შემდგომი წევრის რესურსის განტავსებას).

ჩამოყალიბებული პირობებიდან ერთერთის დარღვევის შემთხვევაში სისტემაში არ გვაქვს ურთიერთბლოკირების მდგომარეობა.

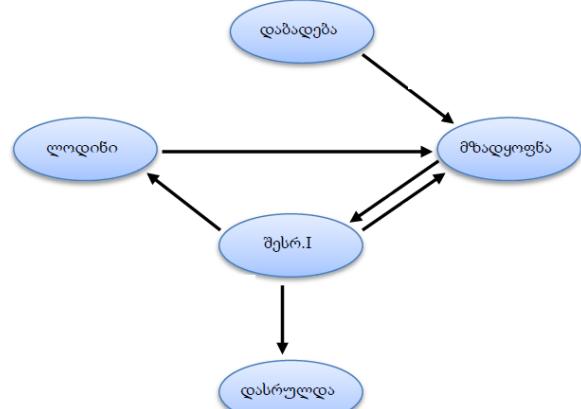
1. ახალი დირექტორიის შესაქმნებლად გამოყენება ბრძანება (1 ქულა)

- ა) chdir; ბ) createdir; გ) mkdir; დ) dircreate ; ე) ჩამოთვლილთაგან არცერთი.

2. თუ პროცესი აღჭურვილია შესასრულებლად საჭირო რესურსებით, მაშინ მდგომარეობის დიაგრამაზე ის გადაადგილდება შემდეგი მიმდევრობით (1 ქულა)

- ა) შესრულება → ბლოკირება → დასრულება ;
ბ) მზადყოფნა → ბლოკირება → შესრულება → დასრულება ;
გ) მზადყოფნა → შესრულება → ბლოკირება → შესრულება → დასრულება ;
დ) ჩამოთვლილთაგან არცერთი.

პასუხი: რადგანაც პირობის თანახმად პროცესი აღჭურვილია შესასრულებლად საჭირო ყველა რესურსით, ამიტომ პროცესების დიაგრამაზე გადაადგილება მოხდება შემდეგი მიმდევრობით:
მზადყოფნა → შესრულება → დასრულება



3. pwd ბრძანების მეშვეობით ხდება (1 ქულა)

- ა) სამუშაო დირექტორიის გამოტანა
ბ) სამუშაო დირექტორიის შეცვლა ;
გ) სამუშაო დირექტორიაში შემავალი ფაილებისა და დირექტორიის ჩამონათვალის გამოტანა ;
დ) ჩამოთვლილთაგან არცერთი.

4. f1.txt და f2.txt ფაილში არსებული ინფორმაციის (მითითებული თანმიმდევრობით) f.txt ფაილში ჩასაწერად ბრძანება ტერმინალში უნდა შესრულდეს ფორმით (1 ქულა)

- ა) cat f1.txt f2.txt < f.txt ; ბ) mv f1.txt f2.txt > f.txt ; გ) mv f1.txt f2.txt f.txt ;
დ) ჩამოთვლილთაგან არცერთი.

პასუხი: cat f1.txt f2.txt > f.txt

5. ტერმინალში შემდეგი ბრძანების შესრულების შედეგად (1 ქულა)

mv f[a-c]* dir (dir - დირექტორიაა)

- ა) Desktop დირექტორიიდან რეკურსიულად წაიშლება ყველა ფაილი, რომლის სახელიც იწყება fa, fb ან fc-თი ;
ბ) სამუშაო დირექტორიდან ყველა ფაილი, რომლის სახელიც იწყება fa, fb ან fc-თი გადატანილი იქნება dir დირექტორიაში;
გ) dir დირექტორიიდან წაიშლება ყველა ფაილი, რომლის სახელიც იწყება fa, fb ან fc-თი;
დ) ჩამოთვლილთაგან არცერთი.

პასუხი: f - აღნიშნავს მხოლოდ ერთ (f) სიმბოლოს

[a-c] - აღნიშნავს ან a ან b ან c სიმბოლოს

* - აღნიშნავს ნებისმიერ სიმბოლოს ნებისმიერი რაოდენობით

mv (move) ბრძანება გამოიყენება ფაილის ან დირექტორიის ერთი დირექტორიიდან სხვა დირექტორიაში გადასაადგილებლად (ამოჭრა და სხვაგან დაკოპირება);

6. შემდეგი პროგრამის შესრულების შედეგად (1 ქულა)

```
int main(){
    execl("cp","cp -r dir1 dir2",NULL);
    return 0;
}
```

- ა) მოხდება dir1 დირექტორიის რეკურსიული კოპირება dir2 დირექტორიაში ;
- ბ) მოხდება dir1 დირექტორიის რეკურსიული გადატანა dir2 დირექტორიაში (ამოჭრა და გადატანა);
- გ) მოხდება r და dir1 დირექტორიების კოპირება dir2 დირექტორიაში ;
- დ) ჩამოთვლილთაგან არცერთი.

პასუხი: cp (copy) ბრძანება გამოიყენება ფაილის ან დირექტორიის კოპირებისათვის
 -r ოფცია გამოიყენება დირექტორიის რეკურსიული (მასში შემავალ ფაილებთან და
 ქვედირექტორიებთან ერთად) კოპირებისათვის

7. პროცესს, რომელიც დასრულების შემდეგ დარჩა სისტემაში ეწოდება (1 ქულა)

- ა) შვილი-პროცესი; ბ) დემონი; გ) ზომბი; დ) ჩამოთვლილთაგან არცერთი.

8. მოცემული ცხრილისათვის

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅	P ₆	P ₇
შესრულების დრო	11	5	10	3	8	7	2

I. FCFS ალგორითმით P₃ და P₅ პროცესების შესრულების დროების ჯამი იქნება (1 ქულა)

- ა) 55 ; ბ) 59 ; გ) 63 ; დ) 70 ; ე) ჩამოთვლილთაგან არცერთი.

პასუხი: P₃ პროცესი შესასრულებლად საჭიროებს $11+5+10 = 26$ ერთეულს

P₅ პროცესი შესასრულებლად საჭიროებს $11+5+10+3+8 = 37$ ერთეულს

II. SJF ალგორითმით P₁ და P₆ პროცესების ლოდინის დროების ჯამი იქნება (1 ქულა)

- ა) 53 ; ბ) 45 ; გ) 40; დ) 47 ; ე) ჩამოთვლილთაგან არცერთი.

პასუხი: P₁ პროცესი ელოდება $2+3+5+7+8+10 = 35$ ერთეულს

P₆ პროცესი ელოდება $2+3+5 = 10$ ერთეულს

9. შემდეგი ფრაგმენტის შესრულების შედეგად დაიბეჭდება (1 ქულა)

```
char mass[N];
int fd = open("abc fifo", O_RDWR);
if (write("abc fifo", mass, N) != N) printf("fifo არხში ჩაიწერა ინფორმაცია");
else printf("fifo არხში არ ჩაიწერა ინფორმაცია");
```

- ა) fifo არხში ჩაიწერა ინფორმაცია;

- ბ) **fifo არხში არ ჩაიწერა ინფორმაცია;**

პასუხი: write() სისტემური გამოძახების გამოყენების სწორი ფორმაა write(fd, mass, N), სადაც fd open()
 სისტემური გამოძახების მიერ დაბრუნებული მნიშვნელობაა

1. ტერმინალში შემდეგი ბრძანების შესრულების შედეგად (1 ქულა)

ls /home/usr/Desktop

- ა) გამოტანილი იქნება /home/usr/Desktop დირექტორიაში შემავალი ფაილებისა და დირექტორიების ჩამონათვალი;
- ბ) გამოტანილი იქნება ინფორმაცია სამუშაო დირექტორიაზე ;
- გ) /home/usr/Desktop დირექტორია იქცევა სამუშაო დირექტორიად ;
- დ) ჩამოთვლილთაგან არცერთი.

2. ტერმინალში შემდეგი ბრძანების შესრულების შედეგად (1 ქულა)

< g.txt cat > f.txt

- ა) ტერმინალის ეკრანზე არსებული ინფორმაცია ჩაიწერება f.txt ფაილში;
- ბ) f.txt ფაილში არსებული ინფორმაცია ჩაიწერება g.txt ფაილში;
- გ) g.txt ფაილში არსებული ინფორმაცია ჩაიწერება f.txt ფაილში;
- დ) ჩამოთვლილთაგან არცერთი.

პასუხი: სიმბოლო ‘<’ შეესაბამება გამოტანის ნაკადს, ხოლო ‘>’ სიმბოლო კი შეტანის ნაკადს.

ტერმინალში გამოსატანი ნაკადი < g.txt გადამისამართებული იქნება f.txt ფაილში

3. ტერმინალში შემდეგი ბრძანების შერულებით (1 ქულა)

mkdir /home/usr/dir

- ა) ხორციელდება /home/usr/dir დირექტორიის რეკურსიული წაშლა სამუშაო დირექტორიიდან ;
- ბ) ხორციელდება /home/usr/dir დირექტორიის გადაადგილება სამუშაო დირექტორიაში ;
- გ) /home/usr/dir დირექტორია იქცევა სამუშაო დირექტორიად ;
- დ) ჩამოთვლილთაგან არცერთი.

პასუხი: mkdir (make directory) ბრძანება გამოიყენება ახალი დირექტორიის შესაქმნელად.

შეიქმნება ახალი დირექტორია სრული სახელით /home/usr/dir

4. /home/usr/f.txt ფაილზე მომხმარებლის კითხვის უფლების ჩაწერის უფლებით შესაცვლელად

ტერმინალში ბრძანება უნდა შესრულდეს ფორმით (1 ქულა)

- ა) chmod a+r /home/usr/f.txt ;
- ბ) chmod /home/usr/f.txt a+r ;
- გ) chmod g+r=w /home/bin/f.txt ;
- დ) ჩამოთვლილთაგან არცერთი.

პასუხი: chmod u r=w /home/bin/f.txt (chmod ბრძანება გამოიყენება ფაილზე მომხმარებლის

უფლებების შესაცვლელად, u (user) მომხმარებელი, r=w (კითხვა = ჩაწერა))

5. ტერმინალზე ფაილში ჩაწერილი ინფორმაციის გამოსატანად გამოიყენება ბრძანება (1 ქულა)

- ა) mkdir ;
- ბ) mkfifo ;
- გ) cat ;
- დ) ls -al ;
- ე) ჩამოთვლილთაგან არცერთი.

6. შემდეგი პროგრამის შესრულების შედეგად (1 ქულა)

```
int main(){  
    execl("ls","ls -al > f.txt",NULL);  
    return 0;  
}
```

- ა) /home/usr/Desktop დირექტორიაში არსებულ ფაილებსა და დირექტორიბზე დაწვრილებითი ინფორმაცია (სახელი, მოცულობა, დაშვების უფლებები და ა.შ.) ჩაწერილი იქნება file ფაილში;
- ბ) **სამუშაო დირექტორიაში არსებულ ფაილებსა და დირექტორიბზე დაწვრილებითი ინფორმაცია (სახელი, მოცულობა, დაშვების უფლებები და ა.შ.) ჩაწერილი იქნება file ფაილში;**
- გ) სამუშაო დირექტორიდან წაიშლება f.txt ფაილი;
- დ) ჩამოთვლილთაგან არცერთი.

7. fifo არხის გამოყენება შეუძლია (1 ქულა)

- ა) მხოლოდ შემობელი-შვილი დამოკიდებულებით დაკავშირებულ პროცესებს ;
- ბ) **სხვადასხვა მანქანებზე განთავსებულ მემკვიდრე პროცესებს ;**
- გ) სისტემაში ან მოშორებულ მანქანებზე არსებულ ნებისმიერ პროცესს ;
- დ) ჩამოთვლილთაგან არცერთი.

პასუხი: fifo არხის გამოყენება შეუძლია, როგორც ერთი გამოთვლითი სისტემის შიგნით არსებულ პროცესებს ისე, მოშორებულ გამოთვლით მანქანებზე არსებულ პროცესებს.

8. მოცემული ცხრილისათვის

პროცესი (კვანტი 5)	P ₁	P ₂	P ₃	P ₄	P ₅	P ₆	P ₇
შესრულების დრო	12	5	3	6	8	70	9

I. RR ალგორითმით P₁ პროცესის შესრულების დრო იქნება (1 ქულა)

- ა) 60 ; ბ) 50 ; გ) 63 ; დ) **53** ; ე) ჩამოთვლილთაგან არცერთი.

პასუხი: რადგანაც კვანტი 5 ერთეულია და P₁ პროცესი შესასრულებლად საჭიროებს 12 ერთეულს, ამიტომ ციკლი სრულად უნდა დატრიალდეს 2-ჯერ და მესამე ჯერზე P₁ პროცესი დასრულდება. შესაბამისად P₁ პროცესი შესასრულებლად საჭიროებს

$$12+5+3+6+8+10+9=53 \text{ ერთეულს}$$

II. RR ალგორითმით P₃ და P₅ პროცესების ლოდინის დროების ჯამი იქნება (1 ქულა)

- ა) 34 ; ბ) 39 ; გ) 50; დ) 52 ; ე) ჩამოთვლილთაგან არცერთი.

პასუხი: ანალოგიური მსჯელობის ჩატარებით P₃ პროცესი ელოდება $5+5=10$ ერთეული, ხოლო P₅ პროცესი ელოდება $10+5+3+6+5+5=34$

9. შემდეგი ფრაგმენტის შესრულების შედეგად (ყველა სისტემური გამოძახება წარმატებით დასრულდა) პროცესების რაოდენობა იქნება (1 ქულა)

```

int a, b, c;
if ((a=fork())==0){
    fork();
}
else {
    if((b=fork())!=0){
        fork();
    }
    c = fork();
    fork();
}

```

ორი პროცესი

ორი პროცესი

ოთხი პროცესი

- ა) **8** ; ბ) 6 ; გ) 10 ; დ) 4 ; დ) ჩამოთვლილთაგან არცერთი.

1. perror() ფუნქციის გამოყენებით ხდება (1 ქულა)

- ა) წარმოქმნილი შეცდომის გამოტანა;
- ბ) ახალი პროცესის წარმოქმნა;
- გ) განაწილებადი მეხსიერების შექმნა;
- დ) ჩამოთვლილთაგან არცერთი.

2. სამუშაო დირექტორიის შესაცვლელად გამოიყენება ბრძანება (1 ქულა)

- ა) cd ; ბ) pwd ; გ) ls -al ; დ) mkdir ; ე) ჩამოთვლილთაგან არცერთი.

3. ტერმინალიდან შემდეგი ბრძანების შესრულების შედეგად (1 ქულა)

cat > f.txt

- ა) ტერმინალის ეკრანზე გამოტანილი იქნება f.txt ფაილში არსებული ინფორმაცია;
- ბ) შეიქმნება f.txt ფაილი და გადავალთ მისი რედაქტირების რეჟიმში;
- გ) file ფაილში ჩაიწერება სამუშაო დირექტორიაში არსებულ ფაილებსა და დირექტორიზე დაწვრილებითი ინფორმაცია (სახელი, მოცულობა, დაშვების უფლებები და ა.შ.);
- დ) ჩამოთვლილთაგან არცერთი.

4. f.txt ფაილზე მომხმარებლის შესაცვლელად ტერმინალში ბრძანება უნდა შესრულდეს შემდეგი ფორმით (1 ქულა)

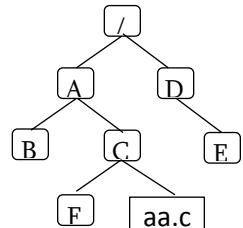
- ა) chown f.txt ; ბ) chmod o-w+x f.txt ; გ) chgrp GID f.txt ;
- დ) chgrp UID f.txt ; ე) ჩამოთვლილთაგან არცერთი.

პასუხი: chown user f.txt ან chown UID f.txt

5. თუ B მიმდინარე დირექტორიაა, მაშინ aa.c ფაილის მიმართებითი მისამართი იქნება (1 ქულა)

- ა) ../B/A/C/aa.c ; ბ) /A/C/aa.c ; გ) .. /A/C/aa.c ;
- დ) ჩამოთვლილთაგან არცერთი.

პასუხი: /B/A/C/aa.c, სიმბოლო ‘..’ ვიყენებთ იმ შემთხვევაში, თუ დანიშნულების ფაილისაკენ ან დირექტორიისაკენ გზაზე გვიჩევს საბაზო დირექტორიის გავლა



6. შემდეგი პროგრამის შესრულების შედეგად (1 ქულა)

```
int main(){  
    (void) execl("/usr/bin/gcc","gcc","/home/usr/prog.c","-o","/home/bin/prog.out",NULL);  
    int a = execl("/home/usr/prog.out","./home/usr/prog.out ",NULL);  
    if (a<0) printf("prog.out ფაილი არ შესრულდა");  
    else printf("prog.out ფაილი შესრულდა");  
    return 0; }
```

- ა) prog.c ფაილი დაკომპილირდება და დაიბეჭდება "prog.out ფაილი შესრულდა" ;
- ბ) prog.c ფაილი დაკომპილირდება და დაიბეჭდება "prog.out ფაილი არ შესრულდა" ;
- გ) prog.c ფაილი დაკომპილირდება და დაიბეჭდება შეცდომის აღმნიშვნელი ფრაზა;
- დ) ჩამოთვლილთაგან არცერთი.

პასუხი: prog.c პროგრამული ფაილი კომპილირების შედეგი იწერება ერთ დირექტორიაში (/home/bin), ხოლო მისი შესრულება ხორციელდება მეორე დირექტორიიდან (/home/usr), რაც გამოიწვევს exec() სისტემური გამოძახების წარუმატებელ დასრულებას (ანუ a=-1)

7. f.txt და g.txt ფაილების ერთი დირექტორიიდან სხვა დირექტორიაში გადასაადგილებლად გამოიყენება ბრძანება (1 ქულა)

- ა) cp f.txt g.txt ; ბ) rm -r f.txt g.txt ; გ) cat f.txt g.txt ; დ) mv f.txt g.txt ;
- ე) ჩამოთვლილთაგან არცერთი.

პასუხი: mv (move) ბრძანება გამოიყენაბა ფაილის (დირექტორიის) ერთი დირექტორიიდან მეორე დირექტორიაში გადასაადგილებლად ან იმავე დირექტორიაში ფაილისატვის სახელის გადასარქმევად. მისი გამოყენების ფორმაა

mv “ფაილის (ან ფაილების) სახელი“ “დირექტორიის სახელი“

8. მოცემული ცხრილისათვის (შენიშვნა: მაღალი მნიშვნელობა შეესაბამება დაბალ პრიორიტეტს)

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	8	2	5	7	2
პრიორიტეტი	2	3	1	1	0

I. პრიორიტეტული SJF ალგორითმით P₂ და P₄ პროცესების ლოდინის დროების ჯამი იქნება (1 ქულა)

- ა) 13 ; ბ) 17 ; გ) 15 ; დ) 21 ; ე) ჩამოთვლილთაგან არცერთი.

პასუხი: რადგანაც დაბალი მნიშვნელობა აღნიშნავს მაღალ პრიორიტეტს, ამიტომ პირველი შესრულდება მაღალი პრიორიტეტის პროცესები. თანაბარი პრიორიტეტის შემთხვევაში შესრულდება ნაკლები დროის საჭიროების მქონე პროცესი.

P₂ პროცესი ელოდება $2+5+7+8 = 22$ ერთეულს

P₄ პროცესი ელოდება $2+5 = 7$ ერთეულს

II. FCFS აგმორითმით (პრიორიტეტი არ გვექნება) P₄ პროცესის შესასრულებლად საჭირო დრო იქნება (1 ქულა)

- ა) 22 ; ბ) 25 ; გ) 19 ; დ) 20 ; ე) ჩამოთვლილთაგან არცერთი.

პასუხი: P₄ პროცესი შესასრულებლად საჭიროებს $8+2+5+7 = 22$ ერთეულს

9. FIFO ფაილის შესაქმნელად გამოიყენება სისტემური გამოძახება (1 ქულა)

- ა) S_IFIFO() ; ბ) mknod() ; გ) shmat(); დ) shmget(); ე) ჩამოთვლილთაგან არცერთი.

- ფაილში ჩაწერილი ინფორმაციის დასათვალიერება შეიძლება ბრძანებით
 - cp **bcat**
 - pwd
 - mv
 - არცერთი ჩამოთვლილთაგან
- პროცესს ეწოდება დემონი, თუ
 - ის წარმოიქმნა სისტემის ჩართვასთან ერთად
 - ის მუშაობს ფონურ რეჟიმში**
 - ის დასრულების შემდეგ გარკვეული დროით დარჩა სისტემაში
 - ჩამოთვლილთაგან არცერტი პასუხი არაა სწორი
- სისტემაში წარმოქმნილ ყოველ პროცესს მდგომარეობათა დიაგრამაზე გადაადგილება შეუძლია შემდეგი მიმდევრობით
 - დაბადება ->მზადყოფნა->ბლოკირება->შესრულება->დასრულება
 - დაბადება->მზადყოფნა->შესრულება->ბლოკირება->დასრულება
 - დაბადება->მზადყოფნა->შესრულება->ბლოკირება->შესრულება->დასრულება
 - ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი**
- რამდენი პროცესი გვეყოლება შემდეგი პროგრამული ფრაგმენტის წარმატებით შესრულების შედეგად სისტემაში

```

fork();
if(fork() !=0)
    if(fork() !=0)  fork();
    else {  fork();  fork(); }
else { fork(); fork(); }

```

- ა)20 **ბ) 12** გ)16 დ)28 ე) არცერთი

- თუ პროცესები მოცემულია ქვემოთ მოყვანილი ცხრილით, მაშინ დაგეგმვის RR ალგორითმის გამოყენებით დათვლილი P₂ და P₄ პროცესების შესრულების დროის ჯამის მეხუთედი იქნება

	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	3	6	12	5	4

- ა) 5,5 ბ)4,2 გ) 6.0 დ) 4,0 ე) 5,2

// კვანტი არის 2 და 6.8 გამოდის პასუხი. 15+19=34 34/5= 6.8

- დაწერეთ ფრაგმენტი, რომელშიც პროცესი pipe არხიდან წაიკითხულ პროგრამულ კოდს ჩაწერს არსებულ file.txt ფაილზე დაშვების rwxr---w- უფლებით და შემდეგ გადაარქმევს მას სახელს (ვგულისხმობთ pipe კავშირის არხი არსებობს).

- 7.

- 1) რამდენი პროცესი გვეყოლება შემდეგი პროგრამული ფრაგმენტის წარმატებით შესრულების შედეგად სისტემაში

```

fork();  

fork();  

if ( fork() !=0)  

    if ( fork() !=0) fork();  

else { fork(); fork(); }  

else { fork(); fork(); }

```

ა) 28 ბ) 35 გ) 44 დ) 40 ე) არცერთი

- 2) მდგომარეობას, რომლის დროსაც ინახება მიმდინარე პროცესი და მის მიერ შესრულებული სამუშაო და შესასრულებლად იტვირთება ახალი პროცესი და მისი მონაცემები, ეწოდება

ა) კონტექსტის გადართვა

ბ) შესრულების კონტექსტი

გ) ბლოკირებული მდგომარეობა

დ) ჩამოთვლილთაგან არცერთი

- 3) ბლოკირებული პროცესი შესაბამისი მოვლენის დადგომის შემთხვევაში შეიძლება გადავიდეს მდგომარეობაში

ა) შესრულება ბ) დასრულება გ) დაბადება დ) მზადყოფნა ე) არცერთი

- 4) FIFO არხის შესაქმნელად გამოიყენება სისტემური გამოძახება

ა)mkrpipe ბ)mknod გ)pipe დ)არცერთი

- 5) პროცესები მოცემულია ქვემოთ მოყვანილი ცხრილით, მაშინ დაგეგმვის პროპროტეტული, გამევებადი SJF ალგორითმის გამოყენებით დათვლილი P₃ და P₄ პროცესების შესრულების დროის ჯამის მეხუთედი იქნება

	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	11	6	1	8	4
გამოჩენის დრო	7	3	5	1	0
პრიორიტეტი	1	1	2	2	0

ა) 10.4 ბ) 11.6 გ) 9.8 დ) 10 ე) 12

- 6) დაწერეთ ფრაგმენტი, რომელშიც პროცესის მიერ მოხდება ახალი ფაილის შექმნა და მასში რაიმე მასივში არსებული N მოცულობის პროგრამული კოდის მონაცემების ჩაწერა და შემდეგ ფაილის კომპილაცია.

- 7) დაწერეთ ფრაგმენტი, რომელშიც მოხდება file.txt ფაილისათვის მომხმარებლის შეცვლა root მომხმარებლით.

1. ფაილისათვის სახელის გადარქმევა შესაძლებელია ბრძანებით

ა) rm ბ) cat გ) mv დ) pwd ე) mkdir

2. რამდენი პროცესი გვეყოლება შემდეგი პროგრამული ფრაგმენტის წარმატებით შესრულების
შედეგად სისტემაში

```
fork();  
  
if ( fork() !=0)  
  
    if ( fork() !=0) { fork(); fork(); }  
  
    else { fork(); fork(); }  
  
else { fork(); fork(); }
```

ა) 30 ბ) 24 გ) 16 დ) 20 ე) არცერთი

3. chmod ბრძანების გამოყენებით შესაძლებელია

ა)პროცესის შესრულების კპნტექსტის მნიშვნელობის შეცვლა

ბ)ფაილისათვის სხვადასხვა მომხმარებლების დაშვების უფლებების შეცვლა

გ)pipe კავშირის არხის წარმოქმნა

დ)არცერთი

4. კრიტიკული სექციის მკაცრი მიმდევრობების ალგორითმის მიხედვით

ა)პროცესს აუცილებლობის შემთხვევაში ნებისმიერი მიმდევრობით შეუძლია საკუთარ
კრიტიკულ სექციაში

ბ)პროცესს შეუძლია შევიდეს საკუთარ კრიტიკულ სექციაში, თუ იქ იმყოფება მის წინ მდგომი
პროცესი

გ)პროცესს შეუზღვის შევიდეს საკუთარ კრიტიკულ სექციაში, მას შემდეგ რაც მიმდევრობაში
მის წინ მდგომი პროცესი

დ) არცერთი

5. თუ პროცესები მოცემულია ქვემოთ მოყვანილი ცხრილით, მაშინ დაგეგმვის პროპროტეტული,
გაძევებადი SJF ალგორითმის გამოყენებით დათვლილი ლოდინის და P₂ პროცესების
შესრულების დროის ჯამის მეხუთედი იქნება

	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	5	11	8	3	3
გამოჩენის დრო	6	0	5	7	2
პრიორიტეტი	0	1	1	2	0

ა) 5.6 ბ) 6.4 გ) 4.8 დ) 5.0 ე) 6.0

6. დაწერეთ ფრაგმენტი, რომელშიც პროცესი prog.c ფაილიდან წაიკითხავს მონაცემებს, ჩაწერს pipe
არხში და შემდეგ ფაილს გადაადგილებს /bin დირექტივაში

7. დაწერეთ ფრაგმენტი, რომელშიც პროცესი განახორციელებს file.txt მომხმარებლის ჯგუფის
შეცვლას root ჯგუფით

- 1) Is>file.txt ბრძამენოს შესრულების შედეგად
- a) file.txt ფაილში ჩაიწერება სამუშაო დირექტორიაში არსებული ფაილების სახელები
 b) file.txt ფაილი გაიხსნება კლავიატურიდან შეტანილი მონაცემების ჩასაწერად
 გ) ტერმინალზე გამოდანილი იქნება ინფორმაცია სამუშაო დირექტორიაზე
 დ) არცერთი
- 2) რამდენი პროცესი გვეყოლება შემდეგი პროგრამული ფრაგმენტის წარმატებით შესრულების შედეგად სისტემაში

```

fork();  
  

if ( fork() !=0 )
    if ( fork() !=0 ) fork();
    else { fork(); fork(); }
else    fork();

```

- a) 10 b) 12 g) 16 დ) 20 ე) არცერთი
- 3) პროცესს რომელიც სისტემაში სრულდება ფორნურ რეჟიმში ეწოდება
- ა)ზომბი ბ) შესრულების კონტექსტი გ) დემონი დ) ბირთვის კონტექსტი
- 4) კავშირის არხს, რომლის დროსაც მონაცემების გადაცემა შესაძლებელია მხოლოდ ერთი მიმართულებით, ეწოდება
- ა)სიმპლექსური ბ) დუპლექსური გ)ნახევრადდუპლექსური დ)არცერთი
- 5) თუ პროცესები მოცემულია ქვემოთ მოყვანილი ცხრილით, მაშინ დაგეგმვის პროპროტეტული, გაძვევებადი SJF ალგორითმის გამოყენებით დათვლილი P_1 და P_2 პროცესების ლოდინის დროის ჯამის მეხუთედი იქნება

	P_1	P_2	P_3	P_4	P_5
შესრულების დრო	11	6	1	8	4
გამოჩენის დრო	0	2	2	1	3

- ა) 4.4 ბ)3.2 გ) 4.8 დ) 3.5 ე) 4 //შეცდომაა
- 6) დაწერეთ ფრაგმენტი, რომელშიც მოხდება პროცესის fifo კავშირის არხის შექმნა უფლებით rwxwr-t-- და რაიმე მასივში არსებული N მოცულობის ამ არხში ჩაწერის (მონაცემების მოცულობაა N)
- 7) დაწერეთ ფრაგმენტი, რომელშიც პროცესის მიერ მოხდება file.txt ფაილისათვის სახელის გადარქმევა (იგულისხმება, რომ პროცესი შექმნილია).

- ჩამოთვლილთაგან რომელი ფუნქცია გამოიყენება ნაკადის (thread-ის) შესაქმნელად? **pthread_create()**
- ცნობილია პროგრამაში გამოყენებული ლოგიკური მისამართების დაკავშირება ფიზიკურ მისამართებთან შესაძლებელია რამდენიმე ეტაპზე. რა ეწოდება ეტაპს, რომელიც თუ წინასწარ უცნობია პროგრამის ადგილმდებარეობა აგენერირებს გადატანად კოდს?
- ჩატვირთვის ეტაპი
- ჩამოთვლილთაგან რომელი ფუნქცია გამოიყენება ნაკადის (thread-ის) იდენტიფიკატორის მნიშვნელობის მისაღებად?
- თუ ცნობილია, რომ პროგრამა მოცულობა არის 21010 ბაიტი. რისი ტოლი იქნება ბოლო ფურცელზე პირველივე გამოუყენებელი ბაიტის ნომერი, თუ ბაიტების ინდექსირება იწყება 0-დან და ფურცლის ზომა არის 4096 (4 კბ). (შენიშვნა. პროგრამისთვის გამოყოფილ ყველა ფურცელზე გარდა ბოლო ფურცლისა მონაცემების განსათავსებლად გამოყენებული იქნება ყველა ბაიტი.)

530

- სურათზე ნაჩვენებია რამდენიმე დიაგრამა იმ შემთხვევაში, როცა რესურსები ოპერაციულ სისტემაში წარმოდგენილია სხვადასხვა რაოდენობის რესურსების სახით. მოყვანილი დიაგრამებიდან რამდენ შემთხვევაში იქნება არასაიმედო მდგომარეობა (ანუ წარმოიქმნება ურთიერთბლოკირების მდგომარეობა).

3

- თუ მეხსიერება დაყოფილია ცვლადი მოცულობის ბლოკებად, მაშინ ასეთ დანაწილებას ეწოდება?

სეგმენტური

- ვთქვათ კომპიუტერის მეხსიერება დაყოფილია ფიქსირებული ზომის ბლოკებად. დავუშვათ, რომ კომპიუტერში გვაქვს მხოლოდ ისეთი პროგრამები, რომლის კოდისა და მონაცემების განთავსება არის შესაძლებელი მეხსიერების ერთ ბლოკში და მხოლოდ ერთი პროგრამა შეიძლება განთავსებული იქნას ერთ ბლოკში. ასეთ შემთხვევაში ჩამოთვლილთაგან რომელი ტიპის პრობლემას შეიძლება ჰქონდეს ადგილი?

შიდა ფრაგმენტაციის პრობლემა

- ნაკადის (thread) მიერ pthread_join ფუნქციის გამოყენების შემთხვევაში მონაცემების ფიზიკური მიმდევრობის გათვალისწინებით
- ვთქვათ მოცემულია პროცესების სიმრავლე. ამბობენ, რომ პროცესების ჯგუფი იმყოფება ურთიერთბლოკირების მდგომარეობაში, თუ მიუხედავად იმისა პროცესები გადანომრილია თუ არა ინდექსების მკაცრად ზრდადი მიმდევრობით მიმდევრობაში შემავალი ყოველი პროცესი ელოდება ამავე მიმდევრობაში შემავალი სხვა ერთი ან რამდენიმე პროცესის მიერ დაკავებული ერთი ან რამდენიმე რესურსის გამოთავისუფლებას
- ფაილის ორგანიზაციის მიმდევრობითი მეთოდის გამოყენების შემთხვევაში შესანახ მოწყობილობაზე ფაილის მონაცემების განთავსება ხდება მონაცემების ფიზიკური მიმდევრობის გათვალისწინებით
- სურათზე ნაჩვენები დიაგრამა შეესაბამება პროცესების მიერ რესურსებზე გაკეთებული მოთხოვნათა მიმდევრობის არასაიმედო მდგომარეობას. რამდენი ციკლი იკვრება ამ დიაგრამაზე. (შენიშვნა. ერთიდაიგივე რესურსი ან პროცესი შეიძლება მონაწილეობდეს სხვა ციკლშიც. წრე ააღნიშნავს პროცესს, ხოლო მართვულთხედი კი რესურსს.)
- 3
- თუ ცნობილია, რომ პროგრამა მოცულობა არის 200050 ბაიტი. რამდენი ფურცელი იქნება საჭირო პროგრამის მონაცემების შესანახად, თუ ფურცლის ზომა არის 4096 (4 კბ). (შენიშვნა. პროგრამისთვის გამოყოფილ ყველა ფრუცელზე გარდა ბოლო ფურცლისა მონაცემების

განსათავსებლად გამოყენებული იქნება ყველა ბაიტი.)

49

13. როგორც ცნობილია მეცნიერებმა დაადგინეს ურთიერთბლოკირების წარმოქმნის აუცილებელი და საკმარისი პირობები. ჩამოთვლილთაგან რომელი შეესაბამება პირობას „ურთიერთგამორიცხვა“

პროცესისთვის მის მიერ დაკავებული რესურსის იძულებითი ჩამორთმევა შეუძლებელია.

პროცესმა დაკავებული რესურსი უნდა გაათავისუფლოს საკუთარი სურვილით

დროის ნებისმიერ მომენტში რესურსი ან გამოყოვილია მხოლოდ ერთი პროცესისთვის ან თავისუფალია

14. ქვემოთ ჩამოთვლილთაგან რომელ მონაცემს არ შეიცავს სუპერბლოკი?

დაკავებულ (მონაცემების შემცვლელ) ბლოკზე ინფორმაციას

15. ვთქვათ გამოთვლით სისტემაში პროგრამებისთვის მეხსიერების გამოყოფა ხორციელდება მისი მოთხოვნილების შესაბამისად. თუ რომელიმე პროგრამა დასრულდა მისთვის ადრე გამოყოფილი მისამართების სივრცე საკმარისი ან მეტი პროგრამის განთავსებისთვის, მაშინ აღნიშნული სივრცე იყოფა პროგრამების მოთხოვნის შესაბამისად და ა.შ. მეხსიერების ამ ფორმით დანაწილების შემთხვევაში ჩამოთვლილთაგან რომელი ტიპის პრობლემას შეიძლება ჰქონდეს ადგილი?

გარე ფრაგმენტაციის პრობლემა

16. თუ ფაილის მონაცემების განთავსება შესანახ მოწყობილობაზე ხდება იმ მიმდევრობით, რომელიც ოპერაციული სისტემისთვის ან პროგრამისთვის არის ხელსაყრელი, მაშინ ფაილის ორგანიზაციის ასეთ მეთოდს ეწოდება?

პირდაპირი

17. როგორც ცნობილია მეცნიერებმა დაადგინეს ურთიერთბლოკირების წარმოქმნის აუცილებელი და საკმარისი პირობები. ჩამოთვლილთაგან რომელი შეესაბამება პირობას „რესურსის ლოდინი“?

პროცესს, რომელსაც დაკავებული აქვს გარკვეული რესურსი შეუძლია მოითხოვოს ახალი რესურსი

18. ჩამოთვლილთაგან რომელი ბრძანების გამოყენებით არის შესაძლებელი ფაილზე დაშვების უფლების შეცვლა

chmod

19. როგორც ვიცით პროცესის მიერ მოთხოვნილი რესურსის გამოყოფისა და გამოთავისუფლების მოვლენათა ჯაჭვი შედგება რამდენიმე ეტაპისგან. კერძოდ რამდენი ეტაპისგან შედგება ის?

3

20. -rwxr-x-w- დაშვების უფლებების სიმბოლური ჩანაწერის შესაბამის რვაობით ჩანაწერს ექნება სახე:

0752

21. თუ მომხმარებელმა გადაწყვიტა ოპერაციული სისტემის სწრაფქმედების ამაღლების მიზნით პროცესების გარკვეული ჯგუფი იძულებით გააჩეროს („დააპაუზოს“), მაშინ შესაბამისი პროცესები აღმოჩნდებიან მდგომარეობაში

შექმნებული მზად

22. ჩამოთვლილთაგან კოდის რომელი ფრაგმენტი იძლევა სურათზე ნაჩვენები იერარქიის შესაბამისი დირექტორიების სტრუქტურის მიღების საშუალებას

mkdir Dir_1 Dir_1/Dir_2 Dir_1/Dir_2/Dir_4 Dir_1/Dir_2/Dir_3

23. ვთქვათ ერთპროცესორულ სისტემაში პროცესი იმყოფება შესრულების მდგომარეობაში.

ჩამოთვლილთაგან რომელი წინადადებაა ჭეშმარიტი.

შესრულების მდგომარეობაში მყოფი პროცესის საქმიანობის ბლოკირება შეუძლია მიმდინარე (იმავე) პროცესს

24. რა ეწოდება მექანიზმს, რომელსაც მიმართავს სამომხმარებლო პროგრამა პრივილეგირებული მოქმედენის განხორციელების მიზნით
სისტემური გამოძახება
25. რა ეწოდება პროგრამულ უზრუნველყოფას, რომელიც ოპერაციულ სისტემაში ამა თუ იმ დონეზე ხელს უწყობს ფიზიკური ან აბსტრაქტული კომპონენტის ნორმალურ ფუნქციონირებას
დრაივერი
26. როგორ პროცესს ეწოდება დემონი
პროცესის, თუ ის მუშაობს ფონურ რეჟიმში
27. ჩამოთვლილთაგან რომელი შეესაბამება სისტემაში არსებული (data.in) ფაილის კითხვის უფლებით გახსნის მიზნით იყო სისტემური გამოძახების გამოყენების სწორ ფორმას
`fd = open("data.in", O_RDONLY)`
28. რა ეწოდება კომპონენტს, რომელიც შედგება მიკროსქემის ან მიკროსქემების ნაკრებისგან და ფიზიკურ დონეზე მართავს შესაბამის მოწყობილობას
კონტროლერი
29. ჩამოთვლილთაგან რომელ დონეს მიეკუთვნება ამოცანა, რომელიც ელოდება შესრულებას ზედა დონე
30. ჩამოთვლილთაგან რომელი წარმოადგენს გამოთვლით მაქანაში გამოყენებულ ყველაზე სწრაფ მეხსიერებას.
- რეგისტრი**
31. ჩამოთვლილთაგანა რომელი შეესაბამება საქმიანობას, რომლითაც დაკავებულია ბრძანებათა მთვლელი?
 ინახავს კონკრეტული ერთი პროგრამის ამუშავების შემდეგ ამავე პროგრამის წარმატებული საქმიანობის გაგრძელებისთვის რიგით შემდეგი შესრულებული ბრძანებათა მიმთითებელს
32. სტანდარტულად რამდენი კომპონენტისაგან შედგება შეტანა/გამოტანის მოწყობილობა
2
33. ჩამოთვლილთაგან რომელი შეესაბამება `write` სისტემური გამოძახების გამოყენების სწორ ფორმას, თუ N მასივში ელემენტების რაოდენობაა, ხოლო `fd` კი შესაბამისი ნაკადის სახელი
`char buf[N]`
`write(fd, buf, N);`
34. განსაკუთრებული შემთხვევის რომელ ტიპს მიეკუთვნება პროგრამის შესრულების მიმდინარე მომენტში მეხსიერების ფურცლის არარსებობა
გამოსწორებადი განსაკუთრებული შემთხვევა
35. თუ პროცესორის დაგეგმვა ხდება სტატიკური პრიორიტეტის გამოყენებით, მაშინ ოპერაციულ სისტემაში მზადყოფნის მდგომარეობაში მყოფი პროცესების დალაგება მოხდება
პრიორიტეტის ზრდადი მნიშვნელობის მიხედვით
36. ჩამოთვლილთაგან რომელი წარმოადგენს გამოთვლით მაქანაში გამოყენებულ ყველაზე ნელ მეხსიერებას?
ფლეშ-მეხსიერება
37. ჩამოთვლილთაგან რომელი წინადადებაა ჭეშმარიტი `pipe` კავშირის არხთან მიმართებით
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
38. ჩამოთვლილთაგან რომელი შეესაბამება `fifo` არხის შექმნის სწორ ფორმას
`char name[] = "a fifo"`
`mknod(name, S_FIFO | 0664, 0);`
39. რა ეწოდება აბსტრაქციას, რომელიც გამოთვლით სისტემაში ნაკლები მოცულობის ფიზიკური მეხსიერებსი არსებობის შემთხვევაში ხელს უწყობს მეტი ფიზიკური მეხსიერების საჭიროების მქონე პროგრამის ამუშავებას
ვირტუალური მეხსიერება

40. სისტემაში წარმოქმნილ და წარმატებით დასრულებულ ყოველ პროცესს მდგომარეობათა დიაგრამაზე გადაადგილება შეუძლია შემდეგი მიმდევრობით.
დაბადება → მზადყოფნა → შესრულება → მზადყოფნა → შესრულება → დასრულება
41. ჩამოთვლილთაგან რომელი შეესაბამება ასინქრონული წყვეტის განმარტებას ეს არის წყვეტა, რომელიც წარმოშვა კოდში დაშვებული შეცდომისგან დამოუკიდებლად
42. კავშირის არხის დასრულების პროცესირება რა შემთხვევაში არაა აუცილებელი თუ პროცესებს შორის კავშირის არხის გახსნა არ იყო პროცესირებული სპეციალური მოქმედებით
43. ჩამოთვლილთაგან რომელი სისტემური გამოძახება გამოიყენება მომხმარებლის ჯგუფის იდენტიფიკატორის მისაღებად
getpid
44. 0532 დაშვების უფლების რვაობითი ჩანაწერის შესაბამის სიმბოლურ ჩანაწერს ექნება სახე:
-r-x-wx-w
45. ვთქვათ ოპერაციული სისტემის შექმნისას პროგრამისტმა გამოიყენა ფიქსირებული ზომის სტრუქტურა გარკვეული ტიპის მონაცემების განსათავსებლად, რომელიც დინამიურად ივსება ელემენტებით და თავისუფლდება (იშლება) ელემენტებისგან. როგორი ტიპის განსაკუთრებულ შემთხვევას მიეკუთვნება სიტუაცია, რომლის დროსაც სტრუქტურაში მონაცემების ჩაწერისას მასში შეიძლება არ იყოს ადგილი ახალი მონაცემის განსათავსებლად.
- გამოსწორებადს**
46. წყვეტის რომელ ტიპს მიეკუთვნება კოდში დაშვებული შეცდომის გამო წარმოქმნილი წყვეტა სინქრონულ წყვეტას
47. განსაკუთრებული შემთხვევის რომელ ტიპს მიეკუთვნება პროგრამის შესრულების მიმდინარე მომენტში ისეთ ფაილზე მიმართვა, რომელიც ფაილურ სისტემაში არსებობს, მაგრამ სხვა დირექტორიაში არის განთავსებული
გამოუსწორებელი განსაკუთრებული შემთხვევა
48. წარმატებულად დასრულების შემთხვევაში რას აბრუნებს open სისტემური გამოძახება.
ფაილურ დესკრიპტორს
49. რა ექოდება კომპონენტს, რომელიც ფიზიკურ დონეზე ორ ფიზიკურ კომპონენტს შორის უზრუნველყოფს მონაცემების გადაცემას
სალტე
50. ჩამოთვლილთაგან რომელი წარმოადგენს მონოლიტური არქიტექტურის ნაკლოვანებას?
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
51. ჩამოთვლილთაგან რომელი წინადადებაა ჭეშმარიტი FIFO კავშირის არხთან მიმართებით.
FIFO კავშირის არხის გამოყენება შეუძლია სისტემაში წარმოქმნილ ყველა პროცესს მიუხედავად იმისა როდის ან რომელი პროცესის მიერ მოხდა FIFO კავშირის არხის წარმოქმნა
52. ჩამოთვლილთაგან რომელ მდგომარეობაშ მოხდება პროცესის გადაყვანა თუ მისი შესრულების მომენტში წარმოშობილი წყვეტის გამო პროცესორი გადაერთო წყვეტის დამუშავებაზე და ამ პერიოდში პროცესს ამოეწურა დროითი კვანტი
მზადყოფნის
53. ჩამოთვლილთაგან რომელი სისტემური გამოძახება გამოიყენება მომხმარებლის ჯგუფის იდენტიფიკატორის მნიშვნელობის მისაღებად.
getgid
54. ჩამოთვლილთაგან რომელი სისტემური გამოძახება აბრუნებს ფაილურ დესკრიპტორს?
open()
55. ჩამოთვლილთაგან რომელი წარმოადგენს ენერგო-დამოუკიდებელ მეხსიერებას?
Solid State Disk (SSD)
56. ვთქვათ სისტემაში გვაქვს პროგრამა, რომლის კომპილაციის შედეგად კომპილატორი აგენტერირებს გადასატან კოდს. ასეთ შემთხვევაში ჩამოთვლილთაგან რომელ ეტაპზე იქნება

შესაძლებელი ფიზიკური და ლოგიკური მეხსიერების მისამართების დაკავშირება?

ჩატვირთვის ეტაპი

57. თუ dir5 არის სამუშაო დირექტორია, მაშინ ტერმინალში შემდეგი ბრძანების შესრულების შედეგად rm ..//dir4/file1 dir8/file6
ფაილური სისტემიდან წაიშლება ..//dir4/file1 და dir8/file6 ფაილები
58. ჩამოთვლილთაგან რომელი შეესაბამება კრიტიკული სექციის განმარტებას მეხსიერების არე, რომელსაც იზიარებს ორი ან მეტი პროცესი
59. ვთქვათ სისტემაში პროცესები წარმოდგენილია ქვემოთ მოყვანილი ცხრილის მეშვეობით. RR ალგორითმის გამოყენებით იპოვეთ P_3 , P_5 და P_9 პროცესების შესრულების საერთო დროის ნახევარი, თუ დროით კვანტის მნიშვნელობა არის 2.

პროცესები	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
შესრულების დრო	2	5	3	7	2	6	2	4	1	5

შენიშვნა. შესაბამისი გამოთვლების ჩატარება შესაძლებელია აქვე.

P_3 პროცესისთვის 22; P_5 პროცესისთვის 10; P_9 პროცესისთვის 17; მათი საშუალო = 16,333

60. ჩამოთვლილთაგან, რომელი შეესაბამება დუპლექსური კავშირის არხის განმარტებას. კავშირის არხს, რომლის მეშვეობითაც შესაძლებელია ორივე მიმართულებით მონაცემების გადაცემა ერთდროულად
61. როგორ იშიფრება აბრევიატურა RAM
Random Access Memory
62. ვთქვათ, “Dir 5” არის სამუშაო დირექტორია, რომელშიც ახორციელებთ მიმდინარე საქმიანობას. “Dir 5” დირექტორიასთან მიმართებით ჩამოთვლილთაგან რომელი იქნება “File 11” ფაილის მიმართებითი სახელი
.. $/Dir 7/Dir 9/File 11$
63. ჩამოთვლილთაგან რომელი შეესაბამება სიმპლექსური კავშირის არხის განმარტებას. კავშირის არხს, რომლის მეშვეობითაც შესაძლებელია მემკვიდრე პროცესებს შორის მონაცემების გაცვლა
64. კავშირის არხის დასრულების პროცესირება რა შემთხვევაში არაა აუცილებელი
?
65. ჩამოთვლილთაგან რომელი ბრძანებითაა შესაძლებელი ფაილის შიგთავსის დათვალიერება მისი გახსნის გარეშე.
cat
66. ჩამოთვლილთაგან რომელი წარმოადგენს SSD დისკის ნაკლოვანებას HDD დისკთან მიმართებით
SSD დისკს გააჩნია სიცოცხლის ნაკლები ხანგრძლივობა ვიდრე HDD დისკს
67. როგორ პროცესს ეწოდება ზომბი
პროცესს, თუ ის დასრულების შემდეგ გარკვეული დროით დარჩა სისტემაში
68. რა ეწოდება პროცესს, რომელიც საქმიანობის დასრულების შემდეგ გარკვეული დროით დარჩა სისტემაში
ზომბი
69. პროგრამული კოდის შემუშავებისას პროგრამისტის დასჭრდა 1000 char ტიპის მნიშვნელობებიანი მასივის განთავსება განაწილებად მეხსიერებაში. ქვემოთ ჩამოთვლილთაგან პროგრამული კოდის რომელი ფრაგმენტითაა შესაძლებელი (key გასაღებისთვის) განაწილებადი მეხსიერების შექმნა და გამოყენება?
ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი
70. ჩამოთვლილთაგან რომელი შეესაბამება პროცესის კონტექსტის გადართვის განმარტებას.

ჩამოთვლილთაგან რომელი შეესაბამება პროცესის კონტექსტის გადართვის განმარტებას.

აღნიშვნელი ერთი:

- a. მდგომარეობას, რომლის დროსაც პროცესი გადადის ბლოკირების მდგომარეობიდან მზადყოფნის მდგომარეობაში
- b. მდგომარეობას, რომლის დროსაც აქტიურ მდგომარეობაში მყოფ პროცესი საკუთარი საქმიანობის შემსუბუქების მიზნით ქმნის
- c. მდგომარეობა, რომლის დროსაც ინახება მიმდინარე პროცესის მიერ შესრულებული საქმიანობა და შესასრულებლად იტვირთება
- d. ჩამოთვლილთაგან ყველა პასუხი სწორია (იგულისხმება უარყოფითი პასუხის გარდა ყველა)
- e. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი

სწორი პასუხია:

მდგომარეობას, რომლის დროსაც ინახება მიმდინარე პროცესის მიერ შესრულებული საქმიანობა და შესასრულებლად იტვირთება აპროცესის მონაცემები

71. ჩამოთვლილთაგან რომელი შეესაბამება ფაილური დესკრიპტორის განმარტებას.

ჩამოთვლილთაგან რომელი შეესაბამება ფაილური დესკრიპტორის განმარტებას.

აღნიშვნელი ერთი:

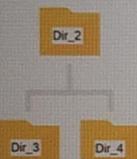
- a. ჩამოთვლილთაგან არცერთი პასუხი არაა სწორი✓
- b. რიცხვითი მნიშვნელობა, რომელიც გამოსახავს პროცესის მიერ შესრულების (ამჟამავების დროიდან მოყოლებული) მთავარი ფაილების რაოდენობას
- c. რიცხვითი მნიშვნელობა, რომელიც გამოსახავს პროცესის მიერ მიმდინარე მომენტში გახსნილი ფაილების რაოდენობას
- d. რიცხვითი მნიშვნელობა, რომელიც გამოსახავს ფაილში ჩაწერილი ბაიტების რაოდენობას
- e. რიცხვითი მნიშვნელობა, რომელიც გამოსახავს ფაილიდან წაკითხული ბაიტების რაოდენობას

72. ჩამოთვლილთაგან რომელი მიეკუთვნება აპარატურული წყვეტის მნიშვნელოვან ტიპს ტაიმერიდან წყვეტა

73. ჩამოთვლილთაგან კოდის რომელი ფრაგმენტი იძლევა სურათზე ნაჩვენები იერარქიის შესაბამისი დირექტორიების სტრუქტურის მიღების საშუალებას

ჩამოთვლილთაგან კოდის რომელი ფრაგმენტი იძლევა სურათზე ნაჩვენები იერარქიის შესაბამისი დირექტორიების სტრუქტურის მიღების საშუალებას

Dir_1



აღნიშვნელი ერთი:

- a. `mkdir Dir_1 Dir_1/Dir_2 Dir_1/Dir_2/Dir_3 Dir_1/Dir_2/Dir_4`
- b. `mkdir Dir_1/Dir_2 Dir_1/Dir_2/Dir_3 Dir_1/Dir_2/Dir_4`
- c. `mkdir Dir_1 Dir_1/Dir_2/Dir_3 Dir_1/Dir_2 Dir_1/Dir_2/Dir_4`
- d. `mkdir Dir_1 Dir_1/Dir_2/Dir_3 Dir_1/Dir_2/Dir_4 Dir_1/Dir_2`

სწორი პასუხია: `mkdir Dir_1 Dir_1/Dir_2 Dir_1/Dir_2/Dir_3 Dir_1/Dir_2/Dir_4`

ჩამოთვალეთ და აღწერეთ პროცესებს შორის მონაცემების გაცვლის საშუალებები

ჩამოთვალეთ და აღწერეთ პროცესებს შორის მონაცემების გაცვლის საშუალებები.



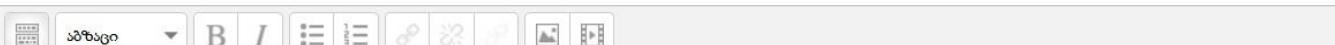
1) სიგნალური - ის გადასცემს ერთ ბაიტს, ის პროცესის ინფორმირებას ახდენს როდესაც რაღაც კონკრეტული მოვლენა ხდება და ასევე რა არის საჭირო ამ მოვლენაზე რეაგირებისთვის. ამ დროს მთავარია პროცესს ჰქონდეს ამ სიგნალის ტიპის ინფორმაციის მიღების და რეაგირების საშუალება

2) არზული - მონაცემების გაცვლა ხდება სპეციალური კავშირის არხებით, მონაცემების ზომა დამოკიდებულია ამ არზების გამტარობაზე და ასევე მონაცემების გაზრდით სწვა პროცესებზე ზემოქმედების საშუალებაც იზრდება

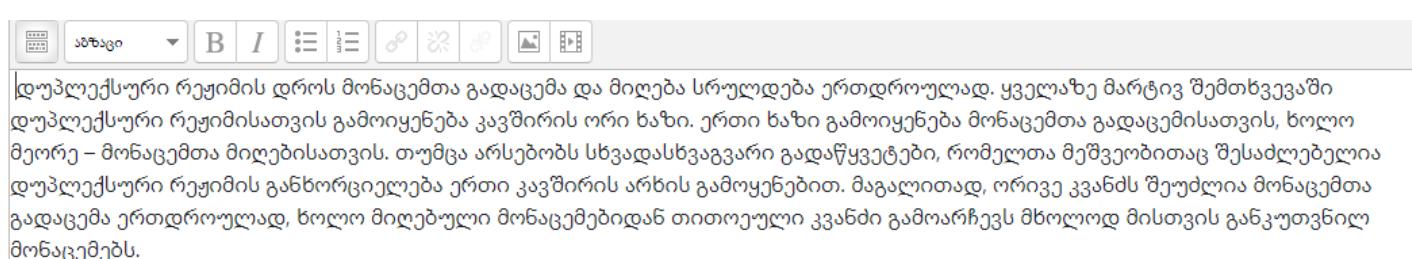
3) განაწილებადი მეხსიერება - რამდენიმე პროცესს შეუძლია გამოიყენოს მეხსიერების მისამართის გარკვეული ნაწილი, მონაცემების ზომა კი ამ მისამართის ზომაზეა დამოკიდებული. ასევე, თუკი დიდ მონაცემებს გადავცემთ ამ გზით, საფრთხილოა, რადგან შეიძლება ამ მისამართის დაკავებით სწვა პროცესების მეხსიერება დავიკავოთ, რამაც შესაძლოა ეს სწვა პროცესები გათიშოს

ახსენით რას ნიშნავს კავშირის არხი არის დუპლექსური

ახსენით რას ნიშნავს კავშირის არხი არის დუპლექსური.



კავშირის არხი არის დუპლექსური, როცა მასთან ასოცირებულ პროცესს აქვს ორმიმართულებიანი კავშირის შესაძლებლობა ერთდროულად. ანუ ინფორმაციის მსიაღებად და მის გადასაცემად ერთდროულად შეუძლია მოქმედება კავშირს.



დუპლექსური რეჟიმის დროს მონაცემთა გადაცემა და მიღება სრულდება ერთდროულად. ყველაზე მარტივ შემთხვევაში დუპლექსური რეჟიმისათვის გამოიყენება კავშირის ორი საზი. ერთი საზი გამოიყენება მონაცემთა გადაცემისათვის, ხოლო მეორე – მონაცემთა მიღებისათვის. თუმცა არსებობს სწვადასხვაგვარი გადაწყვეტები, რომელთა მეშვეობითაც შესაძლებელია დუპლექსური რეჟიმის განხორციელება ერთი კავშირის არხის გამოყენებით. მაგალითად, ორივე კვანძს შეუძლია მონაცემთა გადაცემა ერთდროულად, ხოლო მიღებული მონაცემებიდან თითოეული კვანძი გამოარჩევს მხოლოდ მისთვის განკუთვნილ მონაცემებს.

რას ნიშნავს, რომ ოპერაციული სისტემის არქიტექტურა მონოლიტურია

მონოლიტური არქიტექტურითორგანიზებული ოპერაციული სისტემა ფართოდაა გავრცელებული. ამ შემთხვევაში მთლიანი სისტემა მუშაობს როგორც ერთი პროგრამა, ანუ ოპერაციული სისტემის კოდი დაწერილია როგორც პროცედურათა ნაკრები, რომლებიც ერთ დიდ შესრულებადპროგრამაში არის გაერთიანებული. ამ არქიტექტურაში ყოველპროცედურას შეუძლია გამოიძახოს სხვა პროცედურა, რომელიც გამომძახებელსთვის შეასრულებს სასარგებლოსაქმიანობას.



მოიყვანეთ შეჯიბრის მდგომარეობის განმარტება

პროცესები შესაძლებელია შეთანხმებულად იყენებდნენ იპერაციულსისტემაში არსებულ მონაცემთა საერთოსაცავს. ეს საცავი შეიძლება იყოს განთავსებული ოპერატიულმებსიერებაში ან წარმოდგენილი იყოს რაიმე ფაილის სახით.

აღწერეთ ოპერაციული სისტემის არქიტექტურა "მრავალდონიანი", მისი დადებითი და უარყოფითი მხარეები.

მონოლიტური მიდგომის განზოგადებას წარმოადგენს ოპერაციული სისტემის ორგანიზება დონეების იერარქიის სახით, რომელშიც თითოეული დონეზე თავმოყრილია სისტემაში მსგავსი ფუნქციების განმახორციელებელი კომპონენტები. დონეები ერთიმეორისგან მაღავენ მათმიერ განსახორციელებელი ამოცანის გადაწყვეტის მექანიზმებს, მაგრამ მეზობელი დონეებს სთავაზობენ მომსახურეობას. ასეთნაირადორგანიზებულოპერაციულ სისტემაში, სხვა დონეებზე ზემოქმედების გარეშე შესაძლებელია, ცალკეული დონეების მოდიფიცირება. დონეებს შორის გადასვლა საჭიროებს შუალედური ელემენტების გამოყენებას, რაც ამცირებს სისტემის წარმადობას. გარდა ამისა, ვინაიდან მრავალდონიანი არქიტექტურაში დონეები სარგებლობენ ერთიმეორის მომსახურეობით და რესურსებზე წვდომის შეუზღუდვავი უფლებით, მონოლიტური მიდგომის მსგავსად, ამ შემთხვევაშიც შეიძლება სისტემა აღმოჩნდეს არამდგრადი შეცდომის ან ზიანის შემცველი კოდის მიმართ. THE (Technische Hogeschool Eindhoven) ოპერაციული სისტემა წარმოადგენს მრავალდონიანი ოპერაციული სისტემის მაგალითს (ნახ. 1.10). მრავალი თანამედროვე ოპერაციული სისტემა, მათ შორის Windows XP და Linux-ი შეიძლება გარკვეული თვალსაზრისით განეკუთვნებოდნენ მრავალდონიან სისტემებს.

რას ეწოდება ფაილური დესკრიპტორი

რას ეწოდება ფაილური დესკრიპტორი?

მასივის ელემენტის ინდექსს, რომელიც შეესაბამება შეტანა/გამოტანის გარკვეულ ნაკადს, შემ ნაკადისთვის ფაილური დესკრიპტორი ეწოდება

მოიყვანეთ კრიტიკული სექციის განმარტება
მოიყვანეთ კრიტიკული სექციის განმარტება.

ამჟაღი	B	I	≡	≡	♂	♀	+	-	☒	☒
პროგრამის იმ ნაწილს, რომელშიც გამოიყენება საერთო მეხსიერებაზე წვდომა კრიტიკული სექცია ეწოდება										

დაგეგმვის RR ალგორითმის გამოყენებით დაითვალეთ პროცესების შესრულების

დაგეგმვის RR ალგორითმის გამოყენებით დაითვალეთ პროცესების შესრულების და ლოდინის საერთო დროების საშუალო არითმეტიკული, თუ პროცესები მოცემულია შემდეგი ცხრილით და დროითი კვანტუ არის 5.

პროცესები	P_1	P_2	P_3	P_4	P_5	P_6
შესრულების დრო	7	10	5	8	13	9
გამოშენის დრო	28	18	2	0	13	7

შენიშვნა. არ იზღუდება ამოცანის გაკეთება excel-ის ან word-ის ფაილში.

შესაბამისი ფაილი აუცილებლად უნდა ატვირთოთ.

ამჟაღი	B	I	≡	≡	♂	♀	+	-	☒	☒
--------	---	---	---	---	---	---	---	---	---	---

მოიყვანეთ შეჯიბრის მდგომარეობის განმარტება

მოიყვანეთ შეჯიბრის მდგომარეობის განმარტება.

ამჟაღი	B	I	≡	≡	♂	♀	+	-	☒	☒
შეჯიბრის მდგომარეობა ეწოდება სიტუაციას როდესაც ორი ან რამოდენიმე პროცესი იყენებს საერთო რესურსს და მათი შედეგი დამოკიდებულია ბოლოს შესრულებულ პროცესზე.										

რას ეწოდება წყვეტა ?

წყვეტა არის მექანიზმი, რომელიც software-ს აძლევს საშუალებას, რომ რეაგირება მოახდინოს hardware-იდან შემოსულ სიგნალებზე. სიგნალების დამუშავებისთვის ოპერაციულ სისტემაში გათვალისწინებულია სპეციალური პროგრამები, წყვეტის დამმუშავებლები. წყვეტის სიგნალის აღმოჩენის შემთხვევაში ოპერაციული სისტემა აჩერებს პროცესების დამუშავებას და შემოსულ სიგნალს გადასცემს შესაბამის დამმუშავებელ პროგრამას.

წყვეტა (system calls).interrupt). ეს არის (პროცესორთან მიმართებაში) გარე მოწყობილობის მიერ გენერირებული მოვლენა. აპარატურა აპარატული წყვეტის მეშვეობით ახდენს ცენტრალური პროცესორის ინფორმირებას, რომ მოხდა გარკვეული მოვლენა და საჭიროა დაუყოვნებელი რეაგირება (მაგალითად, მომხმარებელმა დააჭირა ღილაკზე) ან, აცნობებს შეტანა/გამო-ტანის ასინქრონული ოპერაციის დასრულების შესახებ (მაგალითად, დასრულდა დისკიდან მონაცემების კითხვა).

აპარატული წყვეტის მნიშვნელოვანი ტიპია - ტაიმერიდან წყვეტა, რომელიც გენერირდება პერიოდულად გარკვეული დროითი შუალედის გასვლის შემდეგ. ტაიმერიდან წყვეტა ოპერაციული სისტემის მიერ გამოიყენება პროცესების დაგეგმვისას. აპარატული წყვეტის ყოველ ტიპს გააჩნია საკუთარი ნომერი, რომელიც ცალსახად განსაზღვრავს წყვეტის წყაროს. აპარატული წყვეტა ეს არის ასინქრონული მოვლენა ანუ, ის წარმოიშობა იმისგან დამოუკიდებლად დროის მოცემულ მომენტში პროცესორის მიერ კოდის რომელი ფრაგმენტი სრულდება.

აღწერეთ დაგეგმვის SJF ალგორითმის მუშაობის პრინციპი, მისი დადებითი და უარყოფითი ...

აღწერეთ დაგეგმვის SJF ალგორითმის მუშაობის პრინციპი, მისი დადებითი და უარყოფითი მხარეები. მოიყვანეთ მაგალითი.

sjf ishifreba rogorc shortest job first an shortes job next . es aris algoritmi romelic cxriluri saxit naxulobs romeli samushao shesruldeba yvelaze swrafad rom shesrulos shemdegi,

1. rigshi SJFs schirdeba yvelaze patara dro sxva "cxrilur" algoritmebtan shedarebit
2. is aris greedy algoritmi
3. man sheidzleba gamoiwvios starvation tu mokle procesebi gagrdzeldeba . es sheidzleba gadawydes daberebis kocnepciis gamoyenebit sjf sheidzleba gamoyenebul iqnas specialur garemoshi sadac gashvebuli drois zusti shefasebebia xelmisawvdomi

რაში მდგომარეობს წყვეტის აკრძალვის ალგორითმის დადებითი და უარყოფითი მხარეები

(2 ქულა) რაში მდგომარეობს წყვეტის აკრძალვის ალგორითმის დადებითი და უარყოფითი მხარეები (მოკლედ.)

წყვეტის აკრძალვის შემთხვევაში შეუძლებელი იქნება ტაიმერიდან წყვეტის განხორციელება და შესაბამისად პროცესორიც ვერ შეძლებს სწვა პროცესზე გადართვას. ამ შემთხვევაში არ არსებობს საფრთხე რომელიმე პროცესმა განახორციელოს საკუთარ კრიტიკულ სექციაში შესვლა.

მრავალპროცესორული სისტემის შემთხვევაში წყვეტის აკრძალვა მოქმედებს მხოლოდ იმ პროცესორზე, რომელსაც შეეხება ბლოკირების ინსტრუქცია. დანარჩენი პროცესორები ჩვეულ რეჟიმში გააგრძელებენ მუშაობას.

რა ეწოდება პროცესის პრიორიტეტების ინვერსია

რას ეწოდება პროცესის პრიორიტეტების ინვერსია.

პრიორიტეტების ინვერსია, როდესაც დაბალ პრიორიტეტულ პროცეს წვდომას აძლევს რესურსებზე, რომელიც მაღალ პრიორიტეტულ პროცეს სჭირდება და მაღალ პრიორიტეტული პროცესი მუშაობას ვერ განაახლებს სანამ რესურსი არ გათავისუფლდება.

ოპერაციული სისტემები (გამოცდა)

1) mv file1.txt file2.txt რა მოხდება ამ დროს ?

სახელი გადაერქმება file1.txt ს მაგივრად ერქმება file2.txt

2) რესურსს ეწოდება განაწილებადი თუ ...

პროცესის მიერ დაკავებულ რესურსს, რომლის ჩამორთმევაც მისთვის შესაძლებელია „უმტკივნეულოდ“, მიეკუთვნება განაწილებად რესურსს.

3) chmod g-wr file.txt

Chmod ბრძანებების მეშვეუბით ხდება ერთ ან რამდენიმე ფაილზე უფლებების შეცვლა.

4) პროცესისკონტექსტის გადართვა ეწოდება მდგომარეობას , რომლის დროსაც...

კონტექსტის გადართვა. პროცესორის ერთი პროცესიდან მეორეზე კორექტულად გადასართველადაუცილებელია შესრულებადი პროცესის კონტექსტის შენახვა და იმ პროცესის კონტექსტისაღდგენა, რომელზეც იქნება პროცესორი .

5) ოპერაციული სისტემა აგებულია მიკრობირთვული არქიტექტურით ნიშნავს რომ...

მიკრობირთვული არქიტექტურის მიხედვით ოპერაციული სისტემა იყოფა ცალკეულ მოდულებად. ამ მოდულებიდან მხოლოდ ერთი - მიკრობირთვი, მუშაობს ბირთვის რეჟიმში. ყველა მოდული ერთმანეთთან ურთიერთქმედებს მიკრობირთვის მეშვეობით,

6) pipe არხის გამოყენება შეუძლიათ...

მემკვიდრე პროცესებს, მშობელს და შვილს

7) if (fork!=0)

Fork()

Else {

fork()

Fork()

}

8) exec("/bin/mv","bin/mv", "pr1.c", "pr2.c", NULL) რას გააკეთებს?

სახელი გადაერქმება pr1.c დაერქმება pr2.c

9) წარმოქმნილი ნაკადის (thread) ძირითად ნაკადთან მიერთებისას გამოიყენება ფუნქცია..

pthread_join

10) კავშირის არხს რომლის დროსაც მონაცემების გადაცემა შესაძლებელია მხოლოდ ერთი მიმართულებით ეწოდება....

სიმპლექსური

11) კრიტიკული სექციის მკაცრი მიმდევრობის ალგორითმის მიხედვით

ეს არის პროგრამის ის ნაწილი რომელიც გულისხმობს მეხსიერების საერთო ნაწილზე წვდომას.

12) რას ნიშნავს ოპერაციული სისტემა ორგანიზებულია ჰიბრიდულ არქიტექტურაში?

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 იყოს გამოყენებული ბირთვის მიერ ექსპორტირებული ფუნქცია.

13) განსაკუთრებული სიტვაცია რას ეწოდება ?

ეს არის პროგრამის მიერ ბრძანების შესრულების მცდელობის შედეგად წარმოშობილი მოვლენა, რომელიც გარკვეული მიზეზების გამო შეუძლებელია შესრულდეს.

14) რას ეწოდება ფიზიკური მისამართების სივრცე

15) ჩამოაყალიბეთ ურთიერთბლოკირების აუცილებლი და საკმარისი პირობებიდან გაუნაწილებლობის პირობა

განუაწილებლობის პირობა(ურთიერთბლოკირებაშ) პროცესისათვის მის მიერ დაკავებული რესურსის იძულებითი ჩამორთმევა შეუძლებელია. პროცესმა დაკავებული რესურსი უნდა გამოანთავისუფლოს საკუთარი სურვილით;

16) ჩამოთვალეთ ფაილის ორგანიზების მეთოდები

მიმდევრობითი შესანახ მოწყობილობაზე ჩანაწერის განთავსება ხორციელდება მათი ფიზიკური მიმდევრობით, ინდექსირებული მიმდევრობა - შესანახ მოწყობილობაზე ჩანაწერები თავსდება ლოგიკური მიმდევრობით, გასაღების გამოყენებით, რომელიც ინახება თითოეულ ჩანაწერში.

17) ფურცლისებრი მეხსიერებას რას ნიშნავს ?

თუ მონაცემთა ბლოკებს გააჩნიათ ფიქსირებული მოცულობა, მაშინ შესაბამის ბლოკებს ეწოდებათ მეხსიერების ფურცლები , ხოლო სისტემას მეხსიერების ასეთი წესით ორგანიზაციით კი სისტემა მეხსიერების ფურცლისებრი ორგანიზაციით

18) exec(“/bin/rm”, “bin/rm”, “pr1.c”, “pr2.c”, NULL) რას გააკეთებს?

წამლას

19) pipe-s შექმნა... (რა იგულისხმება ვერ მივხვდი)

20) კრიტიკული სექციის წყვეტის ალგორითმის მიხედვით...

21) დემონი,,,

ფონურ რეჟიმში მომუშავე პროგრამები

22) რა ბრძანება გამოიყენება დირექტორიის შესაქმნელად?

mkdir

23) ფაილური სისტემის რესურსები,,,,,

24) pthread_self()

ნაკადის იდენტიფიკატორის მიღება

25) კრიტიკული სექციის განმარტება

ეს არის პროგრამის ის ნაწილი რომელიც გულისხმობს მეხსიერების საერთო ნაწილზე წვდომას.

26) ერთდონიანი ფაილი რას ნიშნავს?

ფაილური სისტემის ყველაზე მარტივ რეალიზაციას წარმოადგენს ერთდონიანი ანუ ბრტყელი ფაილური სისტემა. ასეთ სისტემებში ყველა ფაილი ინახება ერთ დირექტორიაში.

27) პრიორიტეტების ინვერსია რას ნიშნავს?

როდესაც დაბალპრიორიტეტულ პროცესს დაკავებული აქვს რესურსი, რომელსაც საჭიროებს მაღალპრიორიტეტული პროცესი. თუ ეს რესურსი არ არის განაწილებადი, მაშინ დამგეგმავმა უნდა შესთავაზოს დაბალპრიორიტეტულ პროცესს მის დასასრულებლად საჭირო რესურსები, რათა მან გამოანთავისუფლოს მაღალპრიორიტეტული პროცესისათვის საჭირო რესურსი. ამ მიდგომას ეწოდება პრიორიტეტების ინვერსია

28) მრავალდონიანი არქიტექტურა რას ეწოდება?

მონოლიტური მიდგომის განზოგადებას წარმოადგენს ოპერაციული სისტემის ორგანიზება დონეების იერარქიის სახით, რომელშიც თითოეული დონეზე თავმოყრილია სისტემაში მსგავსი ფუნქციების განმახორციელებელი კომპონენტები. დონეები ერთიმეორისაგან მაღავენ მათ მიერ განსახორციელებელი ამოცანის გადაწყვეტის მექანიზმებს, მაგრამ მეზობელი დონეებს სთავაზობენ მომსახურე-ობას.

29) ურთიერთბლოკირებები აუცილებელი და საკმარისი პირობა

ურთიერთბლოკირების წარმოქმნისათვის აუცილებელია და საკმარისი შემდეგი 4 პირობის შესრულება:

1. ურთიერთგამორიცხვის პირობა: დროის ნებისმიერ მომენტში რესურსი ან გამოყოფილია მხოლოდ ერთი პროცესისათვის ან თავისუფალია;
2. რესურსის ლოდინის პირობა -პროცესს, რომელსაც დაკავებული აქვს გარკვეული რესურსი შეუძლია მოითხოვოს ახალი რესურსი;
3. გაუნაწილებლობის პირობა: პროცესისათვის მის მიერ დაკავებული რესურსის იძულებითი ჩამორთმევა შეუძლებელია. პროცესმა დაკავებული რესურსი უნდა გამოანთავისუფლოს საკუთარი სურვილით;
4. ციკლური ლოდინის პირობა: უნდა არსებობდეს ორი ან მეტი პროცესისაგან შემდგარი წრიული მიმდევრობა, რომელშიც მიმდევრობის ყოველი წევრი ელოდება მის შემდეგ მდგომი წევრის მიერ დაკავებული რესურსის გამონთავისუფლებას.

30) ბლოკირებიდან რომელ მდგომარეობაში გადადის?

მზადყოფნის რეჟიმში

31) pthread_self() რას აბრუნებს

ნაკადის იდენტიფიკატორის მიღება

32) pwd რას შვება?

ბრძანებას გამოაქვს მიმდინარე დირექტორიის სახელი

33) ფაილის მიმდევრობითი....

34) ვირტუალურიდან ფიზიკურზე გადასვლის ეტაპები

35) მონოლითური რას ნიშნავს?

მთლიანი სისტემა მუშაობს როგორც ერთი პროგრამა, ანუ ოპერაციული სისტემის კოდი დაწერილია როგორც პროცედურათა ნაკრები, რომელიც ერთ დიდ შესრულებად პროგრამაში არის გაერთიანებული.

36) chown.root() რას აკეთებს?

chown ბრძანების მეშვეობით ხდება ფაილის მფლობელის ჯგუფის შეცვლა

37) სისტემაში წარმოქმნილ ყოველ პროცესს მდგომარეობათა დიაგრამაზე გადაადგილება
შეუძლია შემდეგი მიმდევრობით

D) არცერთი პასუხი არაა სწორი :D

38) cat >file.txt ბრძანების შესრულების შედეგად

შეიქმნება file.txt ფაილი და გაავდივართ მიდი რედაქტირების რეჟიმზე

39) ოპერაციული სისტემა აგებულია მონოლიტური არქიტექტურით ნიშნავს
რომ ის ორგანიზაბულია როგორც ერთი მთლიანი პროგრამა

40) შემდეგი ბრძანების შესრულების შედეგად

chown root file.txt root დირექტორიის სახელია.

ა) ფაილს შეეცვლება მომხმარებლის ჯგუფი

ბ) ფაილს შეცვლილება მომხმარებელი

გ) ფაილს შეიცვლება მომხმარებლის ჯგუფის დაშვების უფლებები ერთ ან რამდენიმე

დ) არცერთი არაა სწორი

41) კავშირის არხს, რომლის დროსაც მონაცმების გადაცემა შესაძლებელია ორივე

მიმართულებით მაგრამ რიგრიგობით ეწოდება

ნახევრადდუპლექსური

42) პროცესის მიერ რესურსებზე გაკეთებულ მოთხოვნათა მიმდევრობას ეწოდება საიმედო თუ

ა) მოთხოვნათა მიმდევრობის წევრების ყველა შესაძლო გადაჯგუფებიდან არსებობს ერთი მაინც
ისეთი გადაჯგუფება რომლის დროსაც ყველა მოთხოვნის დაკმაყოფილება შეიძლება

ბ) მოთხოვნათა მიმდევრობის დაკმაყოფილება შეიძლება რესურსებზე გაკეთებული
მოთხოვნების მკაცრი მიმდევროვბით

გ)

დ) არცერთი არაა სწორი

43) ბანკირის ალგორითმი

ეს ალგორითმი ცნობილია ბანკირის ალგორითმის სახელით და პროცესებისათვის
რესურსების გამოყოფა ხორციელდება იმის მიხედვით, რესურსის გამოყოფის შემდეგ
იქნება თუ არა შენარჩუნებული საიმედო მდგომარეობა.

44) მიმდინარე დირექტორიის ან სხვა ნებისმიერი დირექტორიის შიგთავსის

დასათვალიერებლად გამოიყენება ბრძანება

ls

45) 0456

r rx rw

RR ალგორითმი// 95 შესრულების დრო

P1 P2 P3 P4 P5

3 6 12 5 4

5 2 0 8 1

პროცესებზე:

p1 <- R1 ->R2

p2 ->R1 R3

p3 <- R3 ->R5

p4 <- R2 ->R4 R6

p5 <- R5 ->R3 R4 R5

p6 <- R4 R6 ->R7

p7 <- R7 ->R5

1. ოპერაციული სისტემების არქიტექტურები

მონოლიტური. მთლიანი სისტემა მუშაობს როგორც ერთი პროგრამა, ანუ ოპერაციული სისტემის კოდი დაწერილია როგორც პროცედურათა ნაკრები, რომელიც ერთ დიდ შესრულებად პროგრამაში არის გაერთიანებული. ამ ტექნოლოგით ნებისმიერ პროცედურას შეუძლია გამოიძახოს სხვა პროცედურა, რომელმაც შესაძლებელია მისთვის შეასრულოს სასარგებლო სამუშაო. ვინაიდან პროცედურებს გააჩნიათ განუსაზღვრელი წვდომა ერთიმეორეზე და აპარატურაზე სისტემა მთლიანობაში შეიძლება აღმოჩნდეს არამდგრადი შეცდომის ან ზიანის შემცველი კოდის მიმართ.

მიკრობირთვული. მიკრობირთვული არქიტექტურის მიხედვით ოპერაციული სისტემა იყოფა ცალკეულ მოდულებად. ამ მოდულებიდან მხოლოდ ერთი - მიკრობირთვი, მუშაობს ბირთვის რეჟიმში. ყველა მოდული ერთმანეთთან ურთიერთქმედებს მიკრობირთვის მეშვეობით, რომელიც უზრუნველყოფს პროგრამებს შორის კავშირს, პროცესორის გამოყენების დაგეგმვას, წყვეტის პირველად დამუშვებას, შეტანა/გამოტანის ოპერაციებს და მეხსიერების საბაზო მართვას.

ჰიბრიდული. 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 იყოს გამოყენებული ბირთვის მიერ ექსპორტირებული ფუნქცია.

2. პროცესები

დემონი. ფონურ რეჟიმში მომუშავე პროცესებს, რომლებიც წარმოქმნილია გარკვეული აქტიური საქმიანობის წარმოებისათვის დემონები ეწოდებათ.

ზომბი. „დასრულდა“ მდგომარეობაში მყოფ პროცესებს UNIX მსგავს ოპერაციულ სისტემაში ეწოდებათ ზომბი-პროცესები (zombie).

კონტექსტის გადართვა. პროცესორის ერთი პროცესიდან მეორეზე კორექტულად გადასართველად აუცილებელია შესრულებადი პროცესის კონტექსტის შენახვა და იმ პროცესის კონტექსტის აღდგენა, რომელზეც იქნება პროცესორი გადართული. პროცესის ქმედუნარიანობის შენახვა/აღდგენის ასეთ პროცედურას კონტექსტის გადართვა ეწოდება.

პროცესის იდენტიფიკატორი. UNIX მსგავს სისტემაში წარმოქმნილი ყოველი პროცესი უნიკალურ სახელად ღებულობს რიცხვით მნიშვნელობას - საიდენტიფიკაციო ნომერს PID (process identifier). სხვადასხვა სისტემებში პროცესის იდენტიფიცირებისათვის გამოყოფილი რიცხვითი მნიშვნელობები მერყეობს 0-დან გარკვეულ მაქსიმალურ მნიშვნელობამდე.

სისტემაში პროცესისათვის საიდენტიფიკაციო ნომრის მინიჭებისას უნიკალობის შენარჩუნების მიზნით ყოველ ახალ პროცესს სისტემა რიცხვით მნიშვნელობას ანიჭებს ზრდადი მიმდევრობით, ანუ ბოლოს წარმოქმნილი პროცესის საიდენტიფიკაციო ნომერს + 1. პროცესის მიერ დაკავებული

ნომერი მისი დასრულების შემდეგ თავისუფლდება და მისი გამოყენება შესაძლებელია ახალი პროცესისათვის. სისტემაში თუ მიღწეულია საიდენტიფიკაციო ნომრის მაქსიმალური მნიშვნელობა, მაშინ ახალი პროცესს საიდენტიფიკაციო ნომრად ენიჭება პირველივე მინიმალური თავისუფალი რიცხვითი მნიშვნელობა.

მდგომარეობათა დიაგრამა. ამბობენ, რომ პროცესი სრულდება, ანუ იმყოფება მდგომარეობაში „შესრულება“, თუ მისთვის გამოყოფილია ცენტრალური პროცესორი. პროცესი იმყოფება მდგომარეობაში „მზადყოფნა“, თუ მისთვის ცენტრალური პროცესორის გამოყოფის შემთხვევაში პროცესს შეუძლია გააგრძელოს შესრულება. პროცესი იმყოფება მდგომარეობაში „ბლოკირება“, თუ მას არ შეუძლია შესრულების გაგრძელება გარკვეული მოვლენის დადგომამდე.

3. დაგეგმვის ალგორითმები

FCFS. ამ ალგორითმის გამოყენებისას სისტემაში იქმნება მდგომარეობაში მზადყოფნა მყოფი პროცესების მიმდევრობა. სისტემაში წარმოქმნილი ყოველი ახალი პროცესი ამ მიმდევრობაში ემატება ბოლოდან. შესრულებას იწყებს მიმდევრობის თავში მყოფი პირველივე პროცესი. პროცესს პროცესორი გამოეყოფა იმ დროითი შუალედით რამდენიც საჭიროა მის შესასრულებლად. პროცესი პროცესორს გამოანთავისუფლებს მხოლოდ იმ შემთხვევაში, თუ მან დასრულდა ან გარკვეული მიზეზით მოხდა მისი ბლოკირება. მას შემდეგ რაც სისტემაში დაფიქსირდება მოვლენა, რომელსაც ბლოკირებული პროცესი ელოდებოდა ის გადადის მდგომარეობაში მზადყოფნა და მიმდევრობას ემატება ბოლოდან.

RR. სისტემაში ყოველი წარმოქმნილ პროცესი მდგომარეობაში მზადყოფნა მყოფი პროცესების მიმდევრობას ემატება ბოლოდან. მიმდევრობის ელემენტებისათვის პროცესორის გამოყოფა ხორციელდება გარკვეული დროითი შუალედებით (კვანტით), რომლის ამოწურვის შემდეგ, თუ დრო პროცესის დასასრულებლად არ აღმოჩნდა საკმარისი, მაშინ ის ჩამოერთმევა მიმდინარე პროცესს, პროცესი გადადის მიმდევრობის ბოლოში (თავიდან იკავებს რიგს) და პროცესორი შესასრულებლად გადაეცემა მიმდევრობის შემდეგ წევრს. თუ დროითი კვანტი საკმარისი აღმოჩნდა პროცესის დასასრულებლად, მაშინ პროცესორი გადაეცემა მიმდევრობის შემდეგ წევრს. პროცესორის გამოყოფა ციკლურად ხორციელდება მანამ მიმდევრობა შეიცავს ერთ მაინც წევრს.

SJF. ამ ალგორითმის გამოყენება გულისხმობს დამგეგმავის მიერ შესასრულებლად პირველი არჩეული იყოს მოკლე (ნაკლები პროცესორული დროის საჭიროების მქონე) ამოცანა. ამ ალგორითმშიც გამოიყენება ერთი მიმდევრობა. მიმდევრობა დალაგებულია ზრდადობით შესასრულებლად საჭირო დროითი შუალედის მიხედვით.

SJF ალგორითმში პრიორიტეტის დამატებით მიიღება განსხვავებული ალგორითმი. იგულისხმება, რომ პროცესების შესრულებისათვის საჭირო დრო წინასწარაა ცნობილი. სისტემაში ყოველი წარმოქმნილი პროცესის შესრულებისათვის საჭირო დრო ედრება მიმდინარე პროცესის დასრულდებისათვის საჭირო დროს. თუ ეს უკანასკნელი მეტია წარმოქმნილი პროცესის შესრულების დროზე, მაშინ მიმდინარე პროცესი წყვეტს შესრულებას და მის ადგილს იკავებს წარმოქმნილი პროცესი. თუკი წარმოქმნილი პროცესის შესრულების დრო მეტია მიმდინარე

პორცესის დასრულებისათვის საჭირო დროზე, მაშინ წარმოქმნილი პროცესი ემატება მდგომარეობაში მზადყოფნა მყოფი პროცესების მიმდევრობას და იქ იკავებს შესაბამის ადგილს (პრიორიტეტის მიხედვით). ასეთნაირად რეალიზებული SJF ალგორითმი იძლევა სისტემაში წარმოქმნილი მოკლე დროის საჭიროების მქონე პროცესების დროული შესრულების შესაძლებლობას.

4. კავშირის არხები

pipe. კავშირის არხებით პროცესებს შორის ინფორმაციის გადაცემის საკმაოდ მარტივ მეთოდს წარმოადგენს მონაცემების გადაცემა **pipe**-ით (არხი, მილი). წარმოვიდგინოთ, რომ გამოთვლით სისტემაში გვაქვს გარკვეული მილი, რომლის ერთი ბოლოდან პროცესებს შეუძლიათ ინფორმაციის „გადაღვრა“, ხოლო მეორედან კი შემოსული ნაკადის მიღება. ასეთი მეთოდი ახდენს ნაკადების შეტანა/გამოტანის მოდელის რეალიზებას. ოპერაციულ სისტემაში მილის განთავსების შესახებ ინფორმაციას ფლობს მხოლოდ მისი წარმომქმნელი პროცესი. ეს ინფორმაცია მის მიერ შეიძლება გაზიარებული იყოს მხოლოდ მემკვიდრე პროცესებთან, ანუ მშობელი - შვილი დამოკიდებულებით დაკავშირებულ პროცესებთან. შესაბამისად, კავშირისათვის **pipe**-ის გამოყენება შეუძლიათ მხოლოდ იმ პროცესებს, რომელთაც ყავთ კავშირის მოცემული არხის წარმოქმნელი “წინაპარი”.

fifo. თუ **pipe**-ის შემქმნელ პროცესს ექნებოდა სისტემაში სხვა პროცესებისათვის **pipe**-ის გაზიარების შესაძლებლობა, ანუ შექმნილი **pipe**-ის ხილულად გადაქცევის შესაძლებლობა, მაგალითად, ოპერაციულ სისტემაში მისი გარკვეული სახელით დარეგისტრირების გზით, მივიღებდით ობიექტს, რომელსაც FIFO ეწოდება. FIFO-ს გამოყენებით შესაძლებელია სისტემაში ნებისმიერ პროცესებს შორის კავშირის ორგანიზება.

კავშირის მიმართულება. კავშირი ერთმიმართულებიანია, თუ მასთან ასოცირებულ პროცესს კავშირის საშუალება შეუძლია გამოიყენოს მხოლოდ ერთი მიმართულებით ან ინფორმაციის მისაღებად, ან მის გადასაცემად. ორმიმართულებიანკავშირში ყოველ პროცესს, რომელიც მონაწილეობს ურთიერთქმედებაში, შეუძლია გამოიყენოს კავშირი მონაცემების მისაღებად და მის გადასაცემად. კომუნიკაციურ სისტემებში ერთმიმართულებიან კავშირს ეწოდება სიმპლექსური, ორ მიმართულებიანს კი მონაცემების სხვადასხვა მიმართულებით მიმდევრობით გადაცემის შესაძლებლობით - ნახევრად-დუპლექსური, ხოლო მონაცემების ერთდროული გადაცემის შესაძლებლობით კი დუპლექსური.

5. შეჯიბრის მდგომარეობა, კრიტიკული სექცია, ალგორითმები

შეჯიბრის მდგომარეობა. პროცესები შესაძლებელია შეთანხმებულად იყენებდნენ ოპერაციულ სისტემაში არსებულ მონაცემთა საერთო საცავს. ეს საცავი შეიძლება იყოს განთავსებული ოპერატიულ მეხსიერებაში ან წარმოდგენილი იყოს რაიმე ფაილის სახით.

კრიტიკული სექცია. პროგრამის იმ ნაწილს, რომელშიც გამოიყენება საერთო მეხსიერებაზე წვდომა კრიტიკული სექცია ეწოდება.

პროცესის მიერ შესრულებული სამუშაო, რომ იყოს მისაღები საჭიროა შემდეგი პირობების შესრულება:

1. ორ პროცესს არ შეუძლია ერთდროულად იმყოფებოდეს საკუთარ კრიტიკულ სექციაში;
2. არ უნდა არსებობდეს არანაირი წინასწარი მოსაზრება პროცესორების სავარაუდო რაოდენობაზე;
3. არცერთი პროცესი, რომელიც სრულდება საკუთარი კრიტიკული სექციის გარეთ არ შეიძლება იყოს ბლოკირებული სხვა პროცესის მიერ;
4. პროცესები უსასრულოდ არ უნდა ელოდებოდნენ საკუთარ კრიტიკულ სექციაში შესვლას.

წყვეტის აკრძალვა. წყვეტის აკრძალვის შემთხვევაში შეუძლებელი იქნება ტაიმერიდან წყვეტის განხორციელება და შესაბამისად პროცესორიც ვერ შეძლებს სხვა პროცესზე გადართვას. ამ შემთხვევაში არ არსებობს საფრთხე რომელიმე პროცესმა განახორციელოს საკუთარ კრიტიკულ სექციაში შესვლა, მაგრამ შეიძლება წაარმოიშვას სხვა ტიპის პრობლემა.

მრავალპროცესორული სისტემის შემთხვევაში წყვეტის აკრძალვა მოქმედებს მხოლოდ იმ პროცესორზე, რომელსაც შეეხება ბლოკირების ინსტრუქცია. სხვა დანარჩენი პროცესორები ჩვეულ რეჟიმში გააგრძელებენ მუშაობას. მიუხედავად იმისა, რომ მრავალპროცესორულ სისტემაში შესაძლებელია მეთოდმა გაამართლოს მაინც დაუშვებელია პროცესისათვის წყვეტის აკრძალვის შესაძლებლობის მიცემა.

ცვლადი-ბოქსლომი. ამ მეთოდის გამოყენება გულისხმობს სისტემაში პროცესებისათვის საერთო ცვლადის (ცვლადი-ბოქსლომი) შემოღებას, რომელიც გააკონტროლებს პროცესების შესვლას საკუთარ კრიტიკულ სექციაში. ცვლადი-ბოქსლომის საწყისი ინიციალიზაცია ხდება 0-ვანი მნიშვნელობით, რაც პროცესებისათვის აღნიშნავს, რომ საკუთარ კრიტიკულ სექციაში არ იმყოფება არცერთი პროცესი და ნებისმიერ მათგანს სურვილის შემთხვევაში შეუძლია შევიდეს იქ. პროცესი საკუთარ კრიტიკულ სექციაში შესვლის შემდეგ ცვლის ცვლადი- ბოქსლომის მნიშვნელობას 1-ით, რაც იმის მანიშნებელია, რომ ურთიერთქმედი პროცესებიდან ერთერთი იმყოფება საკუთარ კრიტიკულ სექციაში და, თუ რომელიმე პროცესი საჭიროებს საკუთარ კრიტიკულ სექციაში შესვლას, მაშინ მას მოუწევს დალოდება სანამ მიმდინარე პროცესი არ გამოვა იქიდან. პროცესი საკუთარი კრიტიკული სექციიდან გამოსვლის შემდეგ ცვლად-ბოქსლომს უბრუნებს საწყის მნიშვნელობას (0-ს).

მკაცრი მიმდევრობა. კრიტიკული სექციის პრობლემის გადაწყვეტის შემდეგი მეთოდი გულისხმობს პროცესების წარმოდგენას მკაცრი მიმდევრობის სახით. პროცესები მიმდევრობით შედიან საკუთარ კრიტიკულ სექციაში. იქიდან გამოსვლის შემდეგ მიმდევრობის შემდეგ წევრს ეძლევა საკუთარ კრიტიკულ სექციაში შესვლის შესაძლებლობა.

6. ურთიერთბლოკირება, ურთიერთბლოკირების წარმოქმნის აუცილებელი და საკმარისი პირობა

ურთიერთბლოკირება. ხშირად პროცესები საკუთარი სამუშაოს შესასრულებლად საჭიროებენ ერთზე მეტ რესურსს. მაგალითად, განვიხილოთ სიტუაცია, რომლის დროსაც სისტემაში გვაქვს ორი პროცესი, P1, P2, და ორი რესურსი: სკანერი და დისკური მოწყობოლიბა. დავუშვათ, რომ ორივე პროცესი საჭიროებს თითოეულ რესურსს და რესურსების გამოყოფა ხორციელდება პროცესებისათვის თითოთითოდ. დავუთვათ, P1 პროცესი დაპროგრამირებულია ისე, რომ მან პირველი გამოიყენოს სკანერი და მეორე დისკური მოწყობილობა, ხოლო P2 პროცესი კი პირიქით. როგორც წესი, პირველი P1 პროცესს გამოეყოფა სკანერი, ხოლო P2 -ს დისკური მოწყობილობა. P1 პროცესი სკანერთან მუშაობის დასრულების და მისგან მონაცემების მიღების შემდეგ საჭიროებს მეორე რესურსს და აკეთებს შესაბამის მოთხოვნას რესურსის გამოყოფაზე. მაგრამ მისი დაკმაყოფილება შეუძლებელია, ვინაიდან დისკურ მოწყობილობას იკავებს P2 პროცესი. ანალოგურად, P2 პროცესი დისკურ მოწყობილობასთან მუშაობის დასრულების შემდეგ გააკეთებს მოთხოვნას სკანერზე და მისი დაკმაყოფილებაც ასევე შეუძლებელია. ასეთ შემთხვევაში ორივე პროცესი იქნება ბლოკირებული სანამ არ გამონთავისუფლდება მათ მიერ მოთხოვნილი რესურსი. წარმოქმნილ სიტუაციას ეწოდება ურთიერთბლოკირება (deadlock) ან ჩიხი.

ურთიერთბლოკირების წარმოქმნის აუცილებელი და საკმარისი პირობები. ურთიერთბლოკირების წარმოქმნისათვის აუცილებელია და საკმარისი შემდეგი 4 პირობის შესრულება:

1. **ურთიერთგამორიცხვის პირობა (mutual exclusion).** დროის ნებისმიერ მომენტში რესურსი ან გამოყოფილია მხოლოდ ერთი პროცესისათვის ან თავისუფალია;
2. **რესურსის ლოდინის პირობა (hold and wait).** პროცესს, რომელსაც დაკავებული აქვს გარკვეული რესურსი შეუძლია მოითხოვოს ახალი რესურსი;
3. **გაუნაწილებლობის პირობა (no preemption).** პროცესისათვის მის მიერ დაკავებული რესურსის იძულებითი ჩამორთმევა შეუძლებელია. პროცესმა დაკავებული რესურსი უნდა გამოანთავისუფლოს საკუთარი სურვილით;
4. **ციკლური ლოდინის პირობა (circular wait).** უნდა არსებობდეს ორი ან მეტი პროცესისაგან შემდგარი წრიული მიმდევრობა, რომელშიც მიმდევრობის ყოველი წევრი ელოდება მის შემდეგ მდგომი წევრის მიერ დაკავებული რესურსის გამონთავისუფლებას.

ჩამოყალიბებული პირობებიდან ერთერთის დარღვევის შემთხვევაში სისტემაში არ გვაქვს ურთიერთბლოკირების მდგომარეობა.

7. რესურსები

რესურსი შეიძლება იყო აპარატული (მყარი დისკი, პროცესორი, მეხსიერება) ან ინფორმაციული (ფაილები, მონაცემთა ბაზა). როგორც უკვე აღვნიშნეთ ყოველი რესურსი დროის ნებისმიერ მომენტში შესაძლებელია გამოეყოს მხოლოდ ერთ პროცესს. შესაბამისად, თუ რამდენიმე პროცესი საჭიროებს ერთიდამავე რესურსს მათ ამ რესურსის მიღება შეუძლიათ რიგრიგობით. თუ ყოველი რესურსი ოპერაციულ სისტემაში წარმოდგენილი იქნებოდა რამდენიმე ასლის სახით, მაშინ ერთი

კონკრეტული რესურსის მომთხოვნი რამდენიმე პროცესის დაკმაყოფილება იქნებოდა შესაძლებელი.

განაწილებადი. პროცესის მიერ დაკავებულ რესურსს, რომლის ჩამორთმევაც მისთვის შესაძლებელია „უმტკივნეულოდ“, მიეკუთვნება განაწილებად რესურსს. ამ ტიპის რესურსის მაგალითს წარმოადგენს მეხსიერება. სისტემაში არასაკმარის მეხსიერების არსებობის შემთხვევაში ხორციელდება პროცესის სრული ან ნაწილობრივი გადატანა მეხსიერების არიდან დისკზე, ხოლო მოგვიანებით კი მისი უკან დაბრუნება.

გაუნაწილებელი. პროცესისათვის გაუნაწილებელი რესურსის ჩამორთმევამ შესაძლებელია გამოიწვიოს პროცესის მიერ შესრულებული მნიშვნელოვანი სამუშაოს დაკარგვა. ურთიერთბლოკირებაში შესაძლებელია მონაწილეობდეს როგორც განაწილებადი, ისე გაუნაწილებელი რესურსი. რესურსების საგულდაგულო გადანაწილების ხარჯზე შესაძლებელია განაწილებადი რესურსებით გამოწვეული ურთიერთბლოკირების აცილება. ამიტომ ყურადღებას გავამახვილებთ მხოლოდ გაუნაწილებელი რესურსებით გამოწვეულ ურთიერთბლოკირებასთან გამკვლავების მეთოდებზე.

საზოგადოდ, სისტემაში რესურსების გამოყენება ხორციელდება მოვლენათა შემდეგი მიმდევრობით:

1. მოთხოვნა;
2. გამოყენება;
3. გამონთავისუფლება.

8. მეხსიერების ორგანიზაცია

იერარქიის თავში განთავსებულია ყველაზე სწრაფქმედი მეხსიერება, ნაკლები მოცულობითა და ინფორმაციის შესანახ ბიტზე მაღალი ღირებულებით, ხოლო იერარქიის ბოლო დონეზე განთავსებულია ნელი, იაფი ღირებულების და დიდი მოცულობის მეორადი მეხსიერების მოწყობილობები.

რეგისტრები - ეს არის სისტემაში გამოყენებული ყველაზე ძვირი და სწრაფქმედი მეხსიერება. ისინი დამზადებული იმავე ტექნოლოგიით რაც პროცესორები და სისწრაფითაც არ ჩამოუვარდებიან მას. შემდეგ დონეზე განთავსებულია ქეშ-მეხსიერება. მისი სისწრაფე იზომება დაყოვნებებით - დროითი შუალედით, რომელიც საჭიროა მონაცემების გადასაცემად. როგორც წესი, დაყოვნებები იზომება ნანოწამებში ან პროცესორის ციკლით.

იერაქიის შემდეგ დონეზე განთავსებულია ოპერატიული (ძირითადი, ფიზიკური) მეხსიერება. ოპერატიული მეხსიერება პროცესორისათვის მონაცემების გადასაცემად იყენებს პროცესორის სისწრაფესთან შედარებით დაბალი სისწრაფის სისტემურ სალტეს (system bus) და შესაბამისად დაყოვნება ამ შემთხვევაში მაღალია.

მეხსიერების შემდეგ დონეებზე განთავსებულია მეხსიერების მეორადი (გარე) მოწყობილობები. ელექტრული დისკები, მაგნიტური დისკები და მაგნიტური ლენტა წარმოადგენს გამოთვლით სისტემაში შენახვის დიდი მოცულობის და დაბალი ღირებულების მქონე მოწყობილობებს. ამ ტიპის მოწყობილობებში ინფორმაციაზე წვდომა ქეშ-მეხსიერებასთან შედარებით ხორციელდება რამდენიმე ასეულ ათასჯერ ნაკლები სისწრაფით. შენახვის მეორადი მოწყობილობების უპირატესობა მდგომარეობს იმაში, რომ ისინი არ არიან დამოკიდებული ელექტროენერგიაზე და უეცარი გამორთვის შემთხვევაში მათზე არსებული ინფორმაცია არ იკარგება. მეორე უპირატესობას წარმოადგენს ის, რომ მათ მონაცემების შესანახად გააჩნიათ დიდი მოცულობა.

9. მისამართების დაკავშირება

ლოგიკური მისამართების სივრცის მაქსიმალური მოცულობა განისაზღვრება პროცესორის თანრიგიანობით და გაცილებით აჭარბებს თანამედროვე სისტემებში ფიზიკური მისამართების სივრცის მოცულობას. შესაბამისად პროცესორს და ოპერაციულ სისტემას უნდა შეეძლოს პროგრამის კოდში მიმართვების (მისამართების) ასახვა რეალურ ფიზიკურ მისამართებზე, რომელიც შესაბამება პროგრამის მიმდინარე მდებარეობას ძირითად მეხსიერებაში. მისამართების ასეთ ასახვას ეწოდება მისამართების ტრანსლირება ან მისამართების დაკავშირება.

ინსტრუქციებისა და მონაცემების დაკავშირება მეხსიერებასთან შესაძლებელია გაკეთდეს შემდეგ ნაბიჯებში:

- **კომპილაციის ეტაპზე** - როდესაც ცნობილია მეხსიერებაში პროცესის ზუსტიადგილმდებარეობა, მაშინ უშუალოდ გენერირდება ფიზიკური მისამართი. პროგრამისსაწყისი მისამართის შეცვლისას საჭიროა მისი კოდის ხელახალი კომპილირება.
- **ჩატვირთვის ეტაპი** - კომპილაციის ეტაპზე თუ უცნობია ინფორმაცია პროგრამის განთავსებაზე, კომპილატორი აგენერირებს გადატანად კოდს. ამ შემთხვევაში საბოლოო დაკავშირების გადადება ხდება ჩატვირთვის მომენტამდე. თუ საწყისი მისამართი იცვლება, საჭიროა მხოლოდ კოდის გადატვირთვა შეცვლილი სიდიდის გათვალისწინებით.
- **შესრულების ეტაპი** - თუ პროცესი შესაძლებელია გადაადგილებული იყოს შესრულების დროს მეხსიერების ერთი მიდამოდან მეორეში, დაკავშირება გადაიდებაშესრულების ეტაპამდე. აქ სასურველია სპეციალიზირებული მოწყობილობების, მაგალითად, გადაადგილების რეგისტრების, არსებობა. მათი მნიშვნელობა ემატებაპროცესის მიერ გენერირებულ ყოველ მისამართს. უმეტესი თანამედროვე ოპერაციულისისტემა ანხორციელებს მისამართების ტრანსლირებას შესრულების ეტაპზე, ამისათვისსპეციალური აპარატურული მექანიზმების გამოყენებით.

10. მეხსიერების მართვის სტრატეგიები (ჩატვირთვის, განთავსების, ჩანაცვლების)

მეხსიერების მართვის სტრატეგიები იქმნება ძირითადი მეხსიერების ოპტიმალური გამოყენების მიზნით. ამ სტრატეგიებს ყოფენ სამ ნაწილად:

1. ჩატვირთვის სტრატეგია;
2. განთავსების სტრატეგია;
3. ჩანაცვლების სტრატეგია.

ჩატვირთვის სტრატეგია განსაზღვრავს, თუ როდის უნდა განხორციელდეს ახალი პროგრამის მონაცემებისა და ინსტრუქციების ჩატვირთვა ოპერატიულ მეხსიერებაში. ეს სტრატეგია იყოფა ორ ნაწილად: ჩატვირთვა მოთხოვნის საფუძველზე და წინასწარი ჩატვირთვა. ბოლო პერიოდამდე გამოიყენებოდა სტრატეგია ჩატვირთვა მოთხოვნის საფუძველზე, რომელიც გულისხმობს ჩასატვირთი პროგრამის მონაცემების და ინსტრუქციების გადატანას შენახვის მეორადი მოწყობილობიდან ოპერატიულ მეხსიერებაში მას შემდეგ რაც მასზე გაკეთდება მოთხოვნა. ამ სტრატეგიისათვის უპირატესობის მინიჭების მიზეზი იყო გამოთვლით სისტემებში დიდი მოცულობის ოპერატიული მეხსიერების არარსებობა. ასეთ შემთხვევაში იპერატიულ მეხსიერებაში სხვადასხვა პროგრამების მონაცემებისა და ინსტრუქციების განთავსება გამოიწვევს მის უსარგებლოდ დაკავებას. გამოთვლით სისტემებში დიდ მოცულობით ოპერატიული მეხსიერების გამოჩენამ შესაძლებელი გახადა წინასწარი ჩატვირთვის სტრატეგიის გამოყენება, რამაც საგრძნობლად გაზარდა პროცესორის მუშაობის სისწრაფე.

ოპერატიულ მეხსიერებაში პროგრამის მონაცემების და ინსტრუქციების განთავსების აუცილებლობისას განთავსების სტრატეგია განსაზღვრავს, თუ სად უნდა განხორციელდეს მათი განთავსება. განთავსების სტრატეგია იყოფა სამ ნაწილად: პირველი შესაფერისი, საუკეთესოდ შესაფერისი და ნაკლებად შესაფერისი. პირველი შესაფერისი სტრატეგიის მიხედვით ახალი პროგრამის მონაცემებისა და ინსტრუქციების განთავსება ხორციელდება მოცულობით შესაფერის პირველივე დანაყოფში. საუკეთესოდ შესაფერისი სტრატეგიის მიხედვით პროგრამის მონაცემებისა და ინსტრუქციების განთავსება ხორციელდება ძირითადი მეხსიერების ისეთ დანაყოფში, რომელშიც მათი განთავსების შემდეგ რჩება მცირემოცულობის ადგილი. ნაკლებად შესაფერისი სტრატეგიის შემთხვევაში პროგრამის მონაცემებისა და ინსტრუქციების განთავსება შესაძლებელია განხორციელდეს ძირითადი მეხსიერების ისეთ დანაყოფში, რომელშიც საკმარისი ადგილი იქნება ახალი პროგრამის მონაცემებისა და ინსტრუქციების განსათავსებლად. ოპერატიულ მეხსიერებაში მონაცემების განთავსებამდე ხორციელდება მისი დაყოფა გარკვეული მოცულობის ბლოკებად. თუ ბლოკები ფიქსირებული მოცულობისაა, მაშინ მათ ფურცლები ეწოდებათ. წინააღმდეგ შემთხვევაში მათ სეგმენტებს უწოდებენ.

თუ ძირითად მეხსიერებაში ახალი პროგრამის მონაცემების და ინსტრუქციების განსათავსებლად საკმარისი ადგილი არაა, მაშინ იქიდან უნდა მოხდეს გარკვეული პროგრამის მონაცემების და ინსტრუქციების წაშლა. თუ რომელი მონაცემები და ინსტრუქციები უნდა წაიშალოს ოპერატიული მეხსიერებიდან განსაზღვრავს ჩანაცვლების სტრატეგია.

11. მეხსიერების დაყოფა (ფურცლისებრი, სეგმენტური, ფურცლისებრ-სეგმენტური)

ფურცლისებრი. თუ მონაცემთა ბლოკებს გააჩნიათ ფიქსირებული მოცულობა, მაშინ შესაბამის ბლოკებს ეწოდებათ მეხსიერების ფურცლები (pages), ხოლო სისტემას მეხსიერების ასეთი წესით ორგანიზაციით კი სისტემა მეხსიერების ფურცლისებრი ორგანიზაციით (paging).

სეგმენტური. თუ მონაცემთა ბლოკებს გააჩნიათ ცვლადი მოცულობა, მაშინ შესაბამის ბლოკებს ეწოდებათ სეგმენტები (segment), ხოლო სისტემას მეხსიერების ასეთი წესით ორგანიზაციით კი სისტემა მეხსიერების სეგმენტური ორგანიზაციით (segmentation).

ფურცლისებრ-სეგმენტური. ზოგიერთ სისტემებში შეჯერებულია მეხსიერების ფურცლისებრი და სეგმენტური მიდგომა. ასეთ სისტემებში მთლიანი მეხსიერება დაყოფილია სეგმენტებად, ხოლო სეგმენტები კი თავის შიგნით ფიქსირებული მოცულობის მქონე ფურცლებად და შესაბამის სისტემას ეწოდება სისტემა მეხსიერების სეგმენტურ-ფურცლისებრი ორგანიზაციით.

12. ფაილური სისტემა (ფაილი, დირექტორია, ფაილური დესკრიპტორი, ერთ- და მრავალდონიანი ფაილური სისტემა, ლინკი)

ფაილები. ფაილი წარმოადგენს მონაცემთა ნაკრებს, რომლებზეც შესაძლებელია გახორციელდეს შემდეგი ოპერაციები:

- შექმნა (create file) - ახალი ფაილის შექმნა;
- გახსნა (open file) - ფაილის გამზადება მასზე დასაშვები მოქმედებების (კითხვა ან ჩაწერა) განსახორციელებლად;
- დახურვა (close file) - ფაილის შემდგომ გახსნამდე მასზე დასაშვები მოქმედებების ბლოკირება;
- განადგურება (destroy file) - ფაილის წაშლა;
- კოპირება (copy file) - ფაილის შიგთავსის ასლის გადატანა სხვა ფაილში;
- ასახვა (list file) - ფაილის შიგთავსის მონიტორის ეკრანზე ან ფურცელზე გამოტანა.

მონაცემთა ცალკეული ელემენტებზე, რომლებიც შენახულია ფაილებში, შესაძლებელია შემდეგი ოპერაციების განხორციელება:

- კითხვა (read data) - ფაილიდან მონაცემების კითხვა პროცესის მეხსიერებაში;
- ჩაწერა (write data) - პროცესის მეხსიერებიდან მონაცემების ჩაწერა ფაილში;
- განახლება (update) - ფაილში არსებული მონაცემთა ელემენტების რაოდენობის შეცვლა;
- ჩასმა (insert data) - ფაილში ახალი ელემენტების ჩამატება;
- წაშლა (delete data) - ფაილიდან მონაცემთა ელემენტების წაშლა.

ფაილები ხასიათდებიან შემდეგი მახასიათებლებით:

- მოცულობა (size of file) - ფაილში შენახული მონაცემების მოცულობა;

- ადგილმდებარეობა (location of file) - ფაილის ადგილმდებარეობა შესანახ მოწყობილობაზე ან ფაილური სისტემაში;
- დაშვების უფლებები (accessibility of file) - ფაილზე წვდომის უფლებები;
- ტიპი (type) - ფაილის ტიპი. მაგალითად შესრულებადი ფაილი პროცესისათვის შეიცავს შესასრულებელ ინსტრუქციებს;
- შეცვლადობა (volatility of file) - ფაილში შენახულ მონაცემებში ცვლილების შეტანის სიხშირე;
- აქტიურობა (activity of file) - დროის მოცემულ შუალედში ფაილის ჩანაწერებზე მიმართვის სიხშირე.

ფაილი შეიძლება შედგებოდეს ერთი ან რამდენიმე ჩანაწერისაგან. ფიზიკური ჩანაწერი ან ფიზიკური ბლოკი ეს არის შესანახ მოწყობილობაზე ჩაწერილი ან მისგან წაკითხული ინფორმაციის ერთეული. ლოგიკური ჩანაწერი ან ლოგიკური ბლოკი ეს არის მონაცემთა ნაკრები, რომელიც პროგრამების მიერ შეიძლება გაგებული იყოს როგორც ინფორმაციის ერთეული. თუ ფიზიკური ჩანაწერი შეიცავს ზუსტად ერთ ლოგიკურ ჩანაწერს, მაშინ ამბობენ, რომ ის შედგება ღია მონაცემებისაგან (unblocked records). წინააღმდეგ შემთხვევაში ამბობენ, რომ ჩანაწერი შედგება ჩაკეტილი მონაცემებისაგან (blocked records).

დირექტორიები. ფაილურ სისტემაში ფაილების ორგანიზებული განთავსებისა და სწრაფი წვდომისათვის გამოიყენება დირექტორიები. დირექტორია ეს არის ფაილი, რომელიც შეიცავს მასში შემავალ ფაილებსა და დირექტორიებზე ჩამონათვალს და ინფორმაციას ფაილურ სისტემაში მათი ლოგიკური ადგილმდებარეობის შესახებ. ჩვეულებრივი ფაილებისაგან განსხვავებით დირექტორიები არ ინახავენ მონაცემებს.

ველები, რომლებსაც შეიძლება შეიცავდეს დირექტორია

ველი	აღწერა
სახელი	ფაილის სახელის შემცველი სიმბოლური სტრიქონი
ადგილმდებარეობა	ფაილურ სისტემაში ფაილის განთავსების ლოგიკური მისამართი (მაგალითად, სრული ან მიმართებითი სახელი)
მოცულობა	ფაილის მიერ დაკავებული ბაიტების რაოდენობა
ტიპი	ფაილის დანიშნულების აღმნიშვნელი სიმბოლო (მაგალითად, მონაცემების ფაილი ან დირექტორია)
მიმართვის დრო	ფაილზე ბოლო მიმართვის დრო
შეცვლის დრო	ფაილში განხორციელებული ბოლო ცვლილების დრო
შექმნის დრო	ფაილის შექმნის დრო

ფაილური დესკრიპტორი. ფაილის გახსნის ოპერაციის შესრულებისას ოპერაციული სისტემა ეძებს ფაილზე ინფორმაციას და დირექტორიათა სტრუქტურაში პოლობს მის ადგილმდებარეობას. ერთსადაიმავე ფაილის მრავალჯერადი გახსნის თავიდან აცილების მიზნით სისტემა ოპერატიულ მეხსიერებაში გახსნილ ფაილზე ინფორმაციას ინახავს ე.წ. ღია ფაილების ცხრილში შესაბამისი ჩანაწერის გაკეთების გზით. ფაილის გახსნის ოპერაცია აბრუნებს არაუარყოფით რიცხვით მნიშვნელობას - ფაილურ დესკრიპტორს, რომელიც წარმოადგენს ღია ფაილების ცხრილში შესაბამის ფაილზე ჩანაწერის ნომერს. ფაილზე განსახორციელებელი ყოველი შემდეგი მოქმედება ხორციელდება ფაილური დესკრიპტორის გამოყენებით.

ღია ფაილების ცხრილი ასევე შეიცავს ფაილის მართვის ველებს. ამ ველებში ინახება ფაილის სამართავად გამოყენებადი ინფორმაცია ე.წ. ფაილის ატრიბუტები. ფაილის ატრიბუტები შეიძლება განსხვავდებოდნენ სისტემიდან სისტემამდე. მაგალითად, ის შეიძლება შეიცავდეს შემდეგ ინფორმაციას:

- ფაილის სიმბოლური სახელი;
- ფაილის ადგილმდებარეობა შესანახ მოწყობილობაზე;
- საორგანიზაციო სტრუქტურა (მაგალითად, მიმდევრობითი ან ნებისმიერი დაშვების ფაილი);
- ინფორმაცია შესანახ მოწყობილობაზე;
- ინფორმაცია ფაილის ტიპზე;
- ინფორმაცია ფაილის ხასიათზე (დროებითი თუ მუდმივი);
- ფაილის შექმნის დრო და თარიღი;
- ფაილზე მიმართვების მთვლელი (მაგალითად, ფაილზე განხორციელებული კითხვის ოპერაციების რაოდენობა);

ერთდონიანი ფაილური სისტემები. ფაილური სისტემის ყველაზე მარტივ რეალიზაციას წარმოადგენს ერთდონიანი ანუ ბრტყელი ფაილური სისტემა. ასეთ სისტემებში ყველა ფაილი ინახება ერთ დირექტორიაში. დირექტორიის შიგნით ფაილების სახელები უნდა იყოს უნიკალური. ვინაიდან სხვადასხვა პროგრამების მიერ გამოყენებული ფაილების სახელები (მონაცემები განსხვავებული) შესაძლებელია იყვენენ იდენტური, ამიტომ ამ ტიპის ფაილურმა სისტემამ ვერ ჰქოვა ფართოდ გავრცელება.

იერარქიულად სტრუქტურირებული ფაილური სისტემები. საბაზო (root) დირექტორია განთავსებულია იერარქიის თავში და აღნიშნავს ფაილური სისტემის დასაწყისს.

ფაილის სახელი უნიკალური უნდა იყოს მხოლოდ მომხმარებლის დირექტორიის შიგნით. იერარქიულად სტრუქტურირებულ ფაილურ სისტემაში ყოველ დირექტორიას შეიძლება გააჩნდეს მრავალი ქვედირექტორია, მაგრამ მხოლოდ ერთი მშობელი დირექტორია. ფაილის სრულ სახელს ფაილურ სისტემაში წარმოადგენს გზა საბაზო დირექტორიიდან დანიშნულების ფაილამდე.

ლინკი (link). ლინკი არის დირექტორიაში განთავსებული ჩანაწერი, რომელიც უთითებს სხვა დირექტორიაზე ან დირექტორიაში განთავსებულ ფაილზე. მომხმარებელი სისტემაში ნავიგაციის გამარტივების მიზნით ხშირად იყენებს ლინკს. სისტემაში ლინკი შეიძლება არსებობდეს ორი ტიპის: ლოგიკური და ფიზიკური.

ლოგიკური ლინკი წარმოადგენს დირექტორიაში არსებულ ჩანაწერს, რომელიც უთითებს სხვა დირექტორიაში განთავსებულ ფაილამდე გზას. ფაილური სისტემა მითითებულ ფაილამდე გზას იკვლევს ლინკში არსებული მისამართით.

ფიზიკური ლინკი კი ეს არის დირექტორიაში არსებული ჩანაწერი, რომელიც უთითებს შესანახ მოწყობილობაზე ფაილის განთავსების ფიზიკურ მისამართს. ამ შემთხვევაში ფაილური სისტემა ფაილს პოულობს მისი ფიზიკური მისამართით.

13. ფაილის ორგანიზაციის მეთოდები (მიმდევრობითი, ინდექსირებული მიმდევრობა)

ფაილების ორგანიზაცია ეწოდება შესანახ მოწყობილობაზე მათი განთავსების წესს.

მიმდევრობითი (sequential) - შესანახ მოწყობილობაზე ჩანაწერის განთავსება ხორციელდება მათი ფიზიკური მიმდევრობით. ასეთი ორგანიზაცია დამახასიათებელია მაგნიტურ ლენტაზე შენახული ფაილებისათვის, ანუ მიმდევრობითი დაშვების მოწყობილობებისათვის;

ინდექსირებული მიმდევრობა (indexed sequential) - შესანახ მოწყობილობაზე ჩანაწერები თავსდება ლოგიკური მიმდევრობით, გასაღების გამოყენებით, რომელიც ინახება თითოეულ ჩანაწერში. სისტემაში მხარდაჭერილია ინდექსები, რომლებით უთითებენ გარკვეულ ძირითად ჩანაწერებს. ინდექსირებული მიმდევრობის ჩანაწერებზე დაშვება შესაძლებელია მათი გასაღებების მიმდევრობით, ან პირდაპირ სისტემაში შექმნილი ინდექსების მეშვეობით;

სემინარი 1: UNIX ოპერაციული სისტემა

სასწავლო კურსის მსვლელობისას სემინარული მეცადინეობის მასალა იღუსტრი-რებული იქნება UNIX ოპერაციული სისტემის ერთერთი ნაირსახეობის - Linux ოპერაციული სისტემის - მაგალითზე, თუმცა ჩვენი საუბარი არ შეეხება მხოლოდ მის თავისებურებებს.

Linux ოპერაციული სისტემის ბირთვი წარმოადგენს მონოლიტურ სისტემას. მისი ბირთვის კომპილაციისას შესაძლებელია ბირთვის მრავალი კომპონენტის - მოდულის, დინამიური ჩატვირთვა/ამოტვირთვა. ჩატვირთვის მომენტში მისი კოდი შესასრულებლად გადადის პრივილეგირებულ რეჟიმში და უკავშირდება ბირთვის დანარჩენ ნაწილებს. მოდულის შიგნით შესაძლებელია გამოყენებული იყოს ბირთვის მიერ ექსპორტირებული ნებისმიერი ფუნქცია.

1.1. სისტემური გამოძახება და ბიბლიოთეკა `libc`

როგორც უკვე აღვნიშნეთ, UNIX ოპერაციული სისტემის ძირითად და მუდმივად ფუნქციონირებად ნაწილს წარმოადგენს მისი ბირთვი. სხვა (სისტემური ან გამოყენებითი) პროგრამები ბირთვთან ურთიერთქმედებენ სისტემური გამოძახებების (**system call**) მეშვეობით, რომლებიც პროგრამებისათვის წარმოადგენენ ბირთვზე დაშვების წერტილებს. სისტემური გამოძახებები არის სპეციალურად შექმნილი პროცედურები, რომლებსაც სისტემური ან გამოყენებითი პროგრამები გარკვეული დროით გადაჰყავთ პრივილეგირებულ რეჟიმში. ამ რეჟიმში პროგრამები ღებულობენ წვდომას ისეთ რესურსებზე, რომლებიც მომხმარებლის რეჟიმში მათთვის იყო მიუწვდომელი.

სისტემური გამოძახების შესასრულებლად საჭირო მანქანური ბრძანებები განსხვავდება მანქანიდან მანქანამდე. ასევე, განსხვავდება სხვადასხვა ოპერაციულ სისტემაში გამოყენებული სისტემური გამოძახებების ნაკრებები. C ენის პროგრამისტის თვალსაზრისით სისტემური გამოძახების გამოყენება არაფრით არ განსხვავდება C ენის სტანდარტული ANSI ბიბლიოთეკის რომელიმე ფუნქციის გამოყენებისაგან, მაგალითად, როგორიცაა სტრიქონებთან სამუშაო ფუნქციები: `strlen()`, `strcpy()` და ა.შ. UNIX-ის სტანდარტული ბიბლიოთეკა `libc` უზრუნველყოფს ყოველი სისტემური გამოძახების C ინტერფეისს. ამას მივყავართ იქამდე, რომ ყოველი სისტემური გამოძახება პროგრამისტისათვის წააგავს C ენის ფუნქციას.

სისტემურ გამოძახება შეიძლება დასრულდეს წარმატებით ან წარუმატებლად. წარუმატებლად დასრულების აღსანიშნავად უმეტეს სისტემურ გამოძახებაში გამოიყენება მნიშვნელობა -1, ხოლო წარმატებული შესრულების კი - რაიმე არაუარყოფითი მნიშვნელობა. სისტემური გამოძახებები, რომლებიც აბრუნებენ მიმთითებელს, შეცდომის შემცველი სიტუაციის იდენტიფიცირებისათვის, იყენებენ მნიშვნელობას `NULL`. შეცდომის ზუსტ მიზეზებში გასარკვევად C-ინტერფეისი გვთავაზობს `<errno.h>` ფაილში განსაზღვრულ გლობალურ ცვლადს - `errno`, მის შესაძლო მნიშვნელობებთან და მათ მოკლე აღწერებთან ერთად. შევნიშნოთ, რომ `errno` ცვლადის გაანალიზება საჭიროა შეცდომის შემცველი სიტუაციის წარმოშობისთანავე, ვინაიდან წარმატებით დასრულებული სისტემური გამოძახება არ ცვლის მის მნიშვნელობას. პროგრამის შესრულებისას სტანდარტული შეცდომაზე სიმბოლური ინფორმაციის მისაღებად შესაძლებელია გამოყენებული

იყოს სტანდარტული UNIX-ფუნქცია perror().

perror() ფუნქციის პროტოტიპი

```
#include<stdio.h>
void perror(char *str);
```

ფუნქციის აღწერა

perror() ფუნქცია გამოიყენება შეცდომაზე შეტყობინების გამოსატანად, რომელიც შეესაბამება გამოტანის სტანდარტული ნაკადის შეცდომის დაფიქსირებისას errno სისტემური ცვლადის მნიშვნელობას. ფუნქცია ბეჭდავს მთლიანად str სტრიქონს (თუ str პარამეტრი არ უდრის NULL-ს), ორწერტილს, ჰარის და წარმოშობილი შეცდომის შესაბამისი შეტყობინების ტექსტს, ახალ ხაზზე გადასვლის სიმბოლოთი ('\\').

1.2. ფაილის სრული და მიმართებითი სახელები

ყველა ოპერაციული სისტემისათვის ფაილის ცნება წარმოადგენს ერთერთ მნიშვნელოვან ცნებას. UNIX-მსგავს ოპერაციულ სისტემებში ფაილის ცნების მნიშვნელობა უფრო ცხადია ვიდრე Windows-ის ოპერაციულ სისტემებში. ყველაფერი რაც ხდება UNIX-მსგავს ოპერაციულ სისტემებში რეალიზებულია ფაილის ცნებაზე დამოკიდებულებით. სანამ ფაილურ სისტემას და ფაილს დაწვრილებით განვიხილავდეთ ფაილებთან დაკავშირებული ზოგიერთი ცნების განსასაზღვრავად შეგვიძლია დავკამაყოფილდეთ ფაილზე ზოგადი ცნებით: ფაილი ეს არის მყარ დისკზე განთავსებული ერთი ტიპის მონაცემებთან დაკავშირებული მეხსიერების მისამართების ერთობლიობა (ნაკრები).

ისევე, როგორც სხვა ოპერაციულ სისტემებში, UNIX-მსგავს ოპერაციულ სისტემაში ყოველი ფაილი გაერთიანებულია ხისებრ ლოგიკური სტრუქტურაში. ფაილები შეიძლება გაერთიანდნენ დირექტორიებში ან კატალოგებში¹. არ არსებობს ფაილი, რომელიც არ შედის არცერთ დირექტორიაში. დირექტორიები თავის მხრივ შეიძლება წარმოადგენდნენ სხვა დირექტორის ქვედირექტორიებს. არსებობს დირექტორია, რომელშიც არ შედის არც ფაილი და არც ქვედირექტორია (ცარიელი დირექტორია) (ნახ. 1.1). ყველა დირექტორიას შორის არსებობს ერთადერთი დირექტორია, რომელიც არ წარმოადგენს სხვა დირექტორის ქვედირექტორიას, ასეთ დირექტორიას საბაზო ან root დირექტორია ეწოდება. UNIX ოპერაციულ სისტემაში არსებობს 5 ტიპის ფაილი. ჩვენ ძირითადად განვიხილავთ ორს: ჩვეულებრივი ანუ რეგულარული ფაილი (რომელიც შეიძლება შეიცავდეს პროგრამის მონაცემებს, შესრულებად კოდს, მონაცემებს და ა.შ.) და დირექტორია.

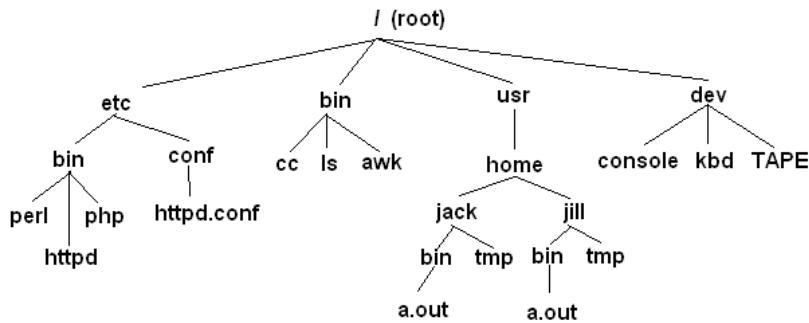
ყოველ ფაილს უნდა გააჩნდეს სახელი. სხვა ოპერაციულ სისტემების მსგავსად UNIX ოპერაციულ სისტემაში ფაილისათვის სახელის მინიჭებისას არსებობს გარკვეული შეზღუდვები. POSIX სტანდარტში UNIX ოპერაციული სისტემისათვის სისტემური გამოძრების ინტერფეისი შეიცავს მხოლოდ სამ ცხად შეზღუდვას:

- სახელის სიგრძე არ უნდა აღემატებოდეს სისტემაში გათვალისწინებულ სიგრძეს (Linux-სათვის - 255 სიმბოლო);
- არ შეიძლება NULL სიმბოლოს გამოყენება;
- არ შეიძლება '/' სიმბოლოს გამოყენება².

¹ Windows-ის ოპერაციულ სისტემაში გამოიყენება ცნება ფოლდერი

² ერთადერთ გამონაკლის წარმოადგენს საბაზო დირექტორია, რომლის სახელიცაა '/'. სწორედ ის წარმოადგენს ერთადერთ ფაილს, რომელსაც ფაილურ სისტემაში გააჩნია უნიკალური სახელი

აკრძალული სიმბოლოების რიცხვს ასევე შეიძლება მივაკუთვნოთ სიმბოლოები: ‘*’, ‘?’ , ‘*’ , ‘?’ და ‘\’.



ნახ. 1.1. ფაილური სისტემის სტრუქტურის მაგალითი

სისტემაში არსებულ ფაილებს უნდა გააჩნდეთ სახელი, რომელიც უნიკალურია მხოლოდ შესაბამისი დირექტორიის ფარგლებში. UNIX ოპერაციულ სისტემაში ფაილთან დაკავშირებით გვაქვს ფაილის სრული და მიმართებითი სახელის ცნება. როგორც ნახ. 1.1-დან ჩანს „jack“ და „jill“ დირექტორიაში შედის ერთიდაიმავე სახელის მქონე სხვადასხვა დირექტორია („bin“), რომლებიც თავის მხრივ შეიცავენ ერთიდაიმავე სახელის მქონე სხვადასხვა ფაილს („a.out“). jack დირექტორიის შესაბამისი a.out ფაილისათვის ამოვწეროთ ყველა იმ დირექტორიის სახელი, რომლებიც საბაზო დირექტორიიდან დაწყებული ამ ფაილამდე მდებარობენ ფაილური სისტემის ლოგიკურ ხეზე (/ usr home jack bin a.out). მსგავს მიმდევრობაში პირველი ყოველთვის იქნება საბაზო დირექტორიის სახელი, ხოლო ბოლო დანიშნულების ფაილის სახელი. ეს სახელები, გარდა საბაზო დირექტორიისა და მის შემდგომ პირველი დირექტორიის სახელისა, ერთმანეთისაგან გამოვყოთ სიმბოლოთი ‘/’ („/usr/home/jack/bin/a.out“). მიღებული ჩანაწერი ცალსახად განსაზღვრავს ფაილს ფაილური სისტემის ლოგიკურ ხეზე. სწორედ ასეთ ჩანაწერს ეწოდება ფაილის სრული სახელი.

ფაილის სრული სახელი შეიძლება შეიცავდეს დირექტორიების საკმაო რაოდენობას და იყოს საკმარისად გრძელი, ასეთ სახელებთან მუშაობა კი ყოველთვის არაა მოხერხებული. ოპერაციულ სისტემაში მომუშავე ყოველი პროგრამისათვის, ბრძანებათა ინტერპრეტატორის³ ჩათვლით, რომელიც ასრულებს შეტანილ ბრძანებებს და გამოაქვს ახალი ბრძანების შეტანის ველი, ფაილური სისტემის ლოგიკურ ხეზე გამოიყოფა ერთერთი დირექტორია, რომელიც ინიშნება მოცემული პროგრამისათვის მიმდინარე ანუ სამუშაო დირექტორიად. იმის გაგება, თუ რომელი დირექტორია წარმოადგენს ბრძანებათა ინტერპრეტატორისათვის მიმდინარე დირექტორიას შესაძლებელია pwd ბრძანებით.

pwd ბრძანების სინტაქსისი

pwd

ბრძანების აღწერა

pwd ბრძანებას გამოაქვს მიმდინარე დირექტორიის სახელი მუშა ბრძანებათა ინტერპრეტატორისათვის.

³ ბრძანებათა ინტერპრეტატორი არის პროგრამა, რომელიც ინტერაქტიულ რეჟიმში მუშაობს და მომხმარებელს აძლევს ბრძანებების შესრულების საშუალებას. მაგალითად, Linux-ში ასეთი პროგრამაა terminal

მიმდინარე დირექტორიის ცოდნით შესაძლებელია ფაილური სისტემის ხეზე მისგან დანიშნულების ფაილამდე გზის გადება. კვანძების (დირექტორიების) მიმდევრობა, რომლებიც შეგვხდება ამ გზაზე ჩავწეროთ შემდეგი სახით: კვანძი, რომელიც შეესაბა-მება მიმდინარე დირექტორიას ამ ჩანაწერში არ შევა; საბაზო კატალოგის მიმართულე-ბით მოძრაობისას შემხვედრ ყოველ დირექტორიას აღვნიშნავთ სიმბოლოთი ‘..’, ხოლო მისგან მოძრაობისას შემხვედრ დირექტორიებს კი ჩავწერთ. ამ ჩანაწერში სხვადასხვა კვანძები გამოვყოთ სიმბოლოთი ‘/’. მიღებულ ჩანაწერს ეწოდება ფაილის მიმართებითი სახელი. ფაილის მიმართებითი სახელი იცვლება მიმდინარე დირექტორიის შეცვლისას. მაგალითად, ჩვენს მაგალითში (ნახ. 1.1), თუ მიმდინარე დირექტორია არის „/usr/home/jill“, მაშინ „/usr/home/jack/bin/a.out“ ფაილისათვის მიმართებითი მისამართი იქნება „/../../usr/home/jack/bin/a.out“, ხოლო, თუ მიმდინარე დირექტორია არის „/usr/home/jack“, მაშინ მისი მიმართებითი სახელი იქნება „bin/a.out“.

1.2.1. მომხმარებლის საშინაო დირექტორია

სისტემაში დარეგისტრირებული ყოველი მომხმარებლისათვის იქნება სპეცია-ლური დირექტორია, რომელიც მისი სისტემაში რეგისტრაციის შემდეგ იქცევა მისთვის მიმდინარე დირექტორიად. ამ დირექტორიამ მიიღო სახელწოდება მომხმარებლის საშინაო დირექტორია. `pwd` ბრძანების მეშვეობით შესაძლებელია იმის გარკვევა, თუ რომელია მომხმარებლის საშინაო დირექტორია.

1.2.2. უნივერსალური ცნობარი - ბრძანება `man`

აუცილებელია ყველა ოპერაციულ სისტემას გააჩნდეს საკუთარი საცნობარო მასალა, რომელშიც აღწერილი იქნება ბრძანებები, მათი სინტაქსისი, ოფციები და სხვა დამატებითი ინფორმაცია, რომელიც ხელშემწყობი იქნება ოპერაციულ სისტემასთან მუშაობისას. Windows-ის ოპერაციული სისტემისათვის ასეთი საცნობარო მასალის შემცველი არის უტილიტი `help`⁴. UNIX-მსგავს ოპერაციული სისტემებში გამოიყენება უტილიტი `man`. მისი გამოყენებით შესაძლებელია ამომწურავი ინფორმაციის მიღება სასწავლო კურსის მსვლე-ლობისას გამოყენებულ ბრძანებებზე და სისტემურ გამოძახებებზე, მათი გამოყენების სინტაქსისზე, პარამეტრებზე და ოფციებზე, რომლებიც დამატებით შესაძლებლობებს მატებენ მათ.

შევნიშნოთ, რომ ამ კურსის მსვლელობისას გამოყენებულ ბრძანებები და სისტემური გამოძახებები არ იქნება გამოყენებული სრულად. ამ ბრძანებებზე ამომწურავი ინფორმა-ციის მიღება შესაძლებელია `UNIX Manual`-ით. `UNIX Manual`-ში ინფორმაციის დიდი ნაწილი ხელმისაწვდომია ინტერაქტიული ფორმით `man` უტილიტის გამოყენებით.

`man` უტილიტის გამოყენება ადვილია, მისი სინტაქსისა

`man NAME`

სადაც `NAME` - ეს არის ის ბრძანება, უტილიტი, სისტემური გამოძახება, ბიბლიოთეკური ფუნქცია ან ფაილი, რომელზეც ვსაჭიროებთ დამატებითი ინფორმაციის მიღებას. ბრძანების შესრულებით მიღება მისი დანიშნულების და გამოყენების აღმწერი გვერდი, რომელიც გამოტანილი იქნება ტერმინალის შიგნით. თუ აღმწერი ინფორმაცია რამდენიმე გვერდი-

⁴ Windows 7-დან დაწყებული დამატებით შეიძლება გამოყენებული იქნას უტილიტი `get-help`

საგან შედგება, მაშინ გვერდების გადასაფურცლად გამოიყენება ღილაკი <space>, წინა გვერდზე დასაბრუნებლად - კომბინაცია <ctrl> + , ხოლო ინფორმაციის დათვალიერების რეჟიმიდან გამოსასვლელად კი - ღილაკი <q>. გვერდის დათვალიერების რეჟიმიდან გამოსვლის შემდეგ ისევ ვუბრუნდებით ტერმინალს და შესაძლებელია ინტერაქტიულ რეჟიმში ბრძანებების შეყვანა.

თანამედროვე ოპერაციულ სისტემებში თავმოყრილია დიდი მოცულობის ინფორმაცია. ამ ინფორმაციაში შესაძლებელია მეორდებოდეს გარკვეული (ბრძანების ან სისტემური გამოძახების) სახელი. სასურველ ბრძანებაზე ამომწურავი ინფორმაციის მიღების მცდელობისას საჭირო ინფორმაციასთან ერთად შესაძლებელია მიღებულ იქნას ინფორმაცია, რომელიც სცდება ინტერესის სფეროს. დიდი მოცულობის ინფორმაციაში ძებნის ოპერაციის გაადვილების მიზნით მთელი ინფორმაცია დაყოფილია ჯგუფებად. ჯგუფებში გაერთიანებულია ბრძანებების დანიშნულების და ფუნქციების მიხედვით, მაგალითად, სისტემური გამოძახებები, ბიბლიოთეკური ფუნქციები, ბირთვის ბრძანებები და ა.შ.

ინფორმაციის დაყოფა ჯგუფებად შესაძლებელია განსხვავდებოდეს UNIX-ის სხვადასხვა ვერსიებს შორის. მაგალითად, Linux-ში ინფორმაცია დაყოფილია ჯგუფებად:

1. შესრულებადი ფაილები ან ინტერპრეტატორის ბრძანებები;
2. სისტემური გამოძახებები;
3. ბიბლიოთეკური ფუნქციები;
4. სპეციალური ფაილები (ჩვეულებრივ მოწყობილობათა ფაილები);
5. სისტემური ფაილების ფორმატები და მიღებული შეთანხმებები;
6. თამაშები;
7. მაკროპაკეტები და უტილიტები - ისეთი როგორიცაა man;
8. სისტემური ადმინისტრატორის ბრძანებები;
9. ბირთვის ქვეპროგრამები (არასტანდარტული განყოფილება).

სასურველი ბრძანებაზე ამომწურავი ინფორმაციის მისაღებად დამატებით სასურველია იმის ცოდნა თუ რომელ ჯგუფს მიეკუთვნება შესაბამისი ბრძანება. იმისათვის, რომ გავარკვიოთ როგორაა დაყოფილი მთელი ინფორმაცია ჯგუფად საჭირო man ბრძანება გამოვიყენოთ ფორმით: man man, ხოლო იმისათვის, რომ სასურველ ბრძანებაზე მივიღოთ ამომწურავი ინფორმაცია შესაბამისი ჯგუფიდან საჭიროა man ბრძანება გამოვიყენოთ ფორმით:

man GROUP NAME

სადაც GROUP არის შესაბამისი ჯგუფის სახელი, ხოლო NAME კი არის სასურველი ბრძანების სახელი.

1.3. ფაილური სისტემის უმარტივესი ბრძანებები

როგორც ზემოთ აღვნიშნეთ, ყოველ პროგრამას ოპერაციულ სისტემაში გამოეყოფა სამუშაო დირექტორია. ნებისმიერი მოქმედება (როგორიცაა ახალი ფაილის შექმნა, ფაილის რედაქტირება და ა.შ.), რომელიც იქნება განხორციელებული ამ პროგრამის მიერ ყოველგვარი ცვლილების გარეშე რეალიზებული იქნება მხოლოდ პროგრამის სამუშაო დირექტორიაში. მაგალითად, გრაფიკულ რეჟიმში ტექსტური რედაქტორით ფაილის შექმნისას

ფაილი შეიქმნება ტექსტური რედაქტორის სამუშაო დირექტორიაში, თუ მომხმარებელი არ შეცვლის მას. ხშირად ბრძანებათა ინტერპრეტატორით მუშაობისას შესაძლებელია მოგვიწიოს სხვადასხვა ბრძანებების შესრულება ერთი კონკრეტული დირექტორიისათვის, რომელიც განსხვავდება სამუშაო დირექტორიისაგან და განთავსებულია საბაზო დირექტორიის სხვა მხარეს. ასეთ შემთხვევაში ბრძანების ყოველი გამოყენება შესაბამისი დირექტორიისათვის გულისხმობს ბრძანებაში მითითებული იყოს შესაბამის დირექტორიის სრული ან მიმართებითი სახელი. ბრძანების ამ ფორმით გამოყენება ართულებს მის გამოყენებას. ბრძანების გამოყენების გამარტივება შესაძლებელია თუ შესაბამის დირექტორიას გადავაქცევთ სამუშაო დირექტორიად. ამ შემთხვევაში საკმარისი იქნება ბრძანებაში მითითებული იყოს შესაქმნელი ფაილის მხოლოდ სახელი, რაც ამარტივებს ბრძანების ჩაწერას.

სამუშაო დირექტორიად ფაილური სისტემის ნებისმიერი დირექტორიის გადასაქცევად გამოიყენება ბრძანება cd (change directory). მისი სინტაქსია

cd DIRNAME

სადაც DIRNAME⁵ არის იმ დირექტორიის სრული ან მიმართებითი სახელი, რომელიც უნდა იქცეს სამუშაო დირექტორიად.

ხშირად ჩვენ ვსაჭიროებთ ვიცოდეთ თუ რა ფაილებია განთავსებული ამა თუ იმ დირექტორიაში, როგორია ფაილებზე დაშვების უფლებები, არის თუ არა იქ დამალული ფაილები და ა.შ. დირექტორიის შიგთავსის (ხილული დირექტორიების და ფაილების) დასათვალიერებლად გამოიყენება ბრძანება ls (list), მისი სინტაქსისა

ls DIRNAME

სადაც DIRNAME არის იმ დირექტორიის სრული ან მიმართებითი სახელი, რომლის შიგთავსის დათვალიერებასაც ვაპირებთ.

ls ბრძანების გამოყენებით ოფციისა და ფაილის სახელის მითითების გარეშე გამოაქვს სამუშაო დირექტორიაში განთავსებული ხილული ფაილების ჩამონათვალი. -a ოფციით ბრძანებას გამოაქვს მითითებულ ფაილში არსებული ხილული და დამალული⁶ ფაილების ჩამონათვალი, ხოლო -l (ლ) ოფციით კი - დირექტორიაში განთავსებული ფაილების ჩამონათვალი მათივე ატრიბუტებით: დაშვების უფლებები, მფლობელი, ჯგუფი, ფაილის სახელი და ა.შ.

ბრძანებათა ინტერპრეტატორიდან შესაძლებელია ტექსტური მონაცემების შემცველი ფაილის შიგთავსის დათვალიერება მის გაუსწეველად. ამ მიზნით გამოიყენება ბრძანება cat, მისი სინტაქსისა

cat FILENAME⁷

სადაც FILENAME ტექსტური მონაცემების შემცველი ფაილია⁸. cat ბრძანებით

⁵ შევნიშნოთ, რომ სადაც შეგვხვდება სიტყვა DIRNAME, მის ქვეშ ვიგულისხმებთ დირექტორიის სრულ ან მიმართებით სახელს

⁶ სისტემაში დამალული ფაილების სახელები იწყება სიმბოლოთი ‘:’ (წერტილი). ასეთი ფაილი შეიძლება შექმნილი იყოს სისტემის ან გამოყენებითი პროგრამის მიერ

⁷ შევნიშნოთ, რომ სადაც შეგვხვდება სიტყვა FILENAME, მის ქვეშ ვიგულისხმებთ ფაილის სრულ ან მიმართებით მისამართს

⁸ თუ დასათვალიერებელი ფაილი დიდი მოცულობისაა, მაშინ ტერმინალში გამოტანილი იქნება მისი

შესაძლებელია ერთზე მეტი ფაილის შიგთავსის გამოტანა. ამ შემთხვევაში საჭიროა cat ბრძანებას მიუწეროთ ფაილის სახელები იმ მიმდევრობით თუ როგორი მიმდევრობით გვინდა ისინი გამოჩნდენ ტერმინალში.

გარდა ფაილის შიგთავსის დათვალიერებისა cat ბრძანებით ასევე შესაძლებელია ახალი ფაილის შექმნა მისი რედაქტირების რეჟიმში გადასვლით ან რაიმე ფაილებში არსებული ტექსტური მონაცემების ერთ ფაილში ჩასაწერად, ამ მიზნით საჭიროა მონაცემთა ნაკადის შეტანის შესაბამისი სიმბოლოს ‘>’ გამოყენება. ბრძანების სინტაქსისია

```
cat > FILENAME  
cat FILENAME1 FILENAME2 ... FILENAMEN > FILENAME
```

პირველი ბრძანების შესრულების შედეგად შეიქმნება ფაილი - FILENAME და გადავდივართ მისი რედაქტირების რეჟიმში. თუ ასეთი სახელის მქონე ფაილი არსებობდა, მაშინ მოხდება მასში კავიატურიდან შეტანილი მნიშვნელობების გადაწერა არსებულ მონაცემებზე. რედაქტირების რეჟიმიდან გამოსასვლელად გამოიყენება კომბინაცია <Ctrl> + <D>. მეორე ბრძანების შესრულების შედეგად FILENAME1 FILENAME2 ... FILENAMEN ფაილებში განთავსებული მონაცემები, მიმდევრობის შენარჩუნებით, ჩაიწერება FILENAME ფაილში. ამასთან, ფაილებში არსებული მონაცემების გამოტანა ტერმინალის ეკრაზზე არ მოხდება.

მონაცემთა ნაკადის შეტანის სიმბოლოს გამოყენებით ასევე შესაძლებელია ნებისმიერი ბრძანების (რომელიც იძლევა ტექსტურ შედეგს) შესრულების შედეგის ჩაწერა ფაილში (ტერმინალში გამოტანის გარეშე). ამ შემთხვევაში, მაგალითად, ls -al ბრძანების შესრულების შედეგის ფაილში FILENAME ჩასაწერად, ბრძანება უნდა შესრულდეს ფორმით

```
ls -al > FILENAME.
```

შევნიშნოთ, რომ თუ ფაილი FILENAME ბრძანების შესრულების მომენტში არ არსებობდა, მაშინ ის ავტომატურად შეიქმნება. თუ ფაილი არსებობდა, მაშინ იქ არსებულ მონაცემებს გადაეწერება ბრძანების შესრულების შედეგი.

ახალი ფაილის შექმნა ტერმინალიდან ასევე შესაძლებელია nano ტექსტური რედაქტორის გამოყენებით, რომელიც გადაგვიყვანს მისი რედაქტირების რეჟიმში. მისი სინტაქსისია

```
nano FILENAME
```

თუ ფაილი FILENAME სახელით უკვე არსებობდა შესაბამის დირექტორიაში, მაშინ გადავალთ მისი რედაქტირების რეჟიმში. nano ტექსტური რედაქტორის ქვედა ნაწილში განთავსებულია ღილაკების ჩამონათვალი (ნახ. 1.2), რომელთა გამოყენებითაც <Ctrl> ღილაკთან კომბინაციით შესაძლებელია სხვადასხვა მოქმედებების განხორციელება. მაგალითად, <Ctrl> + <X> კომბინაციით შესაძლებელია დოკუმენტის შენახვა და რედაქტირების რეჟიმიდან გამოსვლა.



ნახ. 1.2. nano ტექსტური რედაქტორის მართვის ღილაკები

ფაილურ სისტემაში ახალი დირექტორის შესაქმნელად გამოიყენება ბრძანება mkdir

ბოლო გვერდი. ასეთ ფაილებთან სამუშაოდ გამოიყენება ბრძანება more

(make directory), მისი სინტაქსისია

```
mkdir DIRNAME.
```

`mkdir` ბრძანების შესრულებით ასევე შესაძლებელია რამდენიმე დირექტორიის შექმნა. ამასთან, შესაძლებელია რამდენიმე ჩალაგებული დირექტორიის შექმნა, რაც ნიშნავს შემდეგს: თუ სამუშაო დირექტორიაში (`/home/student`) არ გვაქვს არცერთი დირექტორია და გვინდა, მაგალითად, 3 დონით ჩალაგებული დირექტორია (`DIR1/DIR2/DIR3`) შევქმნათ საჭიროა ტერმინალში ბრძანება შევასრულოთ შემდეგი მიმდევრობით

```
mkdir DIR1 DIR1/DIR2 DIR1/DIR2/DIR3
```

ანუ მიმდევრობით უნდა შევქმნათ ჯერ `DIR1`, შემდეგ `DIR1/DIR2` და ბოლოს `DIR1/DIR2/DIR3` დირექტორია.

ერთი დირექტორიიდან მეორეში ფაილის (ან ფაილების) გადასაკოპირებლად (`copy-paste`) გამოიყენება ბრძანება `cp` (`copy`). ის ასევე გამოიყენება ერთი დირექტორიის (ან დირექტორიების) სხვა დირექტორიაში რეკურსიული⁹ გადაკოპირებისათვის. მისი სინტაქსისია

```
cp FILENAME DDIRNAME  
cp FILENAME1 FILENAME2 ... FILENAMEN DDIRNAME  
cp -r SDIRNAME DDIRNAME  
cp -r DIR1 DIR2 ... DIRN DDIRNAME
```

სადაც `SDIRNAME` (source directory) საწყისი დირექტორიაა, `RRNAME` (destination directory) დანიშნულების დირექტორია, სადაც უნდა მოხდეს მონაცემების გადაკოპირება, `-r` ოფცია გამოიყენება რეკურსიული გადაკოპირებისათვის.

ფაილის ერთი დირექტორიიდან მეორეში გადასაადგილებლად (`cut-paste`) გამოიყენება ბრძანება `mv` (`move`). მისი სინტაქსისია

```
mv SFILENAME DFILENAME  
mv FILENAME1 FILENAME2 ... FILENAMEN DDIRNAME
```

თუ პირველ ბრძანებაში ორივე მისამართი უთითებს ერთიდაიმავე დირექტორიას, მაშინ მოხდება `SFILENAME` ფაილისათვის სახელის გადარქმევა და ახალი სახელი იქნება `DDIRNAME`. სხვა შემთხვევაში მოხდება ფაილის ერთი დირექტორიიდან მეორეში გადაადგილება. მეორე ბრძანებით ხდება `FILENAME1 FILENAME2 ... FILENAMEN` ფაილების გადაადგილება `DDIRNAME` დირექტორიაში. ასევე, `mv` ბრძანებით, დამატებითი ოფციის გამოყენების გარეშე, შესაძლებელია ერთი დირექტორიის (ან დირექტორიების) გადაადგილება მეორე დირექტორიაში.

დირექტორიიდან ფაილის (ან ფაილების) წასაშლელად გამოიყენება ბრძანება `rm` (`remove`). მისი სინტაქსისია

```
rm FILENAME1 FILENAME2 ... FILENAMEN
```

⁹ რეკურსიული კოპირება ნიშნავს დირექტორიის გადაკოპირებას მასში არსებული ქვედირექტორიებითა და ფაილებით

დირექტორიიდან მთლიანი ქვედირექტორიის წასაშლელად საჭიროა -r ოფციის გამოყენება, ანუ საჭიროა რეკურსიული წაშლის განხორციელება ფორმით

```
rm -r DIR1 DIR2 ... DIRN
```

შევნიშნოთ, რომ ზემოთ გამოყენებულ ბრძანებებში ფაილის სრული ან მიმართებითი სახელები შეიძლება უთითებდნენ როგორც ერთიდაიმავე დირექტორიის, ისე სხვა-დასხვა დირექტორიის ფაილებს.

1.4. ფაილების სახელების შაბლონი

ოპერაციულ სისტემაში სასურველი ფაილის მოძებნა შეიძლება გართულდეს რამდენიმე მიზეზის გამო: ფაილის სახელი უცნობია, მისი ადგილმდებარეობა უცნობია და ა.შ. ასეთ შემთხვევაში ოპერაციულ სისტემაში უნდა არსებობდეს ე.წ. მეტასიმბოლოგი, რომლებიც გამოიწვევენ ძებნის ოპერაციის გამარტივებას. ასეთი მეტასიმბოლოგი შეიძლება გამოყენებულიქნას არამხოლოდ ძებნის მოქმედების განხორციელებისას, არამედ ფაილების მიმართ სხვადასხვა ბრძანებების გამოყენების დროსაც. ხშირად ამ მეტასიმბოლოგი გამოყენებით ბრძანებებთან გამოსაყენებელ სიმბოლოთა კომბინაციას შაბლონს უწოდებენ. მეტასიმბოლოგი, რომლებიც ამარტივებენ ამა თუ იმ ბრძანების გამოყენებას მოცემულია შემდეგ ცხრილში

*	-	შეესაბამება ყველა სიმბოლოს ჰათვლით;
?	-	შეესაბამება მხოლოდ ერთ სიმბოლოს;
[...]	-	შეესაბამება კვადრატულ ფრჩხილებში მითითებულ ნებისმიერ სიმბოლოს ან ალფავიტის ასოს, რომლებიც ერთიმეორისაგან შეიძლება გამოყოფილი იყოს სიმბოლოთი ' ; ' ან '-' (თუ ეთითება უწყვეტი შუალედი)

მაგალითად, *.c შაბლონს აკმაყოფილებს მიმდინარე დირექტორიის ყველა ის ფაილი, რომლის გაფართოებაც არის .c. [a-d]* შაბლონს აკმაყოფილებს მიმდინარე დირექტორიის ყველა ის ფაილი, რომლის სახელიც იწყება a, b, c, d სიმბოლოებიდან ნებისმიერით. არსებობს ერთი შეზღუდვა * მეტასიმბოლოს გამოყენებაზე ფაილის სახელის დასაწყისში, მაგალითად, *c შაბლონის შემთხვევაში. ასეთი შაბლონებისათვის იმ ფაილების სახელები, რომლებიც იწყება სიმბოლოთი ' ; ' ითვლება, რომ ისინი არ აკმაყოფილებენ შაბლონს.

1.5. მომხმარებელი და მისი ჯგუფი

მომხმარებელს ოპერაციულ სისტემაში მუშაობა შეუძლია მხოლოდ მასში გარკვეული სახელით დარეგისტრირების შემდეგ. ოპერაციულ სისტემას არ შეუძლია ლოგიკური სახელებით მანიპულირება, ამიტომ სისტემაში დარეგისტრირებულ ყოველ მომხმარებელს ენიჭება საკუთარი საიდენტიფიკაციო ნომერი UID (user identifier), გარკვეული რიცხვითი მნიშვნელობა, რომელიც უნიკალურია სისტემის მასშტაბით.

სისტემაში დარეგისტრირებული მომხმარებელი აუცილებლად საჭიროებს ოპერაციული სისტემის გარკვეულ რესურსზე მიმართვას. მომხმარებლები შეიძლება იყვნენ სხვადასხვა პრივილეგიებით და თანაბრად არ საჭიროებენ რესურსებზე წვდომას. ოპერაციულ სისტემაში მომხმარებლები პრივილეგიების მიხედვით ერთიანდება სხვადასხვა ჯგუფებში და ღებულობენ რესურსებზე შესაბამის წვდომას. მომხმარებლების ჯგუფების ერთიმეორისაგან განსხვავების მიზნით იყენებს ჯგუფის იდენტიფიკატორს GID (group

identifier), გარკვეული რიცხვითი მნიშვნელობა, რომელიც უნიკალურია სისტემის მასშტაბით.

მომხმარებლის იდენტიფიკატორის - UID, და მისი ჯგუფის იდენტიფიკატორის - GID, მნიშვნელობის მისაღებად შესაბამისად გამოიყენება სისტემური გამოძახებები getuid() და getgid().

getuid() და getgid() სისტემურ გამოძახებათა პროტოტიპი

```
#include<sys/types.h>
```

```
#include<unistd.h>
```

```
uid_t getuid(void);
```

```
gid_t getgid(void);
```

სისტემურ გამოძახებათა აღწერა

getuid სისტემური გამოძახება აბრუნებს მიმდინარე პროცესის მომხმარებლის იდენტიფიკატორს.

getgid სისტემური გამოძახება აბრუნებს მიმდინარე პროცესის მომხმარებლის ჯგუფის იდენტიფიკატორს.

uid_t და gid_t არის C ენის მთელრიცხვა ტიპის სინონიმი.

UNIX-მსგავს სისტემაში განასხვავებენ სამი კატეგორიის მომხმარებელს: მომხმარებელი, რომელიც რეგისტრირებულია სისტემაში გარკვეული სახელით, ჯგუფი, რომელსაც ის მიეკუთვნება და სხვა მომხმარებლები, რომლებიც მომხმარებლის ჯგუფს არ მიეკუთვნებიან. თითოეული კატეგორიის აუცილებლად საჭიროებს გარკვეულ დაშვების უფლებებს.

1.6. ფაილზე დაშვების უფლებები

გამოთვლით სისტემას შეიძლება ჰყავდეს ბევრი მომხმარებელი. თითოეული მომხმარებელი ინახავს მისთვის მნიშვნელოვან სხვადასხვა ფაილებს (იურიდიული, ფინანსური და ა.შ.). ოპერაციულ სისტემას უნდა შეეძლოს მომხმარებლისათვის მნიშვნელოვანი ფაილების დაცვა არასანქცირებული დაშვებისაგან. ამ მიზნით UNIX-მსგავს სისტემებში განასხვავებენ დაშვების სამ უფლებას:

- კითხვის უფლება - r (read);
- რედაქტირების უფლება - w (write);
- შესრულების უფლება - x (execute).

მონაცემების შემცველი ფაილებისათვის ამ უფლებების არსი ემთხვევა მათ შინაარსს. დირექტორიებისათვის ის რამდენადმე განსხვავებულია. დირექტორიისათვის კითხვის უფლება ნიშნავს ამ დირექტორიაში არსებული ფაილების სახელების კითხვას. ვინაიდან დირექტორიისათვის უფლება „შესრულება“ ყოველგვარ აზრს მოკლებულია (ისევე როგორც არაშესრულებადი რეგულარული ფაილისათვის), მისთვის შესრულების უფლება იცვლის მნიშვნელობას: ამ უფლების ქონა იძლევა დირექტორიაში შემავალ ფაილებზე დამატებითი ინფორმაციის მიღების შესაძლებლობას (მათი მოცულობა, მფლობელი, შექმნის თარიღი და ა.შ.). ამ უფლების გარეშე შეუძლებელია დირექტორიაში არსებული ფაილის შემცველობის დათვალიერება, რედაქტირება და შესრულება. შესრულების უფლება ასევე საჭიროა დირექტორიისათვის იმისათვის, რომ მისი გადაქცევა შესაძლებელი იყოს მიმდინარე დირექტორიად. ეს უფლება საჭიროა ასევე მისკენ მიმავალ

გზაზე განთავსებული ყველა დირექტორიისათვის. დირექტორიისათვის ჩაწერის უფლება იძლევა მისი შემცველობის შეცვლის შესაძლებლობას: ფაილის შექმნასა და წაშლას, მათი სახელის გადარქმევას. აღვნიშნოთ, რომ ფაილის წასაშლელად საკმარისია იმ დირექტორიისათვის შესრულების და ჩაწერის უფლების ქონა, რომელშიც შედის ფაილი, მიუხედავად თვითონ ფაილზე დაშვების უფლების ქონისა.

ფაილზე დაშვების უფლებების დასათვალიერებლად გამოიყენება ls ბრძანება ოფციით -l (ლ).

ფაილურ სისტემაში წარმოქმნილი ყოველი ფაილისათვის ინახება მისი მფლობელისა და მფლობელის ჯგუფის სახელები. შევნიშნოთ, რომ მფლობელთა ჯგუფის სახელი აუცილებელი არაა ემთხვეოდეს ფაილის შემქმნელის ჯგუფის სახელს. გამარტივებულად შეიძლება ჩაითვალოს ის, რომ Linux ოპერაციულ სისტემაში ახალი ფაილის შექმნისას მის მფლობელად ითვლება მომხმარებელი, რომელმაც შექმნა ის, ხოლო მფლობელთა ჯგუფად კი - ის ჯგუფი, რომელსაც მიეკუთვნება მომხმარებელი. ფაილის მფლობელს ან სისტემურ ადმინისტრატორს შეუძლია შეცვალოს ფაილის მფლობელი ან მფლობელთა ჯგუფი chown და chgrp ბრძანებების გამოყენებით შესაბამისად.

chown ბრძანების სინტაქსისი

```
chown owner FILENAME1 FILENAME2 ... FILENAMEN
```

ბრძანების აღწერა

chown ბრძანების მეშვეობით ხდება ფაილის მფლობელის ჯგუფის შეცვლა.

owner პარამეტრით შესაძლებელია ახალი მფლობელის მოცემა როგორც სიმბოლური ფორმით, მფლობელის ფაილის სახელი username, ან რიცხვითი მნიშვნელობა, როგორც მისი UID.

FILENAME1 FILENAME2 ... FILENAMEN პარამეტრები კი - იმ ფაილების სახელებია, რომელთაც ეცვლებათ მფლობელები. მათ ნაცვლად შესაძლებელია შაბლონის გამოყენებაც

chgrp ბრძანების სინტაქსისი

```
chgrp group FILENAME1 FILENAME2 ... FILENAMEN
```

ბრძანების აღწერა

chgrp ბრძანების მეშვეობით ხდება ფაილის მფლობელის ჯგუფის შეცვლა.

group პარამეტრით შესაძლებელია ფაილის მფლობელთა ჯგუფის მოცემა როგორც სიმბოლური ფორმით, ფაილის მფლობელთა ჯგუფის სახელი, ასევე რიცხვითი მნიშვნელობით, როგორც მისი GID.

FILENAME1 FILENAME2 ... FILENAMEN პარამეტრები კი - იმ ფაილების სახელებია, რომელთაც ეცვლებათ მფლობელთა ჯგუფი. მათ ნაცვლად შესაძლებელია შაბლონის გამოყენებაც

შევნიშნოთ, რომ შესაძლებელია ბრძანების შესრულება საჭიროებდეს ადმინისტრატორის უფლებებს. ადმინისტრატორის სახელით ბრძანების შესასრულებლად საჭიროა ჩვეულებრივი ფორმით აკრეფილი ბრძანების წინ მიეთითოს სიტყვა sudo. ბრძანების ასეთნაირად შესრულების შემთხვევაში სისტემა მოითხოვს ადმინისტრატორის პაროლის შეყვანას.

ფაილის მფლობელის და ჯგუფის შეცვლასთან ერთად შესაძლებელია ფაილზე დაშვების უფლებების შეცვლა. ამ მიზნით გამოიყენება ბრძანება chmod.

chmod ბრძანების სინტაქსისი

chmod [who] { + | - | = } [perm] FILENAME1 FILENAME2 ... FILENAMEN

ბრძანების აღწერა

chmod ბრძანების მეშვეობით ხდება ერთ ან რამდენიმე ფაილზე უფლებების შეცვლა.

who პარამეტრი განსაზღვრავს რომელი კატეგორიის მომხმარებლებისათვის ხდება დაშვების უფლებების შეცვლა. ის შეიძლება შეიცავდეს ერთ ან რამდენიმე სიმბოლოს:

a - ყველა ტიპის მომხმარებლის უფლებების მომართვა. თუ პარამეტრი who არაა მითითებული, მაშინ გაჩუმებით გამოიყენება a. დაშვების უფლებების მომართვისას ამ მნიშვნელობით მოცემული უფლებების მომართვა ხდება ფაილის შექმნის ნიღაბის მნიშვნელობის გათვალისწინებით;

u - ფაილის მფლობელისათვის დაშვების უფლებების მომართვა;

g - ფაილის მფლობელის ჯგუფში შემავალი მომხმარებლებისათვის დაშვების უფლებების მომართვა;

o - ყველა დანარჩენი მომხმარებლებისათვის დაშვების უფლებების მომართვა.

ოპერაცია, რომელიც სრულდება დაშვების უფლებებზე მომხმარებელთა მითითებული კატეგორიისათვის, განისაზღვრება ერთერთით შემდეგი სიმბოლოებიდან:

+ - დაშვების უფლებების დამატება;

- - დაშვების უფლებების გაუქმება;

= - დაშვების უფლებების შეცვლა, ე.ი. ყველა არსებულის გაუქმება და ჩამოთვლილის დამატება.

თუკი perm პარამეტრი არაა განსაზღვრული, მაშინ დაშვების ყველა არსებული უფლება იქნება უარყოფილი. perm პარამეტრი განსაზღვრავს დაშვების იმ უფლებებს, რომლებიც იქნებიან დამატებული, უარყოფილი ან მომართული შესაბამისი ბრძანების სანაცვლოდ. ის წარმოადგენს შემდეგი სიმბოლოებიდან ერთერთს ან მათ კომბინაციას:

r - უფლება კითხვაზე;

w - უფლება რედაქტირებაზე;

x - უფლება შესრულებაზე.

FILENAME1 FILENAME2 ... FILENAMEN პარამეტრები არის იმ ფაილების სახელები, რომელთათვისაც ხდება დაშვების უფლებების შეცვლა. სახელების ნაცვლად შესაძლებელია გამოყენებული იქნას მათი შაბლონებიც

ახალი ფაილის შექმნისას ოპერაციული სისტემა მას ანიჭებს დაშვების გარკვეულ უფლებებს გაჩუმებით. ისმის კითხვა: რითი სარგებლობს ოპერაციული სისტემა ფაილისათვის უფლებების მინიჭებისას? ამ მიზნით ის იყენებს ფაილის შექმნის ნიღაბს იმ პროგრამისათვის, რომელიც ქმნის ფაილს.

1.7. მიდინარე პროცესის ფაილების შექმნის ნიღაბი

მიმდინარე პროცესის ფაილის შექმნის ნიღაბი (umask) გამოიყენება open() და mknod() სისტემური გამოძახებების მიერ ახლად შექმნილი ფაილისათვის ან FIFO-სათვის დაშვების საწყისი უფლებების მოსამართად. ფაილის შექმნის ნიღაბის უმცროსი 9 ბიტი შეესაბამება მომხმარებლის, რომელმაც შექმნა ფაილი, ჯგუფის, რომელსაც ის მიეკუთვნება, და სხვა დანარჩენი მომხმარებლის დაშვების უფლებებს, როგორც ეს ნაჩვენებია ქვემოთ რვაობითი ჩანაწერის ფორმით:

0400 – ფაილის შექმნელი მომხმარებლისათვის კითხვის უფლება;

0200 – ფაილის შექმნელი მომხმარებლისათვის ჩაწერის უფლება;

- 0100 – ფაილის შემქმნელი მომხმარებლისათვის შესრულების უფლება;
- 0040 – ფაილის შემქმნელი მომხმარებლის ჯგუფისათვის კითხვის უფლება;
- 0020 – ფაილის შემქმნელი მომხმარებლის ჯგუფისათვის ჩაწერის უფლება;
- 0010 – ფაილის შემქმნელი მომხმარებლის ჯგუფისათვის შესრულების უფლება;
- 0004 – სხვა დანარჩენი მომხმარებლებისათვის კითხვის უფლება;
- 0002 – სხვა დანარჩენი მომხმარებლებისათვის ჩაწერის უფლება;
- 0001 – სხვა დანარჩენი მომხმარებლებისათვის შესრულების უფლება;

რომელიმე ბიტის 1-ის ტოლი მნიშვნელობით ჩაწერა ახლად შექმნილი ფაილისათვის კრძალავს დაშვების უფლების შესაბამის ინიციალიზაციას. ფაილის შექმნის ნიღაბის მნიშვნელობა შეიძლება შეიცვალოს umask() სისტემური გამოძახების ან umask ბრძანების მეშვეობით. ფაილის შექმნის ნიღაბი მეკვიდრეობით გადადის შვილპროცესზე fork() სისტემური გამოძახების საშუალებით ახალი პროცესის წარმოქმნისას და შედის პროცესის სისტემური კონტექსტის უცვლელ ნაწილში exec() სისტემური გამოძახებისას. ამ მეკვიდრეობის შედეგად umask ბრძანების გამოყენებით ნიღაბის შეცვლა ზემოქმედებს ყველა პროცესზე, რომლებიც წარმოქმნებიან ბრძანებათა გარსით, ახლად შექმნილი ფაილის დაშვების ატრიბუტებზე.

პროგრამა-ბირთვის ნიღაბის მიმდინარე მნიშვნელობის შეცვლა ან მისი დათვალიერება შესაძლებელია umask ბრძანების მეშვეობით. მაგალითად, მისათვის, რომ გავანულოთ სისტემის მიერ ახალი შესაქმნელი ფაილისათვის გაჩუმებით გადაცემული მნიშვნელობები საჭიროა ფაილის შექმნამდე umask ბრძანება გამოვიყენოთ ფორმით umask(0).

umask ბრძანების სინტაქსისი

umask [value]

ბრძანების აღწერა

umask ბრძანება გნკუთვნილია ბრძანებათა ბირთვის ფაილის ნიღაბის შესაქმნელად ან მისი მიმდინარე მნიშვნელობის დასათვალიერებლად. პარამეტრის გარეშე ბრძანებას გამოაქვს ფაილის შექმნის ნიღაბის მომართული მნიშვნელობა რვაობითი ფორმით. ახალი მნიშვნელობის მოსამართად ის მოიცემა როგორც value პარამეტრი რვაობითი ფორმით.

1.8. პროგრამული კოდი, მისი კომპილაცია და შესრულება

სასწავლო კურსის მსვლელობისას პროგრამული კოდს დავწერთ C++ ენაზე და შესაბამისად კოდისათვის გამოვიყენებთ `cpp` გაფართოების ფაილს. ჩვენს მიერ პროგრამული კოდი შეიძლება აკრეფილი იყოს უშუალოდ ტერმინალიდან `nano` ტექსტური რედაქტორის გამოყენებით ან ტექსტურ რედაქტორში `kate` (ნახ. 1.3), რომელშიც სამუშაო ველი იყოფა ორ ნაწილად. პირველ ნაწილში შესაძლებელია უშუალოდ პროგრამული კოდის აკრეფა, ხოლო მეორე ნაწილში გამოტანილია პროგრამში ინტეგრირებული ბირთვის რეჟიმი, საიდანაც შესაძლებელია ბრძანების დაკომპილირება და შემდგომ მისი შესრულება.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/types.h>
5 int main(int argc, char *argv[])
6 {
7     pid_t a = fork();
8     if (a == -1)
9     {
10         printf("პროცესი ვერ შეიქმნა");
11         exit(EXIT_FAILURE);
12     }
13     else
14     {
15         if (a == 0) // შვილი პროცესი
16     }

```

INS LINE UTF-8 fork.c
student@ubuntu:~/Desktop\$ cd /home/student/Desktop
student@ubuntu:~/Desktop\$

ნახ. 1.3. kate პროგრამის სამომხმარებლო ინტერფეისი

UNIX-მსგავს სისტემებში პროგრამის კომპილიაციისათვის გამოიყენება ინტეგრირებული კომპილატორები, როგორიცაა gcc, cc, g++ და ა.შ. სასწავლო კურსის მსვლელობისას ჩვენ გამოვიყენებთ g++ კომპილატორს. g++ კომპილატორის გამოყენებით კოდის (main ფუნქციის) შემცველი ფაილის კომპილაცია შესაძლებელია ფორმით

g++ CODEFILE

სადაც CODEFILE პროგრამული კოდის შემცველი ფაილია. უნდა აღინიშნოს, რომ ფაილის კომპილაციისას g++ კომპილატორი კომპილაციის შედეგის ფაილისათვის გაჩუმების წესით აგენერირებს სახელს a.out. ეს სახელი იქნება დაგენერირებული g++ კომპილატორი კომპილირებული ყველა ფაილისათვის მიუხედავად იმისა, არიან თუ არა ეს ფაილები ერთ დირექტორიაში. ერთ დირექტორიაში განთავსებული რამდენიმე ფაილის დაკომპილირების შემთხვევაში დირექტორიაში დარჩება ბოლოს კომპილირებული ფაილის შედეგი, სხვა ფაილების კომპილაციის შედეგი იკარგება. იმისათვის, რომ არ მოხდეს სხვადასხვა ფაილების კომპილაციისას შესაბამისი შედეგის ფაილების ერთიმეორებზე გადაწერა სასურველია ფაილის კომპილაციისას მისთვის პროგრამისტის მიერ იყოს შერჩეული შესაბამისი სახელი. ამ მიზნით g++ კომპილატორიში გამოიყენება ოფცია -o (ო). ამისათვის g++ კომპილატორი უნდა გამოვიყენოთ ფორმით

g++ CODEFILE -o OUTNAME

სადაც OUTNAME არის კომპილაციის შედეგად მიღებული ფაილის სახელი. შევნიშნოთ, რომ კომპილაციის შედეგის ფაილისათვის აუცილებელობას არ წარმოადგენს ".out" გაფართოების მიწერა.

კომპილირებული ფაილის შესრულება შესაძლებელია ფორმით

./OUTNAME

სემინარი 2. პროცესები Unix მსგავს სისტემებში

2.1. პროცესები და მისი კონტექსტი

ნებისმიერი ოპერაციული სისტემისათვის პროცესი თამაშობს მნიშვნელოვან როლს. ყველაფერი რაც გამოთვლით სისტემაში ხდება აგებულია პროცესების ცნების კონცეფციაზე. სისტემის ჩატვირთვიდან დაწყებული მისი მუშაობის დასრულებამდე. ამ ცნებაზე დამოკიდებულებით არის რეალიზებული თანამედროვე ოპერაციულ სისტემაში სხვადასხვა ტექნოლოგია, როგორიცაა ფაილური სისტემა, ვირტუალური მეხსიერება და ა.შ. ყველაზე მეტად პროცესის ცნების მნიშვნელობა ოპერაციული სისტემისათვის შეიძლება დანახული იქნას UNIX -მსგავს ოპერაციულ სისტემებთან მუშაობისას. ოპერაციულ სისტემაში არსებული ბირთვის და მომხმარებლის რეჟიმის მუშაობა ემყარება პროცესის ცნებას (სისტემური გამოძახებების შესრულება, სიგნალების გენერირება, მეხსიერების მართვა, განსაკუთრებული სიტუაციის დამუშავება, სხვადასხვა სერვისები).

სისტემაში წარმოქმნილი ყოველი პროცესი შესასრულებლად იყენებს სისტემაში არსებულ რესურსებს (მეხსიერება, პროცესორი, ფაილური სისტემის მიერ შეთავაზებული სამსახური, შეტანა/გამოტანის მოწყობილობა და ა.შ.). ის შესასრულებელი სამუშაოს მიხედვით შეიძლება სრულდებოდეს როგორც მომხმარებლის (user-mode), ასევე ბირთვის (kernel mode) რეჟიმში. მომხმარებლის რეჟიმში პროცესს შეუძლია განახორციელოს არაპრივილეგირებული საქმიანობა. მომხმარებლის რეჟიმიდან (შესაბამის სისტემური გამოძახების შესრულებით) ის გადადის ბირთვის რეჟიმში, სადაც მისთვის ნებადართულია პრივილეგირებული ბრძანებების შესრულება და ღებულობს შესაბამის მომსახურეობას (კითხულობს მონაცემებს, აწარმოებს გამოთვლებს და ა.შ.). რადგანაც პროცესი შეიძლება საჭიროებდეს ორივე რეჟიმში შესრულებას, ამიტომ მასთან ასოცირებულ ინფორმაციას ყოფენ ორ ნაწილად: მომხმარებლის კონტექსტი და ბირთვის კონტექსტი. მომხმარებლის კონტექსტის ქვეშ იგულისხმება ყველა ის ინფორმაცია, რომელიც საჭიროა მომხმარებლის რეჟიმში პროცესის შესასრულებლად, ესენია

- ინიცირებადი უცვლელი მონაცემები (მუდმივები);
- ინიცირებადი ცვლადი მონაცემები (ყველა ცვლადი, რომელთაც საწყისი მნიშვნელობა ენიჭება კომპილაციის ეტაპზე);
- არაინიცირებადი უცვლელი მონაცემები (ყველა სტატიკური ცვლადი, რომელთაც საწყისი მნიშვნელობა არააქვთ მინიჭებული კომპილაციის ეტაპზე);
- მომხმარებლის სტეკი;
- მონაცემები, რომლებიც განთავსებული არიან დინამიურად განაწილებად მეხსიერები, და ა.შ.

„ბირთვის კონტექსტი“ ცნების ქვეშ გაერთიანებულია სისტემური და რეგისტრული კონტექსტი. ბირთვის კონტექსტში გამოყოფენ ბირთვის სტეკს, რომელიც გამოიყენება პროცესის მუშაობისას ბირთვის რეჟიმში, და ბირთვის მონაცემებს, რომლებიც ინახება პროცესის მართვის ბლოკის (PCB) ანალოგიურ სტრუქტურებში. ბირთვის მონაცემებში შედის:

- მომხმარებლის იდენტიფიკატორი (UID);
- მომხმარებლის ჯგუფის იდენტიფიკატორი (GID);

- პროცესის იდენტიფიკატორი (PID);
- მშობელი-პროცესის იდენტიფიკატორი (PPID), და ა.შ.

2.2. პროცესის იდენტიფიკაცია

ყოველი ოპერაციულ სისტემა სისტემაში წარმოქმნილი პროცესების იდენტიფიცირებისათვის საჭიროებს გარკვეულ მექანიზმს. სხვადასხვა ოპერაციულ სისტემაში შეიძლება იდენტიფიკატორის როლში გამოყენებულ იქნას რიცხვითი მნიშვნელობა ან სიმბოლური სახელები ან მათი კომბინაცია. UNIX მსგავს სისტემებში წარმოქმნილი ყოველი პროცესი უნიკალური იდენტიფიკატორის როლში ღებულობს რიცხვით მნიშვნელობას - PID (process identifier). სხვადასხვა სისტემებში პროცესის იდენტიფიცირებისათვის გამოყოფილი რიცხვითი მნიშვნელობები მერყეობს 0-დან გარკვეულ მაქსიმალურ მნიშვნელობამდე. მაგალითად, Intel-ის ტიპის 32-თანრიგა პროცესორის ბაზაზე Linux სისტემაში პროცესების იდენტიფიცირებისათვის გამოყოფილია შუალედი $[0, 2^{32}-1]$. სისტემაში პროცესისათვის საიდენტიფიკაციო ნომრის მინიჭებისას უნიკალობის შენარჩუნების მიზნით ყოველ ახალ პროცესს სისტემა რიცხვით მნიშვნელობას ანიჭებს ზრდადი მიმდევრობით (ანუ ბოლოს წარმოქმნილი პროცესის საიდენტიფიკაციო ნომერს + 1). პროცესის მიერ დაკავებული ნომერი მისი დასრულების შემდეგ თავისუფლდება და მისი გამოყენება შესაძლებელია ახალი წარმოქმნილი პროცესისათვის. იმის გამო, რომ ოპერაციულ სისტემაში არ დაირღვეს პროცესების იდენტიფიცირების პროცესი სისტემაში დასრულებული პროცესებიდან გამონთავისუფლებული საიდენტიფიკაციო მნიშვნელობების გამოყენება ახალი პროცესებისათვის იდენტიფიკატორის როლში ხდება მხოლოდ მას შემდეგ რაც მიღწეული იქნება იდენტიფიცირებისათვის გამოყოფილი მნიშვნელობების მაქსიმალური მნიშვნელობა. ამის შემდეგ სისტემაში საიდენტიფიკაციო მნიშვნელობების მინიჭება იწყება მინიმალური თავისუფალი ნომრით და გრძელდება ზრდადი მიმდევრობით.

პროცესის და მისი მშობელი პროცესის იდენტიფიკატორის მნიშვნელობის მისაღებად გამოიყენება სისტემური გამოძახებები getpid და getppid. მათი პროტოტიპები და მონაცემთა შესაბამისი ტიპები განსაზღვრულია სისტემურ ფაილებში <sys/types.h> და <unistd.h>.

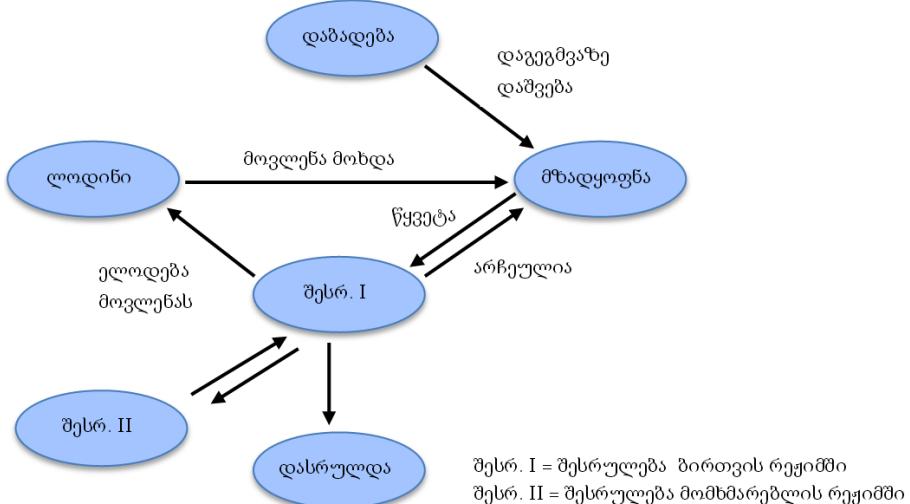
```
#include<unistd.h>
#include<sys/types.h>
pid_t getpid(void); // პროცესის იდენტიფიკატორი
pid_t getppid(void); // მშობელი პროცესის იდენტიფიკატორი
```

2.3. პროცესის სიცოცხლის ციკლი

ოპერაციულ სისტემაში წარმოქმნილი პროცესი თავისი არსებობის მანძილზე (წარმოქმნიდან დასრულებამდე) გადის რამდენიმე დისკრეტულ მდგომარეობას, სხვადასავა ოპერაციულ სისტემაში ეს რიცხვი შეიძლება იყოს განსხვავებული. ნახ. 2.1-ზე ნაჩვენებია UNIX მსგავს ოპერაციულ სისტემებში პროცესების მდგომარეობის მოკლე დიაგრამა.

სისტემაში წარმოქმნილი ყოველი პროცესი იმყოფება მდგომარეობაში „დაბადება“. მას შემდეგ რაც ის აღიჭურვება მიმდინარე მომენტისათვის შესასრულებლად საჭირო რესურსებით გადადის მდგომარეობაში „მზადყოფნა“ და იკავებს რიგს. როგორც კი პრო-

ცესისათვის ხელმისაწვდომი გახდება პროცესორი ის მდგომარეობიდან „მზადყოფნა“ გადადის მდგომარეობაში „შესრულება“. როგორც ნახ. 2.1-დან ჩანს, მდგომარეობა „შესრულება“ დაყოფილია ორ ნაწილად: „შესრულება ბირთვის რეჟიმში“ და „შესრულება მომხმარებლის რეჟიმში“. პროცესი მდგომარეობაში „შესრულება მომხმარებლის რეჟიმში“ ასრულებს მომხმარებლის გამოყენებით ინსტრუქციებს. მდგომარეობაში „შესრულება ბირთვის რეჟიმში“ სრულდება ოპერაციული სისტემის ბირთვის ინსტრუქციები მიმდინარე პროცესთან მიმართებაში (მაგალითად, სისტემური გამოძახების ან წყვეტის დამუშავებისას). მდგომარეობიდან „შესრულება“ პროცესი შეიძლება გამოვიდეს რამდენიმე მიზეზით:



ნახ. 2.1. პროცესის მდგომარეობის დიაგრამა

- დროითი კვანტის ამოწურვის გამო.** ამ შემთხვევაში პროცესი გადადის მდგომარეობაში „მზადყოფნა“, იკავებს რიგს და ელოდება პროცესორის გამონთავისუფლებას;
- საჭირო რესურსის არარსებობა.** თუ პროცესი საჭიროებს გარკვეულ რესურსს (მეხსიერებას, შეტანა/გამოტანის მოწყობილობას, სხვა პროცესის მიერ განხორციელებული გამოთვლების შედეგი, მონაცემები და ა.შ.), რომელიც საჭიროა მისი შემდგომი შესრულებისათვის, მაშინ ოპერაციულ სისტემას ის გადაყავს მდგომარეობაში „ლოდინი“;
- დასრულება.** პროცესი დასრულების შემთხვევაში გადადის მდგომარეობაში „დასრულდა“. ოპერაციული სისტემა ანთავისუფლებს მის მიერ დაკავებულ ყველა რესურს და მიღებულ შედეგს ტოვებს მეხსიერებაში იმ დრომდე სანამ მას არ მოითხოვს წარმოქმნელი პროცესი ან არ მოხდება სისტემის გადატვირთვა.

სისტემაში ამა თუ იმ პროცესის მიერ რესურსის გამონთავისუფლების ან შეტანა/გამოტანის მოწყობილობიდან წყვეტის შემდეგ სისტემა ამოწმებს „ლოდინი“ მდგომარეობაში მყოფ პროცესებს ხომ არ იმყოფება იქ რაიმე პროცესი, რომელიც ელოდებოდა სისტემაში წარმოქმნილ მოვლენას. ასეთის არსებობის შემთხვევაში პროცესი იღებს საჭირო რესურს ან მონაცემებს და თუ ის საკმარისია მისი შესრულების გაგრძელებისათვის, მაშინ პროცესი გადადის მდგომარეობაში „მზადყოფნა“ და იკავებს რიგს. სხვა შემთხვევაში ელოდება საჭირო მოვლენის დადგომას.

2.4. პროცესების იერარქია

UNIX ოპერაციულ სისტემაში არსებული ყველა პროცესი გარდა ერთი პროცესისა (kernel¹ იდენტიფიკატორით 0) შეიძლება წარმოქმნილი იყოს მხოლოდ რაიმე სხვა პროცესით. UNIX-მსგავს სისტემებში სხვა დანარჩენი პროცესის წარმომქმნელის როლში შეიძლება გამოდიოდეს პროცესი ნომრით 0 ან 1.

პროცესი, რომელიც წარმოქმნის ახალ პროცესს მშობელი პროცესი ეწოდება, ხოლო წარმოქმნილ პროცესს კი - შვილი პროცესი. თავის მხრივ შვილი პროცესს შეუძლია წარმოქმნას საკუთარი შვილი პროცესები და ა.შ. ამ ფორმით პროცესების წარმოქმნისას იქმნება ხისებრი სტრუქტურა, რომელსაც პროცესების იერარქიული ხე ეწოდება. UNIX სისტემაში არსებული ყოველი პროცესი ერთმანეთთან დაკავშირებულია დამოკიდებულებით მშობელი - შვილი. როდესაც მშობელი პროცესი შვილ პროცესზე ადრე ასრულებს საკუთარ სიცოცხლეს (გადაგადის მდგომარეობაში დასრულდა) შვილი პროცესის PCB-ში, იერარქიის მთლიანობის შენარჩუნების მიზნით, მშობელი პროცესის იდენტიფიკატორის მნიშვნელობის (PPID – parent process identifier) როლში (ნაცვლად მისი რეალური მნიშვნელობისა) იწერება მნიშვნელობა 1, რომელიც შეესაბამება init² პროცესის იდენტიფიკატორს. რითაც ფაქტიურად init პროცესი „იშვილებს დაობლებულ“ პროცესს.

2.5. UNIX-ში პროცესის წარმოქმნა

ყოველი ოპერაციული სისტემა საჭიროებს გარკვეულ მექანიზმს პროცესის შესაქმნელად. Windows-ის სისტემაში ამ მიზნით გამოიყენება ფუნქცია createprocess. ამ ფუნქციას ჩვენ არ გამოვიყენებთ და ამიტომ დავვმაყოფილდებით მხოლოდ მისი განაცხადით. ფუნქციას აქვს სახე

```
BOOL CreateProcess(
    LPCWSTR     pszImageName,
    LPCWSTR     pszCmdLine,
    LPSECURITY_ATTRIBUTES  psaProcess,
    LPSECURITY_ATTRIBUTES  psaThread,
    BOOL        fInheritHandles,
    DWORD       fdwCreate,
    LPVOID      pvEnvironment,
    LPWSTR      pszCurDir,
    LPSTARTUPINFO         psiStartInfo,
    LPPROCESS_INFORMATION pProcInfo
);
```

UNIX მსგავს ოპერაციულ სისტემებში პროცესის წარმოსამნელად გამოიყენება სისტემური გამოძახება fork(). წარმოქმნილი პროცესი პრაქტიკულად წარმოადგენს მშობელი პროცესის სრულ ასლს. პროგრამისტის მიერ ცხადად თუ არ იქნა განსაზღვრული პროცესების საქმიანობა, მაშინ მშობელი და შვილი პროცესი გამოიყენებს ერთიდაიმავე რესურ-

¹ სხვადასხვა სისტემაში ასეთი პროცესის სახელი შეიძლება იყოს განსხვავებული. ასევე განსხვავებული შეიძლება იყოს მისი იდენტიფიკატორის მნიშვნელობა, მაგალითად, 1

² init არის სისტემური პროცესი, რომლის სიცოცხლის ხანგრძლივობაც განსაზღვრავს ოპერაციული სისტემის ფუნქციონირების დროს

სებს და მონაცემებს და შესაბამისად დაკავდება ერთიდაიმავე საქმით.

fork() სისტემური გამოძახების პროცესი

```
#include<sys/types.h>
```

```
#include<unistd.h>
```

```
pid_t fork(void);
```

სისტემური გამოძახების აღწერა

სისტემური გამოძახება fork() გამოიყენება UNIX ოპერაციულ სისტემაში ახალი პროცესის წარმოსამნელად. წარმოქმნილ პროცესს მშობელ-პროცესთან მიმართებაში ეცვლება შემდეგი პარამეტრების მნიშვნელობები:

- პროცესის იდენტიფიკატორი;
- მშობელი პროცესის იდენტიფიკატორი;
- დრო, დარჩენილი SIGALRM სიგნალის მისაღებად;
- მშობელი პროცესისათვის განკუთვნილი სიგნალები არ მიუვათ შვილ პროცესებს.

fork სისტემური გამოძახების წარმატებით დასრულების შემთხვევაში ის აბრუნებს ორ მნიშვნელობას: პირველი მშობელ პროცესში და მეორე წარმოქმნილ პროცესში. ახალი პროცესის წარმოქმნისას წარმოქმნილ პროცესში სისტემური გამოძახება აბრუნებს მნიშვნელობას 0, ხოლო მშობელ პროცესში კი - შვილი პროცესის იდენტიფიკატორის ტოლ მნიშვნელობას. რაიმე მიზეზით fork სისტემური გამოძახების წარუმატებლად³ დასრულების შემთხვევაში სისტემური გამოძახება მის მაინიცირებელ პროცესში აბრუნებს -1 - ის ტოლ მნიშვნელობას.

რადგანაც fork სისტემური გამოძახება წარმატებით დასრულების შემთხვევაში აბრუნებს ორ განსხვავებულ მნიშვნელობას, ამიტომ ჩვენ შეგვიძლია მშობელი და შვილი პროცესების სამუშაოს დაგეგმისათვის გამოვიყენოთ if-else კონსტრუქცია უშუალოდ fork სისტემური გამოძახების გამოყენების შემდეგ. fork სისტემური გამოძახების მუშაობის პრინციპი-დან გამომდინარე ორჯერ შესრულდება მის შემდეგ მომავალი პროგრამული კოდი ანუ, შესრულდება if-else კონსტრუქციის ორივე ნაწილი ერთხელ მშობლისთვის და ერთხელ შვილისთვის⁴. fork სისტემური გამოძახების გამოყენებით ახალი პროცესის წარმოქმნისა და მშობელი და შვილი პროცესის სამუშაოს დაგეგმვის შესაბამისი პროგრამული ფრაგმენტი გამოიყურება შემდეგნაირად:

```
pid = fork();
if(pid == -1){ // შეცდომა
    ...
    // შეცდომის შესაბამისი კოდი და მიზეზი
} else if (pid == 0) { // შვილი
    ...
    // შვილის შესაბამისი ოპერატორები
} else { // მშობელი
    ...
    // მშობელის შესაბამისი ოპერატორები
}
```

³ სისტემური გამოძახება წარუმატებლად შეიძლება დასრულდეს რამდენიმე მიზეზით: სისტემაში მიღწეულია პროცესების მაქსიმალური რაოდენობა ან მეხსიერებაში არა ახალი პროცესის განსათავსებლად საკმარისი ადგილი

⁴ პროცესების შესრულების თანმიმდევრობა წინასწარ უცნობია

2.6. პროცესის დასრულება

C ენაზე დაწერილი პროგრამა კორექტულად შეიძლება დასრულდეს ორი მეთოდით: პირველი, როცა ის სრულდება return ოპერატორის გამოყენებით; მეორე მეთოდი გამოიყენება პროგრამის ნებისმიერ ადგილას პროცესის დასრულების საჭიროებისას. ამისათვის გამოიყენება C ენის სტანდარტული ბიბლიოთეკის ფუნქციებიდან exit() ფუნქცია. ამ ფუნქციის შესრულებისას ხდება შეტანა/გამოტანის ნაწილობრივ შევსებული ბუფერის გასუფთავება შესაბამისი ნაკადების დახურვით, რის შემდეგაც ხდება პროცესის მუშაობის შეწყვეტის და „დასრულდა“ მდგომარეობაში გადამყვანი სისტემური გამოძახების ინიცირება.

ფუნქციიდან დაბრუნება მიმდინარე პროცესში არ ხდება და, შესაბამისად, ფუნქციაც არაფერს არ აბრუნებს.

exit() ფუნქციის პარამეტრის მნიშვნელობა გადაეცემა ოპერაციული სისტემის ბირთვს და შემდეგ შეიძლება მიიღოს იმ პროცესმა, რომელმაც წარმოქმნა შვილი პროცესი.

სინამდვილეში main() ფუნქციის ბოლოს მიღწევით ასევე ხდება ამ ფუნქციის არაცხადი გამოძახება პარამეტრის მნიშვნელობით 0.

თუ შვილი პროცესი საკუთარ სამუშაოს ამთავრებს მშობელ პროცესზე ადრე და მშობელ პროცესს ცხადად არ მიუთითებია, რომ ის საჭიროებს ინფორმაციის მიღებას დასრულებული პროცესის სტატუსზე, მაშინ დასრულებული პროცესი სისტემიდან არ იშლება და რჩება მდგომარეობაში „დასრულდა“ სანამ ან დასრულდება მშობელი პროცესი ან იმ დრომდე, სანამ მშობელი პროცესი არ მიიღებს მასზე ინფორმაციას. „დასრულდა“ მდგომარეობაში მყოფ პროცესებს UNIX მსგავს ოპერაციულ სისტემაში ეწოდებათ ზომბი პროცესები (zombie).

```
exit() ფუნქციის პროტოტიპი
#include<stdlib.h>
void exit(int status);
```

ფუნქციის აღწერა

exit() ფუნქცია გამოიყენება პროცესის კორექტული დასრულებისათვის. ამ ფუნქციის გამოყენების შემდეგ ხდება შეტანა/გამოტანის ყველა ნაწილობრივ შევსებული ბუფერის გასუფთავება შესაბამისი ნაკადების დახურვით (ფაილების, pipe, FIFO, სოკეტების).

status - პროცესის კოდის დასრულების - პარამეტრის მნიშვნელობა გადაეცემა ოპერაციული სისტემის ბირთვს და შემდეგ შეიძლება მიღებული იქნას დასრულებული პროცესის წარმოქმნელი პროცესის მიერ.

დასრულებულ პროცესზე ინფორმაციის მისაღებად მშობელ პროცესს შეუძლია გამოიყენოს სისტემური გამოძახება waitpid() ან მისი შემოკლებული ფორმა wait(). wait() სისტემური გამოძახების შესრულებით ხორციელდება მშობელი პროცესის ბლოკირება იმ დრომდე სანამ არ დასრულდება წარმოქმნილი პროცესი. წარმოქმნილი პროცესის დასრულების შემდეგ მშობელ პროცესს გადაეცემა ინფორმაცია შვილი პროცესის სტატუსზე და მართვა უბრუნდება მშობელ პროცესს. wait() სისტემური გამოძახებისაგან განსხვავებით waitpid() სისტემური გამოძახებით შესაძლებელია მშობელი პროცესის შესრულების ბლოკირება კონკრეტული იდენტიფიკატორის მქონე პროცესის დასრულებამდე.

wait() და waitpid() სისტემური გამოძახება

```
#include<sys/types.h>
#include<sys/wait.h>
pid_t waitpid(pid_t pid, int *status, int options);
pid_t wait(int *status);
```

სისტემური გამოძახების აღწერა

waitpid() სისტემური გამოძახებით ხდება მიმდინარე პროცესის ბლოკირება სანამ ან არ დასრულდება მის მიერ წარმოქმნილი `pid` იდენტიფიკატორის მქონე პროცესი.

- თუ `pid > 0`, მაშინ ხდება `pid` იდენტიფიკატორით პროცესის დასრულებაზე დალოდება
- თუ `pid = 0`, მაშინ ველოდებით იმ ჯგუფის ყველა პროცესის დასრულებას, რომელსაც მშობელი პროცესი მიეკუთვნება.
- თუ `pid = -1`, მაშინ ველოდებით ნებისმიერი წარმოქმნილი პროცესის დასრულებას.
- თუ `pid < -1`, მაშინ ველოდებით ნებისმიერი პროცესის ან პროცესების ჯგუფის დასრულებას, რომლის იდენტიფიკატორიც `pid`-ის აბსოლუტური მნიშვნელობის ტოლია.

თუ სისტემურმა გამოძახებამ აღმოაჩინა წარმოქმნილი დასრულებული პროცესი `pid` სპეციფიური პარამეტრით, მაშინ ეს პროცესი იშლება სისტემიდან, ხოლო `status` პარამეტრით მითითებულ მისამართზე ინახება მის დასრულებაზე ინფორმაცია. `status` პარამეტრი შეიძლება მოცემული იყოს მნიშვნელობით `NULL` იმ შემთხვევაში, თუ ამ ინფორმაციას ჩვენთვის არააქვს მნიშვნელობა.

wait სისტემური გამოძახება წარმოადგენს `waitpid` სისტემური გამოძახების სინონიმს პარამეტრების მნიშვნელობით: `pid = -1, options = 0`.

2.7. C ენაში `main()` ფუნქციის პარამეტრები

C ენაზე დაწერილი პროგრამის შესრულება იწყება `main()` ფუნქციის გამოძახებით.

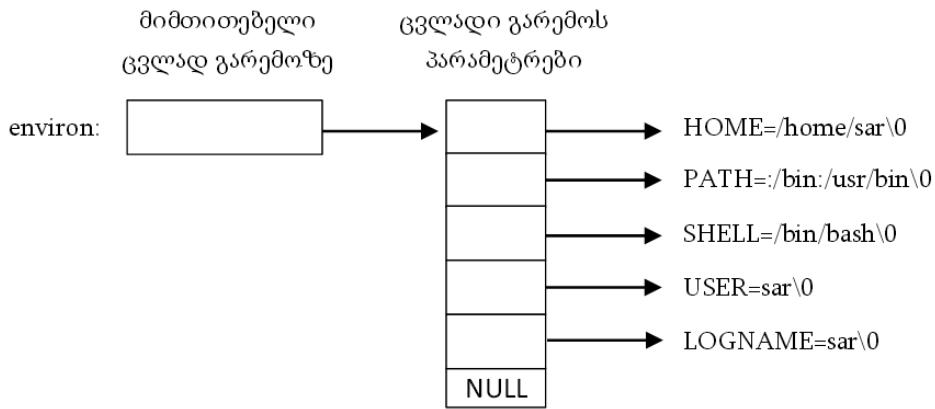
`main()` ფუნქციის სრული პროტოტიპი გამოიყურება შემდეგნაირად:

```
int main(int argc, char *argv[]);
```

სადაც პირველი პარამეტრი გამოსახავს ბრძანებათა ინტერპრეტატორის ველში აკრეფ-ილი არგუმენტების რაოდენობას, ხოლო მეორე - არგუმენტებზე მიმთითებლების მასივს.

როდესაც ბირთვი ასრულებს C ენაზე დაწერილ პროგრამას `main` ფუნქციის გამოძახებამდე ის ასრულებს სპეციალურ პროცედურას. ამ პროცედურის მისამართი ეთითება შესრულებადი პროგრამის ფაილში, როგორც შესვლის წერტილი. პროცედურის მისამართს განსაზღვრავს გამოყენებული კომპილერის მიერ გამოძახებული კავშირების რედაქტორი. წინასწარი ჩატვირთვის პროცედურა ბირთვისგან ღებულობს ბრძანებათა ველის პარამეტრებს და ცვლადი გარემოს მნიშვნელობებს. მხოლოდ ამის შემდეგ ხდება მიმართვა `main` ფუნქციაზე.

პროგრამა გარდა ბრძანებათა ინტერპრეტატორის პარამეტრებისა ასევე იღებს ცვლადი გარემოს პარამეტრების ჩამონათვალს. ბრძანებათა ინტერპრეტატორის პარამეტრების მსგავსად ცვლადი გარემოს პარამეტრებიც წარმოადგენენ მიმთითებელთა მასივს, რომელთაგან თითოეული უთითებს სიმბოლურ სტრიქონზე. სისტემაში არსებული გლობალური ცვლადით - `environ`, შესაძლებელია ცვლადი გარემოს პარამეტრების მნიშვნელობების მითითება. ნახ. 2.2-ზე გამოსახულია 5 პარამეტრიანი ცვლადი გარემოს მაგალითი.



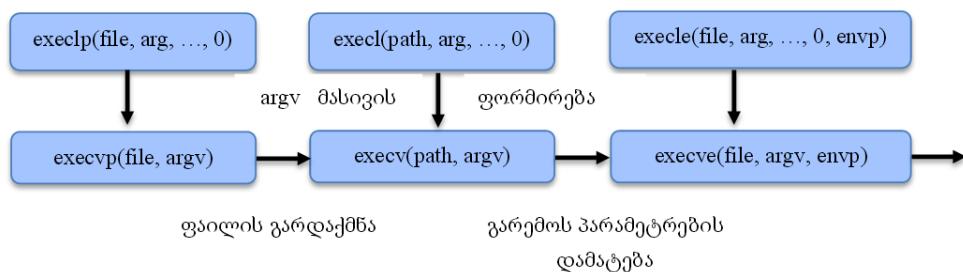
ცვლადი გარემოს პარამეტრების მისათითებლად main ფუნქცია შეიცავდა მესამე არგუმენს. სამ არგუმენტიანი main ფუნქცია გამოიყენებოდა უმეტეს UNIX მსგავს სისტემაში. ამ შემთხვევაში main ფუნქციის პროტოტიპს გააჩნია სახე

```
int main(int argc, char *argv[], char *envp[]);
```

ISO C სტანდარტიზაციით მიღებული შეთანხმების შედეგად main ფუნქციაში არ გამოიყენება მესამე არგუმენტი და ცვლად გარემოს პარამეტრების გადაცემა ხორციელდება გლობალური environ ცვლადის გამოყენებით. ცვლადი პარამეტრების მნიშვნელობების ასეთნაირად გადაცემის მეთოდი არაფრით განსხვავდება main ფუნქციის მესამე არგუმენტით იმავე მნიშვნელობების გადაცემისაგან. ამის გამო, POSIX სტანდარტშიც შენარჩუნებულია ორ არგუმენტიანი main ფუნქცია.

2.8. პროცესის მომხმარებლის კონტექსტის შეცვლა

პროცესის მომხმარებლის კონტექსტის შესაცვლელად გამოიყენება სისტემური გამოძახება exec(), რომლის გამოძახებაც უშუალოდ არ შეუძლია მომხმარებელს. exec() სისტემური გამოძახება ცვლის მიმდინარე პროცესის მომხმარებლის კონტექსტს რომელიმე შესრულებადი ფაილის შემცველობით და აყენებს პროცესორის რეგისტრების საწყის მნიშვნელობას (მათ შორის აყენებს პროგრამულ მთვლელს ჩასატვირთი პროგრამის



ნახ. 2.3. exec() სისიტემური გამოძახების შესასრულებლად სხვადასხვა ფუნქციების ურთიერთკავშირი

საწყისზე). ეს სისტემური გამოძახება საკუთარი მუშაობისათვის მოითხოვს შესასრულებელი ფაილის სახელის, ბრძანებათა ველის არგუმენტების და გარემოს პარამეტრების მოცემას. სისტემური გამოძახების განსახორციელებლად პროგრამისტს შეუძლია გამოიყენოს ერთერთი შემდეგი ფუნქციებიდან: execvp(), execcl(), execve(), exec(), execle(),

`execve()`, რომლებიც ერთმანეთისაგან განსხვავდებიან `exec()` სისტემური გამოძახების მუშაობისათვის აუცილებელი პარამეტრების მიწოდების მეთოდებით. მითითებული ფუნქციების ურთიერთკავშირი გამოსახულია ნახ. 2.3-ზე.

პროცესის მომხმარებლის კონტექსტის შეცვლის ფუნქციის პროტოტიპი

```
#include<unistd.h>
int execcl(const char *file, const char *arg0, ..., const char *argN, (char *) NULL);
int execvp(const char *file, char *argv[]);
int execl(const char *path, const char *arg0,... , const char *argN, (char *) NULL);
int execv(const char *path, char *argv[]);
int execle(const char *path, constchar *arg0, ... , const char *argN, (char *) NULL, char *envp[]);
int execve(const char *path, char *argv[], char *envp[]);
```

ფუნქციის აღწერა

ახალი პროგრამის ჩასატვირთად მიმდინარე პროცესის სისტემურ კონტექსტში გამოიყენება ურთიერთდაკავშირებული ფუნქციების ოჯახი, რომლებიც ერთმანეთისაგან განსხვავდებიან პარამეტრების წარდგენის ფორმით. პარამეტრი

- file უთითებს იმ ფაილის სახელს, რომელიც უნდა იქნას ჩატვირთული;
- path უთითებს ჩასატვირთ ფაილამდე სრულ გზას.
- arg0, ..., argN წარმოადგენს ბრძანებათა ველის არგუმენტებზე მიმთითებლებს. შევნიშნოთ, რომ arg0 პარამეტრი უნდა უთითებდეს ჩასატვირთი ფაილის სახელზე.
- argv წარმოადგენს ბრძანებათა ველის პარამეტრებზე მიმთითებლების მასივს. მასივის საწყისი ელემენტი უნდა უთითებდეს ჩასატვირთი პროგრამის სახელზე, ხოლო მასივი უნდა მთავრდებოდეს ელემენტით, რომელიც შეიცავს მიმთითებელს NULL.
- envp არგუმენტი წარმოადგენს ცვლადი გარემოს პარამეტრებზე მიმთითებლების მასივს, რომელიც მოცემულია სტრიქონის სახით „ცვლადი = სტრიქონი“. ამ მასივის ბოლო ელემენტი უნდა შეიცავდეს მიმთითებელს NULL.

ვინაიდან ფუნქციის გამოძახება არ ცვლის მიმდინარე პროცესის სისტემურ კონტექსტს, ჩატვირთული პროგრამა მისი ჩამტვირთავი პროცესისაგან მიიღებს შემდეგ ატრიბუტებს:

- პროცესის იდენტიფიკატორი;
- მშობელი-პროცესის იდენტიფიკატორი;
- პროცესის ჯალფის იდენტიფიკატორი;
- სეანსის იდენტიფიკატორი;
- დრო, SIGALRM სიგნალის წარმოქმნამდე;
- მიმდინარე სამუშო დირექტორია;
- ფაილების შექმნის ნიღაბი;
- მომხმარებლის იდენტიფიკატორი;
- მომხმარებლის ჯგუფის იდენტიფიკატორი;
- სიგნალების ცხადი იგნორირება;
- ღია ფაილების ცხრილი;

გამოძახების განმახორციელებელი ფუნქციიდან პროგრამაში მნიშვნელობის დაბრუნება არ ხდება შესრულების წარმატებულად დასრულების შემთხვევაში და მართვა გადაეცემა ჩატვირთულ პროგრამას. წარუმატებელი შესრულების შემთხვევაში გამოძახების მაინიცირებელ პროგრამაში ბრუნდება უარყოფითი მნიშვნელობა

ვინაიდან პროცესის სისტემური კონტექსტი `exec()`-ის გამოძახებისას პრაქტიკულად

უცვლელი რჩება, მომხმარებლისათვის სისტემური გამოძახების მეშვეობით დაშვებული პროცესის ატრიბუტების უმეტესი (UID, GID, PID, PPID და სხვა) ახალი პროგრამის ამუშავების შემდეგაც არ იცვლება.

მნიშვნელოვანია fork() და exec() სისტემურ გამოძახებებს შორის განსხვავების დაჭერა. fork() წარმოქმნის ახალ პროცესს, რომლის მომხმარებლის კონტექსტიც ემთხვევა მომხმარებლის მშობელი-პროცესის კონტექსტს. სისტემური გამოძახება exec() ცვლის მიმდინარე პროცესის მომხმარებლის კონტექსტს, მაგრამ არ წარმოქმნის ახალ პროცესს.

2.9. მაგალითები

- fork სისტემური გამოძახების გამოყენებით შევქმნათ ახალი პროცესი. მშობელი პროცესი დაელოდოს შვილი პროცესის დასრულებას და შემდეგ გააგრძელოს შესრულება.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

int main(int argc, char *argv[]){
    pid_t a;           // ცვლადი fork-ის მიერ დაბრუნებული მნიშვნელობისათვის
    int status;        // ცვლადი შვილი პროცესის მდგომარეობისათვის

    if ((a = fork()) == -1) {
        perror("პროცესი არ შეიქმნა\n");
        exit(EXIT_FAILURE5);
    }

    if (a == 0){       // შვილი პროცესი
        printf("შვ-იდენტ %d, მშ-იდენტ. %d\n", (int) getpid(), (int) getppid());
    } else {          // მშობელი პროცესი
        wait(&status); // მოხდა მშობელი პროცესის შეჩერება შვილი დასრულებამდე
        printf("\n მშობელი პროცესი აგრძელებს შესრულებას\n\n");
        printf("შვ-იდენტ %d, მშ-იდენტ. %d\n", (int) getpid(), (int) getppid());
    }

    exit(EXIT_SUCCESS);
}
```

prog.2.1.c

- exec() სისტემური გამოძახების გამოყენებით დავგეგმოთ პროცესის საქმიანობა: შვილმა პროცესი დააკომპილიროს წინა პროგრამა (ფაილი prog.2.1.c), ხოლო მშობელმა პროცესმა კი შეასრულოს კომპილაციის შედეგის ფაილს (prog.2.1.out).

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>

int main(int argc, char *argv[]){
    int status;           // ცვლადი შვილი პროცესის სტატუსის განსასაზღვრავად
```

⁵ EXIT_FAILURE და EXIT_SUCCESS არის stdlib.h ბიბლიოთეკაში განსაზღვრული სკეციალური ცვლადებია, რომლებიც შესაბამისად ღებულობენ რაიმე არანულოვან ან ნულოვან მნიშვნელობას

```

pid_t a = fork();
if (a == -1){
    perror("პროცესი არ შეიქმნა\n");
    exit(EXIT_FAILURE);
}
if (a != 0){ // მშობელი პროცესი
    wait(&status);      // მშობელი ელოდება შვილის დასრულებას
    // მშობელი-პროცესი ახდენს 2.1.out ფაილის შესრულებას, რომელიც
    // წარმოადგენს 2.1.c ფაილის კომპილაციის შედეგს
    execl ("prog.2.1.out", "./2.1.out", NULL);
} else { // შვილი-პროცესი ასრულებს 2.1.c ფაილის კომპილაციას
    execl ("./bin/gcc", "gcc", "prog.2.1.c", "-o", "prog.2.1.out", NULL);
}
exit(EXIT_SUCCESS);
}

```

prog.2.2.c

3. დაწერეთ პროგრამა რომელშიც მშობელ პროცესს ეყოლება 2 შვილი პროცესი და ერთერთ მათგანს ეყოლება საკუთარი შვილი პროცესი. "შვილიშვილმა" განახორციელოს სამუშაო დირექტორიაში 3 ქვედირექტორიის შექმნა, ხოლო მეორე შვილმა განახორციელოს ამ დირექტორიიებიდან ორის მესამეში გადატანა

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>
int main(int argc, char *argv[]){
    // ცვლადები პროცესის მიმდინარე სტატუსისათვის
    int st1, st2, st3;
    // ცვლადების პროცესების შექმნის შესამოწმებლად
    pid_t pro1, pro2, pro3;
    // შექმნათ პირველი შვილი პროცესი
    if ((pro1 = fork()) == -1){
        printf("პირველი პროცესი არ შეიქმნა\n");
        exit(EXIT_FAILURE);
    } else { // პირველი პროცესი შეიქმნა
        if (pro1 == 0) { // შვილმა უნდა შექმნას საკუთარი შვილი
            if ((pro2 = fork()) == -1){ // შვილიშვილი არ წარმოიქმნა
                printf("შვილიშვილი პროცესი არ წარმოიქმნა\n");
                exit(EXIT_FAILURE);
            } else { // შვილიშვილი შეიქმნა. დავგეგმოთ მისი სამუშაო
                if (pro2 == 0) {
                    printf("შვილიშვილი პროცესი\n");
                    execl(argv[1], argv[2], argv[3], argv[4], argv[5], NULL);
                } else { // დავგეგმოთ პირველი შვილი პროცესის სამუშაო
                    printf("პირველი პროცესი\n");
                    // პირველი პროცესი ვაიძულოთ დაელოდოს შვილის დასრულებას
                }
            }
        }
    }
}

```

```

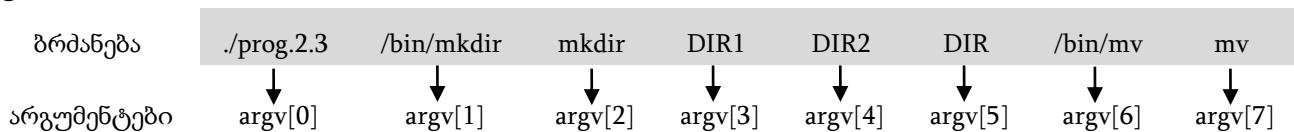
        wait(&st3);
    }
}

} else { // მშობელი პროცესი. ვაიძულოთ ის დაელოდოს პირველი შვილის
// დასრულებას და შემდეგ შექმნას მეორე შვილი პროცესი
wait(&st2);
// შევქმნათ მეორე შვილი პროცესი
if ((pro3 = fork()) == -1) {
    printf("მეორე შვილი პროცესი არ შეიქმნა\n");
    exit(EXIT_FAILURE);
}
else { // მეორე შვილი პროცესი შეიქმნა
if (pro3 == 0){ // მეორე შვილი პროცესი
    // დავგეგმოთ მეორე შვილი პროცესის სამუშაო
    printf("მეორე შვილი პროცესი\n");
    exec(argv[6], argv[7], argv[3], argv[4], argv[5], NULL);
}
else { // დავგეგმოთ მშობელი პროცესის სამუშაო
    // მშობელი ვაიძულოთ დაელოდოს მეორე შვილის დასრულებას
    printf("მშობელი პროცესი\n");
    wait(&st1);
}
}
}
exit(EXIT_SUCCESS);
}

```

prog.2.3.c

პროგრამაში გამოყენებული `argv[1]`, `argv[2]` და ა.შ. პარამეტრები უთითებენ კლავიატურიდან შეტანილ მნიშვნელობებს. კერძოდ, რადგანაც ჩვენ ჯერ გვინდა შევქმნათ რამდენიმე დირექტორია, ხოლო შემდეგ მოვახდინთ მათი გადაადგილება ერთერთ დირექტორიაში, ამიტომ პროგრამის შესასრულებლად ბრძანების შესრულება უნდა მოხდეს შემდეგი ფორმით



სემინარი 3. პროცესების დაგეგმვა

გამოთვლითი სისტემა აღჭურვილია მრავალი რესურსით. პროცესი, რომელიც წარმოიქმნება ოპერაციულ სისტემაში მიმართავს მას მოხვენით შესასრულებლად საჭირო რესურსების გამოყოფაზე. ოპერაციული სისტემის ყველაზე მნიშვნელოვან რესურსს წარმოადგენს პროცესორი, რომელიც თანაბრად უნდა გამოეყოს სისტემაში წარმოქმნილ ყველა პროცესს. იმისათვის, რომ ოპერაციულმა სისტემამ სწორად განახორციელოს პროცესორის გამოყენება ის მიმართავს მასში არსებულ პროგრამას ე.წ. **პროცესორის დამგეგმავს**, რომელიც განსაზღვრავს თუ რა თანმიმდევრობით უნდა მოხდეს პროცესებისათვის პროცესორის გამოყოფა.

ყოველი პროცესი, რომელიც იქმნება პროგრამის შესრულების მომენტში, შესასრულებლად საჭიროებს გარკვეულ დროით შუალედს. ოპერაციულ სისტემაში წარმოქმნისას შესაძლებელია ეს დროითი შუალედი არ იყოს საკმარისი პროცესის რეალური დასრულებისათვის. შეიძლება ამის არსებობდეს სხვადასხვა მიზეზები: სისტემის დატვირთულობა, გამოყენებული ალგორითმი და ა.შ. ოპერაციული სისტემის მთავარ ამოცანას წარმოადგენს მაქსიმალურად მიაახლოვოს პროცესის შესასრულებლად საჭირო დრო იმ დროსთან¹, რომელიც დასჭირდება მას სისტემაში წარმოქმნის მომენტიდან დასრულებამდე. ამ მიზნით გამოყენებული ალგორითმი მით უფრო კარგია რაც უფრო ნაკლების ზემოხსენებულ ორ დროს შორის სხვაობა.

ოპერაციული სისტემების არსებობის პირველი დღიდან შეიქმნა პროცესების შესრულების დაგეგმვის მრავალი ალგორითმი. ნაწილი ამ ალგორითმებიდან დღეს არ გამოიყენება და წარმოადგენს ისტორიის საკუთრებას, მეორე ნაწილმა კი განიცადა გარკვეული ცვლილება და აქტიურად გამოიყენება ამა თუ იმ სისტემაში (ტრანზაქციების, დროის გაყოფის, ინტერაქტიულ და ა.შ.).

შევნიშნოთ, რომ ოპერაციულ სისტემაში წარმოქმნილი ყოველი პროცესი ძირითადად დაკავებულია ორი ტიპის საქმიანობით: **აქტიური გამოთვლებით და აქტიური შეტანა/გამოტანით**. ორივე სახის საქმიანობის საჭიროების მქონე პროცესების შესრულებისას შეიძლება რამდენჯერმე მოხდეს მათი შეჩერება მონაცემების შეტანა/გამოტანის მიზნით, რაც ზრდის პროცესის შესასრულებლად საჭირო დროს. სიმარტივისათვის ქვემოთ განხილულ ალგორითმებში ვიგულისხმებთ, რომ პროცესი დასრულდება მის შესასრულებლად მითითებულ დროში და ის ერთდროულად არაა დაკავებული ორივე საქმიანობით.

განვიხილოთ ეს ალგორითმები

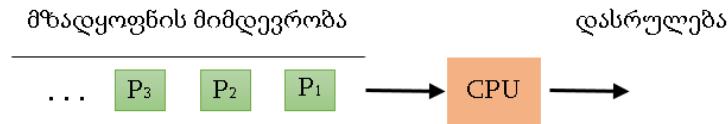
3.1. FCFS (First Come – First Served)

დაგეგმვის ყველაზე მარტივ ალგორითმს წარმოადგენს ალგორითმი FCFS (პირველი

შემოვიდა - პირველი მომსახურდება) (ნახ. 3.1). ამ ალგორითმის გამოყენებისას სისტემაში იქმნება მზაყოფნის ცხრილი, რომელშიც პროცესები თავსდებიან სისტემაში წარმოქმნის თანმიმდევრობით. სისტემაში წარმოქმნილი ყოველი ახალი პროცესი ამ მიმდევრობაში ემატება ბოლოდან. შესრულებას იწყებს ცხრილის თავში მყოფი პირველივე პროცესი.

¹ შესასრულებლად საჭირო პროცესორული დრო

პროცესისათვის პროცესორის გამოყოფა ხდება მის შესასრულებლად საჭირო დროითი შუალედით. პროცესისაგან პროცესორის გამონთავისუფლება მოხდება მხოლოდ მისი დასრულების ან გარკვეული მიზეზით (შეტანა/გამოტანის ოპერაციის ლოდინის გამო) მისი ბლოკირების შემთხვევაში. ბლოკირებული პროცესი სისტემაში მის გასაგრძელებლად საჭირო მოვლენის (მაგალითად, გამონთავისუფლდა შეტანა/გამოტანის მოწყობილობა) დადგომის შემდეგ (თუ ის საკმარისია) გამოდის ბლოკირების მდგომარეობიდან და თავსდება მზადყოფნის ცხრილის ბოლოში.



ნახ. 3.1. დაგეგმვა FCFS პრინციპის გამოყენებით

განვიხილოთ მაგალითი. ვთქვათ პროცესები მოცემულია შემდეგი ცხრილით

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	5	7	3	6	4

შევნიშნოთ, რომ შემდეგში თუ პროცესების შესრულებასთან ერთად არ იქნება მითითებული შესრულების დრო, მაშინ ვიგულისხმებთ, რომ ყველა ჩამოთვლილი პროცესი ერთდროულად გვაქვს სისტემაში. ერთი პროცესორის შემთხვევაში რეალურად ოპერაციულ სისტემაში ერთდროულად შეუძლებელია წარმოიქმნას ერთზე მეტი პროცესი.

რადგანაც ყველა პროცესი გვაქვს სისტემაში, ამიტომ მათი შესასრულებლად არჩევა მოხდება მათი ინდექსების მიხედვით - პირველი P₁ პროცესი, მეორე P₂ პროცესი და ა.შ. რადგანაც FCFS ალგორითმი ყოველ პროცესს გამოუყოფს მის შესასრულებლად საჭირო დროს, ამიტომ პროცესების შესრულებისათვის გვექნება შემდეგი ცხრილი

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	5	12	15	21	25
ლოდინის დრო	0	5	12	15	21

უფრო დეტალიზირებულ ცხრილს ექნება სახე

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
P ₁	+	+	+	+	+																				
P ₂	-	-	-	-	-	+	+	+	+	+	+	+													
P ₃	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+										
P ₄	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+	+	+	+	+				
P ₅	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+	+	

სადაც სიმბოლო '+' აღნიშნავს პროცესორზე პროცესის შესრულების მომენტს, ხოლო სიმბოლო '-' კი ლოდინს. ალგორითმის შესაფასებლად ჩვენ დაგვჭირდება დავითვალოთ ლოდინის („მინუსიების რაოდენობა“) და შესასრულებლად საჭირო (მისი შექმნიდან დასრულებამდე) დროების საშუალო.

ამრიგად, ლოდინის და შესრულების დროისათვის შესაბამისად გვექნება:

$$\text{ლოდინის საშუალო არის } \rightarrow (0+5+12+15+21)/5 = 10.6$$

$$\text{შესრულების დროის საშუალო } \rightarrow (5+12+15+21+25)/5 = 15.6$$

ანუ თითოეულ პროცესს პროცესორის მისაღებად საშუალოდ მოუწია 10.6 დროითი ერთეულით ლოდინი, ხოლო მის შესრულებას დასჭირდა 15.6 დროითი ერთეული.

თუ განხილულ ამოცანაში შემოვიღებთ პროცესის გამოჩენის (შექმნის) დროს, მაშინ ამოცანა უმნიშველოს გართულდება. ვთქვათ, პროცესების ცხრილს აქვს სახე

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	5	7	3	6	4
გამოჩენის დრო	1	4	0	7	2

რადგანაც ამ შემთხვევაში ჩვენი ცხრილი დამატებით შეიცავს პროცესების გამოჩენის დროს, ამიტომ პროცესები მზადყოფნის ცხრილში ადგილს დაიკავებენ გამოჩენის დროის მიხედვით. პირველი P₃ პროცესი, მეორე P₁ პროცესი და ა.შ. ამ მიმდევრობით მოხდება მათი შესრულებაც.

პროცესების შესრულების დეტალიზირებულ ცხრილს ექნება სახე

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
P ₁		-	-	+	+	+	+	+																	
P ₂					-	-	-	-	-	-	-	-	+	+	+	+	+	+	+						
P ₃	+	+	+																						
P ₄						-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+	+	+	+	
P ₅			-	-	-	-	-	-	+	+	+	+													

ამრიგად, ლოდინის და შესრულების დროისათვის შესაბამისად გვექნება:

$$\text{ლოდინის საშუალო } \text{არის} \rightarrow (2+8+0+12+6)/5 = 5.6$$

$$\text{შესრულების დროის საშუალო } \rightarrow (7+15+3+18+10)/5 = 8.6$$

ანუ თითოეულ პროცესს პროცესორის მისაღებად საშუალოდ მოუწია 5.6 დროითი ერთეულით ლოდინი, ხოლო მის შესრულებას დასჭირდა 8.6 დროითი ერთეული.

როგორც ვხედავთ ამ შემთხვევაში მიიღწევა უკეთესი შედეგი.

დამოუკიდებელი სამუშაო

- დაგეგმვის FCFS ალგორითმის გამოყენებით დაითვალეთ პროცესების შესრულების და ლოდინის დროის საშუალო, თუ ყველა პროცესი სისტემაში არსებობს ერთდროულად.

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	6	1	5	2	8

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	16	7	8	2	1

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	8	10	7	2	15

- დაგეგმვის FCFS ალგორითმის გამოყენებით დაითვალეთ პროცესების შესრულების და ლოდინის დროის საშუალო, თუ პროცესები სისტემაში გამოჩნდნენ სხვადასხვა დროს:

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	5	7	3	2	8
გამოჩენის დრო	0	2	10	5	1

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	16	7	8	2	1
გამოჩენის დრო	2	0	3	7	5

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	8	10	7	2	15
გამოჩენის დრო	2	3	0	5	10

3.2. SJF (Short Job First)

როგორც ზემოთ განხილული ამოცანების შემთხვევაში ვნახეთ, ოპერაციულ სისტემაში შესასრულებლად ნაკლები დროის საჭიროების მქონე პროცესს შეიძლება მოუწიოს შესასრულებლად დიდი დროით ლოდინი. თუ შესასრულებლად ნაკლები დროის საჭიროების პროცესები შესრულდებოდნენ თავიდან და შემდეგ მეტი დროის საჭიროების მქონე პროცესები მაშინ, ცხადია, რომ ლოდინის და შესრულების დროების საშუალო უფრო ნაკლები იქნებოდა.

ოპერაციულ სისტემისათვის ხშირად უცნობია პროცესების შესასრულებლად საჭირო დრო. თუ იარსებებდა პროცესების შესასრულებლად საჭირო დროის დადგენის შესაძლებლობა რაიმე მექანიზმით, მაშინ პროცესების დაგეგმვის ამოცანა ბევრად გამარტივდებოდა. SJF ალგორითმი (პირველი მოკლე ამოცანა) აგებულია ამ პრინციპით. SJF ალგორითმის გამოყენება გულისხმობს, რომ წინასწარ ცნობილია პროცესების შესრულების დრო და მასში გამოყენებულ მზადყოფნის ცხრილში პროცესები განთავსებულია მათი შესრულების დროის მიხედვით - ნაკლები პროცესორული დროის საჭიროების მქონე პროცესი ცხრილის თავში. დამგეგმავის მიერ შესასრულებლად პირველი აირჩევა ცხრილის თავში განთავსებული პროცესი. სისტემაში წარმოქმნილი ყოველი პროცესი ფასდება მისი შესრულების დროით და მზადყოფნის ცხრილში შესრულების დროის მიხედვით თავსდება შესაბამის ადგილას.

განვიხილოთ მაგალითი:

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	5	7	3	6	4

რადგანაც ყველა პროცესი თავიდანვე გვაქვს სისტემაში, ამიტომ დამგეგმავის მიერ მათი შესარულებლად არჩევა მოხდება მათი შესრულების დროის მიხედვით - პირველი P₃ პროცესი, მეორე P₅ პროცესი და ა.შ. რადგანაც SJF ალგორითმი ყოველ პროცესს გამოუყოფს მის შესასრულებლად საჭირო დროს, ამიტომ პროცესების შესრულებისათვის გვექნება შემდეგი დეტალიზირებული ცხრილი

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
P ₁	-	-	-	-	-	-	-	+	+	+	+	+													
P ₂	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+	+	+	+	
P ₃	+	+	+																						
P ₄	-	-	-	-	-	-	-	-	-	-	-	+	+	+	+	+	+	+							
P ₅	-	-	-	+	+	+	+																		

ამრიგად, ლოდინის და შესრულების დროისათვის შესაბამისად გვექნება:

$$\text{ლოდინის საშუალო არის } \rightarrow (7+18+0+12+3)/5 = 8$$

$$\text{შესრულების დროის საშუალო } \rightarrow (12+25+3+18+7)/5 = 13$$

ანუ თითოეულ პროცესს პროცესორის მისაღებად საშუალოდ მოუწია 8 დროითი ერთეულით ლოდინი, ხოლო მის შესრულებას დასჭირდა 13 დროითი ერთეული.

როგორც ვხედავთ SJF ალგორითმი FCFS ალგორითმთან შედარებით (თუ პროცესები თავიდანვე იარსებებენ სისტემაში) იძლევა უკეთეს შედეგს.

თუ განხილულ ამოცანას დავამატებთ გამოჩენის დროს ის გარკვეულწილად გართულდება. ვთქვათ, გვაქვს პროცესების შემდეგი ცხრილი

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	5	7	3	6	4
გამოჩენის დრო	1	4	0	7	2

რადგანაც ამ შემთხვევაში ჩვენი ცხრილი დამატებით შეიცავს პროცესების გამოჩენის დროს, ამიტომ პროცესები მზადყოფნის ცხრილში ადგილს დაიკავებენ შესრულების დროის მიხედვით. პირველი P₃ პროცესი. მიუხედავად იმისა, P₁ პროცესი სისტემაში შეიქმნა P₅ პროცესზე ადრე ის შესასრულებლად საჭიროებს P₅ პროცესის შესასრულებლად საჭირო დროზე მეტს, ამიტომ მეორე სისტემაში შესრულდება P₅ პროცესი და ა.შ.

პროცესების შესრულების დეტალიზირებულ ცხრილს ექნება სახე:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
P ₁		-	-	-	-	-	-	+	+	+	+	+													
P ₂					-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+	+	+	+	
P ₃	+	+	+																						
P ₄							-	-	-	-	-	-	+	+	+	+	+	+							
P ₅			-	+	+	+	+	+																	

ამრიგად, ლოდინის და შესრულების დროისათვის შესაბამისად გვექნება:

$$\text{ლოდინის საშუალო } \text{არის} \rightarrow (6+14+0+5+1)/5 = 5.2$$

$$\text{შესრულების დროის საშუალო } \rightarrow (11+21+3+11+5)/5 = 10.2$$

ანუ თითოეულ პროცესს პროცესორის მისაღებად საშუალოდ მოუწია 5.2 დროითი ერთეულით ლოდინი, ხოლო მის შესრულებას დასჭირდა 10.2 დროითი ერთეული.

დამოუკიდებელი სამუშაო

1. დაგეგმვის SJF ალგორითმის გამოყენებით დაითვალეთ პროცესების შესრულების და ლოდინის დროის საშუალო, თუ ყველა პროცესი სისტემაში არსებობს ერთდროულად:

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	6	7	1	2	8

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	5	10	7	2	3

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	16	7	15	2	1

2. დაგეგმვის SJF ალგორითმის გამოყენებით დაითვალეთ პროცესების შესრულების და ლოდინის დროის საშუალო, თუ პროცესები სისტემაში გამოჩენდნენ სხვადასხვა დროს:

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	5	7	3	2	8
გამოჩენის დრო	0	2	2	5	1

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	8	10	7	2	15
გამოჩენის დრო	2	3	0	5	0

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	16	7	8	2	1
გამოჩენის დრო	2	0	3	7	5

3.3. პრიორიტეტული SJF

პრიორიტეტის შემოღებით სისტემაში შესაძლებელია პროცესების დაყოფა პრიორიტეტების ჯგუფების მიხედვით. პრიორიტეტის მნიშვნელობა შეიძლება იყოს გარკვეული მთელი რიცხვითი მნიშვნელობა. ის უპირატესობაში აყენებს გარკვეულ პროცესებს სხვა პროცესებთან მიმართებაში. პრიორიტეტის დანიშვნა შესაძლებელია მოხდეს ორი მეთოდით: სტატიკურად და დინამიურად. სტატიკური პრიორიტეტი ინიშნება პროგრამისტის ან სისტემური ადმინისტრატორის მიერ. მისი შეცვლა პროცესის დასრულებამდე შეუძლებელია. დინამიური პრიორიტეტი ინიშნება სისტემის მიერ გამოყენებული ალგორითმით და ის შესაძლებელია გარკვეული მოვლენის დადგომის შემდეგ (მაგალითად, პროცესი დასრულდა) შეიცვალოს.

პრიორიტეტის გამოყენებელი ალგორითმი შეიძლება იყოს ორი სახის: ალგორითმი პრიორიტეტული გაძვებით და ალგორითმი პრიორიტეტული გაძვების გარეშე. პრიორიტეტული გაძვების გარეშე (გაუმევებელი) ალგორითმის გამოყენებისას სისტემაში ახალი მაღალპრიორიტეტული პროცესის წარმოქმნის შემთხვევაში მიმდინარე პროცესი, რომელსაც დაკავებული აქვს პროცესორი და სრულდება (მიუხედავად იმისა როგორია მისი პრიორიტეტი წარმოქმნილ პროცესთან მიმართებაში), აგრძელებს შესრულებას და მხოლოდ მისი დასრულების შემდეგ შეეძლება მაღალპრიორიტეტულ პროცესს შესრულება. პრიორიტეტული გაძვებით (გაძვებადი) ალგორითმის გამოყენებისას სისტემაში ახალი მაღალპრიორიტეტული პროცესის წარმოქმნის შემთხვევაში მიმდინარე პროცესი, რომელსაც დაკავებული აქვს პროცესორი და სრულდება (თუ მისი პრიორიტეტი წარმოქმნილ პროცესთან მიმართებაში დაბალია), იქნება შეჩერებული სისტემის მიერ და შესრულებას დაიწყებს მაღალპრიორიტეტულ პროცესი.

SJF ალგორითმი შეიძლება იყოს როგორც გაძვებადი ისე გაუმევებელი. მასში პრიორიტეტი შეიძლება განსაზღვრული იყოს როგორც სტატიკურად (გარკვეული რიცხვითი მნიშვნელობის მინიჭებით), ასევე დინამიურად (შესრულების დროის მიხედვით²).

განვიხილოთ მაგალითები.

1. გაუმევებელი SJF ალგორითმი სტატიკური პრიორიტეტით (გამოჩენის დროის გარეშე)

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	5	7	3	6	4
პრიორიტეტი	0	3	4	2	1

შევნიშნოთ, რომ ამ შემთხვევაში ალგორითმი გაძვებადი იქნება თუ გაუმევებელი არსებითი მნიშვნელობა არ გააჩნია, ვინაიდან ყველა პროცესი თავიდანვე გვაქვს სისტემაში.

რადგანაც გვაქვს პრიორიტეტული ალგორითმი, ამიტომ ამ შემთხვევაში მათი შესრულება მოხდება პრიორიტეტის მნიშვნელობის მიხედვით. თუ გვექნებოდა ერთნაირი პრიორიტეტის მქონე ორი ან მეტი პროცესი, მაშინ ფიქსირებული პრიორიტეტის ფარგლებში უპირატესობა მიენიჭებოდა მათი შესრულების დროს. განსახილველ მაგალითში პრიორიტეტის მნიშვნელობა არ მეორდება, ამიტომ ისინი შესრულდებიან ერთიმეორის მიყოლებით პრიორიტეტის მნიშვნელობის მიხედვით. შესაბამისად პროცესები შესასრულებ-

² ნაკლები შესრულების დრო მაღალი პრიორიტეტი

ლად არჩეული იქნება მიმდევრობით პირველი P_1 პროცესი, მეორე P_5 პროცესი და ა.შ., ხოლო პროცესების შესრულებისათვის გვექნება შემდეგი დეტალიზირებული ცხრილი

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
P_1	+	+	+	+	+																				
P_2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+	+	+	+	+				
P_3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+	
P_4	-	-	-	-	-	-	-	-	-	+	+	+	+	+	+										
P_5	-	-	-	-	-	+	+	+	+																

ამრიგად, ლოდინის და შესრულების დროისათვის შესაბამისად გვექნება:

$$\text{ლოდინის საშუალო არის } \rightarrow (0+15+22+9+5)/5 = 10.2$$

$$\text{შესრულების დროის საშუალო } \rightarrow (5+22+25+15+9)/5 = 15.2$$

ანუ თითოეულ პროცესს პროცესორის მისაღებად საშუალოდ მოუწია 10.2 დროითი ერთეულით ლოდინი, ხოლო მის შესრულებას დასჭირდა 15.2 დროითი ერთეული.

2. გაუძევებელი SJF ალგორითმი სტატიკური პრიორიტეტით (გამოჩენის დროით) ვთქვათ, გვაქვს პროცესების შემდეგი ცხრილი

პროცესი	P_1	P_2	P_3	P_4	P_5
შესრულების დრო	5	7	3	6	4
გამოჩენის დრო	1	4	0	7	2
პრიორიტეტი	0	3	4	2	1

რადგანაც ალგორითმი გაუძევებელია, ამიტომ პროცესი, რომელიც დაიკავებს პროცესორს არ გამოანთავისუფლებს მას დასრულებამდე, მიუხედავად იმისა მისი შესრულების მომენტში გამოჩნდა თუ არა მაღალპრიორიტეტული პროცესი. პროცესის დასრულების შემდეგ პროცესორს იკავებს მოცემული მომენტისათვის არსებული მაღალპრიორიტეტული პროცესი. ამრიგად, პროცესები შესრულდებიან მიმდევრობით: რადგანაც სისტემაში პირველი გამოჩნდა P_3 პროცესი, ამიტომ ის დაიწყებს შესრულებას და პროცესორს არ გამოანთავისუფლებს სანამ არ დასრულდება. მისი დასრულების (3 დროითი ერთეულის) შემდეგ სისტემაში უკვე გვაქვს ორი პროცესი (P_1 და P_5). რადგანაც P_1 პროცესს აქვს მაღალი პრიორიტეტი (0), ვიდრე P_5 პროცესს (1), ამიტომ შესრულებას დაიწყებს P_1 პროცესი. და ა.შ.

პროცესების შესრულების დეტალიზირებულ ცხრილს ექნება სახე

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
P_1	-	-	+	+	+	+	+	+																	
P_2				-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+	+	+		
P_3	+	+	+																						
P_4							-	-	-	-	-	-	+	+	+	+	+	+							
P_5			-	-	-	-	-	-	+	+	+	+	+	+	+										

ამრიგად, ლოდინის და შესრულების დროისათვის შესაბამისად გვექნება:

$$\text{ლოდინის საშუალო არის } \rightarrow (2+14+0+5+6)/5 = 5.4$$

$$\text{შესრულების დროის საშუალო } \rightarrow (7+21+3+11+10)/5 = 8.6$$

ანუ თითოეულ პროცესს პროცესორის მისაღებად საშუალოდ მოუწია 5.4 დროითი ერთეულით ლოდინი, ხოლო მის შესრულებას დასჭირდა 8.6 დროითი ერთეული.

3. გაძევებადი SJF ალგორითმი სტატიკური პრიორიტეტით (გამოჩენის დროით) ვთქვათ, გვაქვს პროცესების შემდეგი ცხრილი

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	5	7	3	6	4
გამოჩენის დრო	1	4	0	7	2
პრიორიტეტი	0	3	4	2	1

რადგანაც ალგორითმი გაძევებადია, ამიტომ პროცესი, რომელიც დაიკავებს პროცესორს გამოანთავისუფლებს მას (დასრულებამდე) მხოლოდ სისტემაში მაღალ-პრიორიტეტული პროცესის გამოჩენის შემთხვევაში. პროცესორს დაიკავებს მაღალ-პრიორიტეტული პროცესი და დაიწყებს შესრულებას. ამრიგად, პროცესები შესრულდებიან მიმდევრობით: რადგანაც სისტემაში პირველი გამოჩნდა P₃ პროცესი, ამიტომ ის დაიწყებს შესრულებას. 1 დროითი ერთეულის შემდეგ სისტემაში გამოჩნდება პროცესი P₁, რომელსაც გააჩნია მაღალი პრიორიტეტი. P₃ პროცესი დროებით შეწყვეტს შესრულებას და შესრულებას გააგრძლებს P₁ პროცესი. 2 დროითი ერთეულის შემდეგ სისტემაში გამოჩნდება P₅ პროცესი, რომელსაც P₁ პროცესზე დაბალი პრიორიტეტი გააჩნია, ამიტომ შესრულებას გააგრძელებს P₁ პროცესი და ა.შ.

პროცესების შესრულების დეტალიზირებულ ცხრილს ექნება სახე

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
P ₁	+	+		+	+	+																			
P ₂				-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+	+	+	+	+		
P ₃	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+
P ₄							-	-	-	+	+	+	+	+	+	+									
P ₅			-	-	-	-	-	+	+	+	+	+	+	+	+										

ამრიგად, ლოდინის და შესრულების დროისათვის შესაბამისად გვექნება:

$$\text{ლოდინის საშუალო არის } \rightarrow (0+12+22+3+4)/5 = 8.2$$

$$\text{შესრულების დროის საშუალო } \rightarrow (5+19+25+9+8)/5 = 13.2$$

ანუ თითოეულ პროცესს პროცესორის მისაღებად საშუალოდ მოუწია 8.2 დროითი ერთეულით ლოდინი, ხოლო მის შესრულებას დასჭირდა 13.2 დროითი ერთეული.

4. გაძევებადი SJF ალგორითმი დინამიური პრიორიტეტით (გამოჩენის დროით). პრიორიტეტის როლში აღებული იქნება პროცესისათვის მიმდინარე მომენტში შესასრულებლად საჭირო დრო.

ვთქვათ, გვაქვს პროცესების შემდეგი ცხრილი

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	5	7	3	6	4
გამოჩენის დრო	1	4	0	7	2

რადგანაც პრიორიტეტის როლში აღებულია შესრულების დრო, ამიტომ დროის ყოველ მომენტში შესაძლებელია იცვლებოდეს პროცესის მიმდინარე პრიორიტეტი და პროცესის შესასრულებლად არჩევისას დამგეგმავდა უნდა გაითვალისწინოს ის. ასევე, გაძევებადობის გამო სისტემაში წარმოქმნილი მაღალპრიორიტეტული პროცესი ჩაანაცვლებს შესრულებად დაბალპრიორიტეტულ პროცესს. ამრიგად, პროცესები შესრულდებიან მიმდევრობით: რადგანაც სისტემაში პირველი გამოჩნდა P₃ პროცესი, ამიტომ ის დაიწყებს შესრულებას. მაგალითის მიხედვით პროცესი P₁ შესასრულებლად

საჭიროებს ყველაზე ნაკლებ დროს. ამიტომ ის დაიკავებს პროცესორს და არ გამოანთავი-სუფლებს მას სანამ არ დასრულდება. 3 დროითი ერთეულის შემდეგ სისტემაში გვაქვს ორი პროცესი (P₁ და P₅). შესრულების დროის მიხედვით P₅ პროცესი საჭიროებს ნაკლებ დროს, ამიტომ ის უპირატესია (მაღალპრიორიტეტულია) და დაიწყებს შესრულებას. და ა.შ.

პროცესების შესრულების დეტალიზირებულ ცხრილს ექნება სახე

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
P ₁		-	-	-	-	-	-	+	+	+	+	+													
P ₂					-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+	+	+	+	
P ₃	+	+	+																						
P ₄								-	-	-	-	-	+	+	+	+	+	+							
P ₅		-	+	+	+	+	+																		

ამრიგად, ლოდინის და შესრულების დროისათვის შესაბამისად გვექნება:

$$\text{ლოდინის საშუალო არის } \rightarrow (6+14+0+5+1)/5 = 5.2$$

$$\text{შესრულების დროის საშუალო } \rightarrow (11+21+3+11+5)/5 = 10.2$$

ანუ თითოეულ პროცესს პროცესორის მისაღებად საშუალოდ მოუწია 5.2 დროითი ერთეულით ლოდინი, ხოლო მის შესრულებას დასჭირდა 10.2 დროითი ერთეული.

დამოუკიდებელი სამუშაო

1. დაგეგმვის პრიორიტეტული SJF ალგორითმის გამოყენებით დაითვალეთ პროცესების შესრულების დროის საშუალო, თუ ყველა პროცესი სისტემაში არსებობს ერთდროულად (მაღალ პრიორიტეტს შეესაბამება ნაკლები რიცხვითი მნიშვნელობა):

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	5	7	3	2	8
პრიორიტეტი	1	0	2	2	1

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	16	7	8	2	1
პრიორიტეტი	2	0	3	7	5

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	7	2	1	10	11
პრიორიტეტი	0	1	3	7	5

2. დაგეგმვის პრიორიტეტული SJF ალგორითმის გამოყენებით დაითვალეთ პროცესების შესრულების დროის საშუალო, თუ პროცესები სისტემაში გამოჩნდნენ სხვადასხვა დროს:

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	5	7	3	2	8
გამოჩნდნენის დრო	0	2	8	3	1
პრიორიტეტი	1	0	2	2	1

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	16	7	8	2	1
გამოჩნდნენის დრო	0	5	7	1	2
პრიორიტეტი	2	0	3	7	5

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	7	2	1	10	11
გამოჩნდნენის დრო	1	5	1	0	3
პრიორიტეტი	0	1	3	7	5

3. დაგეგმვის პრიორიტეტული SJF ალგორითმის გამოყენებით დაითვალეთ პროცესების შესრულების დროის საშუალო, თუ პროცესები სისტემაში გამოჩნდნენ სხვადასხვა დროს. პრიორიტეტის როლში აიღეთ პროცესის შესასრულებლად საჭირო

მიმდინარე დრო:

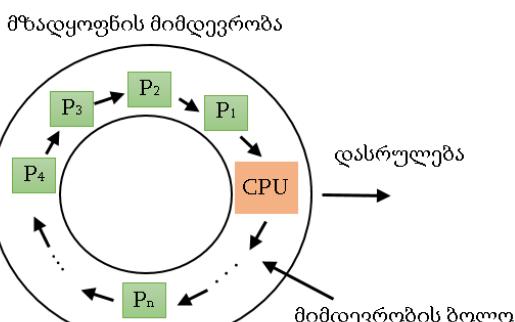
პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	5	7	3	2	5
გამოჩენის დრო	0	3	8	2	1

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	16	7	8	2	1
გამოჩენის დრო	0	5	7	1	4

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	7	2	1	10	11
გამოჩენის დრო	1	5	1	0	3

3.4. ციკლური დაგეგმვა (RR - Round Robin)

ყველა ზემოთ განხილულ ალგორითმში თუ პროცესი მიზანმიმართულად ან უნებლიერ ხელში ჩაიგდებს პროცესორს და ჩაიციკლება, მაშინ იქიდან გამოსვლა საჭიროებს დამატებით მექანიზმს, რომელსაც ალგორითმი არ გულისხმობს. გარდა ამისა, ისინი ახერხებენ მხოლოდ ერთ პროცესთან სანამ ეს უკანასკნელი არ დასრულდება და ნებაყოფლობით არ დაუთმობს პროცესორს სხვა პროცესს.



ნახ. 3.2. დაგეგმვა RR ალგორითმის გამოყენებით

პროცესორის დაგეგმვის ერთერთ მველ, მარტივ, სამართლიან და ხშირად გამოყენებად ალგორითმს წარმოადგენს ციკლური დაგეგმვის ალგორითმი (ნახ. 3.2), რომელიც გულისხმობს პროცესებისათვის პროცესორის გამოყოფას გარკვეული დროითი შუალედით (დროითი კვანტით). ალგორითმში მხარდაჭერილია მზადყოფნის ერთი ცხრილი, რომელშიც მზადყოფნის მდგომარეობაში მყოფი პროცესები ემატება ბოლოდან.

პროცესი, რომელსაც შესასრულებლად არ ეყო დროითი კვანტი, იძულებით ჩამოერთმევა პროცესორი და ის გადაეცემა შემდეგ შესასრულებელ პროცესს, ხოლო მიმდინარე პროცესი გადაინაცვლებს მზადყოფნის ცხრილის ბოლოში. თუ პროცესი ჩაეტია დროით კვანტში, მაშინ ის ნებაყოფლობით აბრუნებს პროცესორს, რომელიც გადაეცემა შემდეგ შესასრულებელ პროცესს. პროცესორის გამოყოფა ხდება ციკლურად პროცესებისათვის სანამ მზადყოფნის ცხრილში არსებობს ერთი მაინც პროცესი.

განვიხილოთ მაგალითი. ვთქვათ, დროითი კვანტი 2-ის ტოლია.

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	5	7	3	6	4

რადგანაც ყველა პროცესი გვაქვს სისტემაში, ამიტომ დამგეგმავის მიერ მათი შესასრულებლად არჩევა მოხდება მათი მიმდევრობით დროითი კვანტით 2 - პირველი P₁ პროცესი, მეორე P₂ პროცესი და ა.შ.

პროცესების შესრულებისათვის გვექნება შემდეგი დეტალიზირებული ცხრილი

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
P ₁	+	+	-	-	-	-	-	-	-	-	+	+	-	-	-	-	-	-	-	+					
P ₂	-	-	+	+	-	-	-	-	-	-	-	-	+	+	-	-	-	-	-	-	+	+	+		
P ₃	-	-	-	-	+	+	-	-	-	-	-	-	-	-	+										

P ₄	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	-	-	-	-	+	+	+
P ₅	-	-	-	-	-	-	-	-	+	+	+	-	-	-	-	-	-	-	-	-	-	+	+	-

ამრიგად, ლოდინის და შესრულების დროისათვის შესაბამისად გვექნება:

$$\text{ლოდინის საშუალო არის } \rightarrow (16+17+13+19+16)/5 = 16.2$$

$$\text{შესრულების დროის საშუალო } \rightarrow (21+23+16+25+20)/5 = 21$$

ანუ თითოეულ პროცესს პროცესორის მისაღებად საშუალოდ მოუწია 16.2 დროითი ერთეულით ლოდინი, ხოლო მის შესრულებას დასჭირდა 21. დროითი ერთეული.

თუ განხილულ ამოცანას დავამატებთ გამოჩენის დროს ის გარკვეულწილად გართულდება. ვთქვათ, გვაქვს პროცესების შემდეგი ცხრილი

პროცესი	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	5	7	3	6	4
გამოჩენის დრო	1	4	0	7	2

რადგანაც ამ შემთხვევაში ჩვენი ცხრილი დამატებით შეიცავს პროცესების გამოჩენის დროს, ამიტომ პროცესების გამოჩენის და შესრულებიდან პროცესის გამოსვლა უნდა გაკონტროლდეს საგულდაგულოდ.

შევნიშნოთ, რომ თუ სისტემაში პროცესის წარმოქმნასთან ერთად შესრულებად პროცესს ამოეწურა დროითი კვანტი, მაშინ ვინაიდან შესრულებადი პროცესი საჭიროებს კონტექსტის შენახვას ჩავთვლით, რომ მზადყოფნის ცხრილში პირველი ჩაიწერება წარმოქმნილი პროცესი, ხოლო შემდეგ პროცესი, რომელმაც დროითი კვანტის ამოწურვის გამო იძულებით დატოვა პროცესორი. პროცესების მიმდევრობა, რომლითაც მოხდება დამგეგმვის მიერ მათი შესრულება, შეირჩევა შემდეგნაირად: რადგანაც სისტემაში პირველი გამოჩნდა P₃ პროცესი, ამიტომ ის დაიწყებს შესრულებას. 1 ერთეულის სისტემაში გამოჩნდება P₁ პროცესი. ის მოექცევა მზადყოფნის ცხრილის ბოლოში (მოცემული მომენტისათვის მიმდევრობას ექნება სახე - P₃ P₁). 2 ერთეულის გასვლის შემდეგ მიმდინარე პროცესს იძულებით ჩამოერთმევა პროცესორი და სისტემაში გამოჩნდება ახალი პროცესი P₅. რადგანაც P₃ პროცესის საჭიროებს კონტექსტის შენახვას, ამიტომ ჯერ მიმდევრობის ბოლოში მოექცევა P₅ პროცესი, ხოლო შემდეგ რადგანაც P₃ პროცესი უნდა შესრულდეს კიდევ 1 დროითი ერთეულით, ის გადმოინაცვლებს მიმდევრობის ბოლოში. მოცემული მომენტისათვის მიმდევრობას ექნება სახე - P₁ P₅ P₃.

პროცესების შესრულების დეტალიზირებულ ცხრილს ექნება სახე

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
P ₁	-	+	+	-	-	-	-	-	+	+	-	-	-	-	-	-	+								
P ₂				-	-	-	-	+	+	-	-	-	-	-	-	+	+	-	-	+	+	+	-	+	
P ₃	+	+	-	-	-	-	+																		
P ₄								-	-	-	-	-	-	+	+	-	-	-	+	+	-	-	+	+	
P ₅		-	-	+	+	-	-	-	-	-	-	+	+												

ამრიგად, ლოდინის და შესრულების დროისათვის შესაბამისად გვექნება:

$$\text{ლოდინის საშუალო არის } \rightarrow (12+14+4+11+7)/5 = 9.8$$

$$\text{შესრულების დროის საშუალო } \rightarrow (17+21+7+17+11)/5 = 14.6$$

ანუ თითოეულ პროცესს პროცესორის მისაღებად საშუალოდ მოუწია 9.8 დროითი

ერთეულით ლოდინი, ხოლო მის შესრულებას დასჭირდა 14.6 დროითი ერთეული.

დამოუკიდებელი სამუშაო

1. დაგეგმვის ციკლური ალგორითმის გამოყენებით დაითვალეთ პროცესების შესრულების დროის საშუალო, თუ ყველა პროცესი სისტემაში არსებობს ერთდროულად:

პროცესი (კვანტით 3)	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	15	10	5	2	15

პროცესი (კვანტით 1)	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	16	7	8	2	1

პროცესი (კვანტით 2)	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	10	1	5	10	11

2. დაგეგმვის ციკლური ალგორითმის გამოყენებით დაითვალეთ პროცესების შესრულების დროის საშუალო, თუ პროცესები სისტემაში გამოჩნდნენ სხვადასხვა დროს:

პროცესი (კვანტით 2)	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	5	7	3	2	8
გამოჩენის დრო	0	2	2	5	1

პროცესი (კვანტით 2)	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	8	10	7	2	15
გამოჩენის დრო	2	3	0	5	0

პროცესი (კვანტით 2)	P ₁	P ₂	P ₃	P ₄	P ₅
შესრულების დრო	16	7	8	2	1
გამოჩენის დრო	2	0	3	7	5

სემინარი 4. UNIX პროცესების ურთიერთქმედების ორგანიზაცია

პროცესები, რომლებიც იქმნებიან სხვადასხვა გამოყენებითი პროგრამების მიერ, აუცილებლად იქმნებიან გარკვეული მიზნით. შექმნილი პროცესი ასრულებს წარმომქმნელის მიერ მისთვის დაკისრებულ გარკვეულ ამოცანას. ამოცანის დასრულების შემდეგ შექმნილმა პროცესმა მის მიერ შესრულებულ სამუშაოზე ინფორმაცია უნდა მიაწოდოს მის წარმომქმნელს. გარდა ამისა წამროქმნილი პროცესი მისთვის დაკისრებული ამოცანის შესრულების მომენტში შეიძლება საჭიროებდეს შემქმნელისაგან გარკვეული მონაცემების მიღებას ან წარმომქმნელისათვის გარკვეული მონაცემების გადაცემას. ასეთი კომუნიკაციის საჭიროების აუცილებლობისას ოპერაციულ სისტემას აუცილებლად უნდა გააჩნდეს შესაბამისი კომუნიკაციის განხორციელების შესაძლებლობა. კომუნიკაციის საშუალებებს შორის ყველაზე მეტად გამოიყენება კავშირის არხი, რომელიც უზრუნველყოფს პროცესების საკმაოდ უსაფრთხო და ინფორმაციულ ურთიერთქმედებას.

კავშირის არხით მონაცემების გადაცემის არსებობს ორი მეთოდი: შეტანა/გამოტანის ნაკადი და შეტყობინება. გამოყენების თვალსაზრისით მათ შორის ყველაზე მარტივია ნაკადების მოდელი, რომელშიც ინფორმაციის მიღება/გადაცემის ოპერაციას საერთოდ არ აინტერესებს რა სახის ინფორმაციის მატარებელია ის. კავშირის არხში მთლიანი ინფორმაცია განიხილება, როგორც ბიტების ნაკრები, რომელსაც არ გააჩნია შიდა სტრუქტურა.

4.1. ფაილთან მუშაობა

ინფორმაციის გადაცემა ნაკადების მეშვეობით შესაძლებელია განხორციელდეს არამხოლოდ პროცესებს შორის, არამედ პროცესსა და შეტანა/გამოტანის მოწყობილობას შორის (მაგალითად, პროცესსა და დისკს შორის), რომელზეც მონაცემები წარმოიდგინება ფაილების სახით. ვინაიდან ფაილებთან ნაკადებით მუშაობისას გამოყენებადი სისტემური გამოძახებები წააგავს პროცესების ნაკადებით ურთიერთქმედებისას გამოყენებად სისტემურ გამოძახებებს, ამიტომ თავდაპირველად განვიხილოთ ფაილებსა და პროცესებს შორის ნაკადებით ინფორმაციის გაცვლის მექანიზმი.

პროგრამირების ყველა ენას გააჩნია ფაილებთან მუშაობის სტანდარტული ბიბლიოთეკური ფუნქციები, რომელთა მეშვეობითაც პროგრამისტს შეუძლია განახორციელოს გარკვეული მოქმედებები ფაილებზე. მაგალითად, C ენისათვის ფაილთან სამუშაო ფუნქციებია fopen(), fread(), fwrite(), fprintf(), fscanf(), fgets() და ა.შ. თითოეული მათგანი გამოიყენება ფაილზე გარკვეული მოქმედების განსახორციელებლად. მაგალითად, fgets() ფუნქცია გამოიყენება ფაილიდან სიმბოლოების წასაკითხად, fscanf() ფუნქცია კი ანხორციელებს ინფორმაციის შეტანას, რომელიც შეესაბამება მითითებულ ფორმატს, და ა.შ. ნაკადების მოდელის მიხედვით ოპერაციები, რომელიც განსაზღვრულია შეტანა/გამოტანის სტანდარტულ ბიბლიოთეკაში, არ წარმოადგენენ ნაკადურ ოპერაციებს, ვინაიდან თითოეული მათგანი მოითხოვს გადასაცემი მონაცემების სტრუქტურას.

UNIX ოპერაციული სისტემაში ეს ფუნქციები წარმოადგენენ სისტემურ გამოძახებზე დანამატს - სერვისულ ინტერფეისს, რომლებიც ანხორციელებენ ნაკადებით ინფორმაციის გაცვლის პირდაპირ ოპერაციებს პროცესებსა და ფაილებს შორის და არ საჭიროებენ იმის ცოდნას თუ რას შეიცავს ის.

4.2. ფაილური დესკრიპტორი

პროცესებზე საუბრისას აღვნიშნეთ, რომ ყოველ პროცესს სისტემაში გამოყოფილი აქვს მეხსიერების საკუთარი სივრცე, რომელშიც ინახება პროცესთან დაკავშირებული ყველა საჭირო ინფორმაცია (პროგრამული ფაილები, შესასრულებლად საჭირო ფაილები, ცვლადები და ა.შ.), და რესურსები. პროცესის მიერ გამოყენებულ ფაილებზე ინფორმაცია შედის მის სისტემურ კონტექსტში და ინახება მის მართვის ბლოკში - PCB. გადმოცემის სიმარტივისათვის შეგვიძლია დავუშვათ, რომ UNIX ოპერაციულ სისტემაში ინფორმაცია იმ ფაილებზე, რომელზედაც პროცესი ნაკადებით ანხორციელებს მონაცემების გაცვლის ოპერაციას, ნაკადების კავშირის არხზე ინფორმაციასთან ერთად (რომელიც აკავშირებს პროცესს სხვა პროცესებთან და შეტანა/გამოტანის სხვა მოწყობილობებთან) ინახება გარკვეულ მასივში, რომელსაც ფაილური დესკრიპტორების ცხრილი ეწოდება. ამ მასივის ელემენტის ინდექსს, რომელიც შეესაბამება შეტანა/გამოტანის გარკვეულ ნაკადს, ეწოდება ფაილური დესკრიპტორი ამ ნაკადისათვის. ამრიგად, ფაილური დესკრიპტორი წარმოადგენს პატარა არაუარყოფით რიცხვს, რომელიც მიმდინარე პროცესისათვის (დროის მოცემულ მომენტში) განსაზღვრავს შეტანა/გამოტანის გარკვეულ მოქმედ არხს. ზოგიერთი ფაილური დესკრიპტორი ნებისმიერი პროგრამის დაწყების ეტაპზე ასოცირდება შეტანა/გამოტანის სტანდარტულ ნაკადთან. მაგალითად, ფაილური დესკრიპტორი 0 შეესაბამება მონაცემთა შეტანის სტანდარტულ ნაკადს, 1 - მონაცემთა გამოტანის სტანდარტულ ნაკადს, 2 კი - შეცდომის გამოტანის სტანდარტულ ნაკადს. ინტერაქტიულ რეჟიმში ნორმალური მუშაობისას მონაცემთა შეტანის სტანდარტული ნაკადი პროცესს აკავშირებს კლავიატურასთან, ხოლო მონაცემების და შეცდომების გამოტანის სტანდარტული ნაკადი კი - ტერმინალის ეკრანთან.

4.3. ფაილებთან მუშაობა

`open()` სისტემური გამოძახება. ფაილური დესკრიპტორი გამოიყენება სისტემური გამოძახებისათვის შეტანა/გამოტანის ნაკადის აღმწერი პარამეტრის როლში, რომელიც ამ ნაკადზე ანხორციელებს ოპერაციებს. ამიტომ სანამ განვახორციელებდეთ ფაილიდან კითხვას და მასში ჩაწერას ფაილზე ინფორმაცია უნდა განვათავსოთ გახსნილი ფაილების ცხრილში და განვსაზღვროთ შესაბამისი დესკრიპტორი. ამისათვის გამოიყენება ფაილის გახსნის პროცედურა, რომელსაც ანხორციელებს სისტემური გამოძახება `open()`.

`open()` სისტემური გამოძახება flag-ების ნაკრებს იყენებს იმისათვის, რომ მოახდინოს იმ ოპერაციების განსაზღვრა, რომელთა გამოყენება ფაილის მიმართ იგულისხმება მომავალში, ან რომლებიც უნდა იყვნენ შესრულებულნი უშუალოდ ფაილის გახსნის მომენტში. flag-ების ყველა შესაძლო ნაკრებიდან განვიხილავთ მხოლოდ რამდენიმეს: `O_RDONLY`, `O_WRONLY`, `O_RDWR`, `O_CREAT` და `O_EXCL`. პირველი სამი flag-ი არის ურთიერთ-გამომრიცხავი: ერთერთი მათგანის გამოყენება აუცილებელია და მისი არსებობა გამორიცხავს დანარჩენი ორის არსებობის შესაძლებლობას. ეს flag-ები აღწერენ ოპერაციათა ნაკრებს, რომლებიც მომავალში ფაილის წარმატებული გახსნის შემთხვევაში ფაილებზე იქნება დაშვები: მხოლოდ კითხვა, მხოლოდ რედაქტირება, კითხვა და რედაქტირება. როგორც ვიცით ყოველ ფაილს ყავს სამი ტიპის მომხმარებელი, რომლებიც ამ ფაილზე სარგებლობს გარკვეული უფლებებით. თუ ფაილი მოცემული სახელით

არსებობს დისკზე და მასზე დაშვების უფლებები იმ მომხმარებლისათვის, რომლის სახელითაც მუშაობს მიმდინარე პროცესი, არ ეწინააღმდეგება მოთხოვნილ ოპერაციათა ნაკრებს, მაშინ ოპერაციული სისტემა ახდენს ფაილური ცხრილის სკანირებას დასაწყისიდან ბოლომდე პირველი ცარიელი ელემენტის მოსაძებნად, ავსებს მას და აბრუნებს ამ ელემენტის ინდექსს ფაილის დესკრიპტორის როლში. თუ ფაილი დისკზე არაა, არ ყოფნის უფლებები ან გახსნილი ფაილების ცხრილში არ არსებობს ცარიელი ელემენტი, მაშინ წარმოიქმნება შეცდომის შემცველი სიტუაცია.

იმ შემთხვევაში, როდესაც ვუშვებთ, რომ ფაილი დისკზე შეიძლება არ არსებობს და საჭიროა მისი შექმნა, flag-ი ოპერაციათა ნაკრებისათვის უნდა იყოს გამოყენებული O_CREAT flag-თან ერთად. თუ ფაილი არსებობს, მაშინ ყველაფერი ხორციელდება ზემოთ აღწერილი სცენარით. თუ ფაილი არ არსებობს, მაშინ თავიდან ხორციელდება ფაილის შექმნა, სისტემური გამოძახების პარამეტრებში მითითებული უფლებებით. ოპერაციათა ნაკრების შესაბამისობის შემოწმება, გამოცხადებულ დაშვების უფლებებით, შესაძლებელია არც განხორციელდეს (როგორც მაგალითად, Linux-ში).

იმ შემთხვევაში, როდესაც ჩვენ ვითხოვთ, რომ ფაილი დისკზე არ არსებობდეს და იყოს შექმნილი მისი გახსნის მომენტში, flag-ი ოპერაციათა ნაკრებისათვის O_CREAT და O_EXCL flag-ებთან უნდა გამოიყენებოდეს კომბინირებულად.

open სისტემური გამოძახების პროტოკოლი

```
#include <fcntl.h>
int open(char *path, int flags);
int open(char *path, int flags, int mode);
```

სისტემური გამოძახების აღწერა

open სისტემური გამოძახება გამოიყენება ფაილის გახსნის ოპერაციისათვის და მისი განხორციელებისას აბრუნებს გახსნილი ფაილის ფაილურ დესკრიპტორს (პატარა არაუარყოფით მთელ რიცხვს), რომელიც სხვდასხვა სისტემური გამოძახების მიერ გამოიყენება ამ ფაილზე დასაშვები ოპერაციების განსახორციელებლად.

path პარამეტრი წარმოადგენს მიმთითებელს იმ სტრიქონზე, რომელიც შეიცავს გასახსნელი ფაილის მიმართებით ან სრულ სახელს.

flags პარამეტრი განსაზღვრავს გასახსნელ ფაილზე დაშვების უფლებებს. მას შეუძლია მიიღოს შემდეგი სამი მნიშვნელობიდან მხოლოდ ერთი:

- O_RDONLY - მხოლოდ კითხვა;
- O_WRONLY - მხოლოდ რედაქტირება;
- O_RDWR - კითხვა და რედაქტირება;

ამ მნიშვნელობების კომბინირება „ბიტური ან (‘|’“ ოპერაციით შესაძლებელია ერთი ან რამდენიმე flag-თან:

- O_CREAT - თუ ფაილი მითითებული სახელით არ არსებობს და საჭიროა მისი შექმნა;
- O_EXCL - გამოიყენება O_CREAT flag-თან ერთად. მათი ერთობლივი გამოყენებისას და მითითებული სახელის ფაილის არსებობისას არ ხდება ფაილის გახსნა და წარმოიშობა შეცდომის შემცველი სიტუაცია.
- O_NDELAY - კრძალავს პროცესის გადაყვანას ლოდინის მდგომარეობაში ფაილის გახსნის ოპერაციის ან ამ ფაილზე შემდგომი ნებისმიერი ოპერაციისას.
- O_APPEND - ფაილის გახსნისას და ჩაწერის ყოველი ოპერაციის შესრულების

წინა (თუ ის დაშვებულია) ფაილის მიმდინარე პოზიციის მიმთითებელი ყენდება ფაილის ბოლოზე.

- O_TRUNC - თუ ფაილი არსებობს მის მოცულობას ამცირებს 0-მდე, ფაილის არსებული ატრიბუტების შენახვით გარდა შესაძლებელია ფაილზე ბოლო დაშვებისა და მისი ბოლო მოდიფიცირებისა.

გარდა ამისა, UNIX ოპერაციული სისტემის ზოგიერთ ვერსიებში შესაძლებელია გამოიყენებოდეს flag-ების დამატებითი მნიშვნელობები:

- O_SYNC - ფაილში ჩაწერის ნებისმიერი ოპერაცია იქნება ბლოკირებული იმ დრომდე სანამ ჩაწერილი ინფორმაცია ფიზიკურად არ იქნება განთავსებული შესაბამის hardware დონეზე.
- O_NOCTTY - თუ ფაილის სახელი განეკუთვნება ტერმინალურ მოწყობილობას, ის არ იქცევა პროცესის მმართველ ტერმინალად, იმ შემთხვევაშიც კი, თუ ამ დრომდე მას არ გააჩნდა მმართველი ტერმინალი.

mode პარამეტრი ახალი ფაილის შექმნისას მისთვის აყენებს სხვადასხვა კატეგორიის მომხმარებელთათვის დაშვების უფლებებს. ის აუცილებელია, თუ მოცემულ flag-ებს შორის არის flag-ი O_CREAT და წინააღმდეგ შემთხვევაში შეიძლება იქნას გამოტოვებული. ეს პარამეტრი გამოისახება, როგორც შემდეგი რვაობითი მნიშვნელობების ჯამი.

- 0400 - მომხმარებელი სარგებლობს კითხვის უფლებით;
- 0200 - მომხმარებელი სარგებლობს რედაქტირების უფლებით;
- 0100 - მომხმარებელი სარგებლობს შესრულების უფლებით;
- 0040 - მომხმარებლის ჯგუფი სარგებლობს კითხვის უფლებით;
- 0020 - მომხმარებლის ჯგუფი სარგებლობს რედაქტირების უფლებით;
- 0010 - მომხმარებლის ჯგუფი სარგებლობს შესრულების უფლებით;
- 0004 - სხვა მომხმარებელი სარგებლობს კითხვის უფლებით;
- 0002 - სხვა მომხმარებელი სარგებლობს რედაქტირების უფლებით;
- 0001 - სხვა მომხმარებელი სარგებლობს შესრულების უფლებით.

ფაილის შექმნისას რეალურად დასაყენებელი დაშვების უფლებები მიიღება mode პარამეტრის სტანდარტული კომბინაციისაგან და მიმდინარე პროცესის ფაილის შექმნის ნიღაბის umask მნიშვნელობისაგან, კერძოდ, ისინი ტოლია mode = umask. FIFO ტიპის ფაილების გახსნისას სისტემურ გამოძახებას გააჩნია ყოფაქცევის გარკვეული თავისებურება სხვა ტიპის ფაილების გახსნასთან შედარებით. თუ FIFO-ს გახსნა ხდება მხოლოდ წასაკითხად და არა დასმული flag-ი O_NDELAY, მაშინ ხდება სისტემური გამოძახების განმახორციელებელი პროცესის ბლოკირება სანამ რომელიმე სხვა პროცესი არ გახსნის FIFO-ს ჩასაწერად. თუ O_NDELAY flag-ი დასმულია, მაშინ ბრუნდება FIFO -სთან ასოცირებული ფაილური დესკრიპტორის მნიშვნელობა. თუ FIFO გაიხსნა მხოლოდ ჩასაწერად და არა დასმული flag-ი O_NDELAY, მაშინ ხდება სისტემური გამოძახების განმახორციელებელი პროცესის ბლოკირება სანამ რომელიმე სხვა პროცესი არ გახსნის FIFO-ს წასაკითხად. თუ O_NDELAY flag-ი დასმულია, მაშინ წარმოიშობა შეცდომის შემცველი სიტუაცია და ბრუნდება მნიშვნელობა -1.

დასაბრუნებელი მნიშვნელობა

სისტემური გამოძახება ნორმალურად დასრულების შემთხვევაში აბრუნებს გახსნილი ფაილისათვის ფაილური დესკრიპტორის მნიშვნელობას, ხოლო შეცდომის წარმოქმნის შემთხვევაში მნიშვნელობას -1.

სისტემური გამოძახება `read()` და `write()`. როგორც ზემოთ აღვნიშნეთ `open()` სისტემური გამოძახება ნორმალური დასრულების შემთხვევაში აბრუნებს ფაილურ დესკრიპტორს, რომლის გამოყენებაც არის შესაძლებელი ფაილზე სხვადასხვა (კითხვის, რედაქტირების და ა.შ.) ოპერაციების განსახორციელებლად. ფაილიდან მონაცემების წასაკითხად გამოიყენება სისტემური გამოძახება `read()`, ხოლო ფაილში მონაცემების ჩასაწერად კი - `write()`.

read და write სისტემურ გამოძახებათა პროტოტიპი

```
#include<sys/types.h>
#include<unistd.h>

size_t read(int fd, void *addr, size_t nbytes);
size_t write(int fd, void *addr, size_t nbytes);
```

სისტემურ გამოძახებათა აღწერა

სისტემური გამოძახება `write` და `read` გამოიყენება ინფორმაციის შეტანა/გამოტანის ოპერაციების განსახორციელებლად ფაილებში ან მეხსიერებაში

`fd` პარამეტრი წარმოადგენს ფაილურ დესკრიპტორს ადრე შექმნილი კავშირის არხისათვის, რომლის მეშვეობითაც განხორციელდება ინფორმაციის გადაცემა ან მიღება.

`addr` პარამეტრი წარმოადგენს მეხსიერების სივრცის მისამართს, რომლიდანაც დაწყებული იქნება ინფორმაცია წაკითხული ან ინფორმაცია განთავსებული.

`nbytes` პარამეტრი განსაზღვრავს `write` სისტემური გამოძახებისათვის ბაიტების იმ რაოდენობას, რომელთა განთავსებაც სისტემურმა გამოძახებამ განახორციელა მეხსიერების სივრცეში `addr` მისამართიდან დაწყებული, ხოლო `read` სისტემური გამოძახებისათვის კი განსაზღვრავს ბაიტების იმ რაოდენობას, რომელთა კითხვა განახორციელა სისტემურმა გამოძახებამ მეხსიერების სივრციდან `addr` მისამართიდან დაწყებული.

დასაბრუნებელი მნიშვნელობა

წარმატებული დასრულების შემთხვევაში სისტემური გამოძახება აბრუნებს რეალურად გაგზავნილი ან მიღებული ბაიტების რაოდენობას. შევნიშნოთ, რომ ეს მნიშვნელობა შეიძლება არ ემთხვევოდეს `nbytes` პარამეტრის მოცემულ მნიშვნელობას და იყოს მასზე ნაკლები, მონაცემთა გადაცემისას დისკზე ან კავშირის არხში საკმარისი ადგილის არარსებობის ან მისი მიღებისას ინფორმაციის არარსებობის გამო. წარუმატებელი დასრულების ან გადაცემისას შეცდომის წარმოქმნის შემთხვევაში სისტემური გამოძახება აბრუნებს უარყოფით მნიშვნელობას.

ფაილებთან მუშაობისას ყოფაქცევის თავისებურებანი. ფაილებთან მუშაობისას ხორციელდება ფაილიდან ინფორმაციის კითხვა ან იქ მისი ჩაწერა, ფაილის მიმთითებლის მიმდინარე მდგომარეობიდან დაწყებული. მიმთითებლის მნიშვნელობა იზრდება რეალურად ჩაწერილი ან წაკითხული ბაიტების რაოდენობით. ფაილიდან ინფორმაციის წაკითხვისას ის არ იკარგება ფაილიდან. თუ სისტემური გამოძახება `read` აბრუნებს მნიშვნელობას 0, ეს ნიშნავს, რომ ფაილიდან ინფორმაცია წაკითხულია ბოლომდე.

ნაკადების ოპერაციის დასრულების შემდეგ პროცესმა უნდა შეასრულოს შეტანა/გამოტანის ნაკადის დახურვის ოპერაცია, რომლის დროსაც მოხდება კავშირის არხის ბუფერის სრულად გასუფთავება, განთავისუფლდება ოპერაციული სისტემის ის რესურსები, რომლებსაც იკავებდა პროცესი, ფაილური დესკრიპტორების ცხრილის შესაბამისი ელემენტი. ამ მოქმედებებზე პასუხისმგებელია სისტემური გამოძახება `close()`. უნდა აღინიშ-

ნოს, რომ პროცესის მუშაობის დასრულებისას exit() ფუნქციის ცხადი თუ არაცხადი გამო-
ძახებისას შეტანა/გამოტანის გახსნილი ნაკადები დახურვა ხორციელდება ავტომატურად.

close სისტემური გამოძახების პროტოტიპი

```
#include<unistd.h>
```

```
int close(int fd);
```

სისტემური გამოძახების აღწერა

close სისტემური გამოძახება გამოიყენება ფაილებთან მუშაობის ოპერაციების კორექტული
დასრულებისათვის, რომლებიც აღიწერებიან ოპერაციულ სისტემაში ფაილური
დესკრიპტორების მეშვეობით.

დასაბრუნებელი მნიშვნელობა

სისტემური გამოძახება ნორმალური დასრულების შემთხვევაში აბრუნებს მნიშვნელობას -
0 და შეცდომის წარმოქმნის შემთხვევაში მნიშვნელობას -1.

განვიხილოთ მაგალითი. შევქმნათ პროცესი, რომელშიც მასივში არსებული მონაცე-
მების ჩაწერა მოხდება პროგრამით შექმნილ ფაილში.

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

int main(){
    int fd;           // ცვლადი ფაილური დესკრიპტორის მნიშვნელობისათვის
    size_t s1, s2;  // ცვლადები მონაცემების ზუსტად გადაცემის შესამოწმებლად
    char mass[] = "Operating System";
    s2 = sizeof(mass);
    // გავაწულოთ შესაქმნელი ფაილისათვის ოპერაციული სისტემის მიერ გაჩუმებით
    // დანიშვნადი დაშვების უფლებები
    (void) umask(0);
    // სამუშაო დირექტორიაში შევმწარ test.txt ფაილი (უფლებებით 0765)
    fd = open("test.txt", O_WRONLY | O_CREAT, 0765);
    // შევამოწმოთ fd ფაილური დესკრიპტორის მნიშვნელობა
    if(fd < 0){ // თუ ფაილი არ შეიქმნა, მასში მონაცემებს ვერ ჩავწერთ
        printf("Can't open file\n");
        exit(EXIT_FAILURE);
    }
    // ინფორმაციის ჩაწერა კავშირის არხში
    s1 = write(fd, mass, s1);
    // თუ მონაცემების ჩაწერისას სისტემური გამოძახება წარუმატებლად დასრულდა ან
    // მონაცემები სრულად არ იქნა ჩაწერილი, მაშინ ვასრულებთ პროგრამას
    if(s1 != s2){
        printf("Can't write all massing\n");
        exit(EXIT_FAILURE);
    }
    close(fd); //დავხუროთ კავშირის არხი
    return 0; }
```

პროგრამა 04-1.c. open(), write() და close() სისტემური გამოძახების გამოყენება

აკრიფეთ, დააკომპილირეთ და შეასრულეთ პროგრამა.

4.4. კავშირის არხი

მიუხედავად იმისა, პროცესები შეიძლება სრულდებოდნენ ერთ ან რამდენიმე გამოთვლით მანქანაზე, მათ ხშირად უწევთ ერთიმეორესთან ურთიერთქმედება, საერთო რესურსების გამოყენება, ერთიმეორის მიერ მიღებული შედეგების გათვალისწინება საკუთარი სამუშაოს შესრულებისას და ა.შ.

პროცესებს ერთიმეორესთან ურთიერთქმედებენ მონაცემების გაცვლის გზით. მონაცემების გაცვლის არსებობს სამი მექანიზმი:

- სიგნალები;
- კავშირის არხები
- განაწილებადი მეხსიერება.

მიმდინარე სემინარულ მეცადინეობაზე განვიხილავთ პროცესებს შორის მონაცემების გაცვლის კავშირის საშუალებებს.

კავშირის არხებით პროცესებს შორის მონაცემების გაცვლა ხორციელდება ოპერაციული სისტემის მიერ ამ მიზნისათვის სპეციალურად გამოყოფილი კავშირის არხებით. გადასაცემი მონაცემების მოცულობა დამოკიდებულია კავშირის არხის გამტარუნარიანობაზე. გადასაცემი მონაცემების მოცულობის გაზრდით იზრდება სხვა პროცესების ყოფაქცევაზე ზემოქმედების შესაძლებლობა;

4.4.1. კავშირის არხი pipe

UNIX ოპერაციულ სისტემაში მონაცემთა ნაკადების მოდელის გამოყენებით სხვადა-სხვა პროცესებს შორის ან ერთი პროცესის შიგნით ინფორმაციის გადაცემის ყველაზე მარტივ მეთოდს წარმოადგენს pipe (არხი, მილი).

pipe შეგვიძია წარმოვიდგინოთ როგორც ოპერაციული სისტემის მეხსიერების მისამართების სივრცეში განთავსებული სასრული მოცულობის „უხილავი მილი“, რომლის შემავალ და გამომავალ ბოლოებზე მიმართვა ხორციელდება სისტემური გამოძახებების მეშვეობით. pipe ხშირად ორგანიზებულია წრიული ბუფერის სახით. ბუფერში კითხვის და ჩაწერის ოპერაციების შესრულებისას გადაადგილდება შემავალი და გამომავალი ნაკადების შესაბამისი ორი მიმთითებელი. ამასთან, გამომავალ ნაკადს არ შეუძლია გაასწროს შემავალ ნაკადს და პირიქით. ოპერაციული სისტემის შიგნით ასეთი წრიული ბუფერის ახალი ეგზემპლარის შესაქმნელად გამოიყენება სისტემური გამოძახება pipe().

pipe სისტემური გამოძახების პროტოტიპი

```
#include<unistd.h>
```

```
int pipe(int *fd);
```

სისტემური გამოძახების დოკუმენტაცია

სისტემური გამოძახება pipe ემსახურება ოპერაციული სისტემის შიგნით pipe-ის შექმნას.

fd პარამეტრი წარმოადგენს მიმთითებელს ორი მთელი პარამეტრისაგან შემდგარ მასივზე. სისტემური გამოძახების წარმატებით დასრულების შემდეგ მასივის პირველ ელემენტში - fd[0] - შეტანილი იქნება ფაილური დესკრიპტორი, რომელიც შეესაბამება pipe-ის გამოსატანი მონაცემების ნაკადს და იძლევა მხოლოდ კითხვის ოპერაციის განხორციელების შესაძლებლობას, ხოლო მასივის მეორე ელემენტში -

fd[1] - შეტანილი იქნება ფაილური დესკრიპტორი, რომელიც შეესაბამება მონაცემების შესატან ნაკადს და იძლევა მხოლოდ ჩაწერის ოპერაციის განხორციელების შესაძლებლობას.

დასაბრუნებელი მნიშვნელობა

სისტემური გამოძახება აბრუნებს 0-ის ტოლ მნიშვნელობას წარმატებული დასრულების შემთხვევაში და შეცდომის წარმოქმნის შემთხვევაში კი -1-ს.

მუშაობის პროცესში სისტემური გამოძახება ახდენს ბუფერში მეხსიერების მიღამოს და მიმთითებლის გამოყოფის ორგანიზებას და შემავალ და გამომავალ ნაკადებზე შესაბამისი ინფორმაცია შეაქვს ფაილური დესკრიპტორების ცხრილის ორ ელემენტში, რითაც ყოველ pipe-თან აკავშირებს ორივე ფაილურ დესკრიპტორს. ერთერთი მათგანისათვის დაშვებულია მხოლოდ pipe-დან კითხვის ოპერაცია, ხოლო მეორისათვის მხოლოდ pipe-ში ჩაწერის ოპერაცია. ამ ოპერაციების განსახორციელებლად შესაძლებელია გამოყენებულ იქნას იგივე read() და write() სისტემური გამოძახებები, რაც ფაილებთან მუშაობის შემთხვევაში. ბუნებრივია, რომ მონაცემთა ნაკადის შეტანის ან/და გამოტანის ოპერაციების დასრულების შემდეგ საჭიროა შესაბამისი ნაკადის დახურვა და სისტემური რესურსების გამოთავისუფლება close() სისტემური გამოძახების გამოყენებით. უნდა აღინიშნოს, რომ თუ pipe-ის გამოყენებელი ყველა პროცესი ხურავს მათთან ასოცირებულ ფაილურ დესკრიპტორს, მაშინ ოპერაციული სისტემა ახდენს pipe-ის ლიკვიდირებას. ამრიგად, pipe-ის სიცოცხლის ხანგრძლივობა სისტემაში არ შეიძლება აღემატებოდეს მასთან მომუშავე პროცესების სიცოცხლის ხანგრძლივობას.

მნიშვნელოვანი განსხვავება pipe-სა და ფაილს შორის მდგომარეობს იმაში, რომ ინფორმაცია pipe-დან იშლება წაკითხვისთანავე და ხელმეორედ მისი წაკითხვა შეუძლებელია.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
int main(){
    int fd[2]; // ორელემენტიანი მასივი ფაილური დესკრიპტორისათვის
    size_t s1, s2; // ცვლადები მონაცემთა ზუსტად გადაცემის შესამოწმებლად
    char mass[] = "Operating System!";
    s2 = sizeof(mass);
    char re_mass[s2];
    // შევქმნათ კავშირის არხი pipe-ი
    if(pipe(fd) < 0){
        printf("Can't create pipe\n");
        exit(EXIT_FAILURE);
    }
    // pipe არხში შევიტანოთ mass მასივის მონაცემები
    s1 = write(fd[1], mass, s2);
    // შევამოწმოთ რამდენად სწორად მოხდა მონაცემების ჩაწერა
    if(s1 != s2){
        printf("Can't write all massing\n");
    }
}
```

```

        exit(EXIT_FAILURE);
    }
    // pipe არხიდან re_mass მასივში წავიკითხოთ მონაცემები
    s1 = read(fd[0], re_mass, s2);
    // შევამოწმოთ რამდენად სწორად მოხდა მონაცემების ჩაწერა
    if(s1 != s2) {
        printf("Can't read massing\n");
        exit(EXIT_FAILURE);
    }
    printf("%s\n", re_mass); // დავგეჭდოთ re_mass მასივი
    close(fd[0]); // დავხუროთ გახსნილი კავშირის არხები
    close(fd[1]);
    return 0;
}

```

პროგრამა 04-2.c. pipe-ის წარმოქმნა სისტემაში

აკრიფეთ პროგრამა, დააკომპილირეთ და შეასრულეთ ის.

4.4.2. მემკვიდრე პროცესებს შორის ერთმიმართულებიანი კავშირის ორგანიზება

ცხადია, რომ pipe-ს უპირატესობა, თუ შემოფარგლული იქნებოდა ერთი პროცესის შიგნით კოპირების ფუნქციით, რომლის განხორციელებისას მონაცემებს მოუწევდათ ოპერაციულ სისტემის „შემოვლა“, მაშინ ის არ იქნებოდა აღნიშვნის ღირსი. fork() სისტემური გამოძახებისას მემკვიდრე პროცესებს გადაეცემათ ფაილური დესკრიპტორის მნიშვნელობა, რომელიც exec() სისტემური გამოძახების განხორციელებისას შედის პროცესის სისტემური კონტექსტის განუყოფელ ნაწილში. ეს გარემოება საშუალებას გვაძლევს pipe-ის მეშვეობით განვახორციელოთ ინფორმაციის გადაცემა იმ მემკვიდრე პროცესებს შორის, რომელთაც ყავთ pipe-ს წარმომქმნელი წინაპარი.

განვიხილოთ მაგალითი, რომელშიც ”მემკვიდრე“ პროცესებს შორის pipe არხის მეშვეობით მონაცემების გაცვლის მიზნით დემონსტრირებულია ერთმიმართულებიანი კავშირის ორგანიზება.

```

#include<sys/types.h>
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h> int main(){
    int fd[2], pr, st;
    size_t s1, s2;
    char mass[] = "Operating System!";
    s2 = sizeof(mass); char re_mass[s2];
    // შევქმნათ კავშირის არხი pipe-ი
    if(pipe(fd) < 0){
        printf("Can't create pipe\n");
        exit(EXIT_FAILURE);
    }
    // შევქმნათ ახალი პროცესი
    if((pr = fork()) < 0){
        printf("Can't fork child\n");

```

```

        exit(EXIT_FAILURE);
    }
    else {
        if (pr != 0) { //მშობელ-პროცესი
            close(fd[0]); // დავხუროთ მონაცემების კითხვის არხი
            s1 = write(fd[1], mass, s2); // pipe არხში ჩავწეროთ მონაცემები
            if(s1 != s2){
                printf("Can't write all massing\n");
                exit(EXIT_FAILURE);
            }
            close(fd[1]); // დავხუროთ მონაცემების ჩაწერის არხი
            printf("Parent exit\n");
        }
        else { // შვილი-პროცესი
            close(fd[1]); // დავხუროთ მონაცემების ჩაწერის არხი
            s1 = read(fd[0], re_mass, s2); // კავშირის არხიდან წავიკითხოთ მონაცემები
            if(s1 != s2){
                printf("Can't read massing\n");
                exit(EXIT_FAILURE);
            }
            printf("%s\n",re_mass); // დაგბეჭდოთ წაკითხული სტრიქონი
            close(fd[0]); // დავხუროთ მონაცემების კითხვის არხი
        }
    }

    return 0;
}

```

პროგრამა 04-3.c. ერთმიმართულებიანი კავშირი მშობელ- და შვილ-პროცესს შორის აკრიფეთ პროგრამა, დააკომპილირეთ და შეასრულეთ ის.

4.4.3. მემკვიდრე პროცესებს შორის ორმიმართულებიანი კავშირის ორგანიზება

pipe-ი გამოიყენება მხოლოდ ერთმიმართულებიანი ანუ სიმპლექსური კავშირისათვის. წინა მაგალითში pipe-ის მეშვეობით ორმიმართულებიანი კავშირის ორგანიზების შემთხვევაში შესაძლებელია წარმოქმნილიყო შემდეგი სიტუაცია: მშობელი პროცესი კავშირის არხიდან წაიკითხავდა მის მიერვე ჩაწერილ მონაცემებს (თითქოს შვილმა განახორციელა მათი ჩაწერა), ხოლო შვილი პროცესი კი საერთოდ ვერ მიიღებდა ვერაფერს. ერთი pipe-ით ორმიმართულებიანი კავშირის ორგანიზებისათვის საჭიროა პროცესების სინქრონიზაციის სპეციალური საშუალებები. უფრო მარტივი მეთოდი მემკვიდრე პროცესებს შორის ორმიმართულებიანი კავშირის ორგანიზაციისა მდგომარეობს ორივე მიმართულებით თითო pipe-ის გამოყენებაში.

შევნიშნოთ, რომ ზოგიერთ UNIX-მსგავს სისტემაში (მაგალითად, Solaris-ში) რეალიზებულია სრული დუპლექსური pipe. ასეთ სისტემებში ორივე ფაილური დესკრიპტორისათვის, რომელიც ასოცირდება pipe-თან, ნებადართულია კითხვის და ჩაწერის ოპერაცია.

4.4.4. კავშირის არხი FIFO

როგორც უკვე აღვნიშნეთ, ოპერაციულ სისტემაში pipe-ის განთავსებაზე ინფორმა-

ციას ფლობს მხოლოდ მისი წარმომქმნელი პროცესი და ეს ინფორმაცია ინახება ფაილური დესკრიპტორების ცხრილში. ვინაიდან სისტემაში ახალი პროცესს მემკვიდრეობით გადაეცემა ფაილური დესკრიპტორების მნიშვნელობები, ამიტომ pipe-ის გამოყენება შეუძლიათ მხოლოდ pipe-ის წარმომქმნელ და მის მემკვიდრე პროცესებს. UNIX ოპერაციულ სისტემაში არსებობს pipe-ის გამოყენებით არამონათესავე პროცესებს შორის მონაცემების გადაცემის შესაძლებლობა, მაგრამ მისი რეალიზება საკმაოდ რთულია და ჩვენ მას არ შევხებით.

UNIX-მსგავს ოპერაციულ სისტემაში ნაკადებით ნებისმიერი პროცესების ურთიერთქმედებისათვის გამოიყენება კავშირის არხი, რომელსაც FIFO ეწოდება. FIFO ყველაფრით წააგავს pipe-ს გარდა ერთი გამონაკლისისა: ბირთვის მისამართების სივრცეში FIFO -ს განთავსებაზე და მის მდგომარეობაზე მონაცემების მიღება პროცესებს შეუძლიათ არა მემკვიდრე კავშირებით არამედ ფაილური სისტემიდან. ამ მიზნით FIFO -ს შექმნისას დისკზე იქმნება სპეციალური ტიპის ფაილი (.fifo), რომლის გამოყენებითაც პროცესები ცვლიან მონაცემებს ერთმანეთთან. პროცესების ურთიერთქმედების დასრულების შემდეგ FIFO არხი წყვეტს ფუნქციონირებას, ისევე როგორც pipe -ის შემთხვევაში, სპეციალური ტიპის ფაილი კი სისტემაში რჩება. აღნიშნული ფაილის გამოყენება მომავალში პროცესებს შორის ურთიერთქმედებისათვის შესაძლებელია და პროცესების ურთიერთქმედების საჭიროებისას არ არსებობს აუცილებლობა შექმნას ახალი fifo გაფართოების ფაილი. ასეთ შემთხვევაში პროცესების ურთიერთქმედებამდე საჭიროა მათ იცოდნენ (სპეციალურად ამ მიზნისათვის შექმნილ) fifo გაფართოების ფაილამდე სრული ან მიმართებითი სახელი. FIFO -ს შესაქმნელად გამოიყენება სისტემური გამოძახება mknod() (UNIX მსგავსი სისტემების ზოგიერთ ვერსიებში დამატებით არსებობს ფუნქცია mkfifo()).

mknod სისტემური გამოძახების პროტოტიპი

```
#include<sys/stat.h>
#include<unistd.h>
int mknod(char *path, int mode, int dev);
```

სისტემური გამოძახების აღწერა

FIFO არხის შესაქმნელად გამოიყენება mknod სისტემური გამოძახება. ვინაიდან ამ სისტემურ გამოძახებას გამოვიყენებთ მხოლოდ კავშირის არხის შექმნის საილუსტრაციოდ ქვემოთ მოყვანილი აღწერა არაა სრული. ჩვენ ასევე არ განვიხილავთ პარამეტრების გადაცემის ყველა შესაძლო ვარიანტს.

path პარამეტრი წარმოადგენს მიმთითებელს იმ სტრიქონზე, რომელიც შეიცავს ფაილის სრულ ან მიმართებით სახელს (სპეციალური ტიპის ფაილი) და FIFO იყენებს პროცესებს შორის მონაცემების გაცვლის მიზნით. FIFO-ს შექმნისას მსგავსი სახელისა და გაფართოების ფაილი არ უნდა არსებობდეს იმ დირექტორიაში, სადაც ეს ფაილი იქმნება. წინააღმდეგ შემთხვევაში არ მოხდება FIFO-ს შექმნა.

Mode პარამეტრი აყენებს FIFO-ზე სხვადასხვა კატეგორიის მომხმარებელთა დაშვების უფლებების ატრიბუტებს. ეს პარამეტრი მოიცემა S_IFIFO ოფციისა და დაშვების უფლებების კომბინირებით ბიტური „ან“ ოპერაციით. FIFO-სთვის დაშვების უფლებები განისაზღვრება შემდეგი რვაობითი მნიშვნელობების კომბინაციით:

- 0400 – FIFO-ს შემქმნელი მომხმარებლისათვის დაშვებულია კითხვის უფლება;
- 0200 – FIFO-ს შემქმნელი მომხმარებლისათვის დაშვებულია ჩაწერის უფლება;

- 0040 – FIFO-ს შემქმნელი მომხმარებლის ჯგუფისათვის დაშვებულია კითხვის უფლება;
- 0020 – FIFO-ს შემქმნელი მომხმარებლის ჯგუფისათვის დაშვებულია ჩაწერის უფლება;
- 0004 – სხვა მომხმარებლებისათვის დაშვებულია კითხვის უფლება;
- 0002 – სხვა მომხმარებლებისათვის დაშვებულია ჩაწერის უფლება.

Dev პარამეტრი არაა არსებითი ჩვენს შემთხვევაში და ამიტომ ჩვენ მას ყოველთვის მოცემთ მნიშვნელობით 0.

დასაბრუნებელი მნიშვნელობა

FIFO-ს წარმატებით შექმნისას სისტემური გამოძახება აბრუნებს მნიშვნელობას 0, ხოლო წარუმატებელ შემთხვევაში კი უარყოფით მნიშვნელობას.

mkfifo ფუნქციის პროტოტიპი

```
#include<sys/stat.h>
#include<unistd.h>
int mkfifo(char *path, int mode);
```

ფუნქციის აღწერა

ზოგიერთ UNIX-მსგავს სისსტემაში FIFO-ს შექმნა დამატებით შესაძლებელია mkfifo ფუნქციის გამოყენებით.

path - პარამეტრი წარმოადგენს გარკვეულ დირექტორიაში შესაქმნელი სპეციალური ტიპის ფაილის სრულ ან მიმართებით სახელზე მიმთითებელს. ამ შემთხვევაშიც აუცილებელია, რომ fifo გაფართოების ფაილის შექმნისას მსგავსი სახელის ფაილი არ არსებობდეს შესაბამის დირექტორიაში.

mode პარამეტრი აყენებს FIFO-ზე დაშვების უფლებებს სხვადასხვა კატეგორიის მომხმარებლებისათვის.

დასაბრუნებელი მნიშვნელობა

FIFO-ს წარმატებით შექმნისას ფუნქცია აბრუნებს მნიშვნელობას 0, ხოლო წარუმატებელ შემთხვევაში კი უარყოფით მნიშვნელობას.

მნიშვნელოვანია გვახსოვდეს FIFO-ს ტიპის ფაილი არ იქმნება დისკზე ინფორმაციის განსათავსებლად, რომელიც იწერება სახელდებულ pipe-ში. ეს ინფორმაცია თავსდება ოპერაციული სისტემის მისამართების სივრცის შიგნით, ხოლო ფაილი წარმოადგენს მხოლოდ სანიშნეს, რომელიც ქმნის მისი განთავსების გადამისამართებებს.

4.4.5. open() გამოძახების თავისებურება FIFO-ს გახსნისას

FIFO-სთან მუშაობისას read() და write() სისტემურ გამოძახებებს გააჩნიათ იგივე თავისებურებები, რაც pipe-ის შემთხვევაში. open() სისტემურ გამოძახებისას გვაქვს გარკვეული ცვლილებები, რაც გამოწვეულია მისი შესრულებადი პროცესების ბლოკირების შესაძლებლობით. თუ FIFO იხსნება მხოლოდ წასაკითხად და O_NDELAY flag-ი არაა მითითებული, მაშინ ხდება სისტემური გამოძახების შემსრულებელი პროცესის ბლოკირება მანამ, სანამ სხვა რომელიმე პროცესი არ გახსნის FIFO-ს ჩასაწერად. თუ O_NDELAY flag-ი მითითებულია ბრუნდება FIFO-სთან ასოცირებული ფაილური დესკრიპტორის მნიშვნელობა. თუ FIFO იხსნება მხოლოდ ჩასაწერად და O_NDELAY flag-ი არაა მითითებული, მაშინ ხდება სისტემური გამოძახების შემსრულებელი პროცესის ბლოკირება მანამ, სანამ სხვა რომელიმე პროცესი არ გახსნის FIFO-ს წასაკითხად. თუ O_NDELAY flag-ი მითითებულია წარმოიშობა შეცდომა და ბრუნდება მნიშვნელობა -1. open() სისტემური გამოძახების პარამეტრებში O_NDELAY flag-ის მითითებით FIFO-ს გამხსნელ პროცესს

ნაკადიდან მონაცემების კითხვის და მასში ჩაწერის მომდევნო ოპერაციების შესრულებისას ეკრალება ბლოკირება.

```
#include<fcntl.h>
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/stat.h>
#include<sys/wait.h>
#include<sys/types.h>
int main(){
    int fd, pr, st, FIFO;
    size_t s1, s2;
    char mass[] = "Operating System!"; // გადასაცემი მასივი
    s2 = sizeof(mass);
    char re_mass[s2];
    char name[] = "file fifo"; // კავშირის ფაილის სახელი
    (void) umask(0); // გავანულოთ დაშვების საწყისი უფლებები
    // მექანიზმის არხი FIFO
    FIFO = mknod(name, S_IFIFO | 0754, 0)
    if(FIFO < 0){
        printf("შეუძლებელია FIFO-ს შექმნა\n");
        exit(EXIT_FAILURE);
    }
    // პროცესის შექმნა
    if( (pr = (int) fork()) == -1){
        printf("შეუძლებელია პროცესის წარმოქმნა\n");
        exit(EXIT_FAILURE);
    } else {
        if(pr != 0){
            // გავხსნათ FIFO არხი მონაცემების ჩასაწერად
            fd = open(name, O_WRONLY);
            if(fd < 0){
                printf("შეუძლებელია FIFO-ს გახსნა\n");
                exit(EXIT_FAILURE);
            }
            // FIFO არხი ჩავწეროთ მონაცემები
            s1 = write(fd, mass, s2);
            if(s1 != s2){
                printf("სრულად არ ჩაიწერა მონაცემები\n");
                exit(EXIT_FAILURE);
            }
            close(fd); // დავხურო კავშირის არხი
            wait(&st); // მომდელი ელოდება შვილის დასრულებას
        } else {
            // გავხსნათ FIFO არხი მონაცემების წარმოქმნად
            fd = open(name, O_RDONLY);
```

```

if (fd < 0){
    printf("შეუძლებელი FIFO-ს გახსნა\n");
    exit(EXIT_FAILURE);
}
// FIFO არხიდან წავიკითხოთ მონაცემები
s1 = read(fd, re_mass, s2);
if (s1 != s2){
    printf("ინფორმაცია სრულად ვერ იქნა წაკითხული \n");
    exit(EXIT_FAILURE);
}
// გადაცემული მონაცემების ბეჭდვა
printf("\n\t%s\n",re_mass);
close(fd); // დავხურო კავშირის არხი
}
exit(EXIT_SUCCESS);
}

```

პროგრამა 04-4.c. FIFO-ს გამოყენებით ერთმიმართულებიანი კავშირი მშობელ- და შვილ-პროცესს შორის

აკრიფეთ პროგრამა, დააკომპილირეთ და შეასრულეთ ის.

შევნიშნოთ, რომ ამ პროგრამის ყოველი შემდეგი შესრულება მიგვიყვანს შეცდომამდე, რაც გამოწვეულია იმითი, რომ პროგრამით შექმნილი fifo გაფართოების სპეციალური ფაილი უკვე არსებობს სამუშაო დირექტორიაში. პრობლემის გადაწყვეტა მოცემულ შემთხვევაში წარმოადგენს მისი შესრულების წინ ამ ფაილის წაშლა დირექტორიიდან, ან პირველი შესრულების შემდეგ განხორციელდებს პროგრამის ტექსტის რედაქტირება: იქიდან წაიშალოს ის ნაწილი, რომელიც დაკავშირებულია FIFO არხის წარმოქმნასთან ან მშობელ პროცესს მოვთხოვოთ, რომ სანამ დასრულდებოდეს წაშალოს FIFO არხი.

სემინარი 5. SystemVIPC საშუალებები. განაწილებადი მეხსიერება

წინა სემინარზე ჩვენ გავეცანით პროცესებს შორის კავშირის არხებით მონაცემების გადაცემის მექანიზმებს, კერძოდ, pipe-ს და FIFO-ს. კავშირის არხებით მონაცემების გაცვლის ეს მექანიზმები საკმაოდ ადვილია გამოყენებაში, მაგრამ გააჩნიათ მთელი რიგი ნაკლოვანებები:

- კითხვისა და ჩაწერის ოპერაციები არ აანალიზებენ გადასაცემი მონაცემების შემცველობას. ისინი არ ინტერესდებიან მიღებული ინფორმაცია გადაცემულია ერთი წყაროდან თუ ის სხვადასხვა წყაროდან შემოვიდა;
- პროცესებს შორის მონაცემების გადაცემისას საჭიროა კოპირების მინიმუმ ორი ოპერაცია: მონაცემების გადამცემი პროცესის მისამართების სივრციდან სისტემურ ბუფერში და სისტემური ბუფერიდან მიმღები პროცესის მისამართების სივრცეში.
- ინფორმაციის გაცვლის მომენტში გადამცემი და მიმღები პროცესები ერთდროულად უნდა არსებობდნენ სისტემაში. შეუძლებელია გადამცემა პროცესმა კავშირის არხში განათავსოს მონაცემები, ხოლო გარკვეული დროის შემდეგ კი მიმღებმა პროცესმა განახორციელოს ინფორმაციის კითხვა არხიდან.

5.1. SystemVIPC-ის ცნება

კავშირის არხებით ინფორმაციის გადაცემის აღწერილმა ნაკლოვანებებმა საჭირო გახადა შექმნილიყო პროცესებს შორის ინფორმაციის გადაცემის სხვა მექანიზმები. ამ მექანიზმების ნაწილმა, რომელიც პირველად გამოჩნდა *UNIX System V*-ში და შემდეგ პრაქტიკულად UNIX ოპერაციული სისტემის ყველა თანამედროვე ვერსიაში, მიიღო სახელწოდება *SystemVIPC* (*IPC - inter process communications*). *SystemVIPC* ჯგუფში შედის: შეტყობინებათა მიმდევრობები, განაწილებადი მეხსიერება და სემაფორები. პროცესების ურთიერთქმედების ორგანიზაციის ეს საშუალებები დაკავშირებულია არამხოლოდ წარმოშობის ზოგადობით, არამედ მსგავსი ოპერაციების (მაგალითად, სისტემაში რესურსების გამოყოფა და გამონთავისუფლება) შესრულებისას მათ გააჩნიათ მსგავსი ინტერფეისი.

5.1.1. SystemVIPC სახელების სივრცე

SystemVIPC-დან კავშირის ყველა საშუალება, ისევე, როგორც ჩვენს მიერ განხილული pipe და FIFO, წარმოადგენს კავშირის საშუალებას არაპირდაპირი დამისამართებით. როგორც ზემოთ აღვნიშნეთ, არამონათესავე პროცესებს შორის კავშირის არხის მეშვეობით ურთიერთქმედების ორგანიზაციისათვის არაპირდაპირი დამისამართებისას საჭიროა, რომ კავშირის ამ საშუალებას გააჩნდეს სახელი. pipe-ისათვის სახელის არქონა მონათესავე პროცესებს შესაძლებლობას აძლევს მიიღონ ინფორმაცია სისტემაში მის ადგილმდებარეობასა და მდგომარეობაზე მხოლოდ მემკვიდრე კავშირებით. ფაილურ სისტემაში FIFO-სათვის სახელის მინიჭების (სისტემაში დარეგისტრირების) შესაძლებლობა არამერკვიდრე პროცესებს საშუალებას აძლევს მიიღონ ეს ინფორმაცია ფაილური სისტემის ინტერფეისის მეშვეობით.

რაიმე სახის ობიექტების შესაძლო სახელების სიმრავლეს ეწოდება შესაბამისი ტიპის ობიექტების სახელების სივრცე. FIFO-სათვის სახელების სივრცეს წარმოადგენს ფაილურ სისტემაში ფაილების შესაძლო სახელების სიმრავლე. *SystemVIPC*-სათვის სახელების ასეთ სივრცეს წარმოადგენს მონაცემთა გარკვეული მთელმნიშვნელობიანი რიცხვების (key_t

გასაღებების) სიმრავლე. ამასთან პროგრამისტს არ შეუძლია პირდაპირ მიანიჭოს გასაღების მნიშვნელობა, ეს მნიშვნელობა მოიცემა გასაშუალოებით: ფაილურ სისტემაში არსებული გარკვეული ფაილის სახელის და პატარა მთელი რიცხვის კომბინაციით, მაგალითად, კავშირის საშუალების ასლის ნომრით.

გასაღების მნიშვნელობის მიღების ასეთი მეთოდი დაკავშირებულია შემდეგ მოსაზრებასთან:

- თუ პროგრამისტს ექნება კავშირის საშუალებისათვის საიდენტიფიკაციო ნომრის მინიჭების შესაძლებლობა, მაშინ გამორიცხული არაა, რომ ორმა პროგრამისტმა შემთხვევით გამოიყენოს ერთიდაიგივე რიცხვითი მნიშვნელობა. ამ შემთხვევაში, სხვადასხვა პროცესები შეეცდებიან არასანქცირებულად გამოიყენონ კომუნიკაციის საშუალებები, რამაც შეიძლება მიგვიყვანოს პროცესების არასტანდარტულ ყოფაქცევამდე. ამიტომ გასაღების მნიშვნელობის ძირითად კომპონენტს წარმოადგენს იმ ფაილის სრული სახელის გარდაქმნა რიცხვით მნიშვნელობაში, რომელზეც პროცესებს გააჩნიათ წვდომა. ამ მიზნით პროგრამისტს შეუძლია გამოიყენოს საკუთარი სპეციფიური ფაილი, მაგალითად, შესრულებადი ფაილი, რომელიც დაკავშირებულია ურთიერთქმედ პროცესებიდან ერთერთთან. შევნიშნოთ, რომ პროცესების ურთიერთქმედების განმავლობაში არ უნდა ხდებოდეს გასაღების ფორმირებისათვის გამოყენებული ფაილის ადგილმდებარეობის შეცვლა;
- გასაღების მნიშვნელობის მეორე კომპონენტი გამოიყენება იმისათვის, რომ პროგრამისტს ჰქონდეს შესაძლებლობა ფაილის ერთსა და იმავე სახელს დაუკავშიროს კავშირის საშუალების ერთზე მეტი ასლი. ასეთი კომპონენტის როლში შესაძლებელია გამოდიოდეს შესაბამის ასლის რიგითი ნომერი.

ორი კომპონენტისგან გასაღების მნიშვნელობის მიღება ხორციელდება ფუნქციით ***ftok()***.

***ftok()* ფუნქციის პროტოტიპი**

```
#include <sys/types.h>
#include <sys/ipc.h>
key_t ftok(char * pathname, char proj_id);
```

ფუნქციის აღწერა

ftok ფუნქცია გამოიყენება არსებული ფაილის სახელის პატარა მთელ რიცხვში გარდაქმნისათვის, მაგალითად, კავშირის საშუალების ეზემპლარის რიგითი ნომრის SystemVIPC გასაღებში.

pathname პარამეტრი უნდა იყოს მიმთითებელი არსებულ ფაილზე, რომელზეც წვდომა გააჩნია *ftok* ფუნქციის გამომახებელ პროცესს.

proj_id - ეს არის პატარა მთელი რიცხვი, რომელიც ახასიათებს კავშირის საშუალების ასლს. გასაღების გენერაციის შეუძლებლობის შემთხვევაში ფუნქცია აბრუნებს უარყოფით მნიშვნელობას წინააღმდეგ შემთხვევაში აბრუნებს გენერირებული გასაღების მნიშვნელობას.

key_t მონაცემთა ტიპი წარმოადგენს 32-თანრიგა მთელ რიცხვს.

აღვნიშნოთ გასაღების მისაღებად ფაილის გამოყენების სამი მნიშვნელოვანი მომენტი:

- გასაღების გენერირებისათვის საჭიროა ფაილურ სისტემაში არსებული რეალური ფაილის სახელი, რომელზეც ინფორმაციის გამცვლელ პროცესებს ექნება კითხვის უფლება;

- გასაღების გენერირებისათვის გამოყენებული ფაილი არ უნდა იცვლიდეს ადგილმდებარებას პროცესებს შორის ურთიერთქმედების დასრულებამდე;
- გასაღების გენერირებისათვის ფაილის სახელის მითითება არ ნიშნავს, რომ კავშირის საშუალებით გადასაცემი ინფორმაცია განთავსებული იქნება ამ ფაილში. ფაილის სახელის მითითება სხვადასხვა პროცესებს საშუალებას აძლევს დააგენერირონ იდენტური გასაღებები და ინფორმაციის გაცვლისათვის გამოიყენონ ოპერაციული სისტემის მიერ სპეციალურად ამ მიზნისათვის გამოყოფილი მისამართების სივრცე.

5.1.2. SystemVIPC დესკრიპტორი

როგორც ადრე აღვნიშნეთ, ოპერაციული სისტემა პროცესთან დაკავშირებულ ფაილებსა და კავშირის არხებზე ინფორმაციას ინახავს ფაილური დესკრიპტორების ცხრილში. სისტემური გამოძახება, რომლიც ანხორციელებს კავშირის არხით მონაცემების გადაცემას, პარამეტრის როლში იყენებს ფაილური დესკრიპტორების ცხრილის შესაბამის ჩანაწერს (ფაილურ დესკრიპტორს). ფაილური დესკრიპტორების გამოყენება პროცესების შიგნით იძლევა კავშირის არხის იდენტიფიცირების და ფაილებთან მუშაობისათვის უკვე არსებული ინტერფეისის გამოყენების შესაძლებლობას. პროცესის დასრულებისას მას მივყავართ კავშირის არხის ავტომატურ დახურვამდე რითაც აიხსნება კავშირის არხებით ინფორმაციის გადაცემის ზემოთ აღწერილი ნაკლოვანებები.

System VIPC-ის კომპონენტების რეალიზაციისას მიღებული იყო სხვა კონცეფცია. ერთერთი მათგანი მდგომარეობს შემდეგში: ოპერაციული სისტემის ბირთვი ინფორმაციას სისტემაში გამოყენებულ ყველა *System VIPC* საშუალებებზე ინახავს მომხმარებლის კონტექსტის გარეთ. თუ პროცესი საჭიროებს კავშირის საშუალების გამოყენებას, მაშინ ის ღებულობს წარმოქმნილი ან არსებული კავშირის საშუალების მაიდენტიფიცირებელ არაუარყოფით მნიშვნელობას (დესკრიპტორს) და სარგებლობს ამ მნიშვნელობის შესაბამისი კავშირის საშუალებით. დესკრიპტორის მნიშვნელობა სისტემურ გამოძახებას, რომელიც *System VIPC* საშუალებებზე ანხორციელებს შემდგომ ოპერაციებს უნდა გადაეცემოდეს პარამეტრის სახით.

მსგავსი კონცეფცია იძლევა ნაკადების მეშვეობით კავშირის საშუალებების ერთერთ ყველაზე მნიშვნელოვანი ნაკლოვანების (ურთიერთქმედი პროცესების ერთდროულად არსებობის) აცილების შესაძლებლობას, მაგრამ იმავდროულად საჭიროებს განსაკუთრებულ სიფრთხილეს იმისათვის, რომ პროცესმა, რომელმაც მიიღო ინფორმაცია, ახალი ინფორმაციის ნაცვლად არ გაითვალისწინოს ძველი, რომელიც შემთხვევით დარჩა კომუნიკაციის მექანიზმში.

5.2. განაწილებადი მეხსიერება

პროცესებს შორის კავშირის ორგანიზაციის უზრუნველსაყოფად *System VIPC* კავშირის საშუალებები წინასწარ ითხოვენ ინიციალიზირებად მოქმედებებს. გარკვეული გასაღებით განაწილებადი მეხსიერების მიდამოს შექმნისათვის ან უკვე არსებულ განაწილებად მიდამოზე დაშვებისათვის გამოიყენება სისტემური გამოძახება *shmget()*. ახალი განაწილებადი მეხსიერების შესაქმნელად *shmget()* სისტემური გამოძახების გამოყენების არსებობს ორი მეთოდი:

- **სტანდარტული მეთოდი.** სისტემურ გამოძახებაში გასაღების მნიშვნელობის როლში

ყენდება *ftok()* ფუნქციის მიერ გარკვეული ფაილის სახელისათვის და განაწილებადი მეხსიერების მიდამოს ასლის ნომრისათვის გენერირებული მნიშვნელობა. flag-ების როლში ყენდება შესაქმნელ სეგმენტზე დაშვების უფლებების და *IPC_CREAT* flag-ის კომბინაციები. თუ სეგმენტი მოცემული გასაღებისათვის არ არსებობს, მაშინ სისტემა შეეცდება მის შექმნას დაშვების მითითებული უფლებებით. თუ ის არსებობს, მაშინ ვიღებთ მის დესკრიპტორს. flag-ების ამ კომბინაციისათვის შესაძლებელია *IPC_EXCL* flag-ის დამატება. ეს flag-ი გარანტირებს სისტემური გამოძახების ნორმალურ დასრულებას მხოლოდ იმ შემთხვევაში, თუ სეგმენტი მართლა შეიქმნა (ანუ ის ადრე არ არსებობდა), თუ სეგმენტი არსებობდა, მაშინ სისტემური გამოძახება დასრულდება შეცდომით და *<errno.h>* ფაილში აღწერილი *errno* სისტემური ცვლადის მნიშვნელობა შეიცვლება *EEXIST*-ით (*errno = EEXIST*).

- **არასტანდარტული მეთოდი.** გასაღების მნიშვნელობის როლში ეთითება სპეციალური მნიშვნელობა *IPC_PRIVATE*. *IPC_PRIVATE* მნიშვნელობის გამოყენებას ყოველთვის მივყავართ განაწილებადი მეხსიერების ახალი სეგმენტის შექმნის მცდელობამდე მითითებული დაშვების უფლებებით და გასაღებით, რომელიც არ ემთხვევა არცერთი უკვე არსებული სეგმენტის გასაღების მნიშვნელობას და შეუძლებელია მიღებული იყოს *ftok()* ფუნქციის მეშვეობით. ამ შემთხვევაში ხდება *IPC_CREAT* და *IPC_EXCL* flag-ების მნიშვნელობების იგნორირება.

shmget() სისტემური გამოძახების პროტოტიპი

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(key_t key, int size, int shmlflg);
```

სისტემური გამოძახების აღწერა

shmget სისტემური გამოძახება გამოიყენება განაწილებადი მეხსიერების სეგმენტზე დაშვებისათვის და მისი წარმატებით დასრულების შემთხვევაში ის აბრუნებს ამ სეგმენტისათვის *System VIPC* დესკრიპტორს¹.

key პარამეტრი წარმოადგენს *System VIPC* გასაღებს სეგმენტისათვის, ანუ ფაქტიურად მის სახელს *System VIPC* სახელების სივრციდან. ამ პარამეტრის მნიშვნელობის როლში შეიძლება გამოყენებული იყოს *ftok()* ფუნქციის მეშვეობით მიღებული გასაღების მნიშვნელობა ან სპეციალური მნიშვნელობა *IPC_PRIVATE*. *IPC_PRIVATE* მნიშვნელობის გამოყენებას ყოველთვის მივყავართ განაწილებადი მეხსიერების ახალი სეგმენტის შექმნის მცდელობამდე გასაღებით, რომელიც არ ემთხვევა არსებული სეგმენტის გასაღების მნიშვნელობებს და არ შეიძლება მიღებული იყოს *ftok()*-ის მეშვეობით.

size პარამეტრი განსაზღვრავს არსებული ან შესაქმნელი სეგმენტის მოცულობას ბაიტებში. თუ სეგმენტი მითითებული გასაღებით უკვე არსებობს, მაგრამ მისი მოცულობა არ ემთხვევა size პარამეტრით მითითებულ მნიშვნელობას წარმოიშობა შეცდომა.

shmlflg პარამეტრი გამოიყენება მხოლოდ განაწილებადი მეხსიერების ახალი სეგმენტის შექმნისას და განსაზღვრავს მომხმარებლების სეგმენტზე დაშვების უფლებებს, ასევე ახალი სეგმენტის შექმნის აუცილებლობას და სისტემური გამოძახების ყოფაქცევას შექმნის მცდელობისას. ის წარმოადგენს შემდეგი მნიშვნელობების გარკვეულ კომბინაციას (ბიტური ოპერაციით ან ('|')):

¹ მთელი არაურყოფითი მნიშვნელობა, რომელიც ცალსახად ახასიათებს გამოთვლითი სისტემის შიგნით სეგმენტს და მომავალში გამოიყენება მასთან სხვა ოპერაციებისას

- *IPC_CREAT* – თუ სეგმენტი მითითებული გასაღებისათვის არ არსებობს, მაშინ ის უნდა შეიქმნას;
- *IPC_EXCL* – გამოიყენება *IPC_CREAT* flag-თან ერთად. მათი ერთობლივი გამოყენების და მითითებული გასაღებით სეგმენტის არსებობისას, არ ხორციელდება სეგმენტზე დაშვება და წარმოიშობა შეცდომის შემცველი სიტუაცია, ამასთან <errno.h> ფაილში აღწერილი *errno* ცვლადი დებულობს მნიშვნელობას *EEXIST*;
- 0400 – სეგმენტის შემქმნელი მომხმარებლისათვის ნებადართულია კითხვა;
- 0200 – სეგმენტის შემქმნელი მომხმარებლისათვის ნებადართულია ჩაწერა;
- 0040 – სეგმენტის შემქმნელი მომხმარებლის ჯგუფისათვის ნებადართულია კითხვა;
- 0020 – სეგმენტის შემქმნელი მომხმარებლის ჯგუფისათვის ნებადართულია ჩაწერა;
- 0004 – სხვა მომხმარებლისათვის ნებადართულია კითხვა;
- 0002 – სხვა მომხმარებლისათვის ნებადართულია ჩაწერა.

დასაბრუნებელი მნიშვნელობა

სისტემური გამოძახება წარმატებით დასრულებისას განაწილებადი მეხსიერების სეგმენტისათვის აბრუნებს *System V IPC* დესკრიპტორის მნიშვნელობას და -1-ს შეცდომის წარმოქმნის შემთხვევაში.

შექმნილ განაწილებად მეხსიერებაზე წვდომა ხორციელდება მისი დესკრიპტორით, რომელსაც აბრუნებს სისტემური გამოძახება *shmget()*. უკვე არსებულ მიდამოზე დაშვება შეიძლება განხორციელდეს შემდეგი ორი მეთოდით:

- თუ ვიყენებთ პროცესისათვის გენერირებულ გასაღებს, მაშინ *shmget()* გამოძახების გამოყენებით შეგვიძლია მივიღოთ განაწილებადი მეხსიერების დესკრიპტორი. ამ შემთხვევაში არ შეიძლება flag-ების შემადგენელ ნაწილად *IPC_EXCL* flag-ის მნიშვნელობის მითითება, ხოლო გასაღების მნიშვნელობა კი არ შეიძლება იყოს *IPC_PRIVATE*. ხდება დაშვების უფლებების იგნორირება, ხოლო მიდამოს მოცულობა უნდა ემთხვეოდეს მისი შექმნისას მითითებულ მოცულობას.
- ან შეგვიძლია ვისარგებლოთ იმითი, რომ *System V IPC* დესკრიპტორი ნამდვილია ოპერაციული სისტემის ფარგლებში და გადავცეთ მისი მნიშვნელობა განაწილებადი მეხსიერების შემქმნელი პროცესებიდან მიმდინარე პროცესს.

დესკრიპტორის მიღების შემდეგ საჭიროა განაწილებადი მეხსიერების მიდამოს განთავსება მიმდინარე პროცესის მისამართების სივრცეში. ეს ხორციელდება *shmat()* სისტემური გამოძახების მეშვეობით. ნორმალურად დასრულების შემთხვევაში ის აბრუნებს მიმდინარე პროცესის მისამართების სივრცეში განაწილებადი მეხსიერების მისამართს. შემდგომი დაშვება ამ მისამართზე ხორციელდება პროგრამირების ენის ჩვეულებრივი საშუალებებით.

shmat() სისტემური გამოძახების პროტოტიპი

```
#include<sys/types.h>
#include <sys/shm.h>
char *shmat(int shmid, char *shmaddr, int shmflg);
```

სისტემური გამოძახების აღწერა

shmat სისტემური გამოძახება გამოიყენება პროცესის მისამრთების სივრცეში განაწილებადი მეხსიერების სეგმენტის განსათავსებლად.

shmid პარამეტრი წარმოადგენს განაწილებადი მეხსიერების სეგმენტისათვის *System V IPC* დესკრიპტორს, ანუ *shmget()* სისტემური გამოძახების მიერ დაბრუნებულ მნიშვნელობას.

shmaddr პარამეტრის როლში ვიყენებთ მნიშვნელობას NULL, რითაც ოპერაციულ სისტემას ეძლევა შესაძლებლობა განათავსოს განაწილებადი მეხსიერება პროცესის მისამართების სივრცეში.

shmflg პარამეტრისათვის გამოვიყენებთ ორ მნიშვნელობას: 0 - კითხვის დაჩაწერის ოპერაციების განსახორციელებლად ან SHM_RDONLY - თუ გვინდა მისგან მხოლოდ კითხვა. ამასთან პროცესს უნდა გააჩნდეს სეგმენტზე დაშვების შესაბამისი უფლება.

დასაბრუნებელი მნიშვნელობა

სისტემური გამოძახება წარმატებით დასრულების შემთხვევაში აბრუნებს განაწილებადი მეხსიერების მისამართს პროცესის მისამართების სივრცეში და -1-ს შეცდომის წარმოქმნის შემთხვევაში.

პროცესს განაწილებადი მეხსიერების გამოყენების დასრულების შემდეგ შეუძლია საკუთარი მისამართების სივრციდან ამოშალოს განაწილებადი მეხსიერება. განაწილებადი მეხსიერების ამოსაშლელად გამოიყენება **shmdt()** სისტემური გამოძახება. შევნიშნოთ, რომ **shmdt()** სისტემურ გამოძახება პარამეტრის მნიშვნელობის როლში ითხოვს პროცესის მისამართების სივრცეში განაწილებადი მეხსიერების მიდამოს დასაწყისის მითითებას, ანუ მნიშვნელობას, რომელიც დააბრუნა **shmat()** სისტემურმა გამოძახებამ, ამიტომ საჭიროა მოცემული მნიშვნელობის შენახვა განაწილებადი მეხსიერების გამოყენების მთელს პერიოდში.

shmdt() სისტემური გამოძახების პროტოტიპი

```
#include<sys/types.h>
#include <sys/shm.h>
int shmdt(char *shmaddr);
```

სისტემური გამოძახების აღწერა

shmdt სისტემური გამოძახება გამოიყენება მიმდინარე პროცესის მისამართების სივრციდან განაწილებადი მეხსიერების სეგმენტის ამოსაშლელად.

shmaddr პარამეტრი წარმოადგენს განაწილებადი მეხსიერების სეგმენტის მისამართს ანუ, მნიშვნელობას რომელიც დააბრუნა სისტემურმა გამოძახებამ **shmat()**.

დასაბრუნებელი მნიშვნელობა

სისტემური გამოძახება აბრუნებს 0-ს ნორმალურად დასრულებისას და -1-ს შეცდომის წარმოქმნის შემთხვევაში.

განაწილებადი მეხსიერების ორგანიზების საილუსტრაციოდ განვიხილოთ მაგალითი. ვთქვათ, გვაქვს ორი პროცესი, რომლებიც იყენებენ განაწილებად მეხსიერებას. განაწილებად მეხსიერებაში განვათავსებთ სამელემენტიან მასივს, რომლის პირველ ორ ელემენტში ჩაიწერება, შესაბამისად, პირველი და მეორე პროცესის (ცალცალკე) შესრულებათა რაოდენობა, ხოლო მესამეში მათი შესრულებათა საერთო რაოდენობა.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
int main(){
```

```

int *mass; /* მიმთითებელი განაწილებად მეხსიერებაზე */
int shm_id; /* ცვლადი განაწილებადი მეხსიერების სეგმენტის იდენტ. */
int flag = 1; /* მასივის ელემენტების ინიციალიზაციის აუცილებლობის flag-ი */
/* ფაილი, რომელსაც გავიყენებთ SystemVIPC გასაღების ფორმირებისათვის */
char name[] = "prog-5.1.c";
key_t key;
/* ფაილისათვის SystemVIPC გასაღების გენერირება */
if ((key = ftok(name,0)) < 0){
    perror("შეუძლებელია გასაღების გენერირება\n");
    exit(EXIT_FAILURE);
}
/* განაწილებადი მეხსიერების შემქნა */
shm_id = shmget(key, 3*sizeof(int), IPC_CREAT | IPC_EXCL | 0644);
if (shm_id < 0){
    /* შეცდომა გამოწვეულია განაწილებადი მეხსიერების არსებობით */
    if (errno == EEXIST){
        /* არსებული განაწილებადი მეხსიერების დესკრიპტორის მიღება */
        shm_id = shmget(key, 3*sizeof(int), 0);
        if(shm_id < 0){
            perror("შეუძლებელია განაწილებადი მეხსიერების მოძებნა\n");
            exit(EXIT_FAILURE);
        }
        flag = 0;
    }
    else { /* შეცდომა გამოწვეულია გაურკვეველი მიზეზით */
        perror("შეუძლებელია განაწილებადი მეხსიერების შექმნა\n");
        exit(EXIT_FAILURE);
    }
}
/* მასივის განთავსება განაწილებად მეხსიერებაში */
mass = (int *) shmat(shm_id, NULL, 0);
if(mass == (int *)(-1)) {
    perror("შეუძლებელია მასივის განთავსება განაწილებად მეხსიერებაში\n");
    exit(EXIT_FAILURE);
}
if(flag){
    mass[0] = 1; /* პროცესი 1-ის შესრულებათა დასათვლელად */
    mass[1] = 0; /* პროცესი 2-ის შესრულებათა დასათვლელად */
    mass[2] = 1; /* ორივე პროცესის ჯამური შესრულებების დასათვლელად */
} else {
    mass[0] += 1;
    mass[2] += 1;
}
printf("პროცესი 1-ის შესრულებათა რაოდენობა \t%d\n", mass[0]);
printf("პროცესი 2-ის შესრულებათა რაოდენობა \t%d\n", mass[1]);
printf("ორივე პროცესის შესრულებათა ჯამური რაოდენობა %d\n", mass[2]);
/* პროცესის მისამართების სივრციდან განაწილებადი მეხსიერების წაშლა */
if (shmdt(mass) < 0){

```

```

        perror("შეუძლებელია განაწილებადი მეხსიერების წაშლა\п");
        exit(EXIT_FAILURE);
    }
    exit(EXIT_SUCCESS);
}

```

prog-5.1.c

```

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
int main(){
    int *mass; /* მიმთითებელი განაწილებად მეხსიერებაზე */
    int shm_id; /* ცვლადი განაწილებადი მეხსიერების სეგმენტის იდენტ. */
    int flag = 1; /* მასივის ელემენტების ინიციალიზაციის აუცილებლობის flag-ი */
    /* ფაილი, რომელსაც გავიყენებთ SystemVIPC გასაღების ფორმირებისათვის */
    char name[] = "prog-5.1.c";
    key_t key;
    /* ფაილისათვის SystemVIPC გასაღების გენერირება */
    if(key = ftok(name,0) < 0){
        perror("შეუძლებელია გასაღების გენერირება\п");
        exit(EXIT_FAILURE);
    }
    /* განაწილებადი მეხსიერების შემქნა */
    shm_id = shmget(key, 3*sizeof(int), IPC_CREAT | IPC_EXCL | 0644);
    if(shm_id < 0){
        /* შეცდომა გამოწვეულია განაწილებადი მეხსიერების არსებობით */
        if(errno == EEXIST){
            /*არსებული განაწილებადი მეხსიერების დესკრიპტორის მიღება */
            shm_id = shmget(key, 3*sizeof(int), 0);
            if(shm_id < 0){
                perror("შეუძლებელია განაწილებადი მეხსიერების მოძებნა\п");
                exit(EXIT_FAILURE);
            }
            flag = 0;
        }
        else { /* შეცდომა გამოწვეულია გაურკვეველი მიზეზით */
            perror("შეუძლებელია განაწილებადი მეხსიერების შექმნა\п");
            exit(EXIT_FAILURE);
        }
    }
    /* მასივის განთავსება განაწილებად მეხსიერებაში */
    mass = (int *) shmat(shm_id, NULL, 0);
    if(mass == (int *)(-1)){
        perror("შეუძლებელია მასივის განთავსება განაწილებად მეხსიერებაში\п");

```

```

        exit(EXIT_FAILURE);
    }
    if (flag){
        mass[0] = 0; /* პროცესი 1-ის შესრულებათა დასათვლელად */
        mass[1] = 1; /* პროცესი 2-ის შესრულებათა დასათვლელად */
        mass[2] = 1; /* ორივე პროცესის ჯამური შესრულებების დასათვლელად */
    } else {
        mass[1] += 1;
        mass[2] += 1;
    }
    printf("პროცესი 1-ის შესრულებათა რაოდენობა \t%d\n", mass[0]);
    printf("პროცესი 2-ის შესრულებათა რაოდენობა \t %d\n", mass[1]);
    printf("ორივე პროცესის შესრულებათა ჯამური რაოდენობა %d\n", mass[2]);
    /* პროცესის მისამართების სივრციდან განაწილებადი მეხსიერების წაშლა */
    if (shmctl(mass) < 0){
        perror("შეუძლებელია განაწილებადი მეხსიერების წაშლა\p");
        exit(EXIT_FAILURE);
    }
    exit(EXIT_SUCCESS);
}

```

prog-5.2.c

ეს ორი პროგრამა ძალიან წააგავს ერთიმეორებს. ისინი განაწილებად მეხსიერებას იყენებენ თითოეული პროგრამის შესრულების რაოდენობის და მათი შესრულების საერთო რაოდენობის შესანახად. განაწილებადი მეხსიერების სეგმენტში განთავსებულია სამელემენტიანი მასივი. მასივის პირველი ელემენტი გამოიყენება, როგორც პროგრამა 1-ის შესრულების მთვლელი, მეორე ელემენტი - პროგრამა 2-ის შესრულების მთვლელი, ხოლო მესამე ელემენტი კი მათი ჯამისათვის. დამატებითი ნიუანსი პროგრამაში წარმოიშობა განაწილებადი მეხსიერების შექმნისას მასივის ელემენტების ინიციალიზაციის აუცილებლობით. ამისათვის გვჭირდება, რომ პროგრამებმა შეძლონ განსხვავება შემთხვევისა როდის შექმნეს მათ ის და შემთხვევისა, ის რომ არსებობდა. განსხავებას ვაღწევთ დასაწყისში *shmget()* სისტემური გამოძახების გამოყენებით flag-ებით *IPC_CREAT* და *IPC_EXCL*. თუ გამოძახება დასრულდება წარმატებით, მაშინ შეიქმნება განაწილებადი მეხსიერება. თუ გამოძახება დასრულდა შეცდომით და *errno* ცვლადმა მიიღო მნიშვნელობა *EEXIST*, მაშინ ეს ნიშნავს, რომ განაწილებადი მეხსიერება უკვე არსებობს და შეგვიძლია მივიღოთ მისი IPC დესკრიპტორი იმავე გამოძახების გამოყენებით flag-ის ნულოვანი მნიშვნელობის მითითებით.

5.3. განაწილებადი მეხსიერების ყოფაქცევა

მნიშვნელოვან საკითხს წარმოადგენს განაწილებადი მეხსიერების სეგმენტის ყოფაქცევა სხვადასხვა სისტემური გამოძახებების შესრულებისას, კერძოდ, *fork()* და *exec()* სისტემური გამოძახებებისა და *exit()* ფუნქციის შესრულებისას.

fork() სისტემური გამოძახების შესრულებისას შვილი პროცესი მემკვიდრეობით იღებს ინფორმაციას მშობელი პროცესისაგან მის მისამართების სივრცეში განთავსებულ ყველა განაწილებადი მეხსიერების მიდამოზე.

`exec()` სისტემური გამოძახებისა და `exit()` ფუნქციის შესრულებისას პროცესის მისამართების სივრცეში განთავსებული განაწილებადი მეხსიერების მიდამოები იშლება იქიდან, მაგრამ არსებობას განაგრძობენ ოპერაციულ სისტემაში.

ამოდანა 1. დაწერეთ პროგრამა, რომელშიც პროცესი შექმნის ორ შვილ პროცესს. პირველი შვილი პროცესი ფაილიდან წაიკითხავს N მთელ მნიშვნელობას და ჩაწერს განაწილებად მეხსიერებაში, ხოლო მეორე შვილი პროცესი დაითვლის ამ მონაცემთა საშუალო არითმეტიკულს და დაბეჭდავს

```
#include <fstream>
#include <iostream>
#include <errno.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <sys/wait.h>
#include <sys/types.h>
using namespace std;
int main(){
    int N = 10; // ფაილში ელემენტების რაოდენობა
    int *arrow; // ამ მისამართზე განთავსდება განაწილებადი მეხსიერება
    int shmid; // ცვლადი განაწილებადი მეხსიერების დესკრიპტორისათვის
    int st1, st2; // სტატუსის ცვლადები
    pid_t p1, p2; // ცვლადების პროცესის შექმნის შესამოწმებლად
    key_t key; // ცვლადი გასაღების მნიშვნელობისათვის
    // მოვახდინოთ გასაღების გენერაცია SheMem.cpp ფაილისათვის და შემოწმება
    key = ftok("prog-5.3.cpp", 0);
    if (key == -1) {
        cout << "გასაღები მითითებული ფაილისათვის არ შეიქმნა.\n";
        exit(EXIT_FAILURE);
    }
    // გენერირებული გასაღებისათვის შევქმნათ საჭირო მოცულობის განაწილებადი მეხსიერება
    shmid = shmget(key, N*sizeof(int), IPC_CREAT | IPC_EXCL | 0600);
    if (shmid == -1){ // თუ განაწილებადი მეხსიერება არ შეიქმნა
        if (errno == EEXIST){ // ასეთი განაწილებადი მეხსიერება ხომ არ არსებობს?
            shmid = shmget(key, N*sizeof(int), 0); // გამოვიყენოთ არსებული განაწილებადი მეხსიერება
            if (shmid == -1){// 
                cout << "ვერ მოხერხდა არსებული განაწილებადი მეხსიერების დესკრიპტორის მიღება.\n";
                exit(EXIT_FAILURE);
            }
        }
    }
    else {
        cout << "განაწილებადი მეხსიერება არ შეიქმნა სხვა მიზეზით.\n";
        exit(EXIT_FAILURE);
    }
}
// შევქმნათ პროცესი
p1 = fork();
if (p1 == -1){
    cout << "პირველი შვილი პროცესი არ შეიქმნა.\n";
    exit(EXIT_FAILURE);
```

```

}

if (p1 == 0){ // პირველი შვილი პროცესი
    // პირველი შვილი პროცესის მისამართების სივრცეს მივაბათ განაწილებადი მეხსიერება
    arrow = (int*)shmat(shmid, NULL, 0);
    if (arrow == (int*)(-1)){
        cout << "ვერ მოხერხდა განაწილებადი მეხსიერების მიერთება"
            << " პროცესის მისამართების სივრცეზე.\n";
        exit(EXIT_FAILURE);
    }
    int tmp; // დროებითი ცვლადი ფაილიდან წაკითხული მონაცემების ჩასაწერად
    ifstream Input("data.in"); // შევქმნათ მონაცემთა ნაკადი
    for (int i = 0; i<N; i++){
        Input >> tmp;
        arrow[i] = tmp;
    }
    Input.close(); // დავხუროთ მონაცემთა ნაკადი
    // განაწილებად მეხსიერებაში მთელი მნიშვნელობების ჩაწერის შემდეგ პროცესის
    // მისამართების სივრციდან ამოვშალოთ განაწილებადი მეხსიერება
    if (shmctl(arrow) == -1)
        cout << "არ მოხდა განაწილებადი მეხსიერების წაშლა.\n";
    // პირველი პროცესის დასრულდა
}
else {
    wait(&st1); // მშობელი პროცესი ელოდება პირველი შვილის დასრულებას
    // შევქმნათ პროცესი
    p2 = fork();
    if (p2 == -1){
        cout << "მეორე შვილი პროცესი არ შეიქმნა.\n";
        exit(EXIT_FAILURE);
    }
    if (p2 == 0){ // მეორე შვილი პროცესი
        // მეორე შვილი პროცესის მისამართების სივრცეს მივაბათ განაწილებადი მეხსიერება
        arrow = (int*)shmat(shmid, NULL, 0);
        if (arrow == (int*)(-1)){
            cout << "ვერ მოხერხდა განაწილებადი მეხსიერების მიერთება"
                << " პროცესის მისამართების სივრცეზე.\n";
            exit(EXIT_FAILURE);
        }
        double average = 0; // საშუალო მნიშვნელობის დასათვლელად
        // average ცვლადში ჩავწეროთ განაწილებად მეხსიერებაში არსებული რიცხვების ჯამი
        for (int i = 0; i<N; i++){
            average += arrow[i];
        }
        // პროცესის მისამართების სივრციდან ამოვშალოთ განაწილებადი მეხსიერება
        if (shmctl(arrow) == -1)
            cout << "არ მოხდა განაწილებადი მეხსიერების წაშლა.\n";
        // დავბეჭდოთ საშუალო არითმეტიკულის მნიშვნელობა
        cout << "average = " << average / N << endl;
        // მეორე პროცესის დასრულდა
    }
    else { // მშობელი პროცესი

```

```

        wait(&st1); // მშობელი პროცესი ელოდება მეორე შვილის დასრულებას
        // მშობელი პროცესი დასრულდა
    }
}
exit(EXIT_SUCCESS);
}

```

prog-5.3.cpp

ამოცანა 2. დაწერეთ პროგრამა, რომელშიც პროცესი შექმნის ორ შვილ პროცესს. პირველი შვილი პროცესი სტრიქონს ჩაწერს განაწილებად მეხსიერებაში, ხოლო მეორე შვილი პროცესი განაწილებად მეხსიერებაში არსებულ მონაცემებს ჩაწერს ფაილში, ხოლო მშობელი პროცესი ტერმინალში გამოიტანს ფაილის შიგთავსი.

```

#include <fstream>
#include <iostream>
#include <errno.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <sys/wait.h>
#include <sys/types.h>
using namespace std;
int main(){
    char str[] = "Operating System."; // ფაილში ელემენტების რაოდენობა
    size_t N = sizeof(str);
    char *arrow; // ამ მისამართზე განთავსდება განაწილებადი მეხიერება
    int shmid; // ცვლადი განაწილებადი მეხსიერების დესკრიპტორისათვის
    int st1, st2; // სტატუსის ცვლადები
    pid_t p1, p2; // ცვლადების პროცესის შექმნის შესამოწმებლად
    key_t key; // ცვლადი გასაღების მნიშვნელობისათვის
    // მოვახდინოთ გასაღების გენერაცია SheMem.cpp ფაილისათვის და შემოწმება
    key = ftok("prog-5.4.cpp", 0);
    if (key == -1) {
        cout << "გასაღები მითითებული ფაილისათვის არ შეიქმნა.\n";
        exit(EXIT_FAILURE);
    }
    // გენერირებული გასაღებისათვის შევქმნათ საჭირო მოცულობის განაწილებადი მეხსიერება
    shmid = shmget(key, N, IPC_CREAT | IPC_EXCL | 0600);
    if (shmid == -1){ // თუ განაწილებადი მეხსიერება არ შეიქმნა
        if (errno == EEXIST){ // ასეთი განაწილებადი მეხსიერება ხომ არ არსებობს?
            shmid = shmget(key, N, 0); // გამოვიყენოთ არსებული განაწილებადი მეხსიერება
            if (shmid == -1){
                cout << "ვერ მოხერხდა არსებული განაწილებადი მეხსიერების დესკრიპტორის მიღება.\n";
                exit(EXIT_FAILURE);
            }
        }
    }
    else {
        cout << "განაწილებადი მეხსიერება არ შეიქმნა სხვა მიზეზით.\n";
        exit(EXIT_FAILURE);
    }
}

```

```

// შევქმნათ პროცესი
p1 = fork();
if (p1 == -1){
    cout << "პირველი შვილი პროცესი არ შეიქმნა.\n";
    exit(EXIT_FAILURE);
}
if (p1 == 0){ // პირველი შვილი პროცესი
    // პირველი შვილი პროცესის მისამართების სივრცეს მივაბათ განაწილებადი მეხსიერება
    arrow = (char*)shmat(shmid, NULL, 0);
    if (arrow == (char*)(-1)){
        cout << "ვერ მოხერხდა განაწილებადი მეხსიერების მიერთება"
            << " პროცესის მისამართების სივრცეზე.\n";
        exit(EXIT_FAILURE);
    }
    // ჩავწეროთ განაწილებად მეხსიერებაში სტრიქონი
    for (int i = 0; i<N; i++){
        arrow[i] = str[i];
    }
    // განაწილებად მეხსიერებაში მთელი მნიშვნელობების ჩაწერის შემდეგ პროცესის
    // მისამართების სივრციდან ამოვშალოთ განაწილებადი მეხსიერება
    if (shmdt(arrow) == -1)
        cout << "არ მოხდა განაწილებადი მეხსიერების წაშლა.\n";
    // პირველი პროცესის დასრულდა
}
else {
    wait(&st1); // მშობელი პროცესი ელოდება პირველი შვილის დასრულებას
    // შევქმნათ პროცესი
    p2 = fork();
    if (p2 == -1){
        cout << "მეორე შვილი პროცესი არ შეიქმნა.\n";
        exit(EXIT_FAILURE);
    }
    if (p2 == 0){ // მეორე შვილი პროცესი
        // მეორე შვილი პროცესის მისამართების სივრცეს მივაბათ განაწილებადი მეხსიერება
        arrow = (char*)shmat(shmid, NULL, 0);
        if (arrow == (char*)(-1)){
            cout << "ვერ მოხერხდა განაწილებადი მეხსიერების მიერთება"
                << " პროცესის მისამართების სივრცეზე.\n";
            exit(EXIT_FAILURE);
        }
        // შევქმნათ მონაცემთა გამოტანის ნაკადი
        ofstream Out("Data.out");
        for (int i = 0; i<N; i++){
            Out << arrow[i];
        }
        Out.close();
        // პროცესის მისამართების სივრციდან ამოვშალოთ განაწილებადი მეხსიერება
        if (shmdt(arrow) == -1)
            cout << "არ მოხდა განაწილებადი მეხსიერების წაშლა.\n";
        // მეორე პროცესის დასრულდა
    }
}

```

```
else { // მშობელი პროცესი
    wait(&st1); // მშობელი პროცესი ელოდება მეორე შვილის დასრულებას
    execl("/bin/cat", "cat", "Data.out", NULL);
    // მშობელი პროცესი დასრულდა
}
exit(EXIT_SUCCESS);
}
```

prog-5.4.cpp

სემინარი 6. ურთიერთბლოკირება

თანამედროვე კომპიუტერი აღჭურვილია მრავალი რესურსით. სისტემაში წარმოქმნილი ყოველი პროცესი მიმართავს მას იმ რესურსის გამოყოფის მოთხოვნით, რომელსაც პროცესი საჭიროებს დასახული ამოცანის გადასაწყვეტად. დროის ნებისმიერ მომენტში კონკრეტული რესურსი შეიძლება გამოყოფს მხოლოდ ერთ პროცესს. ხშირად რამდენიმე პროცესი შეიძლება მიმართავდეს ოპერაციულ სისტემას ერთიდაიმავე რესურსის გამოყოფაზე. პროცესებისათვის საზიარო რესურსის გამოყოფა შესაძლებელია დაკავშირებული იყოს გარკვეულ პრობლემებთან, ამიტომ ოპერაციული სისტემა პროცესებს რესურსებს უყოფს მცირე დროითი შუალედით, მათზე დაშვების სრული უფლებებით.

ხშირად პროცესები საკუთარი სამუშაოს შესასრულებლად საჭიროებენ ერთზე მეტ რესურსს. ოპერაციულმა სისტემამ პროცესის მოთხოვნილი რესურსები შეიძლება გამოყოფს მოთხოვნილი მიმდევრობის შესაბამისად სათითაოდ ან ერთიანად. ორივე მიდგომას აქვს საკუთარი უპირატესობა და ნაკლოვანება. პროცესებისათვის რესურსის გამოყოფის შემთხვევაში შეიძლება წარმოიქმნას სიტუაცია, რომლის დროსაც შეიძლება გამოიყოს პროცესების ჯგუფი, რომლებიც იკავებენ სისტემის გარკვეულ რესურსს და ამავდროულად ითხოვენ ამავე ჯგუფის სხვა პროცესის მიერ დაკავებული რესურსის გამოყოფას. ვინაიდან პროცესმა შეიძლება არ გამოანთავისუფლოს მისთვის გამოყოფილი რესურსი დასრულებამდე, ამიტომ სისტემა ვერ შეძლებს პროცესების ჯგუფის ვერცერთი პროცესის მიერ გაკეთებული მოთხოვნის დაკმაყოფილებას. წარმოქმნილ სიტუაციას ეწოდება **ურთიერთბლოკირება (deadlock) ან ჩიხი.**

6.1. რესურსები

როგორც უკვე აღვნიშნეთ, კომპიუტერულ სისტემას გააჩნია მრავალი რესურსი, რომელთაც ის სთვაზობს პროცესებს შესასრულებელი სამუშაოს განსახორციელებლად. რესურსი შეიძლება იყო აპარატული (მყარი დისკი, პროცესორი, მეხსიერება) ან ინფორმაციული (ფაილები, მონაცემთა ბაზა).

ოპერაციულ სისტემაში წარმოდგენილი ყოველი რესურსი შეიძლება იყოს ორი სახის: **განაწილებადი ან გაუნაწილებელი.** პროცესის მიერ დაკავებულ რესურსს, რომლის ჩამორთმევაც მისთვის შესაძლებელია „უმტკივნეულოდ“ (შესრულებული მნიშვნელოვანი სამუშაოს დაკარგვის გარეშე), მიეკუთვნება განაწილებად რესურსს. გაუნაწილებელი კი პირიქით. მაგალითად, განაწილებადი რესურსის მაგალითს წარმოადგენს მეხსიერება. სისტემაში არასაკმარის მეხსიერების არსებობის შემთხვევაში ხორციელდება პროცესის სრული ან ნაწილობრივი გადატანა მეხსიერების არიდან დისკზე, ხოლო მოგვიანებით კი მისი უკან დაბრუნება.

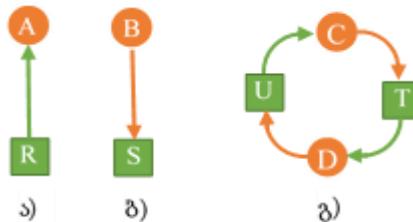
ოპერაციულმა სისტემამ გაუნაწილებელი რესურსი შეიძლება გამოუყოს მხოლოდ ერთ პროცესს. პროცესი ასეთ რესურსს დააბრუნებს მხოლოდ დასრულების შემდეგ.

ურთიერთბლოკირების წარმოშობა სისტემაში დაკავშირებულია იმასთან, რომ რესურსი სისტემაში შეიძლება წარმოდგენილი იყოს მხოლოდ ერთი ასლის სახით. თუ რესურსები იქნებოდა წარმოდგენილი რამდენიმე ასლის სახით, რამდენიმე პროცესისათვის შესაძლებელი იქნებოდა ერთიდაიმავე რესურსის გამოყოფა, დააჩქარებდა პროცესების შესრულებას.

6.2. ურთიერთბლოკირების მოდელირება

ოპერაციულ სისტემას გარკვეული ალგორითმების საშუალებით შეუძლია გააანალიზოს მოთხოვნათა მიმდევრობა. ანალიზის შედეგად (მივყავართ თუ არა მოთხოვნათა მიმდევრობას ურთიერთბლოკირებამდე) მიღებული დასკვნის შესაბამისად ის ღებულობს გადაწყვეტილებას გამოყოს თუ არა პროცესებისათვის შესაბამისი რესურსები.

იმისათვის, რომ გავერკვეთ თუ როგორ აკეთებს ამას ოპერაციული სისტემა დაგვჭირდება ავაგოთ მოთხოვნათა მომდევრობის მოდელი გრაფთა თეორიის გამოყენებით. ყოველ გრაფს გააჩნია ორი სახის კვანძი: პროცესი, რომელიც გამოსახულია წრის ფორმით, და რესურსი, რომელიც გამოსახულია კვადრატის ფორმით. მიმართული მონაკვეთი რესურსიდან პროცესისაკენ აღნიშნავს, რომ პროცესმა მოითხოვა შესაბამისი რესურსი და ის იკავებს მას (ნახ. 6.1. ა)). პროცესიდან რესურსისაკენ მიმართული მონაკვეთი აღნიშნავს, რომ პროცესი ელოდება შესაბამის რესურსს და ამის გამო ის ბლოკირებულია (ნახ. 6.1. ბ)). ნახ. 6.1. გ)-ზე კი გამოსახულია შემდეგი სიტუაცია: C პროცესი იკავებს U რესურსს და ელოდება T რესურსს, რომელსაც იკავებს D პროცესი, ხოლო D პროცესი კი პირიქით იკავებს T რესურსს და ელოდება U-ს.



ნახ. 6.1. რესურსების განაწილების გრაფი: რესურსი დაკავებულია (ა); რესურსის მოთხოვნა (ბ); ურთიერთბლოკირება (გ)

ქვემოთ ჩვენ განვიხილავთ ორ შემთხვევას, როდესაც რესურსები წარმოდგენილია მხოლოდ 1 ასლის სახით და როდესაც რესურსები წარმოდგენილია რამდენიმე ასლის სახით. თითოეული შემთხვევისასთვის განვიხილავთ შესაბამის ალგორითმს.

6.3. ურთიერთბლოკირების ალმოჩენა

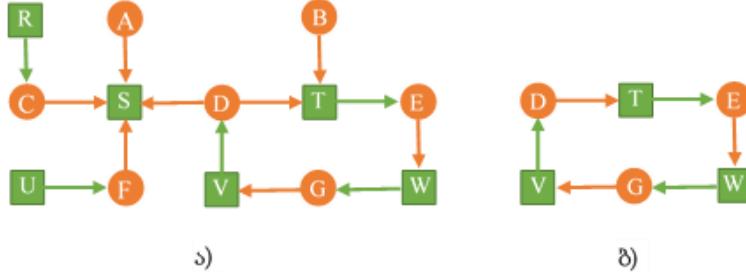
1. რესურსების ერთი ასლი. განხილვა დავიწყოთ მარტივი შემთხვევით, როდესაც სისტემაში არსებული ყოველი რესურსი წარმოდგენილია ერთი ასლის სახით. ასეთი სისტემისათვის შესაძლებელია ავაგოთ ნახ. 6.1-ზე გამოსახული გრაფის ანალოგიური გრაფი. თუ გრაფი შეიცავს ერთ ან მეტ ციკლს, მაშინ საქმე გვაქვს ურთიერთბლოკირებასთან. ციკლში მონაწილე ყოველი პროცესი ბლოკირებულია და მათი იქიდან თავის დაღწევა შეუძლებელია. თუ გრაფში არ გვაქვს ციკლი შესაბამისად არ გვაქვს ურთიერთბლოკირებაც.

განვიხილოთ მაგალითი. ვთქვათ, მოცემული გვაქვს 7 პროცესი (A - G) და 6 რესურსი (R - W). დავუშვათ რესურსებზე გაკეთებულ მოთხოვნები წარმოდგენილია შემდეგი მიმდევრობის სახით:

1. A პროცესი იკავებს R რესურსს და ითხოვს S რესურსს;
2. B პროცესი არ იკავებს არცერთ რესურსს და ითხოვს T რესურსს;
3. C პროცესი არ იკავებს არცერთ რესურსს და ითხოვს S რესურსს;
4. D პროცესი იკავებს U რესურსს და ითხოვს S და T რესურსს;
5. E პროცესი იკავებს T რესურსს და ითხოვს V რესურსს;

6. F პროცესი იკავებს W რესურსს და ითხოვს S რესურსს;
7. G პროცესი იკავებს V რესურსს და ითხოვს U რესურსს.

იმყოფება თუ არა სისტემა ურთიერთბლოკირების მდგომარეობაში და თუ პასუხი დადებითია, მაშინ რომელი რესურსები მონაწილეობენ ურთიერთბლოკირებაში?



ნახ. 6.2. რესურსების გრაფი (ა); რესურსების ციკლი (ბ)

ამ კითხვაზე პასუხს იძლევა ნახ. 6.2. ა)-ზე გამოსახული გრაფი, რომელიც აგებულია მოთხოვნების მიმდევრობის შესაბამისად. როგორც ა)-დან ვხედავთ, გრაფი შეიცავს ერთ ციკლს. ციკლში მონაწილე პროცესები (D, E, G (ნახ. ბ)) იმყოფებიან ურთიერთბლოკირების მდგომარეობაში. A, C, F პროცესები არ მონაწილეობენ ურთიერთბლოკირებაში, ვინაიდან S რესურსი შესაძლებელია (ნებისმიერი მიმდევრობით) გამოეყოს თითოეულ მათგანს, სამუშაოს დასრულების შემდეგ კი დააბრუნებს მას და სხვა პროცესებსაც შეეძლებათ ამ რესურსის გამოყენება. (შევნიშნოთ, რომ ამ რესურსის დაუფლება ასევე შეუძლია D პროცესსაც, რომელიც არ გამოანთავისუფლებს მას დასრულებამდე.)

მიუხედავად იმისა, რომ გრაფების ცხრილიდან ვიზუალურად ადვილია ურთიერთბლოკირების აღმოჩენა, რეალურ სისტემებში საჭიროა ფორმალური ალგორითმი. განვიხილოთ ბლოკირების აღმოჩენის ყველაზე მარტივი ალგორითმი, რომელიც ამოწმებს რესურსების გრაფს და ციკლის აღმოჩენის შემთხვევაში წყვეტს მუშაობას. თუ გრაფი არ შეიცავს ციკლს, მაშინ ალგორითმი სრულდება გრაფის ბოლომდე შემოწმების შემდეგ. ალგორითმში იგება ერთი დინამიური სტრუქტურა (L), რომელშიც იწერება კვანძების მიმდევრობა და ინიშნება მიმართული მონაკვეთები. ალგორითმის მუშაობის პროცესში მიმართული მონაკვეთის მონიშვნა აღნიშნავს, რომ შესაბამისი გადასვლა შემოწმებულია და ის არ საჭიროებს ხელმეორედ გადამოწმებას.

ალგორითმი შედგება შემდეგი ეტაპებისაგან:

1. გრაფზე არსებული ყოველი i-ური კვანძისათვის, რომელსაც გრაფი გამოიყენებს საწყის წერტილად, შესრულდება 2 - 6 ეტაპები;
2. ხორციელდება L ჩამონათვალის ინიციალიზირება და ყველა მიმართულ მონაკვეთს ეხსნება მონიშვნა;
3. მიმდინარე კვანძი L ჩამონათვალს ემატება ბოლოდან და მოწმდება, ხომ არ არის ის ამ ჩამონათვალში. ჩამონათვალში რომელიმე კვანძის მეორედ გამოჩენა ნიშნავს, რომ გრაფი შეიცავს ციკლს და ალგორითმი წყვეტს მუშაობას;
4. მითითებული კვანძისათვის მოწმდება ხომ არ არსებობს კვანძიდან გამომავალი სხვა მიმართული მონაკვეთი. ასეთის არსებობის შემთხვევაში გადავდივართ მე-5 ეტაპზე. წინააღმდეგ შემთხვევაში გადავდივართ მე-6 ეტაპზე;
5. მიმდინარე კვანძიდან მიმართული მონაკვეთის არჩევა ხდება ნებისმიერად.

- მიმართულ მონაკვეთს ეხსნება მონიშვნა და გადავდივართ ახალ კვანძზე ამ მიმართულებით. ალგორითმი შესრულებას აგრძელებს მე-3 ეტაპიდან;
6. თუ კვანძი წარმოადგენს საწყის წერტილს ეს ნიშნავს, რომ გრაფი არ შეიცავს ციკლს და ალგორითმი წყვეტს მუშაობას. წინააღმდეგ შემთხვევაში გვაქვს ჩიხი. მიმდინარე კვანძი იშლება და ალგორითმი უბრუნდება იმ კვანძს, რომლიდანაც მიმდინარე კვანძზე მოხდა გადასვლა. შემდეგ ალგორითმი ახალი კვანძისათვის შესრულებას იწყებს მე-3 ეტაპიდან.

ალგორითმი ყოველ კვანძს განიხილავს საწყისი წერტილის როლში, აგებს ხეს და მის შიგნით ეძებს ციკლს. თუ შესრულების მომენტში ალგორითმმა ორჯერ აღმოჩინა რომელიმე კვანძი ეს ნიშნავს, რომ ციკლი აღმოჩენილია და ალგორითმი წყვეტს შესრულებას. ყოველი კვანძისათვის ალგორითმი გადადის ყველა შესაძლო მიმართულებით, რომელსაც უთითებს მიმართული მონაკვეთი (მოთხოვნა). თუ ალგორითმს კვანძიდან რომელიმე მიმართულებით გადასავლა არ შეუძლია (გადასვლა სხვა კვანძზე არ ხდება), მაშინ ის უბრუნდება ერთი დონით მაღლა მყოფ კვანძს. თუ ალგორითმს რომელიმე კვანძთან მიმართებაში შესამოწმებელი არაფერი დარჩა ეს ნიშნავს, რომ კვანძი არ მონაწილეობს ურთიერთბლოკირებაში. თუ ყველა კვანძისათვის ეს პირობა შესრულებულია, მაშინ სისტემა არ იმყოფება ურთიერთბლოკირების მდგომარეობაში.

პრაქტიკული ხასიათის ამოცანაზე განვახორციელოთ ალგორითმის მუშაობის ილუსტრირება. განვიხილოთ ნახ. 6.2. ა)-ზე წარმოდგენილი გრაფი. ალგორითმის მიერ კვანძების განხილვის მიმდევრობა ნებისმიერია. ჩვენს შემთხვევაში კვანძები ავიღოთ შემდეგი მიმდევრობით: R, A, B, C, S, D, T, E და ა.შ. შევნიშნოთ, რომ ციკლის აღმოჩენის შემთხვევაში ალგორითმი წყვეტს მუშაობას.

პირველი ალგორითმი საწყის წერტილად იღებს R კვანძს და ხდება L სტურქტურის ინიციალიზირება ($L=[R]$). რადგანაც R-დან გადასვლა გვაქვს C-ზე, ამიტომ ის აღმოჩნდება L-ში მეორე ელემენტად ($L=[RC]$). C-დან გადასვლა ხდება S-ზე და შესაბამისად ის იქნება L-ში მესამე ელემენტი S ($L=[RCS]$). რადგანაც S-დან არსად არ ხდება გადასვლა, ამიტომ L-ის ფორმირება R კვანძისათვის დასრულებულია და შესაბამისად ვდებულობთ, რომ R ურთიერთბლოკირებაში არ მონაწილეობს. ალგორითმი მეორე საწყის წერტილად ირჩევს C კვანძს. რადგანაც C-დან გვაქვს გადასვლა მხოლოდ S-ზე და იქიდან კი არსად, ამიტომ ამ შემთხვევაში გვაქვს L=[CS], საიდანაც ვასკვნით, რომ C-ც არ მონაწილეობს ურთიერთბლოკირებაში. შემდევ ნაბიჯზე ალგორითმი საწყის წერტილად ირჩევს B-ს და ანალოგიური მსჯელობით D-მდე მისვლისას გვაქვს L=[BTEWGVD]. D-დან ორი მიმართულებითაა შესაძლებელი გადასვლა (S და T). S-ის მიმართულებით გადასვლის შემთხვევაში, რადგანაც S-დან არცერთი მიმართულებით არ ხდება გადასვლა, ამიტომ ვდებულობთ L=[BTEWGVD], საიდანაც ვასკვნით, რომ ამ მიმართულებით არ შეიძლება გვქონდეს ურთიერთბლოკირება. D-ს მიმართულებით გადასვლით კი გვაქვს L=[BTEGVDT]. რადგანაც T L-ში გამოჩნდა მეორეჯერ ეს ნიშნავს, რომ ციკლი აღმოჩენილია და ალგორითმი წყვეტს მუშაობას.

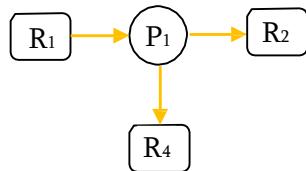
ამოცანა 1. იძლევა თუ არა პროცესების მიერ რესურსებზე გაკეთებულ მოთხოვნათა მიმდევრობა საიმედო მდგომარეობას, თუ მოთხოვნათა მიმდევრობა მოცემულია სახით:

პროცესი	იკავებს	ითხოვს
P ₁	R ₁	R ₂ , R ₄
P ₂	R ₇	R ₁ , R ₃ , R ₄
P ₃		R ₂ , R ₄ , R ₅
P ₄	R ₃	R ₆ , R ₇

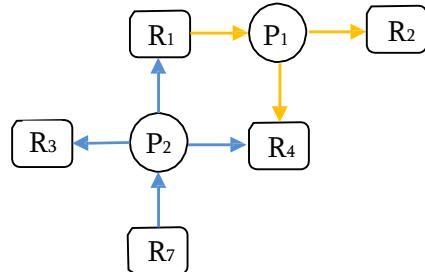
განსაზღვროთ მოთხოვნათა მოცემულ მიმდევრობას მივყავართ თუ არა არასაიმედო მდგომარეობამდე. დადებითი პასუხის შემთხვევაში განსაზღვრეთ ურთიერთბლოკირებაში მონაწილე რესურსები.

ავაგოთ მოთხოვნების შესაბამისი დიაგრამა: პირველი მოთხოვნისათვის (P₁ იკავებს R₁-ს, ითხოვს R₂ და R₄-ს)

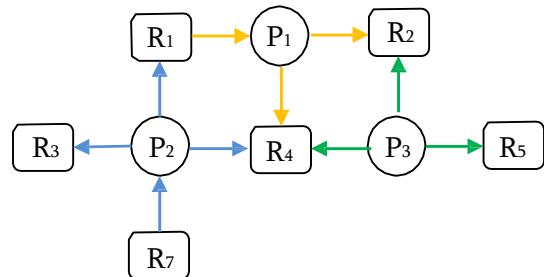
გვექნება



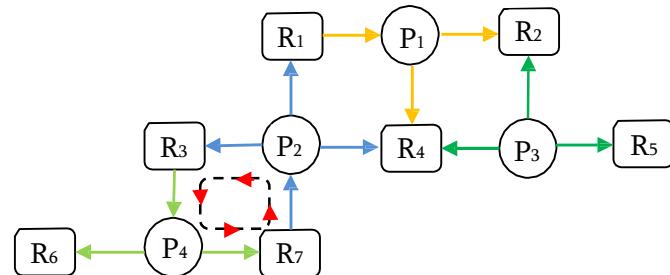
მეორე მოთხოვნისათვის (P₂ იკავებს R₇-ს, ითხოვს R₁, R₃ და R₄-ს) გვექნება



მესამე მოთხოვნისათვის (P₃ ითხოვს R₂, R₄ და R₅-ს) გვექნება



მეოთხე, ბოლო მოთხოვნისათვის (P₄ ითხოვს R₃ და R₆, R₇) გვექნება



რადგანაც დიაგრამის მიხედვით მიმართული მონაკვეთებით შეიკრა წრე (P₂->R₃->

$P_4 \rightarrow R_7$), ამიტომ მოთხოვნათა მოყვანილი მიმდევრობა გვაძლევს რესურსების გადანაწილების არასაიმედო მდგომარეობას. ურთიერთბლოკირებაში მონაწილეობს შემდეგი რესურსები: R_3 და R_7 .

მაგალითები. იძლევა თუ არა პროცესების მიერ რესურსებზე გაკეთებულ მოთხოვნათა მიმდევრობა საიმედო მდგომარეობას, თუ მოთხოვნათა მიმდევრობა მოცემულია სახით:

	პროცესი	იკავებს	ითხოვს
P ₁	R ₂	R ₁	
P ₂	R ₃	R ₁	
P ₃	R ₁	R ₂ , R ₄	
P ₄	R ₄ , R ₅		
P ₅	R ₆	R ₄ , R ₅	

	პროცესი	იკავებს	ითხოვს
P ₁		R ₁ , R ₂	
P ₂		R ₃	R ₁
P ₃			R ₂ , R ₅
P ₄			R ₃ , R ₄
P ₅		R ₅	R ₄

	პროცესი	იკავებს	ითხოვს
P ₁			R ₁ , R ₂
P ₂			R ₁ , R ₃
P ₃	R ₁		R ₂ , R ₄
P ₄	R ₃ , R ₅		R ₄
P ₅	R ₄		R ₅
P ₆			R ₅

	პროცესი	იკავებს	ითხოვს
P ₁		R ₁ , R ₂	
P ₂		R ₃	R ₁ , R ₂
P ₃		R ₅	R ₂ , R ₄
P ₄			R ₃ , R ₅
P ₅		R ₄ , R ₆	
P ₆			R ₅ , R ₆

	პროცესი	იკავებს	ითხოვს
P ₁	R ₃	R ₁	
P ₂	R ₁	R ₂ , R ₄	
P ₃	R ₂	R ₅	
P ₄	R ₅	R ₄ , R ₇	
P ₅	R ₆	R ₃	
P ₆	R ₈	R ₄ , R ₆ , R ₇	
P ₇		R ₇ , R ₈	

	პროცესი	იკავებს	ითხოვს
P ₁			R ₁ , R ₂
P ₂			R ₁ , R ₃
P ₃		R ₁ , R ₂	R ₃ , R ₄
P ₄		R ₃	R ₅
P ₅		R ₅	R ₄
P ₆		R ₄ , R ₆	
P ₇			R ₅ , R ₆

	პროცესი	იკავებს	ითხოვს
P ₁	R ₁	R ₂	
P ₂	R ₂	R ₃	
P ₃	R ₄	R ₁	
P ₄	R ₃	R ₅	
P ₅		R ₃ , R ₅ , R ₇	
P ₆	R ₆	R ₄	
P ₇	R ₅	R ₆ , R ₇	

	პროცესი	იკავებს	ითხოვს
P ₁		R ₁	R ₂
P ₂		R ₄	R ₁
P ₃		R ₂	R ₃
P ₄		R ₆	R ₄
P ₅			R ₂ , R ₄ , R ₅ , R ₇
P ₆		R ₃	R ₅
P ₇		R ₇	R ₆
P ₈		R ₅	R ₈
P ₉		R ₈	R ₇

2. რესურსების ერთი ასლი. იმ შემთხვევაში, როდესაც ოპერაციულ სისტემაში ერთიდაგივე რესურსი წარმოდგენილია რამდენიმე ასლის სახით გამოიყენება სხვა მიდგომა. ამ შემთხვევაში n პროცესს შორის (P_1, \dots, P_n) ურთიერთბლოკირების აღმოჩენის ალგორითმში გამოიყენება მატრიცები.

შემოვილოთ აღნიშვნები: ვთქვათ, m არის გამოთვლით სისტემაში განსხვავებული რესურსების რაოდენობა და რესურსები გადანომრილია 1-დან m -მდე. E_j -ით ($1 \leq j \leq m$) აღვნიშნოთ j -ური რესურსის ასლების რაოდენობა. E -თი აღვნიშნოთ გამოთვლითი სისტემის რესურსების საერთო რაოდენობა (ანუ ვექტორი $A = (A_1, \dots, A_m)$). A -თი აღვნიშნოთ დროის მიმდინარე მომენტში რესურსების თავისუფალი (ხელმისაწვდომი) ასლების რაოდენობის აღმნიშვნელი ვექტორი $A = (A_1, \dots, A_m)$, სადაც A_j -ით ($1 \leq j \leq m$) აღნიშნულია j -ური თავისუფალი რესურსის რაოდენობა. დამატებით შემოვილოთ კიდევ ორი მატრიცა R და C , სადაც R მატრიცის R_i ($1 \leq i \leq n$) სტრიქონი (ვექტორი) აღნიშნავს P_i პროცესის მიერ მოთხოვნილი რესურსების საერთო რაოდენობას, ხოლო C მატრიცის C_i ($1 \leq i \leq n$) სტრიქონი (ვექტორი) კი აღნიშნავს P_i პროცესისათვის დროის მიმდინარე მომენტში გამოყოფილი რესურსების საერთო რაოდენობას.

$$P = (P_1, \dots, P_n) \quad A = (A_1, \dots, A_m) \quad E = (E_1, \dots, E_m)$$

$$R = \begin{pmatrix} R_{11} & \cdots & R_{1m} \\ \vdots & \cdots & \vdots \\ R_{ln} & \cdots & R_{nm} \end{pmatrix} \quad C = \begin{pmatrix} C_{11} & \cdots & C_{1m} \\ \vdots & \cdots & \vdots \\ C_{ln} & \cdots & C_{nm} \end{pmatrix}$$

ცხადია, რომ რესურსებისათვის გვექნება შემდეგი დამოკიდებულება

$$E_i = A_i + \sum_{j=1}^m C_{ij} \quad (1 \leq i \leq n)$$

ნებისმიერ ორ $X = (X_1, \dots, X_m)$ და $Y = (Y_1, \dots, Y_m)$ ვექტორს შორის შედარების ოპერაცია განვსაზღვროთ შემდეგნაირად: $X \leq Y$, მაშინ და მხოლოდ მაშინ, როდესაც $X_i \leq Y_i$ ყოველი i -სთვის $1 \leq i \leq m$.

ალგორითმის მუშაობის დაწყებამდე ყველა პროცესი ცხადდება არამარკირებულად. შესრულების პროცესში მოხდება პროცესების მარკირება იმის აღსანიშნავად, რომ ისინი არ მონაწილეობენ ურთიერთბლოკირებაში და შეუძლიათ დასრულება. ალგორითმის დასრულების შემდეგ არამარკირებული პროცესის არსებობა ნიშნავს, რომ ის მონაწილეობს ურთიერთბლოკირებაში. ალგორითმის გამოყენებისას ვითვალისწინებთ ყველაზე უარეს შემთხვევას, რომლის დროსაც პროცესი იკავებს ყველა რესურსს დასრულებამდე.

ურთიერთბლოკირების ალგორითმის მუშაობა იყოფა სამ ეტაპად:

1. ხორციელდება არამარკირებული პროცესის (P_i) ძებნა, რომლისათვისაც შესრულებულია პირობა $C_i < A_i$, ანუ დასასრულებლად საჭირო რესურსების რაოდენობა არ აღემატება თავისუფალი რესურსების რაოდენობას;
2. თუ ასეთი პროცესი მოიძებნა A ვექტორით იცვლება C მატრიცის შესაბამისი სტრიქონი და ალგორითმი უბრუნდება ეტაპ 1-ს;

3. თუ ასეთი პროცესი არ არსებობს, მაშინ ალგორითმი ასრულებს მუშაობას.

საზოგადოდ, ალგორითმის მუშაობის პრინციპი მდგომარეობს შემდეგში. პირველ ეტაპზე ალგორითმი ეძებს პროცესებს, რომელთა დაკმაყოფილებაც შესაძლებელია სისტემაში არსებული თავისუფალი რესურსების ხარჯზე. შემდეგ პროცესს გამოყოფა რესურსები და ის იწყებს შესრულებას. პროცესი დასრულების შემდეგ ანთავისუფლებს მის მიერ დაკავებულ ყველა რესურსს და მათი გამოყენება შეუძლია სისტემაში არსებულ სხვა პროცესებს. ხდება პროცესის მარკირება. თუ ალგორითმის დასრულების შემდეგ სისტემაში აღმოგვაჩნდა მხოლოდ მარკირებული პროცესები ანუ ყველა პროცესს შეუძლია დასრულება, ეს ნიშნავს, რომ სისტემაში ურთიერთბლოკირების წარმოშობის საფრთხე არ არსებობს. წინააღმდეგ შემთხვევაში დროის ნებისმიერ მომენტში შესაძლებელია წარმოიშვას ურთიერთბლოკირება.

ალგორითმის მუშაობის საილუსტრაციოდ განვიხილოთ მაგალითი.

ამოცანა 2. პროცესების მიერ რესურსებზე გაკეთებული მოთხოვნის მიხედვით გაარკვიეთ არის თუ არა მოთხოვნათა მიმდევრობა საიმედო. თუ მიმდევრობა არასაიმედოა, მაშინ მიუთითეთ რომელი რესურსები მონაწილეობენ ურთიერთბლოკირებაში.

პროცესების მიერ რესურსებზე გაკეთებული მოთხოვნების საფუძველზე დიაგრამის აგების მეთოდი სამართლიანი იმ შემთხვევაში, როდესაც რესურსები სისტემაში წარმოდგენილია ერთი ეგზემპლარის სახით. თუ სისტემაში ყოველი რესურსები წარმოდგენილია რამდენიმე ეგზემპლარის სახით, მაშინ მოთხოვნათა საიმედოობის გასარკვევად საჭიროა სხვა მეთოდი ძიება.

განვიხილოთ მაგალითი. ვთქვათ, სისტემაში მოცემული გვაქვს 4 პროცესი, $P = (P_1 P_2 P_3 P_4)$ და 4 რესურსი, $E = (6, 8, 5, 8)$, მათ შორის მიმდინარე მომენტისათვის თავისუფალი რესურსები გვაქვს შემდეგი რაოდენობით $A = (2, 2, 0, 0)$. დავუშვათ პროცესებისათვის მიმდინარე მომენტში გამოყოფილი რესურსების მატრიცას და მოთხოვნილი რესურსების მატრიცას შესაბამისად აქვს სახე

$$R = \begin{pmatrix} 1 & 2 & 1 & 3 \\ 3 & 2 & 2 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 2 \end{pmatrix} \quad \text{და} \quad C = \begin{pmatrix} 1 & 6 & 2 & 1 \\ 1 & 0 & 2 & 1 \\ 2 & 2 & 0 & 0 \\ 0 & 2 & 1 & 0 \end{pmatrix}.$$

შევნიშნოთ, რომ გამოყოფილი და მოთხოვნილი რესურსების მატრიცაში სვეტი აღნიშნავს ნუმერაციის შესაბამის რესურსს და თუ რა რაოდენობითაა ის პროცესისათვის გამოყოფილი ან მის მიერ მოთხოვნილი, ხოლო სტრიქონი აღნიშნავს ნუმერაციის შესაბამის პროცესს და თუ რა რაოდენობითაა მისთვის სხვადასხვა რესურსები გამოყოფილი ან მოთხოვნილი. მაგალითად, მეორე სვეტი აღნიშნავს A_2 რესურსს, ხოლო მეოთხე სტრიქონი კი P_4 პროცესს.

თავისუფალი რესურსების მიხედვით მხოლოდ P_3 პროცესის მოთხოვნის დაკმაყოფილებაა შესაძლებელი. მისი დასრულების შემდეგ თავისუფალი რესურსების რაოდენობა იქნება $A = (2 \ 3 \ 1 \ 0)$. რესურსების ამ რაოდენობით მხოლოდ P_4 პროცესის მოთხოვნის დაკმაყოფილებას შევძლებთ, რომლის დასრულების შემდეგ თავისუფალი რესურსების რაოდენობა იქნება $A = (2 \ 4 \ 2 \ 2)$. დარჩენილი ორი პროცესიდან თავიდან შესაძლებელი იქნება P_2 პროცესის მოთხოვნის დაკმაყოფილება ხოლო შემდეგ კი P_1

პროცესის მიერ გაკეთებული მოთხოვნის დაკმაყოფილება. მაშასადამე, მოთხოვნათა მოცემული მიმდევრობა იძლევა საიმედო მდგომარეობას, ხოლო თანმიმდევრობა რომელიც გვაძლევს საიმედო მდგომარეობას არის შემდეგი: $P_3 \rightarrow P_4 \rightarrow P_2 \rightarrow P_1$.

თუ პროცესების მიერ რესურსების მოთხოვნათა მატრიცაში P_2 პროცესს მოვთხოვთ A_3 რესურსი მოითხოვოს 3 ერთეული, მაშინ მივიღებთ არასაიმედო მდგომარეობას.

მაგალითები. პროცესების მიერ რესურსებზე გაკეთებული მოთხოვნების მიხედვით განსაზღვრეთ სისტემა რჩება თუ არა საიმედო მდგომარეობაში. დადებითი პასუხის შემთხვევაში განსაზღვრეთ მოთხოვნათა დაკმაყოფილების თანმიმდევრობა.

$$\text{ს)} \quad A = \begin{pmatrix} 1 & 2 & 0 & 2 & 1 \end{pmatrix} \quad R = \begin{pmatrix} 2 & 3 & 2 & 0 & 0 \\ 0 & 1 & 2 & 2 & 1 \\ 3 & 4 & 1 & 1 & 3 \\ 0 & 2 & 2 & 1 & 0 \end{pmatrix} \quad C = \begin{pmatrix} 2 & 3 & 2 & 0 & 0 \\ 0 & 1 & 2 & 2 & 1 \\ 1 & 1 & 3 & 1 & 2 \\ 0 & 2 & 2 & 1 & 0 \end{pmatrix}$$

$$\text{ბ)} \quad A = \begin{pmatrix} 2 & 1 & 3 & 0 & 2 \end{pmatrix} \quad R = \begin{pmatrix} 1 & 0 & 2 & 0 & 1 \\ 0 & 3 & 2 & 1 & 2 \\ 1 & 2 & 1 & 2 & 0 \\ 0 & 2 & 2 & 1 & 0 \end{pmatrix} \quad C = \begin{pmatrix} 0 & 1 & 2 & 0 & 0 \\ 0 & 1 & 2 & 1 & 1 \\ 1 & 2 & 0 & 1 & 2 \\ 0 & 1 & 2 & 1 & 0 \end{pmatrix}$$

$$\text{გ)} \quad A = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \quad R = \begin{pmatrix} 2 & 2 & 0 & 0 & 2 & 2 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 2 & 1 & 1 & 0 & 0 \\ 2 & 0 & 2 & 5 & 0 & 2 \end{pmatrix} \quad C = \begin{pmatrix} 2 & 2 & 3 & 0 & 0 & 2 \\ 1 & 0 & 1 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 2 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 2 & 3 & 3 \end{pmatrix}$$

$$\text{დ)} \quad A = \begin{pmatrix} 1 & 1 & 1 & 2 & 0 & 1 \end{pmatrix} \quad R = \begin{pmatrix} 1 & 2 & 1 & 3 & 1 & 0 \\ 2 & 1 & 0 & 0 & 2 & 3 \\ 1 & 2 & 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 0 & 1 & 0 \\ 2 & 0 & 2 & 2 & 0 & 1 \end{pmatrix} \quad C = \begin{pmatrix} 2 & 2 & 3 & 0 & 3 & 3 \\ 0 & 0 & 2 & 3 & 0 & 2 \\ 5 & 4 & 2 & 0 & 2 & 1 \\ 4 & 1 & 0 & 2 & 2 & 3 \\ 1 & 0 & 1 & 2 & 0 & 0 \end{pmatrix}$$

$$\text{ე)} \quad A = \begin{pmatrix} 1 & 2 & 1 & 0 & 1 & 2 & 1 \end{pmatrix} \quad R = \begin{pmatrix} 2 & 1 & 3 & 1 & 0 & 2 & 4 \\ 2 & 2 & 0 & 2 & 3 & 0 & 0 \\ 0 & 3 & 2 & 3 & 0 & 3 & 2 \\ 2 & 0 & 1 & 0 & 2 & 1 & 0 \\ 1 & 1 & 2 & 2 & 3 & 0 & 1 \\ 0 & 1 & 2 & 1 & 0 & 2 & 0 \end{pmatrix} \quad C = \begin{pmatrix} 0 & 2 & 0 & 0 & 1 & 2 & 0 \\ 5 & 3 & 9 & 7 & 5 & 4 & 8 \\ 3 & 2 & 0 & 1 & 0 & 3 & 3 \\ 0 & 0 & 9 & 4 & 3 & 8 & 5 \\ 2 & 2 & 5 & 3 & 1 & 3 & 3 \\ 2 & 2 & 0 & 5 & 3 & 6 & 5 \end{pmatrix}$$

$$\text{ს)} \quad A = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} \quad R = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 2 & 1 & 0 \\ 1 & 1 & 2 & 0 & 0 & 1 & 2 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 2 & 0 & 1 & 1 \\ 2 & 0 & 0 & 0 & 0 & 2 & 1 \end{pmatrix} \quad C = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 2 & 0 & 1 & 0 & 2 & 2 & 0 \\ 2 & 1 & 1 & 3 & 0 & 3 & 2 \\ 4 & 2 & 2 & 0 & 2 & 3 & 2 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 3 & 1 & 0 & 2 \end{pmatrix}$$

საკითხები:

თეორიული

1. ოპერაციულ სისტემასთან დაკავშირებული ძირითადი ცნებები;
2. ოპერაციული სისტემების სტრუქტურა (არქიტექტურა);
3. პროცესის მოდელი. პროცესის წარმოქმნა, დასრულება, მდგომარეობა, ცხრილი. მართვის ბლოკი და კონტექსტი. კონტექსტის გადართვა;
4. პროცესების დაგეგმვა. დაგეგმვის ალგორითმები (FCFS, SJF, RR, პრიორიტეტული დაგეგმვა, მრავალდონიანი დაგეგმვა);
5. მონაცემების გაცვლის საშუალებები. კომუნიკაციის საშუალებების ლოგიკური ორგანიზაცია;
6. thread-ების რეალიზაცია;
7. შეჯიბრის მდგომარეობა. კრიტიკული სექცია. ურთიერთგამორიცხვა აქტიური ლოდინით (წყვეტის აკრძალვა, ცვლადი-ბოქლომი, მვაცრი მიმდევრობა, აქტივაცია და დეაქტივაცია);
8. ურთიერთბლოკირება. განაწილებადი და გაუნაწილებელი რესურსი;
9. ურთიერთბლოკირების წარმოქმნის აუცილებელი და საკმარისი პირობები;

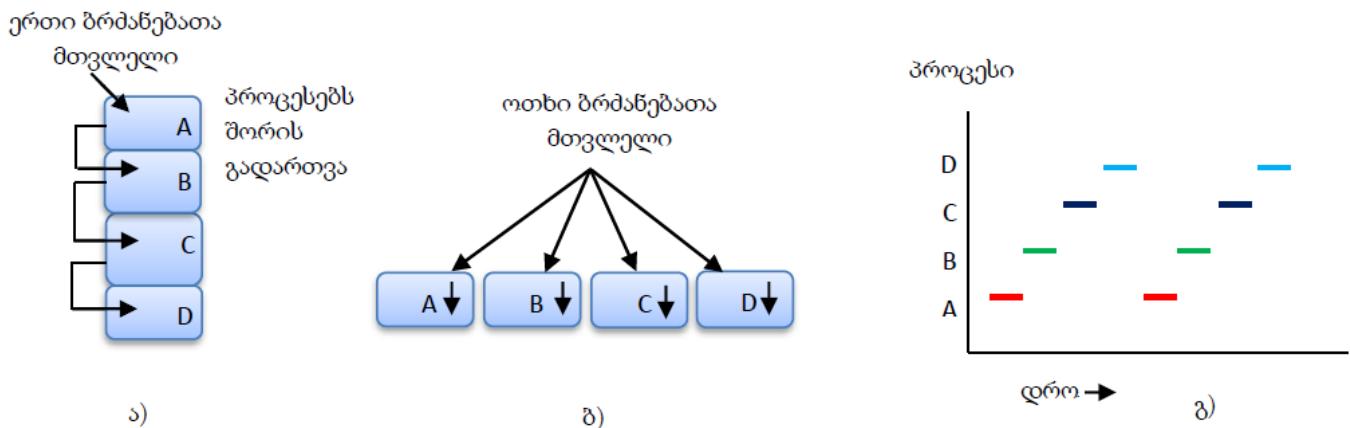
პრაქტიკული

1. სემინარი 1 სრულად;
2. პროცესის იდენტიფიკაცია, პროცესის სიცოცხლის ციკლი, პროცესის წარმოქმნა, პროცესის დასრულება, პროცესის მომხმარებლის კონტექსტის შეცვლა.
3. ფაილური დესკრიპტორი, სისტემური გამოძახება open(), read(), write(), close(). pipe და FIFO.

პროცესის მოდელი

პროცესის მოდელის მიხედვით შესრულებადი პროგრამული უზრუნველყოფა, რიგ შემთხვევაში ოპერაციული სისტემის ჩათვლით, დაიყვანება პროცესების მიმდევრობაზე. პროცესი ეს არის შესრულებადი პროგრამა მასთან ასოცირებული ბრძანებათა მთვლელის მიმდინარე მდგომარეობით, რეგისტრებითა და ცვლადებით. კონცეპტუალურად ყოველ პროცესს გააჩნია საკუთარი ვირტუალური პროცესორი. ცხადია, რომ ცენტრალური პროცესორი მუდმივად ახდენს პროცესებს შორის გადართვას, მაგრამ პროცესების გაგების სიმარტივისათვის უმჯობესია ვიფიქროთ პარალელურად შესრულებად პროცესების ნაკრებზე.

ნახ. 2.1 ა)-ზე ნაჩვენებია მარავალამოცანიანი კომპიუტერის მეხსიერებაში ჩატვირთული 4 პროცესი. ბ)-ზე ნაჩვენებია 4 პროცესი მართვის საკუთარი ალგორითმით (საკუთარი ბრძანებათა მთვლელით) და თითოეული პროცესი სრულდება ერთმანეთისაგან დამოუკიდებლად. ყოველ კომპიუტერში არსებობს ერთი ფიზიკური ბრძანებათა მთვლელი, ამიტომ პროცესის შესრულებისას მისი ბრძანებათა ლოგიკური მთვლელი იტვირთება რეალურ მთვლელში. როდესაც პროცესის შესრულება ჩერდება გარკვეული დროით ბრძანებათა მთვლელის მნიშვნელობა ინახება პროცესის ბრძანებათა ლოგიკურ მთვლელში. გ)-ზე ნაჩვენებია გარკვეული დროითი შუალედით დაკვირვების შედეგად მიღებული პროცესების შესრულების სურათი.



ნახ. 2.1. მარავალამოცანიან რეჟიმში მომუშავე 4 პროგრამა (ა)). ერთმანეთისაგან დამოუკიდებელი ფსევდოპარალელური პროცესი (ბ)). დროის ნებისმიერ მომენტში აქტიურია მხოლოდ ერთი პროგრამა (გ))

ვინაიდან ცენტრალური პროცესორი ახდენს პროცესებს შორის გადართვას, ამიტომ სიჩქარე, რომლითაც პროცესი ანხორციელებს საკუთარ გამოთვლებს არ იქნება მუდმივი და შესაბამისად, პროცესების პროგრამირებაც არ უნდა ხორციელდებოდეს შესრულების დროზე მკაცრად განსაზღვრული პირობებით. უნდა აღინიშნოს, რომ სისტემაში ორჯერ ამუშავებული ერთიდაიგივე პროგრამა ქმნის ორ პროცესს. მაგალითად ერთდროულად შესაძლებელია ტექსტური რედაქტორის ორჯერ ამუშავება ან ერთიდაიგივე ფაილის გადაგზავნა პრინტერზე დასაბეჭდად. ოპერაციულ სისტემაში მსგავს შემთხვევაში იტვირთება ძირითად პროგრამული კოდის მხოლოდ ერთი ასლი, რომლის გამოყენებაც შეუძლია ორივე პროცესს.

პროცესის წარმოქმნა

ოპერაციული სისტემები პროცესების შესაქმნელად საჭიროებენ გარკვეულ მექანიზმებს. ყველაზე მარტივ სისტემებში, რომლებიც შექმნილია ერთი კონკრეტული პროგრამის ასამუშავებლად, ჩატვირთვის პროცესში სისტემის ნორმალური ფუნქციონირებისათვის შესაძლებელია ერთდროულად ამუშავდეს სისტემაში არსებული ყველა პროცესი. უნივერსალურ სისტემებში საჭიროებისამებრ დამატებით უნდა არსებობდეს პროცესების წარმოქმნის გარკვეული მექანიზმები.

პროცესის წარმოქმნა შესაძლებელია გამოწვეული იყოს რამდენიმე მიზეზებით:

1. სისტემის ინიციალიზაცია;
2. შესრულებადი პროცესის მიერ ახალი პროცესის წარმოსაქმნელი სისტემურიგამოძახების შესრულება;
3. მომხმარებლის მოთხოვნა ახალი პროცესის წარმოქმნაზე;
4. პაკეტური ამოცანის ინიციალიზაცია.

ოპერაციული სისტემის ჩატვირთვის მომენტში მასში წარმოიქმნება რამდენიმეპროცესი. ზოგიერთ მათგანს გააჩნია მაღალი პრიორიტეტი, ანუ ურთიერთქმედებს მომხმარებელთან და მისთვის ასრულებს გარკვეულ სამუშაოს. დანარჩენები წარმოადგენენ ფონურ რეჟიმში მომუშავე პროცესებს, არ არიან დაკავშირებული კონკრეტულ მომხმარებელთან, მაგრამ ასრულებენ სპეციფიურ სამუშაოს. ფონური პროცესის მაგალითს წარმოადგენს პროცესი, რომელიც ამოწმებს ელექტრონულ ფოსტას ახალი წერილის შემოსვლაზე. ფონურ რეჟიმში მომუშავე პროცესებს, რომლებიც წარმოქმნილია გარკვეული აქტიური საქმიანობის წარმოებისათვის, მაგალითად, როგორიცაა, ელექტრონული ფოსტის შემოწმება, პრინტერზე ფაილის ბეჭდვა და ა.შ., დემონები ეწოდებათ. პროცესების წარმოქმნა სისტემაში გარდა მისი ჩატვირთვის მომენტისა შესაძლებელია მოხდეს დროის ნებისმიერ მომენტში. ხშირადაა სიტუაცია, როდესაც პროცესები სისტემური გამოძახების მეშვეობით წარმოქმნიან ახალ პროცესებს. ახალი პროცესის წარმოქმნა განსაკუთრებით სასარგებლოა იმ შემთხვევაში, როდესაც გამოსავალი ამოცანა შესაძლებელია დაიყოს „რამდენიმე ურთიერთდამოკიდებული“ ცნებით დაკავშირებულ, მაგრამ სხვა მხრივ დამოუკიდებელ პროცესებად. მაგალითად, შესასრულებელი ამოცანის გადაწყვეტის დაჩქარება მრავალპროცესორულ (მრავალბირთვიან) სისტემაში შესაძლებელია თითოეულ პროცესორზე ცალკეული პროცესების შესრულებით.

ინტერაქტიულ სისტემებში მომხმარებელს ახალი პროგრამა შეუძლია აამუშოს ბრძანებათა ინტერაქტურის ველში ბრძანების შეყვანით ან პროგრამისათვის განკუთვნილ შესაბამის გრაფიკულ გამოსახულებაზე მაუსის ორჯერადი დაწკაპუნებით. ამ მოქმედებებიდან თითოეულს მივყართ სისტემაში ახალი პროცესის წარმოქმნამდე.

პროცესის წარმოქმნის მეოთხე მექანიზმი გამოიყენება პაკეტური დამუშავების სისტემებში. როდესაც გამოთვლითი სისტემა ჩათვლის, რომ მასში გარკვეული გამოთვლების დასრულების შემდეგ გამონთავისუფლებული რესურსები საკმარისია ახალი ამოცანის შესასრულებლად ის წარმოქმნის ახალ პროცესს და ტვირთავს ამოცანას შესასრულებლად.

თითოეულ განხილულ შემთხვევაში ახალი პროცესი წარმოიქმნება უკვე არსებული პროცესის ბაზაზე, რომელიც ასრულებს ახალი პროცესის წარმოსაქმნელ სისტემურ გამოძახებას. ეს პროცესები შეიძლება იყვნენ მომხმარებლის მუშა პროცესები, მაუსის ან კლავიატურის მოქმედებით გამოწვეული სისტემური პროცესები, ან პაკეტური ამოცანების მართვის პროცესები.

UNIX-ში ახალი პროცესის წარმოსაქმნელად არსებობს ერთადერთი სისტემური გამოძახება fork. ეს გამოძახება წამოქმნის ახალ პროცესს, რომელიც წარმოადგენს წარმომქმნელი პროცესის სრულ ასლს. fork სისტემური გამოძახების შესრულების შემდეგ ორი პროცესი, მშობელი (წარმომქმნელი) და შვილი (წარმოქმნილი), სარგებლობს მეხსიერების ერთიდაიმავე სივრცით, ერთიდაიმავე რესურსებით, ერთიდაიმავე პროგრამული კოდით და გახსნილი ფაილებით. ამის შემდეგ შვილი პროცესი execve() ან მისი მსგავსი სხვა რომელიმე სისტემური გამოძახების შესრულებით ცვლის საკუთარ მონაცემებს, მეხსიერების სივრცეს, ხდება რესურსების გადანაწილება. ამის შემდეგაც მშობელი და შვილი პროცესები წარმოადგენს სისტემაში არსებულ დამოუკიდებელ პროცესებს.

Windows-ის სისტემაში ყველაფერი სხვაგვარადაა ორგანიზებული: Win32-ის ფუნქციის CreateProcess ერთი გამოძახებით წარმოიქმნება ახალი პროცესი და მასში იტვირთება საჭირო პროგრამა. ამ გამოძახებას გააჩნია 10 პარამეტრი: შესასრულებელი პროგრამა, ამ პროგრამისათვის ბრძანებათა ინტერპრეტატორის პარამეტრები, უსაფრთხოების სხვადასხვა პარამეტრები, ინფორმაცია პრიორიტეტზე და ა.შ. Win32-ის CreateProcess ფუნქციისათვის პროცესების სამართავად და მათი სინქრონიზაციისათვის დამატებით არსებობს 100-მდე სხვა ფუნქცია.

ორივე, UNIX და Windows, სისტემაში ახალი პროცესის წარმოქმნის შემდეგ მშობელ და შვილ პროცესებს გააჩნიათ საკუთარი მისამართების სივრცე. ნებისმიერი მათგანის მიერ საკუთარ მისამართების სივრცეში განხორციელებული ცვლილება მიუწვდომელია მეორისათვის.

პროცესის დასრულება

სისტემაში პროცესის წარმოქმნის შემდეგ ის იწყებს დასახული ამოცანის შესრულებას. ამოცანის შესრულების შემდეგ პროცესი წყვეტს არსებობას. პროცესის დასრულება შესაძლებელია გამოწვეული იყოს ერერთით შემდეგი მიზეზებიდან:

- 1.ნორმალური დასრულება;
- 2.შეცდომით გამოწვეული დასრულება;
- 3.ფატალური შეცდომით გამოწვეული დასრულება (იძულებითი);
- 4.სხვა პროცესის მიერ შესრულებული სისტემური გამოძახებით გამოწვეული დასრულება (იძულებითი).

დასახული ამოცანის შესრულების შემდეგ პროცესი წყვეტს არსებობას (დასრულდა). როდესაც კომპილიატორი დააკომპილირებს მისთვის გადაცემულ პროგრამულ ფაილს, ის ასრულებს სისტემურ გამოძახებას რითაც ოპერაციულ სისტემას აცნობებს საკუთარი მუშაობის დასრულებაზე. ასეთი სისტემური გამოძახება UNIX-ში არის exit, ხოლო Windows-ში კი - ExitProcess.

პროცესის დასრულების მეორე და მესამე მიზეზი გამოწვეულია სისტემის მიერ პროგრამის შესრულებისას აღმოჩენილი შეცდომით. მეორე შემთხვევაში შეცდომა შესაძლებელია გამოწვეული იყოს პროგრამული ფაილის ასამუშავებლას ბრძანებათა კომბინაციის არასწორად აკრეფით. მაგალითად, gcc კომპილიატორით test.c ფაილის კომპილაციისას არასწორად აკრეფილი ბრძანებით cc test.c. მესამე შეცდომა შესაძლებელია გამოწვეული იყოს უშუალოდ პროგრამულ ფრაგმენტში დაშვებული შეცდომით. მაგალითად, მეხსიერების არარსებულ უჯრედზე მიმართვა ან ნულზე გაყოფა. ზოგიერთ სისტემაში (მაგალითად, UNIX) პროცესს შეუძლია გარკვეული ტიპის შეცდომების დამუშავება. თუ პროცესმა დააფიქსირა ერთერთი ასეთი შეცდომა, მაშინ ის იღებს შესაბამის სიგნალს (ხდება წყვეტა), მაგრამ არ ხდება მისი დასრულება.

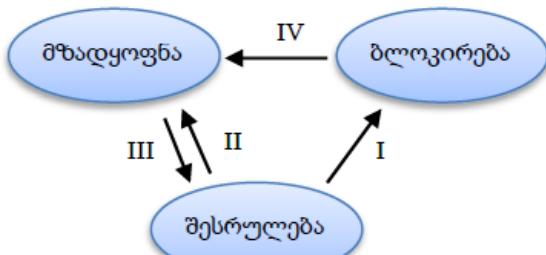
მეოთხე მიზეზი, რომლის გამოც შესაძლებელია დასრულდეს პროცესი ეს არის პროცესის მიერ ისეთი სისტემური გამოძახების შესრულება, რომელიც პროცესს აიძულებს დასრულებას. UNIX

სისტემაში ასეთი გამოძახება kill, Windows-ის სისტემაში კი - TerminateProcess. ორივე შემთხვევაში ამ გამოძახების შემსრულებელ პროცესს უნდა გააჩნდეს შესაბამისი უფლება.

პროცესების მდგომარეობა

ოპერაციული სისტემა ვალდებულია სისტემაში წარმოქმნილ ყოველ პროცესს გამოუყოს პროცესორი იმ დროით რა დროც საჭიროა პროცესის შესასრულებლად. რეალურად სისტემაში პარალელურად შეიძლება მუშაობდეს იმდენი პროცესი რამდენიც პროცესორია (ბირთვი). ცხადია, რომ პროცესების რაოდენობა ყოველთვის აჭარბებს სისტემაში პროცესორების რაოდენობას.

პროცესის ცხოვრების ციკლი შედგება მიმდევრობითი დისკრეტული მდგომარეობები- საგან. პროცესის ერთი მდგომარეობიდან მეორე მდგომარეობაში გადასვლა შესაძლებელია გამოწვეული იყოს სხვადასხვა მიზეზით. ამბობენ, რომ პროცესი სრულდება, ანუ იმყოფება მდგომარეობაში "შესრულება", თუ მისთვის გამოყოფილია ცენტრალური პროცესორი. პროცესი იმყოფება მდგომარეობაში "მზადყოფნა", თუ მისთვის ცენტრალური პროცესორის გამოყოფის შემთხვევაში პროცესს შეუძლია გააგრძელოს შესრულება. პროცესი იმყოფება მდგომარეობაში "ბლოკირება", თუ მას არ შეუძლია შესრულების გაგრძელება გარკვეული მოვლენის დადგომამდე (მაგალითად, შეტანა/გამოტანის ოპერაციის დასრულებამდე, ან სხვა პროცესის მიერ ჩასატარებელი გამოთვლების დასრულებამდე, რომელსაც ის საჭიროებს).



- I. პროცესი ბლოკირებულია, ელოდება შეტანას;
- II. შესასრულებლად არჩეულია სხვა პროცესი;
- III. შესასრულებლად არჩეულია მიმდინარე პროცესი;
- IV. შესრულების გასაგრძელებლად საჭირო მონაცემები ხელმისაწვდომია

ნახ. 2.2. პროცესი შესაძლებელია იმყოფებოდეს მდგომარეობაში შესრულება, მზადყოფნა და ბლოკირება. ისრებით წაჩვენებია ამ მდგომარეობებს შორის გადასვლა

როგორც ნახ. 2.2-დან ჩანს, ამ მდგომარეობებს შორის შესაძლებელია 4 გადასვლა. გადასვლა I ხდება იმ შემთხვევაში, როდესაც ოპერაციული სისტემა გადაწყვეტს, რომ პროცესს მიმდინარე მომენტში არ შეუძლია შესრულების გაგრძელება. გადასვლა II და III გამოწვეულია პროცესის დამგეგმვის მიერ მისი ინფორმირების გარეშე, რომელიც წარმოადგენს ოპერაციული სისტემის ნაწილს. კერძოდ, გადასვლა II ხორციელდება, როდესაც ოპერაციული სისტემა გადაწყვეტს, რომ პროცესმა საკმაო დრო დაჰყო პროცესორთან (ამოიწურა დროითი კვანტი) და უკვე საჭიროა პროცესორის გამოყოფა სხვა პროცესისათვის.

გადასვლა IV ხორციელდება, როდესაც ოპერაციულ სისტემაში დაფიქსირდება გარკვეული მოვლენა (დასრულდა შეტანა/გამოტანის ოპერაცია, ან გარკვეული რესურსი ხელმისაწვდომია). ამ შემთხვევაში ოპერაციული სისტემა ამოწმებს ხომ არ იმყოფებოდა რომელიმე ბლოკირებული პროცესი მსგავსი მოვლენის ლოდინის მდგომარეობაში. თუ ასეთი პროცესი აღმოჩნდა დამგეგმავს გადაყავს ის მზადყოფნის მდგომარეობაში.

პროცესების ცხრილი. პროცესების მართვის ბლოკი და პროცესის კონტექსტი

პროცესის მოდელის რეალიზაციისათვის ოპერაციულ სისტემაში არსებობს ე.წ. პროცესების ცხრილი, რომელშიც არსებული ყოველი ჩანაწერი შეესაბამება სისტემაში მიმდინარე მომენტში არსებულ რომელიმე პროცესს. ეს ჩანაწერი შეიცავს მოცემული პროცესისათვის სპეციფიურ ინფორმაციას:

- მპროცესის მიმდინარე მდგომარეობა;
- მპროცესორის რეგისტრების შემცველობა;
- მპროცესორის გამოყენების დაგეგმვისათვის და მეხსიერების მართვისათვისაუცილებელი მონაცემები (პროცესის პრიორიტეტი, მისამართების სივრცის ზომა დაადგილმდებარეობა და ა.შ.);
- მაღრიცხვის მონაცემები (პროცესის საიდენტიფიკაციო ნომერი, რომელიმომხმარებელის მიერაა ინიცირებული მისი მუშაობა, მიმდინარე პროცესის მიერპროცესორის გამოყენების საერთო დრო და ა.შ.);
- მინფორმაცია პროცესთან დაკავშირებული შეტანა/გამოტნის მოწყობილობებზე(რომელი მოწყობილობაა მიბმული პროცესთან, გახსნილი ფაილების ჩამონათვალი)და სხვა რესურსებზე.

შევნიშნოთ, რომ მონაცემთა სტრუქტურის შემადგენლობა და აგებულება დამოკი-დებულია კონკრეტულ ოპერაციულ სისტემაზე. ზოგიერთ ოპერაციულ სისტემაში პროცესის მახასიათებელი ინფორმაცია ინახება არა ერთ არამედ ერთმანეთთან დაკავშირებულ მონაცემთა რამდენიმე სტრუქტურაში. ამ სტრუქტურებს შეიძლება გააჩნდეთ სხვადასხვა სახელები, შეიცავდნენ სხვადასხვა დამატებით ინფორმაციას ან, პირიქით, შეიცავდნენ ზემოთ ჩამოთვლილი ინფორმაციის მხოლოდ ნაწილს. ჩვენთვის მნიშვნელოვანია არა ის, თუ რამდენად სრულია ინფორმაცია ოპერაციულ სისტემაში მიმდინარე პროცესებზე, არამედ ის თუ რამდენად იძლევა ეს ინფორმაცია ოპერაციული სისტემის მიერ პროცესებზე შესასრულებელი აუცილებელი მოქმედებების განხორციელების შესაძლებლობას. გადმოცემის სიმარტივისათვის ჩავთვალოთ, რომ ეს ინფორმაცია ინახება მონაცემთა ერთ სტრუქტურაში და მას ვუწოდოთ PCB (Process Control Block - პროცესის მართვის ბლოკი). ოპერაციული სისტემისთვის პროცესის მართვის ბლოკი წარმოადგენს პროცესის მოდელს. ოპერაციული სისტემის მიერ პროცესზე განსახორციელებელი ნებისმიერი ოპერაცია მის PCB-ში იწვევს გარკვეულ ცვლილებებს. მიღებული მოდელის ფარგლებში პროცესების მდგომარეობის PCB -ს შემცველობა ოპერაციებს შორის რჩება უცვლელი.

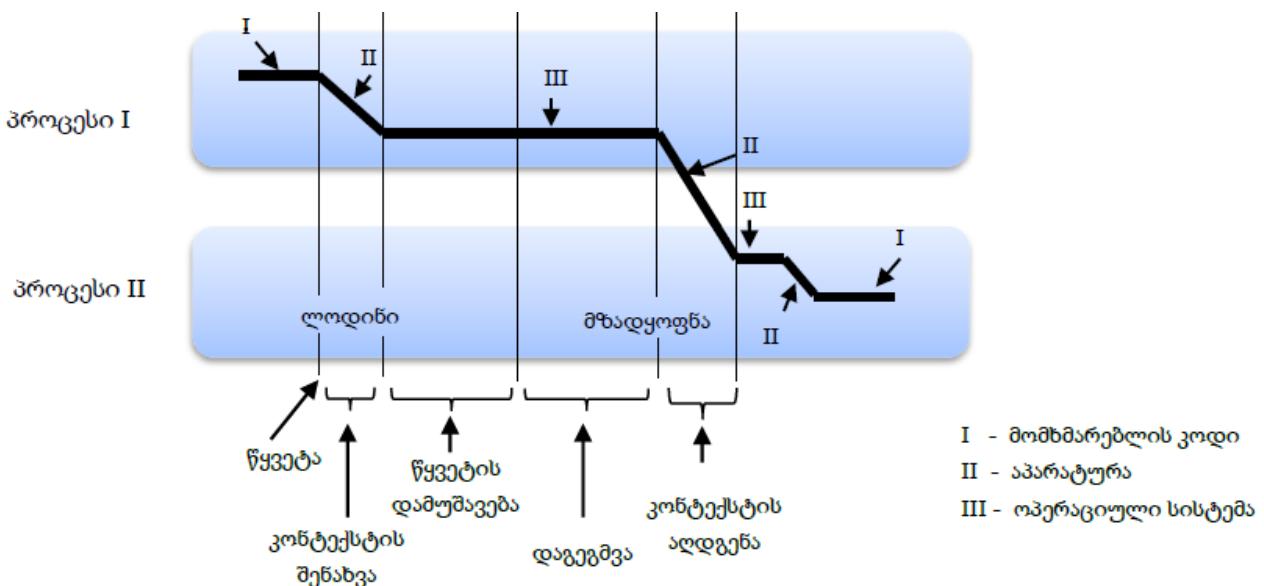
ინფორმაცია, რომლის შესანახადაცაა განკუთვნილი პროცესის მართვის ბლოკი, შედგომი გადმოცემისათვის სასარგებლოა გავყოთ ორ ნაწილად: პროცესორის ყველა რეგისტრის შემცველობას (პროგრამული მთვლელის მნიშვნელობის ჩათვლით) ვუწოდოთ პროცესის რეგისტრული კონტექსტი, ხოლო სხვა დანარჩენს კი - პროცესის სისტემური კონტექსტი. პროცესის რეგისტრული და სისტემური კონტექსტების ცოდნა საკმარისია პროცესზე ოპერაციების შესრულებისას მისი მუშაობის სამართავად, მაგრამ ის არაა საკმარისი პროცესის სრულად დასახასიათებლად. ოპერაციული სისტემის დაინტერესებაში არ შედის, თუ როგორი გამოთვლებითაა დაკავებული პროცესი, ანუ როგორი კოდი და მონაცემებია ჩატვირთული მის მისამართების სივრცეში. მომხმარებლის თვალსაზრისით კი პირიქით, მისთვის უდიდეს ინტერესს წარმოადგენს პროცესის მისამართების სივრცის შემცველობა რეგისტრულ კონტექსტთან ერთად, რომელიც განსაზღვრავს მონაცემთა გარდაქმნის და მიღებული შედეგების მიმდევრობას. პროცესის მისამართების სივრცეში ჩატვირთულ კოდს და მონაცემებს ვუწოდოთ მისი მომხმარებლის კონტექსტი. პროცესის რეგისტრული, სისტემური და მომხმარებლის

კონტექსტის ერთობლიობას სიმოკლისათვის უწოდებენ პროცესის კონტექსტს. დროის ნებიმიერ მომენტში პროცესის სრულად დახასიათება შესაძლებელია მისი კონტექსტის საშუალებით.

კონტექსტის გადართვა

მულტიპროცესული ოპერაციული სისტემის საქმიანობა შედგება ოპერაციათა ჯაჭვისაგან, რომლებიც სრულდება სხვადასხვა პროცესებზე და თანდართულია პროცესორის გადართვით ერთი პროცესიდან მეორეზე.

ნახ.2.5-ზე ნაჩვენებია კონტექსტის გადართვის პროცესი. პროცესი 1-ის შესრულებისას დროითი კვანტის ამოწურვის გამო ხდება წყვეტა. სისტემაში წყვეტის დაფიქსირებისას პროცესორს ეგზავნება წყვეტის შესაბამისი ტიპის აღმნიშვნელი სიგნალი. პროცესორი ამუშავებს სიგნალს და აჩერებს პროცესის შესრულებას. ინახავს მასთან ასოცირებულ ყველა ინფორმაციას (პროგრამული კოდი, ღია ფაილები, რესურსები და ა.შ.). დამგეგმავი შესასრულებლად ირჩევს ახალ პროცესს და იტვირთება ამ პროცესთან ასოცირებული მონაცემები. მონაცემების ჩატვირთვის შემდეგ პროცესორი იწყებს ახალი პროცესის შესრულებას. პროცესორის ერთი პროცესიდან მეორეზე კორექტულად გადასართველად აუცილებელია შესრულებადი პროცესის კონტექსტის შენახვა და იმ პროცესის კონტექსტის აღდგენა, რომელზეც იქნება პროცესორი გადართული. პროცესის ქმედუნარიანობის შენახვა/აღდგენის ასეთ პროცედურას კონტექსტის გადართვა ეწოდება.



ნახ. 2.5. კონტექსტის გადართვა.

კონტექსტის გადართვაზე დახარჯული დრო გამოთვლითი სისტემის მიერ არ გამოიყენება სასარგებლო სამუშაოს შესასრულებლად და წარმოადგენს ზედნადებ დანახარჯს, რომელიც ამცირებს სისტემის წარმადობას. ის იცვლება მანქანიდან მანქანამდე და ჩვეულებრივ მერყეობს დიაპაზონში 1-1000 მწ. თანამედროვე ოპერაციულ სისტემებში ზედნადები დანახარჯების არსებითი შემცირების შესაძლებლობას იძლევა პროცესების განზოგადოებული მოდელი, რომელიც შეიცავს ცნებას threads of execution (შესრულების ნაკადი).

პროცესების დაგეგმვა

მრავალამოცანიან რეჟიმში გამოთვლით სისტემას უწევს ერთდროულად რამდენიმე ამოცანის (პროცესის) შესრულება, რომლებიც დასახული ამოცანის შესასარულებლად იბრძიან პროცესორის მოპოვებაზე. თუ გამოთვლითი მანქანა ერთპროცესორულია, მაშინ პროცესორი დროის ნებისმიერ მომენტში შეიძლება გამოეყოს მხოლოდ ერთ პროცესს. მრავალპროცესორულ სისტემებში ერთდროულად შესაძლებელია იმდენი პროცესის ამუშავება რამდენიც პროცესორია, მაგრამ თითოეული პროცესორი მხოლოდ ერთ პროცესს შეიძლება გამოეყოს. შემდგომი გადმოცემის მიზნით დავუშვათ, რომ გვაქვს ერთპროცესორული სისტემა. იმ შემთხვევაში, როდესაც მდგომარეობაში მზადყოფნა იმყოფება რამდენიმე პროცესი საჭიროა ამ პროცესებს შორის ერთერთის არჩევა და მისთვის პროცესორის გამოყოფა. ოპერაციული სისტემის იმ ნაწილს, რომელიც განსაზღვრავს თუ რომელ პროცესს უნდა გამოეყოს პროცესორი დამგეგმავი ეწოდება, ხოლო მის მიერ გამოყენებულ ალგორითმს კი - დაგეგმვის ალგორითმი.

დაგეგმვა

პაკეტური დამუშავების სისტემებში, სადაც ამოცანების შეტანა ხორციელდებოდა მაგნიტურ ლენტაზე გადატანილი მონაცემების სახით, დაგეგმვა არ წარმოადგენდა სირთულეს: საჭირო იყო ლენტაზე არსებული რიგით შემდეგი ამოცანის შესრულება. მრავალამოცანიანი სისტემების გამოჩენასთან ერთად გართულდა დაგეგმვის ალგორითმები, ვინაიდან ამ შემთხვევაში შესაძლებელი გახდა რამდენიმე ამოცანის (ფსევდო) პარალელური შესრულება. ზოგიერთ უნივერსალურ მანქანაში პაკეტურ და დროის გაყოფის ამოცანას შორის არ არსებობს განსხვავება და დამგეგმავს უწევს შემდეგი შესასრულებელი ამოცანის არჩევა: უზრუნველყოს პაკეტური ამოცანის შესრულება ან შესასრულოს ტერმინალთან მყოფი მომხმარებლის მიმართვა (ბრძანება). ვინაიდან ასეთ სისტემებში პროცესორული დრო დეფიციტურია, კარგმა დამგეგმავმა შესაძლებელია არსებითი ზეგავლენა იქნის გამოთვლითი მანქანის წარმადობაზე.

პერსონალური კომპიუტერების გამოჩენით სიტუაცია შეიცვალა ორი მიმართულებით. პირველი შემთხვევაში, პროცესორული დრო სრულად ეთმობოდა ერთ აქტიურ პროცესს. ამ შემთხვევაში მომხმარებელს, რომელიც, მაგალითად, მუშაობდა ტექსტურ რედაქტორთან, არ შეეძლო ფონურ რეჟიმში განეხორციელებინა პროგრამული ფაილის კომპილაცია.

მეორე შემთხვევაში, კომპიუტერები დღითიდღე იძენენ ახალახალ შესაძლებლობებს და მუშაობენ საკმაოდ სწრაფად და, ამიტომ, პროცესორული დრო პრაქტიკულად არ წარმოადგენს დეფიციტს. პერსონალური კომპიუტერის პროგრამების უმეტესი შეზღუდულია მომხმარებლის მხრიდან მონაცემების (მაგალითად, ტექსტის აკრეფა) შეტანის სისწრაფეში, მაგრამ არა პროცესორის მეირ შეტანილი მონაცემების დამუშავების სისწრაფეში. პროგრამის კომპილაციის ამოცანაც, რომელიც პირველ გამოთვლით მანქანებზე წარმოადგენდა პროცესორული დროის მთავარ მომხმარებელს, უმეტეს შემთხვევაში საჭიროებს რამდენიმე წამს. ამიტომ მარტივ პერსონალურ კომპიუტერებზე დაგეგმვა არ თამაშობს მნიშვნელოვან როლს. რათქმაუნდა არსებობს პროგრამები, რომლებიც სრულად ტვირთავენ კომპიუტერის რესურსებს, მაგრამ ისინი წარმოადგენენ გამონაკლისს.

ქსელურ სამსახურებთან მიმართებით სიტუაცია მკვეთრად იცვლება. აქ უკვე პროცესორული დროის მოპოვებაზე ერთდროულად იბრძვის რამდენიმე მოთხოვნა (პროცესი) და შესაბამისად დაგეგმვაც იძენს მნიშვნელობას. მაგალითად, როდესაც ცენტრალური პროცესორს უწევს ორ პროცესს შორის არჩევანის გაკეთება (მაგალითად, ერთი, რომელიც აგროვებს ყოველდღიურ სტატისტიკურ მონაცემებს და მეორე, რომელიც ემსახურება მომხმარებლის მოთხოვნას), მაშინ უმჯობესია პროცესებისათვის თავიდანვე განსაზღვრული იყოს მათი მნიშვნელოვნობის საკითხი (პრიორიტეტი).

ვინაიდან პროცესების გადართვა წარმოადგენს ძვირადღირებულ საქმიანობას, პროცესების „სწორად“ არჩევის გარდა დამგეგმავმა ასევე უნდა იზრუნოს პროცესორის ეფექტურ გამოყენებაზე. როგორც უკვე აღვნიშნეთ, პროცესის გადართვისას საჭიროა მომხმარებლის რეჟიმიდან ბირთვის რეჟიმში გადასვლა, პროცესების ცხრილში მიმდინარე პროცესთან დაკავშირებული მონაცემების შენახვა, რეგისტრების მნიშვნელობების ჩათვლით. უმეტეს სისტემაში საჭიროა შეინახოს მეხსიერების რუკა (მეხსიერების უჯრედებზე მიმართვა). ამის შემდეგ დამგეგმავი შესასრულებლად ირჩევს ახალ პროცესს, ტვირთავს მისი მეხსიერების რუკას, რეგისტრებს და სხვა საჭირო მონაცემებს, რის შემდეგაც ახალი პროცესი იწყებს შესრულებას. ამიტომ პროცესორის ხშირმა გადართვამ შესაძლებელია მოითხოვოს პროცესორული დროის მნიშვნელოვანი ნაწილი.

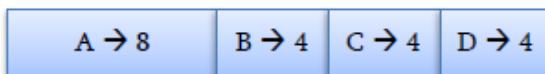
დაგეგმვის ალგორითმები

არსებობს დაგეგმვის ალგორითმების მრავალფეროვნება, რომლებიც გამოიყენება სხვადასხვა ტიპის ამოცანების გადასაწყვეტად და ეფექტურობის ხარისხის მისაღწევად. მრავალი მათგანი შესაძლებელია გამოყენებული იქნას დაგეგმვის სხვადასხვა დონეზე. განვიხილოთ ეს ალგორითმები

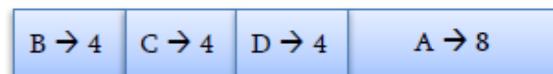
FCFS (First Come – First Served)

დაგეგმვის ყველაზე მარტივ ალგორითმს წარმოადგენს ალგორითმი FCFS (პირველი შემოვიდა - პირველი მომსახურდება). ამ ალგორითმის გამოყენებისას სისტემაში იქმნება მდგომარეობაში მზაყოფნა მყოფი პროცესების მიმდევრობა. სისტემაში წარმოქმნილი ყოველი ახალი პროცესი ამ მიმდევრობაში ემატება ბოლოდან. შესრულებას იწყებს მიმდევრობის თავში მყოფი პირველივე პროცესი. პროცესს პროცესორი გამოეყოფა იმ დროითი შუალედით რამდენიც საჭიროა მის შესასრულებლად. პროცესი პროცესორს გამოანთავისუფლებს მხოლოდ იმ შემთხვევაში, თუ მან დასრულდა ან გარკვეული მიზეზით (შეტანა/გამოტანის ოპერაციის ლოდინის გამო) მოხდა მისი ბლოკირება. მას შემდეგ რაც სისტემაში დაფიქსირდება მოვლენა, რომელსაც ბლოკირებული პროცესი ელოდებოდა ის გადადის მდგომარეობაში მზადყოფნა და მიმდევრობას ემატება ბოლოდან. ამ ალგორითმის უპირატესობა მდგომარეობს იმაში, რომ მისი რეალიზაცია და დაპროგრამება მარტივია. სისტემაში წარმოქმნილი ყოველი პროცესი იმართება ერთი მიმდევრობით. ამ მიმდევრობიდან შესასრულებლად ახალი პროცესის არჩევა ხორციელდება მიმდევრობის თავიდან, ხოლო სისტემაში ახალი პროცესის წარმოქმნისას ის ემატება მიმდევრობის ბოლოში. ალგორითმს ასევე გააჩნია ნაკლოვანება. დავუშვათ სისტემაში გვაქვს გამოთვლების სისწრაფით შემოფარგლული 1 პროცესი, რომელიც ყოველი ამუშავებისას პროცესორთან იმყოფება 1 წმ.-ის განმავლობაში და რამდენიმე შეტანა/გამოტანის მოწყობილობის მუშაობის სისწრაფით შემოფარგლული პროცესი, რომელიც პროცესორთან ატარებს მცირე დროს, მაგრამ საჭიროებს 1000 კითხვის ოპერაციას. გამოთვლების სისწრაფით შემოფარგლული პროცესი 1 წმ. პროცესორთან დაყოფის შემდეგ იწყებს მონაცემების კითხვას დისკიდან. შემდეგ იტვირთება მიმდევრობით შეტანა/გამოტანის მოწყობილობის სისწრაფით შემოფარგლული პროცესები და ასე გრძელდება სანამ არ დასრულდება ყველა პროცესი. თუკი დავუშვებთ, რომ შეტანა/გამოტანის მოწყობილობის მუშაობის სისწრაფით შემოფარგლული პროცესი დისკიდან მონაცემთა ერთი ბლოკის კითხვას ანდომებს 1 წმ.-ს, მაშინ მივიღებთ, რომ ის საკუთარ სამუშაოს დასასრულებლად საჭიროებს 1000 წმ.-ს. თუკი გამოვიყენებდით დაგეგმვის ისეთ ალგორითმს, რომელიც შეძლებდა გამოთვლის სისწრაფით შემოფარგლულ პროცესის ბლოკირებას ყოველ 10 წმ.-ში, მაშინ შეტანა/გამოტანის მოწყობილობის მუშაობის სისწრაფით შემოფარგლული პროცესები ნაცვლად 1000 წმ.-სა დასრულდებოდნენ 10 წმ.-ში, რაც არ გამოიწვევდა სისტემის საგრძნობ შენელებას. SJF (Short Job First)

ბუნებრივია, რომ თუკი სისტემაში ყველა პროცესი იარსებებდა ერთდროულად და წინასწარ იქნებოდა ცნობილი მათი შესრულებისათვის საჭირო პროცესორული დრო, მაშინ მარტივი იქნებოდა პროცესების დაგეგმვის ამოცანაც. SJF ალგორითმი (პირველი მოკლე ამოცანა) აგებულია ამ პრინციპით. SJF ალგორითმის გამოყენება გულისხმობს დამგეგმავის მიერ შესასრულებლად პირველი არჩეული იყოს მოკლე (ნაკლები პროცესორული დროის საჭიროების მქონე) ამოცანა. ამ ალგორითმშიც გამოიყენება ერთი მიმდევრობა. მიმდევრობა დალაგებულია ზრდადობით შესასრულებლად საჭირო დროითი შუალედის მიხედვით. განვიხილოთ მაგალითი. ნახ. 3.2-ზე გამოსახულია 4 პროცესისაგან შემდგარი მიმდევრობა, შესასრულებლად საჭირო დროითი შუალედებით 8, 4, 4, 4 დროითი ერთეული შესაბამისად. წარმოდგენილი მიმდევრობით პროცესების შესასრულებლად დაგვჭირდება შემდეგი დროები: A --> 8, B --> 12, C --> 16, d --> 20, ხოლო მათი შესრულების საშუალო დრო კი იქნება $(8 + 12 + 16 + 20) / 4 = 14$ ერთეული.



ა)



ბ)

ნახ. 3.2. დაგეგმვა. პირველი სრულდება მიმდევრობაში გამოჩენილი პირველივე პროცესი (ა). პირველი სრულდება შესასრულებლად ნაკლების დროის საჭიროების მქონე პროცესი (ბ).

თუკი იმავე პროცესების შესრულებას განვახორციელებთ ბ)-ზე წარმოდგენილი მიმდევრობით, მაშინ პროცესების შესასრულებლად დაგვჭირდება შემდეგი დროები: A --> 4, B --> 8, C --> 12, d --> 20, ხოლო მათი შესრულების საშუალო დრო კი იქნება $(4 + 8 + 12 + 20) / 4 = 11$ ერთეული.

შევნიშნოთ, შესაძლებელია იმის ჩვენება, რომ SJF ალგორითმი, თუ თავიდანვე ცნობლია სისტემაში არსებული ყველა პროცესი და მათი შესრულების დრო, მაშინ ის არისოპტიმალური. განვიხილოთ მაგალითი. ვთქვათ, მოცემულია 5 პროცესი - A – E, შესრულების დროით 2, 4, 1, 1, 1 შესაბამისად და მიმდევრობაში გამოჩენის დროით 0, 0, 3, 3, 3. ცხადია, რომ ამ შემთხვევაში პროცესების შესრულება განხორციელდება წარმოდგენილი თანმიმდევრობით (A – E) და შესაბამისად შესრულების დროის საშუალო იქნება $(2 + 6 + 4 + 5 + 6) / 5 = 4,6$ დროითი ერთეული. თუკი ყველა პროცესი თავიდანვე გვექნებოდა, მაშინ მათი შესრულების თანმიმდევრობა იქნებოდა C D E A B, ხოლო შესრულების საშუალო დრო კი $-(1 + 2 + 3 + 5 + 9) / 5 = 4$ ერთეული.

პრიორიტეტული SJF

SJF ალგორითმში პრიორიტეტის დამატებით მიიღება განსხვავებული ალგორითმი. ამ შემთხვევაშიც იგულისხმება, რომ პროცესების შესრულებისათვის საჭირო დრო წინასწარაა ცნობილი. სისტემაში ყოველი წარმოქმნილი პროცესის შესრულებისათვის საჭირო დრო ედრება მიმდინარე პროცესის დასრულდებისათვის საჭირო დროს. თუ ეს უკანასკნელი მეტია წარმოქმნილი პროცესის შესრულების დროზე, მაშინ მიმდინარე პროცესი წყვეტს შესრულებას და მის ადგილს იკავებს წარმოქმნილი პროცესი. თუკი წარმოქმნილი პროცესის შესრულების დრო მეტია მიმდინარე პროცესის დასრულებისათვის საჭირო დროზე, მაშინ წარმოქმნილი პროცესი ემატება მდგომარეობაში მზადყოფნა მყოფი პროცესების მიმდევრობას და იქ იკავებს შესაბამის ადგილს (პრიორიტეტის

მიხედვით). ასეთნაირად რეალიზებული SJF ალგორითმი იძლევა სისტემაში წარმოქმნილი მოკლე დროის საჭიროების მქონე პროცესების დროული შესრულების შესაძლებლობას.

ციკლური დაგეგმვა (RR - Round Robin)

ამოცანების დაგეგმვის ერთერთ მექანიზმი არის სამართლიან და ხშირად გამოყენებადალგორითმის წარმოადგენს ციკლური დაგეგმვის ალგორითმი. ამ შემთხვევაშიც სისტემაში ყოველი წარმოქმნილ პროცესი მდგომარეობაში მზადყოფნა მყოფი პროცესების მიმდევრობას ემატება ბოლოდან.

მიმდევრობის ელემენტებისათვის პროცესორის გამოყოფა ხორციელდება გარკვეული დროითი შუალედებით (კვანტით), რომლის ამოწურვის შემდეგ, თუ დრო პროცესის დასასრულებლად არ აღმოჩნდა საკმარისი, მაშინ ის ჩამოერთმევა მიმდინარე პროცესს, პროცესი გადადის მიმდევრობის ბოლოში (თავიდან იკავებს რიგს) და პროცესორი შესასრულებლად გადაეცემა მიმდევრობის შემდეგ წევრს. თუ დროითი კვანტი საკმარისი აღმოჩნდა პროცესის დასასრულებლად, მაშინ პროცესორი გადაეცემა მიმდევრობის შემდეგ წევრს. პროცესორის გამოყოფა ციკლურად ხორციელდება მანამ მიმდევრობა შეიცავს ერთ მაინც წევრს.

ციკლური დაგეგმვისათვის მნიშვნელოვან მომენტს წარმოადგენს დროითი კვანტისგანსაზღვრა. ერთი პროცესიდან მეორე პროცესზე გადართვა ადმინისტრირების ამოცანების შესასრულებლად (რეგისტრებისა და მეხსიერების რუკის შენახვა, სხვადასხვა ცხრილების განახლება) მოითხოვს გარკვეულ დროით შუალედს. თუ დავუშვებთ, რომ პროცესის გადართვა ადმინისტრირების ამოცანის განსახორციელებლად საჭიროებს 1 ნწმ. და დროითი კვანტი 4 ნწმ.-ია, მაშინ გამოდის, პროცესორის მიერ 4 ნწმ. სასარგებლო სამუშაოში გატარებულ დროს ყოველთვის მოყვება 1 ნწმ. (20%) არამომგებიან სამუშაოში დახარჯული დრო.

პროცესორის ეფექტურად გამოყენების თვალსაზრისით შესაძლებელია კვანტის გაზრდა 100 ნწმ.-დე. ამ შემთხვევაში იკარგება მხოლოდ 1%, მაგრამ ასეთი დიდი დროითი კვანტის გამოყენებას შეიძლება მოჰყვეს უარყოფითი შედეგი. განვიხილოთ სერვერული სისტემის მაგალითი, რომელშიც დროის მცირე შუალედში შემოსულია შესასრულებელი სხავდასხვა დროითი შუალედის მქონე 50 მოთხოვნა. ყველა მოთხოვნა შემოსვლისთანავე განთავსდება მიმდევრობაში და თუ პროცესორი არაა დაკავებული ის გამოყოფა მიმდევრობის პირველივე მოთხოვნას (პროცესს). მეორე პროცესი პროცესორს მიიღებს მხოლოდ 100 ნწმ.-ის გასვლის შემდეგ და ა.შ. თუკი დავუშვებთ, რომ ყველა პროცესი საკუთარ კვანტით დროს გამოიყენებს სრულად, მაშინ აღმოჩნდება, რომ მიმდევრობის ბოლო პროცესს მოუწევს 5 წმ.-ს დალოდება პროცესორის მისაღებად. რაც გამოიწვევს ნებისმიერი მომხმარებლის უკმაყოფილებას. შედეგი განსაკუთრებით არასასურველი იქნება იმ შემთხვევაში, თუკი პროცესი საჭიროებს რამდენიმე ნწმ.-ს. თუკი ამ შეთხვევაში დროითი კვანტის მნიშვნელობა იქნებოდა პატარა სიდიდე, მაშინ თითოეული მოთხოვნის მომსახურეობა რათქმაუნდა გაუმჯობესდებოდა.

მეორე თავისებურება მდგომარეობს იმაში, რომ თუ კვანტი მეტია პროცესორის ჩართულობის საშუალოზე, მაშინ პროცესები არ განახორციელებენ ხშირ გადართვას. ნაცვლად ამისა, პროცესების უმეტესი დაიკავებს პროცესორს კვანტის ამოწურვამდე (რომელიც გამოიწვევს პროცესის იძულებით გადართვას). იძულებითი წყვეტის არარსებობა ზრდის სისტემის წარმადობას, ვინაიდან პროცესების გადართვა განხორციელდება ლოგიკური აუცილებლობით, ანუ როდესაც მოხდა პროცესის ბლოკირება და ის ვერ აგრძელებს შესრულებას.

მაშასადამე, დროითი კვანტის საკმარისად მცირე სიდიდით განსაზღვრისას, მოხდება პროცესების ხშირი გადართვა და პროცესორის გამოყენების ეფექტურობა მცირდება, ხოლოდიდი მნიშვნელობით მოცემამ კი შეიძლება მიგვიყვანოს იქამდე, რომ მოკლე მოთხოვნებსდასჭირდეთ დიდი ხნით დალოდება.

პრიორიტეტული დაგეგმვა

დაგეგმვისას გარე ფაქტორების გათვალისწინებას მივყავართ პრიორიტეტულ ალგორითმებამდე. პრიორიტეტული დაგეგმვის ალგორითმების არსი მარტივია: სისტემაში გამოჩენილ ყოველ პროცესს ენიჭება გარკვეული რიცხვითი მნიშვნელობა. მდგომარეობაში მზადყოფნა მყოფი პროცესებიდან შესასრულებლად აირჩევა მაღალი პრიორიტეტის მქონე პროცესს.

მიუხედავად იმისა, თუ მიმდინარე მომენტში რამდენი მომხმარებელი სარგებლობს გამოთვლითი სისტემის მომსახურეობით გამოთვლით სისტემაში შესაძლებელია არსებობდეს რამდენიმე განსხვავებული პრიორიტეტის მქონე პროცესი. მაგალითად, ელექტრონული ფოსტის შემოწმების პროცესს უნდა გააჩნდეს ნაკლები პრიორიტეტი ვიდრე მომხმარებლის მიერ გააქტიურებულ ვიდეო ფაილის შესრულების პროცესს.

იმის გამო, რომ ოპერაციულ სისტემაში მუდმივად არ მოხდეს მაღალპრიორიტეტული პროცესების შესრულება და ამან არ გამოიწვიოს სხვა პროცესების დიდი ხნით ლოდინი ამ მიზნით სისტემაში შემოღებულია მექანიზმი, რომელიც უზრუნველყოფს დაბალპრიორიტეტული პროცესების შესრულებასაც. ამ მექანიზმის არსი მდგომარეობს შემდეგში: რამდენჯერაც განხორციელდება მაღალპრიორიტეტული პროცესისათვის პროცესორის გამოყოფა იმდენით მცირდება (იზრდება) პროცესის პრიორიტეტი (სხვადასხვასისტემებში პროცესებს პრიორიტეტები შეიძლება სხვადასხვაგვარად: მაღალი პრიორიტეტი შეიძლება იყოს ნაკლები მნიშვნელობა ან პირიქით). დამგეგმვი პრიორიტეტის მნიშვნელობის შეცვლას ანხორციელებს ტაიმერიდან მიღებული წყვეტის საფუძველზე.

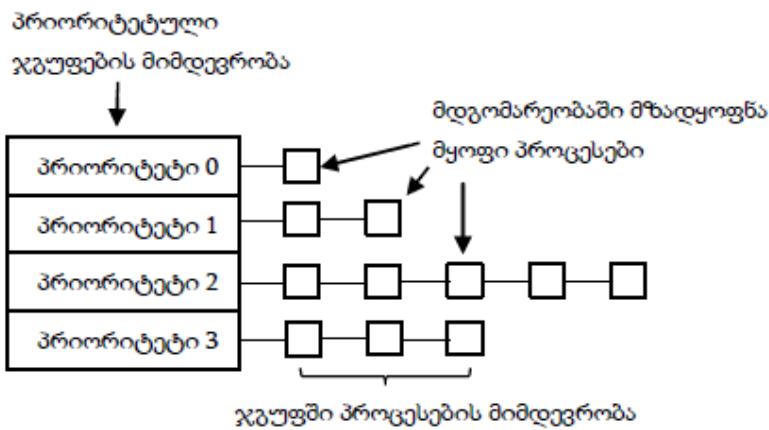
პროცესისათვის პრიორიტეტის მინიჭება შესაძლებელია განხორციელდეს დინამიურად ან სტატიკურად. Unix-მსგავს სისტემებში არსებობს ბრძანება nice, რომელიც გამოიყენება პროცესისათვის პრიორიტეტის შესაცვლელად.

პრიორიტეტი ასევე შეიძლება მინიჭებული იყოს დინამიურად გარკვეული სისტემური ამოცანების შესრულების მიზნით. მაგალითად, ზოგიერთი პროცესი შემოიფარგლება შეტანა/გამოტანის მოწყობილობის მუშაობის სისწრაფით და საკუთარი არსებობის მნიშვნელოვან ნაწილს ატარებენ ამ ოპერაციის დასრულების ლოდინში. როდესაც ასეთი პროცესი საჭიროებს პროცესორს მას დაუყოვნებლის უნდა მიეწოდოს ის, რათა პროცესმა შეძლოს შეტანა/გამოტანაზე მოთხოვნის დამუშავება, რომელიც შეიძლება მიმდინარეობდეს გამოთვლების პარალელურად. თუკი შეტანა/გამოტანის მოწყობილობის სისწრაფით შემოფარგლულ პროცესს ვაიძულებთ დიდი ხნით დაელოდოს პროცესორს, მაშინ აღმოჩნდება, რომ მის მიერ უაზროდაა დაკავებული ოპერატიული მეხსიერება დიდი ხნით.

მრავალდონიანი მიმდევრობა (Multilevel Queue)

ზოგჯერ ოპერაციულ სისტემაში მოხერხებულია პროცესების დაჯგუფება პრიორიტეტების ჯგუფების მიხედვით. ამ შემთხვევაში სისტემაში გვექნება მდგომარეობაში მზადყოფნა მყოფი პროცესების იმდენი მიმდევრობა რამდენიც განსხვავებული ჯგუფი პლიუს ერთი, ანუ ყველ ჯგუფში თითო მიმდევრობა და ერთიც ჯგუფების მიმდევრობა. ჯგუფების მიმდევრობის მიმართ შესაძლებელია გამოყენებული იქნას პრიორიტეტული დაგეგმვა, ხოლო პრიორიტეტების ჯგუფებში კი ზემოთ განხილული ალგორითმებიდან ნებისმიერი. პროცესების შესრულება ხორციელდება შემდეგნაირად: პირველი შესრულდება მაღალპრიორიტეტული ჯგუფის პროცესები. ამის შემდეგ პროცესორი შეიძლება გამოეყოს პრიორიტეტით შემდეგ ჯგუფს, თუ მასში ყველა პროცესი დასრულებულია ან რომელიმე მათგანი იმყოფება ლოდინის მდგომარეობაში. ლოდინის მდგომარეობიდან მდგომარეობაში მზადყოფნა გადასვლის შემთხვევაში მაღალპრიორიტეტულ პროცესს უბრუნდება პროცესორი.

პრიორიტეტით შემდეგი ჯგუფის ყველა პროცესის დასრულების შემდეგ პროცესორი გადაეცემა მომდევნო ჯგუფს და ასე გრძელდება სანამ არ ამოიწურება პროცესები ყველა ჯგუფში. ასეთნაირად აღწერილ ალგორითმს მრავალდონიანი მიმდევორბა ეწოდება(ნახ. 2.3).



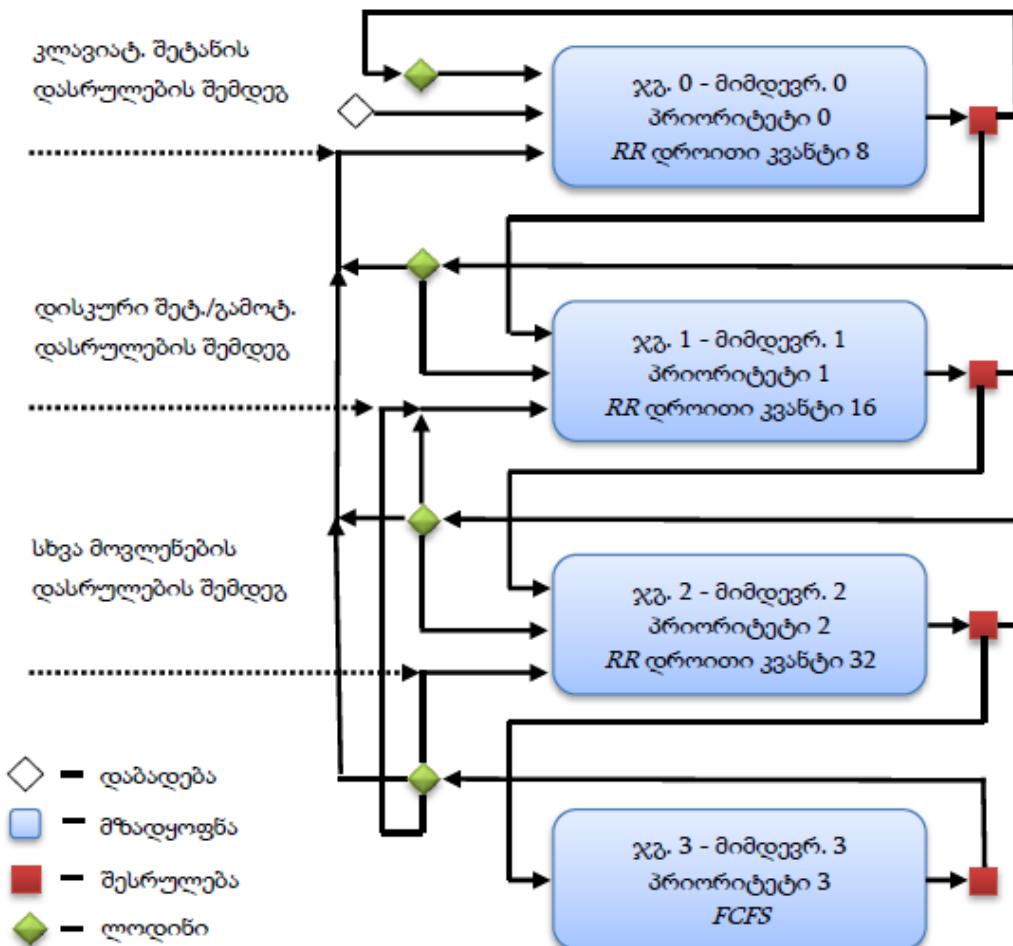
ნახ. 3.3. დაგეგმვის მრავალდონიანი მიმდევრობის ალგორითმი 4 პრიორიტეტული ჯგუფით (დაბალი მნიშვნელობა აღნიშნავს მაღალ პრიორიტეტს).

მრავალდონიანი მიმდევრობის გამოყენებას გააჩნია თავისი ნაკლოვანება, რაც გამოიხატება შემდეგში: სისტემაში მუდმივად წარმოიქმნება ახალი პროცესები, რომლებიც შესასრულებლად საჭიროებენ სხვადასხვა რესურსებსა და შესრულების განსხვავებულდოროს. შეიძლება აღმოჩნდეს, რომ მაღალპრიორიტეტული პროცესები, რომლებიც მოითხოვნ შესრულების ხანგრძლივ პერიოდს პროცესორს იკავებენ დიდი ხნით და ამ ხნის განმავლობაში არაერთი მაღალპრიორიტეტული პროცესი შეიძლება წარმოქმნას სისტემაში. შესაბამისად, წარმოიშობა საფრთხე, რომ პროცესორი შეიძლება საერთოდ არ გამოეყოს დაბალპრიორიტეტული ჯგუფის პროცესებს ან ამ ჯგუფის რომელიმე პროცესის, რომელიც შესასრულებლად საჭიროებს ხანგრძლივ პერიოდს, შესრულება გაგრძელდეს უსასრულოდ.

ასეთი სიტუაციის თავიდან აცილების მიზნით საჭიროა დამატებითი მექანიზმის გათვალისწინება, რომლის მიხედვითაც მოხდება პრიორიტეტის კონკრეტული მნიშვნელობის მქონე ჯგუფის პროცესის, რომელსაც ერთხელ მაინც გამოეყო პროცესორი და არ დაუმთავრებია შესრულება, გადაყვანა ამა თუ იმ ჯგუფში. მრავალდონიან მიმდევრობაში მსგავსი მექანიზმის გათვალისწინებით მიღებულ ალგორითმს ეწოდება მრავალდონიანი მიმდევრობები უკუკავშირით (**Multilevel Feedback Queue**).

დაგეგმვის „მრავალდონიანი მიმდევრობები უკუკავშირით“ ალგორითმის მუშაობის საილუსტრაციოდ განვიხილოთ მაგალითი, რომელშიც გვაქვს პრიორიტეტის 4 ჯგუფი (ნახ. 3.4) (პრიორიტეტის დაბალი მნიშვნელობა აღნიშნავს მაღალ პრიორიტეტს). დავუშვათ ჯგუფებისათვის პრიორიტეტის მნიშვნელობა განსაზღვრულია მასში შემავალი პროცესების შესასრულებლად საჭირო დროით. ჯგუფების მიმდევრობისათვის ხორციელდება პრიორიტეტული დაგეგმვა, რაც ნიშნავს, რომ პირველი შესრულდება ჯგუფი 0-ის ყველა პროცესი, შემდეგი შესრულდება ჯგუფი 1-ის ყველა პროცესი და ასე გაგრძელდება სანამრომელიმე ჯგუფი შეიცავს ერთ მაინც პროცესს. როგორც ნახ. 3.4-დან ჩანს 0-2 ჯგუფებში შიგნით არსებული მიმდევრობებისათვის დაგეგმვა ხორციელდება RR ალგორითმით, ხოლო ჯგუფ 3-ში არსებული მიმდევრობისათვის კი - FCFS ალგორითმით.

სისტემაში წარმოქმნილი ყოველი ახალი პროცესის განთავსება ამა თუ იმ ჯგუფში ხორციელდება იმის მიხედვით, თუ რამდენ დროს საჭიროებს ის შესასრულებლად, ანუ თუ ის შესასრულებლად საჭიროებს 1 – 8 ერთეულს მოხვდება ჯგუფ 0-ში; თუ საჭიროებს 9 – 16 ერთეულს მოხვდება ჯგუფში 1; თუ საჭიროებს 17 – 32 ერთეულს მოხვდება ჯგუფ 2-ში. წინააღმდეგ შემთხვევაში მოხვდება ჯგუფ 3-ში. როგორც უკვე აღვნიშნეთ ჯგუფი 1-ის პროცესების შესრულება დაიწყება მას შემდეგ რაც ჯგუფ 0-ში არ გვექნება არცერთი პროცესი. ჯგუფი 1-ის რომელიმე პროცესის შესრულებისას, თუ სისტემაში წარმოქმნა რაიმე პროცესი, რომელიც შეიძლება მიეკუთვნებოდეს ჯგუფ 0-ს, მაშინ მიმდინარე პროცესს ჩამოერთმევა პროცესორი და ის გადაეცემა წარმოქმნილ პროცესს. თუკი მოცემული მომენტისათვის



ნახ. 3.6. პროცესების მიგრაციის სქემა დაგეგმვის მრავალდონიანი მიმდევრობებში უკუკავშირით. სქემაში არაა ნაჩვენები პროცესების გამევება და დასრულება

პროცესორის ჩამორთმევის შემდეგ მიმდინარე პროცესი შესასრულებლად საჭიროებს 1 – 8 ერთეულს, მაშინ მოხდება მისი გადაყვანა ჯგუფ 0-ში. ანალოგიურად მოხდება სხვა ჯგუფების შემთხვევაშიც. მაგალითად, თუ მოცემულ მომენტში სრულდებოდა ჯგუფი 3-ის რაიმე პროცესი და მას სისტემაში წარმოქმნილი მაღალპრიორიტეტული პროცესის გამო ჩამოერთვა პროცესორი, მაშინ მიმდინარე პროცესის განთავსება რომელიმე ჯგუფში მოხდება იმის მიხედვით, თუ რამდენი დრო

დარჩა მას შესასრულებლად: 1 – 8 ერთეულის შემთხვევაში ჯგუფ 0-ში, 9 – 16 ერთეულის შემთხვევაში ჯგუფ 1-ში და ა.შ.

მრავალდონიანი მიმდევრობები უკუკავშირით წარმოადგენს პროცესების დაგეგმვისადმი უფრო ზოგად მიდგომას ჩვენს მიერ განხილული მიდგომებიდან. ის რეალიზაციაში საკმაოდ რთულია და ამავდროულად მოქნილიც.

ცხადია, რომ ჩვენს მიერ განხილული დაგეგმვის ალგორითმების გარდა არსებობს სხვა მრავალი ალგორითმიც. მათი კონკრეტული რეალიზაციის სრულად აღწერისათვის საჭიროა მივუთითოთ:

- მდგომარეობაში მზადყოფნა მყოფი პროცესების მიმდევრობების რაოდენობა;
- პრიორიტეტების ჯგუფების მიმდევრობისათვის გამოყენებული დაგეგმვის ალგორითმი;
- პრიორიტეტების ჯგუფების შიგნით გამოყენებული დაგეგმვის ალგორითმი;
- წესი, რომლითაც განხორციელდება სისტემაში წარმოქმნილი პროცესის რომელიმეჯგუფში განთავსება;
- პროცესების ერთი მიმდევრობიდან მეორეში გადაყვანის წესი.

მონაცემების გაცვლის საშუალებები

ურთიერთქმედ პროცესებს შორის გადასაცემი მონაცემების მოცულობის და პროცესის ყოფაქცევაზე შესაძლო ზემოქმედების მიხედვით გაცვლის საშუალებები იყოფა სამ ნაწილად:

- **სიგნალური.** გადასაცემი ინფორმაციის მოცულობა მინიმალურია - ერთი ბიტი. სიგნალური საშუალებით ხდება პროცესორის ინფორმირება გარკვეული მოვლენისდადგომის და მასზე შესაბამისი რეაგირების საჭიროების თაობაზე. ამ შემთხვევაში მონაცემების მიმღები პროცესის ყოფაქცევაზე ზემოქმედება მინიმალურია. ყველაფერიდამოკიდებულია იმაზე, გააჩნია თუ არა მას მიღებული სიგნალის ტიპზე ინფორმაციადა როგორ დაამუშაოს ის. სიგნალზე არასწორმა რეაგირებამ ან მისმა უგულვებელყოფამშესაძლებელია მიგვიყვანოს გაუთვალისწინებელ შემთხვევამდე;
- **არხული.** პროცესებს შორის მონაცემების გაცვლა ხორციელდება ოპერაციულისისტემის მიერ ამ მიზნისათვის სპეციალურად გამოყოფილი კავშირის არხებით. გადასაცემი მონაცემების მოცულობა დამოკიდებულია კავშირის არხისგამტარუნარიანობაზე. გადასაცემი მონაცემების მოცულობის გაზრდით იზრდება სხვაპროცესების ყოფაქცევაზე ზემოქმედების შესაძლებლობა;
- **განაწილებადი მეხსიერება.** ორ ან რამდენიმე პროცესს შეუძლია შეთანხმებულადგამოიყენოს მისამართების სივრცის გარკვეული ნაწილი. გადასაცემი მონაცემებისმოცულობა დამოკიდებულია მეხსიერების გამოყოფილ ნაწილზე. მონაცემებისმაქსიმალური მოცულობით გადაცემა საჭიროებს გარკვეულ სიფრთხილეს, ვინაიდანმან შეიძლება ზეგავლენა იქონიოს სხვა პროცესების ყოფაქცევაზე. განაწილებადიმეხსიერება წარმოადგენს ერთ გამოთვლით სისტემაში პროცესების ურთიერთქმედებისსაკმაოდ ჩქარ მეთოდს.

ურთიერთქმედ პროცესებს შორის მონაცემების მიღება/გადაცემის პროგრამირება, განაწილებადი მეხსიერების გამოყენებით, ხორციელდება პროგრამირების ენების საშუალებებით, ხოლო სიგნალური და არხული საშუალებები კი საჭიროებენ სპეციალურ სისტემურ გამოძახებებს.

კომუნიკაციის საშუალებების ლოგიკური ორგანიზაცია

კომუნიკაციის საშუალებების განხილვისას ჩვენი ინტერესის სფეროს წარმოადგენს არა მათი ფიზიკური რეალიზაცია, არამედ ლოგიკური, რომელიც განსაზღვრავს მათი შესრულების მექანიზმებს. მოკლედ აღვწეროთ ისინი.

კავშირის დამყარება

ბუნებრივია ისმის კითხვა: ურთიერთქმედ პროცესებს შორის მონაცემების გასაცვლელად კავშირის საშუალება უშუალოდ პროცესის წარმოქმნასთან ერთად უნდა იყოს გამოყენებული, თუ მონაცემების გაცვლამდე საჭიროა გარკვეული ინიციალიზაცია? მაგალითად, სხვადასხვა პროცესები საერთო მეხსიერების გამოყენების მიზნით ოპერაციულ სისტემას მიმართავენ სპეციალური ფორმით, რომელიც შემდგომ გამოყოფს მას. მაგრამ პროცესებს შორის მონაცემების გადაცემა არ საჭიროებს ინიციალიზირებას.

კავშირის დამყარებასთან მჭიდრო კავშირშია კავშირის საშუალებების გამოყენებისას დამისამართების მეთოდები. ინფორმაციის გადაცემა/მიღებისას რიგ შემთხვევაში საჭიროა ცხადად იყოს მითითებული ინფორმაციის გამგზავნი და მიმღები.

განასხვავებენ დამისამართების ორ მეთოდს: პირდაპირი და არაპირდაპირი. პირდაპირი დამისამართებისას ურთიერთქმედი პროცესები პირდაპირ ურთიერთქმედებენ ერთიმეორებულად. გაცვლის ყოველი ოპერაციისას ცხადად ეთითება იმ პროცესის სახელი ან ნომერი, რომელსაც ეგზავნება მონაცემები, ან რომელმაც უნდა გამოაგზავონს ისინი. თუ გამგზავნი და მიმღები პროცესი ცხადად უთითებს ურთიერთქმედების პარტნიორს, მაშინ დამისამართების ასეთ სქემას სიმეტრიული პირდაპირი დამისამართება ეწოდება. წინააღმდეგ შემთხვევაში დამისამართების სქემას ეწოდება ასიმეტრიული პირდაპირი დამისამართება. პროცესების სიმეტრიული პირდაპირი ურთიერთქმედებისას სხვა პროცესს არ შეუძლია ჩაერიოს მათ საქმიანობაში, დაიჭიროს გაგზავნილი მონაცემები ან შეცვალოს ისინი.

არაპირდაპირი დამისამართებისას გამგზავნი პროცესის მიერ გაგზავნილი მონაცემები თავსდება გარკვეულ შუალედურ ობიექტში, საიდანაც მისი აღება შეუძლია მსგავსი მონაცემების საჭიროების მქონე რომელიმე პროცესს. ამასთან, არცერთი მხარე, რომელმაც განათავსა მონაცემები შუალედურ ობიექტში და რომელმაც აიღო ისინი იქიდან, არ საჭიროებს მეორის იდენტიფიცირებას.

პირდაპირი დამისამართების გამოყენებისას პროცესებს შორის კავშირი კლასიკურ ოპერაციულ სისტემებში მყარდება ავტომატურად, დამატებითი მაინიცირებელი მოქმედებების გარეშე. ამ შემთხვევაში კავშირის საშუალებების გამოსაყენებლად საჭიროა გაცვლაში მონაწილე პროცესების იდენტიფიცირების პროცედურის ცოდნა.

არაპირდაპირი დამისამართების შემთხვევაში შესაძლებელია არც იყოს საჭირო კავშირის საშუალებების ინიციალიზირება. ამ შემთხვევაში პროცესს უნდა გააჩნდეს ინფორმაცია მონაცემების შესანახი შუალედური ობიექტის იდენტიფიკატორზე.

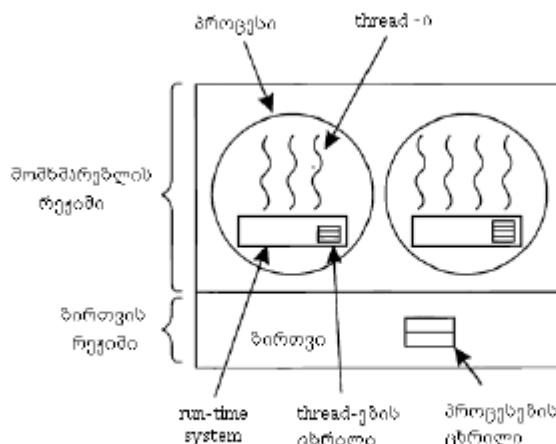
thread-ების რეალიზაცია

thread-ები რეალიზაციის შესაძლებელია განხორციელდეს როგორც ბირთვის ისე, მომხმარებლის რეჟიმში. ასევე განიხილება thread-ები რეალიზების კიდევ ერთი მეთოდი, რომელიც წარმოადგენს დავის საგანს და ცნობილია ჰიბრიდული მეთოდის სახელით. ჰიბრიდული მეთოდი გულისხმობს

მომხმარებლის და ბირთვის რეჟიმში ერთდროულად thread-ების არსებობას ისე, რომ შენარჩუნებული იყო მათი გამოყენებით მიღებული უპირატესობები.

thread-ების რეალიზაცია მომხმარებლის რეჟიმში

thread-ის რეალიზაციის პირველი მეთოდი გულისხმობს thread-ების სრული ნაკრების განთავსებას მომხმარებლის რეჟიმში. ამ ნაკრების არსებობაზე ბირთვის წარმოდგენა არ გააჩნია. ის ურთიერთქმედებს პროცესებთან, რომლებიც მომხმარებლის რეჟიმში წარმოდგენლია thread-ების ნაკრების სახით. thread-ების გამოყენების მნიშვნელოვანი უპირატესობა მდგომარეობს იმაში, რომ მათი რეალიზება შესაძლებელია ერთპროცესორულ სისტემაში, რომელსაც შეიძლება არც გააჩნია thread-ების მხარდაჭერა. თითქმის ყველა თანამედროვე ოპერაციული სისტემა განეკუთვნება ამ კატეგორიას. ასეთ სისტემებში thread-ები რეალიზებულია სპეციალური ბიბლიოთეკების მეშვეობით. სისტემაში thread-ები რეალიზებულია ერთიანი სტრუქტურით (ნახ. 4.4). ისინი სრულდებიან პროგრამების შესრულების სისტემაში (run-time systems), რომელშიც თავმოყრილია thread-ების მმართველ პროცედურათა ნაკრებები (pthread_create, pthread_exit, pthread_join, pthread_yield და ა.შ.). მომხმარებლის რეჟიმში ყოველ პროცესს გააჩნია thread-ების ცხრილის მხარდაჭერა, რომელშიც ინახება პროცესის მის შიგნით არსებულ thread-ებზე ინფორმაცია. ის წარმოადგენს ბირთვის რეჟიმში არსებული პროცესების ცხრილის ანალოგს, იმ განსხვავებით, რომ ყოველი thread-ისათვის ის შეიცავს მის ბრძანებათა მთვლელს, სტეკის მიმთითებელს, რეგისტრებს, მდგომარეობას და ა.შ., და პროცესი მას იყენებს thread-ებზე გარკვეული მოქმედებების გასახორციელებლად (წარმოქმნა, წაშლა და ა.შ.). thread-ის წარმოქმნის, მდგომარეობის შეცვლის, ან მის მიერ გარკვეული სამუშაოს შესრულების შემდეგ ხდება thread-ების ცხრილის შესაბამის ჩანაწერის განახლება.



ნახ. 4.4. thread-ების ნაკრები მომხმარებლის რეჟიმში

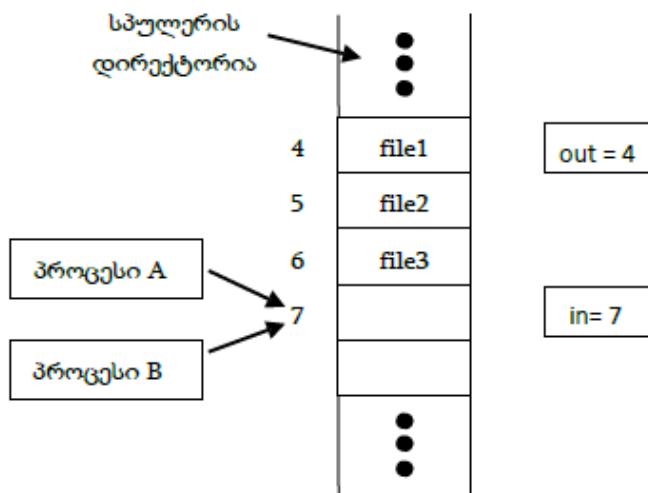
თუ thread-ის მიერ განხორცილებულმა მოქმედებამ შეიძლება გამოიწვიოს მისი ბლოკირება, მაშინ ამ მოქმედების განხორცილებამდე ის იძახებს run-time system-ის შესაბამის პროცედურას, რომელიც აფასებს ბლოკირების შესაძლებლობას. თუ ბლოკირება გარდაუვალია, მაშინ ის thread-ების ცხრილში ინახავს საკუთარი რეგისტრების შემცველობას და წყდება მისი შესრულება. ამის შემდეგ thread-ების ცხრილიდან რეგისტრებში იტვირთება ახალი thread-ის შესაბამისი ინსტრუქციები და შესრულებას იწყებს ის.

ერთერთი მნიშვნელოვანი განსხვავება thread-ისა პროცესისაგან მდგომარეობს იმაში, რომ, თუ thread-ის მიერ შესრულებული პროცედურა იწყვევს მის შეჩერებას გარკვეული დროითი შუალედით,

მაშინ გამოყენებული პროცედურა ინახავს ინფორმაცის thread-ის მიმდინარე მდგომარეობაზე და მის მიერ განხორციელებულ სამუშაოზე. პროცედურას ასევე შეუძლია გამოიძახოს დამგეგმავი შესასრულებლად შემდგომი thread-ის ასარჩევად. მსგავსი პროცედურები ლოკალურია და მათი გამოყენება უფრო ეფექტურია ვიდრე ბირთვის რეჟიმში შესაბამის გამოძახებების გამოყენება. მომხმარებლის რეჟიმში რეალიზებულ thread-ებს, მრავალ უპირატესობასთან ერთად, გაჩნია ნაკლოვანებებიც. ასეთი საკითხების რიცხვს განკუთვნება შემდეგი: როგორ უნდა მოხდეს thread-ის ბლოკირება; ერთპროცესორულ სისტემაში დროის ნებისმიერ მომენტში პროცესორთან იმყოფება ერთი thread-ი. ერთი პროცესის ფარგლებში არ არსებობს ტაიმერიდან წყვეტა. როგორ უნდა მოხდეს thread-ისათვის პროცესორის ჩამორთმევა, თუ მან უსასრულოდ გადაწყვიტა შესრულება; არ არსებობს thread-ების სიგნალებით ინფორმირების მექანიზმი.

შეჯიბრის მდგომარეობა

პროცესები შესაძლებელია შეთანხმებულად იყენებდნენ ოპერაციულ სისტემაში არსებულ მონაცემთა საერთო საცავს. ეს საცავი შეიძლება იყოს განთავსებული ოპერატიულ მეხსიერებაში ან წარმოდგენილი იყოს რაიმე ფაილის სახით. პროცესების ურთიერთქმედების საკითხში უკეთ გასარკვევად განვიხილოთ მაგალითი. განვიხილოთ ბეჭდვის სპულერის კატალოგი. როდესაც პროცესი საჭიროებს რაიმე ფაილის ბეჭდვას ის ანთავსებს სპულერის კატალოგში ფაილის სახელს. ოპერაციულ სისტემაში არსებული მეორე პროცესი - პრინტერის დემონი, პერიოდულად ამოწმებს სპულერის კატალოგს და იქ ჩანაწერის (ფაილზე მიმთითებლის) აღმოჩენის შემთხვევაში მიმართავს შესაბამის ფაილს, ბეჭდვას მას და კატალოგიდან შლის ჩანაწერს (ფაილზე მიმთითებელს).



ნახ. 5.1. შეჯიბრის მდგომარეობა

დავუშვათ, სპულერის კატალოგში განთავსებულია გარკვეული რაოდენობის მეხსიერების არეები, 1,2,..., რომელთაგან თითოეულში შეიძლება ინახებოდეს ფაილის სახელი. ასევე დავუშვათ გვაქვს ორი ცვლადი: in, რომელიც უთითებს კატალოგში შემდეგ ცარიელ არეზე და out, რომელიც უთითებს შემდეგი დასაბეჭდი ფაილის სახელზე. დავუშვათ, დროის გარკვეულ მომენტში 1 – 3 არეები ცარიელია, ხოლო 4 – 6 არეებში კი განთავსებულია დასაბეჭდი ფაილების სახელები (ნახ. 5.1) და თითქმის ერთდროულად ორმა პროცესმა (A და B) გადაწყვიტა საკუთარი მონაცემების განთავსება

დასაბეჭდი ფაილების მიმდევრობაებში (სპულერის კატალოგში). ამ შემთხვევაში შეიძლება წარმოიქმნას შემდეგი სიტუაცია: A პროცესმა წაიკითხა in ცვლადის მნიშვნელობას და ის შეინახა საკუთარ ლოკალურ ცვლადში next_free_slot (მომდევნო ცარიელი არე). ამის შემდეგ ტაიმერიდან წყვეტის გამო პროცესორი A პროცესიდან გადაერთო B პროცესზე. B პროცესიც ხედავს in ცვლადის მნიშვნელობას და ინახავს მას საკუთარ ლოკალურ ცვლადში (next_free_slot). მიმდინარე მომენტისათვის ორივე პროცესი თვლის, რომ საკუთარი მონაცემების განსათავსებლად მათ შეუძლიათ მიმართონ არეს ნომრით 7 (მონაცემების განთავსება სპულერის კატალოგში ხორციელდება ზრდადი მიმდევრობით). ვინაიდან სრულდება B პროცესი, ის ამ არეში ანთავსებს საკუთარ მონაცემებს და in ცვლადის მნიშვნელობას ზრდის 1-ით. ამის შემდეგ ის ასრულებს გარკვეულ მოქმედებებს და დროითი კვანტის გასვლის გამო პროცესორი გადაერთვება A პროცესზე. (ვგულისხმობთ, რომ დროითი კვანტის განმავლობაში არ განხორციელებულა B პროცესის მიერ დასაბეჭდად გადაცემული ფაილის ბეჭდვა.) ვინაიდან A პროცესი განახლდება იმ ადგილიდან, სადაც მოხდა მისი შეჩერება და მას საკუთარ ლოკალურ ცვლადში ჩაწერილი აქვს მნიშვნელობა 7, ის მიმართავს ამ არეს და იქ განათავსებს საკუთარი ფაილის სახელს, რომელიც გადაეწერება იქ არსებულს (B პროცესის მიერ ჩაწერილ სახელს). შემდეგ in ცვლადის მნიშვნელობას ზრდის 1-ით. სპულერში შიდა წინააღმდეგობა არ არსებობს, ამიტომ პრინტერის დემონი ვერასოდეს დააფიქსირებს მომხდარ ცვლილებას. მომხმარებელი კი, რომელმაც გარკვეული სამუშაოს შესასრულებლად წარმოქმნა B პროცესი, ნაბეჭდი ფორმით ვერასოდეს ვერ მიიღებს მის მიერ განხორციელებული სამუშაოს შედეგს.

სიტუაციას, რომლის დროსაც ორი ან მეტი პროცესი იყენებს საერთო რესურსს და მიღებული შედეგი დამოკიდებულია ბოლოს შესრულებულ პროცესზე ეწოდება შეჯიბრის მდგომარეობა.

კრიტიკული სექცია

ბუნებრივია ისმის კითხვა: როგორ უნდა ავიცილოთ შეჯიბრის მდგომარეობა? საზოგადოდ, თუ პროცესები იყენებენ „საერთო“ ცნებით დაკავშირებულ რესურსებს, მაშინ შეჯიბრის მდგომარეობის თავიდან აცილების მიზნით საჭიროა განსაზღვრული იყოს მეთოდი, რომელიც გამორიცხავს რამდენიმე პროცესის მხრიდან რესურსებზე ერთდროული წვდომის შესაძლებლობას. სხვა სიტყვებით, რომ ვთქვათ, საჭიროა ურთიერთგამორიცხვის მეთოდი, რომლის მიხედვითაც, თუ რაიმე პროცესი იყენებს სხვა პროცესებთან საზიარო რესურსს, მაშინ სანამ ის არ დაასრულებს ამ რესურსთან მუშაობას სხვა პროცესი ვერ მიიღებს მას. ზემოთ განხილულ მაგალითში პრობლემა მდგომარეობდა იმაში, რომ B პროცესმა იმაზე ადრე მიმართა საზიარო რესურსს (ცვლადს) ვიდრე A პროცესი დაასრულებდა მასთან მუშაობას. ოპერაციული სისტემების კონსტრუქციის მნიშვნელოვან ნაწილს წარმოადგენს იმ ელემენტარული მოქმედებების განსაზღვრა, რომლებიც გამოიყენებიან ურთიერთგამორიცხვის მიღწევის მიზნით.

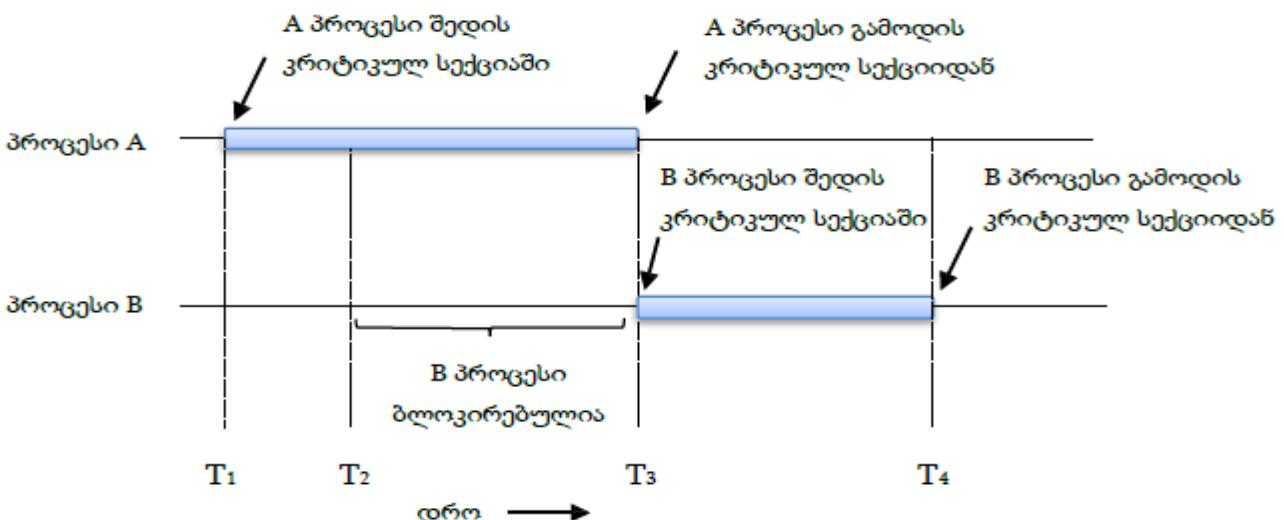
შეჯიბრის მდგომარეობის ფორმულირება შესაძლებელია აბსტრაქტული ფორმით:

შესრულებისას პროცესი დაკავებულია გამოთვლებით ან ლოგიკური მოქმედებებით, რომელიც შესაძლებელია იწვევდეს შეჯიბრის მდგომარეობას. პროგრამის იმ ნაწილს, რომელშიც გამოიყენება საერთო მეხსიერებაზე წვდომა კრიტიკული სექცია ეწოდება. თუ გვექნებოდა მექანიზმი, რომელიც უზრუნველყოფდა ურთიერთქმედი პროცესებიდან (დროის ნებისმიერ მომენტში) საკუთარ კრიტიკულ სექციაში მხოლოდ ერთის არსებობას, მაშინ შევძლებდით შეჯიბრის მდგომარეობის აცილებას.

მიუხედავად იმისა, რომ ეს მოთხოვნა იძლევა შეჯიბრის მდგომარეობის აცილების საშუალებას ის არაა საკმარისი იმისათვის, რომ პარალელურმა პროცესებმა სწორად წარმართონ საკუთარი მუშაობა და ეფექტურად გამოიყენონ საერთო რესურსები. პროცესის მიერ შესრულებული სამუშაო, რომ იყოს მისაღები საჭიროა შემდეგი პირობების შესრულება:

1. ორ პროცესს არ შეუძლია ერთდღოულად იმყოფებოდეს საკუთარ კრიტიკულ სექციაში;
2. არ უნდა არსებობდეს არანაირი წინასწარი მოსაზრება პროცესორების სავარაუდო რაოდენობაზე;
3. არცერთი პროცესი, რომელიც სრულდება საკუთარი კრიტიკული სექციის გარეთ არ შეიძლება იყოს ბლოკირებული სხვა პროცესის მიერ;
4. პროცესები უსასრულოდ არ უნდა ელოდებოდნენ საკუთარ კრიტიკულ სექციაში შესვლას.

ნახ. 5.2-ზე ნაჩვენებია კრიტიკული სექციების ურთიერთგამორიცხვის მაგალითი. A პროცესი შედის საკუთარ კრიტიკულ სექციაში T_1 დროს. T_2 დროს საკუთარ კრიტიკულ სექციაში შესვლას ცდილობს B პროცესი, მაგრამ ვერ ახერხებს იმის გამო, რომ A პროცესი იმყოფება საკუთარ კრიტიკულ სექციაში. ამის შემდეგ ხდება B პროცესის ბლოკირება სანამ A პროცესი არ გამოვა იქიდან. A პროცესის კრიტიკული სექციიდან გამოსვლის შემდეგ B პროცესი გამოდის ბლოკირების მდგომარეობიდან და შედის საკუთარ კრიტიკულ სექციაში (T_3 დროს), საიდანაც გამოდის T_4 დროს და ვუბრუნდებით საწყის მდგომარეობას.



ნახ. 5.2. კრიტიკული სექციების ურთიერთგამორიცხვა

ურთიერთგამორიცხვა აქტიური ლოდინით

შემუშავებული იყო ურთიერთგამორიცხვის პრობლემის გადაწყვეტის მრავალი მეთოდი, რომელთაგან თითოეულს გააჩნია საკუთარი უპირატესობა და ნაკლოვანება. განვიხილოთ ზოგიერთი მეთოდი.

წყვეტის აკრძალვა

ერთპროცესორული სისტემის შემთხვევაში კრიტიკული სექციის პრობლემის გადაწყვეტა მარტივია. ამ შემთხვევაში შეიძლება პროცესს მიეცეს უფლება განახორციელოს ყველანაირი წყვეტის აკრძალვა და კრიტიკული სექციიდან გამოსვლის შემდეგ კი ჩამოერთვას ის. წყვეტის აკრძალვის შემთხვევაში შეუძლებელი იქნება ტაიმერიდან წყვეტის განხორციელება და შესაბამისად პროცესორიც ვერ შეძლებს სხვა პროცესზე გადართვას. ამ შემთხვევაში არ არსებობს საფრთხე რომელიმე პროცესმა განახორციელოს საკუთარ კრიტიკულ სექციაში შესვლა, მაგრამ შეიძლება წაარმოიშვას სხვა ტიპის პრობლემა. რა შეიძლება მოჰყვეს პროცესის გადაწყვეტილებას აკრძალოს წყვეტა და შემდეგ არ დაუშვას მისი შესაძლებლობა? ამან შესაძლებელია მიგვიყვანოს მთლიანი სისტემის უმოქმედობამდე.

მრავალპროცესორული სისტემის შემთხვევაში წყვეტის აკრძალვა მოქმედებს მხოლოდ იმ პროცესორზე, რომელსაც შეეხება ბლოკირების ინსტრუქცია. სხვა დანარჩენი პროცესორები ჩვეულ რეჟიმში გააგრძელებენ მუშაობას. მიუხედავად იმისა, რომ მრავალპროცესორულ სისტემაში შესაძლებელია მეთოდმა გაამართლოს მაინც დაუშვებელია პროცესისათვის წყვეტის აკრძალვის შესაძლებლობის მიცემა.

ცვლადი-ბოქლომი

ამ მეთოდის გამოყენება გულისხმობს სისტემაში პროცესებისათვის საერთო ცვლადის (ცვლადი-ბოქლომი) შემოღებას, რომელიც გააკონტროლებს პროცესების შესვლას საკუთარ კრიტიკულ სექციაში. ცვლადი-ბოქლომის საწყისი ინიციალიზაცია ხდება 0-ვანი მნიშვნელობით, რაც პროცესებისათვის აღნიშნავს, რომ საკუთარ კრიტიკულ სექციაში არ იმყოფება არცერთი პროცესი და ნებისმიერ მათგანს სურვილის შემთხვევაში შეუძლია შევიდეს იქ. პროცესი საკუთარ კრიტიკულ სექციაში შესვლის შემდეგ ცვლის ცვლადი-ბოქლომის მნიშვნელობას 1-ით, რაც იმის მანიშნებელია, რომ ურთიერთქმედი პროცესებიდან ერთერთი იმყოფება საკუთარ კრიტიკულ სექციაში და, თუ რომელიმე პროცესი საჭიროებს საკუთარ კრიტიკულ სექციაში შესვლას, მაშინ მას მოუწევს დალოდება სანამ მიმდინარე პროცესი არ გამოვა იქიდან. პროცესი საკუთარი კრიტიკული სექციიდან გამოსვლის შემდეგ ცვლად-ბოქლომს უბრუნებს საწყის მნიშვნელობას (0-ს).

ამ მიღომასაც გააჩნია ნაკლოვანება, რომლის მსგავსიც უკვე ვნახეთ სპულერის კატალოგის შემთხვევაში. მართლაც, დავუშვათ რაიმე პროცესი ამოწმებს ცვლადი-ბოქლომის მნიშვნელობას და აღმოაჩინა, რომ ის 0-ის ტოლია. სანამ პროცესი განახორციელებდეს საკუთარ კრიტიკულ სექციაში შესვლას შესაძლებელია მას დამგეგმავმა ჩამოართვას პროცესორი და გადასცა სხვა პროცესს. ახალი პროცესიც შეამოწმებს ცვლადი-ბოქლომის მნიშვნელობას და ნახავს, რომ ის 0-ის ტოლია, ანუ არცერთი პროცესი არ იმყოფება საკუთარ კრიტიკულ სექციაში. პროცესი შედის საკუთარ კრიტიკულ სექციაში. თუ ახალი პროცესის საკუთარი კრიტიკული სექციიდან გამოსვლამდე პროცესორი დაუბრუნდა პირველ პროცესს, მაშინ რადგანაც ის შესრულებას აგრძელებს შეჩერებული ადგილიდან და შეჩერების დროს ცვლადი-ბოქლომის მნიშვნელობა იყო 0, ის არ ახდენს ამ მნიშვნელობის გადამოწმებას, შედის საკუთარ კრიტიკულ სექციაში და ცვლადი-ბოქლომის მნიშვნელობას ზრდის 1-ით. ამრიგად მივიღეთ, რომ ორი პროცესი იმყოფება საკუთარ კრიტიკულ სექციაში.

შეიძლება ვიფიქროთ, რომ პრობლემის გადაწყვეტა მარტივია, თუ პროცესს მოვთხოვთ საკუთარ კრიტიკულ სექციაში შესვლამდე ხელმეორედ განახორციელოს ცვლადი-ბოქლომის მნიშვნელობის შემოწმება. მაგრამ პრობლემის გადაწყვეტისათვის ასეთი მოთხოვნაც არაა საკმარისი. პრობლემა წარმოიშობა იმ შემთხვევაში, როდესაც სხვა პროცესმა შეცვალა ცვლადი-ბოქლომის მნიშვნელობა პირველი პროცესის მიერ ცვლადი-ბოქლომის მნიშვნელობის შემოწმების პარალელურად (დასრულებისთანავე).

მკაცრი მიმდევრობა

კრიტიკული სექციის პრობლემის გადაწყვეტის შემდეგი მეთოდი გულისხმობს პროცესების წარმოდგენას მკაცრი მიმდევრობის სახით. პროცესები მიმდევრობით შედიან საკუთარ კრიტიკულ სექციაში. იქიდან გამოსვლის შემდეგ მიმდევრობის შემდეგ წევრს ეძლევა საკუთარ კრიტიკულ სექციაში შესვლის შესაძლებლობა. ამ მეთოდის საილუსტრაციოდ განვიხილოთ შემდეგი პრაქტიკული ხასიათის მაგალითი. ვთქვათ, მოცემული ორი პროცესი: P_0 და P_1 . ნახ. 5.3-ით წარმოდგენილი ფრაგმენტები განსაზღვრავენ ამ პროცესების მიერ კრიტიკულ სექციაში შესვლის მომენტს.

```

while (TRUE) {
    while (turn != 0) ; // ცარიელი ციკლი
    critical_region();
    turn = 1; // P: პროცესს ეძლევა კრიტიკულ
        // სექციაში შესვლის შესაძლებლობა
    noncritical_region();
}

```

```

while (TRUE) {
    while (turn != 1) ; // ცარიელი ციკლი
    critical_region();
    turn = 0; // P: პროცესს ეძლევა კრიტიკულ
        // სექციაში შესვლის შესაძლებლობა
    noncritical_region();
}

```

ა)

ნახ. 5.3. კრიტიკული სექციის პრობლემის პროგრამული გადაწყვეტა;
P: პროცესი (ა); P₁: პროცესი (ბ)

ბ)

turn ცვლადი განსაზღვრას, თუ რომელი პროცესი უნდა შევიდეს საკუთარ კრიტიკულ სექციაში. საკუთარ კრიტიკულ სექციაში შესვლის წინ ორივე პროცესი ამოწმებს turn ცვლადის მნიშვნელობას. რადგანაც turn = 0, პირველი (P₀) პროცესი შედის საკუთარ კრიტიკულ სექციაში, ხოლო P₁ პროცესი კი ასრულებს ცარიელ ციკლს. P₀ პროცესი საკუთარი კრიტიკული სექციიდან გამოსვლის შემდეგ ზრდის turn ცვლადის მნიშვნელობას 1-ით. რადგანაც უკვე turn = 1, P₁ პროცესი გამოდის ცარიელი ციკლიდან და შედის საკუთარ კრიტიკულ სექციაში. იქიდან გამოსვლის შემდეგ turn ცვლადის მნიშვნელობას ხდის 0-ის ტოლად, რითაც P₀ პროცესს საჭიროების შემთხვევაში ეძლევა საკუთარ კრიტიკულ სექციაში შესვლის შესაძლებლობა.

ამ მეთოდის ნაკლოვანება მდგომარეობს იმაში, რომ მიმდევრობაში მყოფი პროცესები ციკლურად შედიან საკუთარ კრიტიკულ სექციაში და, თუ მიმდევრობის ერთი რომელიმე წევრი არ საჭიროებს საკუთარ კრიტიკულ სექციაში ხშირ შესვლას, ხოლო სხვა რომელიმე წევრი კი პირიქით, მაშინ აღმოჩნდება, რომ ამ უკანასკნელს (რომელიც ხშირ შესვლას საჭიროებს) მოუწევს დიდი ხნით ლოდინი სანამ ციკლი დატრიალდება, რაც ეწინააღმდეგება ზემოთ ჩამოყალიბებულ ერთერთ პირობას (ურთიერთქმედი პროცესები არ უნდა ახდენდნენ ერთიმეორის ბლოკირებას).

აქტივაცია და დეაქტივაცია

პიტერსონის ალგორითმი ქმედუნარიანია, მაგრამ მას მაინც გააჩნია ნაკლოვანება. პრობლემა მდგომარეობს შემდეგში: პროცესი საკუთარ კრიტიკულ სექციაში შესვლის საჭიროების შემთხვევაში ამოწმებს ამ მოქმედების განხორციელების შესაძლებლობას. თუ მას არ შეუძლია კრიტიკულ სექციაში შესვლა ის ასრულებს ცარიელ ციკლს და ელოდება კრიტიკულ სექციაში შესვლას. ამ მომენტს არამხოლოდ მივყართ პროცესორული დროის უსარგებლოდ ხარჯვამდე, არამედ შეიძლება გამოიწვიოს გაურკვეველი შედეგიც. განვიხილოთ ორი პროცესი: H (მაღალპრიორიტეტული) და L (დაბალპრიორიტეტული). მათი მუშაობის დაგეგმვის წესის თანახმად მზადყოფნის მდგომარეობაში მყოფი პროცესების მიმდევრობაში H პროცესი გამოჩენისთანავე დაიკავებს პროცესორს. დავუშვათ დროის მიმდინარე მომენტში, როდესაც L პროცესი იმყოფება საკუთარ კრიტიკულ სექციაში სისტემაში წარმოიქმნა H პროცესი. მაღალი პრიორიტეტის გამო H პროცესი დაიწყებს შესრულებას. თუ აღმოჩნდა, რომ შესრულების მომენტში H გადავიდა აქტიური ლოდინის მდგომარეობაში, მაშინ H-ის მაღალი პრიორიტეტის გამო პირველი უნდა დასრულდეს ის. L ვერასოდეს ვერ გამოვა საკუთარი კრიტიკული სექციიდან, ამიტომ H პროცესს მოუწევს უსასრულო ციკლში ყოფნა.

იმ მიზნით, რომ პროცესების ურთიერთქმედებისას არ მოხდეს პროცესორული დროის უსარგებლოდ ხარჯვა, განვიხილოთ გარკვეული პრიმიტივები, რომლებიც ახდენენ სამუშაოს ბლოკირებას სანამ მათთვის არ იქნება ნებადართული კრიტიკულ სექციაში შესვლა. ასეთი პრიმიტივების უმარტივეს ნაკრებს წარმოადგენს sleep და wakeup. sleep სისტემური გამოძახება ანხორციელებს მისი შემსრულებელი პროცესის ბლოკირებას, რომელიც იქნება შეჩერებული სანამ არ მოხდება მისი გააქტიურება სხვა პროცესის მიერ. გააქტიურების wakeup სისტემური გამოძახება ერთ არგუმენტად იყენებს გასააქტიურებელი პროცესის სახელს. sleep და wakeup სისტემური გამოძახებები ერთმანეთთან დასაკავშირებლად დამატებით პარამეტრად იყენებენ მეხსიერების სივრცის ნაწილს.

ურთიერთბლოკირება

თანამედროვე კომპიუტერი აღჭურვილია მრავალი რესურსით (პროცესორი, მყარი დისკი, მეხსიერება, ფაილები და ა.შ.). სისტემაში წარმოქმნილი ყოველი პროცესი მიმართავს მას იმ რესურსის გამოყოფის მოთხოვნით, რომელსაც პროცესი საჭიროებს საკუთარი სამუშაოს შესასრულებლად. დროის ნებისმიერ მომენტში კონკრეტული რესურსი შეიძლება გამოყოფს მხოლოდ ერთ პროცესს. ხშირად რამდენიმე პროცესი მიმართავს ოპერაციულ სისტემას ერთიდაიმავე რესურსის გამოყოფაზე. პროცესებისათვის საერთო რესურსის გამოყოფა შესაძლებელია დაკავშირებული იყოს გარკვეულ პრობლემებთან, ამიტომ ოპერაციული სისტემა საკუთარ რესურსებს მცირე დროითი შუალედით უნაწილებს პროცესებს მათზე დაშვების სრული უფლებებით.

ხშირად პროცესები საკუთარი სამუშაოს შესასრულებლად საჭიროებენ ერთზე მეტ რესურსს. მაგალითად, განვიხილოთ სიტუაცია, რომლის დროსაც სისტემაში გვაქვს ორი პროცესი, P_1 , P_2 , და ორი რესურსი: სკანერი და დისკური მოწყობოლიბა. დავუშვათ, რომ ორივე პროცესი საჭიროებს თითოეულ რესურსს და რესურსების გამოყოფა ხორციელდება პროცესებისათვის თითოთითოდ. დავუთვათ, P_1 პროცესი დაპროგრამირებულია ისე, რომ მან პირველი გამოიყენოს სკანერი და მეორე დისკური მოწყობილობა, ხოლო P_2 პროცესი კი პირიქით. როგორც წესი, პირველი P_1 პროცესს გამოეყოფა სკანერი, ხოლო P_2 -ს დისკური მოწყობილობა. P_1 პროცესი სკანერთან მუშაობის დასრულების და მისგან მონაცემების მიღების შემდეგ საჭიროებს მეორე რესურსს და აკეთებს შესაბამის მოთხოვნას რესურსის გამოყოფაზე. მაგრამ მისი დაკამაყოფილება შეუძლებელია, ვინაიდან დისკურ მოწყობილობას იკავებს P_2 პროცესი. ანალოგურად, P_2 პროცესი დისკურ მოწყობილობასთან მუშაობის დასრულების შემდეგ გააკეთებს მოთხოვნას სკანერზე და მისი დაკამაყოფილებაც ასევე შეუძლებელია. ასეთ შემთხვევაში ორივე პროცესი იქნება ბლოკირებული სანამ არ გამონთავისუფლდება მათ მიერ მოთხოვნილი რესურსი. წარმოქმნილ სიტუაციას ეწოდება ურთიერთბლოკირება (deadlock) ან ჩიხი.

განაწილებადი და გაუნაწილებელი რესურსი

ოპერაციულ სისტემაში წარმოდგენილი ყოველი რესურსი შეიძლება იყოს ორი სახის: განაწილებადი ან გაუნაწილებელი. პროცესის მიერ დაკავებულ რესურსს, რომლის ჩამორთმევაც მისთვის შესაძლებელია „უმტკივნეულოდ“ (შესრულებული მნიშვნელოვანი სამუშაოს დაკარგვის გარეშე), მიეკუთვნება განაწილებად რესურსს. ამ ტიპის რესურსის მაგალითს წარმოადგენს მეხსიერება. სისტემაში არასაკმარის მეხსიერების არსებობის შემთხვევაში ხორციელდება პროცესის სრული ან ნაწილობრივი გადატანა მეხსიერების არიდან დისკზე, ხოლო მოგვიანებით კი მისი უკან დაბრუნება. პროცესისათვის გაუნაწილებელი რესურსის ჩამორთმევამ შესაძლებელია გამოიწვიოს პროცესის მიერ შესრულებული მნიშვნელოვანი სამუშაოს დაკარგვა. მაგალითად, პროცესისათვის, რომელიც

დისკურსი მოწყობილობის გამოყენებით ანხორციელებს კომპაქტ- დისკზე მონაცემების ჩაწერას მის დასრულებამდე დისკურსი მოწყობილობის ჩამორთმევა დაუშვებელია. რესურსის ჩამორთმევის შემთხვევაში ინფორმაციის ჩაწერა ვერ განხორციელდება და კომპაქტ-დისკიც გახდება უსარგებლო ხელმეორედ გამოყენებისათვის. ურთიერთბლოკირებაში შესაძლებელია მონაწილეობდეს როგორც განაწილებადი, ისე გაუნაწილებელი რესურსი. რესურსების საგულდაგულო გადანაწილების ხარჯზე შესაძლებელია განაწილებადი რესურსებით გამოწვეული ურთიერთბლოკირების აცილება. ამიტომ ყურადღებას გავამახვილებთ მხოლოდ გაუნაწილებელი რესურსებით გამოწვეულ

ურთიერთბლოკირებასთან გამკვლავების მეთოდებზე.

საზოგადოდ, სისტემაში რესურსების გამოყენება ხორციელდება მოვლენათა შემდეგი მიმდევრობით:

1. მოთხოვნა;
2. გამოყენება;
3. გამონთავისუფლება

თუ პროცესის მიერ რესურსზე გაკეთებული მოთხოვნის მომენტში ის მიუწვდომელია (დაკავებულია სხვა პროცესის მიერ), მაშინ პროცესს იძულებით უწევს ლოდინის მდგომარეობაში გადასვლა.

ზოგიერთ ოპერაციულ სისტემაში მსგავს შემთხვევებში პროცესი გადადის ბლოკირების მდგომარეობაში და როგორც კი რესურსი გახდება ხელმისაწვდომი ის გამოდის იქიდან და იღებს მას. სხვა სისტემებში რესურსის არ გამოყოფა თანდართულია შეცდომის კოდით და შემდგომი მოქმედებების განხორციელება, გააკეთოს რესურსზე მოთხოვნა ხელმეორედ, თუ დაელოდოს მას, ეკისრება რესურსის მომთხოვნ პროცესს.

ურთიერთბლოკირების წარმოქმნის აუცილებელი და საკმარისი პირობები

ურთიერთბლოკირების წარმოქმნისათვის აუცილებელია და საკმარისი შემდეგი 4 პირობის შესრულება:

1. **ურთიერთგამორიცხვის პირობა (mutual exclusion).** დროის ნებისმიერ მომენტში რესურსი ან გამოყოფილია მხოლოდ ერთი პროცესისათვის ან თავისუფალია;
2. **რესურსის ლოდინის პირობა (hold and wait).** პროცესს, რომელსაც დაკავებული აქვს გარკვეული რესურსი შეუძლია მოითხოვოს ახალი რესურსი;
3. **გაუნაწილებლობის პირობა (no preemption).** პროცესისათვის მის მიერ დაკავებული რესურსის იძულებითი ჩამორთმევა შეუძლებელია. პროცესმა დაკავებული რესურსი უნდა გამოანთავისუფლოს საკუთარი სურვილით;
4. **ციკლური ლოდინის პირობა (circular wait).** უნდა არსებობდეს ორი ან მეტი პროცესისაგან შემდგარი წრიული მიმდევრობა, რომელშიც მიმდევრობის ყოველი წევრი ელოდება მის შემდეგ მდგომი წევრის მიერ დაკავებული რესურსის გამონთავისუფლებას.

ჩამოყალიბებული პირობებიდან ერთერთის დარღვევის შემთხვევაში სისტემაში არ გვაქვს ურთიერთბლოკირების მდგომარეობა.

უნდა აღინიშნოს, რომ ჩამოყალიბებული პირობები არ არღვევს იმ პოლიტიკას, რომელსაც სისტემა ან ემხრობა ან არა.

3000777

```

* [1, 1000], ხოლო მეორე პირი რაც შეიცვლა გარემოების მიმართ უკავშირდება
* 10 thread-ი, რომელთაგან თითოეული სხვადასხვა სტრიქისზე დაიფლოს. 1-ის ჯერადა რიცხვების
* საშუალო არითმეტიკული. მეორე პროცესის ფაილში უნდა დამკერთოს სხვა რიცხვებს მართს მინიმუმური.
*
*/
#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>
#include <cstdlib>
#include <errno.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <ctime>
#include <iostream>
using namespace std;
const int N = 10;

class myMemory {
    /* განსაზღვრეთ თქვენი შეხედულებით შესაბამისი რაოდენობის კლასის ატრიბუტები*/
public:
    myMemory(/* თქვენი შეხედულებით განსაზღვრეთ პარამეტრების რაოდენობა */)
        /* class constructor */
    }

    void fill(/* თქვენი შეხედულებით განსაზღვრეთ პარამეტრების რაოდენობა */)) {
        // ფუნქციის უნდა უზრუნველყოს გარანტილებადი მეხსიერების შევსება
        // წინასწარ დასხელებული შეულედიდან
    }

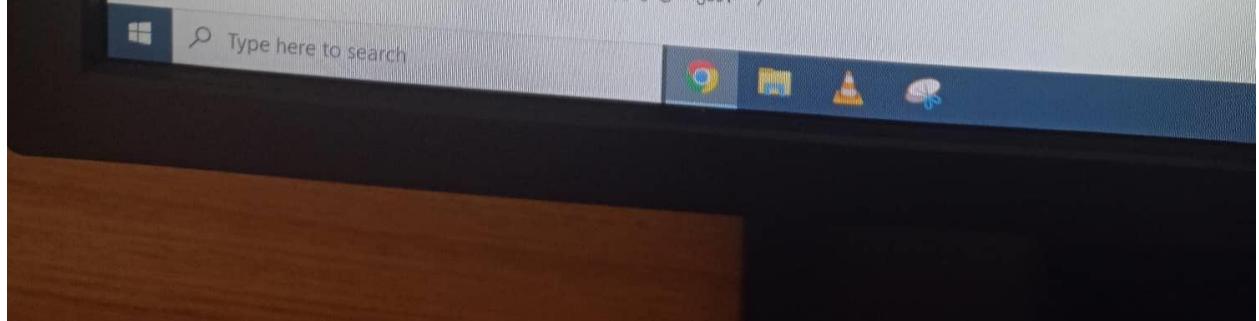
    friend ostream& operator<<(ostream& out, const myMemory obj) {
        /* მოახდინეთ ოპერატორის გადატვირთვა */
        return out;
    }

    friend ostream& operator<<(ostream& out, const myMemory obj) {
        /* მოახდინეთ ოპერატორის გადატვირთვა */
        return out;
    }

    ~myMemory(){
        /* Class destructor */
    }
};

int main() {
    srand(time(0));
    char name[] = "FinalExam.cpp";
    /* მოახდინეთ ამოცანის შესაბამისი ლოგიკის იმპლემენტირება. */
}

```



```

192.168.60.50/moodle/pluginfile.php/15/question/questiontext/1388/8/1020/FinalExam1%20%28GE9

/* სუფასება: 10 ქულა
 *
 * სუკრემით განაწილებადი მეხსიერების საკუთარი კლასი და მას განსაზღვრეთ ატრიბუტები (თქვენი
 * შეხედულებით) და მეთოდები:
 *
 * 1. კლასის მარამეტრიანი კონსტრუქტორი, რომელმაც გადაცემული პარამეტრების მიხედვით უნდა
 * უზრუნველყოს შესაბამისი განწილებადი მეხსიერების შექმნა შესაბამისი შემოწმებითა და
 * არსებული განაწილებადი მეხსიერების გამოყენების შესაძლებლობით
 *
 * 2. განაწილებადი მეხსიერების შექსების მეთოდი, რომელმაც უნდა უზრუნველყოს მითითებული
 * შეალებითან აღებული შემთხვევით მისმამდებით განაწილებადი მეხსიერების შექვება
 *
 * 3. გადატვირთეთ მონაცემების კონსოლური გამოტენის ოპერატორი, რომელიც დაბეჭდავს განაწილებად
 * მეხსიერებაში განთავსებულ მისმამდებებს.
 *
 * 4. გადატვირთეთ მონაცემების კონსოლური გამოტენის ოპერატორი, რომელიც დაბეჭდავს განაწილებად
 * მეხსიერებაში განთავსებულ მისმამდებებს.
 *
 * 5. გაშვაზღვრეთ კლასის დესტრუქტორი.
 *
 * დაწერეთ პროგრამა, რომელმაც მძიმელ პროცესს ეყოლება თუ შეიძლება (P1 და P2).
 * მძიმელმა პროცესს უნდა შექმნას ერთობლებითან განწილებადი მეხსიერება, რომელსაც ჩა
 * გამოიყენება მის განსაზღვრავად. თუ რომელი შეიძლება დაიწყება დაწინობას.
 * კორმოდ, თუ მძიმელმა მოაძინონ შემთხვევითი პრიოტიტით განაწილებად მეხსიერებაში 1 -ის
 * ჩაწერ საქმიანობას დაიწყებას P1 პროცესი, ხოლო თუ ჩაწერა 2, მაშინ საქმიანობას დაწინობას
 * მეორე პროცესი.
 *
 * P1 და P2 პროცესებს შორის უნდა იყოს გამოყენებული N (=100 (10x10)) ელემენტიანი მეორე
 * განაწილებადი მეხსიერება, რომლის შექმნაც თანაბრად შეეძლებათ ორივე პროცესს (ანუ პარალელი რომელი
 * უნდა იქნას დამუშავებული, როგორც 10x10 კვადრატული მატრიცა.
 *
 * განაწილებადი მეხსიერების შევსება აუცილებლად უნდა განახორციელოს პირველმა პროცესმა შეუალედიდან
 * [1, 1000], ხოლო მეორე პროცესს კი მს შერე რაც შეივსება განაწილებადი მეხსიერება შექმნას
 * 10 thread-ი, რომელთანა თითოეული სხვადასხვა სტრიქონზე დაითვლის 3-ის ჯერადი რიცხვების
 * საშუალო არითმეტიკულს. მეორე პროცესმა ფაილში უნდა დაბეჭდოს სკეთ რიცხვებს შორის მიშიმალური.
 */

#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>
#include <cstdlib>
#include <errno.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <ctime>
#include <iostream>
using namespace std;
const int N = 10;

class myMemory {
    /* გაშვაზღვრეთ თქვენი შეხედულებით შესაბამისი რაოდენობის კლასის ატრიბუტები*/
public:
    myMemory(/* თქვენი შეხედულებით გაშვაზღვრეთ პარამეტრების რაოდენობა */)
        /* class constructor */
}

```

```

#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>

```

```
#include <cstdlib>
#include <errno.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <ctime>
#include <iostream>
using namespace std;
const int N = 10;

class myMemory {
public:
    myMemory(int key) {
        int shmid = shmget(key, sizeof(int), IPC_CREAT | 0666);
        if (shmid == -1) {
            cerr << "Shared memory allocation failed" << endl;
            exit(1);
        }

        memory = static_cast<int*>(shmat(shmid, nullptr, 0));
        if (memory == reinterpret_cast<void*>(-1)) {
            cerr << "Shared memory attachment failed" << endl;
            exit(1);
        }
    }

    ~myMemory() {
        int status = shmdt(memory);
        if (status == -1) {
            cerr << "Shared memory detachment failed" << endl;
            exit(1);
        }
    }
}
```

```

        }

    }

void fill(int lowerBound, int upperBound) {
    srand(time(nullptr));
    *memory = rand() % (upperBound - lowerBound + 1) + lowerBound;
}

friend ostream& operator<<(ostream& out, const myMemory& obj) {
    out << "Value in allocated memory: " << *(obj.memory);
    return out;
}

private:
    int* memory;
};

void* calculateMean(void* arg) {
    int* row = static_cast<int*>(arg);
    int sum = 0;
    int count = 0;

    for (int i = 0; i < N; ++i) {
        if (row[i] % 3 == 0) {
            sum += row[i];
            count++;
        }
    }

    double mean = static_cast<double>(sum) / count;
}

```

```
ofstream outputFile("mean_output.txt", ios::app);
outputFile << "Mean: " << mean << endl;
outputFile.close();

pthread_exit(nullptr);
}

int main() {
    srand(time(nullptr));
    int key = rand(); // Generate a random key for shared memory

    myMemory allocMem(key);

    pid_t pid;
    pid = fork();

    if (pid == 0) {
        // Child process P2
        int* matrix;

        int shmid = shmget(key, N * N * sizeof(int), IPC_CREAT | 0666);
        if (shmid == -1) {
            cerr << "Shared memory allocation failed" << endl;
            exit(1);
        }

        matrix = static_cast<int*>(shmat(shmid, nullptr, 0));
        if (matrix == reinterpret_cast<void*>(-1)) {
            cerr << "Shared memory attachment failed" << endl;
        }
    }
}
```

```
    exit(1);

}

for (int i = 0; i < N * N; ++i) {
    matrix[i] = rand() % 100 + 1;
}

pthread_t threads[N];

for (int i = 0; i < N; ++i) {
    pthread_create(&threads[i], nullptr, calculateMean, matrix + i * N);
}

for (int i = 0; i < N; ++i) {
    pthread_join(threads[i], nullptr);
}

shmdt(matrix);
shmctl(shmid, IPC_RMID, nullptr);
} else if (pid > 0) {
    // Parent process P1
    wait(nullptr);

    int lowerBound = 1;
    int upperBound = 1000;
    allocMem.fill(lowerBound, upperBound);

    cout << allocMem << endl;
} else {
    cerr << "Fork failed" << endl;
```

```
    return 1;  
}
```

```
return 0;  
}
```

ნინა კოდი 2022 წლისაა

კოდი 2023

```
401 - PC10
192.168.68.9/moodle/pluginfile.php
Not secure | 192.168.68.9/moodle/pluginfile.php/5990/question/questiontext/5880/1/5496/task1.cpp

/*
 * შეფასება: 10 ტულა
 *
 * მქონენით განაწილებადი მებმიერების სკოტარი კლასი და მას განუსაზღვრეთ ატრიბუტები (თქვენი
 * შეზღუდვებით) და მეოთხები:
 *
 * 1. კლასის პარამეტრები კონსტრუქტორი, რომელსაც გადაცემული პარამეტრების მიხედვით უნდა
 * უზრუნველყოს მესამამის განაწილებადი მესამეების მეტენა შესაბამისი მემორიუმებითა და
 * არაესულ განაწილებადი მესამეების გამოყენების მესამელისტით
 *
 * 2. განაწილებადი მებმიერების მევების მეთოდი, რომელსაც წინასწარ განსაზღვრული მასივის
 * გამოყენებით უნდა უზრუნველყოს განაწილებადი მესამეების მევები
 *
 * 3. გადატენებით მონაცემების კომსიულური გამორჩევის იყენაზე, რომელიც დაძლდას განაწილებად
 * მებმიერებამი განათავსებულ მომენტების შემდეგ.
 *
 * 4. როგორიც დათვების და დასტანციას განაწილებად მებმიერებამი გამოყენებული მიმდევარი
 * გარემონტი თვისების მქონე ელემენტების საშუალო არითმეტიკულს. (პირისა დაზუსტებულია ქვემოთ)
 *
 * 5. განასაზღვრეთ კლასის დესტრუქტორი.
 *
 * // 1 ტულა =
 *
 * დწერეთ thread -ის ფუნქცია, რომელიც გასინა მესამეების ელემენტების დათვების და შედეგის
 * სათა დამტკიცებას სტრიქნინი ა სიმორდს წინ ერთს გამოტივებით რამდენჯერ გაზიდება ხ სიმორდი
 * (მაგალითად, hxa სამცული ამჟამოდიებებს პირობის, ხოლო ხა არა).
 * thread -მ ანუმენტად უდა მიღება საშუალო (მასივი, სტრიქნინის წამერი, სეეტის წომერი), რისთვისაც
 * დაკირდებათ ახარი ტანის (ლასის ან სტრიქნის მექმა).
 */
*
* ვთქვათ მოყენებულია 2 განაწილებისანი მასივი (array[4][4]), რომელიც შეიცემს სხვადასხვა ფაილიდან
* წაკონტალ სტრიქნინის სიგრძით 1000 სიმბოლო.
*
* დაწერეთ პროგრამა, რომელიც პროცესი (main ფუნქცია) მექმას მასივის ელემენტების რაოდენობის ტული
* რაოდენობით thread -ის. ყველაზე მაღალია უდა უზრუნველყოს მესამეების მდგრადი დალექტისთვის
* thread -ის უზრუნველყოს. უზრუნველყოფით thread -ების შემთხვევა მასივზე სტრიქნიზირებული წედომა.
* მიღება მეტენას გამოყენების უდა მიღება პროგრამის შემმომის განაწილებაზე მებმიერების შესვენება.
* მიღება დამტკიცების განაწილებადი მებმიერებამი ანურილი მიმდევარი განაწილებაზე და [5, 10] შეულებელი მოცული
* მიმდევარის საშუალო არითმეტიკული.
```

401 - PC10

```

192.168.68.9/moodle/pluginfile.php/5990/question/questiontext/S880/1/5496/task1.cpp
Not secure | 192.168.68.9/moodle/pluginfile.php/5990/question/questiontext/S880/1/5496/task1.cpp

// 0.5 ქულა =
bool predict(int x) {
    // ფუნქცია აღნებს მისთვის ბადაცემულ არჩეულებს გააჩნია თუ არა გარკვეული თვისება
    // ფუნქცია აღნებს მისთვის ბადაცემულ არჩეულებს გააჩნია თუ არა გარკვეული თვისება
    // ასდაკეთულ ფუნქცია ამოცანის პირობის შესაბამისად
    return true;
}

class myMemory {
    // პირტკლდი ტიპის გამოყენებით განსაზღვრეთ თქვენი შეხედულებით შესაბამისი
    // რაოდენობის კლასის პრინციპები
public:
    // 1 ქულა =
    myMemory() /* იძებნი შეხედულებით განსაზღვრეთ პარამეტრების რაოდენობა */
    /* class constructor */

};

// 0.5 ქულა =
void fill/* თქვენი შეხედულებით განსაზღვრეთ პარამეტრების რაოდენობა */ {
    // ფუნქციამ უნდა უზრუნველყოს განაწილებად მემორიების შევსება
    // წინასწარ განსაზღვრული მასივის გამოყენებით
}

// 0.5 ქულა =
friend ostream& operator<<(ostream& out, const myMemory obj) {
    /* მოახდინეთ ისერატორის გადატვირთვა */
    return out;
}

// 1 ქულა =
float getAver(/*აჩვენების რაოდენობა-გარსაზღვრეთ თქვენი შეხედულებით*/) {
    // ფუნქცია ერთადერთ (თუ ერთ-აგუმენტანია) ან ერთერთ არგუმენტად იღებს
    // ლექციას მინდერს (ეს ხდება ფორმით getAver(ret_type (*func)(int));
    // ასეუცემულ გადაცემული ფუნქციის გამოყენებით ამორჩებს განაწილებად
    // მემორიების გამოყენებულ მიმშენელობას აქს თუ არა შესაბამისი თვისება
    // დადგენით გასუბის მემორიების უნდა იყოს გათვალისწინებული.
    // სერთ ფუნქციის გამოძახება უნდა მოხდეს ფორმით getAver(&func)
    return 0;
}

// 0.5 ქულა =
~myMemory() {
    /* Class destructor */
};

int main() {
    char name[] = "FinalExam.cpp";
    /* მოახდინეთ ამოცანის შესაბამისი ლინკის იმპლემენტირება. */
}

```

/* შეფასება: 10 ქულა

*

* შექმენით განაწილებადი მეხსიერების საკუთარი კლასი და მას განუსაზღვრეთ ატრიბუტები (თქვენი

* შეხედულებით) და მეთოდები:

*

* 1. კლასის პარამეტრიანი კონსტრუქტორი, რომელმაც გადაცემული პარამეტრების მიხედვით უნდა

* უზრუნველყოს შესაბამისი განაწილებადი მეხსიერების შექმნა შესაბამისი შემოწმებებითა და

* არსებული განაწილებადი მეხსიერების გამოყენების შესაძლებლობით

*

* 2. განაწილებადი მეხსიერების შევსების მეთოდი, რომელმაც წინასწარ განსაზღვრული მასივის

* გამოყენებით უნდა უზრუნველყოს განაწილებადი მეხსიერების შევსება

*

* 3. გადატვირთეთ მონაცემების კონსოლური გამოტანის ოპერატორი, რომელიც დაბეჭდავს განაწილებად

* მეხსიერებაში განთავსებულ მნიშვნელობებს.

*

* 4. რომელიც დაითვლის და დაბეჭდავს განაწილებად მეხსიერებაში გამოყენებული მნიშვნელობებიდან

* გარკვეული თვისების მქონე ელემენტების საშუალო არითმეტიკულს. (პირობა დაზუსტებულია ქვემოთ)

*

* 5. განსაზღვრეთ კლასის დესტრუქტორი.

*

* // 1 ქულა =

* დაწერეთ thread -ის ფუნქცია, რომელიც მასივის შესაბამისი ელემენტისთვის დაითვალის და შედეგის

* სახით დააბრუნებს სტრიქონში a სომბოლოს წინ ერთის გამოტოვებით რამდენჯერ გვხვდება b სიმბოლო

* (მაგალითად, bxa სამეული აკმაყოფილებს პირობას, ხოლო xba არა).

* thread -მა არგუმენტად უნდა მიიღოს სამეული (მასივი, სტრიქონის ნომერი, სვეტის ნომერი), რისთვისაც

* დაგჭირდებათ ახალი ტიპის (კლასის ან სტრუქტურის შექმნა).

*

*

- * ვთქვათ მოცემულია 2 განზომილებიანი მასივი (array[4][4]), რომელიც შეიცავს სხვადასხვა ფაილიდან
 - * წაკითხულ სტრიქონებს სიგრძით 1000 სიმბოლო.
 - *
 - * დაწერეთ პროგრამა, რომელშიც პროცესი (main ფუნქცია) შექმნის მასივის ელემენტების რაოდენობის ტოლი
 - * რაოდენობით thread -ებს. ყოველმა მათგანმა უნდა უზრუნველყოს შესაბამის ინდექსზე მდგომი ელემენტისთვის
 - * thread -ის ფუნქციის ამუშავება. უზრუნველყოფით thread -ების მხრიდან მასივზე სინქრონიზირებული წვდომა.
 - * მიღებული შედეგების გამოყენებით უნდა მოხდეს პროგრამაში შექმნილი განაწილებადი მეხსიერების შევსება.
 - * ბოლოში დაბეჭდეთ განაწილებადი მეხსიერებაში ჩაწერილი მნიშვნელობები და [5, 10] შეაღედში მოქცეული
 - * მნიშვნელობების საშუალო არითმეტიკული.
 - */

```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>
#include <cstdlib>
#include <errno.h>
#include <fcntl.h>
#include <algorithm>
#include <sys/wait.h>
#include <sys/types.h>
#include <pthread.h>
#include <iostream>
#include <time.h>
#include <stdio.h>
#include <ctime>
```

```
using namespace std;
```

```
const int N = 10;

// 0.5 ქულა =
bool predcat(string x) {
    for(int i{}; i < x.length() - 1; i++)
        if(x[i] == 'b' && x[i+1] == 'a')
```

```

        return true;
    }

template<typename T>
class myMemory {
    size_t* size, SHM_ID;

    key_t* KEY;
public:
    T * memory;
    // 1 ဂျွဲး =
    myMemory(string name, int UNIC_ID, size_t size) {
        this->size = size;
        this->KEY = ftok(name, UNIC_ID);
        if(this->KEY == -1)
            exit(-1);

        this->SHM_ID = shmget(this->KEY, sizeof(char) * size, IPC_CREAT | IPC_EXCL | 0700);

        if(SHM_ID == -1)
        {
            if(errno == EEXIST)
                this->SHM_ID = shmget(KEY, sizeof(char) * size, 0);
        }
        else
            exit(-1);
    }

    // 0.5 ဂျွဲး =
    void fill(char* arr, size_t n) {
        if(this->size() != n)
            return;

        for(int i = 0; i < this->size(); i++)
            memory[i] = arr[i];
    }

    // 0.5 ဂျွဲး =
friend ostream& operator<<(ostream& out, const myMemory obj) {
    for(int i = 0; i < obj.size(); i++)
        out << obj->memory[i];
}

```

```

        return out;
    }

// 1 גג��ה =
float getAver(void * func, myMemory * OBJ) {
    int c{}, sum{};
    for(int i = 0; i < OBJ->size(); i++)
    {
        if(func(*OBJ->memory))
        {
            for(int j = 0; < obj->size(); i++)
            if(OBJ->memory[j] >= 5 && OBJ->memory[j] <= 10)
            {
                sum += OBJ->memory[j];
                c++;
            }
        }
    }
    return sum / c;
}

// 0.5 גג��ה =
~myMemory(){
    shmdt(&memory);
}
};

const int M = 4;

pthread_mutex_t mt;

struct threadSTRCT
{
    string * arr;
    size_t A_ind, B_ind;

    threadSTRCT(string * AR, size_t A, size_t B) : arr(AR), A_ind(A), B_ind(B) {}

void* threadFuction(void * strctOBJ)
{
    threadSTRCT * thisObj = (threadSTRCT *) strctOBJ;

```

```

size_t A = thisObj->A_ind, B = thisObj->B_ind;

string currentSTR = thisObj->arr[A][B];

int c_if = count_if(currentSTR.begin(), currentSTR.end(), predcat);
pthread_exit((void *) c_if);
}

int main() {
    char name[] = "main.cpp";

        /* მოახდინეთ ამოცანის შესაბამისი ლოგიკის
იმპლემენტირება. */

    string name_to_str = name;
    srand(time(NULL));

    myMemory * main_obj = new myMemory(name_to_str, 1, N);
    main_obj->attach();

    switch(fork())
    {
        case -1:
            exit(-1);
        case 0: // CHILD
        {
            myMemory * child_obj = new myMemory(name_to_str, 1, N);
            child_obj->attach();
            delete child_obj; break;
        }
        default: // MAIN
        {
            string array[M][M];

            pthread_t th[M*M];
            for(int i = 0; i < M; i++)
            {
                for(int j = 0; j < M; j++)
                {
                    pthread_mutex_lock(&mt);
                    threadSTRCT tmp(*array ,i, j);
                    if(pthread_create(&th[i], nullptr, &threadFuction, (void *)
&tmp) != 0)

```

```
    exit(-1);
    pthread_mutex_unlock(&mt);
}
}

for(int i{}; i < M*M; i++)
{
    int this_index_count(0);
    if(pthread_join(th[i] , (void **) &this_index_count) != 0)
        exit(-1);
    main_obj->memory[i] = this_index_count;
}

cout << main_obj << endl;
delete main_obj;
}

}
```