სემინარი 1. UNIX ოპრაციული სისტემა

სასწავლო კურსის მსვლელობისას სემინარული მეცადინეობის მასალა ილუსტრირებული იქნება UNIX ოპერაციული სისტემის ერთერთი ნაირსახეობის (Linux) მაგალითზე, თუმცა ჩვენი საუბარი არ შეეხება მხოლოდ ამ სისტემის 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 პროგრამებისთვის 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 მიუწვდომელი იყო.

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 გამოყენებისგან, 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 პროგრამისტისთვის წააგავს C ენის ფუნქციას.

სისტემურ გამოძახება შეიძლება დასრულდეს წარმატებით ან წარუმატებლად. წარუმატებლად დასრულების აღსანშნავად უმეტეს სისტემურ გამოძახებაში გამოიყენება მნიშვნელობა -1, ხოლო წარმატებული დასრულების აღსანიშნავად კი რაიმე არაუარყოფითი მნიშვნელობა. სისტემური გამოძახებები, რომლებიც აბრუნებენ მიმთითებელს, შეცდომის შემცველი სიტუაციის იდენტიფიცირებისთვის, 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 იქნას სტანდარტული UNIX-ფუნქცია perror().

perror() ფუნქციის პროტოტიპი

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

ფუნქციის აღწერა

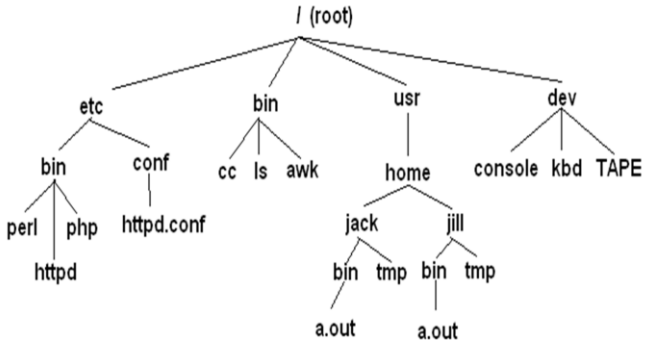
perror() ფუნქცია გამოიყენება შეცდომაზე შეტყობინების გამოსატანად, რომელიც შეესაბამება გამოტანის სტანდარტული ნაკადის შეცდომის დაფიქსირებისას errno სისტემური ცვლადის მნიშვნელობას. ფუნქცია ბეჭდავს მთლიანად str სტრიქონს (თუ str პარამეტრი არ უდრის NULL-ს), ორწერტილს, ჰარის და წარმოშობილი შეცდომის შესაბამისი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

1.2. ფაილის სრული და მიმართებითი სახელები

ყველა ოპერაციული სისტემისთვის ფაილის ცნება წარმოადგენს ერთერთ მნიშვნელოვან ცნებას. UNIX-მსგავს ოპერაციულ სისტემებში ფაილის ცნების მნიშვნელობა უფრო გამოჩანს 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 დავკმაყოფილდეთ ფაილზე ზოგადი ცნებით: ფაილი ეს არის მყარ დისკზე განთავსებული ერთ მონაცემებთან დაკავშირებული მეხსიერების მისამართების ერთობლიობა (ნაკრები).

ისევე, როგორც სხვა ოპერაციულ სისტემებში, UNIX-მსგავს ოპერაციულ სისტემაში ყოველი ფაილი გაერთიანებულია ხისებრ ლოგიკურ სტრუქტურაში. ფაილები შეიძლება გაერთიანდნენ დირექტორიებში ან კატალოგებში[1]. არ არსებობს ფაილი, რომელიც არ შედის არცერთ დირექტორიაში. დირექტორიები თავის მხრივ შეიძლება წარმოადგენდნენ სხვა დირექტორიის ქვედირექტორიებს. არსებობს დირექტორია, რომელშიც არ შედის არც ფაილი და არც ქვედირექტორია (ცარიელი დირექტორია) (ნახ. 1.1). ყველა დირექტორიას შორის არსებობს ერთადერთ 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, მომხმარებლის მონაცემებს და ა.შ.) და დირექტორია.

ნახ. 1.1. ფაილური სისტემის სტრუქტურის მაგალითი



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

- სახელის სიგრძე არ უნდა აღემატებოდეს სისტემაში გათვალისწინებულ სიგრძეს (Linux-სთვის - 255 სიმბოლო);
- არ შეიძლება NULL სიმბოლოს გამოყენება;
- არ შეიძლება '/' სიმბოლოს გამოყენება[2].

აკრძალული სიმბოლოების რიცხვს ასევე შეიძლება მივაკუთვნოთ სიმბოლოები: '*', '?', '\"', '\'' და '\\'.

[1] Windows-ის ოპერაციულ სისტემაში გამოიყენება ცნება ფოლდერი

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

სისტემაში არსებულ ფაილებს უნდა გააჩნდეთ სახელი, რომელიც უნიკალური იქნება 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 ფაილისთვის ამოვწეროთ ყველა იმ დირექტორიის სახელი, რომლებიც მდებარობენ ფაილური სისტემის ლოგიკურ ხეზე საბაზო დირექტორიიდან დაწყებული ამ ფაილამდე (/ 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 შემდეგ პირველი დირექტორიის სახელისა, ერთმანეთისგან გამოვყოთ სიმბოლოთი ‘/’ („/usr/home/jack/bin/a.out“). მიღებული ჩანაწერი ცალსახად განსაზღვრავს ფაილის ადგილმდებარეობას 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 პროგრამისთვის, ბრძანებათა ინტერპრეტატორის[3] ჩათვლით, რომელიც ასრულებს შეტანილ ბრძანებებს და გამოაქვს ახალი ბრძანების შეტანის ველი, ფაილური სისტემის ლოგიკურ ხეზე გამოიყოფა ერთერთი დირექტორია, რომელიც ინიშნება მოცემული პროგრამისთვის მიმდინარე ანუ სამუშაო დირექტორიად. იმის გაგება, თუ რომელი დირექტორია წარმოადგენს ბრძანებათა ინტერპრეტატორისთვის მიმდინარე დირექტორიას შესაძლებელია pwd ბრძანებით.

pwd ბრძანების სინტაქსისი

pwd

ბრძანების აღწერა

pwd ბრძანებას გამოაქვს მიმდინარე დირექტორიის სახელი მუშა ბრძანებათა ინტერპრეტატორისთვის.

მიმდინარე დირექტორიის ცოდნით შესაძლებელია ფაილური სისტემის ხეზე მისგან დანიშნულების ფაილამდე გზის გადება. კვანძების (დირექტორიების) მიმდევრობა, რომლებიც შეგვხვდება ამ გზაზე ჩავწეროთ შემდეგი სახით: კვანძი, რომელიც შეესაბამება მიმდინარე დირექტორიას ამ ჩანაწერში არ შევა; საბაზო კატალოგის მიმართულებით 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“ ფაილისთვის მიმართებითი მისამართი იქნება „../../usr/home/jack/bin/a.out“[4], ხოლო, თუ მიმდინარე დირექტორია არის „/usr/home/jack“, მაშინ მისი მიმართებითი სახელი იქნება „bin/a.out “.

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

[4] შევნიშნოთ, რომ მოცემულ შემთხვევაში, მოცემული /usr/home/jack/bin/a.out ფაილისთვის მიმართებითი სახელის მისაღებად აუცილებლობას არ წარმოადგენ მიმართებით სახელში ფაილამდე სრული სახელი გამოვიყენოთ (/../../usr/home/jack/bin/a.out). რადგანაც jill და jack დირექტორიებისთვის home წარმოადგენს „მშობელ დირექტორიას“, ამიტომ მიმართებით სახელის მისაღებად საჭიროა ერთი დონით ასვლა და სასურველ დირექტორიამდე გზის გადება, ანუ /usr/home/jack/bin/a.out ფაილისთვის მიმართებითი სახელი იქნება ../jack/bin/a.out

1.2.1. მომხმარებლის საშინაო დირექტორია

სისტემაში დარეგისტრირებული ყოველი მომხმარებლისთვის იქმნება სპეციალური დირექტორია (Desktop), 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 სისტემისთვის ასეთი საცნობარო მასალის შემცველი არის უტილიტი help[5]. UNIX-მსგავს ოპერაციული სისტემებში გამოიყენება უტილიტი man. მისი გამოყენებით შესაძლებელია ამომწურავი ინფორმაციის მიღება სასწავლო კურსის მსვლელობისას 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 (შესაძლებლობების თვალსაზრისით). ამ ბრძანებებზე ამომწურავი ინფორმაციის მიღება შესაძლებელია UNIX Manual-ით. UNIX Manual-ში ინფორმაციის დიდი ნაწილი ხელმისაწვდომია ინტერაქტიული ფორმით man უტილიტის გამოყენებით.

man უტილიტის გამოყენება ადვილია, მისი სინტაქსისია

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 გვერდისგან შედგება, მაშინ გვერდების გადასაფურცლად გამოიყენება ღილაკი <space>, წინა გვერდზე დასაბრუნებლად - კომბინაცია <ctrl> + <b>, 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 განსხვავებული კონტექსტით. სასურველ ბრძანებაზე ამომწურავი ინფორმაციის მიღების მცდელობისას შეიძლება მიღებული იქნას ინფორმაცია, რომელიც პასუხობს საძიებო ობიექტს და, ასევე, ინფორმაცია, რომელიც კონტექსტიდან გამომდინარე სცდება ინტერესის სფეროს. დიდი მოცულობის ინფორმაციაში ძებნის ოპერაციის გაადვილების მიზნით მთელი ინფორმაცია დაყოფილია ჯგუფებად. ჯგუფებში ინფორმაცია გაერთიანებულია ბრძანებების, დანიშნულების და ფუნქციების მიხედვით, მაგალითად, სისტემური გამოძახებები, ბიბლიოთეკური ფუნქციები, ბირთვის ბრძანებები და ა.შ.

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

[5] Windows 7-დან დაწყებული დამატებით შეიძლება გამოყენებული იქნას უტილიტი get-help

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).

საჭირო ბრძანებაზე ამომწურავი ინფორმაციის მისაღებად დამატებით სასურველია იმის ცოდნა თუ რომელ ჯგუფს მიეკუთვნება შესაბამისი ბრძანება. იმისთვის, რომ გავარკვიოთ როგორაა დაყოფილი მთელი ინფორმაცია ჯგუფად საჭირო man ბრძანება გამოვიყენოთ ფორმით: man man, ხოლო იმისთვის, რომ საჭირო ბრძანებაზე მივიღოთ ამომწურავი ინფორმაცია შესაბამისი ჯგუფიდან man ბრძანება უნდა 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 ინტერპრეტატორთან მუშაობისას შესაძლებელია მოგვიწიოს სხვადასხვა ბრძანებების შესრულება სამუშაო დირექტორიისგან განსხვავებულ საბაზო დირექტორიის სხვა მხარეს განთავსებულ დირექტორიაში. ამ შემთხვევაში ბრძანებების შესრულება მოითხოვს ყოველ ბრძანებაში მითითებული იყოს საჭირო დირექტორიის სრული ან მიმართებითი სახელი. ბრძანების ამ ფორმით გამოყენება ართულებს მას. ბრძანების გამოყენების გამარტივება შესაძლებელია თუ შესაბამის დირექტორიას გადავაქცევთ სამუშაო დირექტორიად. ამ შემთხვევაში საკმარისი იქნება ბრძანებაში მითითებული იყოს შესაბამისი ფაილის მხოლოდ სახელი, რაც ამარტივებს ბრძანების ჩაწერას.

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

cd DIRNAME

სადაც DIRNAME [6] არის იმ დირექტორიის სრული ან მიმართებითი სახელი, რომელიც უნდა იქცეს სამუშაო დირექტორიად.

ხშირად ჩვენ ვსაჭიროებთ ვიცოდეთ თუ რა ფაილებია განთავსებული ამა თუ იმ

[6] შევნიშნოთ, რომ სადაც შეგვხვდება სიტყვა DIRNAME, მის ქვეშ ვიგულისხმებთ დირექტორიის სრულ ან მიმართებით სახელს

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 სინტაქსისია

ls DIRNAME

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

ტერმინალში შესრულებულ ls ბრძანებას, ოფციისა და ფაილის სახელის მითითების გარეშე, გამოაქვს სამუშაო დირექტორიაში განთავსებული ხილული ფაილების ჩამონათვალი. -a ოფციით ბრძანებას გამოაქვს მითითებულ ფაილში არსებული ხილული და დამალული[7] 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 გაუხსნელად. ამ მიზნით გამოიყენება ბრძაბება cat, მისი სინტაქსისია

cat FILENAME [8]

სადაც FILENAME ტექსტური მონაცემების შემცველი ფაილია[9]. cat ბრძანებით შესაძლებელია ერთზე მეტი ფაილის შიგთავსის გამოტანა. ამ შემთხვევაში საჭიროა cat ბრძანებას მივუწეროთ ფაილის სახელები იმ მიმდევრობით თუ როგორი მიმდევრობით გვინდა ისინი გამოჩნდნენ ტერმინალში.

გარდა ფაილის შიგთავსის დათვალიერებისა cat ბრძანებით ასევე შესაძლებელია ახალი ფაილის შექმნა და მისი რედაქტირების რეჟიმში გადასვლა ან რაიმე ფაილებში არსებული ტექსტური მონაცემების ერთ ფაილში ჩაწერა. 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 მასში არსებულ მონაცემებზე გადაეწერება კავიატურიდან შეტანილი მნიშვნელობები. 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 ეკრანზე 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

---

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

[8] შევნიშნოთ, რომ სადაც შეგვხვდება სიტყვა FILENAME, მის ქვეშ ვიგულისხმებთ ფაილის სრულ ან მიმართებით მისამართს

[9] თუ დასათვალიერებელი ფაილი დიდი მოცულობისაა, მაშინ ტერმინალში გამოტანილი იქნება მისი ბოლო გვერდი. ასეთ ფაილებთან სამუშაოდ გამოიყენება ბრძანება more

ls -al > FILENAME.

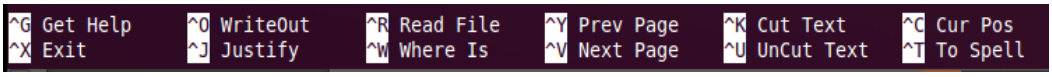
შევნიშნოთ, რომ თუ ფაილი FILENAME ბრძანების შესრულების მომენტში არ არსებობდა, მაშინ ის ავტომატურად შეიქმნება. თუ ფაილი არსებობდა, მაშინ იქ არსებულ მონაცემებს გადაეწერება ბრძანების შესრულების შედეგი.

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 მართვის 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 დირექტორიის შესაქმნელად გამოიყენება ბრძანება mkdir (make directory), მისი სინტაქსისია

mkdir DIRNAME.

mkdir ბრძანების მეშვეობით ასევე შესაძლებელია რამდენიმე დირექტორიის ერთდროული შექმნა. ამასთან, შესაძლებელია რამდენიმე ჩალაგებული დირექტორიის ანუ იერარქიის შექმნა, რაც ნიშნავს შემდეგს: თუ სამუშაო დირექტორიაში (/home/student) არ გვაქვს არცერთი დირექტორია და გვინდა შევქმნათ მაგალითად, 3 დონით ჩალაგებული დირექტორია (DIR1/DIR2/DIR3) 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 დიერექტორიების) სხვა დირექტორიაში რეკურსიული[10] გადაკოპირებისთვის. მისი სინტაქსისია

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

სადაც SDIRNAME (source directory) საწყისი დირექტორიაა, რომლიდანაც უნდა მოხდეს ფაილების გადაკოპირება, DESTINNAME (destination directory) დანიშნულების დირექტორია, სადაც უნდა მოხდეს მონაცემების გადაკოპირება, -r ოფცია გამოიყენება რეკურსიული გადაკოპირებისთვის.

[10] რეკურსიული კოპირება ნიშნავს დირექტორიის გადაკოპირებას მასში არსებული ქვედირექტორიებითა და ფაილებით

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

mv SOURCENAME DFILENAME
mv FILENAME1 FILENAME2 ... FILENAMEN DESTINNAME

თუ პირველ ბრძანებაში ორივე მისამართი უთითებს ერთიდაიმავე დირექტორიას, მაშინ მოხდება SOURCENAME ფაილისთვის სახელის გადარქმევა და ახალი სახელი იქნება DFILENAME. სხვა შემთხვევაში მოხდება ფაილის ერთი დირექტორიიდან მეორეში გადაადგილება. მეორე ბრძანებით ხდება FILENAME1 FILENAME2 ... FILENAMEN ფაილების გადაადგილება DESTINNAME 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

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 სხვადასხვა 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. მეტასიმბოლოების მხარდაჭერა, რომლებიც ძებნის ოპერაციის გამარტივებას გამოიწვევენ. ასეთი მეტასიმბოლოები შეიძლება გამოყენებულიქნას არამხოლოდ ძებნის მოქმედების განხორციელებისას, არამედ ფაილების მიმართ სხვადასხვა ბრძანებების გამოყენების დროსაც. ხშირად ამ მეტასიმბოლოების გამოყენებით ბრძანებებთან გამოსაყენებელ სიმბოლოთა კომბინაციას **შაბლონს** უწოდებენ. მეტასიმბოლოები, რომლებიც ამარტივებენ ამა თუ იმ ბრძანების გამოყენებას მოცემულია შემდეგ ცხრილში

* -	შეესაბამება ყველა სიმბოლოს ჰარის (space) ჩათვლით;
? -	შეესაბამება მხოლოდ ერთ სიმბოლოს;
[...] -	შეესაბამება კვადრატულ ფრჩხილებში მითითებულ ნებისმიერ სიმბოლოს ან ალფავიტის ასოს, რომლებიც ერთიმეორისგან 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 შაბლონებისთვის 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[11] (user identifier).

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. ოპერაციული ჯგუფების ერთიმეორისგან განსხვავების მიზნით იყენებს ჯგუფის იდენტიფიკატორს GID[12] (group identifier).

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 მიმდინარე 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 მომხმარებლისთვის მნიშვნელოვანი ფაილების დაცვა არასანქცირებული დაშვებისგან. ამ მიზნით UNIX-მსგავს სისტემებში განასხვავებენ დაშვების სამ უფლებას:

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

მონაცემების შემცველი ფაილებისთვის ამ უფლებების არსი ემთხვევა მათ შინაარსს. დირექტორიებისთვის ის რამდენადმე განსხვავებულია. დირექტორიისთვის კითხვის უფლება ნიშნავს ამ დირექტორიაში არსებული ფაილების სახელების კითხვას. ვინაიდან დირექტორიისათვის უფლება „შესრულება“ ყოველგვარ აზრს მოკლებულია (ისევე როგორც არაშესრულებადი რეგულარული ფაილისთვის), მისთვის შესრულების უფლება იცვლის მნიშვნელობას:

[11] გარკვეული რიცხვითი მნიშვნელობა, რომელიც უნიკალურია სისტემის მასშტაბით

[12] გარკვეული რიცხვითი მნიშვნელობა, რომელიც უნიკალურია სისტემის მასშტაბით

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 დირექტორიისთვის საჭიროა იმისთვის, რომ მისი გადაქცევა შესაძლებელი იყოს მიმდინარე დირექტორიად. ეს უფლება საჭიროა 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 დირექტორიისთვის 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 მომხმარებლებისთვის 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 მფლობელისთვის დაშვების უფლებების მომართვა;

g - ფაილის მფლობელის ჯგუფში შემავალი მომხმარებლებისთვის დაშვების უფლებების მომართვა;

o - ყველა დანარჩენი მომხმარებლებისთვის დაშვების უფლებების მომართვა.

ოპერაცია, რომელიც სრულდება დაშვების უფლებებზე მომხმარებელთა მითითებული კატეგორიისთვის, 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 სიმბოლოების ან ერთერთი მათგანის 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 პროგრამისთვის, 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-სთვის 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 შემქმნელი მომხმარებლისთვის კითხვის უფლება;

0200 – ფაილის შემქმნელი მომხმარებლისთვის ჩაწერის უფლება;

0100 – ფაილის შემქმნელი მომხმარებლისთვის შესრულების უფლება;

0040 – ფაილის შემქმნელი მომხმარებლის ჯგუფისთვის კითხვის უფლება;

0020 – ფაილის შემქმნელი მომხმარებლის ჯგუფისთვის ჩაწერის უფლება;

0010 – ფაილის შემქმნელი მომხმარებლის ჯგუფისთვის შესრულების უფლება;

0004 – სხვა დანარჩენი მომხმარებლებისთვის კითხვის უფლება;

0002 – სხვა დანარჩენი მომხმარებლებისთვის ჩაწერის უფლება;

0001 – სხვა დანარჩენი მომხმარებლებისთვის შესრულების უფლება;

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

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

umask ბრძანების სინტაქსისი

umask [value]

ბრძანების აღწერა

umask ბრძანება განკუთვნილია 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

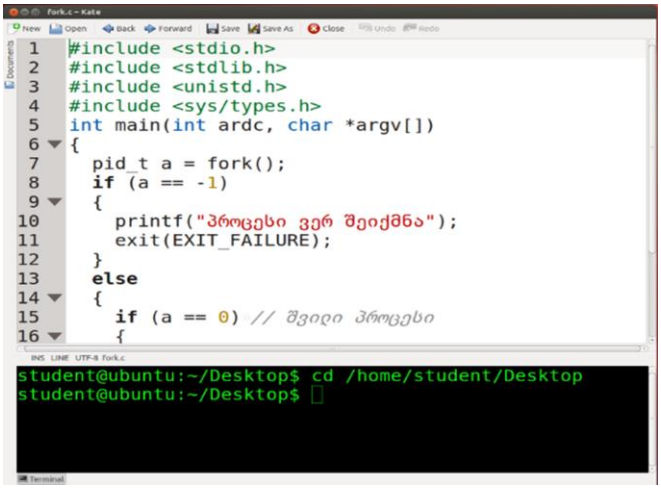
სასწავლო კურსის მსვლელობისას პროგრამული კოდს დავწერთ C++ ენაზე და შესაბამისად კოდისთვის გამოვიყენებთ cpp გაფართოების ფაილს. ჩვენს მიერ პროგრამული კოდი შეიძლება აკრეფილი იყოს უშუალოდ ტერმინალიდან nano ტექსტური რედაქტორის გამოყენებით ან ტექსტურ რედაქტორში kate (ნახ. 1.3), რომელშიც სამუშაო ველი იყოფა ორ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

UNIX-მსგავს სისტემებში პროგრამის კომპილიაციისთვის გამოიყენება ინტეგრირებული კომპილიატორები, როგორიცაა gcc, cc, g++ და ა.შ. სასწავლო კურსის მსვლელობისას ჩვენ გამოვიყენებთ g++ კომპილიატორს. 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 ფაილისთვის

ნახ. 1.3. kate პროგრამის სამომხმარებლო ინტერფეისი



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

g++ CODEFILE -o OUTNAME

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

კომპილირებული ფაილის შესრულება შესაძლებელია ფორმით

./OUTNAME