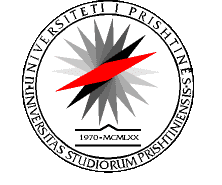
**UNIVERSITETI I PRISHTINËS “HASAN PRISHTINA”**

**FAKULTETI I INXHINIERISË ELEKTRIKE DHE KOMPJUTERIKE**

**DEPARTAMENTI: INXHINIERIA KOMPJUTERIKE – BACHELOR**



**Lënda**: Sisteme Operative

Projekti final

**Studentët**:                                                                          **Profesor**:

Arbena Musa       170714100093       Artan Mazrekaj

Butrint Beqiri       170714100046                              

Prishtinë, Janar 2020

Përmbajtja

[1. Përshkrimi i problemit 3](#_Toc29420429)

[2. Teknologjia 4](#_Toc29420430)

[3. Implementimi 5](#_Toc29420431)

[3.1 Deklarimi i variablave 13](#_Toc29420432)

[3.2 Struktura msgbuf 13](#_Toc29420433)

[3.3 Funksioni *funk(char \*s)* 13](#_Toc29420434)

[3.4 Funksioni notify() 14](#_Toc29420435)

[3.5 Funksioni sendmsg() 14](#_Toc29420436)

[3.7 Funksioni handle\_sigint() 14](#_Toc29420437)

[3.9 Funksioni main() 15](#_Toc29420438)

[3.10 Ekzekutimi I programit 15](#_Toc29420439)

[Referencat 17](#_Toc29420440)

# 1. Përshkrimi i problemit

Proceset në një sistem mund të jenë të pavarura ose kooperative.

Proceset e pavarura nuk mund të afektojnë proceset tjera dhe nuk afektohen nga ekzekutimi i tyre.

Proceset kooperative mund të afektojnë ose afektohen nga proceset e tjera, duke përfshirë edhe shkëmbimin e të dhënave. Prandaj ato kanë nevojë për **Interprocess Communication (IPC)**.

Metoda e komunikimit mes procesve (IPC) është mekanizëm që i’u mundëson proceseve komunikimin dhe sinkronizimin e veprimeve. Komunikimi mes këtyre proceseve mund të shihet si metodë e kooperimit (bashkëveprimit).

Proceset kanë mundësi të komunikojnë mes vete në këto dy forma: shared memory dhe **message passing**.

**Message passing** është mekanizëm që mundëson komunikimin dhe sinkronizimin e proceseve pa përdorur të njëjtën hapësirë të adresave memorike. Proceset komunikojnë mes vete duke shkëmbyer mesazhe. Madhësia e mesazheve mund të jetë fikse ose variabile. Proceset e dërgimit dhe pranimit të mesazheve mund të realizohen në kompjuterë të njëjtë ose të ndryshëm. Një arsye për popullaritetin e këtij mekanizmi është mbështetja që ofron për ndërveprimin klient-server.

Për të qenë i mundur shkëmbimi i mesazheve, një linjë e komunikimit duhet të ekzistojë mes procesve. Në *message passing*, për të implementuar logjikisht linjën e komunikimit dhe operacionet dërgo/prano mund të përdoret metoda **sinkrone** dhe asinkrone.

Sistemet që përdorin **synchronous message passing** kërkojnë që dërguesi dhe pranuesi të presin njëri-tjetrin deri sa mesazhi të transferohet. Kjo metodë e komunikimit ndryshe quhet edhe **blocked state**. Pra, bllokimi konsiderohet si sinkronizim. Dërguesi bllokohet deri sa mesazhi të pranohet kurse pranuesi bllokohet deri sa mesazhi të jetë i gatshëm.

Pasi asnjë konflikt nuk ka nevojë të shmanget gjatë shkëmbimit të sasive të vogla të të dhënave, *message passing* zgjedhet para formave tjera të komunikimit. Po ashtu mund të implementohet më thjeshtë.

Gjatë procesit të komunikimit, një proces mund të ndalet në formë jo të rregullt ose të ndodh një **event (ngjarje)** tjetër e caktuar. Në këto raste, procesi mund të dërgojë ose pranojë ndonjë **sinjal** për të treguar ose kuptuar se çfarë ka ndodhur. **Sinjalet** janë interrupte të gjeneruara nga softueri që dërgohen tek një proces kur ndodh ndonjë *event*. Ato poashtu mund të jenë sinkrone ose asinkrone. Sinjalet i dorëzohen një procesi kur sistemi detekton një event në softuer, si p.sh. kërkesa e shfrytëzuesit për *stop* ose *kill*. Sinjalet po ashtu mund të vijnë direkt nga *kerneli* i sistemit operativ kur ndodh ndonjë *event* në harduer, si p.sh. *bus error* ose hasja e ndonjë instruksioni të gabuar. Disa prej sinjaleve mund të injorohen, por për shumë të tjerë nëse nuk ndërmerret ndonjë veprim për t’i trajtuar, do t’i ndalojnë proceset ose do i’u japin fund.

Problemi që duhet të zgjidhet është implementimi i një programi përmes metodës së komunikimit mes proceseve (IPC) me *message passing* për gjendjen *blocked*, duke trajtuar rastet e “sjelleve” të procesve në raport me sinjalet.

# 2. Teknologjia

Gjatë zhvillimit të projektit është përdorur editori **Text Editor**, gjuha programuese **C** dhe platforma e sistemit operativ **Kali Linux**.

# 3. Implementimi

User1.c

|  |
| --- |
| #include <sys/types.h> #include <sys/ipc.h> #include <sys/msg.h> #include <stdio.h> #include <string.h> #include <stdlib.h> #include <signal.h>  #include <unistd.h>  #include <math.h>  /\* Header file që përmbanë deklarimet e funksioneve për konvertim  të numrave intexherë në stringje dhe anasjelltas. \*/ #include "parsing.h" #define MAXSIZE 128  char user1[], user2[]; int send\_counter = 0, rcv\_counter = 0;  // Id e radhës së mesazhit, id e procesit aktual, id e procesit me te cilin // komunikohet, statusi i procesit aktual. int msqid, pid1, pid2, status = 1;  int msgflg = IPC\_CREAT | 0666;  size\_t buflen; // Ruhet madhësia e buferit. key\_t key = 1234;  // Deklarimi i tipit të mesazhit. struct msgbuf {  long mtype;  char mtext[MAXSIZE]; };  // Deklarimi i buffer të mesazheve për dërgim dhe pranim. struct msgbuf sbuf; struct msgbuf rcvbuffer;  // Shfaq emrin e funksionit ku ka ndodhur gabimi. void funk(char \*s) {  perror(s);  exit(1); }  // Funksioni mundëson shfaqjen e njoftimit kur pranohet mesazhi. void notify() {  char command[100];  strcpy(command, "notify-send "); // Shfaq system notifications.  strcat(command, user2);  strcat(command, " \'");  strcat(command, rcvbuffer.mtext);  // Cakton kohën sa qëndron notification, cakton ikonën e notification.  strcat(command, "\' -t 5000 -i \'/root/Desktop/msg\_icon.jpg\'");   system(command); // Ekzekuton komandën në terminal. }  // Funksioni që mundëson pranimin e mesazheve. void rcvmsg() {  // Teston nëse ‘message queue id’ është negative.  if ((msqid = msgget(key, 0666)) < 0)  funk("msgget()"); //Thirrja e funksionit për të shfaqur errorin.  // Pranimi i përgjigjes nga mesazhi i tipit 1  if (msgrcv(msqid, &rcvbuffer, MAXSIZE, 1, 0) < 0)  funk("msgrcv");  // Në pranimin e parë, ruaj emrin e dërguesit.   if(rcv\_counter == 0)  strcpy(user2, rcvbuffer.mtext);  else if (rcv\_counter == 1) // Në pranimin e dytë, ruaj id e dërguesit.  pid2 = toInt(rcvbuffer.mtext);  else  {   sleep(1);  // Shfaq emrin e bashkëbiseduesit.  printf(user2);  // Shfaq mesazhin e marrë nga bashkëbiseduesi.  printf(": %s\n", rcvbuffer.mtext);  // Njoftimi me tingull për mesazhin e pranuar.  printf("\a");  // Shfaq system notification për mesazhin e pranuar.  notify();   }  rcv\_counter = rcv\_counter + 1;  status = 1;  sleep(1); }  void sendmsg() {  // Merr ID-në e radhës së mesazhit për çelësin e caktuar.  if ((msqid = msgget(key, msgflg)) < 0)  funk("msgget");  // Caktohet tipi i mesazhit.  sbuf.mtype = 1;  /\*  Kontrollon a është hera e parë që dërgohet mesazh  Nëse po, kërkon të regjistrojë emrin e përdoruesit aktual.  Inputin e marrë nga tastiera e ruan në sbuf.mtext dhe në user1.  \*/  if (send\_counter == 0)  {  printf("Sheno username-in tend: ");  scanf("%[^\n]", sbuf.mtext);  getchar();  strcpy(user1, sbuf.mtext);  }  /\*  Kontrollon a është hera e dytë që dërgohet mesazh.  Nëse po, vendos id e procesit aktual në sbuf.mtext.  \*/  else if(send\_counter == 1)  toString(sbuf.mtext, pid1);  /\*  Shfaq emrin e përdoruesit aktual.  Pranon inputin nga tastiera dhe e ruan në sbuf.mtext.  \*/  else  {  printf(user1);  printf(": ");  scanf("%[^\n]", sbuf.mtext);  getchar();   }    buflen = strlen(sbuf.mtext) + 1;  // Trego mesazhin që nuk u dërgua së bashku me arsyen.  if (msgsnd(msqid, &sbuf, buflen, IPC\_NOWAIT) < 0)  {  printf("%d, %d, %s, %d\n", msqid, sbuf.mtype, sbuf.mtext, buflen);  funk("msgsnd");  }  else  {  status = 0;  send\_counter = send\_counter + 1;  } }  // Trajtimi i sinjalit për interrupt. void handle\_sigint()  {   printf("\nPerfundim i bisedes.\n");  kill(pid2, SIGUSR1);  sleep(5);   exit(0); }  // Trajtimi i sinjalit të definuar nga shfrytëzuesi. void handle\_sigusr1() {  printf("Te fala nga ");  printf(user2);  printf(".\nBiseda mbaroi.\n");  sleep(5);  exit(0); }  int main() {  // Merr ‘id’ e procesit të shfrytëzuesit të parë.  pid1 = getpid();  // Shoqërimi i trajtimeve përkatëse për sinjalet.  signal(SIGINT, handle\_sigint);  signal(SIGUSR1, handle\_sigusr1);  // Pranimi dhe dërgimi i mesazheve.  while (1)  {  // Përderisa statusi është i ndryshëm nga 1, prano mesazhe.  while (status != 1)  {  rcvmsg();   }  sendmsg();  }  // Tregon përfundim të suksesshëm të programit.  exit(0); } |
|  |

User2.c

|  |
| --- |
| #include <sys/types.h> #include <sys/ipc.h> #include <sys/msg.h> #include <stdio.h> #include <string.h> #include <stdlib.h> #include <signal.h> #include <unistd.h>  #include <math.h> #include "parsing.h" #define MAXSIZE 128  char user1[], user2[]; int send\_counter = 0, rcv\_counter = 0; int msqid, pid1, pid2, status = 1; int msgflg = IPC\_CREAT | 0666; size\_t buflen; key\_t key = 1234;  struct msgbuf {  long mtype;  char mtext[MAXSIZE]; };  struct msgbuf sbuf; struct msgbuf rcvbuffer;  void funk(char \*s) {  perror(s);  exit(1); }  void notify() {  char command[100];  strcpy(command, "notify-send ");  strcat(command, user2);  strcat(command, " \'");  strcat(command, rcvbuffer.mtext);  strcat(command, "\' -t 5000 -i \'/root/Desktop/msg\_icon.jpg\'");  system(command); }  void rcvmsg() {   if ((msqid = msgget(key, 0666)) < 0)  funk("msgget()");  //Pranimi i pergjigjes nga mesazhi i tipit 1  if (msgrcv(msqid, &rcvbuffer, MAXSIZE, 1, 0) < 0)  funk("msgrcv");  if(rcv\_counter == 0)  strcpy(user2, rcvbuffer.mtext);  else if (rcv\_counter == 1)  pid2 = toInt(rcvbuffer.mtext);  else  {  sleep(1);  printf(user2);   printf(": %s\n", rcvbuffer.mtext);  printf("\a");   notify();  }  rcv\_counter = rcv\_counter + 1;  status = 0; }  void sendmsg() {  // Merr ID-ne e radhes se mesazhit per celesin e caktuar  if ((msqid = msgget(key, msgflg)) < 0)  funk("msgget");  // Tipi i mesazhit  sbuf.mtype = 1;  if (send\_counter == 0)  {  printf("Sheno username-in tend: ");  scanf("%[^\n]", sbuf.mtext);  getchar();  strcpy(user1, sbuf.mtext);  }  else if(send\_counter == 1)  toString(sbuf.mtext, pid1);   else  {  printf(user1);  printf(": ");  scanf("%[^\n]", sbuf.mtext);  getchar();   }   buflen = strlen(sbuf.mtext) + 1;  if (msgsnd(msqid, &sbuf, buflen, IPC\_NOWAIT) < 0)  {  printf("%d, %d, %s, %d\n", msqid, sbuf.mtype, sbuf.mtext, buflen);  funk("msgsnd");  }  else  {  status = 1;  send\_counter = send\_counter + 1;  } }  void handle\_sigint()  {   printf("\nPerfundim i bisedes.\n");  kill(pid2, SIGUSR1);  sleep(5);   exit(0); }   void handle\_sigusr1() {  printf("Te fala nga ");  printf(user2);  printf(".\nBiseda mbaroi.\n");  sleep(5);  exit(0); }  int main() {  pid1 = getpid();   signal(SIGINT, handle\_sigint);  signal(SIGUSR1, handle\_sigusr1);   while (1)  {  while (status != 0)  {  rcvmsg();  }  sendmsg();  }   exit(0); } |

Fajlli parsing.h përmban funksionet toString() dhe toInt(). Funksioni toString() pranon dy parametra, një char array dhe një numër. Puna e tij është të konvertojë numrin dhe ta ruajë në char array të dhënë. Funksioni toInt() pranon si parametër një char array të cilin e konverton në intexher.

Parsing.h

|  |
| --- |
| // Konverton një intexher në varg të karaktereve.  void toString(char str[], int num) {  int i, rem, len = 0, n;   n = num;  while (n != 0)  {  len++;  n /= 10;  }  for (i = 0; i < len; i++)  {  rem = num % 10;  num = num / 10;  str[len - (i + 1)] = rem + '0';  }  str[len] = '\0'; }  // Konverton një varg të karaktereve në intexher. int toInt(char str[]) {  int len = strlen(str);  int i, num = 0;   for (i = 0; i < len; i++)  {  num = num + ((str[len - (i + 1)] - '0') \* pow(10, i));  }   return num; } |

## 3.1 Deklarimi i variablave

Janë deklaruar dy vargje të karaktereve të emëruara *user1* dhe *user2* për të ruajtur emrin e shfrytëzuesve që do të zhvillojnë komunikimin.

Janë deklaruar dy intexherë *send\_counter* dhe *rcv\_counter* dhe janë inicializuar me zero. Këta dy numërues shërbejnë për të njoftuar procesin sa herë është dërguar përkatësisht pranuar një mesazh gjatë komunikimit.

Variabla *msqid* e tipit intexherruan id e një *message queue* që ekziston.

Variablat *pid1* dhe *pid2* të tipit intexherë regjistrojnë id e procesit aktual përkatësisht id e procesit me të cilin komunikohet.

Variabla *status* e tipit intexher përdoret për të kontrolluar procesin e pranimit/dërgimit të mesazheve mes procesve. Tek *user1.c*, vetëm nëse *status* ka vlerë të ndryshme prej 1 mund të pranohet mesazhi kurse tek *user2.c* vetëm nëse vlera është e ndryshme prej 0. Në rast se shfrytëzuesi nuk hyn në unazën për pranim të mesazhit, atëherë ai mund të dërgoj mesazh. Nga kjo kuptohet se shfrytëzuesi është i bllokuar dhe nuk mund të realizojë edhe pranimin edhe dërgimin në të njëtën kohë. Rrjedhimisht, variabla *status* bën që komunikimi të realizohet në formë të sinkronizuar.

Për të krijuar një *message queue* duhet gjithsesi të ketë vlera për një *key* bazik dhe *message* *flag*.

Secili mesazh identifikohet në mënyrë unike ashtu që proceset mund të zgjedhin saktë mesazhin e duhur. Por, procesi paraprakisht duhet të dijë vlerën e *key* të *message queue* për të fituar qasje në të. Kjo vlerë është ruajtur në variablën *key* të tipit *key\_t* që në esencë është intexher.

Variabla *msgflg* e tipit intexher është inicializuar me vlerën *IPC\_CREAT | 0666*. IPC\_CREAT e krijon një *message queue* të re nëse ajo nuk ekziston kurse 0666 është kodi në Linux që jep leje për qasje. Pas *bitwiseOR* mes tyre, gjenerohet vlera për *permission* dhe *control flag.*

## 3.2 Struktura msgbuf

Është definuar ky tip i të dhënave që përmbanë dy anëtarë: *mtype* dhe *mtext[MAXSIZE]*. Anëtari i parë është tipi i mesazhit të pranuar, i specifikuar në procesin e dërgimit. Anëtari i dytë është i tipit *char* dhe ruan mesazhin që pranon si varg të karaktereve.

## 3.3 Funksioni *funk(char \*s)*

Në rastet kur ndodhin gabime brenda një funksioni të caktuar, si argument *funk* i dërgohet si varg i karakaterve emri i funksionit ku ndodhi gabimi dhe ekzekutohen dy funksione me radhë, *perror(s)* dhe *exit(1)*.

Funksioni *void perror(const char \*str)* fillimisht e printon stringun *str* që i vie si parametër, ndjeket nga dy pika, një hapësirë dhe një mesazh përshkrues për errorin që ka ndodhur.

Funksioni *exit()* mund të ketë parametër 0 ose 1. *exit(0)* tregon për përfundim të suksesshëm të programit dhe pa gabime kurse *exit(1)* zakonisht tregon përfundim të pasuksesshëm të programit dhe me gabime. Pasi që printohet mesazhi për error ekzekutohet *exit(1)* për të ndaluar ekzekutimin e mëtejshëm të programit dhe për të treguar se ekzekutimi nuk ka qenë në rregull.

## 3.4 Funksioni notify()

Funksioni notify() ka për detyrë të iniciojë shfaqjen e njeftimeve për marrjen e mesazheve nga bashkëbiseduesi.

Brenda këtij funksioni është deklaruar command e cila është array e tipit char me gjatësi prej 100 karakteresh në të cilën ruhet commanda e cila duhet ekzekutuar në terminal. Përmes strcpy(command, <param>) në variablën command vendosen komandat përkatëse.

Komanda notify-send <Notification name> <Notification message> -t <Time in ms> -I <Image path> bën shfaqjen e system notifications me titull dhe mesazh specifik e cila qëndron si popup për një kohë të caktuar që matet me milisekonda dhe me ikon e cila mirret nga path I caktuar.

## 3.5 Funksioni sendmsg()

Fillimisht, merret ID e radhës së mesazhit për çelësin e caktuar dhe testohet nëse është vlerë negative. Nëse është vlerë negative, atëherë ka ndodhur ndonjë gabim prandaj dhe shfaqet gabimi dhe programi përfundon ekzekutimin në formë të pasuksesshme. Më pas caktohet tipi i mesazhit.

Programi kontrollon nëse është hera e parë që dërgohet mesazh. Nëse po, kërkon të regjistrojë emrin e përdoruesit aktual. Inputin e marrë nga tastiera e ruan në *sbuf.mtext* dhe në *user1*. Nëse është hera e dytë që dërgohet mesazh, vendoset *id* e procesit aktual në *sbuf.mtext*. Në herët tjera, shfaqet emri i përdoruesit aktual, pranohet inputi nga tastiera dhe ruhet në *sbuf.mtext*.

Nëse mesazhi nuk dërgohet, shfaqet mesazhi që nuk është dërguar së bashku me arsyen pse ka ndodhur ky veprim. Përndryshe, mesazhi është dërguar prandaj duhet bllokuar kjo gjendje e dërgimit dhe duhet pritur për përgjigjje duke ndërruar vlerën e statusit.

## 3.7 Funksioni handle\_sigint()

Ky funksion thirret në rastet kur shfaqet interrupti SIGINT që gjenerohet me kombinimin Crtl+C si input.

Përmes pritnf shfaqet mesazhi “Përfundimi i bisedës.”.

Thirret funksioni kill(<param1>,<param2>) që përdoret për dërgimin e interrupteve tek një proces tjetër. Ky funksion ka dy parametra, I pari është pid e procesit tek I cili do të dërgohet interrupti dhe I dyti është lloji I interruptit që do të dërgohet.

Përmes sleep(5) vendoset pauza prej 5 sekondave dhe pas kësaj procesi terminon përmes komandës exit(0).

## 3.9 Funksioni main()

Në vazhdinësi funksionet signal(<param1>, <param2>) presin për interrupte. Funksioni signal() I ka dy parametra, I pari është emri I sinjalit specifik, ndërsa parametri I dytë është një funksion I tipit void I cili nuk kthen rezultat. Ky funksion quhet signal handler për shkak se përcakton rrjedhën e programit pasi të ndodh interrupti.

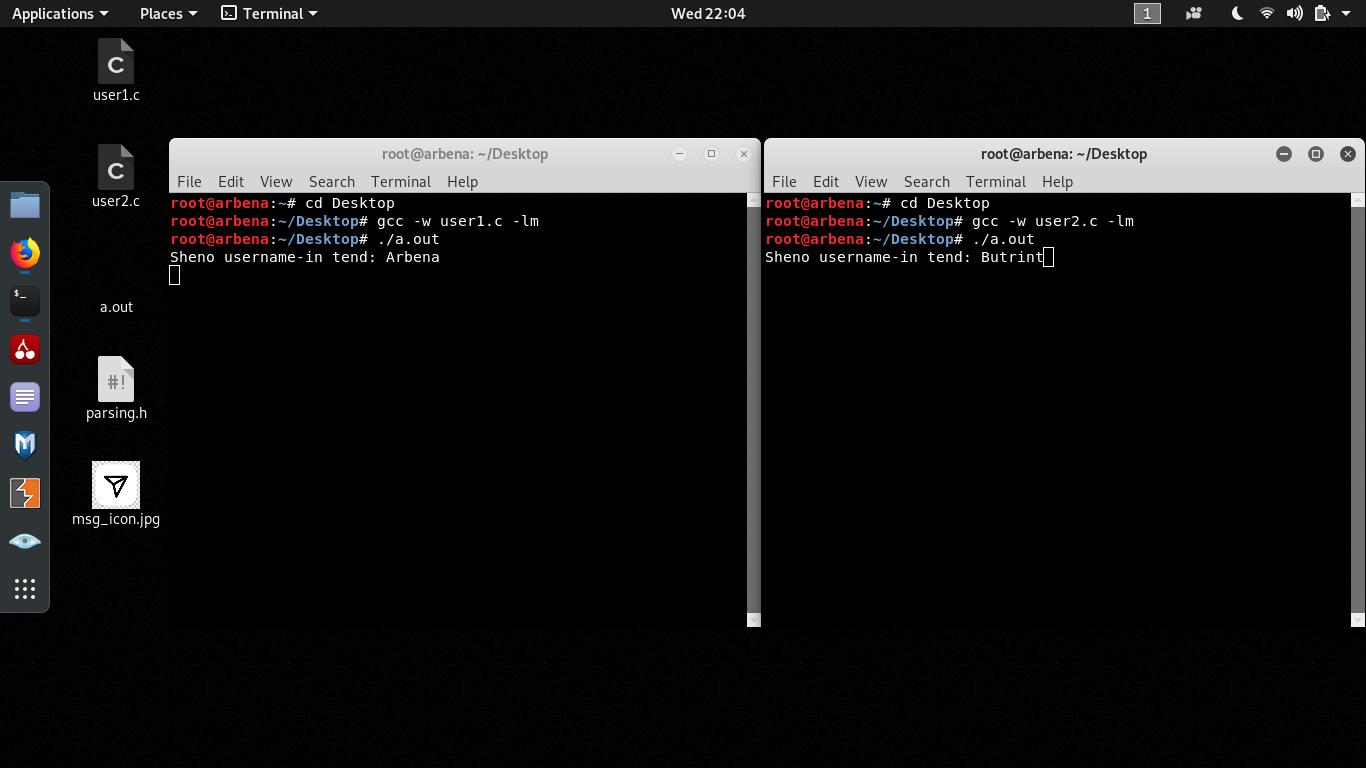
Përmes while true loop-ës programi është I gatshëm që të komunikojë gjatë gjithë kohës përderisa nuk ka ndërprerje për shkak të interrupteve.

Programi kontrollon statusin e përdoruesit. Kur statusi ë shtë I ndryshëm nga 1 thirret funksioni për pranimin e mesazheve ndërsa kur ndodh e kundërta vetëm vazhdohet me thirrjen e funksionit për dërgimin e mesazheve.

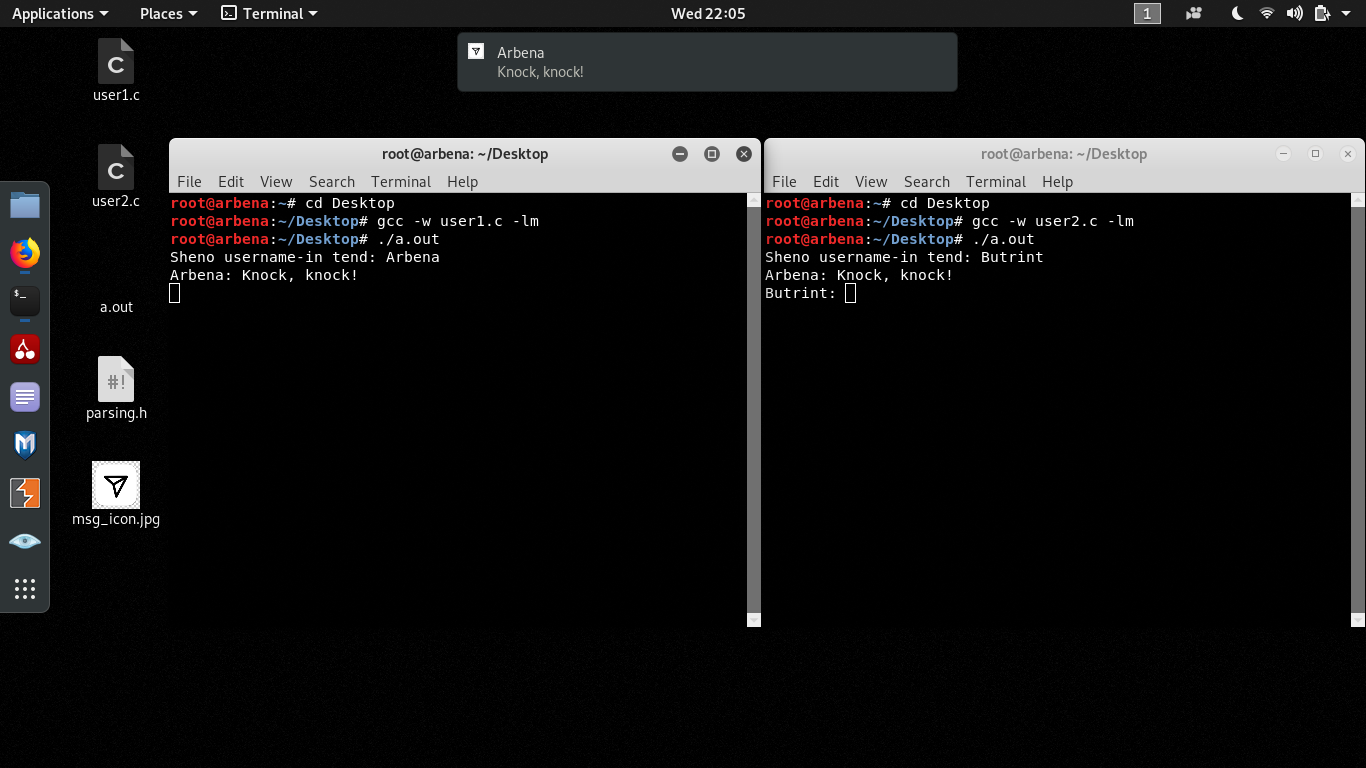
User1 është krijuar që të fillojë I pari dërgimin e mesazheve prandaj vlera dillestare e variablës së tij status është 1, ndërsa tek user2 ndodh e kundërta pasi është krijuar që si fillim të pranojë mesazhe.

## 3.10 Ekzekutimi I programit

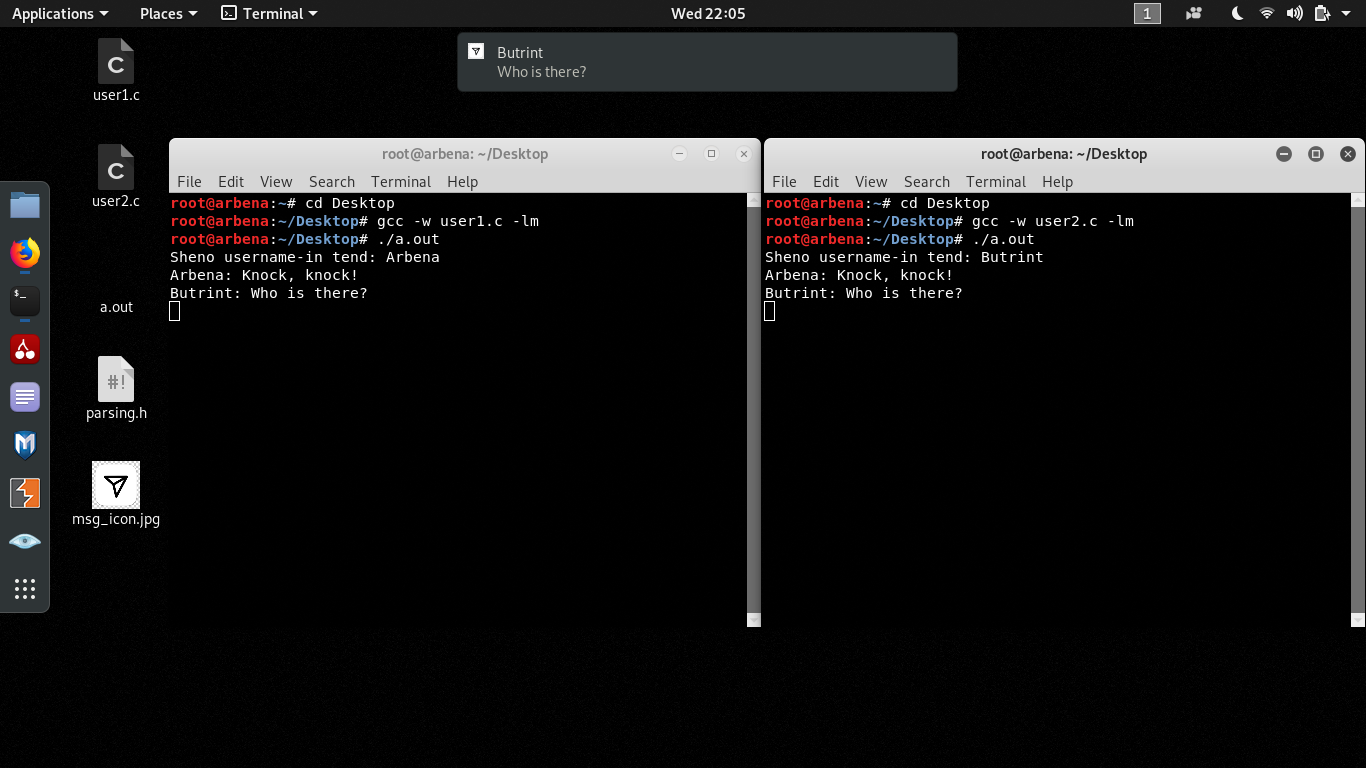
Programi në terminal kompajllohet përmes komandës *gcc –w userc –lm dhe ekzekutohet p*ërmes *./a.out.*



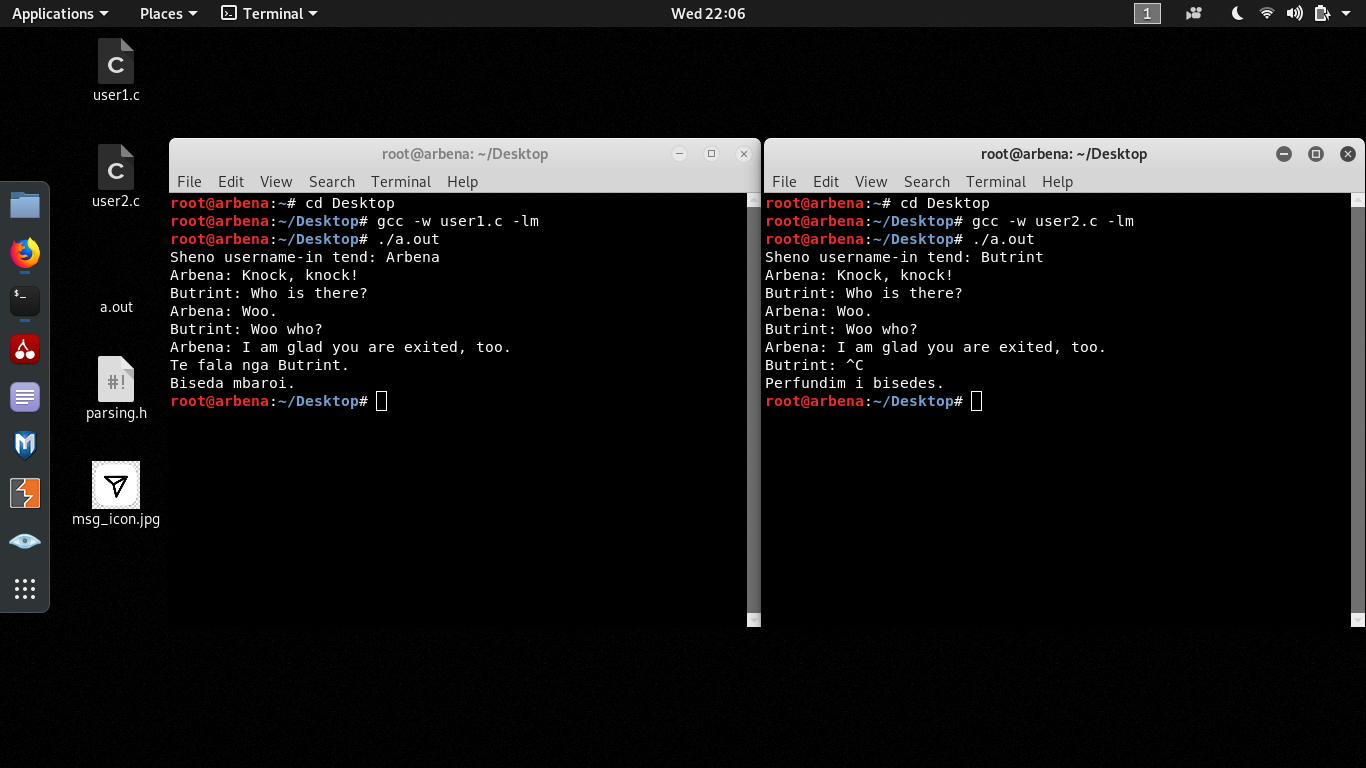
Përdoruesit të parë I kërkohet të shkruaj usermane të cilin do ta përdorë për identifikim gjatë komunikimit. Username I përdoruesit të parë I dërgohet përdoruesit të dytë. Edhe përdoruesit të dytë I kërkohet po e njejta gjë. Pas shkëmbimit të usermane-ave shkëmbehen id e proceseve dhe fillon komunikimi.



Ky komunikim është zhvilluar si chat I improvizuar. Sa herë që pranohet një mesazh sistemi shfaq një notification me emrin e bashkëbiseduesit dhe mesazhin që ka dërguar duke e shoqëruar edhe me njoftimin me tingull që mesazhi është pranuar.



Kur njëri prej përdoruesve tenton të terminojë procesin e tij do të dërgojë interrupt tek bashkëbiseduesi duke e përshëndetur dhe duke e imponuar edhe terminimin e këtij të fundit.



# Referencat

[1] [IPC using Message Queues](http://www.geeksforgeeks.org/ipc-using-message-queues)

[2] [IPC : Message Queues: <sys/msg.h>](https://users.cs.cf.ac.uk/Dave.Marshall/C/node25.html?fbclid=IwAR2HJihHFb9jsHzXW4qL6ZmrJXJcCtZ7f24gh9d0mIkX_32W4Or1QZ-jn1s)

[3] [Linux Signals](https://www.thegeekstuff.com/2012/03/catch-signals-sample-c-code/)