Prozesu konkurrenteen arteko komunikazioa eta sinkronizazioa

Egilea
Kepa Bengoetxea Kortazar
kepa.bengoetxea@ehu.es

Bibliografia

–http://www.chuidiang.com/clinux/procesos/procesoshilos.php

–Descripción Funcional de los Sistemas Operativos.-Iñaki Alegria

–UNIX.Programación Avanzada.-Manuel Márquez

–Programación en Linux con ejemplos. Kurt Wall

–http://wwwdi.ujaen.es/~lina/TemasSO/CONCURRENCIA/1ComunicacionySincronizacion.htm

–http://tiny.uasnet.mx/prof/cln/ccu/mario/sisop/sisop03.htm(exclusión mutua)

–http://tiny.uasnet.mx/prof/cln/ccu/mario/sisop/sisop05.htm(semáforos)

Motibazioa

Sistema eragile multiprogramatua: bi prozesu edo bi prozesu baino gehiago era konkurrentean egikaritzen ari dira.

Baliabideren bat erabiltzerakoan prozesuek kooperatu edota lehiatu egiten dute. Prozesuek kooperatu egiten dute helburu berdina lortzeko eta lehiatu baliabideak mugatuak direlako: prozesadorea, memoria, fitxategiak eta abar.

Motibazioa

- Prozesu ezberdinen artean komunikazioa eta sinkronizazioa erabiliko da, baliabideak partekatzeko.
- Baliabide posibleak: PUZ, S/I-rako gailuak, aldagaiak, errutinak etab.
- Balibide mota bi daude: partekagarriak / ez partekagarriak.

Motibazioa

Momentu berean partekagarriak diren baliabideak:

- Aldi berean konpartitu daitezke.
- k graduko baliabide partekagarriak ere badaude, k prozesuek batera atzitu dezakete baliabidea. Adibidez: buffer bat p1 idatzi eta p2 irakurri.
- Adibidez: irakurtzeko fitxategiak, memoria gune edo datu ez aldagarriak

Motibazioa

Baliabide ez partekagarriak:

- Aldi berean prozesu batek bakarrik atzitu ditzake
- Idazteko soilik irekitako fitxategiak, teklatua, aldagai globalak, inpresora ...

Komunikaziorako eta sinkronizaziorako mekanismoak

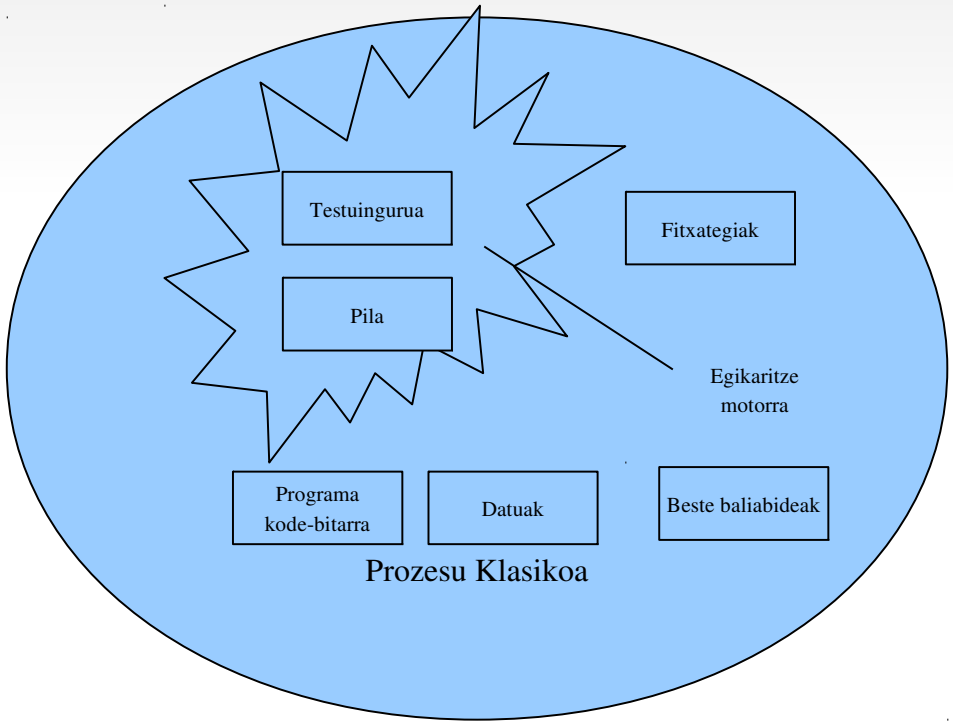
Komunikatzeko mekanismo ezberdinak daude:

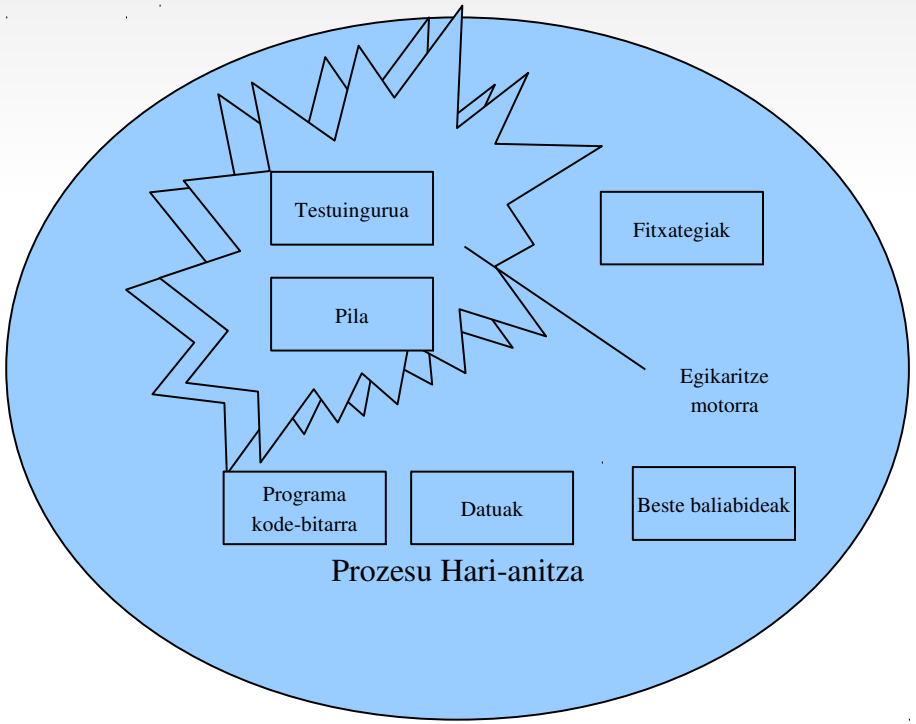
- Seinaleen bidez (ikusita)
- Fitxategien bidez (ikusita)
- Fitxategi bereziak: Tutuak (prozesuak lotzeko adib. ls | less)
- Aldagai partekagarriak erabiliz =Memoria partekagarria (ikusteko)
- ...

Aldagai partekagarriak. Hariak vs Prozesuak

- Prozesua hurrengo ingurune konputazionalean egikaritzen da:
  - Testu segmentua: programaren kode bitarra.
  - Datuen segmentua: aldagai globalak eta estatikoak
  - Zabaldutako fitxategiak eta beste baliabide batzuk: inpresora, teklatua eta abar.
  - Egikaritze-motorra:
    - Pila segmentua: hariko/prozesuko funtzio bati deitu ostean, pilan pila-egitura bat sartzen da funtzioak erabiltzen dituen aldagai lokal eta parametroen informazioarekin.
    - SEko testuinguru informazioa: SE-ak hariak/prozesuak kudeatzeko: PUZ erregistroen egoera eta programa kontagailua (egikaritutako azken agindua zein den) gordeko du.

Aldagai partekagarriak. Hariak vs Prozesuak





Aldagai partekagarriak. Hariak vs Prozesuak

- Gaur egungo SE-etan prozesuek hari-anitz izan ditzake.
- Hari-anitzeko prozesuetan, hari ezberdinak kode-bitarra, aldagai globalak eta estatikoak, fitxategiak, eta hainbat baliabide konpartitzen dituzte, baina hari bakoitzak bere pila (aldagai lokalak, funtzio parametroak …) eta bere testuingurunea erabiltzen ditu (PUZ erregistroak, kontadore programa etab.)
- Prozesu klasikoa: Prozesu bakoitza hari bakarra da, hau da, ez du ezer partekatzen.

Aldagai partekagarriak. Hariak vs Prozesuak

- Hari-anitzeko prozesu bat izanez gero sistema multiprogramatu batean bi hari edo gehiago une berean egikaritzen egon ahal dira, nahiz eta datu eta kode bera egikaritu. PUZ bat erabiltzen bada denboran txandakatuko dira, baina bi PUZ edo gehiago izanez gero paraleloki egikarituko dira, bakoitza PUZ batean.

Aldagai partekagarriak. Hariak vs Prozesuak

- Hariak, prozesuak baino arinago sortzen eta bukatzen dira. Ez da beharrezkoa, baliabideak esleitzea, guztiak konpartitzen baitituzte.
- PUZ bat egonez gero harien txanda aldaketa azkarragoa da, soilik pila eta testu-ingurunea gorde behar baitira.
- Hariak memoria gune bera konpartitzen dute. Eta euren arteko komunikazioa errazten da. Adibidez: aldagai globalak erabil ditzakegu.
- Hariak ikusteko: ps -eLf (-e Select all processes eta -f When used with -L, the NLWP (number of light-weight processes) and LWP (light-weight process ID) columns will be added.

Aldagai partekagarriak. Hariak vs Prozesuak

- pthread_create: hari bat sortzeko. Haria func funtzioarekin hasten da eta parametroak args-en bidez:int pthread_create (pthread_t *thread, const pthread_attr_t *attr, void *(*rutina) (void *), void *arg)
  - pthread_t * **thread: hariaren helbidea**
  - const pthread_attr_t *attr: hari-atributoen egituraren helbidea (NULL defektuz)
  - void *(*start_routine) (void *): funtzioaren helbide
  - void *arg: argumentuaren helbidea(asko egituraren bidez)
- pthread_join (thread_id): Lo geratzen da thread_id haria bukatu arte.
- pthread_exit (status): haria bukatzeko.

Thread

- Suppose we have to declare integer pointer,character pointer and float pointer then we need to declare 3 pointer variables.
- Instead of declaring different types of pointer variable it is feasible to declare single pointer variable which can act as integer pointer,character pointer.

```
void *ptr;    // ptr is declared as Void pointer

char cnum;

int inum;

float fnum;

ptr = &cnum;  // ptr has address of character data

ptr = &inum;  // ptr has address of integer data

ptr = &fnum;  // ptr has address of float data
```

Aldagai partekagarriak. Hariak vs Prozesuak

```
#include <stddef.h>
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
int sum=0;
void * process(void * arg){
int * num;
num=(int *)arg;
sum=sum+*num;
pthread_exit(0);
}
```

```
int main(){
pthread_t th_a, th_b;
int arg1=100,arg2=200;
pthread_create(&th_a, NULL, process, (void*)(&arg1));
pthread_create(&th_b, NULL, process, (void*)(&arg2));

pthread_join(th_a, NULL);

pthread_join(th_b, NULL);

printf("%d ", sum);
exit(0);
}

gcc -pthread -o hariak2 hariak2.c
```

Aldagai partekagarriak. Hariak vs Prozesuak

```
#include <stddef.h>

#include <stdio.h>

#include <pthread.h>

#include <stdlib.h>

int sum=0;

void * process(void * arg)

{

int i; int * num;

num=(int *)arg;

for (i= 1;i<=*num;i++)

   {sum=sum+1;}

pthread_exit(0);

}
```

```
int main(){

pthread_t th_a, th_b;

int arg1=10000,arg2=20000;

pthread_create(&th_a, NULL, process, (void*)
    (&arg1));

pthread_create(&th_b, NULL, process, (void*)
    (&arg2));


pthread_join(th_a, NULL);


pthread_join(th_b, NULL);


printf("%d ", sum);

exit(0);

}
```

Aldagai partekagarriak. Hariak vs Prozesuak

gcc -pthread -o hariakzenbatzen2 hariakzenbatzen2.c

./hariakzenbatzen2

22905

Zergatik?

bi hari erabiliko ditugu lerro-kopurua zenbatzeko, P1 hariak 1etik 10000ra eta P2 1etik 20000ra:

SK konpartitu ezin den kodea hurrengo hau izango da: sum=sum+1

Bi prozesuak momentu berean SK-an sartuz gero, batuketaren bat galdu ahal da. Nahiz eta pentsatu agindu hau atomikoa dela, ez da horrela:

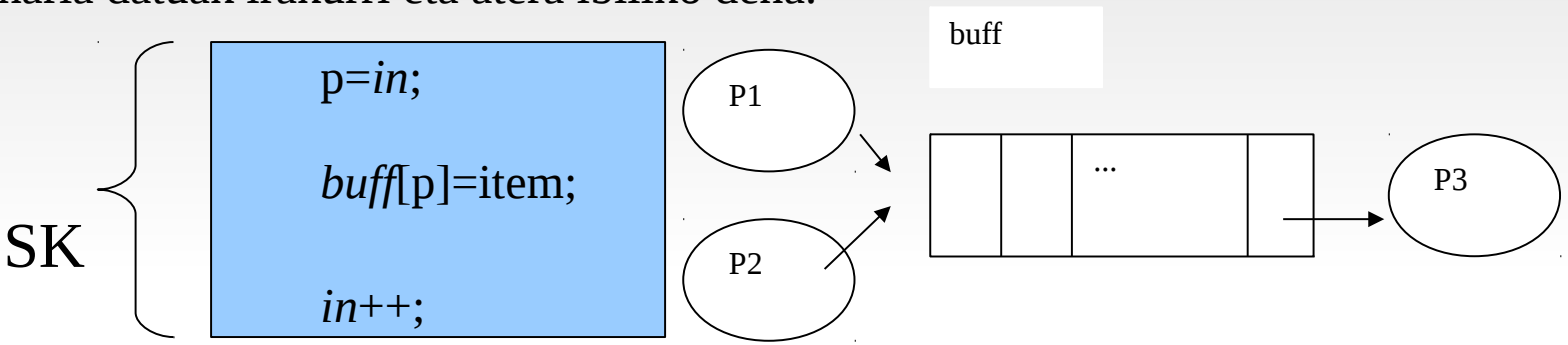
mov sum regX;

inc regX;

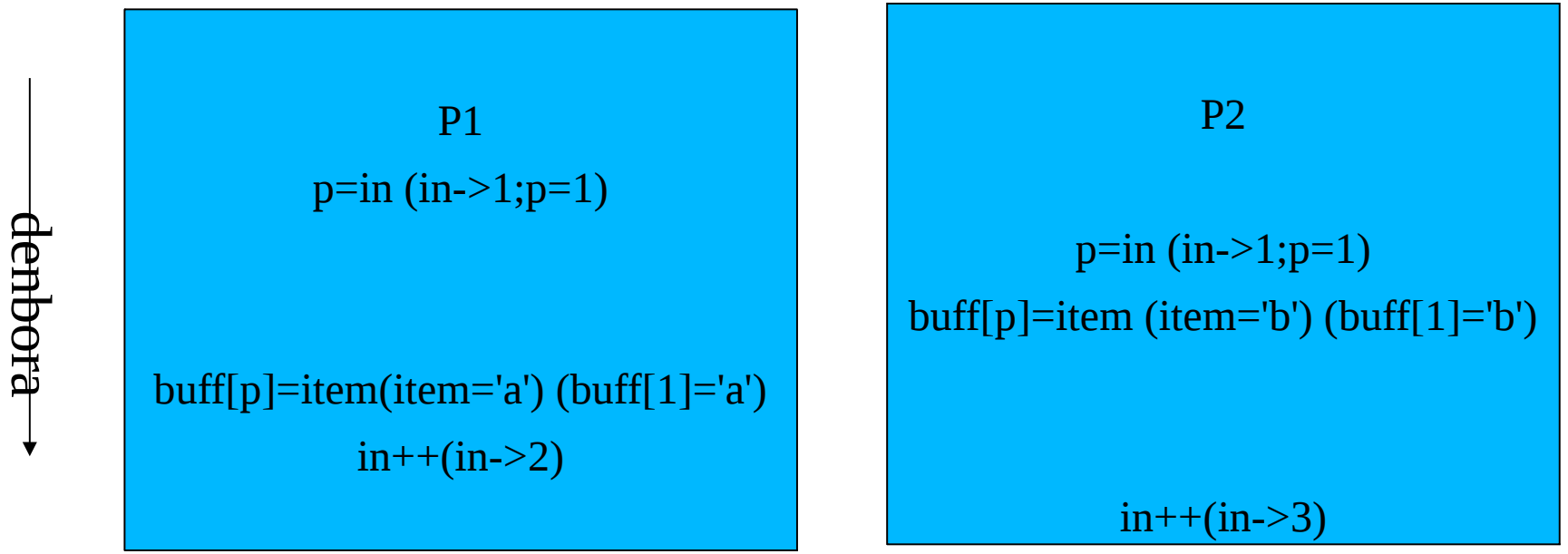
mov regX sum;

Aldagai partekagarriak. Sekzio kritikoa

- Sekzio kritikoa: Zenbait prozesu konpartitutako baliabideak atzitzen duen kodea. Adb: Hurrengo prozesu honek bi aldagai global eta partekagarriak ditu, “buff" eta "*in*". Programa honek bi hari ditu p1 eta p2 buff bufferrean datuak sartzen ibiliko direnak. Eta p3 haria datuak irakurri eta atera ibiliko dena.



- Zer gertatuko zen p1 eta p2 era honetan egikarituz gero?



Aldagai partekagarriak. Sekzio kritikoa

Adb: Telemaratoia, teleoperadore bakoitzak exekutagarri bat du, eta datu-basea konpartitzen dute.

BEGIN WORK*

Select dinero_acum,num_llamada from SOLIDARIDAD;

dinero=dinero_acum+dfdinero

numllamada=num_llamada+1

update SOLIDARIDAD set dinero_acum =dinero,num_llamada=numllamada

COMMIT*

*A COMMIT statement in SQL ends a transaction within a relational database management system (RDBMS) and makes all changes visible to other users. The general format is to issue a BEGIN WORK statement, one or more SQL statements, and then the COMMIT statement. Alternatively, a ROLLBACK statement can be issued, which undoes all the work performed since BEGIN WORK was issued.

Aldagai partekagarriak. Sekzio kritikoa

- Exekuzioaren arabera emaitzak ezberdinak ez izateko, SK zein den, detektatu behar da. Eta sarrera eta irteera protokolo bat ezarriko dugu. Hurrengo propietateak aztertuz:
    - Elkarrekiko esklusioa
    - Progrezio finitua
    - Itxarote mugatua
    - Elkar-blokeaketarik ez

Aldagai partekagarriak. Sekzio kritikoa

Elkarrekiko esklusioa:

-Prozesu bat baino gehiago, ezin da SKan aldi berean egon. k graduko baliabidea partekagarria baldin bada, k prozesu baino gehiago ezin izango dira aldi berean bere SKan egon. Adibidez, p=in; buff[p]=item; in++; SK bat da. Hori esan nahi du prozesu batek instrukzio hauek egikaritzen duen bitartean, ezin dela besteren bat sartu hauek egikaritzera.

Progrezio finitua:

- SKan ez badago prozesurik, SKan sartu nahi direnen artean erabaki behar da, ze prozesu sartu behar den. Sartu nahi ez dutenak ezin dute erabakian parte hartu. Gainera erabaki hori denbora finitu batean gertatu behar da.

Aldagai partekagarriak. Sekzio kritikoa

Itxarote mugatua: Prozesu bat ezin da denbora luzez itxaroten egon SKan sartzeko.

Elkar-blokeaketarik ez: bi prozesu p1 eta p2-en artean elkar-blokeaketa ematen da, baliabide batzuk partekatzen eta hauek lortzeko leiatzerakoan, bata eta bestea ezin jarraituta geratzen direnean, p1-ek p2 behar duen baliabidea duelako eta p2-k p1-ek behar duena duelako, hori dela eta biak ezin aurrera jarraituta geratuko dira.

Komunikatzeko metodoak

- Sekzio kritikoa= kode zati bat da, atzipen esklusiboa duen baliabide partekatu kudeatzeko, hau da, kode zati hori momentu berean ezin dena elkarbanatu k prozesu baino gehiagoekin. Gehienetan k=1 izango da.
- SKaren kodea egikaritzeko orduan, momentu oro prozesu bakarra dagoela kontrolatzeko, SK sartu baino lehen, sarrera-protokolo bat eta SKtik irten ostean irteera-protokolo bat ezarriko dugu.

  sarrera-protokoloa

  SK

  irteera-protokoloa

- Sarrera eta irteera protokoloak definitzerako orduan metodo ezberdinak erabil ditzakegu:
  - Itxarote aktiboa (inkesta)
  - Blokeatzearen bidezko itxarotea: Semaforoak

Komunikatzeko metodoak

- Protokoloak hurrengo usteetarako soilik erabiliko ditugu:
  - PUZ bat (taskset -c 1 ./exekutagarria)
  - Bi hari aldi berean egikaritzen.
  - Bi hariak SK partitu egiten dute.
  - Testuingurune aldaketa edozein momentutan gerta daiteke.
  - Hariak ondo bukatuko dira (errorerik ez dela gertatzen).
  - Nahiz eta hari bat bukatu, bestea jarraitu ahal du hainbat aldiz SKan sartzen.

Komunikatzeko metodoak

itxarote aktiboa: Lehenengo proposamena:

Itxarote aktiboa: Hariak, bere txanda den edo ez jakiteko, behin eta berriro, etengabe, aldagai baten balioaz galdetuko du. VAB aldagai globala erabiliko da. VAB=i, izanez gero Pi hariak aurrera egin dezake. Adibidez: int VAB=1 bezala hasieratuz gero

```
i=1;//P1
while (i<5){
//sarrera-protokoloa_1:
while (VAB==2) {NOP;}
SK
//irteera-protokola_1:
VAB=2;
i++;}
```

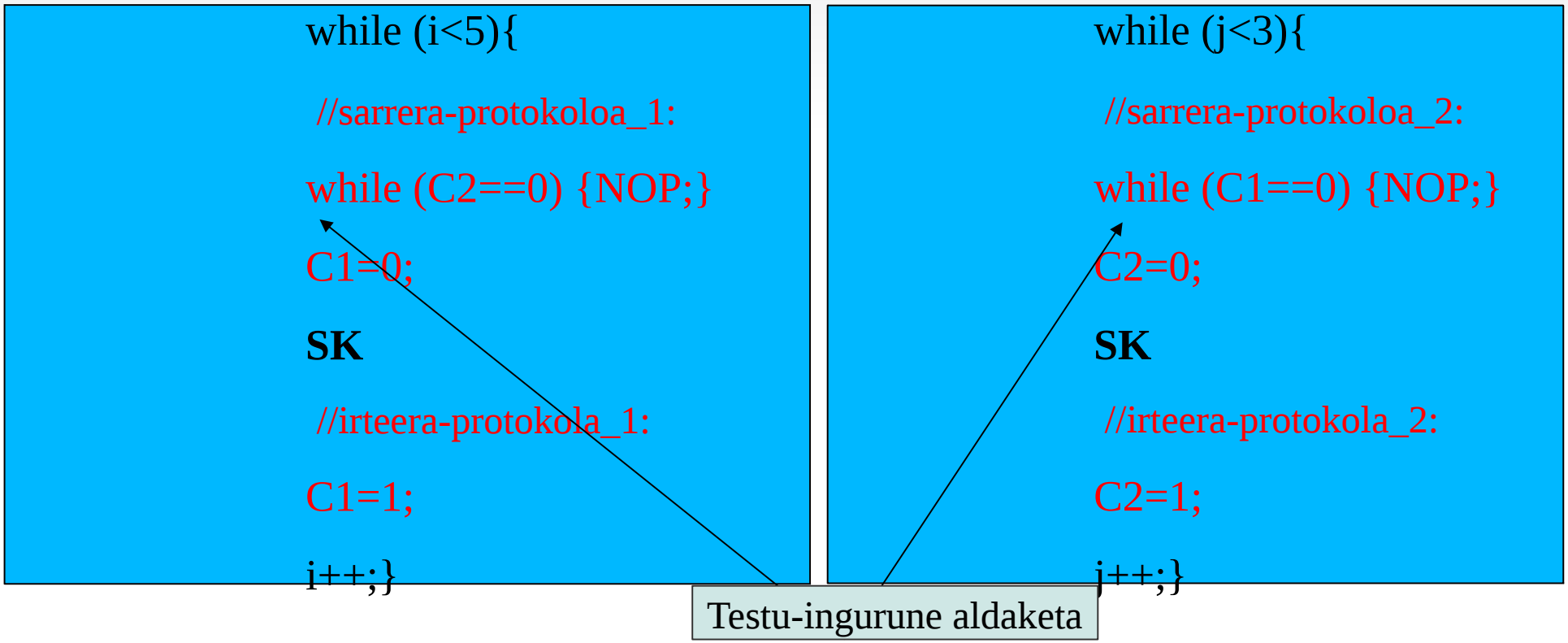
```
j=1;//P2
while (j<3){
//sarrera-protokoloa_2:
while (VAB==1) {NOP;}
SK
//irteera-protokola_2:
VAB=1;
j++;}
```

- Arazoa: Progrezio finitua ez da betetzen, hau da, bigarren prozesuak j<3 daukana seguraski arinago bukatuko du, eta prozesu hau bukatu ostean ezin izango du besteak SK-an sartu.

Komunikatzeko metodoak

Itxarote aktiboa: Bigarren prosamena:

Hari bakoitzak (P1, P2) aldagai propio bat erabiltzen du. C1=0 baldin bada SK sartzeko txanda P1ena izango da, eta C2=0 izanez gero P2rena dela esan nahi du. Hasieran C1 eta C2 aldagai globalak 1era hasieratzen dira.

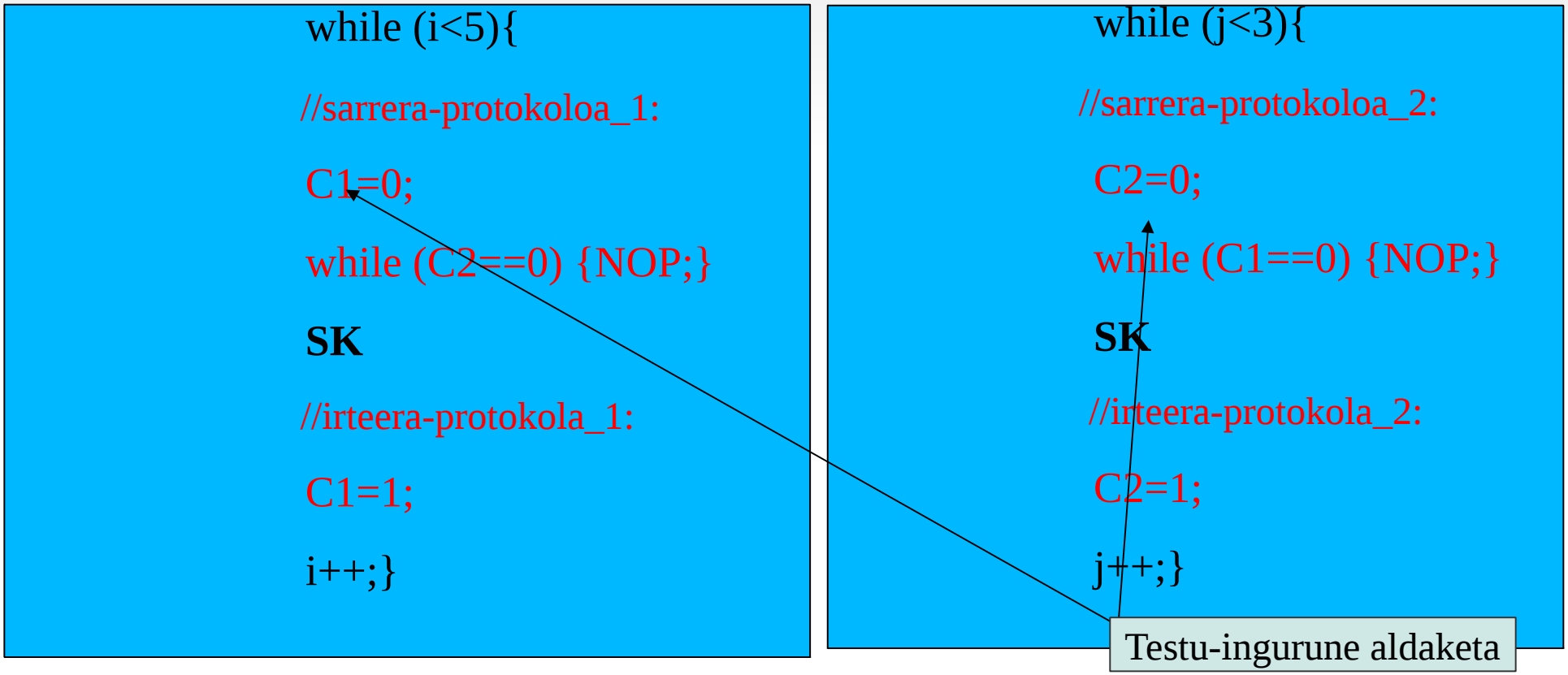


Arazoa:Elkarrekiko esklusioa ez da betetzen. Gertatu ahal da C1=0 edo C2=0 markatu baino lehen testuingune aldaketa gertatzea, eta biak SKan sartzea.

Komunikatzeko metodoak

itxarote aktiboa: Hirugarren proposamena

Aurreko proposamenaren aldagai berak erabiliko ditugu, baino oraingoan sarrerako_protokoloan lehenengo eta behin 0ra jarriko dugu aldagaia eta gero galdetuko dugu. Hau da, P1 SKan sartu nahi izanez gero C1=0 jarriko du. Eta gero galdetuko du, bestea sartu nahi den “while”aren bitartez. C1 eta C2 1era hasieratzen dira.



Arazoa:bi prozesuen arteko elkar-blokeaketa gertatu ahal da, biak C1=0 eta C2=0 jartzea, eta ezin denez euren balorea 1 jarri “while”aren barruan blokeatuta geratuko dira biak.

Komunikatzeko metodoak

itxarote aktiboa: Laugarren proposamena

Aurreko proposamenaren aldagai berak erabiliko ditugu, eta lehenengo eta behin 0ra jarriko dugu aldagaia eta gero galdetuko dugu. Baina proposamen honetan NOP agindua jarri beharrean, elkar-blokeoa ekiditzeko C1=1;C1=0 aginduak jarriko ditugu. Horrela C1=1 aginduaren ostean testuingurune aldaketa gertatuz gero elkar-blokeoa ekiditzeko. C1 eta C2 1 hasieratuko ditugu.

```
while (i<5){
 //sarrera-protokoloa_1:
C1=0;
 while (C2==0) {C1=1;C1=0;}
SK
 //irteera-protokola_1:
C1=1;
i++;}
```

```
while (j<3){
 //sarrera-protokoloa_2:
C2=0;
while (C1==0) {C2=1;C2=0;}
SK
//irteera-protokola_2:
C2=1;
j++;}
```

Arazoa: Itxarote mugatua ez betetzea gerta daiteke. Adibidez P1 eta P2 “tandem”ean (agindu bat eta testuingurune aldaketa gertatzea) egikarituz gero mugagabeko atzerapena gerta daiteke. Proposamen hau ez du balio hegaldi edo taupada-markagailu baterako.

Komunikatzeko metodoak

itxarote aktiboa: soluzioa erakusteko hurrengo adibidea aurkeztuko dugu: bi hari erabiliko ditugu sum aldagai globalari bi balio banan-banan gehitzeko, p1 hariak 1tik 10000ra eta p2 1tik 20000ra:

SK konpartitu ezin den kodea hurrengo hau izango da: sum = sum + 1;

Bi prozesuak momentu berean SK-an sartuz gero, lerro batuketaren bat galdu ahal da. Nahiz eta pentsatu agindu hau atomikoa dela, ez da horrela:

```
mov sum regX;

inc regX;

mov regX sum;
```

Komunikatzeko metodoak

Denb	Haria	Aginduak	erreg1	erreg2	sum
1	P1	mv sum erreg1	0	0	0
2	P1	inc erreg1	1	0	0
3	P2	mv sum erreg2	1	0	0
4	P2	inc erreg2	1	1	0
5	P1	mv erreg1 sum	1	1	1
6	P2	mv erreg2 sum	1	1	1

Komunikatzeko metodoak

itxarote aktiboa: Dekker soluzioa:

```
#include <stddef.h>

#include <stdio.h>

#include <pthread.h>

#include <stdlib.h>

void * process1(void * arg);
void * process2(void * arg);
int sum=0;

int turno=1;

int c1=1;

int c2=1;
```

```
int main(){
pthread_t th_a, th_b;
int arg1=10000,arg2=20000;
pthread_create(&th_a, NULL, process1,(void *) (&arg1));
pthread_create(&th_b, NULL, process2,(void *) (&arg2));

  /// wait all threads by joining them


  pthread_join(th_a, NULL);


  pthread_join(th_b, NULL);

printf("batuketa: %d\n ", sum);
 exit(0);
}
gcc -pthread -o dekker dekker.c
```

Itxarote aktiboa: Dekker soluzioa

Haria 1:

```
void * process1(void * arg){
int *num;
int i;
num=(int *)arg;
for (i=1;i<=*num;i++)
{
//PE:
c1=0;
while (c2==0)
   {if (turno==2)
        {c1=1;
         while (turno==2);
         c1=0;
        }
   }
//SK:
sum=sum+1;
//PS:
c1=1;
turno=2;
}
pthread_exit(0);
}
```

Haria 2:

```
void * process2(void * arg){
int *num;
int i;
num=(int *)arg;
for (i=1;i<=*num;i++)
{
//PE:
c2=0;
while (c1==0)
   {if (turno==1)
        {c2=1;
         while (turno==1);
         c2=0;
        }
   }
//SK:
sum=sum+1;
//PS:
c2=1;
turno=1;
}
pthread_exit(0);
}
```

Komunikatzeko metodoak

itxarote aktiboa: Dekker soluzioa

Soluzio honetan “c1” eta “c2” 0 izanez gero, “turno” hirugarren aldagaia erabiliko da prozesu bietatik zein jarraituko duen esateko.

- **P1** SK-an sartu nahi denean “c1” 0ra jarri, eta “c2”ren baloreagatik galdetuko du, “c2” 0 izanez gero P2 ere sartzeko asmoa duela esan nahi du eta bietatik SKan zein sartuko den jakiteko “turno” aldagaian begiratzen du.
  - “turno” berdin 1 izanez gero P1 SKan sartuko zen.
  - “turno” berdin 2, izanez gero “c1=1” jarriko du, P2ri SK-an sartzeko aukera emateko eta elkar-blokeoa ekiditzeko; “turno” berdin 2 den bitartean P1 ez du ezer egingo. “turno” berdin 1 jarriz gero, “c1=0” jarriko da, eta P1 zuzenean SKan sartuko da.
- P1ek SKtik irteterakoan “turno=2” eta “c1=1” jarriko ditu.

Komunikatzeko metodoak

itxarote aktiboa: Dekker soluzioa

Elkarrekiko esklusioa: Probatu biak sartu nahi izanez gero, bakarra sartu ahal dela. BAI, biak sartu nahi izanez gero “turno”ren balorearen arabera, bat sartuko da eta bestea blokeatuta geratuko da, “turno” ezin delako momentu berean 1 edo 2 izan. Adibidez: P1 eta P2, “c1” eta “c2” 0ra jarriz gero, eta “turno” berdin 2 baldin bada. Zer gertatuko zen? P1 “if”ko “then”aren barruan sartuko zen, “c1=1” jarriz, eta “turno=2” den bitartean NOP geratuko da. Orduan P2, kanpoko while egonez gero “c1=1” denez zuzenean SKan sartuko zen.

Komunikatzeko metodoak

itxarote aktiboa: Dekker soluzioa

Progrezio finitua: Hau probatzeko, hari batek bukatuz gero, besteak jarraitu ahal duen ikustea da.

Abibidez: P2k bukatzerakoan “c2=1” eta “turno=1” jartzen ditu, hori dela eta, P1 kanpoko “while”tik SKan sartu ahal da edo”turno=1”ekin barruko “while”tik atara, kanpoko “while”ra joan eta bertatik zuzenean SKan sartuko zan.

Komunikatzeko metodoak

itxarote aktiboa: Dekker soluzioa

Elkar-blokeoa: bi hari p1 eta p2ren artean elkar-blokeoa ematen dela esaten dugu. Bi hariak baliabide batzuk partekatzen eta hauek lortzeko leiatzen direnean, hurrengo hau gerta denean, bata eta bestea ezin jarraituta geratzea, p1ek p2 behar duen baliabidea duelako eta p2k p1ek behar duena, hori dela eta biak ezin aurrera jarraituta geratuko dira. Adibide honetan ezin dira elkar-blokeatuta geratu, “turno” aldagaiak ezin duelako bi balore ezberdin izan momentu berean. Beren balorearen arabera bata edo bestea SKan sartuko da.

Komunikatzeko metodoak

itxarote aktiboa: Dekker soluzioa

Itxarote mugatua: Prozesu bat ezin da denbora luzez itxaroten egon SKan sartzeko. Hau probatzeko ikusi behar da, ia prozesu batek denbora luzez egon ahal den SKan sartzen besteari utzi gabe. Soluzio honetan hau ez da gertatzen. Adibidez, P2k sartu nahi izanez gero eta P1 SKtik irten eta berriro sartu nahi izanez gero, zer gertatuko den aztertuko dugu. P1ek SKtik irtetzerakoan “c1=1” eta “turno=2” jartzen ditu. Orduan P2 kanpoko “while”an egonez gero zuzenean SKan sartuko da eta barruko “while”an egonez gero , bertatik irten “c2=0” jarri eta kanpoko “while”ra joango gara. P1 eta P2 kanpoko “while”an egonda; “c1” eta “c2” 0ra daude baina “turno=2” da, hori dela eta P1 barruko “while”an geratuko da eta P2 SKan sartuko da P1ek “c1=1” jartzen duen momentutik.

Komunikatzeko metodoak

itxarote aktiboa: Peterson soluzioa:

```
#include <stddef.h>

#include <stdio.h>

#include <pthread.h>

#include <stdlib.h>

void * process1(void * arg);
void * process2(void * arg);
int sum=0;

int turno=1;

int c1=1;

int c2=1;
```

```
int main(){
pthread_t th_a, th_b;
int arg1=10000,arg2=20000;
pthread_create(&th_a, NULL, process1,(void *) (&arg1));
pthread_create(&th_b, NULL, process2,(void *) (&arg2));

  /// wait all threads by joining them


  pthread_join(th_a, NULL);


  pthread_join(th_b, NULL);

printf("batuketa: %d\n ", sum);
 exit(0);
}
```

Komunikatzeko metodoak

itxarote aktiboa: Peterson soluzioa:

Haria 1:

```
void * process1(void * arg){
int *num;
int i;
num=(int *)arg;
for (i=1;i<=*num;i++)
{
//PE:
c1=0;
turno=2;
while ((c2==0)&&(turno==2));
//SK:
sum=sum+1;
//PS:
c1=1;
}
pthread_exit(0);
}
```

Haria 2:

```
void * process2(void * arg){
int *num;
int i;
num=(int *)arg;
for (i=1;i<=*num;i++)
{
//PE:
c2=0;
turno=1;
while ((c1==0)&&(turno==1));
//SK:
sum=sum+1;
//PS
c2=1;
}
pthread_exit(0);
}
```

Komunikatzeko metodoak

itxarote aktiboa: Peterson soluzioa:

$gcc -pthread -o peterson peterson.c

**$taskset -c 1 ./peterson

**En algoritmos con un fuerte dependencia al ordén de ejecución de las instrucciones, ya que, cuando se trabaja con procesadores multicore por optimización, los procesadores utilizan un órden de ejecución débil (adelantando y atrasando ciertas instrucciones para optimizar en tiempo de procesador).

Komunikatzeko metodoak

itxarote aktiboa: Peterson soluzioa:

- P1-ek SK-an sartzeko C1=0 eta turno=2 jartzen ditu. Honela P2 sartu nahi izanez SK -an sartuko da.
- Momentu berean sartzeko ahaleginak eginez gero, “turno”ren balioak 1 eta 2 balioak ia momentu berean izango ditu. Baina soilik bat iraunduko du, besteak berehala galduko du bere balorea. Eta irauten duen “turno” baloreak erabakiko du zein sartuko den SK-an.

Komunikatzeko metodoak

- **Blokeo bidezko itxarotea: Semaforoak**
  - Itxarote aktiboan Prozesadore denbora xahutzen edo alferrigaltzen du. SK-rekiko itxarote-aldia laburra denean erabiltzen da. Adibidez: produzitzaile-kontsumitzaile buffer batekin. **Blokeo bidezko itxarotea:** SK-rekiko itxarote-aldia luzea denean erabiltzen da. Adibidez: disko S/I. Sistema honek prozesua lotan jartzen du, eta ez da esnatuko itxaroten(lotan) jarritako gertaera bukatu arte.

Komunikatzeko metodoak

-Blokeo bidezko itxarotea: Semaforoak

- **Semaforoa** bat, aldagai babestu bat da, non bere balioa solik wait, signal eta initial metodoen bitartez atzitu eta aldatu daitekeen.
- **Semaforo bitarrek** (mutex deritzonak) soilik 0 eta 1 balioak har ditzakete.
- **Semaforo-kontagailuek** berriz, bat baino handiagoko zenbaki osoak har ditzakete.

Komunikatzeko metodoak

Adibidez:

initial(s,1)

wait (s)

S.K

signal(s)

* wait,signal,initial agindu atomikoak dira.

Komunikatzeko metodoak

Wait(S) eragiketak, horrela egiten du lan:

```
S=S-1;
if S<0 then itxaron(S);
```

} Atomikoa da

* wait-ek prozesua blokeatuta jarriko du, eta S baloreari dagokion itxarote ilaran.

Signal(S) eragiketak, horrela funtziotzen du:

```
S=S+1;
if S<=0 then esnatu(S);
```

} Atomikoa da

*signal-ek, S baloreari dagokion itxarote ilaratik, prozesu bat atera eta esnatuko du.

initial(S,balorea):

Semaforoa balorea-ren edukinarekin jarri. Balorea momentu berean SK-an zenbat prozesu sartu ahal diren adierazten du. Adibidez: initial(S,5) S=5 jartea bezala da.

Semaforoak

- pthread_mutex_t mutex; semaforo baten erazagupena
- pthread_mutex_init(&mutex,NULL); //init bezala, semaforaren erakuslea argumentu bezala eta semaforoaren ezaugarriak defektuz NULL(semaforo mutex arrunta)
- pthread_mutex_lock(&mutex); //wait bezala, semaforaren erakuslea argumentu bezala.
- pthread_mutex_unlock(&mutex); // signal bezala, semaforaren erakuslea argumentu bezala.

semaforoak

```
#include <stddef.h>
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>

int sum=0;
pthread_mutex_t mutex;
void * process(void * arg){
int * num;
num=(int *)arg;
int i;
for (i= 1;i<=*num;i++)
  {
    pthread_mutex_lock(&mutex);
    sum=sum+1;
    pthread_mutex_unlock(&mutex);
 }
printf("%d haria bukatuta\n", *num);
pthread_exit(0);}
```

```
int main(){
pthread_t th_a, th_b;
pthread_mutex_init(&mutex,NULL);
int arg1=10000,arg2=20000;
pthread_create(&th_a, NULL, process,(void *)
  (&arg1));
pthread_create(&th_b, NULL, process,(void *)
  (&arg2));
/// wait all threads by joining them
pthread_join(th_a, NULL);
pthread_join(th_b, NULL);
printf("batuketa: %d\n ", sum);
exit(0);
}
```

```
gcc -o hariakzenbatzensemaforoekin
  hariakzenbatzensemaforoekin.c -lpthread
```