

5. gaia (II)

Monitoreak eta baldintzen sinkronizazioa (II)

5.2 Semaforoak

s semaforoa:

soilik balio ez-negatiboak har ditzakeen osoko aldagaia da.

s-k onartzen dituen eragiketa bakarrak: *gora()* eta *behera()*.

```
behera(s):  if s > 0
              then gutxitu s
              else deia egin duen prozesuaren exekuzioa blokeatu
```

```
gora(s):    if prozesu blokeatuak daude s-n
              then haietako bat esnatu
              else gehitu s
```

Blokeatutako prozesuak FIFO ilara batean gelditzen dira.

Semaforoak modelatzen

Analizatu ahal izateko, balio multzo finitua har dezaketen semaforoak modelatuko ditugu.

Balio horietatik ateratzen bada **ERROR** itzuliko du.

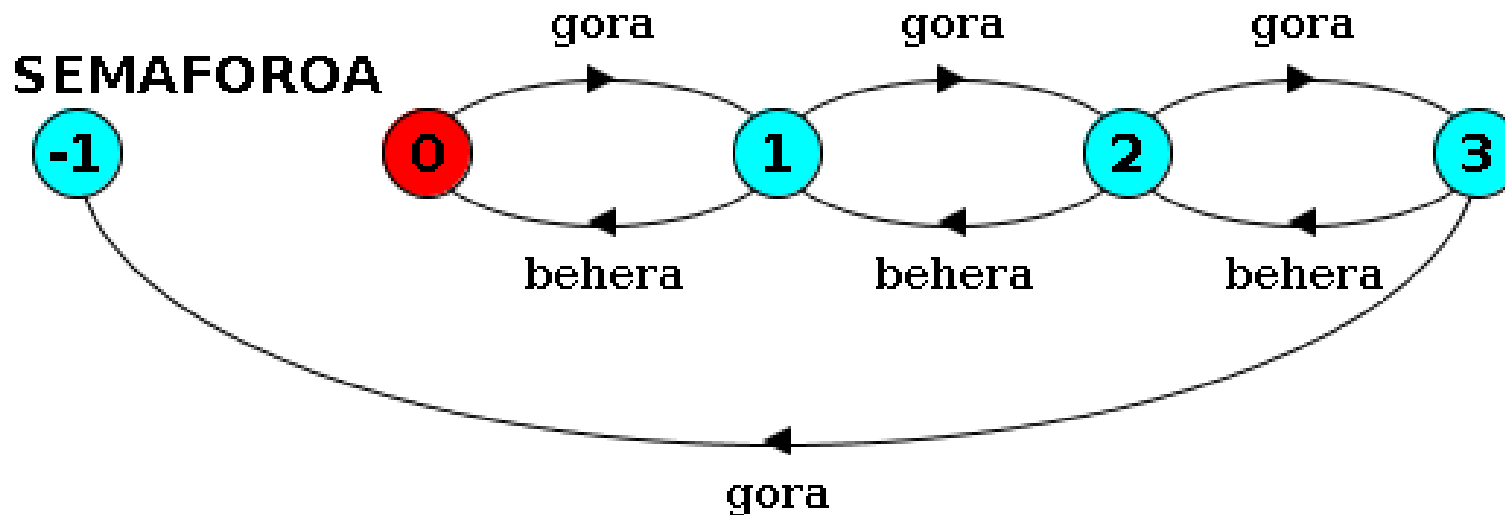
```
const Max = 3
range Int = 0..Max

SEMAFOROA (N=0) = SEMA[N] ,
SEMA[b:Int]    = (
                    | when (b>0) gora ->SEMA[b+1]
                    | when (b>0) behera->SEMA[b-1]
                ) ,
SEMA[Max+1]    = ERROR.
```

N hasierako balioa da.

lerro hau ez da
beharrezkoa

Semaforoak modelatzen



behera ekintza onartzen da soilik
semaforoaren v balioa 0 baino handiagoa denean ($v > 0$).

gora ekintza ez da babesten.

Errorera eramaten duen traza: **gora** \rightarrow **gora** \rightarrow **gora** \rightarrow **gora**

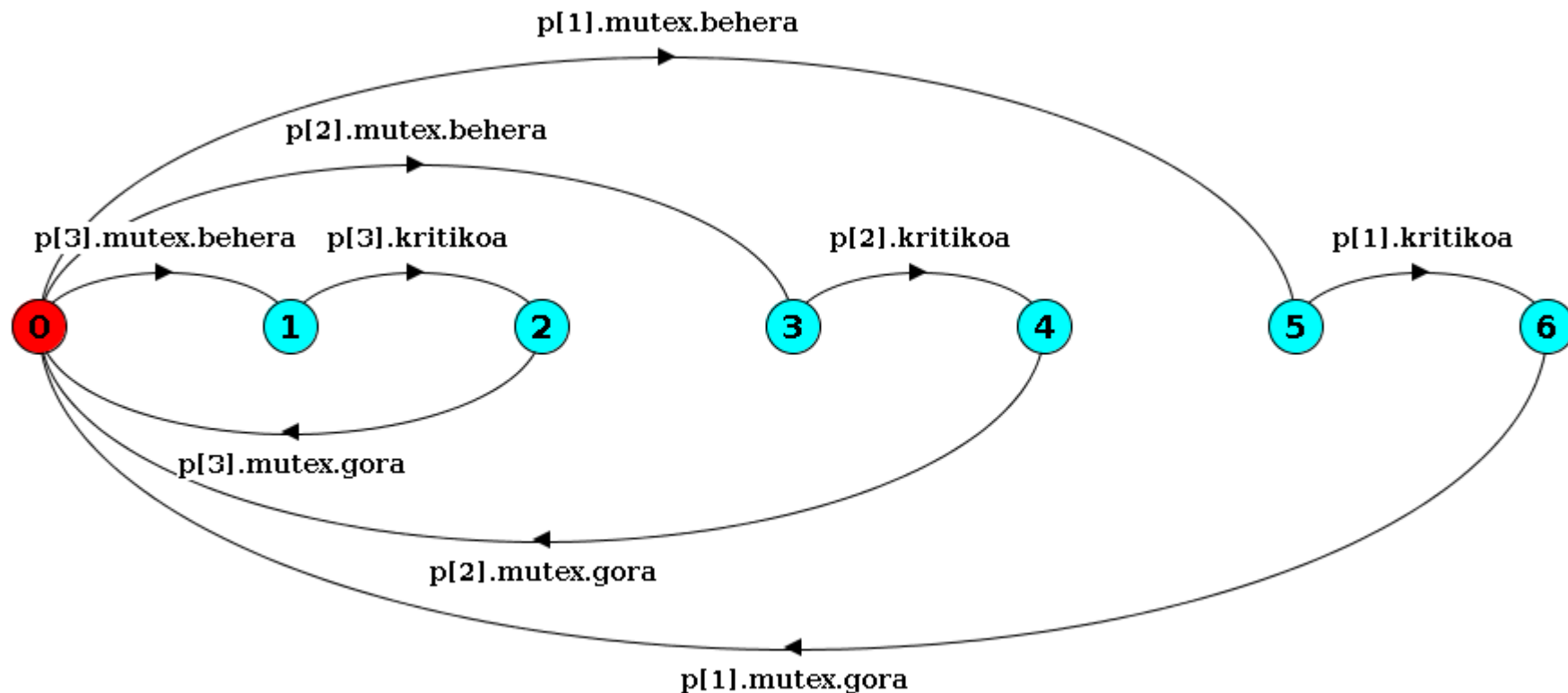
Semaforoekin adibidea - eredua

Adibide bezala semaforo baten erabilera modelatuko dugu elkar-bazterketa ziurtatzeko.

Hiru prozesu **p[1..3]** erabiltzen dute **mutex** semaforo bat baliabide baten atzipenean (**kritikoa** ekintza) elkar-bazterketa ziurtatzeko

```
P = (mutex.behera->kritikoa->mutex.gora->P) .  
|| SEMA DEMO = (    p[1..3]:P  
                  || {p[1..3]} :: mutex:SEMAFOROA(1) ) .
```

Semaforoekin adibidea - eredua



- Elkar-bazterketa ziurtatzeko, semaforoaren hasierako balioa 1 da. **Zergatik?**
- **SEMADEMO** iritsi daiteke **ERROR** egoerara?
- Semaforo **bitarra** nahikoa al da (i.e. **Max=1**) ?

Semaforoak Java-n

Semaforoa objektu pasiboa da, **monitore** batekin implementatua.

(Semaforoa behe-mailako mekanismo bat da, askotan goi-mailako monitoreak eraikitzeke erabilita)

```
public class Semaforo {  
    private int balioa;  
  
    public Semaforo (int hasierakoa)  
        {balioa = hasierakoa;}  
  
    synchronized public void gora() {  
        ++balioa;  
        notify();  
    }  
  
    synchronized public void behera()  
        throws InterruptedException {  
        while !(balioa>0) wait();  
        --balioa;  
    }  
}
```

SEMADEMO programa - MutexLoop

```
class MutexLoop extends Thread {
    Semaforo mutex;
    String tarteia;
    int luz;
    MutexLoop (Semaforo sema, int zenbat, String tabul){
        mutex=sema; luze=zenbat; tarteia=tabul;
    }
    public void run(){
        try {while(true) {
            for (int i=1;i<=6;i++)      ekintza("|");    // Ekintza ez kritikoa
            mutex.behera();              // Eskuratu elkar-bazterk
            for (int i=1;i<=luze;i++) ekintza("*");      // Ekintza kritikoa
            mutex.gora();                // Askatu elkar-bazterk
        }
        } catch (InterruptedException e){}
    }
    void ekintza(String ikurra) {
        try {
            System.out.println(tarteia+ikurra);
            sleep((int) (Math.random()*1000));
        } catch (InterruptedException e) {}
    }
}
```

Programa nagusia idatzi eta hariak eta semaforoa **main** metodoan sortu.
(Hari kopurua parametrizatuta)

Pantaila klasea sortu eta trazaren idazketak klase horretan egin.

Probatu sekzio kritikoan ematen den **tarteia aldatzen**, eta aztertu gatazka gehiago edo gutxiago ematen diren.

SEMADEMO pantailan

| : ekintza ez-kritikoa

* : ekintza kritikoa

2. haria zain dago

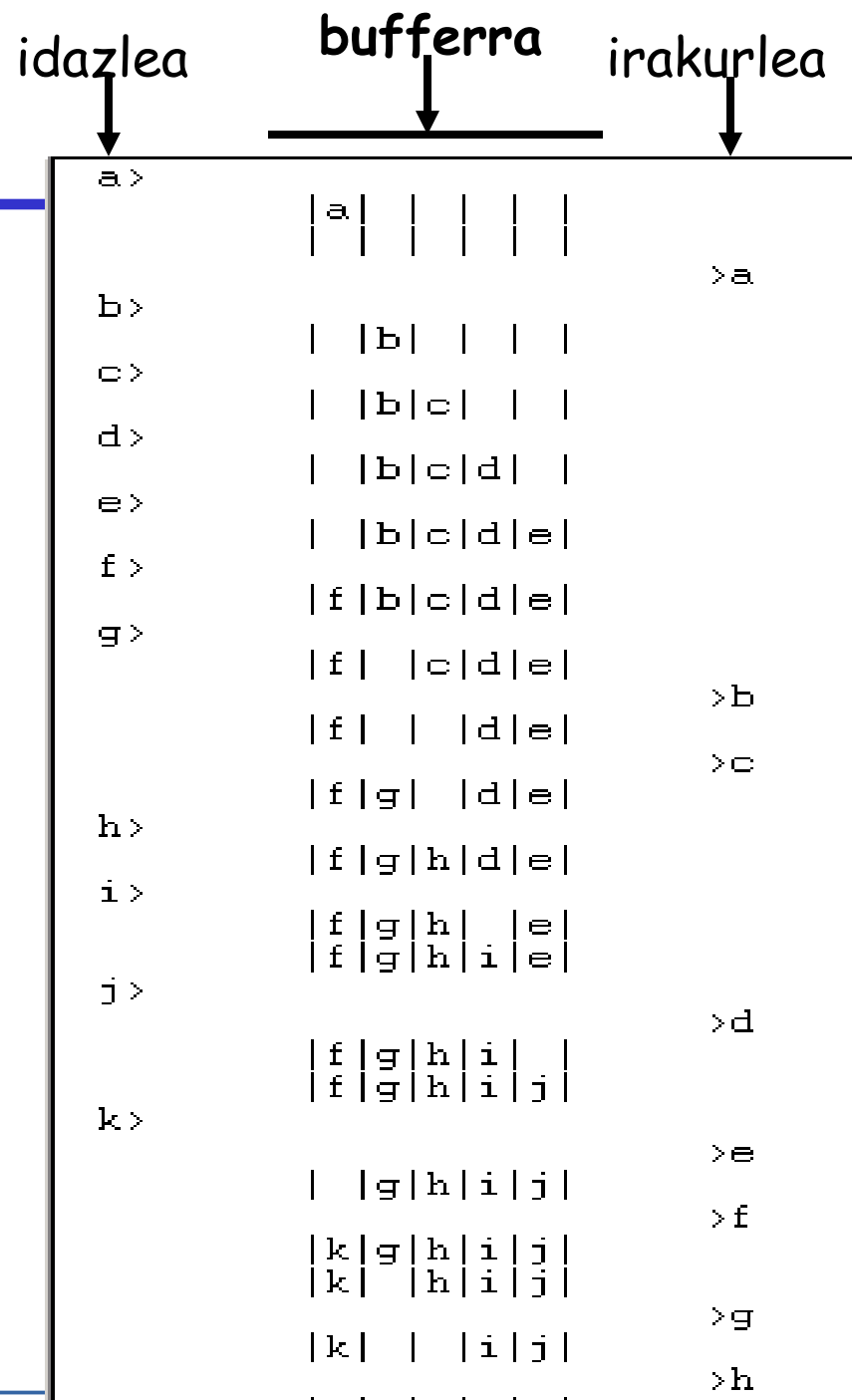
[illegible]

5.3 Buffer mugatuak

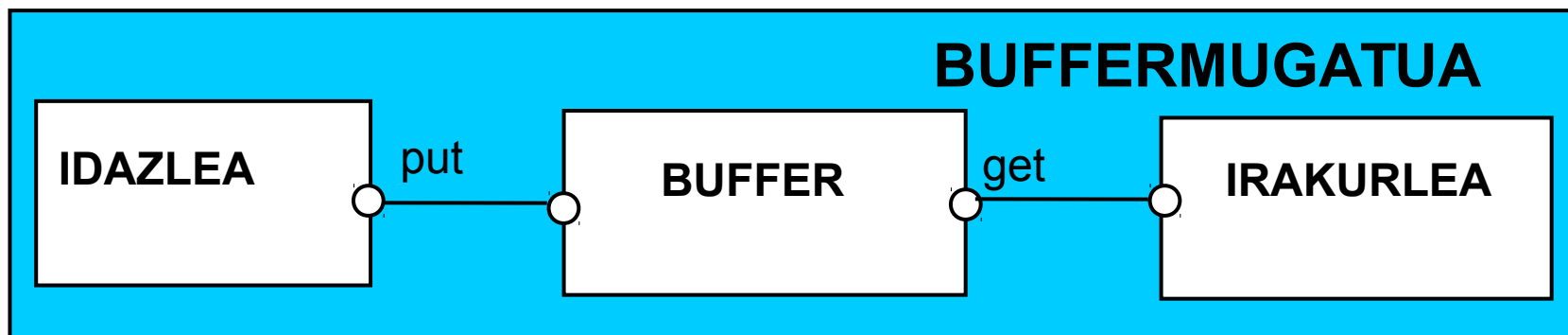
Buffer mugatu bat slot kopuru finko batez osatua dago.

Bufferrean:

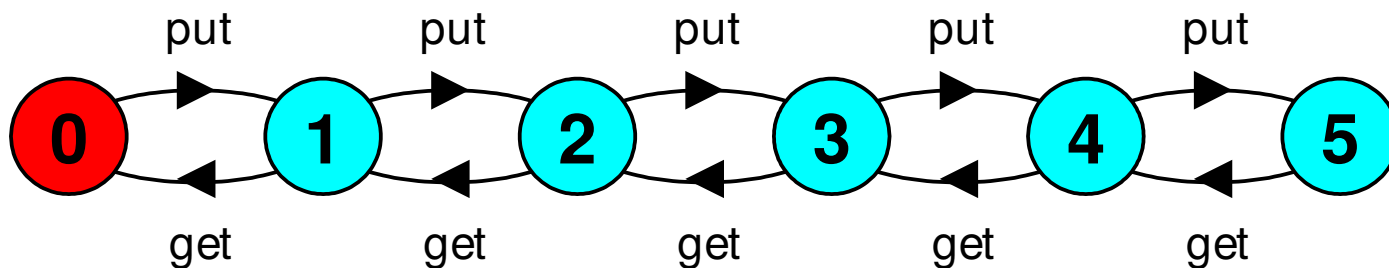
- > **idazle** prozesu batek sartzen ditu itemak, eta
- > **irakurle** prozesu batek ezabatzen ditu.



Buffer mugatua



LTS:



Buffer mugatua

```
const N = 5
range R = 0..N

IDAZLEA    = (put->IDAZLEA) .
IRAKURLEA  = (get->IRAKURLEA) .

BUFFER = BUFFER[0] ,
BUFFER[i:R] =          ( when (i<N) put->BUFFER[i+1]
                        | when (i>0) get->BUFFER[i-1]
                        ) .

|| BUFFERMUGATUA = (IDAZLEA || IRAKURLEA || BUFFER) .
```

Ariketa:

6*. Egokitu buffer mugatuaren FSP eredua
put eta get egitean, jarri eta hartu behar den posizioa adierazteko

Buffer mugatua implementatzen: Buffer monitoreo

```
class Buffer{
    //aldagai lokalak eta eraikitzailea

    public synchronized void put(char c)
        throws InterruptedException {
        while !(kont<tam) wait();
        buf[in] = c;
        ++kont;
        in=(in+1)%tam;
        pantaila.erakutsi(buf);
        notify();
    }

    public synchronized char get()
        throws InterruptedException {
        while !(kont>0) wait();
        char c = buf[out];
        buf[out]=' ';
        --kont;
        out=(out+1)%tam;
        pantaila.erakutsi(buf);
        notify();
        return(c);
    }
}
```

erakutsi() metodoak
bufferaren edukiera
erakusten duen metodoa da.

Buffer mugatua implementatzen: Idazlea haria

```
class Idazlea extends Thread {
    Buffer buf;
    Pantaila pantaila;
    String alphabet = "abcdefghijklmnopqrstuvwxyz";

    Idazlea(Buffer b, Pantaila pant) {
        buf = b;
        pantaila = pant;
    }

    public void run() {
        try {
            int ai = 0;
            while(true) {
                if (Math.random() < 0.3) sleep(1000);
                pantaila.idatzi(alphabet.charAt(ai) + ">");
                buf.put(alphabet.charAt(ai));
                ai = (ai + 1) % alphabet.length();
            }
        } catch (InterruptedException e) {}
    }
}
```

Irakurlea antzekoa izango da
`buf.get()` deituz.

5.4 Monitore habiratuak

Suposatu *kont* aldagaia eta baldintzen sinkronizazioa zuzenean erabili beharrean, bi semaforo (*okupatuak* eta *libreak*) erabiltzen ditugula bufferraren egoera kontrolatzeko.

```
class SemaBuffer{
...
    Semaforo okupatuak; // item kopurua zenbatzen du
    Semaforo libreak;   // toki libre kopurua zenbatzen du
    SemaBuffer(int tam) {
        this.tam = tam;
        buf = new char[tam];
        for (int i=0; i<tam ; i++) buf[i]= ' ';
        okupatuak = new Semaforo(0);
        libreak    = new Semaforo(tam);
    }
...
}
```

Monitore habiratuak - buffer mugatua implementatzen

```
public synchronized void put(char c)
    throws InterruptedException {
    libreak.behera();
    buf[in]=c; ++kont; in=(in+1)%tam;
    pantaila.erakutsi(buf);
    okupatuak.gora();
}
public synchronized char get()
    throws InterruptedException {
    okupatuak.behera();
    char c=buf[out];
    buf[out]=' '; --kont; out=(out+1)%tam;
    pantaila.erakutsi(buf);
    libreak.gora();
    return (c);
}
```

Ondo ibiltzen al da hau?

libreak dekrementatzen da **put** eragiketan, **libreak** zero bada blokeatuz.

okupatuak dekrementatzen da **get** eragiketan, **okupatuak** zero bada blokeatuz.

Monitore habiratuak - buffer mugatua implementatzen

```
public class Semaforo {  
    private int balioa;  
  
    public Semaforo (int hasierakoa)  
    {balioa = hasierakoa;}  
  
    synchronized public void gora() {  
        ++balioa;  
        notify();  
    }  
  
    synchronized public void behera()  
        throws InterruptedException {  
        while !(balioa>0) wait();  
        --balioa;  
    }  
}
```

Monitore habiratuak - buffer mugatuaren eredua

```
const Max = 5
range Int = 0..Max

IDAZLEA    = (put -> IDAZLEA) .
IRAKURLEA  = (get -> IRAKURLEA) .

//SEMAFOROA ...lehen bezala...

BUFFER = (put -> libreak.behera    ->okupatuak.gora ->BUFFER
          |get -> okupatuak.behera ->libreak.gora   ->BUFFER
          ) .

||BUFFERMUGATUA = (IDAZLEA || IRAKURLEA || BUFFER
                  ||libreak:SEMAFOROA(5)
                  ||okupatuak:SEMAFOROA(0)
                  )@{put,get} .
```

Ondo ibiltzen al da hau?

Monitore habiratuaren arazoa

LTSA –ak aurreikusten du

ELKAR-BLOKEAKETA (DEADLOCK) posiblea:

Composing

potential DEADLOCK

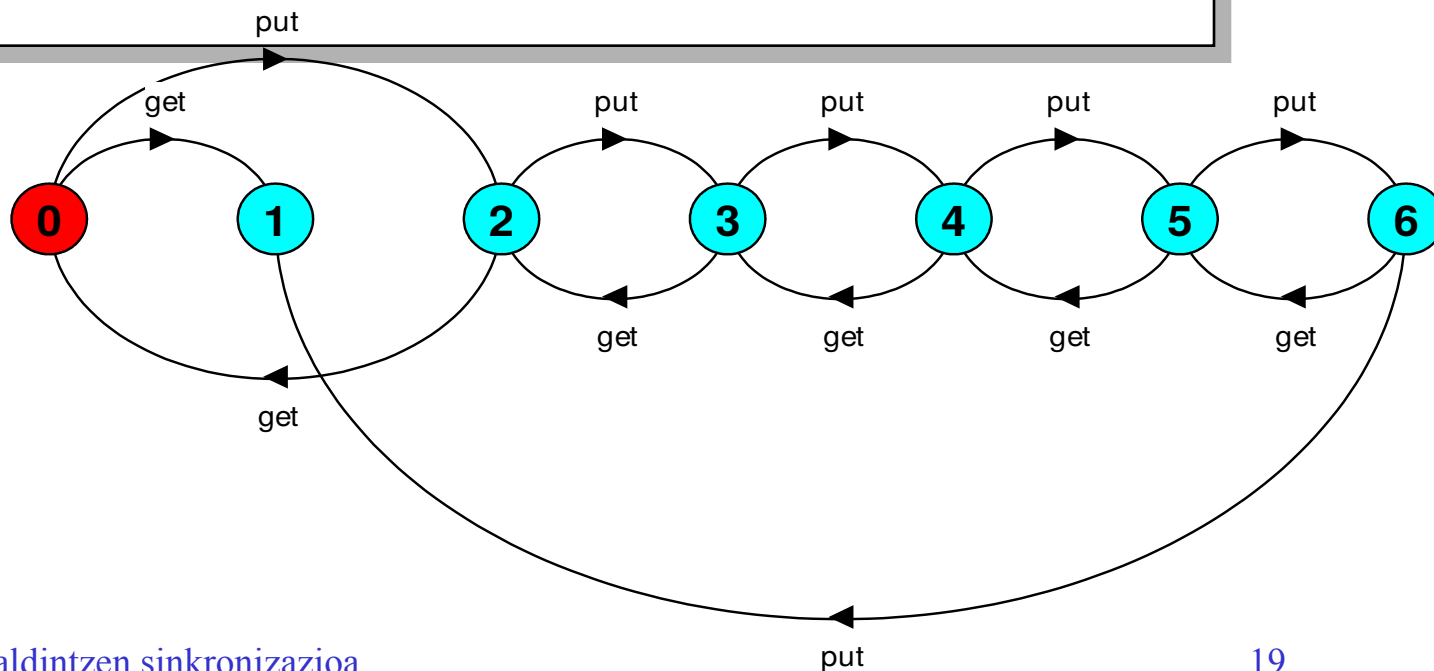
States Composed: 28 Transitions: 32 in 60ms

Trace to DEADLOCK:

get

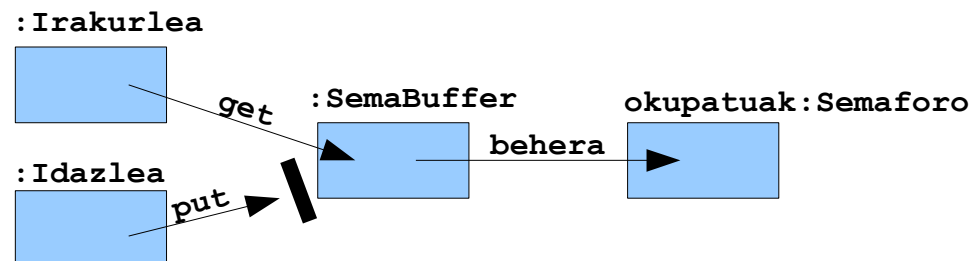
put

Egoera honi
monitore
habiratuaren
arazoa
deitzen zaio.



Monitore habiratuaren arazoa

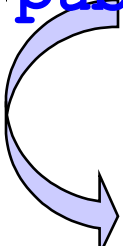
- **Irakurlea** karaktere bat hartzen (**get**) saiatzen da, **Buffer** monitorearen blokeoa eskuratzen du eta **okupatuak.behera()** deitzean **okupatuak** semaforoaren blokeoa eskuratzen du, bufferrean zerbait dagoen ikusteko.
- Hasieran **Buffer** hutsik dagoenez, **okupatuak.behera()** deiak **while (balioa == 0) wait();** eginez **Irakurlea** blokeatzen du eta **okupatuak** semaforoaren blokeoa askatzen du.
- Hala ere ez du **Buffer** monitorearen blokeoa askatzen.
- Beraz **Idazlea** ezin da sartu **Buffer** monitorean karaktere bat jartzeko, eta blokeatzen da.
- Ez **Idazlea** ez **Irakurlea** prozesuek ezin dute aurrerapenik egin.
- Elkar-blokeaketa ematen da...



Monitore habiratuak - buffer mugatua implementatzen II

Arazo hau Java-n saihesteko modu bakarra arretaz diseinatzea da. Adibide honetan elkar-blokeaketa ezabatu daiteke ziurtatzen badugu buffer monitorearen blokeoa ez dela eskuratzen semaforoak dekrementatuak izan arte.

```
public void put(char c)
    throws InterruptedException {
    libreak.behera();
    synchronized(this){
        buf[in] = c; ++kont; in=(in+1)%tam;
    }
    okupatuak.gora();
}
```



Monitore habiratuak - buffer mugatuaren eredua II

```
BUFFER = (put -> BUFFER  
          |get -> BUFFER  
          ) .
```

```
IDAZLEA =  
    (libreak.behera ->put->okupatuak.gora->IDAZLEA) .  
IRAKURLEA =  
    (okupatuak.behera->get->libreak.gora ->IRAKURLEA) .
```

Semaforoaren ekintzak jarri dira idazlean eta irakurlean (semaforoaren ekintzak monitorearen kanpoan dauden implementazioan bezala, hau da monitorearen blokeoa hartu baino lehen).

Ondo ibiltzen al da hau?

LTS minimizatua?

5.5 Monitoreen inbarianteak

Monitore baten **inbariantea** monitorearen aldagaiei buruzko baizeztapen bat da. Baieztapen hau bete behar da beti, hari bat monitore barruan egikaritzen ari denean ezik. Hau da, bete behar da hari bat monitorean sartu aurretik eta ateratzean.

Kontrolatzailea-ren inbariantea:	$0 \leq kop \leq N$
Semaforo-aren inbariantea:	$0 \leq balioa$
Buffer-aren inbariantea:	$0 \leq kont \leq tam$
	and $0 \leq in < tam$
	and $0 \leq out < tam$
	and $in = (out + kont) \% tam$

Inbarianteak lagungarri izan daitezke monitoreen zuzentasunari buruz hausnartzeko, frogapenean oinarritutako hurbilpen logikoa erabiliz. Guk erduetan oinarritutako hurbilpena erabiliko dugu frogatze mekanikoa egin ahal izateko.

Ariketa: Maximoa ukeratu

7*. Array bateko zenbakien artean maximoa aukeratu.

FSP eredua eman, horrela soluzionatuz:

Zenbakiak dituen array-az gain,

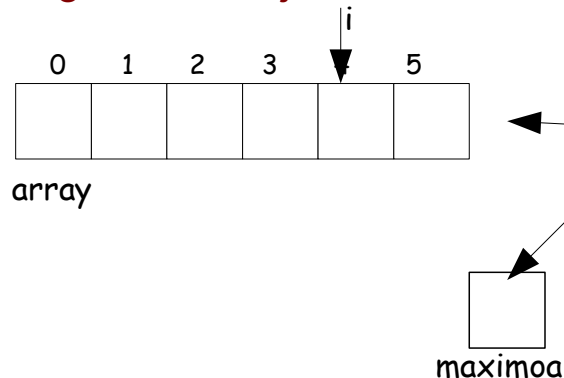
aldagai laguntzaile bat erabiliko dugu maximoa gordetzeko.

Prozesu aktiboek, konkurrenteki lanean, ondokoa egiten dute:

array-tik (hartu gabeko) zenbaki bat hartu

eta uneko maximoa baino handiagoa bada, maximoan sartu zenbaki hori.

Jakiteko zein den array-tik hartu beharreko zenbakia indize bat erabiliko dugu eguneratzen joan beharko duguna.



- Hartu i-n dagoena eta maximoa-n dagoena
- $i=i-1$
- Konparatu bi elementuak eta zenbaki berria maximoa baino handiagoa bada orduan zenbaki berria gorde maximoa-n.

Hausnartu zergatik soluzioa hau ez den batere eraginkorra.

5.6 Agendaren eredua

Agendaren ereduan,
array bateko elementuak prozesatu behar ditugunean ondokoa egiten dugu:

- Elementuak array-tik hartu
- Prozesatu hartutako elementuak
- Prozesatutako emaitza array-an sartu

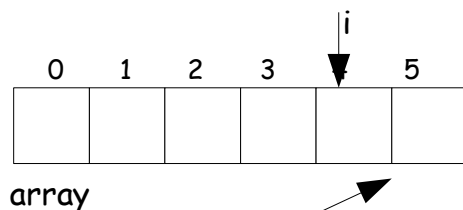
Elementu guztiak prozesatzen dira, bukatu arte.

Jakiteko nondik hartu behar diren prozesatu beharreko elementuak eta non gorde behar den soluzioa, indize bat erabiliko dugu eguneratzen jona beharko dena.

Ariketa: Maximoa aurkitu agendaren eredua erabiliz

8. Array bateko zenbakien artean maximoa aukeratu.

FSP eredua eman eta Javaz implementatu, agendaren eredua erabiliz.



- Hartu i eta $i-1$ posizioetan dauden elementuak
- $i=i-2$
- Konparatu bi elementuak
- Handiena gorde oraingo i posizioan
- $i=i+1$

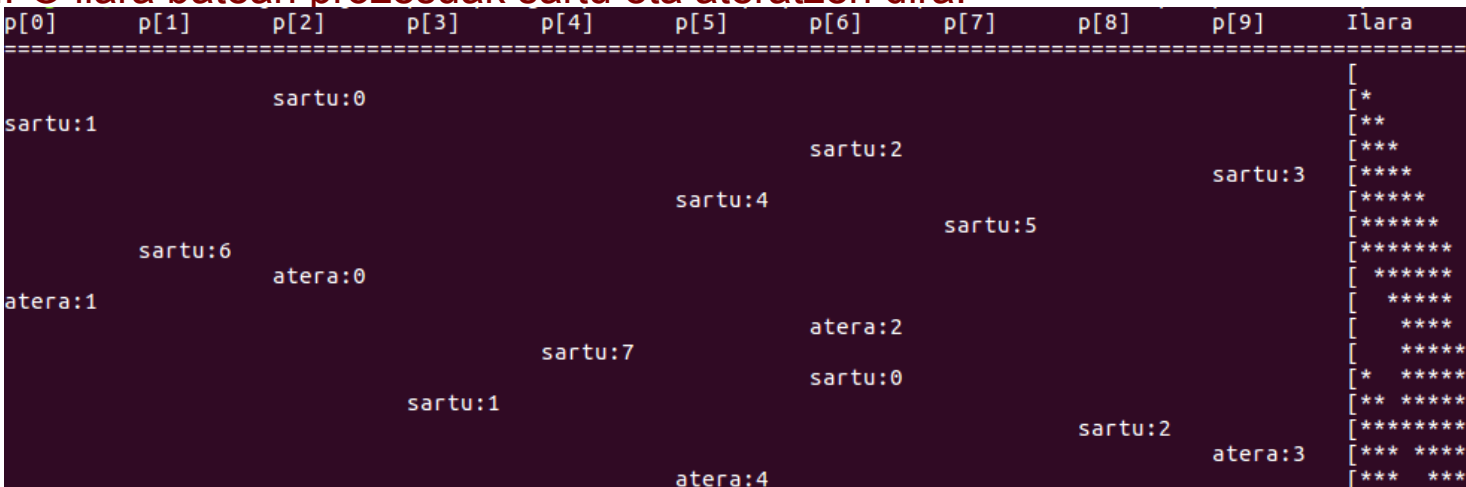
P1	P2	P3	Agenda
hartu[95] [34]			89 86 48 99 50 90 16 77 34 95
		hartu[77] [16]	89 86 48 99 50 90 16 77
sartu[95]	hartu[90] [50]		89 86 48 99
		sartu[77]	89 86 48 99 95
hartu[90] [77]	sartu[90]		89 86 48 99 95 77 90
		hartu[95] [99]	89 86 48
sartu[90]	hartu[48] [86]		89
		sartu[99]	89 90 99
hartu[86] [99]	sartu[86]		89 90 99 86
		hartu[90] [89]	89 90
sartu[99]		sartu[90]	99
	hartu[90] [99]		99 90
	sartu[99]		99
** MAXIMOA: 99 **			

Ondo pentsatu noiz bukatzen den prozesaketa.

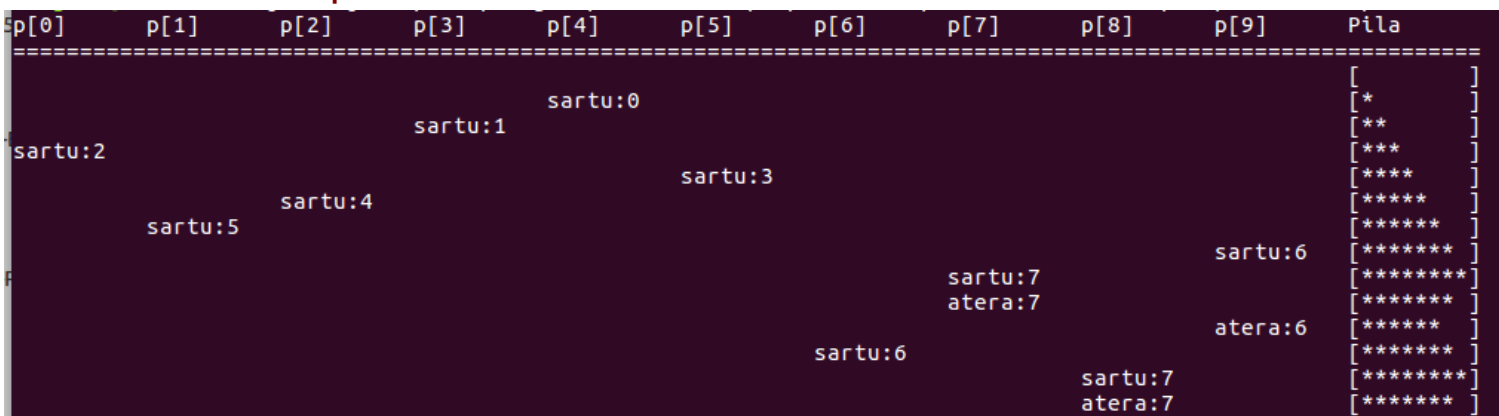
Ariketak

Ondoko problemak FSPz modelatu eta Java-z implementatu:

9. FIFO ilara batean prozesuak sartu eta ateratzen dira.



10. LIFO ilara batean prozesuak sartu eta ateratzen dira.



Ariketak

11*. Array bateko zenbakien batuketa kalkulatu,
agendaren eredua erabiliz eta
bi zenbakien arteko batuketak soilik egin ahal dituzten prozesu aktiboekin.
Javaz inplementatu,

12**. Basatien festa eroa:
Misiolariak iristean, sukaldariak akatzen ditu, zatitu, puskak hozkailuan sartu,
eta hortik lapikora..., begiratzuz beti ea tokia dagoen lapikoan, hozkailuan...