Corso di Programmazione Concorrente Produttori / Consumatori

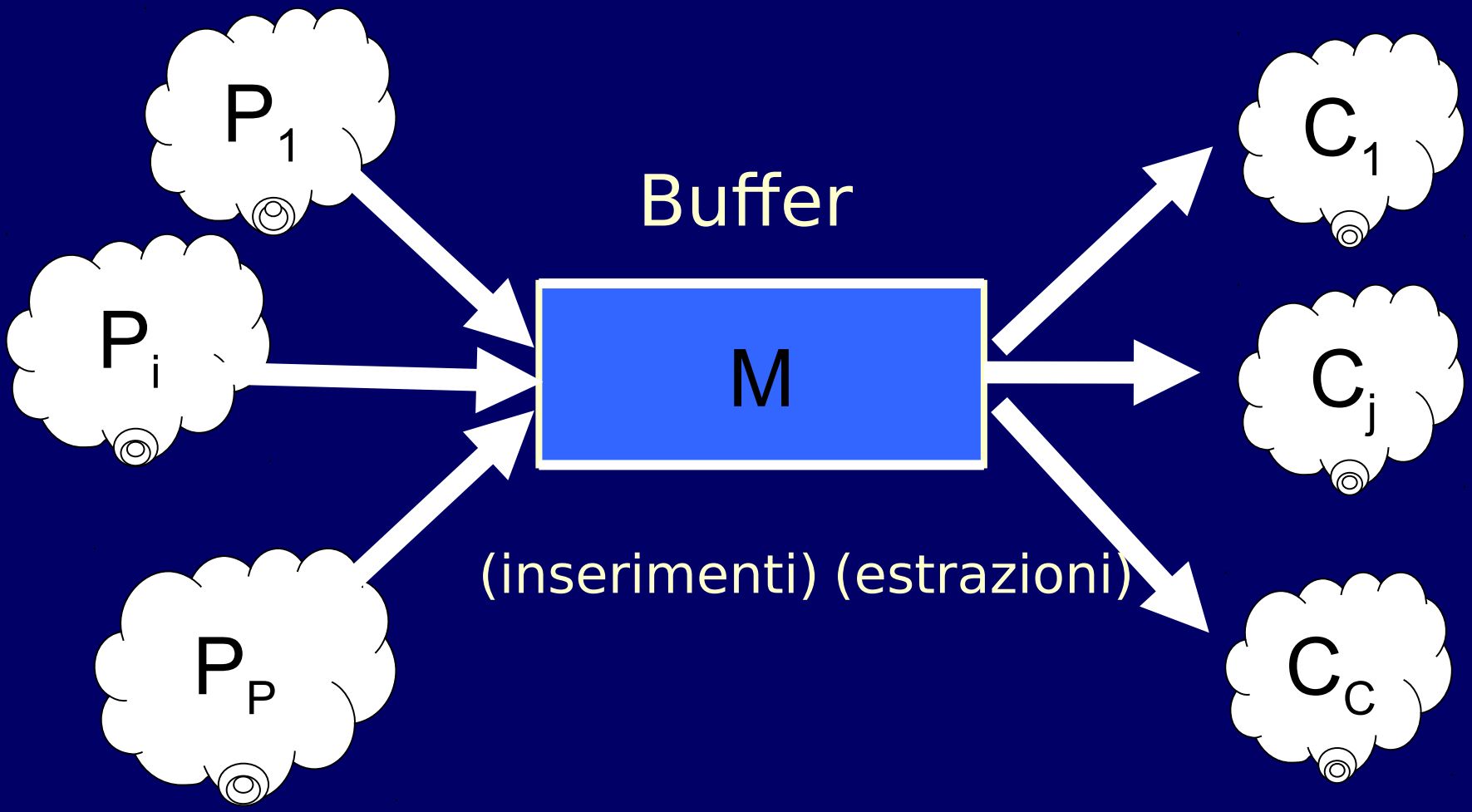
Valter Crescenzi

http://crescenzi.inf.uniroma3.it

Problema Produttori / Consumatori

- Alcuni processi produttori devono continuamente inviare messaggi ad altri processo consumatori che li elaborano nello stesso ordine in cui li ha ricevuti
- Viene utilizzato un buffer di scambio
  - I produttori scrivono i messaggi nel buffer
  - I consumatori leggono i messaggi dal buffer

Problema Produttori / Consumatori



Produttore / Consumatore: Sincronizzazioni Necessarie

- I due processi devono essere opportunamente sincronizzati contro queste eventualità:
  - Un consumatore legge un messaggio senza che un produttore ne abbia depositato alcuno
  - Un produttore sovrascrive un messaggio scritto prima che un consumatore sia riuscito a leggerlo
  - Un messaggio viene letto dai consumatori più di una volta

- Essenzialmente bisogna sincronizzarsi su due eventi a cui associamo due diversi semafori binari
  - DEPOSITATO:
    - ad 1 se e solo se un messaggio è stato depositato ed è prelevabile
  - PRELEVATO:
    - ad 1 se e solo se il buffer è vuoto e pronto ad accogliere un nuovo messaggio

Produttori / Consumatori / Buffer Unitario

```
concurrent program COOPERAZIONE;
type     messaggio=…;
var      M: messaggio;
         DEPOSITATO, PRENOTATO:semaforo_binario;
```

```
concurrent procedure PROD_i
loop begin
    <produci un messaggio in M>
    P(PRELEVATO);
    BUFFER ← M;
    V(DEPOSITATO);
end
```

```
concurrent procedure CONS_j
loop begin
    P(DEPOSITATO);
    M ← BUFFER;
    V(PRELEVATO);
    <consuma il messaggio in M>
end
```

```
begin
    INIZ_SEM(PRELEVATO,1); INIZ_SEM(DEPOSITATO,0);
     cobegin … || PROD_i || CONS_j || … coend
end
```

NproducerMconsumer1buffer.c (1)

```
…include omessi…
#define FILENAME "NproducerMconsumer1buffer.c"
#define RANGE    16
#define TIME     30
#define P        4
#define C        4
#define PRELEVATO     0
#define DEPOSITATO    1
void releaseAll(int,int);
int main(int argc, char **argv) {
  int *buffer;
  int bufferid, semid;
  int i,j,M;
  int delay, time=TIME;
  int p=0;
  pid_t pid;
  pid_t cons_pid[C];
…
```

NproducerMconsumer1buffer.c (2)

```
…
  struct sembuf sb;
  union semun { …omissis… } su;
  sb.sem_flg=0;
  // → Vedi 1producer1consumer1buffer.c ←
  /* allocate shared memory segments */
  /* allocate semaphores */
  /* initialize semaphores and
     shared segments */
…
```

NproducerMconsumer1buffer.c (3)

```
/* fork producers */
pid = -1;
for(i=0; i<P && pid!=0; i++)
  pid=fork();
switch(pid) {
case -1: //fork fallita
  exit(-5);
  break;
case 0:  /* GENERIC PRODUCER i */ …codice a seguire
  return 0;
}
/* fork consumers */
pid = -1;
for (j=0; j<C && pid!=0; j++)
  pid = cons_pid[j] = fork()
switch(pid) {
case -1: //fork fallita
  exit(-5);
  break;
case 0:  /* GENERIC CONSUMER j */…codice a seguire
  return 0;
}
```

NproducerMconsumer1buffer.c (4)

```
…  do {
     wait(NULL); p++;
   } while (p<P);  // wait producers ...

   // wait that consumers consume everything ...
   sb.sem_num=PRELEVATO; sb.sem_op=-1;
   if (semop(semid, &sb,1)==-1) {     // P(PRELEVATO)
     perror("semop() waiting for producers");
     exit(-19);
   }
   for(j=0; j<C; j++) {
     kill(cons_pid[j],SIGKILL); // ... and Kill consumers
     wait(NULL);
   }
   /* dealloca risorse */
   releaseAll(bufferid,semid);
   return 0;
}
```

NproducerMconsumer1buffer.c (5)

```
case 0: /* GENERIC PRODUCER i */
  if ((int)(buffer=shmat(bufferid,NULL,0)) == -1) {
    perror("producer: shmat() for buffer");    exit(-7);   }
  /* PRODUCING */
  srandom(i);
  // produci un messaggio in M
  while (time>0) {
    sb.sem_num=PRELEVATO; sb.sem_op=-1;
    if (semop(semid, &sb,1)==-1) {  /* P(PRELEVATO) */
      perror("semop() producer P(PRELEVATO)");  exit(-9);  }
    M = (int)(random() % RANGE);
    buffer[0] = M;                    /* PRODUCI */
    printf("Producer %d produces: %d\n",i,M);
    sb.sem_num=DEPOSITATO; sb.sem_op= 1;
    if (semop(semid, &sb,1)==-1) {  /* V(DEPOSITATO) */
      perror("semop() producer V(DEPOSITATO)"); exit(-12); }
    delay = (int)(random() % P ) / 2 + 1;
    sleep(delay);
    time -=delay;
  }
  printf("Producer %d exits\n",i);
  shmdt(buffer);
  return 0;
```

…

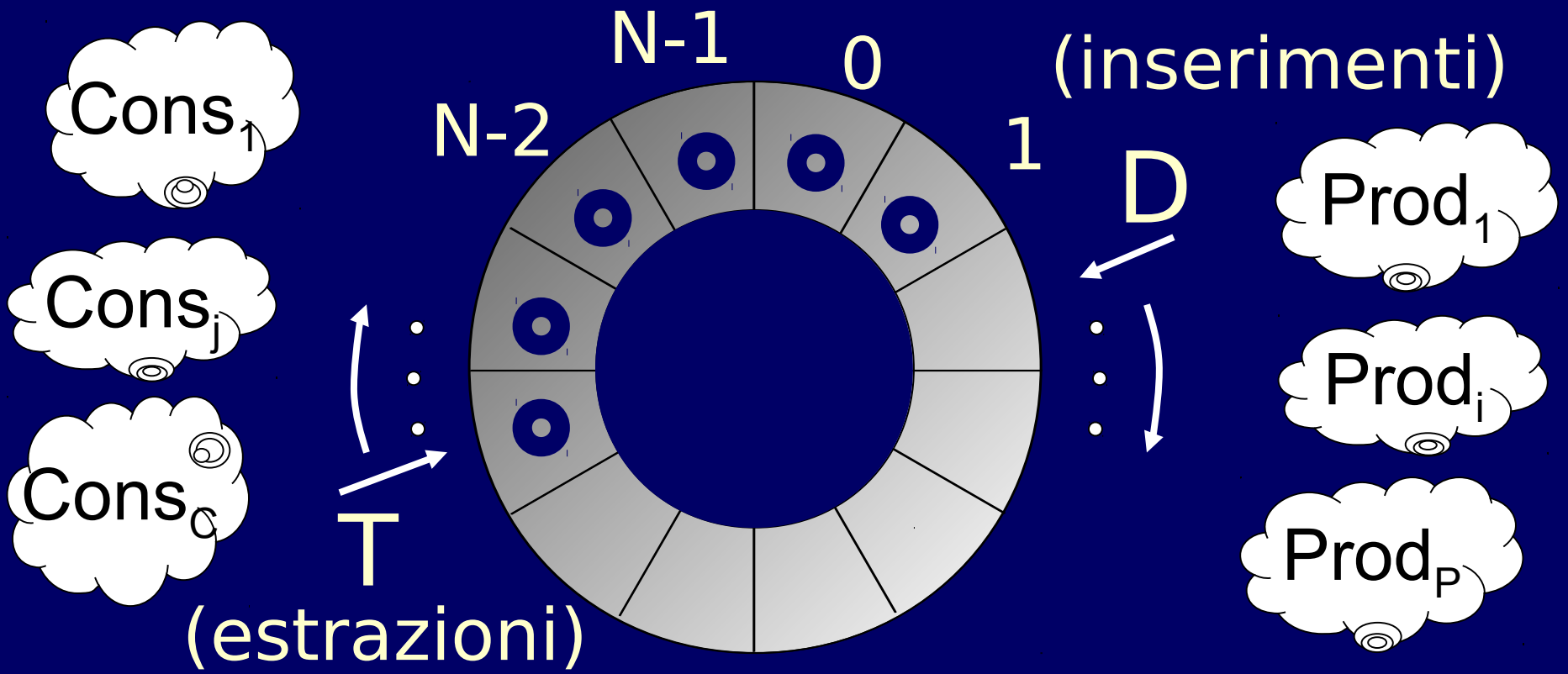
NproducerMconsumer1buffer.c (6)

```
case 0: /* GENERIC CONSUMER j */
   if ((int)(buffer=shmat(bufferid,NULL,0)) == -1) {
     perror("consumer: shmat() fo buffer");      exit(-7);  }
   /* CONSUMING */
   srandom(j);
   // consuma un messaggio in M
   while (1) {
     sb.sem_num=DEPOSITATO; sb.sem_op=-1;
     if (semop(semid, &sb,1)==-1) {  /* P(DEPOSITATO) */
       perror("semop() consumer P(DEPOSITATO)"); exit(-9);  }
     M = buffer[0];                  /* CONSUMA */
     printf("Consumer %d consumes: %d\n",j,M);
     sb.sem_num=PRELEVATO; sb.sem_op= 1;
     if (semop(semid, &sb,1)==-1) {  /* V(PRELEVATO) */
       perror("semop() consumer V(PRELEVATO)");  exit(-12); }
     delay = (int)(random() % C ) / 2 + 2;
     sleep(delay);
   }
   printf("Consumer %d exits\n",i);
   shmdt(buffer);
   return 0;
```

Esempio di Esecuzione

```
$ ./NproducerMconsumer1Buffer
Producer 1 produces: 7
Consumer 1 consumes: 7
Producer 2 produces: 10
Consumer 2 consumes: 10
…
Producer 2 exits
Consumer 3 consumes: 13
Producer 4 produces: 11
Consumer 4 consumes: 11
Producer 3 produces: 2
Consumer 2 consumes: 2
Producer 1 exits
Producer 3 exits
Producer 4 produces: 10
Consumer 1 consumes: 10
Producer 4 produces: 7
Consumer 4 consumes: 7
Producer 4 exits
$
```

Produttori / Consumatori



Buffer Circolare

PRODUTTORI_CONSUMATORI

concurrent program PRODUTTORI_CONSUMATORI;
sia N=...;
type *messaggio=...;*
indice = 0..N-1;
var BUFFER: array[*indice*] of *messaggio*;
T, D:*indice*;
PIENE, VUOTE: *semaforo*;
USO_T, USO_D: *semaforo_binario;*
concurrent procedure $PROD_i$; //generico produttore
concurrent procedure $CONS_j$; //generico consumatore
begin INIZ_SEM(USO_D,1); INIZ_SEM(USO_T,1);
INIZ_SEM(PIENE,0); INIZ_SEM(VUOTE,N);
$T \leftarrow 0$; $D \leftarrow 0$;
cobegin ... || $PROD_i$ || $CONS_j$ || ... **coend**
end

producerconsumer.c (1)

(senza controllo degli errori)

```
#include <stdio.h>
…
#define FILENAME "producerconsumer.c"
#define RANGE    16 // Range dei “messaggi M”
#define TIME     4  // vita dei produttori
#define N        4  // dimensione del buffer
#define P        2  // numero di produttori
#define C        2  // numero di consumatori

#define VUOTE    0  // indice del semaforo VUOTE
#define PIENE    1  // indice del semaforo PIENE
#define USO_T    2  // indice del semaforo USO_T
#define USO_D    3  // indice del semaforo USO_D

void releaseAll(int,int,int,int); // rilascia tutto
void printBuffer(int,int *,int);  // stampa buffer corrente
…
```

producerconsumer.c (2)

```
…
int main(int argc, char **argv) {
  int *buffer,*T,*D;              // segmenti di memoria condivisa
  int bufferid, Tid, Did, semid; // id per memoria e semafori
  int i,j,M;
  int delay, time=TIME;
  int p=0;                        // contatori di produttori morti

  struct sembuf sb;
  union semun {
    int val;                    /* valore SETVAL */
    struct semid_ds *buf;       /* buffer per IPC_STAT, IPC_SET */
    unsigned short int *array; /* array per GETALL, SETALL */
    struct seminfo *__buf;      /* buffer per IPC_INFO */
  } su;
  pid_t pid;
  pid_t cons_pid[C];            // pid dei consumatori

  sb.sem_flg = 0;
…
```

producerconsumer.c (3)

```
…
  /* alloca memoria condivisa e semafori */
  bufferid = shmget(ftok(FILENAME,'B'),N,IPC_CREAT|0666);
  Tid = shmget(ftok(FILENAME,'T'),1,IPC_CREAT|0666);
  Did = shmget(ftok(FILENAME,'D'),1,IPC_CREAT|0666);
  semid = semget(ftok(FILENAME,'S'),5,IPC_CREAT|0666);

  /* inizializza memoria condivisa e semafori */
  T = shmat(Tid,NULL,0);
  D = shmat(Did,NULL,0);
  *T = 0; *D = 0;
  shmdt(T); shmdt(D);
  su.val = 1;
  semctl(semid,USO_D,SETVAL,su);
  semctl(semid,USO_T,SETVAL,su);
  su.val = 0;
  semctl(semid,PIENE,SETVAL,su);
  su.val = N;
  semctl(semid,VUOTE,SETVAL,su);
…
```

producerconsumer.c (4)

```
…
  pid = -1;
  for(i=0; i<P && pid!=0; i++) pid = fork();
  switch(pid) {
    case -1: …
    case  0: /*< GENERICO PRODUTTORE i >*/
  }

  pid = -1;
  for (j=0; j<C && pid!=0; j++) pid = cons_pid[j] = fork();
  switch(pid) {
    case -1: …
    case  0: /*< GENERICO CONSUMATORE j >*/
  }

  do { wait(NULL); p++; } while (p<P); // aspetta produttori …
  sb.sem_num = VUOTE; sb.sem_op=-N;    // aspetta consumatori …
  semop(semid, &sb,1);
  for(j=0; j<C; j++) kill(cons_pid[j],SIGKILL); //…Kill consumatori

  releaseAll(bufferid,Tid,Did,semid); // rilascia risorse allocate
  return 0;
}
…
```

$PROD_i$

```
concurrent procedure PROD_i; //generico produttore
var M: messaggio;
  loop
        <produci un messaggio in M>
        P(VUOTE);
              P(USO_D);
                    BUFFER[D] ← M;
                    D ← (D+1) mod N;
              V(USO_D);
        V(PIENE);
  end_loop
```

producerconsumer.c (5)

```
… case 0: /* GENERICO PRODUTTORE i */
buffer = shmat(bufferid,NULL,0);
D = shmat(Did,NULL,0); T = shmat(Tid,NULL,0);
while (time>0) {
  sb.sem_num = VUOTE; sb.sem_op=-1; semop(semid, &sb,1); /* P(VUOTE) */
  sb.sem_num = USO_D; sb.sem_op=-1; semop(semid, &sb,1); /* P(USO_D) */
  M = (int)(random() % RANGE);
  buffer[*D] = M;                    /* PRODUCI */
  *D = (*D + 1) % N;
  printf("Producer %d produces: %d\n",i,M);
  printBuffer(semid,buffer,*T);
  sb.sem_num = USO_D; sb.sem_op= 1; semop(semid, &sb,1); /* V(USO_D) */
  sb.sem_num = PIENE; sb.sem_op= 1; semop(semid, &sb,1); /* V(PIENE) */
  delay = (int)(random() % P ) / 2 + 1;
  sleep(delay);
  time -=delay;
}
printf("Producer %d exits\n",i);
shmdt(buffer); shmdt(D);
return 0;
```

$CONS_j$

```
concurrent procedure CONS_j; //generico consumatore
var M: messaggio;
  loop
        P(PIENE);
              P(USO_T);
                    M ← BUFFER[T];
                    T ← (T+1) mod N;
              V(USO_T);
        V(VUOTE);
        <consuma il  messaggio in M>
  end_loop
```

producerconsumer.c (6)

```
… case 0:/* GENERICO CONSUMER j */
buffer = shmat(bufferid,NULL,0);
T = shmat(Tid,NULL,0);
while (1) {
 sb.sem_num = PIENE; sb.sem_op=-1; semop(semid, &sb,1); /* P(PIENE) */
 sb.sem_num = USO_T; sb.sem_op=-1; semop(semid, &sb,1); /* P(USO_T) */
 M = buffer[*T];                  /* CONSUMA */
 *T = (*T + 1) % N;
 printf("Consumer %d consumes: %d\n",j,M);
 printBuffer(semid,buffer,*T);
 sb.sem_num = USO_T; sb.sem_op= 1; semop(semid, &sb,1); /* V(USO_T) */
 sb.sem_num = VUOTE; sb.sem_op= 1; semop(semid, &sb,1); /* V(VUOTE) */
 delay = (int)(random() % C ) / 2 + 2;
 sleep(delay);
}
shmdt(buffer); shmdt(T);
return 0;
}
```

Esempio di Esecuzione

```
Producer 1 produces: 7
><
Producer 2 produces: 10
>7<
Consumer 1 consumes: 7
>10<
Consumer 2 consumes: 10
><
Producer 2 produces: 4
><
Producer 1 produces: 9
>4<
Producer 1 produces: 1
>4 9<
Producer 2 produces: 5
>4 9 1<
Consumer 2 consumes: 4
>9 1 5<
```

```
Consumer 1 consumes: 9
>1 5<
Producer 2 produces: 0
>1 5<
Producer 1 produces: 10
>1 5 0<
Consumer 1 consumes: 1
>5 0 10<
Consumer 2 consumes: 5
>0 10<
Producer 1 exits
Producer 2 exits
Consumer 2 consumes: 0
>10<
Consumer 1 consumes: 10
><
```

Esercizi

Esercizio: riscrivere il programma `produttoreconsumatore.c` utilizzando programmi distinti per il produttore e per il consumatore oltre ad un programma principale

Esercizio: dopo aver svolto l'esercizio precedente riscrivere il programma relativo ai consumatori di modo che catturino esplicitamente un segnale lanciato dal programma principale per segnalare la fine delle operazioni e quindi possano terminare autonomamente (senza essere uccisi dal padre).

Esercizio

Esercizio: Cinque filosofi trascorrono la vita alternando, ciascuno indipendentemente dagli altri, periodi in cui pensano a periodi in cui mangiano degli spaghetti.

Per raccogliere gli spaghetti ogni filosofo necessita delle due forchette poste rispettivamente a destra ed a sinistra del proprio piatto.
Simulare questo sistema modellando ogni filosofo con un processo.