

Meccanismi di Comunicazione Interprocesso (IPC)

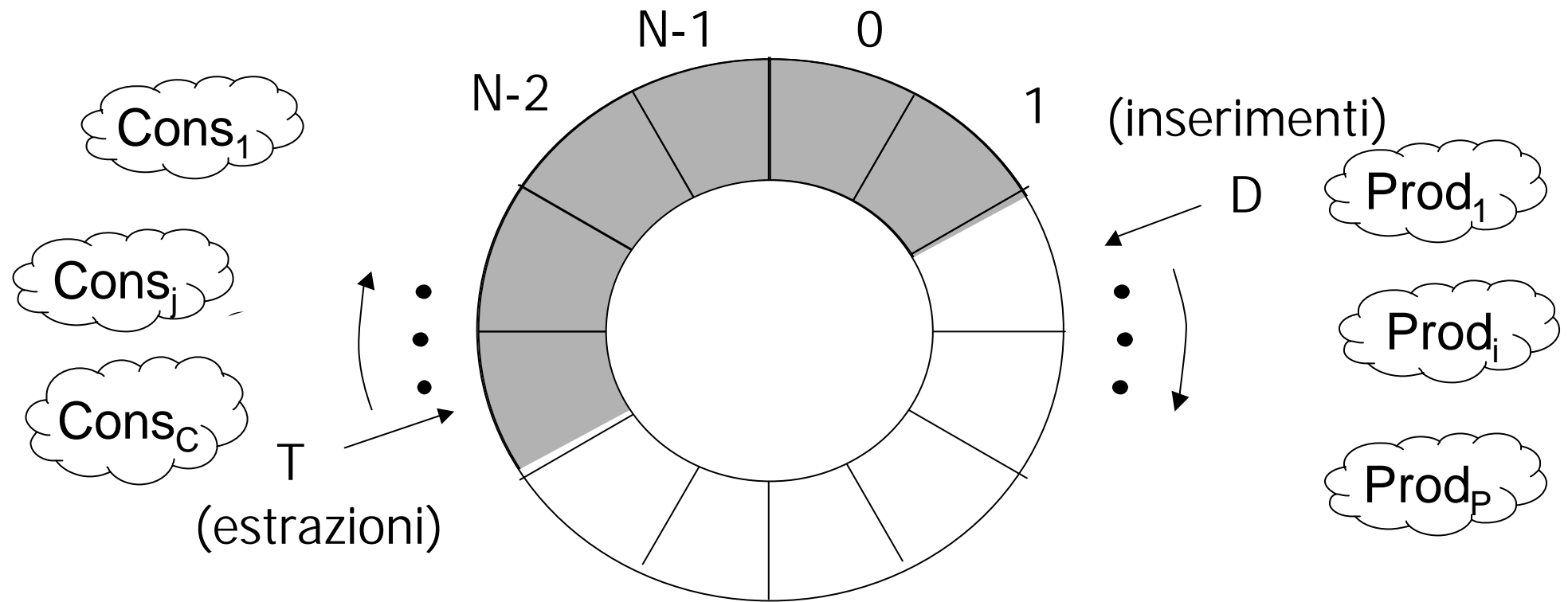
- segnali
 - pipe (anonymous o FIFO)
 - shared memory
 - socket
- ⇒ sincronizzazione tra processi
- semafori

Semafori (Dijkstra)

- Variabili Intere
- Oggetto di due Primitive di Sincronizzazione

```
P(S): begin  $S \leftarrow S - 1;$   
           if  $S < 0$  then <poni il processo in una  
                           coda di attesa passiva  $Q_s$ >  
           end  
V(S): begin  $S \leftarrow S + 1;$   
           if  $S > 0$  then <risveglia se esiste, un  
                           processo di  $Q_s$ >  
           end
```

Produttori / Consumatori



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 PRODi; { generico produttore}  
concurrent procedure CONSj; { generico consumatore}  
begin INIZ_SEM(USO_D,1); INIZ_SEM(USO_T,1);  
        INIZ_SEM(PIENE,0); INIZ_SEM(VUOTE,N);  
        T ← 0; D ← 0;  
        cobegin ... || PRODi || CONSj || ... coend  
end
```

PROD_i

```
concurrent procedure PRODi; {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
```

CONS_j

```
concurrent procedure CONSj; {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
```

Semafori in Linux

- `semget ()`
per ottenere un *vettore* di semafori
- `semop ()`
per eseguire le normali operazioni sui
semafori corrispondenti alle primitive di
sincronizzazione
- `semctl ()`
per varie operazioni di controllo, inclusa la
rimozione di un vettore di semafori

semget ()

```
#include <sys/types.h>
```

```
#include <sys/ipc.h>
```

```
#include <sys/sem.h>
```

man 2 semget

```
int semget(key_t key, int nsems, int semflg)
```

- restituisce un identificatore intero univocamente associato a key
- viene creato un vettore di nsems semafori se
 - key è IPC_PRIVATE
 - key non è associato a nessun vettore di semafori preesistenti e semflg è IPC_CREAT
- semflg contiene 9 bit di permessi ed eventualmente i flag
 - IPC_CREAT
 - IPC_EXCL (creazione in esclusiva)
- restituisce -1 in caso di errore

semop ()

```
#include <sys/types.h>
```

```
#include <sys/ipc.h>
```

```
#include <sys/sem.h>
```

man 2 semop

```
int semop(int semid, struct sembuf *sops, unsigned nsops)
```

- esegue `nsops` operazioni sul vettore di semafori `semid`
- ciascuna operazione è specificata da una struttura `sembuf` contenente almeno:

```
short sem_num; /* semaphore number: 0 = first */
```

```
short sem_op; /* semaphore operation */
```

```
short sem_flg; /* operation flags */
```

- `sem_flg` può assumere i flag
 - `IPC_NOWAIT` (causa fallimenti della chiamata anziché attese)
 - `IPC_UNDO` (se un processo esegue `exit()` viene eseguito l'*undo* di tutte le modifiche che ha apportato al vettore di semafori con questo flag attivo)
- restituisce 0 in caso di successo e -1 in caso di errore

semop () : operazioni

```
int semop(int semid, struct sembuf *sops, unsigned nsops)
```

Il campo **sem_op**

- > 0 **sem_op** viene sommato al valore del semaforo. In questo modo intende evidenziare che una risorsa protetta da questo semaforo è stata da lui rilasciata
- < 0 Se il valore assoluto di **sem_op** è maggiore del valore del semaforo, il processo viene bloccato sino a quando il semaforo raggiunge il valore assoluto di **sem_op**. Quindi questo valore assoluto viene sottratto al valore del semaforo. In questo modo intende evidenziare che una risorsa protetta da questo semaforo è stata da lui allocata
- Zero Il processo viene sospeso sino a quando il semaforo torna a 0

semctl ()

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
```

man 2 semctl

```
int semctl(int semid, int semnum, int cmd, union semun arg)
```

- esegue operazioni di controllo sul vettore di semafori `semid`
- ciascuna operazione è specificata da `cmd` ed utilizza una struttura unione (su alcune installazioni va dichiarata esplicitamente):

```
union semun {
    int val; /*value for SETVAL */
    struct semid_ds *buf; /*buffer for IPC_STAT,IPC_SET*/
    unsigned short int *array; /*array for GETALL,SETALL*/
    struct seminfo *__buf; /*buffer for IPC_INFO */
};
```

- restituisce `-1` in caso di errore ed in caso di successo un valore non negativo dipendente da `cmd`

semctl () : comandi

```
int semctl(int semid, int semnum, int cmd, union semun arg)
```

- valori permessi per cmd:
 - IPC_RMID rimuove il vettore di semafori
 - GETALL restituisce il valore dei semafori dentro
arg.array
 - GETVAL restituisce il valore del semaforo semnum
 - GETZCNT restituisce il numero di processi in attesa
del semaforo semnum
 - SETALL fissa il valore di tutti i semafori del vettore
usando arg.array
 - SETVAL fissa il valore del semaforo semnum to
arg.val
 - GETNCNT...
 - GETPID...

man 2 semctl

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); // release all resources
```

```
void printBuffer(int,int *,int); // print current buffer
```

```
...
```

producerconsumer.c (2)

```
...
int main(int argc, char **argv) {
    int *buffer,*T,*D;                // shared memory segments
    int bufferid, Tid, Did, semid;    // id shared buffer and sem.
    int i,j,M;
    int delay, time=TIME;
    int p=0;                          // counter for dead producers

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

    sb.sem_flg = 0;
    ...
```

producerconsumer.c (3)

```
...
/* allocate shared memory segments and semaphores */
bufferid = shmget(ftok(FILENAME, 'B'), N, IPC_CREAT | 0666);
Tid = shmget(ftok(FILENAME, 'D'), 1, IPC_CREAT | 0666);
Did = shmget(ftok(FILENAME, 'T'), 1, IPC_CREAT | 0666);
semid = semget(ftok(FILENAME, 'S'), 5, IPC_CREAT | 0666);

/* initialize semaphores and shared segments */
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: /*< GENERIC PRODUCER i >*/
}

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

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

releaseAll(bufferid,Tid,Did,semid); // release resources
return 0;
}
...
```

producerconsumer.c (5)

```
... case 0: /* GENERIC PRODUCER 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; /* PRODUCE */
    *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;
```

producerconsumer.c (6)

```
... case 0:/* GENERIC 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]; /* CONSUME */
    *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 Output

```
[valter.../prove]$ ./producerconsumer
```

```
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 provvedano al rilascio della memoria condivisa

Esercizio: Eseguire e stampare il prodotto di due matrici in parallelo: ciascun elemento della matrice risultante è calcolato da un processo dedicato.

Esercizi

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.

