Interfacce Unix per I Semafori

- Esistono due interfacce per i semafori tra processi:
  - Standard XSI (X/Open System Interfaces) antenato di UNIX System V
  - POSIX
- Per usare i semafori POSIX con i processi è sufficiente allocarli in memoria condivisa e all'inizializzazione usare:
    - *sem_init(&sem, 1, …);*

  con secondo parametro il valore 1
- Vediamo assieme l'esempio race_sem.c in appunti_6a

Interfacce Unix per I Semafori

- In XSI i semafori sono:
  - Persistenti: possono sopravvivere ai processi che li hanno creati
  - associati ad un utente e gruppo proprietario
  - corredati di permessi di accesso
    - al proprietario, agli appartenenti al gruppo proprietario, a tutti gli altri utenti (stesso tipo di controllo degli accessi utilizzato per i file)
- Si possono vedere e rimuovere i semafori da linea di comando con:
  - ipcs – lista dei semafori allocati all'utente
  - ipcrm -s <id> rimuove il semaforo con identificativo <id>

Semafori nell'estensione XSI

- Un processo può chiedere che venga allocato un "array" di semafori, con semget (SEMaphore GET):

id = semget(key, nsems, flags)

- restituisce l'identificatore di un vettore di *nsems* semafori associati alla chiave *key* (il terzo parametro *semflg* serve per altre opzioni
- Se si usa IPC_PRIVATE come chiave, si ha la garanzia che il vettore sia nuovo, e può essere condiviso da tutti i processi parenti (attraverso l'identificatore restituito dalla semget)
- In alternativa i processi devono accordarsi sull'uso di una stessa chiave la cui unicità deve essere gestita dall'utente
- Gli *nsems* semafori così allocati possono essere usati dai processi che conoscono il valore di *id*
- Con *id* e un intero *semnum* compreso fra 0 e *nsems* - 1 si identifica un singolo semaforo dell'array

Operazioni sui semafori XSI

- La chiamata di sistema *semctl* effettua diverse operazioni sul semaforo:

  int semctl(int semid, int semnum, int cmd, ...);

  esegue il comando specificato da *cmd* sul semaforo identificato da *semid* o in qualche caso da *semnum*.

Operazioni sui semafori XSI

- I comandi più rilevanti sono:
  - GETVAL : restituisce il valore della componente semnum;
  - GETNCNT : restituisce il numero di processi in attesa
  - IPC_RMID rimuove il vettore di semafori semid
  - SETVAL : assegna alla componente semnum il valore passato nel 4° argomento che è del tipo:

```
union semun {
        int val;
        struct semid_ds *buf;
        ushort *array;
    }
```

dove il valore da assegnare è settato in *val*

Operazioni sui semafori XSI

- Tramite la chiamata di sistema *semctl,* possiamo definire una funzione di inizializzazione di un semaforo:

  int seminit(int semid, int semnum, int initval);

- che assegna il valore *initval* all'elemento *semnum* di *semid*.
- Questa funzione può essere realizzata con la seguente sequenza di operazioni:

  arg.val=initval;

  r=semctl(semid,semnum,SETVAL,arg);

- La *semctl* può essere utilizzata anche per altre operazioni (es. rimozione del vettore di semafori): la funzione svolta è determinata dal valore del 3° parametro.

  r=semctl(semid,0,IPC_RMID); // elimina l'array di semafori

Operazioni sui semafori XSI

- Possiamo inoltre definire le operazioni di up e down:

  down(int semid, int semnum)

  up(int semid, int semnum)

- Entrambe operano sull'elemento *semnum* del semaforo *semid*
- Per la loro implementazione si usa la chiamata di sistema *semop* che necessita l'utilizzo di una struct *sembuf*:

```
struct sembuf sb;
sb.sem_num=semnum; /* quale elem. del vettore */
sb.sem_op=-1 (o =1)/* si cerca di decr./incr. di 1 il sem. */
sb.sem_flg=0; /* ignoriamo, vedere il man */
```

Operazioni sui semafori XSI

- Ad esempio possiamo definire la *down()* come:

```
int down(int semid, int semnum)
{
    struct sembuf sb; int r;
    sb.sem_num=semnum; /* quale elem. del vettore   */
    sb.sem_op=-1; /* si cerca di decrementare di 1 il sem. */
    sb.sem_flg=0; /* ignoriamo, vedere il man */
    r=semop(semid,&sb,1); /* 1 una sola operazione */
    if (r==-1) perror("semop in down");
    return r
}
```

- La *up* è analoga, con *sb.sem_op=1*, incremento di 1 del semaforo

Operazioni sui semafori XSI

- Infatti il valore -1 viene interpretato così da semop:
  - è un tentativo di decrementare di 1 il valore semval del semaforo
  - viene effettuato se il decremento non rende semval negativo (cioé: se semval > 0)
  - se tuttavia il decremento dovesse rendere semval negativo (cioé: se semval = 0) l'operazione non viene effettuata e il processo viene sospeso
- Fino a quando?
- Il valore 1 passato a semop nella funzione up ha il significato di incrementare semval di 1.
- Se semval valeva 0, e c'erano dei processi sospesi a causa di una semop (down) l'effetto collaterale di questo incremento è di risvegliare **uno** dei processi sospesi e di far sì che effettui il decremento di 1
- Nel complesso si ottiene il comportamento di down/up

- Scaricare codice semafori per processi (appunti6a)
- Esaminiamo assieme sem.c e semfun.c
- Verificate la persistenza dei semafori:
  - Eseguite sem.c (eventualmente modificatelo per rallentarlo) ed interrompetelo prima del suo completamento
  - Verificate tramite l'opportuno comando che il semaforo è ancora allocato dopo l'interruzione
  - Cancellate il semaforo tramite l'opportuno comando

- Modificare sem.c affinchè, nel caso riceva un segnale SIGINT, cancelli il semaforo prima di uscire
- A partire da sem.c modificare i due rami dei due processi in modo da garantire l'esecuzione in mutua esclusione di una sezione del loro codice, che simula una "regione critica"

  Simulate una regione critica in ciascun ramo come:

```
stampa "inizio regione critica del processo getpid()"
sleep(5)
stampa "fine regione critica del processo getpid()"
```

- Eseguite codasem.c per verificare l'ordine seguito per il risveglio dei processi (non è garantito, dipende dall'implementazione)

Operazioni sui semafori XSI

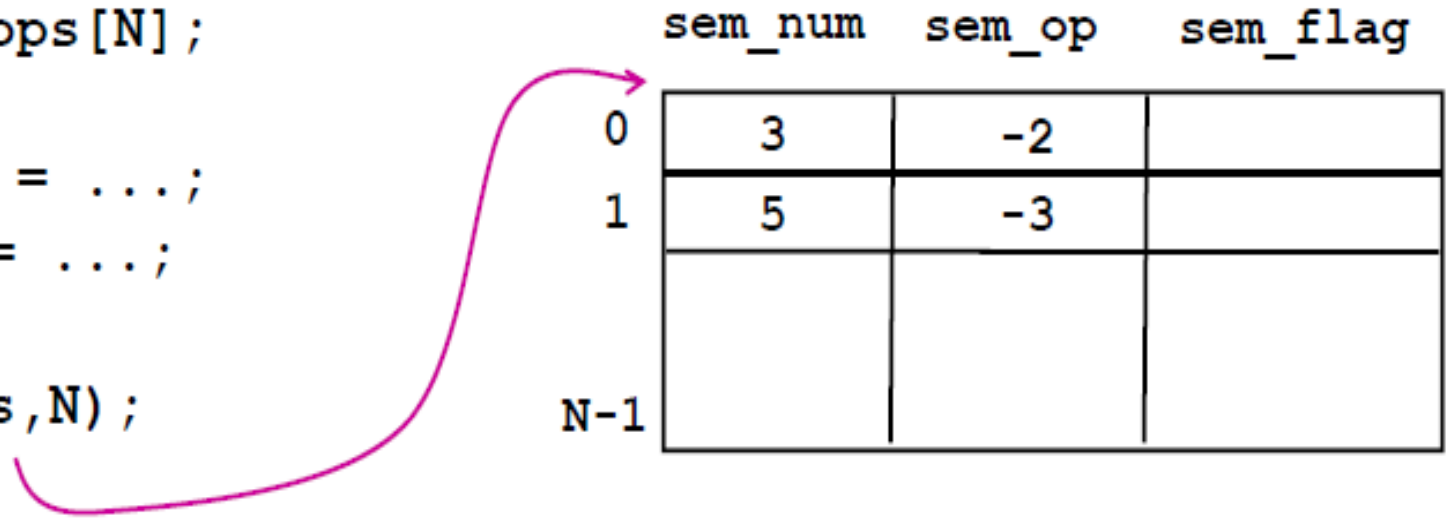
- La *semop* è molto più generale di down/up perché
  - Può incrementare/decrementare il valore del semaforo di n (n >=1)
  - Può operare simultaneamente su più semafori dello stesso vettore
- Naturalmente se si tenta di decrementare di una quantità superiore al valore attuale del semaforo il processo viene sospeso
- L'operazione simultanea su più semafori è bloccante se vi è almeno un decremento che non può essere portato a termine a causa del valore attuale del semaforo.

Operazioni sui semafori XSI

- Alla funzione semop si può passare nel campo sem_op anche:
  - 2,3,4...
  - -2, -3,..
- A *semop* si può passare un vettore di operazioni che vengono eseguite in modo atomico, se e quando possono essere eseguite tutte (quando nessun semaforo coinvolto ha un valore che può causare sospensione)

```
struct sembuf sops[N];

sops[0].sem_num = ...;
sops[0].sem_op = ...;
...
semop(semid,sops,N);
```



Esempio: doppia down

- Due semafori distinti
  - `semnum1` e `semnum2` di uno stesso array di identificatore `semid`
- Sospende il processo che la esegue
  - se almeno uno dei due semafori ha valore nullo

```
int doppiadown(int semid,int semnum1,int semnum2)
{
        struct sembuf sb[2];
        int r;

        sb[0].sem_num=semnum1; /* quale elem. del vettore
        sb[0].sem_op=-1; /* si cerca di decrementare di 1
        sb[0].sem_flg=0; /* ignoriamo, vedere il man */
        sb[1].sem_num=semnum2; /* quale elem. del vettore
        sb[1].sem_op=-1; /* si cerca di decrementare di 1
        sb[1].sem_flg=0; /* ignoriamo, vedere il man */

        r=semop(semid,sb,2); /* 2 = due down inglobate in un'unica
operazione  atomica*/
        if (r==-1) perror("semop in doppiadown");
        return r
}
```

Esercizi

- Esaminiamo il programma filodead.c con soluzione tramite semafori POSIX
- Modificare la soluzione in modo che non sia soggetta a deadlock usando
  - la soluzione con più operazioni atomiche usando semafori XSI
  - la soluzione con semafori POSIX privati vista nelle lezioni di teoria