Corso di Laboratorio di Sistemi Operativi

Lezione 6

Alessandro Dal Palù

email: alessandro.dalpalu@unipr.it

web: www.unipr.it/~dalpalu

Interazione tra Processi

I processi concorrenti possono interagire

- per cooperare (obiettivi comuni, condivisione delle informazioni, accelerazione del calcolo, modularità)

- per competere (risorse condivise).

Il sistema operativo deve fornire strumenti per la comunicazione e la sincronizzazione.

Interazione tra processi: Strumenti Linux

In Linux ci sono diverse astrazioni per la comunicazione e la sincronizzazione:

- Unix pipe: Comunicazione tra 2 processi tramite stream
- Unix RPC: Chiamata a funzione su processo remoto attraverso la rete
- Unix Signals: Notifica di eventi asincroni tra processi
- BSD Socket: Stream tra 2 processi attraverso la rete
- SysV IPC (83) InterProcess Communication:
  - Sezioni di memoria condivisa (`shmget()` `shmctl()` ....)
  - Sincronizzazione con semafori (`semget()` `semctl()` ...)
- Pthread:
  - Comunicazione a memoria condivisa tra piu thread
  - Sincronizzazione con semafori (pthread_mutex_*())

Unix pipe

- Connettono lo standard output di un processo allo standard input di un altro.
- Per esempio: `ls | sort`
- In C (unistd.h) si programmano con il comando `int pipe(int fd[2]);`
- Il comando crea una coppia di FD: fd[0] è usato per leggere dalla pipe e fd[1] è usato per scrivere sulla pipe.
- Una chiamata `fork()` sdoppia i processi e il processo figlio eredita i fd del padre.
- La syscall `dup2(old_fd, new_fd)` sovrascrive `old_fd` con una copia di `new_fd`, questo consente di redirezionare un flusso.

Unix pipe - Esercizio

- Scaricare dalla pagina del corso il file `pipe.c`
- Studiare, compilare e lanciare
- Modificare il file: il processo figlio manda alla pipe del testo mediante `cout / printf`, il testo viene prelevato da standard input tramite `cin / scanf`
- Il processo padre legge da pipe e scrive a video un carattere alla volta (usare la system call `read()` sul file di standard input) finché non appare il carattere ".”
- Tempo 45 minuti

InterProcess Communication

- Capitolo 11 di `gapil.gnulinux.it/files/2011/12/gapil.pdf`
- Ogni oggetto IPC ha un identificatore (ID).
- Due processi che usano un oggetto IPC devono condividere una chiave, la quale serve per ottenere l'ID.
- `ftok()` è usata dai processi per generare in modo (quasi) univoco un valore di chiave:
  ```
  key_t ftok ( char *pathname, char id );
  ```
  Il valore chiave viene generato combinando inode number del primo parametro con il valore a 8 bit di `id`. Esempio:
  ```
  key_t mykey = ftok(".", 'a');
  ```
- In questo caso, tutti i processi eseguiti dalla stessa directory, ottengono la stessa chiave.

Memoria condivisa

- Per ottenere l'ID di un segmento di memoria condivisa:
  `int shmget ( key_t key, size_t size, int flags);`

- `key` è la chiave sotto cui si condivide il segmento, e `size` è la dimensione in byte del segmento

- `IPC_CREAT` è un flag che specifica la creazione del segmento. Se non è utilizzato, il sistema recupera l'eventuale segmento creato da qualcun'altro e lo rende disponibile.

- Altri flags (da combinare con |) specificano i permessi di accesso (come con i files, utilizzano solo r w)

- Valore di ritorno: -1 errore, altrimenti `shmid` (identificatore del segmento).

Controllo IPC

- Ogni meccanismo IPC ha una chiamata di controllo
  `int <ipc>ctl (int ipcid, [int ipcnum], int cmd, ...)`

- Con cmd=`IPC_STAT` si ottengono informazioni sull'oggetto

- Con cmd=`IPC_SET` si impongono delle informazioni.

- Con cmd=`IPC_RMID` si dealloca l'oggetto.

- Se l'oggetto IPC *non* viene deallocato, il kernel lo considera attivo anche se tutti i processi che lo stavano usando sono terminati.

- E' quindi possibile avere comunicazioni che si svolgano anche se i processi interlocutori sono vivi in momenti diversi.

- Se però un oggetto IPC rimane attivo per errore, questo impegna risorse finchè non si rimuove manualmente o si riavvia la macchina.

Memoria condivisa - Attach

- E' possibile allocare dinamicamente segmenti di memoria condivisa (`shmget`) e di "attaccarli" (attach) all'interno dello spazio di indirizzamento dei processi che ne facciano richiesta.

- Quando un processo ha terminato di usare il segmento, lo scollega con `shmdt` (detach). Buona norma: il creatore lo restituisce al sistema.

- La shared memory è la forma più veloce di IPC perché funziona senza intermediazione di oggetti (quali code di messaggi o pipe).

- `int shmat (int shmid, char *shmaddr, int shmflg);`

- Di solito `shmaddr` viene posto a NULL e `shmflg`=0.

- `shmid` è l'identificatore ottenuto da `shmget`

- La chiamata restituisce l'indirizzo a cui accedere al segmento

Controllo IPC

- Per listare gli IPC attivi nel sistema:

  `> ipcs`

- Per rimuovere degli IPC a mano:

  `> ipcrm < msg | sem | shm > < IPC ID >`

- Provare a listare gli IPC presenti.

Esercizio

- Scaricare dalla pagina del corso `shmem.c`
- Capire il funzionamento del codice e compilare.
- Tempo 15 minuti.

Semafori

- I semafori sono dei contatori usati per sincronizzare l'accesso di risorse condivise da parte di processi concorrenti.

- Ogni semaforo UNIX è in realtà una lista di semafori a contatore. E' quindi fare più operazioni su più semafori in modo atomico.

- `semid semget ( key_t key, int numsem, int flags );`

- Per creare un nuovo IPC che contiene `numsem` semafori. Il numero di ID è `semid`

- Esempio: `semid = semget ( key, 2, 0600 | IPC_CREAT | IPC_EXCL)`

Semafori

- La chiamata `semop()` esegue una lista di operazioni, equivalenti a più UP e DOWN su più semafori

- Usa una una lista (`sops[]`) di (`nsops`) operazioni:
  ```
  int semop ( int  semid, struct sembuf *sops, unsigned nsops);
  ```

- Ad esempio: `semop ( sid, sops, 1);` indica che si fa *una* operazione (contenuta in sops[0]).

Semafori

- I corrispondenti campi di `sembuf` per esempio sono:
  ```
  sops[0].sem_num = 0;   // numero di semaforo a cui ci si riferisce
  sops[0].sem_op = -1;   // operazione da eseguire (+1,0, -1)
  sops[0].sem_flg = 0;   // flags
  ```

- Se `sem_op` è negativo, il suo valore viene sottratto dal semaforo. Ciò corrisponde ad ottenere le risorse che il semaforo controlla.

- Se `sem_op` è positivo, il suo valore viene aggiunto al semaforo. Ciò corrisponde a restituire le risorse che il semaforo controlla.

- Se `sem_op` è zero, il chiamante si mette in sleep finché il valore del semaforo è zero. In pratica si attende l'utilizzazione piena del semaforo (non è nella definizione classica di semaforo)

- Le tradizionali WAIT e SIGNAL dei semafori sono realizzate con `sem_op` rispettivamente a -1 e 1.

Semafori - Controllo

- `semctl()` è usata per operazioni di controllo su un semaforo.
- `int semctl ( int semid, int semnum, int cmd, union semun arg);`
- `semnum` è il numero di semaforo dell'IPC.
- `cmd` è SETVAL, GETVAL, GETNCNT o IPC_RMID
- `arg` à di tipo `semun`, dichiarato in `sys/sem.h` come:

```
/* arg for semctl system calls. */
union semun {
        int val;                    /* value for SETVAL */
        struct semid_ds *buf;       /* buffer for IPC_STAT & IPC_SET */
        ushort *array;              /* array for GETALL & SETALL */
        struct seminfo *__buf;      /* buffer for IPC_INFO */
        void *__pad;
};
```

Esercizio

- Scaricare dalla pagina del corso `sem.c`
- Capire il funzionamento del codice e compilare.
- Tempo 15 min.

Esercizio da consegnare

- Sulla base di sem.c implementare il problema dei 5 filosofi.
- `http://en.wikipedia.org/wiki/Dining_philosophers_problem`
- Si creino 5 processi che rappresentano ciascun filosofo.
- Gestire in modo appropriato i semafori in modo da farli mangiare senza starvation.
- Usare delle stampe a video per mostrare il progresso.