laboratorio di sistemi operativi

Semafori - esercizi

Marco Botta
Materiale preparato da Daniele Radicioni

```
#define EXIT_ON_ERROR if (errno) {fprintf(stderr,   \
                "%d: pid %ld; errno: %d (%s)\n",     \
                __LINE__,                            \
                (long) getpid(),                     \
                errno,                               \
                strerror(errno)); exit(EXIT_FAILURE);}
```

propedeutica all'uso dei semafori

1. Scrivere un programma sem_00_gen.c che crei un semaforo e lo inizializzi a 0.
2. Scrivere un programma sem_01_use_V.c che esegua un'operazione V sul semaforo generato da sem_00_gen.c.
3. Scrivere un programma sem_02_use_P.c che esegua un'operazione P sul semaforo generato da sem_00_gen.c.
4. Scrivere un programma sem_03_rm.c che deallochi il semaforo.

- sperimentare l'esecuzione dei quattro programmi tentando di eseguire una (e/o più operazione/i) P prima delle V e verificando il comportamento del semaforo. Per non dovere aprire diversi terminali a causa dei processi in attesa, utilizzare l'opzione & per mettere in background il processo che esegue la P.
- verificare cosa accade se si tenta di rimuovere un semaforo con un processo in attesa di effettuare una P su quel semaforo

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

```c
#define MIAKEY 1492941
// crea e inizializza un semaforo a 0
// da altri programmi
int main(int argc, char ** argv) {
   union semun arg;
   int sem_id; // ID semaforo
   // il secondo argomento della semget() è la dimensione dell'array:
   // nel nostro caso si tratta di un solo elemento
   if((sem_id = semget(MIAKEY, 1, IPC_CREAT | 0666)) == -1)
      errExit("semget_1 error");
   printf("semaforo %d creato correttamente\n", sem_id);
   //
   // per mezzo della semctl() assegnamo il valore (operazione SETVAL)
   // al semaforo identificato da sem_id e posto all'elemento 0
   arg.val = 0;
   printf("arg.val = %d\n", arg.val);
   // ora usiamo Semctl() per inizializzare il sem a 0
   if (semctl(sem_id, /* semnum= */ 0, SETVAL, arg) == -1)
      errExit("semctl SETVAL error");
   fprintf(stdout, "Semaforo %d inizializzato.\n", sem_id);
   exit(EXIT_SUCCESS);
}
```

```c
#define MIAKEY 1492941
// programma che utilizza un semaforo già allocato e inizializzato, su cui
// compie una operazione V (signal - releaseSem)
int main(int argc, char ** argv) {
   int id;  // ID semaforo
   struct sembuf operations[1];   // array composto da una sola operazione
   int retval; // valore di ritorno della semop()

   // ----- recupero l'id del semaforo ------
   if((id = semget(MIAKEY, 1, 0666)) == -1) errExit("non trovo semaforo\n");

   // --------  effettuo una operazione V ---------
   printf(" -- sem_01_use_V sta per effettuare un'operazione V\n");
  printf(" -- processo con PID %d\n", getpid());

   // ----- impostazione della struttura sembuf -------
   operations[0].sem_num = 0; // operazione sul semaforo corrispondende
                              // all'elemento in posizione 0
   operations[0].sem_op = 1;    // codice operazione: incremento di 1
   operations[0].sem_flg = 0; // flag a 0 significa che si attenderà fino
                              // a quando l'operazione sarà possibile
   // ----- azione! qui effettuiamo l'operazione V -----
   if((retval = semop(id, operations, 1)) != 0)
      errExit("errore nella operazione V :-/ ");
   printf("operazione V andata a termine con successo !-) \n");
   exit(EXIT_SUCCESS);
}
```

```c
#define MIAKEY 1492941
// utilizza un semaforo già allocato e inizializzato, su cui
// compie una operazione P (wait - reserveSem)
int main(int argc, char ** argv) {
   int id; // ID semaforo
   struct sembuf operations[1]; // array composto da una sola operazione
                                // da eseguire sul semaforo
   int retval; // valore di ritorno della semop()
   // ----- recupero l'id del semaforo ------
   if((id = semget(MIAKEY, 1, 0666)) == -1)
      errExit("cannot find semaphore, exiting\n");

   // --------  effettuo una operazione P ---------
   printf(" ---- sem_02_use_P sta per effettuare un'operazione P\n\n");
   printf(" ---- processo con PID %d\n", getpid());
   // ----- impostazione della struttura sembuf -------
   operations[0].sem_num =  0; // operazione sul semaforo corrispondende
                               // all'elemento in posizione 0
   operations[0].sem_op  = -1; // codice operazione: incremento di 1
   operations[0].sem_flg =  0; // flag a 0 significa che si attenderà fino
                               // a quando l'operazione sarà possibile
   // ----- azione! qui effettuiamo l'operazione V -----
   if((retval = semop(id, operations, 1)) != 0)
      errExit("errore nella operazione P :-/ ");
   printf(" ---- operazione V andata a termine con successo !-) \n");
   exit(EXIT_SUCCESS);
}
```

```c
#define MIAKEY 1492941
// programma che dealloca un semaforo già allocato
int main(int argc, char ** argv) {
   union semun arg;
   int id;  // ID semaforo

   // ----- recupero l'id del semaforo ------
   if((id = semget(MIAKEY, 1, 0666)) == -1)
      errExit("cannot find semaphore, exiting\n");

   // --------  effettuo la deallocazione ---------
   printf(" -- sem_03_use_P sta per deallocare il semaforo\n\n");
   printf(" -- processo con PID %d\n", getpid());

   if (semctl(id, 0, IPC_RMID, arg) == -1)
      errExit("errore nella deallocazione :-/ ");

   printf(" -- deallocazione terminata con successo ;-)) \n");

   exit(EXIT_SUCCESS);
}
```

sincronizzazione padre-figlio

- Scrivere un programma che alloca un semaforo e lo inizializza a ***0***, quindi forka e poi effettua una ***reserveSem()*** sul semaforo (da implementare tramite ***semop***). Il processo figlio esegue la ***releaseSem()*** (da realizzare con ***semop***) sullo stesso semaforo dopo 5 secondi dal suo avvio e termina. Il processo padre, risvegliato dall'attesa, dealloca il semaforo e termina.

```c
int main(int argc, char** argv) {
  int semid; union semun arg;
  // ------ alloca un semaforo ------


  // ------ e lo inizializza a 0 -------


  switch(fork()) {


    case 0: // ------------- figlio ---------------
    // figlio esegue la V (da realizzare con semop) sullo stesso
    // semaforo dopo 5 secondi dal suo avvio e termina


    default: // -------------- padre ---------------
    // effettua una P sul semaforo (da implementare tramite semop)


    // ----- risvegliato dall'attesa disalloca il semaforo e termina -----

  }
  exit(EXIT_SUCCESS);
}
```

```c
int main(int argc, char** argv) {
  int semid; union semun arg;
  // ------ alloca un semaforo ------
  semid = semget(IPC_PRIVATE, 1, S_IRUSR | S_IWUSR);
  arg.val = 0;
  // ------ e lo inizializza a 0 -------
  if (semctl(semid, /* semnum= */ 0, SETVAL, arg) == -1) {...}
  printf("creato semaforo, con ID = %d\n", semid);
  switch(fork()) {
    int i;
    case -1: {...}
    case 0: // ------------- figlio ---------------
    // figlio esegue la V (da realizzare con semop) sullo stesso
    // semaforo dopo 5 secondi dal suo avvio e termina
    printf("processo figlio: PID %d\n", getpid());
    printf("processo figlio: mi metto in sleep per 5 secondi\n");
    for(i=0; i<5; ++i) {printf(" --- dormo...\n"); sleep(1);}
    printf("processo figlio: eseguo la V...\n");
    if(releaseSem(semid,0) == -1) {...}
    printf("processo figlio: V eseguita\n");
    exit(EXIT_SUCCESS);
    default: // -------------- padre ---------------
    // effettua una P sul semaforo (da implementare tramite semop)
    printf("processo padre: PID %d\n", getpid());
    printf("processo padre: eseguo la P...\n");
    if(reserveSem(semid,0) == -1) {...}
    printf("processo padre: P eseguita\n");
    // ----- risvegliato dall'attesa disalloca il semaforo e termina -----
    if (semctl(semid, 0, IPC_RMID, arg) == -1) {...}
  }
  exit(EXIT_SUCCESS);
}
```

sezione critica (1)

- Scrivere un programma in cui un padre crea tre processi figli che tentano di accedere a una sezione critica. La sezione critica è costituita dalla stampa di una scritta (decidete cosa).
- L'accesso alla sezione critica deve essere regolato con le primitive ***wait()*** e ***signal()*** (o ***reserve_sem()*** e ***release_sem()***, rispettivamente).
- Dopo avere atteso la terminazione dei figli, il padre stampa un messaggio a video, dealloca il semaforo e termina.

```c
int main(int argc, char ** argv) {
  int i,pid,semid,k; union semun arg;
  int n_secs = 5;
  setbuf(stdout, NULL);
  if((semid = semget(IPC_PRIVATE, 1, 0666)) < 0) {...}
  if((initSemWithValue(semid, 0, 1)) == -1) {...}
  for (k=0; k<3; k++) {
    switch(fork()) {
      case -1: {...}
      case 0: // ------- figlio -------
      printf("\nfiglio [PID: %d]: tento di accedere alla s. critica\n",
        getpid());
      // ====================================
      sleep(n_secs-k); // cosa accade con questa istruzione??
      reserveSem(semid,0);
      sezione_critica(); // funzione che stampa un qualche messaggio
      printf("figlio [PID: %d]: esco dalla s. critica\n", getpid());
      releaseSem(semid,0);
      // ====================================
      exit(EXIT_SUCCESS);
    }
  }
  printf("[padre - PID: %d] attendo terminazione dei figli\n",getpid());
  for(i=0; i<3; ++i) { wait(NULL); }
  printf("[padre - PID: %d] dealloco il semaforo\n", getpid());
  if ((semctl(semid, 0, IPC_RMID, arg)) == -1) {...}
  exit(EXIT_SUCCESS);
}
```

```
int main(i
  int i,pi
  int n_se
  setbuf(s
  if((semi
  if((init
  for (k=0
    switch
      case
      case 0: // ------- figlio -------
      printf("\nfiglio [PID: %d]: tento di accedere alla s. critica\n",
        getpid());
      // ====================================
      sleep(n_secs-k); // cosa accade con questa istruzione??
      reserveSem(semid,0);
      sezione_critica(); // funzione che stampa un qualche messaggio
      printf("figlio [PID: %d]: esco dalla s. critica\n", getpid());
      releaseSem(semid,0);
      // ====================================
      exit(EXIT_SUCCESS);
    }
  }
  printf("[padre - PID: %d] attendo terminazione dei figli\n",getpid());
  for(i=0; i<3; ++i) { wait(NULL); }
  printf("[padre - PID: %d] dealloco il semaforo\n", getpid());
  if ((semctl(semid, 0, IPC_RMID, arg)) == -1) {...}
  exit(EXIT_SUCCESS);
}
```

```
// inizializza il semaforo semNum del set identificato da
// semId, semNum al valore _val
int initSemWithValue(int semId, int semNum, int _val) {
  union semun arg;

  arg.val = _val;
  return semctl(semId, semNum, SETVAL, arg);
}
```

```c
int main(int argc, char ** argv) {
  int i,pid,semid,k; union semun arg;
  int n_secs = 5;
  setbuf(stdout, NULL);
  if((semid = semget(IPC_PRIVATE, 1, 0666)) < 0) {...}
  if((initSemWithValue(semid, 0, 1)) == -1) {...}
  for (k=0; k<3; k++) {
    switch(fork()) {
      case -1: {...}
      case 0: // ------- figlio -------
      printf("\nfiglio [PID: %d]: tento di accedere alla s. critica\n",
        getpid());
      // ====================================
      sleep(n_secs-k); // cosa accade con questa istruzione??
      reserveSem(semid,0);
      sezione_critica(); // funzione che stampa un qualche messaggio
      printf("figlio [PID: %d]: esco dalla s. critica\n", getpid());
      releaseSem(semid,0);
      // ====================================
      exit(EXIT_SUCCESS);
    }
  }
  printf("[padre - PID: %d] attendo terminazione dei figli\n",getpid());
  for(i=0; i<3; ++i) { wait(NULL); }
  printf("[padre - PID: %d] dealloco il semaforo\n", getpid());
  if ((semctl(semid, 0, IPC_RMID, arg)) == -1) {...}
  exit(EXIT_SUCCESS);
}
```

scrittore e lettore

- Scrivere due programmi che interagiscano nella scrittura e lettura, come descritto a lezione: lo scrittore deve leggere il contenuto del file ***file_prova*** e scriverlo nella memoria condivisa; il lettore legge quanto scritto nella memoria condivisa e lo stampa a video.
- I due programmi utilizzano due semafori per sincronizzarsi: uno per organizzare l'attività di scrittura e uno per la lettura. Esempio di interazione con i due programmi:

```
$ ./writer < file_prova &
[1] 87902
$ ./reader
// stampa a video il contenuto del file file_prova ...
[lettore]  : ricevuti 236 bytes (1 n_cycles)
[scrittore]: inviati 236 bytes (1 n_cycles)
[1]+  Done        ./writer < file_prova
```

scrittore e lettore