Caselle riservate

Ex. 1	
Ex. 2	
Ex. 3	
Ex. 4	
Ex. 5	
Ex. 6	
Tot.	

Sistemi Operativi

Compito d'esame

15 Febbraio 2013

Matricola _____ Cognome _____ Nome _____

Docente: ◯ Laface ◯ Quer

Non si possono consultare testi, appunti o calcolatrici. Riportare i passaggi principali. L'ordine sarà oggetto di valutazione.
Durata della prova: 60 **minuti.**

1. Si supponga di avere a disposizione l'implementazione Pthread dei semafori binari attraverso le funzioni:
   - `int pthread_mutex_init (pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);`
   - `int pthread_mutex_lock (pthread_mutex_t *mutex);`
   - `int pthread_mutex_unlock (pthread_mutex_t *mutex);`

   Si mostri come sia possibile implementare un semaforo con conteggio, ovvero non binario, il cui valore iniziale sia uguale a `count`.

   Suggerimento: si ricordino le implementazioni "originali" delle primitive `signal` e `wait`.

```
typedef struct {
  int count;              /* the counter */
  pthread_mutex_t lock;   /* mutex ensuring exclusive access to count */
  pthread_mutex_t s;      /* real semaphore */
} Semaphore;

static void semaphore_init (Semaphore *s, int i) {
  pthread_mutex_init (&s->lock, NULL);
  pthread_mutex_init (&s->s, NULL);
  pthread_mutex_lock(&s->s);
  s->count = i;
}
static void semaphore_wait (Semaphore *s) {
  pthread_mutex_lock (&s->lock);
  s->count--;
  if (s->count < 0){
    pthread_mutex_unlock (&s->lock);
    pthread_mutex_lock (&s->s);
  } else
    pthread_mutex_unlock (&s->lock);
}
static void semaphore_signal (Semaphore *s) {
  pthread_mutex_lock (&s->lock);
  s->count++;
  if (s->count <= 0)
    pthread_mutex_unlock (&s->s);
  pthread_mutex_unlock (&s->lock);
}
```

2. Si descrivano il significato e l'utilizzo delle system call `signal` e `kill`.

   Che cosa è una `corsa critica` (race codition)?

   Quali sono le funzioni principali della system call `signal(SIGCHLD, SIG_IGN)`? Che relazione ha con la system call `wait`?

   Un segnale è un interrupt software, i.e., un evento di sistema inviato a un processo. I segnali permettono di gestire eventi asincroni. Non sono interrupt perchè gli interrupt sono inviati dallhardware al sistema operativo. I segnali sono inviati a processi da altri processi.
   Sono utilizzati per notificare il verificarsi di eventi particolari: Condizioni di errore, violazioni di segmentazione di memoria, errori floating point o istruzioni illegali, etc.
   Per esempio `ctrl-C` invia un segnale `INT` (`SIGINT`) e `ctrl-Z` invia un segnale `TSTP` (`SIGTSTP`).

   Una race condition avviene quando il comportamento di più processi che lavorano su dati comuni dipende dallordine di esecuzione. La segnalazione tra processi può generare race conditions che portano il programma a non funzionare nel modo desiderato.

   ```
   #include <signal.h>
   void (*signal (int sig, void (*func)(int)))(int);
   ```
   Il file `signal.h` definisce i nomi dei segnali (i.e., `SIGABORT` Process abort, `SIGALARM` Alarm clock, etc.).
   La `signal` consente di settare un signal handler. Ha due parametri: `sig` che specifica il segnale da intercettare e `func` che indica la funzione da invocare. `func` ha un solo parametro di tipo int (il segnale ricevuto).
   Quando un segnale si presenta è possibile: (a) Ignorare esplicitamente il segnale (tale comportamento è impossibile per i segnali SIGKILL e SIGSTOP che non possono essere ignorati), (b) Lasciare che si verifichi il comportamento di default, (c) Catturare il segnale.

   ```
   #include <signal.h>
   int kill (pid_t pid, int sig);
   ```
   Invia un segnale a un processo o a un gruppo di processi.
   È possibile spedire segnali solo a processi con lo stesso UID.

   La system call `signal (SIGCHLD, SIG_IGN)` permette a un processo di ignorare (costante `SIG_IGN`) i segnali `SIGCHLD` inviati dai figli che terminano. Questo è anche il comportamento di default. Se un processo vuole essere notificato dalla terminazione dei figli (ed evitare che diventino "zombie") deve utilizzare la system call `wait`.

3. Con riferimento alle soluzioni hardware al problema della sincronizzazione mediante le procedure di `testAndSet`, se ne illustrino le funzionalità, riportandone il codice e il relativo protocollo di utilizzo.

Nei sistemi multiprocessore con memoria comune occorre utilizzare un "lucchetto" (lock) per controllare laccesso. Il lock può essere realizzato mediante una istruzione "atomica". Una istruzione è atomica se viene eseguita in un unico "memory cycle", i.e., non può essere interrotta. Istruzioni atomiche di lock sono la `Test-And-Set` su una variabile di lock e la `Swap` di due variabili, di cui una di lock.

La `Test-And-Set` effettua un test (controllo) seguito da un set (assegnazione) di una variabile di lock globale. Le operazioni vengono eseguite in un solo ciclo indivisibile. La variabile di lock è di tipo char (1 singolo byte) Se il byte è 0 la SC è libera. Se il byte è 1 la SC è occupata. In ogni caso al byte viene assegnato valore 1.

```
char Test-and-Set (char *target) {
  char val;

  val = *target;
  *target = TRUE;                  // 1
  return val;
}
```

Mutua eslusione con `Test-And-Set`:

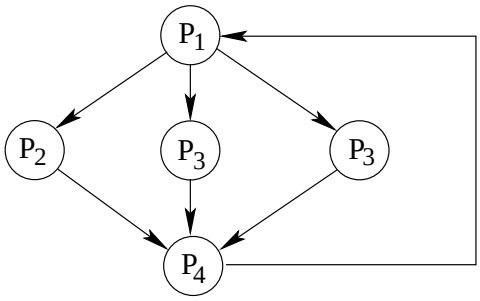
```
char target = FALSE;               // target (lock) GLOBALE
...
while (TRUE) {
   while (Test-And-Set (&target));  // lock
   SC
   target = FALSE;                 // unlock
   sezione non critica
}
```

Mutua eslusione con `Test-And-Set` senza starvation:

```
while (TRUE) {
  waiting[i] = TRUE;
  key = TRUE;
  while (waiting[i] && key)
    key = Test-And-Set (lock);
  waiting[i] = FALSE;
 SC di Pi
  j = (i+1) % N;
  while ((j!=i) and (waiting[j]==FALSE))
    j = (j+1) % N;
  if (j==i) lock = 0;
  else waiting[j] = FALSE;

  sezione non critica
}
```

4. Dato il seguente grafo di precedenza, realizzarlo utilizzando il **minimo** numero possibile di semafori. Si utilizzino le primitive semaforiche implementate tramite `pipe`. Riportare il corpo dei processi $(P_1, \ldots, P_4)$, nonché delle primitive `init`, `signal` e `wait` realizzate mediante pipe, e l'inizializzazione dei semafori. I processi rappresentati devono essere processi ciclici (con corpo del tipo `while(1)`).



Si osservi che il grafo prevede l'esecuzione di 4 processi tra i quali ci sono *due istanze identiche* dello stesso processo P_3.

```
semaphore init () {
  int *sem;
  sem = calloc (2,sizeof(int));
  pipe (sem);
  return sem;
}

void signal (semaphore s) {
  if (write(s[1], "x", 1) <=0) {
    fprintf(stderr, "ERROR : signal\n");
    exit(1);
  }
}
```

```
void wait (semaphore s) {
  int junk;
  if (read(s[0], &junk, 1) <=0) {
    fprintf(stderr, "ERROR : wait\n");
    exit(1);
  }
}
```

```
Fase di inizializzazione:
init (s1, 1);
init (s2, 0); init (s3, 0); init (s4, 0);
init (s, 0); n = 0;

P1
while (1) {
  wait (s1);
  printf ("P1\n");
  signal (s2);
  signal (s3);
}

P3
while (1) {
  wait (s3);
  n++;
  if (n==1)
    signal (s3);
  else {
    n = 0;
    signal (s);
    signal (s);
  }
  wait (s);
  printf ("P3\n");
  signal (s4);
}
```

```
Fase di terminazione:
destroy (s1);
..
destroy (s3);

P2
while (1) {
  wait (s2);
  printf ("P1\n");
  signal (s4);
}

P4
while (1) {
  for (i=0; i<3; i++)
    wait (s4);
  printf ("P4\n");
  signal (s1);
}
```

5. Implementare uno script BASH che riceva sulla linea di comando quattro argomenti: `dir`, `dim1`, `dim2` e `cancella`. Lo script deve
   - controllare il corretto passaggio dei parametri
   - cercare tutti i file
     - nell'albero di direttori con radice `dir` e profondità compresa tra 2 e 4
     - con estensione `txt`
     - con dimensione maggiore di `dim1+dim2`
   - cancellare ogni riga che inizia oppure termina con la stringa contenuta nella variabile cancella.

   Non si ricorra né all'utilizzo di SED né a quello di AWK.

```
1  #/bin/bash
2
3  if [ $# -ne 4 ]
4  then
5    echo "usage $0 dir dim1 dim2 string"
6  fi
7
8  dim=$(( $2+$3 ))
9
10 for file in ‘find $1 -mindepth 2 -maxdepth 4 -name "*.txt" -size +$dim‘
11 do
12   echo $file
13   grep -v "^$4\|$4$" $file > tmp
14   cat tmp > $file
15   rm tmp
16 done
```

6. Realizzare uno script AWK che legga da un file dati.txt, che contiene una linea con sequenze di numeri, separati da uno spazio, come nel seguente esempio:

```
1 1 1 1 1 2 2 2 2 2 3 3 3 4 4 4 4 4 1 1 1 2 2 3 3 3 3 3 1 1 1 1 1 1 2 3 3 3
```

Si può osservare che i numeri appaiono in serie con ripetizioni, ciascuna serie è in ordine crescente. Lo script deve creare un file differente per ciascuna serie, che riporti il numero di ripetizioni di ciascun numero che appare nella serie secondo il formato seguente (corrispondente all'elaborazione del file dati.txt):

```
File out.1      File out.2     File out.3
1 - 5           1 - 3          1 - 6
2 - 6           2 - 2          2 - 1
3 - 3           3 - 5          3 - 3
4 - 5
```

Non si ricorra né all'utilizzo di BASH né a quello di SED.

```
#!/usr/bin/awk -f

BEGIN {
  j=0;
  prec=" "
}

{
  for (i=1;i<=NF;i++) {
    print prec $i
    if (prec!=$i && $i==1) {
       for (n in vett) {
         print n," - ",vett[n] >> "out."j
         delete vett[n];
       }
    j++;
  }

   vett[$i]++;
   prec=$i;
  }
}

END {
  for (n in vett) {
    print n," - ",vett[n] >> "out."j
  }
}
```