Caselle riservate

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

Sistemi Operativi

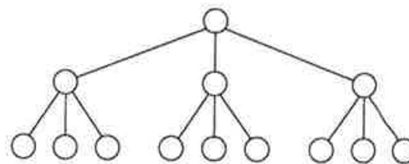
Compito d'esame

21 Febbraio 2015

Matricola ________________ Cognome ____________________________ Nome ____________________________

Non si possono consultare testi, appunti o calcolatrici. Riportare i passaggi principali. L'ordine sarà oggetto di valutazione.
Durata della prova: 100 **minuti.**

1. Si scriva un programma C che ricevuti due valori interi sulla riga di comando, rispettivamente a e n, generi un albero di processi di altezza a e grado n. La figure riporta un esempio nel caso di $a = 2$ e $n = 3$.



Più nel dettaglio ogni nodo dell'albero è un processo. Il processo iniziale deve generare n processi figlio e terminare. La stessa cosa devono fare tutti i processi figli, generando così un numero di processi sulle foglie dell'albero pari a n^a. I processi sulle foglie devono tutti visualizzare il proprio PID e terminare.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>

int main (int argc, char *argv[]) {
  int i, j, a, n, pid;

  a = atoi (argv[1]);
  n = atoi (argv[2]);

  for (i=0; i<a; i++) {
    for (j=0; j<n; j++) {
      pid = fork ();
      if (pid>0) {
        break;
      } else {
        if (j==n-1)
          exit (0);
      }
    }
  }

  printf ("pid=%d\n", getpid());

  return (0);
}
```

2. Si descriva l'utilizzo dei segnali nel sistema operativo UNIX/Linux con relativi vantaggi e svantaggi. Si descrivano le system call `signal`, `kill`, `pause` e `alarm`. In particolare si indichi:
   - Se è possibile per un processo ignorare l'arrivo di un segnale.
   - Che cosa succede se un processo riceve un segnale durante l'esecuzione della funzione di gestione.
   - Se è possibile avere più funzioni di gestione associate a un segnale.
   - Se è possibile avere più segnali associati alla stessa funzione di gestione.
   - Come è possibile implementare la system call `alarm` tramite le system call `fork`, `signal`, `kill` e `pause`.

```
#include <signal.h>
void (*signal (int sig, void (*func)(int)))(int);
int kill (pid_t pid, int sig);
int pause (void);
unsigned int alarm (unsigned int seconds);
```

- Signal: consente di instanziare un gestore di segnali. Parametri: sig indica il segnale da intercettare (SIGCHLD, SIGUSR1, etc.), func specifica l'indirizzo (i.e., puntatore) della funzione da invocare quando si presenta il segnale. Tale funzione ha un solo parametro di tipo int. Esso indica a sua volta il segnale da gestire. Il valore di ritorno è il puntatore al signal handler che gestiva il segnale in precedenza oppure SIG_ERR in caso di errore. signal (SIG..., SIG_DFL) installa il comportamento di default. signal (SIG..., SIG_IGN) ignora il segnale.
- Kill: invia il segnale (sig) a un processo o a un gruppo di processi (pid). Per inviare un segnale a un processo occorre averne diritto. Un processo utente può spedire segnali solo a processi con lo stesso UID.
- Pause: sospende il processo chiamante sino all'arrivo di un segnale.
- Alarm: attiva un timer (count-down). Il parametro seconds specifica il valore del conteggio (in numero di secondi). Al termine del count-down si genera il segnale SIGALRM. L'azione di default è la terminazione.

La memoria è limitaata a due segnali (i successivi sono perduti). È possibile avere una sola funzione instanziata in un certo istante per un determinato segnale. Ogni funzione può gestire più segnali.

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
void myAlarm (int sig) {
  printf ("Alarm\n");
}
main () {
  int n = 10;
  pid_t pid;
  (void) signal (SIGALRM, myAlarm);
  pid = fork();
  if (pid==0) {
    /* child */
    Sleep (n);
    kill (getppid(), SIGALRM);
    exit (0);
  } else {
    /* father */
    pause (); /* suspend until next signal */
    ...
  }
  return (0);
}
```

3. Si faccia riferimento alla sincronizzazione di processi mediante procedura `test-and-set`. Se ne illustrino le principali caratteristiche, riportandone il codice e il relativo protocollo di utilizzo. È possibile implementare una mutua esclusione senza starvation?

   Si supponga inoltre di avere a disposizione, al posto della `test-and-set` e della swap, la seguente funzione atomica:

```
int atomicIncrement (int *var) {
  int tmp = *var;
  *var = tmp + 1;
  return (tmp);
}
```

   Utilizzare tale funzione per scrivere le funzioni `init`, `lock` e `unlock` da inserire nel prologo e nell'epilogo di una sezione critica. *Suggerimento*: si utilizzino due variabili globali `ticketNumber` e `turnNumber`. La prima indica l'ordine di prenotazione per l'accesso alla sezione critica e la seconda il processo che ha il turno all'accesso. Gestire tali variabili con le funzioni di `init`, `lock` e `unlock`.

Le soluzioni hardware per la gestione dei deadlock ...

```
char TestAndSet (char *lock) {
  char val;
  val = *lock;
  *lock = TRUE;
  return val;
}

char lock = FALSE;          // GLOBAL lock
...
while (TRUE) {
   while (TestAndSet (&lock));     // lock
   SC
   lock = FALSE;                   // unlock
   sezione non critica
}
```

Vantaggi delle soluzioni hardware: Utilizzabili in ambienti multi-processore, facilmente estendibili a N processi, semplici da utilizzare dal punto di vista software, cioè dal punto di vista utente.
Svantaggi delle soluzioni hardware: Non facili da implementare a livello hardware, operazioni atomiche su variabili globali (qualsiasi), busy Waiting su spin-lock (spreco di risorse ovvero di cicli di CPU nell'attesa), starvation (la selezione dei processi in busy-waiting per la SC è arbitraria e gestita dai processi stessi).
Pwer evitare starvation occorre estendere la soluzione, ad esempio seguendo Burns [1978].

```
while (TRUE) {
  waiting[i] = TRUE;
  key = TRUE;
  while (waiting[i] && key)
    key = TestAndSet (lock);
  waiting[i] = FALSE;
  SC
  j = (i+1) % N;
  while ((j!=i) and (waiting[j]==FALSE))
    j = (j+1) % N;
  if (j==i)
    lock = FALSE;
  else
    waiting[j] = FALSE;
  sezione non critica
}
```

```
typedef struct lock_s {
    int ticketNumber;
    int turnNumber;
} lock_t;

void init (lock_t lock) {
    lock.ticketNumber = 0;
    lock.turnNumber = 0;
}

void lock (lock_t lock) {
    int myTurn = atomicIncrement(&(lock.ticketNumber));
    while (lock.turnNumber != myTurn);
}

void unlock (lock_t lock) {
    atomicIncrement(&(lock.turnNumber));
}
```

4. Il comando di shell "`ncal 02 2015`" visualizza il calendario del mese di febbraio dell'anno 2015 come evidenziato a sinistra dell'esempio successivo.

   All'interno del direttorio corrente tutti i file di estensione "`.cal`" contengono un insieme di date con il formato riportato al centro dell'esempio. Si scriva uno script BASH in grado di visualizzare (a video) l'elenco delle date presente su tutti i file di estensione "`.cal`" arricchito dal giorno delle settimana. Nel caso del file riportato al centro dell'esempio lo script deve visualizzare a video l'elenco riportato a destra.

```
    February 2015
Su  1  8 15 22
Mo  2  9 16 23
Tu  3 10 17 24
We  4 11 18 25
Th  5 12 19 26
Fr  6 13 20 27
Sa  7 14 21 28
```

```
8 9 2014
1 1 2015
2 2 2015
18 2 2015
...
```

```
8 9 2014 Mo
1 1 2015 Th
2 2 2015 Mo
18 2 2015 We
...
```

```
#!/bin/bash


for file in `find . -name "*.cal"`
do
  while read line
  do
    giorno=`echo $line | cut -d " " -f 1`
    mese=`echo $line | cut -d " " -f 2`
    anno=`echo $line | cut -d " " -f 3`

    if [ `echo $giorno|cut -c 1` == "0" ]
    then
      giorno_cal=`echo $giorno|cut -c 2`
    else
      giorno_cal=$giorno
    fi

    giorno_stringa=`ncal $mese $anno | grep " $giorno_cal " | cut -d " " -f 1`

    echo "$line $giorno_stringa "

    done <$file
done
```

5. Si scriva uno script AWK in grado di sostituire le parole di un testo con un numero intero crescente inversamente proporzionale alla frequenza assoluta con cui ogni parola compare nel testo stesso. Il seguente esempio mostra l'azione dello script.

   A sinistra è riportato un possibile file di ingresso. Si supponga il file includa solo caratteri alfabetici. Caratteri alfabetici minuscoli e maiuscoli vanno considerati come equivalenti.

   Al centro è stato indicato il primo file di uscita. Esso riporta la frequenza assoluta (`freq`) di tutte le parole che compaiono nel file di ingresso con il relativo numero intero crescente (`codice`) ad esso associato. Tale codice è stato assegnato in ordine decrescente di frequenza, i.e., a frequenza maggiore corrisponde valore intero minore.

   A destra è riportato il secondo file di uscita. In esso tutte le parole del file di ingresso sono sostituite dal loro numero intero (`codice`) corrispondente.

```
esempio esempio              esempio freq=4 codice=1        1 1
Esempio per script AWK       per freq=1 codice=4            1 4 3 2
Script AWK                   script freq=2 codice=3         3 2
ESEMPIO awk                  awk freq=3 codice=2            1 2
```

   Il nome dei tre file può essere passato allo script sulla riga di comando o, in alternativa, assegnato a variabili dello script sulla riga di comando stessa. Si ricorda che il comando `str=toupper(str)` (`str=tolower(str)`) trasforma la stringa `str` in caratteri maiuscoli (minuscoli).

```
BEGIN{
  f2=ARGV[2]
  f3=ARGV[3]
  printf "" > f2
  printf "" > f3
  ARGV[2]=""
  ARGV[3]=""
  max_freq=0;
}
{
  #calcolo frequenze
  for(i=1;i<=NF;i++){
   word=tolower($i)
   freq[word]++
   if (max_freq < freq[word])
    max_freq=freq[word]
  }
}

END{
  #assegno gli ID
  num_freq=max_freq
  for(v=1; v<=max_freq;v++) {
  # print v
   vett_inv[v]=num_freq
   num_freq--
  }
  #stampo le frequenze
  for(w in freq)
    print w " freq=" freq[w] " codice=" vett_inv[freq[w]] >> f2
  #sostituisco
  while(getline < ARGV[1] ) {
   line=""
   for(i=1;i<=NF;i++) {
     w=tolower($i)
     line=line" "vett_inv[freq[w]]
   }
   print line >> f3
  }

}
```

6. Si illustri l'*algoritmo del banchiere* riportandone la descrizione e lo pseudo-codice ottimizzato al caso di esattamente tre processi e una sola risorsa.

Analizzando l'esempio successivo, con processi ($P_1, \ldots, P_5$) e risorse ($R_1, \ldots, R_4$), si indichi se lo stato è sicuro oppure non sicuro. Nel primo caso, si illustri l'ordine in cui i processi possono essere completati. Se arrivasse una richiesta di P_1 per $(1, 1, 2, 0)$, tale richiesta potrebbe essere soddisfatta?

	Processo	Fine	Assegnate				Massimo				Necessità				Disponibilità				
			R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	
1	P_1	~~F~~ T	2	0	0	1	4	2	1	2	2	2	1	1	3	3	2	1	→ P1
3	P_2	~~F~~ T	3	1	2	1	5	2	5	2	2	1	3	1	5	3	2	2	→ P4
4	P_3	~~F~~ T	2	1	0	3	2	3	4	6	0	2	4	3	6	3	3	4	→ P2
2	P_4	~~F~~ T	1	0	1	2	1	1	2	4	0	1	1	2	9	4	5	5	→ P3
5	P_5	F T	1	1	3	2	3	6	6	5	2	5	3	3	11	5	5	8	→ P5
															12	6	8	10	

→ P1 (1,1,2,0): disponibilità (3,3,2,1) ok
necessità (2,2,1,1) KO! la richiesta su R3 è > della necessità.

Nome	Dim.	Contenuto e significato
Fine	[n]	Fine[r]=false indica che P_r non ha terminato
Assegnate	[n]	Assegnate[r]=k P_r possiede k istanze di R_c
Massimo	[n]	Massimo[r]=k P_r può richiedere al massimo k istanze di R_c
Necessità	[n]	Necessità[r]=k P_r ha bisogno di altre k istanze di R_c ∀i Necessità[i]=Massimo[i]-Assegnate[i]
Disponibili		Disponibili=k disponibilità pari a k per R

```
n processi Pi
m risorse Rj

Verifica di una richiesta
se
 Richieste[i]≤Necessità[i]
 AND
 Richieste[i]≤Disponibili
 ALLORA
   Disponibili  = Disponibili  - Richieste[i]
   Assegnate[i] = Assegnate[i] + Richieste[i]
   Necessità[i] = Necessità[i] - Richieste[i]
se lo stato risultante è sicuro
  si conferma tale assegnazione
  altrimenti si ripristina lo stato precedente


Verifica di uno stallo
1.
∀i Necessita[i]=Massimo[i]-Assegnate[i]
∀i Fine [i]=false
2.
Trova i per cui
Fine[i]=falso AND ∀j Necessità[i]≤Disponibili
Se tale i non esiste goto step 4
3.
Disponibili=Disponibili+Assegnate[i]
Fine[i]=true
goto step 2
4.
Se ∀i Fine[i]=true il sistema è in uno stato sicuro
```