Caselle riservate

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

Sistemi Operativi

Compito d'esame

17 Febbraio 2014

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: 75 **minuti.**

1. Si scriva una funzione C in grado di eseguire le seguenti operazioni:
   - La funzione, detta "padre", ricevere quali parametri un vettore di interi di nome `vet` e la sua dimensione `n`.
   - Il "padre" genera $n-1$ processi "figli", ciascuno dei quali è numerato da 0 a $n-2$.
     Il processo "figlio" i-esimo:
     – Ordina gli elementi del vettore di posizione i e $i+1$ (due soli elementi) in ordine crescente.
     – Genera un processo "nipote" che si occupa di visualizzare tali elementi (a video) insieme al proprio identificatore (quello del nipote) e a quello del proprio padre (cioe di un "figlio").
     – Ogni "figlio" attende che il "nipote" da lui generato termini.
   - Il processo "padre" attende che tutti i processi "figli" terminino e ritorna il controllo al chiamante.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#define N 10
void p (int [], int);
void f (int [], int);
void n (int [], int);
int main(){
  int vet[N]={9, 8, 7, 6, 5, 4, 3, 2, 1, 0};
  /* Se non metti setbuf la stampa "ordina" rimane nel buffer
     e compare 2 volte ... sembra si lanci il doppio dei figli */
  setbuf(stdout,0);
  /* Padre */
  p (vet, N);
  return (1);
}
void p (int vet[], int n)
{
  pid_t pid;
  int i;
  for (i=0; i<n-1; i++){
    pid = fork();
    if (pid==0) {
      /* Figlio */
      printf ("run figlio %d\n", i);
      f (vet, i);
      exit (1);
    }
  }
  for (i=0; i<n-1; i++){
    pid = wait (NULL);
    printf ("Ricevuta terminazione figlio %d pid=%d\n", i, pid);
  }
  return;
}
void f (int vet[], int i)
{
  pid_t pid;
  int tmp;
  printf ("f %d pid=%d\n", i, getpid());
  if (vet[i]>vet[i+1]) {
    printf ("ordina (%2d,pid=%d): KO ", i, getpid());
    tmp = vet[i];
    vet[i] = vet[i+1];
    vet[i+1] = tmp;
  } else {
    printf ("ordina (%2d,pid=%d): OK ", i, getpid());
  }
  pid = fork();
  if (pid==0) {
    /* Figlio del figlio = Nipote */
    n (vet, i);
    exit (1);
  } else {
    pid = wait (NULL);
    printf ("Ricevuta terminazione nipote %d pid=%d\n", i, pid);
  }
  return;
}
void n (int vet[], int i)
{
  printf ("pid=%d Ppid=%d vet[%d]=%d vet[%d]=%d\n",
    getpid(), getppid(), i, vet[i], i+1, vet[i+1]);
  return;
}
```

2. Si specifichi che cosa si intende per *Process Control Block* e per *Context Switching*. Si rappresenti e si descriva il diagramma degli stati di un processo. Si introduca il concetto di schedulazione dei processi descrivendone le principali caratteristiche e modalità (code, scheduling a breve e a lungo termine, etc.).

   Vedere lucidi e relative spiegazioni oppure i testi consigliati.

3. Si descriva il problema dei "cinque filosofi" illustrandone caratteristiche e possibili soluzioni. Si fornisca una possibile soluzione in linguaggio C, illustrandone il comportamento e la funzione dei vari semafori.

Modello del caso in cui diverse risorse sono comuni a diversi processi concorrenti (Dijkstra [1965]). Un tavolo è imbandito con 5 piatti di di riso e 5 bastoncini (cinesi) ciascuno tra due piatti. Intorno al tavolo siedono 5 filosofi. I filosofi pensano oppure mangiano. Per mangiare ogni filosofo ha bisogno di due bastoncini. I bastoncini possono essere ottenuti uno alla volta.

A parte le soluzioni "filosofiche" (insegnare ai filosofi a mangiare con 1 solo bastoncino, etc.) la seguente soluzione azzerra la concorrenza:

```
init (mutex, 1);
while (true) {
  Pensa ();
  wait (mutex);
  Mangia ();
  signal (mutex);
}
```

Soluzione con deadlock:

```
init (chopstick[0], 1);
...
init (chopstick[4], 1);
while (true) {
  Pensa ();
  wait (chopstick[i]);
  wait (chopstick[(i+1)mod5]);
  Mangia ();
  signal (chopstick[i]);
  signal (chopstick[(i+1)mod5]);
}
```

Soluzione completa:

```
while (TRUE) {
  think ();
  takeForks (i);
  eat ();
  putForks (i);
}
takeForks (int i) {
  wait (mutex);
  state[i] = HUNGRY;
  test (i);
  signal (mutex);
  wait (sem[i]);
}
test (int i) {
 if (state[i]==HUNGRY && state[LEFT]!=EATING &&
     state[RIGHT]!=EATING) {
   state[i] = EATING;
   signal (sem[i]);
 }
}
```

```
int state[N]
init (mutex, 1);
init (sem[0], 0); ...; init (sem[4], 0);

putForks (int i) {
  wait (mutex);
  state[i] = THINKING;
  test (LEFT);
  test (RIGHT);
  signal (mutex);
}
```

Spiegazione ...

4. Si illustri l'*algoritmo del banchiere* riportandone descrizione e pseudo-codice. Analizzando l'esempio successivo, con processi $(P_1, \ldots, P_5)$ e risorse (R_1, R_2, R_3), si indichi se lo stato è sicuro (riportando una possibile sequenza sicura) o non sicuro (indicandone la ragione).

Processo	Fine	Assegnate			Massimo			Necessità			Disponibilità		
		R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	F	0	1	0	3	2	4				1	1	1
P_2	F	0	0	0	3	3	2						
P_3	F	1	0	0	2	1	0						
P_4	F	1	1	1	1	3	2						
P_5	F	0	1	0	2	1	1						

L'algoritmo del banchiere serve per evitare il deaclock nel caso di risorse con instanze multiple (altrimenti è sufficiente un algoritmo di determinazioe dei cicli sul grafo di assegnazione).

Lo stato dell'esempio non è sicuro:
P_3 (disponibilità = 2, 1, 1)
P_5 (disponibilità = 2, 2, 1)
P_4 (disponibilità = 3, 3, 2)
P_2 (disponibilità = 3, 3, 2)
e P_1 non è eseguibile (necessità = 3, 1, 4).

Verifica di una richiesta:

```
se
  per ogni j Richieste[i][j]<=Necessit[i][j]
  AND
  per ogni j Richieste[i][j]<=Disponibili[j]
  ALLORA
    per ogni j Disponibili[j]=Disponibili[j]-Richieste[i][j]
    per ogni j Assegnate[i][j]=Assegnate[i][j]+Richieste[i][j]
    per ogni j Necessit[i][j]=Necessit[i][j]-Richieste[i][j]
```

Verifica di uno stato:

```
1.
Inizializza
Fine [i] = falso per tutti i Pi
2.
Trova un Pi per cui
Fine[i]=falso && per ogni j Necessit[i][j]<=Disponibili[j]
Se tale i non esiste vai al passo 4
3.
per ogni j Disponibili[j]=Disponibili[j]+Assegnate[i][j]
Fine[i] = vero
Vai al passo 2
4.
Se Fine[i]=vero per tutti i Pi
Allora il sistema  in uno stato sicuro
```

5. Si riportino i comandi UNIX per effettuare le operazioni indicate, utilizzando eventuali ridirezioni e pipe:
   - conteggiare il numero di caratteri di tutti i file di estensione ‘‘c’’ nel direttorio corrente, ordinando tale elenco in ordine numerico decrescente.
   - visualizzare l’elenco dei file contenuti nell’albero di direttori con radice la proprio home directory, il cui nome contiene le vocali ‘‘a’’, ‘‘b’’ e ‘‘c’’ (in quest’ordine), estensione ‘‘txt’’, dimensione maggiore di 2048 byte e sui quali l’utente abbia il diritto di scrittura.
   - modificare tutti i permessi dei file di estensione “exe” contenuti nel secondo livello gerarchico di direttori a partire da ‘‘/home/usr/’’ aggiungendo il permesso di esecuzione.
   - ordinare l’elenco delle righe del file ‘‘testo.txt’’ in base al secondo campo in ordine alfabetico crescente.
   - ricercare tutti i file del direttorio corrente che contengono almeno una volta una delle seguenti stringhe ‘‘main’’, ‘‘Main’’, ‘‘MAIN’’.
   - utilizzando SED si sostituiscano il tutte le righe del file ‘‘mio.txt’’ che incominciano per ‘‘START’’ le stringa ‘‘LINUX’’ oppure ‘‘linux’’ con la stringa ‘‘Linux’’ e in tutte le righe che finiscono per ‘‘END’’ la stringa ‘‘Unix’’ oppure ‘‘unix’’ con la stringa ‘‘UNIX’’.

```
1. find . -name "*.c" -type f -exec wc -c \{} \; | sort -nr
2. find ~ -name "*a*b*c*.txt" -type f -size +2048c -writable
3. find /home/usr/ -mindepth 2 -maxdepth 2 -name "*.exe" -type f -exec chmod +x \{} \;
4. sort -k 2 testo.txt
5. grep -e "main" -e "Main" -e "MAIN" ./*
   find . -maxdepth 1 -type f -exec grep -H -e "main" -e "Main" -e "MAIN" \{} \;
6. sed -e '/^START/{s/LINUX/Linux/g;s/linux/Linux/g}' -e '/^END/{s/UNIX/Unix/g;s/unix/Unix/g}' mio.txt
```

6. Un primo file, di tipo “quantità”, specifica su ciascuna riga il nome di un determinato prodotto e la relativa quantità. Un secondo file, di tipo “prezzo”, memorizza su ciascuna riga il nome di un prodotto e il relativo prezzo. Si osservi che lo stesso prodotto può comparire più volte nei due file, indicandone disponibilità e costo in diversi punti vendita.

   Si scriva uno script AWK in grado di:

   - Ricevere sulla riga di comando il nome di tre file. Il primo file è di tipo “quantità” e il secondo di tipo “prezzo”. Il terzo file deve essere generato dallo script.
   - Indicare per quali prodotti viene specificata la quantità ma non il prezzo.
   - Indicare per quali prodotti esiste almeno un prezzo ma non la quantità disponibile.
   - Per i prodotti per i quali è specificata tanto almeno una quantità quanto almeno un prezzo, lo script memorizzi nel terzo file una riga per ciascun prodotto, specificandone la disponibilità totale, il prezzo medio e il valore commerciale del prodotto (prodotto quantità per prezzo medio).

   Il seguente esempio riporta i file di ingresso e di uscita.

File quantità	File prezzo	File di uscita
Book 3	Book 50.5	Pen 30 4.8 144
Pen 10	Pen 5.4	Pencil 7 1.5 10.5
Pencil 4	Pencil 2.0	Book 13 29.7333 386.533
Book 2	Book 20.5	
Pen 20	Pen 4.2	Warning: product Eraser has no price!
Pencil 3	Pencil 1.0	Warning: product Jotter has no quantity!
Ereaser 3	Book 18.2	
Book 8	Jotter 12.3	
Eraser 1		

   Soluzione 1:

```
#!/usr/bin/awk -f
BEGIN {
  fileO = ARGV[3];
  ARGV[3] = "";
  # Erase output file
  command = "rm -rf " fileO
  system (command);
  # File 1 - Quantity
  while ((getline < ARGV[1])) {
    q[$1] = q[$1] + $2;
  }
  # File 2 -  Price
  while ((getline < ARGV[2])) {
    n[$1]++;
    p[$1] = p[$1] + $2;
  }
}
END {
  for (i in p) {
    p[i] = p[i] / n[i];
  }
  for (i in q) {
    if ( !(i in p) ) {
      print "Warning: product " i " has no price!"
    }
  }
  for (i in p) {
    if ( !(i in q) ) {
      print "Warning: product " i " has no quantity!"
    }
  }
  for (i in q) {
    if (i in p) {
      print i " " q[i] " " p[i] " " q[i]*p[i]
      print i " " q[i] " " p[i] " " q[i]*p[i] >> fileO
    }
  }
}
```

Soluzione 2:

```
BEGIN{
fileP="fileP" # has to be passed as parameter see awk commando -v optinon
fileOut="products_results.txt"
}
{
  #Reading Quantity File
  totAmount[$1]+=$2;
}
END{
  #Reading Price File
  while ((getline < fileP ))
  {
    sumPrice[$1]+=$2;
    numPrice[$1]++;
  }
  close(fileP);
  #Store Results in products_results.txt
  for (p in totAmount)
  {
      if (!(p in sumPrice))
       {
          print "warning: product " p " has no price!" > fileOut
          print "warning: product " p " has no price!"
       }else
        {
          avgp=sumPrice[p]/numPrice[p];
          print p " " totAmount[p] " " avgp " " avgp*totAmount[p] > fileOut
        }
  }
}
```