Caselle riservate

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

Sistemi Operativi

Compito d'esame

29 Giugno 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 descrivano le caratteristiche principali delle *pipe* per la comunicazione e la sincronizzazione tra processi. Se ne illustri l'utilizzo realizzando il seguente programma in linguaggio C. Un processo P genera un figlio F_1 che a sua volta genera un figlio F_2. F_2 legge da tastiera una intero, la trasmette a F_1 su una prima pipe e termina. F_1 trasforma il numero nel suo opposto e lo trasmette a P su una seconda pipe e termina. P visualizza il numero ricevuto su standard output e termina.

   I processi concorrenti possono essere indipendenti oppure cooperanti. Un processo è indipendente se non può essere infuenzato dagli altri processi e non può influenzare l'esecuzione di altri processi. Un processo è cooperante in caso contrario. La cooperazione può avvenire tramite lo scambio oppure la condivisione di dati. Scambio e condivissione di dati richiedono limplementazione di meccanismi opportuni.

   Una pipe permette di stabilire un flusso dati tra due processi. Ciascun processo, attraverso un file descriptor, accede a uno degli estremi della pipe. Può essere utilizzata per la comunicazione tra processi con un parente comune. Il flusso di dati half-duplex, i.e., i dati fluiscono solo in una direzione. Lettura e scrittura da e su pipe vengono effettuate mediante `read` e `write`. Esse son o bloccanti per pipe vouta o piena, rispettivamente.

   Il seguente è un esempio di utilizzo:

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

void P (int []);
void F1(int []);
void F2(int []);

int
main (
  int argc,
  char ** argv
  )
{
  int fd1[2];

  // Create pipe
  pipe (fd1);

  if (fork()!=0) {
    P(fd1);
  } else {
    F1(fd1);
  }

  return (0);
}

void P(int fd1[2]) {
  int n;
  close (fd1[1]);
  read (fd1[0], &n, sizeof (int));
  printf ("P  (pid=%d): %d\n", getpid(), n);
  exit (0);
}

void F1(int fd1[2]) {
  int n, fd2[2];
  close (fd1[0]);
  pipe (fd2);
  if (fork()!=0) {
    close (fd2[1]);
    read (fd2[0], &n, sizeof(int));
    printf ("F2 (pid=%d): receives %d\n", getpid (), n);
    n = (-n);
    printf ("F2 (pid=%d): transmit %d\n", getpid (), n);
    write (fd1[1], &n, sizeof(int));
    exit (0);
  } else {
    F2(fd2);
  }
}

void F2(int fd2[2]) {
  int n;
  close (fd2[0]);
  printf ("F2 (pid=%d): ", getpid ());
  scanf ("%d", &n);
  write (fd2[1], &n, sizeof(int));
  exit (0);
}
```

2. Con riferimento allo scheletro di una shell UNIX/Linux si scriva un programma, in linguaggio C, in grado di:
   - leggere da tastiera dei comandi Linux con eventuali parametri sulla riga di comando (ad esempio "`cp file1 file2`", "`ls -laR`", etc.)
   - eseguire tali comandi in background, procedendo (in foreground) alla lettura del comando successivo.
   - terminare il procedimento, ovvero la lettura di nuovi comandi Linux una volta ricevuto da tastiera il comando "`end`".

   Si osservi che, a scelta del candidato, la stringa "`end`" può essere utilizzata anche per delimitare la fine di ciascun comando letto da tastiera.

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

#define OPZIONI 10
#define LEN     10
#define MAX     80

int
main (
  int argc,
  char* argv[]
  )
{
  char tmp[MAX], **mat, *tmpP;
  FILE *fp;
  int i, nString;
  pid_t pid;

  // Alloco a priori ... anche se sasrebbe meglio essere piu' accurati
  mat = (char **) malloc (OPZIONI * sizeof (char *));
  for (i=0; i<OPZIONI; i++) {
    mat[i] = (char *) malloc (LEN * sizeof (char));
  }

  /*
   *  System Call: Exec
   */

  fp = fopen(argv[1], "r");
  if (fp==NULL) {
    fprintf (stderr, "Error opening file.");
    exit (1);
  }

  nString = 0;
  while (fscanf(fp, "%s", tmp) != EOF) {
    if (strcmp(tmp,"end") != 0) {
      strcpy (mat[nString], tmp);
      nString++;
    } else {
      printf ("Comando: ");
      for (i=0; i<nString; i++) {
        printf ("%s ", mat[i]);
      }
      printf ("\n");
      tmpP = mat[nString];
      mat[nString] = NULL;
      pid = fork();
      if (pid==0) {
        execvp (mat[0], mat);
      } else {
        sleep (3);
      }
      mat[nString] = tmpP;
      nString = 0;
    }
  }
  fclose(fp);

  return (1);
}
```

3. Si riportino quattro possibili soluzioni software al problema delle sezioni critiche con due processi denominati P_i e P_j. Si motivi l'erroneità oppure la correttezza di ciascuna soluzione.

Progresso non assicurato: P_i e P_j devono entrare nella SC ma in maniera alternata. `turn` inizializzata a 0 o 1.

```
Pi
while (TRUE) {
  while (turn==j);
  SC di Pi
  turn = j;
  sezione non critica
}
```

```
Pj
while (TRUE) {
  while (turn==i);
  SC di Pj
  turn = i;
  sezione non critica
}
```

Mutua esclusioni non assicurata: P_i e P_j possono entrare entrambi nella SC. `flag` di 2 elementi inizializzati a 0.

```
Pi
while (TRUE) {
  while (flag[j]);
  flag[i] = TRUE;
  SC di Pi
  flag[i] = FALSE;
  sezione non critica
}
```

```
Pj
while (TRUE) {
  while (flag[i]);
  flag[j] = TRUE;
  SC di Pj
  flag[j] = FALSE;
  sezione non critica
}
```

Soluzione con attesa non definita: P_i e P_j possono rimanere bloccati per sempre. `flag` di 2 elementi inizializzati a 0.

```
Pi
while (TRUE) {
  flag[i] = TRUE;
  while (flag[j]);
  SC di Pi
  flag[i] = FALSE;
  sezione non critica
}
```

```
Pj
while (TRUE) {
  flag[j] = TRUE;
  while (flag[i]);
  SC di Pj
  flag[j] = FALSE;
  sezione non critica
}
```

Algoritmo corretto (Peterson). `turn` e `flag` definite e inizializzate come in precedenza.

```
Pi
while (TRUE) {
  flag[i] = TRUE;
  turn = j;
  while (flag[j] && turn==j);
  SC di Pi
  flag[i] = FALSE;
  sezione non critica
}
```

```
Pj
while (TRUE) {
  flag[j] = TRUE;
  turn = i;
  while (flag[i] && turn==i);
  SC di Pj
  flag[j] = FALSE;
  sezione non critica
}
```

4. Si riportino i comandi UNIX per effettuare le operazioni indicate, utilizzando eventuali ridirezioni e pipe:
   - Nel direttorio "`/home/foo`" cercare i file con il nome che inizia con il carattere "`L`" e estensione "`txt`". In questi file ricercare al presenza della stringa "`laib`". Visualizzare il nome del file e l'intera riga in cui tale stringa viene rintracciata.
   - Trovare tutti i file di estensione "`txt`" nel direttorio "`/home`" memorizzati tra il livello di profondità 3 (incluso) e il livello di profondità 5 (incluso) dell'albero dei direttori e che siano leggibili. Di questi modificare il proprietario in "`ugo`".
   - Per ogni file di estensione "`txt`" presente nel direttorio corrente ricavare il nome e il numero di caratteri presenti nel file. L'elenco venga ordinato in base al numero di caratteri in ordine numerico inverso e memorizzato nel file "`stat.txt`".
   - Fare un esempio di utilizzo di ciascuno dei tre filtri `cut`, `tr`, `uniq`, spiegando l'effetto di ciascun comando.
   - Una applicazione C è formata dai file `main.c`, `f1.c`, `f2.c` e `main.h`. Scrivere un Makefile con due target. Il primo sia in grado di compilare l'applicazione denominando l'eseguibile `myapp`. Il secondo rimuova eventuali file temporanei e sposti l'eseguibile nel direttorio "`/user/bin`".

```
find /home/foo -name "L*.txt" -exec grep -H "laib" '{}' \;

find /home -mindepth 3 -maxdepth 5 -name "*.txt" -readable \
  -exec chown "ugo" '{}' \;

find . -name "*.txt" -exec wc -c '{}' \; | sort -rn -k 1,1 > stat.txt

cut -c2 test.txt
```

Visualizza il secondo carattere di tutte le righe del file `test.txt`.

```
echo ciao | tr a-z A-Z
```

Converte minuscole in maiuscole.

```
uniq -d test.txt
```

Stampa solo le linee duplicate del file test.txt

Makefile:

```
compile: main.c f1.c f2.c
        gcc -o myapp main.c f1.c f2.c
install:
        rm *.tmp
        cp myapp /user/bin
```

5. Scrivere un script AWK in grado di effettuare sostituzioni multiple in un testo soddisfacendo alle specifiche successive.

   Un primo file include su righe successive coppie di stringhe, nel formato:

   `stringa1 stringa2`

   Lo script deve sostituire ogni comparsa della `stringa1` con la `stringa2` in un file di ingresso, generando il corrispondente file di uscita. Si osservi che le stringhe possono degenerare in caratteri singoli e devono essere rintracciate e sostituite nel testo in ingresso anche come sotto-stringhe di stringhe di lunghezza maggiore. Si supponga inoltre che tutte le sostituzioni siano indipendenti e possano essere applicate in qualsiasi ordine. Lo script riceve tre stringhe sulla riga di comando. Tali stringhe individuano il file di conversione, quello di ingresso e quello di uscita da generare.

   Esempio
   Si supponga lo script venga eseguito come segue:

   `myScript file.txt inFile.txt outFile.txt`

   e che i file di nome `file.txt` e `inFile.txt` abbiano il contenuto di seguito indicato. Lo script deve generare il file `outFile.txt` riportato.

file.txt	inFile.txt	outFile.txt
awk Awk	ScriptawkInIngresso	ScriptAwkInIngressO
unix UNIX	Contenente unix e UNIX	COntenente UNIX e UNIX
o O	e testo vario	e testO variO

```
#default input: file.txt

BEGIN {
  #copy inFile in outFile (create outFile)
  inFile=ARGV[2]
  outFile=ARGV[3]

  ARGV[2]=""
  ARGV[3]=""

  print > outFile
  while ( (getline < inFile ) > 0) {
     print $0 >>  outFile
  }
  close(outFile);
}

{
  source=$1;
  dest=$2;
  while ( (getline line < outFile ) > 0) {
     gsub(source,dest,line);
     print line > outFile
  }
  close(outFile);
}
```

6. **Per i candidati iscritti al corso nell'anno accademico 2014–2015.**
   Indicare le principali caratteristiche delle tipologie di allocazione di file (contigua, concatenata, indicizzata). Che cosa si intende per FAT? Che cosa si intende per collegamento o link?

   Per i candidati iscritti al corso negli anni accademici 2012–2013 o 2013–2014.
   Si illustri la struttura di un file system UNIX con particolare riferimento alla memorizzazione di file e direttori. Che cosa si intende per collegamento o link?

   Vedere lucidi e relative spiegazioni oppure i testi consigliati.