Examen long de système d'exploitation

Semestre décalé
Département d'informatique IUT Villetaneuse

mercredi 16 janvier 2008

Remarques : tous les documents de cours, td/tp sont autorisés. Le barème est indicatif.

1 Langage de commande bash - 4 pts

Écrire un programme Bash qui prend en paramètre deux entiers i et j ($i \leq j$) et qui affiche des X en diagonale entre la ième colonne et la jème colonne, en commençant à la ligne 1 du terminal texte. Voici par exemple l'affichage produit par `port-cerin:$ /bin/bash e.sh 8 13`

```
|---------------------------------
|       X              <= ligne 1
|        X             <= ligne 2
|         X                 ''
|          X                ''
|           X               ''
|            X        <= ligne 6
|port-cerin:$
|
```

(N'affichez pas les `<= ligne`). Vous utiliserez les commandes suivantes :

```
tput cup 1 1
     Send the sequence to move the cursor to row 1, column 1 (the upper
     left corner of the screen, usually known as the ``home'' cursor
     position). tput m n: move the cursor at position (m,n)

tput clear
     Echo the clear-screen sequence for the current terminal.
```

- Un algorithme possible est de déplacer le curseur sur la colonne i de la ligne 'ligne' initialisee a 1, puis d'afficher 'X'. On incrémente i et ligne

de 1, et on recommence i-j+1 fois. C'est cet algorithme que nous codons ci-dessous :

```
#!/bin/bash

if [ $1 -gt $2 ]
then
    echo ‘‘le deuxieme parametre doit etre superieur ou egal au premier’’
    exit
fi

# on efface l'ecran
tput clear

# on affiche j-i+1 caracteres 'X'

ligne=1
for ((i=$(($1-1)) ; $i<$2; i=$((i+1))))
do
   tput cup $ligne $i
   ligne=$((ligne+1))
   echo -n 'X'
done

# on passe a la ligne
echo
```

2 Langage de commande bash - 5pts

On veut faire un script qui traite des fichiers particulier. Ces fichiers texte contiennent des entiers sur 5 colonnes. Les deux premières colonnes sont des entiers quelconques, la troisième colonne contient la somme des deux premières colonnes, la quatrième colonne contient le plus petit entier présent dans les deux premières colonnes et la cinquième colonne contient le plus grand entier présent dans les deux premières colonnes.
Exemple :

```
fichier 1 :
     3 4 7 1 7
     1 2 3 1 7
     7 6 13 1 7
fichier 2 :
```

```
5 6 11 3 8
6 5 11 3 8
3 8 11 3 8
```

Le programme Bash prend en entrée deux fichiers ayant ces propriétés, et les fusionne en préservant ces propriétés. Pour notre exemple, la sortie devra être le fichier :

```
3 4 7 1 8
1 2 3 1 8
7 6 13 1 8
5 6 11 1 8
6 5 11 1 8
3 8 11 1 8
```

Commencer par expliquer l'algorithme que vous suivez. Puis écrivez le programme Bash qui implémente l'algorithme. Rappel, vous pouvez vous servir du code suivant pour parcourir toutes les lignes d'un fichier texte :

```
#!/bin/bash

echo ``------Fichier en entree: $1------------''
cat < $1 | while true
  do
     read ligne
     # traiter la ligne
     ....
     ....
  done
```

```
#!/bin/bash

# On verifie les arguments
if [ $# -lt 1 ]
then
   echo ``Il faut donner deux noms de fichier en argument''
   exit
fi

if !(test -f $1)
then
   echo ``Fichier $1 inconnu !''
   exit
```

```
fi

if !(test -f $2)
then
   echo ‘‘Fichier $1 inconnu !’’
   exit
fi

# on efface l’ecran
tput clear

# on commence par isoler le max et le min des deux fichiers.
# Pour cela, il suffit de lire la première ligne de chaque
# fichier. On suppose que les fichiers sont bien formatés :
# chaque colonne est séparée par un blanc.
min1=‘head -n 1 $1 | cut -d " " -f 4‘
max1=‘head -n 1 $1 | cut -d " " -f 5‘
min2=‘head -n 1 $2 | cut -d " " -f 4‘
max2=‘head -n 1 $2 | cut -d " " -f 5‘

# Parmi 2 fois deux valeurs, on sait trouver le plus grand et le plus
# petit

min=$min1
max=$max1

if [ $min2 -lt $min ]
then
    min=$min2
fi

if [ $max2 -gt $max ]
then
    max=$max2
fi

# A ce niveau, le min c’est $min, le max c’est $max
# Il reste a parcourir les deux fichiers et a recomposer
# le resultat

cat < $1 | while true
  do
```

```
    read ligne
    # traiter la ligne
    if [ "$ligne" = "" ]; then break; fi
    one=‘echo $ligne | cut -d " " -f 1‘
    two=‘echo $ligne | cut -d " " -f 2‘
    three=‘echo $ligne | cut -d " " -f 3‘
    echo $one $two $three $min $max
  done

# On effectue la meme chose sur le deuxieme fichier
# on commence par recuperer le max et le min

cat < $2 | while true
  do
    read ligne
    # traiter la ligne
    if [ "$ligne" = "" ]; then break; fi
    one=‘echo $ligne | cut -d " " -f 1‘
    two=‘echo $ligne | cut -d " " -f 2‘
    three=‘echo $ligne | cut -d " " -f 3‘
    echo $one $two $three $min $max
  done
```

3 Ordonnancement 3pts

On a deux processeurs et les tâches suivantes données avec leur durée : $T_1 = 1, T_2 = 5, T_3 = 2, T_4 = 6, T_5 = 3, T_6 = 4$.

Question 1 : après avoir trié les tâches selon l'ordre croissant de leur durée, les placer sur les deux processeurs selon l'algorithme suivant : les tâches sont placées dans l'ordre où on les a triées ; la tâche T_i en attente de processeur est placée sur le premier processeur qui se libère.

Question 2 : après avoir trié les tâches selon l'ordre décroissant de leur durée, placer les tâches sur les deux processeurs selon l'algorithme suivant : les tâches sont placées dans l'ordre où on les a triées ; la tâche T_i en attente de processeur est placée sur le premier processeur qui se libère.

Question 3 : est-ce que les deux ordonnancements obtenus ont la même durée ? Même question si toutes les tâches ont la même durée. Justifiez vos réponses.

4 Processus et tubes nommés 8pts

Implémentez la description suivante. On veut écrire trois programmes : 2 écrivains et 1 lecteur. Les écrivains envoient via deux tubes nommés différents des entiers à un autre exécutable. Le lecteur crée un fils (autre lecteur) puis lui délègue la moitié du travail : capter les entiers en provenance d'un des tubes pour en faire la somme. Le père capte les entiers en provenance de l'autre tube et en fait la somme également. C'est au moment où un écrivain termine qu'on affiche, au niveau du lecteur correspondant, la somme des valeurs reçues.

Techniquement, pour que le lecteur termine les écrivains enverront 0 comme dernière valeur comme suit :

```
Fenetre 1 :
port-cerin:$ ./ecrivain2
22
33
0

Fenetre 2 :
port-cerin:$ ./ecrivain1
11
66
0
```

```
Fenetre 3 :
port-cerin:$ ./lecteur
message: 22 (22)
message: 33 (55)
Somme au niveau du pere : 55
port-cerin:$ message: 11 (11)
message: 66 (77)
Somme au niveau du fils : 77
port-cerin:$
```

```
/* Ecrivain 1 */
#include<stdio.h>
#include<assert.h>
#include<errno.h>
#include<sys/fcntl.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <string.h>

#define MSIZE 64
#define FIFO "/tmp/ff-fifo"
int main (int argc, char *argv[])
{ int fd;
  char mbuf[MSIZE];int i;

  fd = open(FIFO, O_WRONLY);
  assert (fd >=0);
  for(i=1; i< argc; i++) {
      strncpy(mbuf, argv[i], MSIZE-1);
```

```
      //mbuf[MSIZE] = '\0';
      write(fd, mbuf, MSIZE);
  }
  close(fd);
  exit(EXIT_SUCCESS);
}


/* Ecrivain 2 */
#include<stdio.h>
#include<assert.h>
#include<errno.h>
#include<sys/fcntl.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <string.h>

#define MSIZE 64
#define FIFO "/tmp/gg-fifo"
int main (int argc, char *argv[])
{ int fd;
  char mbuf[MSIZE];int i;

  fd = open(FIFO, O_WRONLY);
  assert (fd >=0);
  for(i=1; i< argc; i++) {
      strncpy(mbuf, argv[i], MSIZE-1);
      //mbuf[MSIZE] = '\0';
      write(fd, mbuf, MSIZE);
  }
  write(fd, mbuf, 0);
  close(fd);
  exit(EXIT_SUCCESS);
}


/* Lecteur */
#include<stdio.h>
#include<assert.h>
#include<errno.h>
#include<sys/fcntl.h>
#include <stdlib.h>
```

```
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>


#define MSIZE 64
#define FIFOFF "/tmp/ff-fifo"
#define FIFOGG "/tmp/gg-fifo"
int
main()
{
     int             fd_ff, fd_gg;
     int             PID;
     char            mbuf[MSIZE];
     int             status;
     int             nbytes;

     status = mkfifo(FIFOFF, S_IRWXU);
     assert(status != -1 || errno == EEXIST);
     fd_ff = open(FIFOFF, O_RDWR, S_IRUSR | S_IWUSR);
     assert(fd_ff >= 0);

     status = mkfifo(FIFOGG, S_IRWXU);
     assert(status != -1 || errno == EEXIST);
     fd_gg = open(FIFOGG,  O_RDWR, S_IRUSR | S_IWUSR);
     assert(fd_gg >= 0);

     PID = fork();
     if (PID < 0) {          /* error occurred */
          fprintf(stderr, "Fork Failed");
          exit(-1);
     }
     if (PID == 0) {          /* Le fils */
       int             s_fils = 0,j;
          while ((nbytes = read(fd_ff, mbuf, MSIZE)) > 0) {
            j=atoi(mbuf);
            if(j==0) break;
            s_fils += j;
            printf("message: %s (%d)\n", mbuf,s_fils);
          }
          printf("Somme au niveau du fils : %d\n", s_fils);
          close(fd_ff);
```

```
        remove(FIFOFF);
        exit(EXIT_SUCCESS);
    } else {
      int             s_papa = 0,j=0;

        while ((nbytes = read(fd_gg, mbuf, MSIZE)) > 0) {
                j=atoi(mbuf);
            if(j==0) break;
            s_papa += j;
            printf("message: %s (%d)\n", mbuf,s_papa);
        }
        printf("Somme au niveau du pere : %d\n", s_papa);
        close(fd_gg);
        remove(FIFOGG);
        exit(EXIT_SUCCESS);
    }

}
```