Examen long de système d'exploitation

Département d'informatique – IUT Villetaneuse

Jeudi 26 mai 2011 – 14 : 00 — 17 : 00

Remarques : tous les documents de td/tp sont autorisés MAIS pas les documents de cours. Le barème est indicatif. Veuillez expliquer ce que vous faites et ne pas vous contenter d'un programme. Comme il s'agit d'un contrôle long et il est conseillé de lire le sujet dans son intégralité avant de commencer

1 Compréhension et développements de programmes Bash - 7 pts

1.1 Que fait le code suivant ? 1pt

Donnez un exemple d'utilisation ainsi que le résultat pour le code :

```
#!/bin/bash

echo "Entrez une phrase:"
read PH
cmpt=0
for i in $PH
do
    cmpt=‘expr $cmpt + ${i}‘
done
echo $cmpt
```

Le code se lance sans argument, il demande à saisir une phrase (suite de mots séparés par des blancs) qui doivent représenter des entiers et il calcule la somme des entiers de la phrase.

```
dyn208:~ christophecerin$ /bin/bash titi.sh
Entrez une phrase:
1 2 3
6
```

```
dyn208:~ christophecerin$ /bin/bash titi.sh
Entrez une phrase:
a b c
expr: non-numeric argument
expr: syntax error
expr: syntax error
```

1.2 Modification sur le code précédent - 1pt

Modifier le code précédent pour qu'il affiche le nombre de mots qui commencent par 'A' ou 'a'.

```
#!/bin/bash

echo "Entrez une phrase:"
read PH
cmpt=0
for i in $PH
do
     if [ "${i:0:1}" = "A" -o "${i:0:1}" = "a"  ]
     then
          cmpt=‘expr $cmpt + 1‘
     fi
done
echo $cmpt
```

1.3 Compléter le code précédent - 1pt

Modifier le code précédent pour afficher le nombre de mots qui terminent par 'A' ou 'a'.

```
#!/bin/bash

echo "Entrez une phrase:"
read PH
cmpt=0
for i in $PH
do
     if [ "${i:${#i}-1:1}" = "A" -o "${i:${#i}-1:1}" = "a"  ]
       then
          cmpt=‘expr $cmpt + 1‘
       fi
```

```
done
echo $cmpt
```

1.4 Développement d'un code - 4pts

La commande Unix `cal <month> <year>` permet d'afficher un calendrier sous la forme suivante :

```
$ cal 12 2009
   December 2009
 S  M Tu  W Th  F  S
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

On vous demande d'écrire un script Bash qui lira la sortie de la commande `cal` et la transformera, pour notre exemple, en :

```
S      6 13 20 27
M      7 14 21 28
Tu 1   8 15 22 29
W  2   9 16 23 30
Th 3 10 17 24 31
F  4 11 18 25
S  5 12 19 26
```

Veuillez commencer :

* par expliquer les difficultés techniques qui se posent ; (0.5pt)
* puis par expliquer vos approches afin de présenter un algorithme ; (0.5pt)
* et enfin vous coderez cet algorithme en Bash ; (3pts)

```
Idée d'algorithme :
   o préserver l'invariant suivant : à l'itération i de l'algorithme,
la variable LIG1 contient les informations pour le dimanche
   o préserver l'invariant suivant : à l'itération i de l'algorithme,
la variable LIG2 contient les informations pour le lundi
   o ...
   o on a donc besoin de 7 variables. Le problème technique qui se pose
est lié à la lecture des lignes et la gestion des blancs. Ainsi la
ligne "           1 3 4" est vue comme "1 3 4" => on perd
l'information sur les blancs de début de ligne.
```

o on va donc d'abord faire passer l'entrée dans le filtre vu dans le controle court afin d'avoir une entree de la forme :

```
Di;Lu;Ma;Me;Je;Ve;Sa
 ;  ;  ;  ;  1; 2; 3
 4; 5; 6; 7; 8; 9; 10
11;12;13;14;15;16; 17
18;19;20;21;22;23; 24
25;26;27;28;29;30; 31
```

(attention aux blancs et aux ; dans le format). Le fichier d'entrée est 1.txt

```
cat 1.txt | while read ligne; do for i in `echo ${ligne}`; do echo $i; done done
```

permet d'isoler chaque mot =>

```
Di;
Lu;
Ma;
Me;
Je;
Ve;
Sa;
;
;
;
;
1;
2;
3;
4;
5;
6;
7;
8;
....
25;
26;
27;
28;
29;
30;
31;
```

Il reste à ranger les différents mots dans les 7 variables intermédiaires...ici des fichiers :

```
#!/bin/bash
```

```
cat 1.txt | while read ligne; do j=1; for i in ‘echo ${ligne}‘;
do
case "$j" in
    "1" )
        LIG1="$LIG1 $i" ; echo -n "$i" >> LIG1;;
    "2" )
        LIG2="$LIG2 $i"; echo -n "$i" >> LIG2;;
    "3" )
        LIG3="$LIG3 $i"; echo -n "$i" >> LIG3;;
    "4" )
        LIG4="$LIG4 $i"; echo -n "$i" >> LIG4;;
    "5" )
        LIG5="$LIG5 $i"; echo -n "$i" >> LIG5;;
    "6" )
        LIG6="$LIG6 $i"; echo -n "$i" >> LIG6;;
    "7" )
        LIG6="$LIG6 $i"; echo -n "$i" >> LIG7 ;;
esac
j=‘expr $j + 1‘
done done
cat LIG1
echo
cat LIG2
echo
cat LIG3
echo
cat LIG4
echo
cat LIG5
echo
cat LIG6
echo
cat LIG7

new-host:SE christophecerin$ /bin/bash e.sh
Di;;4;11;18;25;
Lu;;5;12;19;26;
Ma;;6;13;20;27;
Me;;7;14;21;28;
Je;1;8;15;22;29;
Ve;2;9;16;23;30;

Note : on peut rajouter une instruction sed pour supprimer
les ; et faire de la mise en page :

....
done done
```

```
printf "%.2s %.2i %.2i %.2i %.2i %.2i\n" ‘sed ’s/;;/ 00 /g’ LIG1 | sed ’s/;/ /g’‘
printf "%.2s %.2i %.2i %.2i %.2i %.2i\n" ‘sed ’s/;;/ 00 /g’ LIG2 | sed ’s/;/ /g’‘
printf "%.2s %.2i %.2i %.2i %.2i %.2i\n" ‘sed ’s/;;/ 00 /g’ LIG3 | sed ’s/;/ /g’‘
printf "%.2s %.2i %.2i %.2i %.2i %.2i\n" ‘sed ’s/;;/ 00 /g’ LIG4 | sed ’s/;/ /g’‘
printf "%.2s %.2i %.2i %.2i %.2i %.2i\n" ‘sed ’s/;;/ 00 /g’ LIG5 | sed ’s/;/ /g’‘
printf "%.2s %.2i %.2i %.2i %.2i %.2i\n" ‘sed ’s/;;/ 00 /g’ LIG6 | sed ’s/;/ /g’‘
printf "%.2s %.2i %.2i %.2i %.2i %.2i\n" ‘sed ’s/;;/ 00 /g’ LIG7 | sed ’s/;/ /g’‘

rm LIG1 LIG2 LIG3 LIG4 LIG5 LIG6 LIG7
```

```
Ce qui donne :
new-host:SE christophecerin$ /bin/bash e.sh
Di 00 04 11 18 25
Lu 00 05 12 19 26
Ma 00 06 13 20 27
Me 00 07 14 21 28
Je 01 08 15 22 29
Ve 02 09 16 23 30
Sa 03 10 17 24 31
```

2 Mémoire partagée - 5pts

Écrire un programme C qui prend en paramètre un fichier contenant 7 lignes à 6 colonnes de la forme suivante :

```
Di 00 04 11 18 25
Lu 00 05 12 19 26
Ma 00 06 13 20 27
Me 00 07 14 21 28
Je 01 08 15 22 29
Ve 02 09 16 23 30
Sa 03 10 17 24 31
```

qui va créer 7 processus. Le premier processus fait la somme des entiers sur la première ligne, le deuxième processus fait la somme des entiers sur la deuxième ligne etc Chacune des sommes partielles est rangée en mémoire partagée. Enfin le processus père fait la somme des 7 sommes partielles et l'affiche.
Note : on peut utiliser popen() qui fera appel à un script ou une commande Bash permettant d'isoler la ième ligne du fichier passé en paramètre.
a) Commencez par expliquer les difficultés techniques et dégagez une idée d'algorithme ;
b) Donnez un programme C correspondant à votre idée d'algorithme.

```
/*
*
 */

#include <stdio.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/wait.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <errno.h>
#include <stdlib.h>

/* reverse:  reverse string s in place */
void
reverse (char s[])
{
  int i, j;
  char c;

  for (i = 0, j = strlen (s) - 1; i < j; i++, j--)
    {
      c = s[i];
      s[i] = s[j];
      s[j] = c;
    }
}

/* itoa:  convert n to characters in s */
void
itoa (int n, char s[])
{
  int i, sign;

  if ((sign = n) < 0)          /* record sign */
    n = -n;               /* make n positive */
  i = 0;
  do
    {                    /* generate digits in reverse order */
      s[i++] = n % 10 + '0';     /* get next digit */
```

```
    }
  while ((n /= 10) > 0);     /* delete it */
  if (sign < 0)
    s[i++] = '-';
  s[i] = '\0';
  reverse (s);
}

struct region
{                     /* Defines "structure" of shared memory */
  int SommePartielle[7];
};

int i, j, options;
char *k;

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

 /*  0                          -> program name      */
 /*  1                          -> Fichier d'entre   */

  int numberOfIntervals, interval;
  pid_t n[7];
  int status;
  struct region *rptr;
  int fd;

  if (argc != 2)
    {
      printf ("Please, specify a file name\n");
      exit (0);
    }

  /*  */


/* Create shared memory object and set its size */

  fd = open ("/tmp/myregion", O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
/*
```

```
fd = shm_open("/tmp/myregion", O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
*/
  if (fd == -1)
    {
    }
  /* Handle error */ ;

  if (ftruncate (fd, sizeof (struct region)) == -1)
    {
      printf ("ERROR\n");
    }
  /* Handle error */ ;


/* Map shared memory object */


  rptr = mmap (NULL, sizeof (struct region),
               PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
  if (rptr == MAP_FAILED)
    {
    }
  /* Handle error */ ;
  for(i = 0; i < 7; i++)
    rptr->SommePartielle[i]=0;

  /*  */
  for (i = 0; i < 7; i++)
    {
      n[i] = fork ();
      if (n[i] == 0)
     {               /* fils */
       int s=0;
       FILE *pp;
       char *c;
       char buf[255];

       k = calloc (255, sizeof (char));
       c = calloc (255, sizeof (char));
       /* on construit la commande Bash qui va isoler */
       /* la ième ligne de l'entrée */
       strcat (c, "head -n ");
```

```
      itoa (i+1, buf);
      strcat (c, buf);
      strcat (c, " ");
      strcat (c, argv[1]);
      strcat (c, " | tail -n 1");
      //printf ("c: %s\n", c);

      /* On lance la commande Bash fabriquée précédemment */
      if ((pp = popen (c, "r")) == NULL) perror ("popen error");

      /* On lit le résultat et on le stocke */
      if (fgets (buf, sizeof (buf), pp) == NULL)
        {
          printf ("Buf %s", buf);
          perror ("fgets error");
        }
      //printf ("Buf %s", buf);

      /* On explore le contenu de la ligne après avoir */
      /* éliminé le premier token qui ne sert pas */
      k = strtok (buf, " ");
      k = strtok (NULL, " ");
      for(j=0;j<5;j++){
        s += atoi (k);
           //printf ("s: %d\n", s);
        k = strtok (NULL, " ");
      }
         printf("Somme sur PID %d: %d\n",getpid(),s);
      rptr->SommePartielle[i] = s;
      //printf("%d\n",rptr->SommePartielle[i]);
      exit (0);
    }
  }
/* Suspend l'exécution du père */
/* et synchronisation avec tous les fils. */
for (j = 0; j < 7; j++)
  waitpid (n[j], &status, options);

/*  Print the results. */
int s = 0;
for (j = 0; j < 7; j++)
  s += rptr->SommePartielle[j];
```

```
  printf ("La somme totale est %d\n", s);

  munmap (rptr, sizeof (struct region));
  /*
     shm_unlink("/tmp/myregion");
   */
  close (fd);
  return 0;

}
```

```
cerin@zinzolin:~/public_html/SE$ ./a.out 1.txt
Somme sur PID 505: 66
Somme sur PID 506: 70
Somme sur PID 503: 58
Somme sur PID 509: 80
Somme sur PID 511: 85
Somme sur PID 507: 75
Somme sur PID 504: 62
La somme totale est 496
```

3 Tubes nommés - 5pts

Soit le code suivant :

```
#include<stdio.h>
#include<errno.h>
#include<assert.h>
#include<sys/fcntl.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

/*
File mode bits:

S_IRWXU
read, write, execute/search by owner
S_IRUSR
```

```
read permission, owner
S_IWUSR
write permission, owner
S_IXUSR
execute/search permission, owner
S_IRWXG
read, write, execute/search by group
S_IRGRP
read permission, group
S_IWGRP
write permission, group
S_IXGRP
execute/search permission, group
S_IRWXO
read, write, execute/search by others
S_IROTH
read permission, others
S_IWOTH
write permission, others
*/

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

        status = mkfifo(FIFOFF, S_IRWXU);
        /* on vérifie que la création du tube s'est bien passé */
        assert(status != -1 || errno == EEXIST);
        fd_ff = open(FIFOFF, O_RDWR, S_IRUSR | S_IWUSR);
        /* on vérifie que la création du tube s'est bien passé */
        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;
                while ((nbytes = read(fd_ff, &val, sizeof(int))) > 0) {
                        if (val == 0)
                                break;
                        s_fils += val;
                }
                printf("Somme au niveau du fils du pere: %d\n", s_fils);
                close(fd_ff);
                remove(FIFOFF);
                exit(EXIT_SUCCESS);
        } else {
                do {
                        printf("Enter un entier (0 pour terminer)\n");
                        scanf("%d", &val);
                        if (val >= 0)
                                write(fd_gg, &val, sizeof(int));
                } while (val != 0);
                close(fd_gg);
                remove(FIFOGG);
                exit(EXIT_SUCCESS);
        }

}
```

a) Faites un commentaire du code en précisant par exemple comment on fait terminer le programme.
b) Que va t-il se passer si on lance simultanément deux fois l'exécutable correspondant à ce code ?
c) Modifier le code précédent pour ajouter un deuxième fils qui accède à la sortie du tube de manière concurrente avec le premier fils et qui lui aussi fera la somme des valeurs reçues puis affichera la somme.

a) Le code crée un processus fils. Le père demande à saisir un entier qui est transmis au fils via un tube. Le programme termine à partir du moment où l'utiisateur tape 0... qui est transmis au fils qui peut aussi terminer. Le processus fils fait la somme des valeurs reçues et affiche cette somme. En fait il y a un problème de conception avec le code donné car d'un coté on écrit sur fd_gg et de l'autre on lit sur le tube fd_ff.

b) Imaginons que les problèmes exposés au a) soient réglés.
Les deux programmes utilisent le même tube. Il va y avoir concurrence pour accéder aux valeurs en sortie du tube car le tube est unique. Il va y avoir un problème au moment de terminer car le promier lecteur qui lira consommera la valeur 0 qui le fait terminer alors que l'autre lecteur ne recevra pas le 0 qui le fera terminer.

c) Il ne faut pas oublier d'envoyer une 2ème fois 0 pour que le deuxième fils termine. On obtient :

```
#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, PID1;
        int             val;
        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;
         while ((nbytes = read(fd_gg, &val, sizeof(int))) > 0) {
                 if (val == 0)
                         break;
                 s_fils += val;
         }
         printf("Somme au niveau du fils %d: %d\n",getpid(), s_fils);
         //close(fd_ff);
         //remove(FIFOFF);
         exit(EXIT_SUCCESS);
 } else {

PID1 = fork();
if (PID1 < 0) {                /* error occurred */
  fprintf(stderr, "Fork Failed");
  exit(-1);
}
if (PID1 == 0) {                /* Le fils */
  int             s_fils = 0;
  while ((nbytes = read(fd_gg, &val, sizeof(int))) > 0) {
    if (val == 0)
       break;
    s_fils += val;
  }
  printf("Somme au niveau du fils %d: %d\n", getpid(),s_fils);
  //close(fd_ff);
  //remove(FIFOFF);
  exit(EXIT_SUCCESS);
```

```
        }
                do {
                        printf("Enter un entier (0 pour terminer)\n");
                        scanf("%d", &val);
                        if (val >= 0){
                                write(fd_gg, &val, sizeof(int));
                        }
                } while (val != 0);
                write(fd_gg, &val, sizeof(int));
                close(fd_gg);
                remove(FIFOGG);
                exit(EXIT_SUCCESS);
         }

}
```

```
cerin@weblipn:~/public_html/SE$ ./a.out
Enter un entier (0 pour terminer)
1
Enter un entier (0 pour terminer)
2
Enter un entier (0 pour terminer)
3
Enter un entier (0 pour terminer)
4
Enter un entier (0 pour terminer)
0
Somme au niveau du fils 23437: 10
cerin@weblipn:~/public_html/SE$ Somme au niveau du fils 23438: 0
```

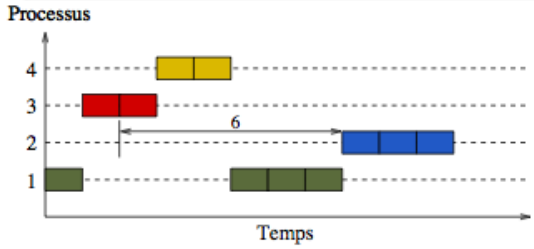
4 Ordonnancement – 3pts

On se propose de revenir sur l'algorithme Shortest Remaining Time First (les processus dont il reste le temps d'exécution le plus court sont exécutés en premier.)
a) L'ordonnnancement donné à la Figure 1 vous semble t-il correct ? Expliquez. Pouvait-on faire d'autres choix ?
b) On suppose que l'on a une machine à 2 processeurs. Veuillez proposer un placement et un ordonnacement pour les tâches de la Figure 1.

a) L'ordonnancement est correct. La discussion que l'on peut avoir

Données des processus

Processus	Durée	Temps de soumission
1	4	0
2	3	2
3	2	1
4	2	3



Processus	Temps d'attente	Temps d'exécution
1	0	8
2	6	9
3	0	2
4	0	2
Moyenne	1.5	5.25

FIGURE 1 – Politique SRT

concerne ce qui se passe au temps T=5 ou il reste à exécuter 3 unités de temps du processus 1 et 3 unités de temsp du processus 2. En cas d'égalité, nous avons choisi ici d'appliquer la règle du "premier arrivé, premier servi".

b) Avec deux processeurs/coeurs, nous pouvons faire le placement suivant :
P1 : 1111222
P2 : 3344