Examen de système d'exploitation

Département d'informatique – IUT Villetaneuse

mercredi 24 juin 2009 – 2H

Prénom : **Nom :**

Remarques : tous les documents de cours, td/tp sont autorisés. Le barème est indicatif. Répondre directement sur le sujet.

1 Langage de commande bash - 8 pts

Un fichier texte est organisé en colonnes avec le point virgule (;) comme séparateur de colonnes. La première colonne contient un nom et la deuxième colonne contient un prénom. Ecrire un script Bash qui prend en paramètre un fichier texte dans ce formalisme et qui va afficher le nombre total de lettres de tous les noms de famille, (sans compter les espaces). Pour le fichier suivant, le script devra afficher 20 :

```
Cérin;Christophe
Leclerc ;Jean
Lapoisse  ; Martine
```

Il convient de remplir le squelette qui suit et de faire une présentation de l'algorithme que vous codez ainsi que des difficultés et problèmes techniques divers :

```
#!/bin/bash
# Présentation de mon algorithme :
#
#
#
#
#
#
#
#
#
#
```

```
#

if [ ! -e $1 ]; then
   echo "le fichier n existe pas!"
   exit 0
fi
echo "------Fichier en entree: $1-------------"
somme=0
cat < $1 | while true
do
    read ligne
    if [ "$ligne" = "" ]; then
       #
       # Afficher le résultat ici
       #




       break;
    fi
    # traitement de $ligne









done
```

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

```
# Présentation de mon algorithme :
#
# Pour toutes les lignes du fichier, on isole la première colonne
# (cut), puis on ne garde que les majuscules et les minuscules (tr -dc)
# puis on passe cette chaine a wc qui renvoie le nombre de caractères
# de la chaine. Cette valeur est rangée dans j puis j est ajoutée à
# somme qui est la valeur résultat.
#
# On aurait aussi pu construire l'élimination des ' ' (au lieu de garder
# les majuscules/minuscules) et calculer la longueur de la chaine a par
# la notation ${#a}. On accède au ieme caractère de a par la notation
# ${a:$i:1}
#
# Notez également que dans l'exemple donné dans le sujet il y a une
# lettre accentuée (é) et donc le résultat retouné sera ici 19.
#

if [ ! -e $1 ]; then
   echo "le fichier n existe pas!"
   exit 0
fi
echo "------Fichier en entree: $1-------------"
somme=0
cat < $1 | while true
do
    read ligne
    if [ "$ligne" = "" ]; then
       #
       # Afficher le résultat ici
       #
       echo "Nombre total de caractères = $somme"
       break;
    fi
    # traitement de $ligne
    j=`echo "$ligne" | cut -d';' -f 1 | tr -dc "A-Za-z" | wc -c`
    somme=$((somme+j))
done
```

2 Communication locale sous Unix - 2pts + 5pts + 5pts

A) À l'aide des instructions `fork()`, `wait()`, `waitpid()`, `exit()` (au choix) écrire un programme C qui gère 4 processus qui affichent un des messages suivants : `Titi`, `Toto`, `Tata`, `Tutu`. Le processus père devra terminer après tous ses fils. Voici un exemple de compilation et d'execution :

```
$ gcc -Wall totititatatutu.c
$ ./a.out
Toto
Tutu
Tata
Titi
Le pere termine
$
```

```
/* Correction Toto, Tata, Titi, Tutu */
/* Ici on a choisi de faire en sorte que c'est le pere */
/* qui crée les 3 fils. On auru pu envisager de faire creer */
/* un fils par un fils... (à faire) */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main(int argc, char *argv[])
{
  int status;
  pid_t wpid, pid1, pid2, pid3;
  /* fork another process */
  pid1 = fork();
  if (pid1 < 0) { /* error occurred */
    fprintf(stderr, "Fork Failed");
    exit(-1);
  }
  else if (pid1 == 0) { /* child process */
    printf("Toto\n");
    exit(0);
  }

  /* fork another process */
  pid2 = fork();
  if (pid2 < 0) { /* error occurred */
    fprintf(stderr, "Fork Failed");
    exit(-1);
  }
  else if (pid2 == 0) { /* child process */
    sleep(1);
    printf("Titi\n");
    exit(0);
```

```
  }

  /* fork another process */
  pid3 = fork();
  if (pid3 < 0) { /* error occurred */
    fprintf(stderr, "Fork Failed");
    exit(-1);
  }
  else if (pid3 == 0) { /* child process */
    printf("Tata\n");
    exit(0);
  }
  else { /* parent process */
    printf("Tutu\n");
    /* parent will wait for the children to complete */
    /* On synchronise les fils */
    wpid = waitpid(pid2, &status, WUNTRACED);
    wpid = waitpid(pid3, &status, WUNTRACED);
    wpid = waitpid(pid1, &status, WUNTRACED);
    /* Le pere peut terminer */
    printf("Le pere termine\n");
    exit(0);
  }
}
```

B) J'ai envoyé un code qui calculait une estimation du nombre π par une méthode de Monte Carlo. Le code est rappelé ci-dessous. On vous demande de le modifier afin de saisir un entier n qui représente le nombre de processus qui vont participer au calcul. Chaque processus calcule alors $niter/n$ points.
Faites d'abord une explication des problèmes à traiter. Si cela vous parait insurmontable ou que vous ne voyez pas du tout où sont les problèmes, traitez au moins le cas $n = 4$.

```
/*
 * Ce code implemente le calcul d'une estimation de pi par une methode
 * de Monte Carlo. Nous activons 2 processus qui effectuent chacun
 * la moitie des calculs. Une memoire partagee est mise en oeuvre afin
 * de recuperer au niveau du processus pere le calcul du fils.
 *
 * Pour comprendre l'algorithme :
 *    - lire http://math.fullerton.edu/mathews/n2003/MonteCarloPiMod.html
 *    - Le code ci dessous n'implemente pas directement l'algorithme
```

```
 *      de la page ci-dessus. Veuillez comprendre les differences.
 */

#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>
#include <stdio.h>
#include <math.h>
#include <string.h>
#define SEED 35791246

struct region { /* Defines "structure" of shared memory */
double          myDouble;
};


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

        int             niter = 0;
        double          x, y;
        int             i, count = 0;/* # of points in the 1st quadrant of
                                        * unit circle */
        double          z;
        double          pi;

        pid_t           n;
        struct region  *rptr;
        int             fd;

        /* The number of intervals is a command line argument. */
        printf("Enter the number of iterations used to estimate pi: ");
        scanf("%d", &niter);

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

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

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

/* Map shared memory object */
rptr = mmap(NULL, sizeof(struct region),
            PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
if (rptr == MAP_FAILED) {
}

/* Now compute the area. */
n = fork();
if (n == 0) {                /* fils */
        /* initialize random numbers */
        srand(SEED + 1);
        count = 0;
        for (i = 0; i < niter / 2; i++) {
                x = (double) rand() / RAND_MAX;
                y = (double) rand() / RAND_MAX;
                z = x * x + y * y;
                if (z <= 1)
                        count++;
        }
        rptr->myDouble = count;
} else {                /* pere */
        /* initialize random numbers */
        srand(SEED);
        count = 0;
        for (i = 0; i < niter / 2; i++) {
                x = (double) rand() / RAND_MAX;
                y = (double) rand() / RAND_MAX;
                z = x * x + y * y;
                if (z <= 1)
                        count++;
        }
        /* Suspend l'exécution du processus dans lequel il est
         * appelé jusqu'à réception d'une valeur status d'un
         * processus enfant terminé.
         */
        wait(0);
        /* Print the results. */
        count = (double) (rptr->myDouble) + count;
        pi = (double) count / niter * 4;
        printf("# of trials= %d , estimate of pi is %1.15f \n", niter,
               (float) pi);
}
```

```
        munmap(rptr, sizeof(struct region));
        close(fd);
        return 0;
}
```

Page réservée à résumer les problèmes, les approches, les solutions envisagées

Le principal problème vient du fait que l'on ne connait pas la valeur de n à l'avance donc on ne peut pas déterminer la taille de la mémoire partagée à l'avance. Il faut donc la créer dynamiquement. La mémoire partagée sert à ce que les fils rangent les résultats partiels de leurs calculs et sert aussi au père pour accéder à l'ensemble des résultats partiels afin de recomposer le résultat final.

Dans le code ci-dessous, c'est après la saisie de n que l'on crée par un `truncate` de taille $n-1$ la zone partagée. On créee également un tableau avec `malloc` pour ranger les identificateurs des $n-1$ processus. Ce tableau dynamique est libéré à la fin par un `free`.

Note : dans le code qui suit, on suppose que $niter/n$ est un entier. Vous pouvez chercher à compléter le code dans le cas contraire.

Page réservée à votre solution

```
/*
 * Ce code implemente le calcul d'une estimation de pi par une methode
 * de Monte Carlo. Nous activons n>1 processus qui effectuent chacun
 * niter/n des calculs. Une memoire partagee est mise en oeuvre afin
 * de recuperer au niveau du processus pere les calculs des fils.
 *
 * Note : on suppose que niter/n est en entier
 * Note : veuillez remarquer les facteur (n-1) qui permettent de
 *        gérer la mémoire partagée (déclaration, utilisation, libération)
 *
 * Pour comprendre l'algorithme :
 *    - lire http://math.fullerton.edu/mathews/n2003/MonteCarloPiMod.html
 *    - Le code ci dessous n'implemente pas directement l'algorithme
 *      de la page ci-dessus. Veuillez comprendre ls differences.
 *
 * Exemple d'exécution :
 * $ ./a.out
 * Enter the number of iterations used to estimate pi:
 * 1000000
 * Enter the number n of processes:
 * Shoud verify: n>1 and niter modulo n == 0:
 * 8
 * Nombre de hit: 98045 pour le processus pere
 * Nombre de hit: 97981 pour le processus 0
 * Nombre de hit: 98169 pour le processus 1
 * Nombre de hit: 98210 pour le processus 2
 * Nombre de hit: 97875 pour le processus 3
 * Nombre de hit: 98268 pour le processus 4
 * Nombre de hit: 98115 pour le processus 5
 * Nombre de hit: 98353 pour le processus 6
 * # of trials= 1000000 , estimate of pi is 3.140064001083374
 */

#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>
#include <stdio.h>
#include <math.h>
#include <string.h>
```

```
#define SEED 35791246

struct region {  /* Defines "structure" of shared memory */
        int          mySomme;
};


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

        int             niter = 0;
        double          x, y;
        int             i, j, n, count = 0;  /* # of points in the 1st
                                              * quadrant of unit circle */
        double          z;
        double          pi;

        pid_t          *nn, wpid;
        struct region  *rptr;
        int             fd, status;

        /* The number of intervals. */
        printf("Enter the number of iterations used to estimate pi:\n");
        scanf("%d", &niter);

        /* The number of processes we activate. */
        do {
                printf("Enter the number n of processes: \n");
                printf("Shoud verify: n>1 and niter modulo n == 0: \n");
                scanf("%d", &n);
        } while ((n < 2) || ((niter % n) != 0));

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

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


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

```
/* Map shared memory object */
rptr = mmap(NULL, sizeof(struct region) * (n - 1),
            PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
if (rptr == MAP_FAILED) {
}
 /* Handle error */ ;

/* Allocate an array to store th (n-1) childrens PID */
nn = (pid_t *) malloc((n - 1) * sizeof(pid_t));
if (nn == NULL) {
  perror("Problem with allocation\n");
}
/* Now compute the area. */
for (i = 0; i < (n-1); i++) {
        nn[i] = fork();
        if (nn[i] == 0) {        /* fils */
                /* initialize random numbers */
                srand(SEED + 1 + i);
                count = 0;
                for (j = 0; j < niter / n; j++) {
                        x = (double) rand() / RAND_MAX;
                        y = (double) rand() / RAND_MAX;
                        z = x * x + y * y;
                        if (z <= 1)
                                count++;
                }

                (rptr+i)->mySomme = count;
                exit(0);
        }
}

/* pere */
/* initialize random numbers */
srand(SEED);
count = 0;
for (i = 0; i < niter / n; i++) {
        x = (double) rand() / RAND_MAX;
        y = (double) rand() / RAND_MAX;
        z = x * x + y * y;
        if (z <= 1)
                count++;
}

/* Suspend l'exécution du processus dans lequel il est appelé */
/*
 * jusqu'à réception d'une valeur status d'un processus enfant
```

```
         * terminé.
         */
        for(i=0;i<(n-1);i++)
          wpid = waitpid(nn[i], &status, WUNTRACED);

        /* Le pere recompose le resultat */
        /* Print the results. */
        printf("Nombre de hit: %d pour le processus pere\n",count);
        for(i=0;i<(n-1);i++){
          printf("Nombre de hit: %d pour le processus %d\n",(rptr+i)->mySomme,i);
          count = (double) ((rptr+i)->mySomme) + count;
        }


        pi = (double) count / niter * 4;
        printf("# of trials= %d , estimate of pi is %1.15f \n", niter,
               (float) pi);

        /* on libére la place occupée */
        free(nn);
        munmap(rptr, sizeof(struct region)* (n - 1));
        close(fd);
        return 0;

}
```

C) Écrire un projet C (c.à.d deux programmes C) qui au moyen des tubes nommés implémente le fonctionnement suivant : le premier exécutable saisit un entier, s'il est positif il l'envoie au deuxième, sinon il recommence une saisie – il n'y a pas de condition de terminaison de la boucle. Le deuxième exécutable réceptionne les entiers, les multiplie par 2 et les renvoie au premier exécutable. Ce dernier affiche l'entier reçu.
Note : les tubes nommés peuvent être créés en dehors du programme.

```
Dans une premiere fenetre :
$ ./test
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)
0
Somme au niveau du fils du pere: 12

Dans une deuxieme fenetre :
```

```
$ ./test1
Recu 1
Recu 2
Recu 3
Recu 0

#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;
        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_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);
        }

}


#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 argc, char *argv[])
{
        int             ff, gg;
        int             val, nbytes;

        ff = open(FIFOFF, O_RDWR, S_IRUSR | S_IWUSR);
        assert(ff >= 0);
        gg = open(FIFOGG, O_RDWR, S_IRUSR | S_IWUSR);
        assert(gg >= 0);

        while ((nbytes = read(gg, &val, sizeof(int))) > 0) {
```

```
                printf("Recu %d\n", val);
                val *= 2;
                write(ff, &val, sizeof(int));
                if (val == 0)
                        break;
        }
        close(ff);
        close(gg);
        exit(EXIT_SUCCESS);
}
```