Rapport

GLO-2001 Systèmes d'exploitation pour ingénieurs

Travail pratique 1 – Hiver 2019

Utilisation pratique de Linux et des threads POSIX

Fait par : Raphaël Sylvain 111 124 564

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| X | IFT |  | GLO |  | GIF |  | Autre |

*Résultat:* \_\_\_\_\_ sur 100

### 1. Ligne de commandes (30 pts)

1.1 Examiner un répertoire (3 points)



etu1@ubuntu:~$ cd /

etu1@ubuntu:/$



etu1@ubuntu:/$ ls -l

total 96

drwxr-xr-x 2 root root 4096 Feb 2 11:34 bin

drwxr-xr-x 3 root root 4096 Feb 2 12:25 boot

drwxrwxr-x 2 root root 4096 Oct 28 2016 cdrom

drwxr-xr-x 18 root root 3840 Feb 2 14:33 dev

drwxr-xr-x 130 root root 12288 Feb 2 12:25 etc

drwxr-xr-x 3 root root 4096 Oct 28 2016 home

lrwxrwxrwx 1 root root 33 Feb 2 12:24 initrd.img -> boot/initrd.img-4.4.0-141-generic

^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v

drwxr-xr-x 2 root root 4096 Apr 20 2016 srv

dr-xr-xr-x 13 root root 0 Feb 2 14:33 sys

drwxrwxrwt 12 root root 4096 Feb 2 19:46 tmp

drwxr-xr-x 10 root root 4096 Apr 20 2016 usr

drwxr-xr-x 14 root root 4096 Apr 20 2016 var

lrwxrwxrwx 1 root root 30 Feb 2 12:24 vmlinuz -> boot/vmlinuz-4.4.0-141-generic

lrwxrwxrwx 1 root root 30 Feb 5 2018 vmlinuz.old -> boot/vmlinuz-4.4.0-112-generic

1. Le premier caractère le type de fichier. Les 9 suivants sont les droits (1er bloc de 3 représente le propriétaire, le 2e le groupe du propriétaire et le 3e groupe de 3 les «Autre». Le nombre représente le nombre de liens vers cet élément. Les deux mots suivants représentent le propriétaire, puis son groupe. Le nombre représente sa taille, puis la date de modification et finalement le nom du fichier.

‘-a’

‘--all’

In directories, do not ignore file names that start with ‘.’.

1. Le répertoire « usr » est utilisé pour conserver des données des programmes utilisateurs. Le « dev » sert aux fichiers de périphérique et le « tmp » a stocké des fichiers temporaires.

1.2 Permissions (6 points)

1. Non

etu1@ubuntu:~$ cd ..

etu1@ubuntu:/home$ touch abc

touch: cannot touch 'abc': Permission denied

1. Oui

etu1@ubuntu:~$ touch /home/etu1/abc

etu1@ubuntu:~$



etu1@ubuntu:~$ touch /home/etu1/gloglo.txt

etu1@ubuntu:~$ ls -l /home/etu1/gloglo.txt

-rw-rw-r-- 1 etu1 etu1 0 Feb 3 10:22 /home/etu1/gloglo.txt

* 1. Non, droit de lecture seulement.
  2. Oui, droit de lecture et d’écriture.
  3. Oui, droit de lecture et d’écriture.

1. Je peux utiliser la commande suivante : chmod g-w /home/etu1/gloglo.txt

etu1@ubuntu:~$ chmod g-w /home/etu1/gloglo.txt

etu1@ubuntu:~$ ls -l /home/etu1/gloglo.txt

-rw-r--r-- 1 etu1 etu1 0 Feb 3 10:22 /home/etu1/gloglo.txt

1.3 Commandes fréquentes (5 pts)

|  |  |  |
| --- | --- | --- |
| 1 | bg | b |
| 2 | cat | e |
| 3 | cd | i |
| 4 | chmod | g |
| 5 | chown | y |
| 6 | echo | c |
| 7 | env | r |
| 8 | export | f |
| 9 | fg | s |
| 10 | find | k |
| 11 | gdb | h |
| 12 | grep | p |
| 13 | kill | u |
| 14 | less | v |
| 15 | ln -s | a |
| 16 | ls | d |
| 17 | ls -a | l |
| 18 | ps | o |
| 19 | pwd | x |
| 20 | strace | q |
| 21 | sudo | n |
| 22 | time | t |
| 23 | touch | j |
| 24 | wc | m |
| 25 | yes | w |

1.4 Flux de sortie (2 points)



etu1@ubuntu:~$ echo "111 124 564" > /home/etu1/mesprocessus.txt && ps >> /home/etu1/mesprocessus.txt

1. La commande > est «Créer ou remplacer le fichier» alors que >> est «Créer ou ajouter dans le fichier». Donc l’un va remplacer le contenu du fichier s’il existe alors que l’autre va simplement ajouter à la fin.

1.5. Opérateur de chaînage (pipe) (4 points)

1. etu1@ubuntu:~$ cat /proc/cpuinfo
2. cat /proc/cpuinfo | grep "cpu [^\n]Hz"

etu1@ubuntu:~$ cat /proc/cpuinfo | grep "cpu [^\n]Hz"

cpu MHz : 3191.398

1. cat /usr/include/math.h | grep "1/pi"

etu1@ubuntu:~$ cat /usr/include/math.h | grep "1/pi"

# define M\_1\_PI 0.31830988618379067154 /\* 1/pi \*/

# define M\_1\_PIl 0.318309886183790671537767526745028724L /\* 1/pi \*/

1. ls -A /usr/bin/ | wc -w

etu1@ubuntu:~$ ls -A /usr/bin/ | wc -w

1591

1.6 Répertoire "dev" (2 points)



etu1@ubuntu:~$ ls -l /dev

total 0

crw------- 1 root root 10, 235 Feb 3 05:14 autofs

drwxr-xr-x 2 root root 300 Feb 3 05:14 block

drwxr-xr-x 2 root root 60 Feb 3 05:14 bsg

crw------- 1 root root 10, 234 Feb 3 05:14 btrfs-control

drwxr-xr-x 3 root root 60 Feb 3 10:14 bus

drwxr-xr-x 2 root root 3400 Feb 3 05:14 char

crw------- 1 root root 5, 1 Feb 3 05:14 console

lrwxrwxrwx 1 root root 11 Feb 3 05:14 core -> /proc/kcore

crw------- 1 root root 10, 59 Feb 3 05:14 cpu\_dma\_latency

crw------- 1 root root 10, 203 Feb 3 05:14 cuse

drwxr-xr-x 4 root root 80 Feb 3 05:14 disk

drwxr-xr-x 2 root root 80 Feb 3 05:14 dri

^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v

crw-rw---- 1 root tty 7, 133 Feb 3 05:14 vcsa5

crw-rw---- 1 root tty 7, 134 Feb 3 05:14 vcsa6

crw-rw---- 1 root tty 7, 135 Feb 3 05:14 vcsa7

drwxr-xr-x 2 root root 60 Feb 3 05:14 vfio

crw------- 1 root root 10, 63 Feb 3 05:14 vga\_arbiter

crw------- 1 root root 10, 137 Feb 3 05:14 vhci

crw------- 1 root root 10, 238 Feb 3 05:14 vhost-net

crw-rw-rw- 1 root root 1, 5 Feb 3 05:14 zero

1. sda : disque connecter au SCSI

psaux : souris et clavier connecter en PS/2

bus/usb/001 : premier périphérique USB

fd0 : Lecteur de disquette

1.7 Gestion des tâches (*jobs*) (8 points)

1. Il sert à faire disparaître ce qu'on lui envoie. C’est utile pour rediriger le flux de sortie de commande que l’on ne veut pas gérer.
2. Mettre fin à l’exécution.
3. Pause l’exécution.

etu1@ubuntu:~$ yes > /dev/null &  
[1] 3448



etu1@ubuntu:~$ jobs

[1]+ Running yes > /dev/null &

1. kill -INT %2
2. kill -STOP %1
3. fg%1
4. Non, car ces jobs ne sont pas lié à ce terminal.

**2. Compilation et exécution (14 points)**

2.1 Création d'un fichier source (1 point)

# include <unistd.h>

# include <stdio.h>

int main() {

printf(" Bonjour! Le numero de l'utilisateur est %d\n", getuid());

printf(" Le numero de ce processus est %d\n", getpid());

printf(" Le numero de papa est %d\n", getppid());

printf(" - Mon IDUL est 111 124 564\n");

return 0;

}

2.2 Compilation (3 points)

1. Génère un fichier compilé « objet », mais ne lie pas les fichiers.
2. gcc num\_process.c -o num\_process

Bonjour! Le numero de l'utilisateur est 1000

Le numero de ce processus est 2323

Le numero de papa est 2268

- Mon IDUL est 111 124 564

2.3 Appels systèmes fait par un programme : strace (2 points)

1. execve("./num\_process", ["./num\_process"], [/\* 65 vars \*/])

Cet appel sert à charger dans la mémoire le code de l’application dans le processus.

1. L’appel système « write » est invoqué par les « printf »

2.4 Débogage (5 points)

1. gcc num\_process.c -g -o num\_process-debug
2. gdb ./num\_process-debug
3. break 7

(gdb) break 7

Breakpoint 1 at 0x804852e: file num\_process.c, line 7

1. run

(gdb) run

Starting program: /home/etu1/Desktop/2/num\_process-debug

Bonjour! Le numero de l'utilisateur est 1000

Le numero de ce processus est 2477

Le numero de papa est 2475

Breakpoint 1, main () at num\_process.c:7

7 printf(" - Mon IDUL est 111 124 564\n");

1. frame

(gdb) frame

#0 main () at num\_process.c:7

7 printf(" - Mon IDUL est 111 124 564\n");

* 1. Core Dump (3 points)



int main() {

int\* ptr = 0;

\*ptr = 0;

}



gdb -c core ./zut  
  
etu1@ubuntu:~/Desktop/2$ gdb -c core ./zut

GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1

^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v

Reading symbols from ./zut...(no debugging symbols found)...done.

[New LWP 2575]

warning: Unexpected size of section `.reg-xstate/2575' in core file.

Core was generated by `./zut'.

Program terminated with signal SIGSEGV, Segmentation fault.

warning: Unexpected size of section `.reg-xstate/2575' in core file.

#0 0x080483eb in main ()



(gdb) frame

d#0 0x080483eb in main ()

**3. Fonction fork (20 points)**



#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

int main()

{

char idul[11] = "111 124 564";

pid\_t childID = fork();

if (childID == 0)

{

printf("Merci papa! Je prends note, je suis le processus %d et notre createur est %s\n", getpid(), idul);

sleep(90);

exit(0);

}

else

{

printf("Fils! Voici ton numero de processus: %d. Notre createur est %s\n", childID, idul);

wait(0);

return 0;

}

}



etu1@ubuntu:~$ ps -ef | grep telperetelfils

etu1 3341 3147 0 19:22 pts/4 00:00:00 ./telperetelfils

etu1 3342 3341 0 19:22 pts/4 00:00:00 ./telperetelfils

etu1 3369 3349 0 19:23 pts/7 00:00:00 grep --color=auto telperetelfils



etu1@ubuntu:~/Desktop/3$ ./telperetelfils

Fils! Voici ton numero de processus: 3342. Notre createur est 111 124 564

Merci papa! Je prends note, je suis le processus 3342 et notre createur est 111 124 564



etu1@ubuntu:~$ ps -ef | grep telperetelfils

etu1 3341 3147 0 19:22 pts/4 00:00:00 ./telperetelfils

etu1 3342 3341 0 19:22 pts/4 00:00:00 ./telperetelfils

etu1 3369 3349 0 19:23 pts/7 00:00:00 grep --color=auto telperetelfils



#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

int main()

{

char idul[11] = "111 124 564";

pid\_t childID = fork();

if (childID == 0)

{

printf("Merci papa! Je prends note, je suis le processus %d et notre createur est %s\n", getpid(), idul);

sleep(90);

exit(0);

}

else

{

printf("Fils! Voici ton numero de processus: %d. Notre createur est %s\n", childID, idul);

wait(0);

return 0;

}

}

**4. Exercice de programmation par threads (36 points)**



#include <stdio.h>

#include <pthread.h>

#define NUMBER\_OF\_THREADS 4

typedef struct

{

int threadNum;

int n;

int debut;

int fin;

unsigned long long produit;

} Parametres;

void \*CalculerProduit(void \*data)

{

Parametres \*mesParametres = data;

printf("Je suis le thread %d qui calcul le produit de %d a %d\n",

mesParametres->threadNum,

mesParametres->debut,

mesParametres->fin);

mesParametres->produit = 1;

for (mesParametres->n = mesParametres->debut; mesParametres->n <= mesParametres->fin; mesParametres->n++)

{

mesParametres->produit \*= mesParametres->n;

}

pthread\_exit(&mesParametres->produit); //&mesParametres->produit ou NULL ??

}

void InitialiseParam(Parametres \*param, int num, int start, int end)

{

param->threadNum = num;

param->debut = start;

param->fin = end;

param->n = 0;

param->produit = 1;

}

void PrintFactorial(int factorialToCalculate)

{

printf("Calcul du factorielle de %d\n", factorialToCalculate);

pthread\_t threads[NUMBER\_OF\_THREADS];

Parametres mesParametres[NUMBER\_OF\_THREADS];

int startValue = 1;

int endValue = 1;

int step = factorialToCalculate / NUMBER\_OF\_THREADS;

for (int i = 0; i < NUMBER\_OF\_THREADS -1; i++)

{

endValue = startValue + step - 1;

InitialiseParam(&mesParametres[i], i + 1, startValue, endValue);

startValue = endValue + 1;

}

//Forcer le dernier thread a prendre les cas restants.

InitialiseParam(&mesParametres[NUMBER\_OF\_THREADS - 1], NUMBER\_OF\_THREADS, startValue, factorialToCalculate);

for(int i = 0; i < NUMBER\_OF\_THREADS; i++)

{

printf("Main(): en train de creer thread %d\n", i+1);

pthread\_create(&threads[i], NULL, CalculerProduit, (void \*)&mesParametres[i]);

}

for(int i = 0; i < NUMBER\_OF\_THREADS; i++)

{

pthread\_join(threads[i], NULL); //NULL ou value??

}

printf("Main(): Tous les threads ont termines! Je calcul la factorielle...\n");

long long fact = 1;

fact \*= mesParametres[0].produit;

fact \*= mesParametres[1].produit;

fact \*= mesParametres[2].produit;

fact \*= mesParametres[3].produit;

printf("Main(): La factorielle de %d est %llu\n", factorialToCalculate, fact);

}

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

{

const int factorialToCalculate = 15;

//for(int zzz=0; zzz<50000;zzz++)

//{

PrintFactorial(factorialToCalculate);

//}

return 0;

}

1. Pour exécuter 50000 fois le programme, cela prend 47 secondes environ.

etu1@ubuntu:~/Desktop/4$ time ./tp1₄

Calcul du factorielle de 15

Main(): en train de creer thread 1

^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v

Je suis le thread 1 qui calcul le produit de 1 a 3

Main(): Tous les threads ont termines! Je calcul la factorielle…

Main(): La factorielle de 15 est 1307674368000

real 0m46.934s

user 0m1.148s

sys 0m3.024s

Lorsque l’on redirige la sortie du programme dans /dev/null pour éviter d’être influencé par le rafraichissement de la console, nous obtenons ces résultats :

etu1@ubuntu:~/Desktop/4$ time (./tp1\_4 > /dev/null)

real 0m2.132s

user 0m0.292s

sys 0m0.720s

1. Avec 4 processeurs, le temps prend un peu plus d’une secondes pour exécuter 50000 fois la factoriel.

etu1@ubuntu:~/Desktop/4$ time ./tp1\_4

Calcul du factorielle de 15

Main(): en train de creer thread 1

Je suis le thread 1 qui calcul le produit de 1 a 3

Main(): en train de creer thread 2

Main(): en train de creer thread 3

Je suis le thread 2 qui calcul le produit de 4 a 6

Main(): en train de creer thread 4

Je suis le thread 3 qui calcul le produit de 7 a 9

Je suis le thread 4 qui calcul le produit de 10 a 15

Main(): Tous les threads ont termines! Je calcul la factorielle...

Main(): La factorielle de 15 est 1307674368000

^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v^v

real 1m2.543s

user 0m2.784s

sys 0m21.500s

Lorsque l’on redirige la sortie du programme dans /dev/null pour éviter d’être influencé par le rafraichissement de la console, nous obtenons ces résultats :

etu1@ubuntu:~/Desktop/4$ time (./tp1\_4 > /dev/null)

real 0m5.090s

user 0m0.412s

sys 0m2.696s

1. Le problème semble prendre plus de temps avec plusieurs processeurs, mais c’est probablement causé par une charge supplémentaire de travail pour gérer les flux de sorties en concurrence, alors qu’il n'y pas beaucoup de gain en temps processeur étant donné qu’il n’y a que ~3 multiplications à faire pour chaque thread. On peut aussi remarquer qu’avec plusieurs threads, l’exécution des printf est beaucoup plus « aléatoire » que lorsqu’on avait un seul processeur.