Rémi MAUBANC – I3A 2023

Rapport du TP2 de Système d'exploitation
Les Processus

I. Création de processus (*fonction fork*)

Exercice n°1

1) *(voir figure 1 en annexe.)*

2) On remarque que les numéros PID sont différents et s'incrémentent au fur et à mesure des exécutions. En effet, les numéros de processus attribué par le système d'exploitation commencent à 0 et au fur et à mesure des demandes, augmente l'ID. Lorsque l'ID maximum est atteint il repart de 0. Dans la VM de l'ESIEE, le PID maximal est de 32768 (cette limite est stockée dans le fichier '*/proc/sys/kernel/pid_max*'.

3) Lorsqu'on place un 'sleep' dans le corpus exécuté par le fils, le processus père se termine, terminant au passage le programme dans le terminal. La ligne affichée par le fils s'affiche après la fin du programme. Entre le moment où le programme se termine et le moment le fils affiche sa ligne, nous pouvons l'observer dans l'observateur.

```
etudiant@VMlinux:/tmp/OS_ESIEE/TP2$ ./a.out
PID du processus courant : 5538
PID du processus pere : 5538
etudiant@VMlinux:/tmp/OS_ESIEE/TP2$ ps -al
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 T  1001  5502  5431  0  80   0 - 14789 signal pts/1    00:00:00 vim
1 S  1001  5539     1  0  80   0 -  1127 hrtime pts/1    00:00:00 a.out
0 R  1001  5540  5431  0  80   0 -  7558 -      pts/1    00:00:00 ps
etudiant@VMlinux:/tmp/OS_ESIEE/TP2$
etudiant@VMlinux:/tmp/OS_ESIEE/TP2$ PID du processus fils : 5539

etudiant@VMlinux:/tmp/OS_ESIEE/TP2$
```

4) Lorsque l'on place un 'sleep' dans le corpus exécuté par le père, le processus fils se termine de manière invisible pour l'utilisateur. Pendant le sommeil du père, le terminal bash est bloqué. Dès que le père termine, le terminal bash est débloqué pour l'utilisateur.

Rémi MAUBANC – I3A 2023

II. Synchronisation de processus père et fils (*mécanisme wait/exit*)

Exercice n°2 : Mécanisme wait/exit

Code source ‘fork-sync.c’ : voir figure 2 en annexe.

Approfondissement :

Lorsque l’on remplace le ‘wait’ par un ‘sleep(30)’. Tant que les deux (père et fils) fonctionnent, rien ne changent par rapport au code précédent.
Cependant lorsque le fils termine son exécution, le père étant toujours dans le sleep, le fils est qualifié par le système de ‘defunct’ ou en d’autres termes de zombie. A la fin du père, il ne parvient pas à retourner le bon code de retour du fils (retourne 0 à la place).

```
etudiant@VMlinux:~/Documents/OS_ESIEE/TP2$ ps -h
 1461 pts/2    Ss+    0:00 bash
 2395 pts/3    S+     0:00 ./a.out
 2399 pts/3    S+     0:00 ./a.out
 2507 pts/1    R+     0:00 ps -h
 2790 pts/1    Ss     0:00 bash
18460 pts/3    Ss     0:00 bash
25744 pts/3    T      0:01 vim child_info.c
etudiant@VMlinux:~/Documents/OS_ESIEE/TP2$ ps -h
 1461 pts/2    Ss+    0:00 bash
 2395 pts/3    S+     0:00 ./a.out
 2399 pts/3    Z+     0:00 [a.out] <defunct>
 2790 pts/1    Ss     0:00 bash
 2955 pts/1    R+     0:00 ps -h
18460 pts/3    Ss     0:00 bash
25744 pts/3    T      0:01 vim child_info.c
```

Quand le père se termine, les deux processus disparaissent de l’observateur.
Lorsque l’on ajoute la ligne ‘signal(SIGCHLD, SIG_IGN)’, une différence est notable : le fils n’est plus considéré comme zombie mais disparaît simplement de l’observateur à la fin de son exécution.
On observe aucune différence dans les cas où le temps de sommeil du père est de 30, 20 ou 10 secondes.

Rémi MAUBANC – I3A 2023

III. Notion de groupe

Exercice n°3 : Groupes

Code source ‘fork-grp.c’ : voir figure 3 en annexe.

Quand un fils devient insensible au signal SIGINT, tous les processus fils créés à partir de ce dernier seront également insensible. En effet, quand un SIGINT est envoyé, tous les processus dans le groupe portant l’ID du père seront tués. Quand un processus fils change pour un groupe avec pour l’ID lui-même, il ne dépend plus du père. Par défaut, le groupe d’un fils est celui du père et comme ce dernier s’est mis à son nom, il dépend donc directement de ce dernier. On peut observer cette séparation dans le gestionnaire des tâches.

```
 5431 etudiant   20   0 25212  6672  4056 S  0.0  0.2  0:00.54                  ├─ bash
 7960 etudiant   20   0  4508   812   752 S  0.0  0.0  0:00.00                  │  ├─ ./a.out
 7962 etudiant   20   0  4508    76     0 S  0.0  0.0  0:00.00                  │  │  ├─ ./a.out
 7964 etudiant   20   0  4508    76     0 S  0.0  0.0  0:00.00                  │  │  │  └─ ./a.out
 7961 etudiant   20   0  4508    68     0 S  0.0  0.0  0:00.00                  │  │  └─ ./a.out
 5502 etudiant   20   0 60580 11132  6308 T  0.0  0.3  0:04.65                  │  └─ vim exo1.c
```

IV. Collecte d’informations sur le système

Exercice n°4 : Information fournies par *sysinfo()*

Les informations affichées sont les suivantes :
- Le temps de fonctionnement depuis le dernier démarrage en secondes
- La charge moyenne sur la dernier minute
- La charge moyenne sur les 5 dernières minutes
- La charge moyenne sur les 15 dernières minutes
- La capacité totale de la mémoire vive en octet
- L’espace libre de la mémoire vive en octet
- L’espace partagé de la mémoire vive en octet
- La mémoire utilisé par les buffers en octet
- L’espace total du swap en octet
- L’espace disponible du swap en octet
- Le nombre de processus en fonctionnement
- La structure du coussin de 64 octets

Code source : voir figure 4 en annexe.

```
Uptime : 2023 secondes
Load average (last 1 min) : 0.014
Load average (last 5 min) : 0.045
Load average (last 15 min) : 0.034
Total RAM : 4136103936 bytes
Total free RAM : 2030702592 bytes
Total shared RAM : 15261696 bytes
Memory user by buffers : 126439424 bytes
Total swap space size : 2147479552 bytes
Swap space still available : 2147479552 bytes
Number of current processes : 310
Pads structure to 64 bytes : 00 00 00 00 FFFFFF80 0D 78 FFFFFFA6 FFFFFFFD 7F 00 00 00 FF
FFFFE2 6B FFFFFF8B FFFFFFE6 7A FFFFFF98 FFFFFFE5 FFFFFF80 79
etudiant@VMlinux:~/Documents/OS_ESIEE/TP2$
```

Rémi MAUBANC – I3A 2023

/!\ : Les charges moyennes renvoyées par la fonction « sysinfo » sont des entiers, or la charge moyenne s'exprime sous forme d'un nombre à virgule. Pour obtenir la valeur correcte, il faut diviser la valeur obtenue par 65536.

Exercice n°5 : Dépendances et rôles des méthodes *getpid*, *getppid*, *getuid*, *geteuid*, *getgid*, *getegid*

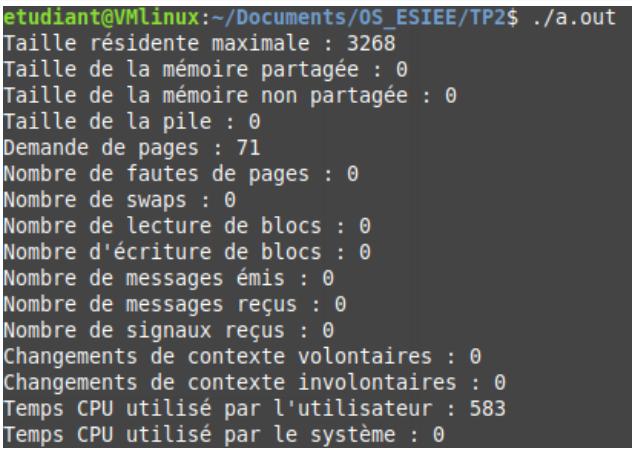
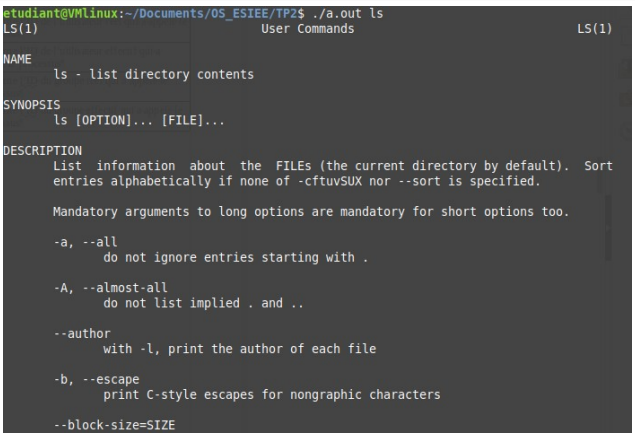
Nom de la fonction	Dépendances	Roles
getpid()	- <sys/systypes.h> - <unistd.h>	Retourne l'ID du processus (PID) qui a appelé la fonction.
getppid()	- <sys/systypes.h> - <unistd.h>	Retourne l'ID du processus père du processus qui a appelé la fonction
getuid()	- <sys/systypes.h> - <unistd.h>	Retourne l'ID de l'utilisateur réel qui a appelé le processus
geteuid()	- <sys/systypes.h> - <unistd.h>	Retourne l'ID de l'utilisateur effectif qui a appelé le processus
getgid()	- <sys/systypes.h> - <unistd.h>	Retourne l'ID du groupe réel qui a appelé le processus
getegid()	- <sys/systypes.h> - <unistd.h>	Retourne l'ID du groupe effectif qui a appelé le processus

Exercice n°6 : Information sur le processus courant

Code source : voir figure 5 en annexe.

Exercice n°7 :

Code source : voir figure 6 en annexe.
Résultats obtenus :

Sans argument	Avec un argument
	

Annexes

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


int main(void)
{
    pid_t self = getpid();
    printf("PID du processus courant : %d\n", self);

    pid_t pid;
    if ((pid = fork()) == 0)
    {
        printf("PID du processus fils : %d\n", getpid());
    } else {
        printf("PID du processus pere : %d\n", getpid());
    }

    return 0;
}
```

Figure 1 : Code source de l'exercice n°1

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

int main(void)
{
    pid_t pid, wpid;
    int status;

    pid = fork();

    if (pid == 0)
    {
        printf("Je m'endors\n");
        sleep(10);
        printf("Mon PID (fils) est %d\n", getpid());
        printf("Je me reveille, mon code de sortie est : %d\n", 42);
        exit(42);
    } else {
        printf("Mon PID (pere) est %d\n", getpid());
        // signal(SIGCHLD, SIG_IGN);
        // sleep(30);
        wpid = wait(&status);
        if (wpid != -1) {
            printf("mon fils est mort ; son code de sortie est : %d\n",
                    WEXITSTATUS(status));
        }
    }
    return 0;
}
```

Figure 2 : Code source de l'exercice n°2 (fork-sync.c)

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


int main(void) {
    pid_t child_1, child_2;
    child_1 = fork();

    if (child_1 == 0) {
        printf("Processus %d, groupe %d\n", getpid(), getpgid(getpid()));
        for (int i = 0; i < 70; ++i) {
            printf("%d survis\n", getpid());
            sleep(1);
        }
    } else {
        printf("Processus %d, groupe %d\n", getpid(), getpgid(getpid()));
        child_2 = fork();
        if (child_2 != 0) {
            for (int i = 0; i < 70; ++i) {
                printf("%d survis\n", getpid());
                sleep(1);
            }
        } else {
            printf("Processus %d, groupe %d\n", getpid(), getpgid(getpid()));
            printf("retour changement groupe : %d\n", setpgid(getpid(), getpid()));
            printf("Processus %d, groupe %d\n", getpid(), getpgid(getpid()));
            sleep(5);
            pid_t insensitive = fork();
            printf("Processus %d, groupe %d\n", getpid(), getpgid(getpid()));
            if (insensitive == 0) {
                for (int i = 0; i < 70; ++i) {
                    printf("%d survis\n", getpid());
                    sleep(1);
                }
            } else {
                for (int i = 0; i < 70; ++i) {
                    printf("%d survis\n", getpid());
                    sleep(1);
                }
            }
        }
    }
}

```

Figure 3 : Code source de l’exercice n°3 (fork-grp.c)

```c
#include <stdio.h>
#include <sys/sysinfo.h>


int main(void) {
    struct sysinfo my_sys;

    if (sysinfo(&my_sys) < 0)
        return 1;

    printf("Uptime : %ld secondes\n", my_sys.uptime);
    printf("Load average (last 1 min) : %.3f\n", (my_sys.loads[0] / 65536.0));
    printf("Load average (last 5 min) : %.3f\n", (my_sys.loads[1] / 65536.0));
    printf("Load average (last 15 min) : %.3f\n", (my_sys.loads[2] / 65536.0));
    printf("Total RAM : %ld bytes\n", my_sys.totalram);
    printf("Total free RAM : %ld bytes\n", my_sys.freeram);
    printf("Total shared RAM : %ld bytes\n", my_sys.sharedram);
    printf("Memory user by buffers : %ld bytes\n", my_sys.bufferram);
    printf("Total swap space size : %ld bytes\n", my_sys.totalswap);
    printf("Swap space still available : %ld bytes\n", my_sys.freeswap);
    printf("Number of current processes : %d\n", my_sys.procs);
    printf("Pads structure to 64 bytes :");
    for (int i = 0; i < 22; ++i)
        printf(" %02X", my_sys._f[i]);
    printf("\n");
}
```

Figure 4 : Code source de l'exercice n°4 (sys-info.c)

```c
#include <stdio.h>
#include <sys/time.h>
#include <sys/resource.h>

int main(void) {
    struct rusage my_sys, my_end;
    if (getrusage(RUSAGE_SELF, &my_sys) < 0)
        return 1;

    printf("Taille résidente maximale : %ld octets\n", my_sys.ru_maxrss);
    printf("Taille de la mémoire partagée : %ld octets\n", my_sys.ru_ixrss);
    printf("Taille de la mémoire non partagée : %ld octets\n", my_sys.ru_idrss);
    printf("Taille de la pile : %ld octets\n", my_sys.ru_isrss);
    printf("Demande de pages : %ld\n", my_sys.ru_minflt);
    printf("Nombre de fautes de pages : %ld\n", my_sys.ru_majflt);
    printf("Nombre de swaps : %ld\n", my_sys.ru_nswap);
    printf("Nombre de lecture de blocs : %ld\n", my_sys.ru_inblock);
    printf("Nombre d'écriture de blocs : %ld\n", my_sys.ru_oublock);
    printf("Nombre de messages émis : %ld\n", my_sys.ru_msgsnd);
    printf("Nombre de messages reçus : %ld\n", my_sys.ru_msgrcv);
    printf("Nombre de signaux reçus : %ld\n", my_sys.ru_nsignals);
    printf("Changements de contexte volontaires : %ld\n", my_sys.ru_nvcsw);
    printf("Changements de contexte involontaires : %ld\n", my_sys.ru_nivcsw);

    if (getrusage(RUSAGE_SELF, &my_end) < 0)
        return 1;

    printf("Temps CPU utilisé par l'utilisateur : %ld microsecondes\n",
           my_end.ru_utime.tv_usec - my_sys.ru_utime.tv_usec);
    printf("Temps CPU utilisé par le système : %ld microsecondes\n",
           my_end.ru_stime.tv_usec - my_sys.ru_stime.tv_usec);
}
```

Figure 5 : Code source de l'exercice n°5 (getrusage)

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


void print_info(struct rusage my_sys) {
    printf("Taille résidente maximale : %ld\n", my_sys.ru_maxrss);
    printf("Taille de la mémoire partagée : %ld\n", my_sys.ru_ixrss);
    printf("Taille de la mémoire non partagée : %ld\n", my_sys.ru_idrss);
    printf("Taille de la pile : %ld\n", my_sys.ru_isrss);
    printf("Demande de pages : %ld\n", my_sys.ru_minflt);
    printf("Nombre de fautes de pages : %ld\n", my_sys.ru_majflt);
    printf("Nombre de swaps : %ld\n", my_sys.ru_nswap);
    printf("Nombre de lecture de blocs : %ld\n", my_sys.ru_inblock);
    printf("Nombre d'écriture de blocs : %ld\n", my_sys.ru_oublock);
    printf("Nombre de messages émis : %ld\n", my_sys.ru_msgsnd);
    printf("Nombre de messages reçus : %ld\n", my_sys.ru_msgrcv);
    printf("Nombre de signaux reçus : %ld\n", my_sys.ru_nsignals);
    printf("Changements de contexte volontaires : %ld\n", my_sys.ru_nvcsw);
    printf("Changements de contexte involontaires : %ld\n", my_sys.ru_nivcsw);
    printf("Temps CPU utilisé par l'utilisateur : %ld\n",
           my_sys.ru_utime.tv_usec);
    printf("Temps CPU utilisé par le système : %ld\n",
           my_sys.ru_stime.tv_usec);
}


int main(int argc, char **argv) {
    struct rusage my_sys;
    pid_t wpid;
    int status;

    if (argc == 1) {
        if (getrusage(RUSAGE_SELF, &my_sys) < 0)
            return 1;
        print_info(my_sys); // Affiche les statistiques concernant le processus
        return 0;
    }
    if (argc != 2)
        return 1;

    FILE *fp;
    char buff[16384];

    char cmd[1024] = {0};
    strcpy(cmd, "/usr/bin/man ");
    strcpy(&(cmd[13]), argv[1]);

    fp = popen(cmd, "r");
    if (fp == NULL)
        exit(1);
    while (fgets(buff, sizeof(buff), fp) != NULL) {
        printf("%s", buff);
    }
    pclose(fp);
    return 0;
}
```

Figure 6 : Code source de l’exercice n°7

Rémi MAUBANC – I3A 2023

Résultats obtenus :

```
etudiant@VMlinux:~/Documents/OS_ESIEE/TP2$ ./a.out
Taille résidente maximale : 2544
Taille de la mémoire partagée : 0
Taille de la mémoire non partagée : 0
Taille de la pile : 0
Demande de pages : 69
Nombre de fautes de pages : 0
Nombre de swaps : 0
Nombre de lecture de blocs : 0
Nombre d'écriture de blocs : 0
Nombre de messages émis : 0
Nombre de messages reçus : 0
Nombre de signaux reçus : 0
Changements de contexte volontaires : 0
Changements de contexte involontaires : 0
Temps CPU utilisé par l'utilisateur : 78
Temps CPU utilisé par le système : 0
```