

ATW 01 Processus et threads

1. Le syscall fork()

1.1 Premier contact

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

int main(){

    pid_t pid;
    pid = fork();

    if (pid == 0)
        printf("Child\n");
    else
        printf("Parent\n");

    return 0;
}
```

Ce que le programme affiche :

```
axel@axel-G3-3500:~/Github/ATI01-Algo_avancee/TP/TP07_processus_et_threads$
gcc -o 1.1 1.1.c
axel@axel-G3-3500:~/Github/ATI01-Algo_avancee/TP/TP07_processus_et_threads$
./1.1
Parent
Child
```

La fonction fork permet de créer un nouveau processus. Autrement dit, l'appel à fork crée un processus copie conforme du processus appelant. Avec le fork cela fait un clone du programme en cours

le parent va renvoyer le pid du fils le fils va renvoyer le 0

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

int main(){

    pid_t pid;
```

```
pid = fork();

if (pid == 0){
    printf("Child\n");
    printf ("PID du père: %ld\n", getpid ());
    printf ("PID du fils : %ld\n", getppid ());
}
else {
    printf("Parent\n");
    printf ("PID du fils: %ld\n", getpid ());
    printf ("PID du père : %ld\n", getppid ());
}
return 0;
}
```

Ce que donne le programme lors donne :

```
axel@axel-G3-3500:~/Github/ATI01-Algo_avancee/TP/TP07_processus_et_threads$
./1.1
Parent
PID du fils: 67020
PID du père : 46349
Child
PID du père: 67021
PID du fils : 67020
```

1.2 Débug de programmes utilisant fork

Pour compiler le programme et utiliser gdb par la suite :

```
axel@axel-G3-3500:~/Github/ATI01-Algo_avancee/TP/TP07_processus_et_threads$
gcc -g -o 1.1 1.1.c
axel@axel-G3-3500:~/Github/ATI01-Algo_avancee/TP/TP07_processus_et_threads$
gdb 1.1
```

```
(gdb) break main
Breakpoint 1 at 0x11d5: file 1.1.c, line 8.
(gdb) run
Starting program: /home/axel/Github/ATI01-
Algo_avancee/TP/TP07_processus_et_threads/1.1

Breakpoint 1, main () at 1.1.c:8
8          pid = fork();
(gdb) show follow-fork-mode
Debugger response to a program call of fork or vfork is "parent".
```

- **follow-fork-mode** : Affiche la réponse actuelle du débogueur à un appel fork.

```
(gdb) break main
Note: breakpoints 1 and 3 also set at pc 0x555555551d5.
Breakpoint 4 at 0x555555551d5: file 1.1.c, line 8.
(gdb) run
Starting program: /home/axel/Github/ATI01-
Algo_avancee/TP/TP07_processus_et_threads/1.1

Breakpoint 1, main () at 1.1.c:8
8         pid = fork();
(gdb) show detach-on-fork
Whether gdb will detach the child of a fork is on.
```

- **detach-on-fork** : Indique si detach-on-fork le mode est activé ou désactivé.

Quand j'exécute le processus fils j'obtiens un pid de 0 sinon quand j'exécute le processus père j'obtiens un nombre >0.

1.3 Fonctionnement des processus

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

int main(){

    pid_t pid;
    pid = fork();
    int i = 0;

    if (pid == 0){
        while (i != 20000){
            printf("Child\n");
            //printf ("PID du père: %ld\n", getpid ());
            //printf ("PID du fils : %ld\n", getppid ());
            i++;
        };
    }

    else {
        while (i != 20000){
            printf("Parent\n");
            //printf ("PID du fils: %ld\n", getpid ());
            //printf ("PID du père : %ld\n", getppid ());
            i++;
        };
    }
}
```

```

        };

    }
    return 0;
};

```

Non les pid ne son pas régulier et prévisible, comme le montre cet extrait de résultat du programme précédent : j'ai fais une boucle qui n'est pas infinie mais qui va sur un grand nombre et la je peux voir que ce n'est pas prévisible. (je suis sur mon pc perso)

```

Child
Parent
Child
Parent
Child
Parent
Child
Child
Child
Child
Child
Child

```

Voici le programme qui tourne a l'infinie comme demandé, mais ce n'est pas avec celui la qui ma permis de répondre a la question

```

#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>

int main(){

    pid_t pid;
    pid = fork();
    int i=0;

    if (pid == 0){
        while (pid ==0){
            printf("Child\n");
            /*printf ("PID du père: %ld\n", getpid ());
            printf ("PID du fils : %ld\n", getppid ());*/
            i++;
            printf ("%d\n", i);
        };

    }

    else {

```

```

        while (pid !=0){
            printf("Parent\n");
            //printf ("PID du fils: %ld\n", getpid ());
            //printf ("PID du père : %ld\n", getppid ());
            i++;
            printf ("%d\n", i);
        };
    }
    return 0;
};

```

Quand je fais la commande ps j'obtient ca :

```

axel@axel-G3-3500:~$ ps -auf
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
axel          141101  0.3  0.0  13428  5080 pts/1    Ss   14:26   0:00 bash
axel          141161  0.0  0.0  14536  3596 pts/1    R+   14:26   0:00 \_ ps -
auf
axel           88455  0.0  0.0  13560  5184 pts/0    Ss   11:44   0:00
/usr/bin/bash
axel          141022 19.5  0.0   2496   516 pts/0    S+   14:26   0:01 \_
./1.3b
axel          141023 19.1  0.0   2496    76 pts/0    S+   14:26   0:01 \_
./1.3
axel           2826  0.0  0.0 166740  6636 tty2      Ssl+ 08:25   0:00
/usr/lib/gdm3
axel           2828  2.3  1.2 26269824 194116 tty2      Sl+   08:25   8:28 \_
/usr/lib/
axel           2908  0.0  0.0 190940 13956 tty2      Sl+   08:25   0:00 \_
/usr/libe
gdm            1868  0.0  0.0 166740  6668 tty1      Ssl+ 08:25   0:00
/usr/lib/gdm3
gdm            1876  0.0  0.6 25745368 110876 tty1      Sl+   08:25   0:01 \_
/usr/lib/
gdm            2237  0.0  0.0   5296  1096 tty1      S+   08:25   0:00 \_
dbus-run-
gdm            2238  0.0  0.0   7576  4604 tty1      S+   08:25   0:00 \_
dbus-
gdm            2239  0.0  0.0 486856 15464 tty1      Sl+   08:25   0:00 \_
/usr/
gdm            2271  0.0  2.2 4481232 369340 tty1      Sl+   08:25   0:09
\_ /
gdm            2315  0.0  0.0  313764  8392 tty1      Sl   08:25   0:00
| i
gdm            2318  0.0  0.0  239368  7108 tty1      Sl   08:25   0:00
| /
gdm            2422  0.0  0.0 165536  7164 tty1      Sl   08:25   0:00
| /
gdm            2366  0.0  0.0 467836 10892 tty1      Sl+  08:25   0:10
\_ /
gdm            2368  0.0  0.1 264740 17300 tty1      Sl+  08:25   0:00
\_ /

```

```

gdm                2369  0.0  0.1 425084 20984 tty1      Sl+  08:25  0:00
\_ /
gdm                2370  0.0  0.1 338968 17960 tty1      Sl+  08:25  0:00
\_ /
gdm                2371  0.0  0.0 251168 11400 tty1      Sl+  08:25  0:00
\_ /
gdm                2372  0.0  0.0 459880  6156 tty1      Sl+  08:25  0:00
\_ /
gdm                2373  0.0  0.0 318096 10136 tty1      Sl+  08:25  0:00
\_ /
gdm                2374  0.0  0.1 376772 16312 tty1      Sl+  08:25  0:00
\_ /
gdm                2378  0.0  0.1 681812 20852 tty1      Sl+  08:25  0:00
\_ /
gdm                2385  0.0  0.0 238288  5880 tty1      Sl+  08:25  0:00
\_ /
gdm                2390  0.0  0.0 322284  9444 tty1      Sl+  08:25  0:00
\_ /
gdm                2391  0.0  0.0 312596  6788 tty1      Sl+  08:25  0:00
\_ /
gdm                2393  0.0  0.0 314652  7892 tty1      Sl+  08:25  0:00
\_ /
gdm                2397  0.0  0.1 689352 29680 tty1      Sl+  08:25  0:00
\_ /
gdm                2543  0.0  0.0 344964 15180 tty1      Sl+  08:25  0:00
/usr/libexec/
gdm                2350  0.0  0.1 3007012 26888 tty1      Sl+  08:25  0:00
/usr/bin/gjs
gdm                2338  0.0  0.0 238460  6524 tty1      Sl+  08:25  0:00
/usr/libexec/
gdm                2334  0.0  0.0 162836  7656 tty1      Sl+  08:25  0:00
/usr/libexec/
gdm                2323  0.0  0.0 239348  7252 tty1      Sl+  08:25  0:00
/usr/libexec/
gdm                2321  0.0  0.1 191304 17056 tty1      Sl  08:25  0:00
/usr/libexec/
gdm                2243  0.0  0.0 305520  6688 tty1      Sl+  08:25  0:00
/usr/libexec/
gdm                2248  0.0  0.0  7292  4056 tty1      S+   08:25  0:00  \_
/usr/bin/
axel@axel-G3-3500:~$

```

Nous pouvons constater que les pid sont bien différents. et pour le ppid c'est la même chose.

et avec la commande top ou htop c'est la même chose.

ce n'est prévisible car ne nombreuses taches sont effectuer en même temps sur le pc et ont la même priorité

1.4 Mémoire virtuelle

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

int main(){

    pid_t pid;
    pid = fork();
    int i = 0;
    int a = 0;

    if (pid == 0){
        while (i != 20){
            printf("Child\n");
            printf ("PID du père: %ld\n", getpid ());
            printf ("PID du fils : %ld\n", getppid ());
            a=a+1;
            i++;
        };
    }

    else {
        while (i != 20){
            printf("Parent\n");
            printf ("PID du fils: %ld\n", getpid ());
            printf ("PID du père : %ld\n", getppid ());
            i++;
        };
    }
    return 0;
};
```

les pid ne changent pas

```
PID du fils: 146229
PID du fils : 146229
PID du père : 88455
Child
Parent
PID du père: 146230
PID du fils: 146229
PID du fils : 146229
PID du père : 88455
Child
Parent
PID du père: 146230
PID du fils: 146229
```

```
PID du fils : 146229
PID du père : 88455
```

2. Les threads Posix (pthreads)

2.1 Premier contact :

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void *print_message_function(void *ptr);
int main(){
    pthread_t thread1, thread2;
    char *message1 = "Thread 1";
    char *message2 = "Thread 2";
    int iret1, iret2;
    /* Create independent threads each of which will execute function */
    iret1 = pthread_create(&thread1, NULL, print_message_function,
(void*)message1);
    iret2 = pthread_create(&thread2, NULL, print_message_function,
(void*)message2);

    /* Wait till thread are complete before main continues. Unless we */
    /* Wait we run the risk of executing an exit which will terminate */
    /* the process and all threads before the threads have completed. */

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    printf("Thread 1 returns: %d\n", iret1);
    printf("Thread 2 returns: %d\n", iret2);
    exit(0);
    return 0;
}
void *print_message_function(void *ptr)
{
    char *message;
    message = (char *)ptr;
    printf("%s \n", message);
    return NULL ;
}
```

résultat obtenus :

```
axel@axel-G3-3500:~/Github/ATI01-Algo_avancee/TP/TP07_processus_et_threads$
gcc -Wall 2.1.c -o 2.1 -g -lm -lpthread
axel@axel-G3-3500:~/Github/ATI01-Algo_avancee/TP/TP07_processus_et_threads$
./2.1
Thread 1
```



```
Thread 2
Thread 1 returns: 0
Thread 2 returns: 0
```

après modification du programme

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void *print_message_function(void *ptr);
int main(){
    pthread_t thread1, thread2;
    char *message1 = "Thread 1";
    char *message2 = "Thread 2";
    int iret1, iret2;
    /* Create independent threads each of which will execute function */
    iret1 = pthread_create(&thread1, NULL, print_message_function,
(void*)message1);
    iret2 = pthread_create(&thread2, NULL, print_message_function,
(void*)message2);

    /* Wait till thread are complete before main continues. Unless we */
    /* Wait we run the risk of executing an exit which will terminate */
    /* the process and all threads before the threads have completed. */

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    printf("Thread 1 returns: %d\n", iret1);
    printf ("PID 1 : %ld\n", getpid ());
    printf ("PPID 1 : %ld\n", getppid ());
    printf("Thread 2 returns: %d\n", iret2);
    printf ("PID 2 : %ld\n", getpid ());
    printf ("PPID 2 : %ld\n", getppid ());
    exit(0);
    return 0;
}
void *print_message_function(void *ptr)
{
    char *message;
    message = (char *)ptr;
    printf("%s \n", message);
    return NULL ;
}
```

résultat :

```
axel@axel-G3-3500:~/Github/ATI01-Algo_avancee/TP/TP07_processus_et_threads$
./2.1
```

```
Thread 1
Thread 2
Thread 1 returns: 0
PID 1 : 68282
PPID 1 : 68137
Thread 2 returns: 0
PID 2 : 68282
PPID 2 : 68137
```

ce programme permet de faire exécuter 2 threads en cours d'exécution de sorte qu'il sont indépendant

3. On va voir si vous avez compris

```
#include <pthread.h>
#include <stdio.h>
#define NTHREADS 12
#define ARRAY_SIZE 10000000

void *Hello(void *threadid)
{
    double A[ARRAY_SIZE];
    int i;
    sleep(3);
    for (i = 0; i < ARRAY_SIZE; i++)
    {
        A[i] = i;
    }
    printf("%d: Hello World! %f\n", threadid, A[ARRAY_SIZE-1]);
    pthread_exit(NULL);
}

int main(){
    pthread_t threads[NTHREADS];
    int rc, t;
    for (t = 0; t < NTHREADS; t++)
    {
        rc = pthread_create(&threads[t], NULL, Hello, (void *)t);
        if (rc){
            printf("ERROR; return code from pthread_create() is %d\n", rc);
            exit(-1);
        }
    }
    printf("Created %d threads\n", t);
    pthread_exit(NULL);
}
```