

La primitive exec

- La famille de primitives exec permet de créer un processus pour exécuter un programme déterminé (qui a auparavant été placé dans un fichier, sous forme binaire exécutable).
- La famille de primitives exec provoque un recouvrement de la mémoire du processus appelant par le nouveau fichier exécutable.
- **La classe de fonctions exec**
Ensemble de fonctions permettant d'exécuter une commande
execlp, execvp, execve, execl, etc.

Hiérarchie de processus, recouvrement

un processus (processus fils) est toujours créé par un autre processus (processus père):

- fork: création d'une copie du processus père
- exec: recouvrement par le processus fils

exec

```
execvp(char *filename, char *argv[]);
```

- On utilise en particulier `execvp` pour exécuter un programme en lui passant un tableau d'arguments. Le paramètre `filename` pointe vers le nom (absolu ou relatif) du fichier exécutable, `argv` vers le tableau contenant les arguments (terminé par `NULL`) qui sera passé à la fonction `main` du programme lancé.

Par convention, le paramètre `argv[0]` contient le nom du fichier exécutable, les arguments suivants étant les paramètres successifs de la commande.

- Quand on exécute un fichier exécutable on peut :
 - *le chercher*
 - *par rapport au répertoire de travail*
 - *p : dans les répertoires de la variable d'environnement `PATH`*
 - *passer les arguments en tant que*
 - *tableau de paramètres*
 - *l : liste explicite de paramètres*
 - *utiliser*
 - l'environnement courant,*
 - e : un nouveau environnement.*

Appel exec

- Si l'on désire exécuter du code à l'intérieur d'un processus, on utilisera un appel de type « exec » : `execl`, `execvp`, `execle`, `execv` ou `execvp`.
- À partir d'un programme en C, il est possible d'exécuter des processus de plusieurs manières avec soit l'appel « `system` » soit les appels « exec »
- `exec` est une famille de fonction permettant de remplacer l'image du processus courant: l'ensemble de l'espace mémoire est effacé et remplacé par l'image du processus exécuté.
- Les paramètres peuvent être passés sous la forme d'une liste de paramètres (avec les fonctions `execl*`), ou sous la forme d'un tableau de chaînes de caractères (avec les fonctions `execv*`).

La primitive system

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main()
4 {
5     printf("Resultat de la commande ps -l :\n");
6     system("ps -l");
7     printf("Fin\n");
8     return 0;
9 }
```

```
bahaj@bahaj-virtual-machine: ~$ micro exec11.c
bahaj@bahaj-virtual-machine:~$ gcc -o exec11 exec11.c
bahaj@bahaj-virtual-machine:~$ ./exec11
Resultat de la commande ps -l :
F S  UID      PID     PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000    115174   115164  0  80   0  -   4810 do_wai pts/0        00:00:00 bash
0 S  1000    116430   115174  0  80   0  -    622 do_wai pts/0        00:00:00 exec11
0 S  1000    116431   116430  0  80   0  -    652 do_wai pts/0        00:00:00 sh
4 R  1000    116432   116431  0  80   0  -   5030 -      pts/0        00:00:00 ps
Fin
bahaj@bahaj-virtual-machine:~$
```

exec11.c + (9,3) | ft:c

```
bahaj@bahaj-virtual-machine: ~  
1 #include <stdio.h>  
2 #include <unistd.h>  
3 int main()  
4 {  
5     printf("Resultat de la commande ps -l :\n");  
6     execvp("ps", "ps", "-l", NULL);  
7     printf("Fin\n");  
8 }
```

```
bahaj@bahaj-virtual-machine: ~  
bahaj@bahaj-virtual-machine:~$ micro exec12.c  
bahaj@bahaj-virtual-machine:~$ gcc -o exec12 exec12.c  
bahaj@bahaj-virtual-machine:~$ ./exec12  
Resultat de la commande ps -l :  
F S  UID      PID     PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD  
0 S  1000    115174   115164  0  80   0 -  4810 do_wai pts/0      00:00:00 bash  
1 S  1000    116563    1458  0  80   0 -  1753 poll_s pts/0      00:00:00 xclip  
4 R  1000    116614   115174  0  80   0 -  5030 -      pts/0      00:00:00 ps  
bahaj@bahaj-virtual-machine:~$
```

exec, avec tableaux

- `int execl(const char * ref , const char * argv[]);`
ref : le nom l'exécutable sur le disque, argv[] : le tableau des paramètres
- `int execlp(const char * ref , const char * argv[]);`
On cherche l'exécutable dans le PATH
- `int execlve(const char * ref , const char * argv[] , const char * arge[]);`
arge[] : tableau contenant l'environnement

```
bahaj@bahaj-virtual-machine: ~  
1 #include<stdio.h>  
2 #include<unistd.h>  
3 int main()  
4 {  
5     printf("Je suis dans execv\n");  
6     printf("PID de execv.c est %d\n",getpid());  
7     char *args[]={"/hello", NULL};  
8     execv(args[0],args);  
9     printf ("retour à execv.c"); // 9a ne sera  
10    return 0;  
11 }
```

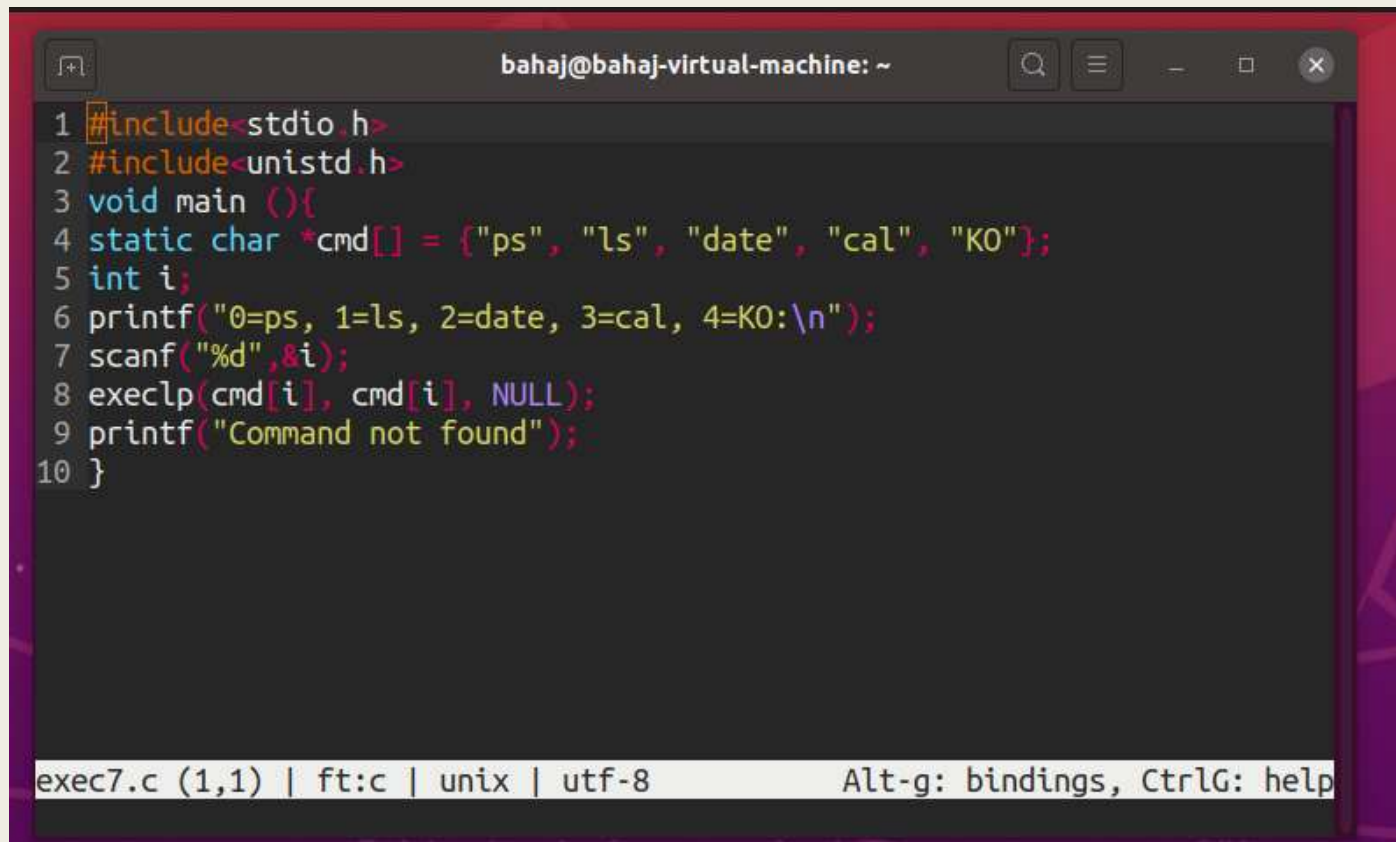
execv0.c (1,1) | ft:c | unix | utf-8Alt-g: bi

```
bahaj@bahaj-virtual-machine:~$ micro execv0.c  
bahaj@bahaj-virtual-machine:~$ micro hello.c  
bahaj@bahaj-virtual-machine:~$ gcc -o hello hello.c  
bahaj@bahaj-virtual-machine:~$ gcc -o execv0 execv0.c  
bahaj@bahaj-virtual-machine:~$ ./execv0  
Je suis dans execv  
PID de execv.c est 154073  
je suis dans hello.c  
PID de hello.c est 154073  
bahaj@bahaj-virtual-machine:~$
```

```
bahaj@bahaj-virtual-machine: ~  
1 #include<stdio.h>  
2 #include<unistd.h>  
3 int main()  
4 {  
5     printf("je suis dans hello.c\n");  
6     printf("PID de hello.c est %d\n",getpid());  
7     return 0;  
8 }  
9 }
```

hello.c (1,1) | ft:c | unix | utf-8 Alt-g: bindings, CtrlG: help

Example



```
1 #include<stdio.h>
2 #include<unistd.h>
3 void main (){
4     static char *cmd[] = {"ps", "ls", "date", "cal", "KO"};
5     int i;
6     printf("0=ps, 1=ls, 2=date, 3=cal, 4=KO:\n");
7     scanf("%d",&i);
8     execlp(cmd[i], cmd[i], NULL);
9     printf("Command not found");
10 }
```

exec7.c (1,1) | ft:c | unix | utf-8 Alt-g: bindings, CtrlG: help


```

bahaj@bahaj-virtual-machine: ~
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 int main(int argc, char**argv) {
6     printf("I am process %d. My PPID: %d\n", getpid(), getppid());
7
8     pid_t ret_val = fork();
9     if(ret_val == 0) {
10         printf("I'm the child process. PID=%d, PPID=%d\n", getpid(), getppid());
11         execlp("ps", "ps", "-l", NULL);
12         printf("This is printed only if execlp fails\n");
13         abort();
14     } else if (ret_val > 0) {
15         printf("I'm the parent process. PID=%d, PPID=%d\n", getpid(), getppid());
16         sleep(1);
17     }
18
19     return EXIT_SUCCESS;
20 }
21

```

```
exec5.c (1,1) | ft:c | unix
```

```

bahaj@bahaj-virtual-machine: ~
bahaj@bahaj-virtual-machine:~$ micro exec5.c
bahaj@bahaj-virtual-machine:~$ gcc -o exec5 exec5.c
bahaj@bahaj-virtual-machine:~$ ./exec5
I am process 118951. My PPID: 115174
I'm the parent process. PID=118951, PPID=115174
I'm the child process. PID=118952, PPID=118951
F S  UID      PID      PPID  C PRI  NI ADDR SZ WCHAN  TTY      TIME CMD
0 S  1000    115174    115164  0  80   0  -   4842 do_wai pts/0    00:00:00 bash
0 S  1000    118951    115174  0  80   0  -    622 hrtim pts/0    00:00:00 exec5
4 R  1000    118952    118951  0  80   0  -   5030 -      pts/0    00:00:00 ps
bahaj@bahaj-virtual-machine:~$

```

TP

- Ecrire un programme qui crée 2 processus, l'un faisant la commande ls, l'autre ps -elf. Le père devra attendre la fin de ses deux fils et afficher quel a été le premier processus à terminer.

```

bahaj@bahaj-virtual-machine: ~
1 #include <unistd.h> /* necessaire pour les fonctions exec */
2 #include <sys/types.h> /* necessaire pour la fonction fork */
3 #include <unistd.h> /* necessaire pour la fonction fork */
4 #include <stdio.h> /* necessaire pour la fonction perror */
5 #include <wait.h>
6 int main(int argc, char * argv[]) {
7     pid_t pid1, pid2, pid_premier;
8     int status;
9     switch(pid1=fork()) {
10     case -1: perror("fork error");
11     break;
12     case 0: execlp("ls", "ls", (char *) 0);
13     break;
14     default: switch(pid2=fork()) {
15     case -1: perror("fork error");
16     break;
17     case 0: execlp("ps", "ps", "-elf", (char *) 0);
18     break;
19     default: break;
20     }
21     break;
22     }
23     pid_premier = wait(&status);
24     wait(&status);
25     if (pid_premier==pid1)
26     {
27     printf("Premier processus a finir : %d\n", pid1);
28     }
29     else
30     {
31     printf("Premier processus a finir : %d\n", pid2);
32     }
33     return 0;
34 }

```

0 S bahaj 151307 1458 3 99 - - 322624 poll_s 11:51 ?
0 S bahaj 151338 150429 0 80 - - 589 do_wai 11:52 pts/0
4 R bahaj 151340 151338 0 80 - - 5050 - 11:52 pts/0
Premier processus a finir : 151339
bahaj@bahaj-virtual-machine:~\$

wait1.c (1,1) | ft:c | unix | utf-8

Alt-g: bindings, Ctrl

TP: Simultanéité vs. séquentialité

- 1) Ecrire un programme C équivalent à la commande shell suivante :
`who & ps & ls -l`
- 2) Ecrire un programme C équivalent à la commande shell suivante :
`who ; ps ; ls -l`
- NB. `who` est une commande UNIX permettant d'afficher des informations concernant les utilisateurs qui sont connectés.
- "&" : lancer un processus en arrière-plan

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <unistd.h>
5 int main(void)
6 { pid_t pid;
7   if ((pid = fork()) == -1)
8   { perror("fork"); exit(1);
9   }
10  if (pid == 0)
11  { execlp("who", "who", NULL);
12    perror("execlp");
13    exit(1);
14  }
15  if ((pid = fork()) == -1)
16  { perror("fork"); exit(1);
17  }
18  if (pid == 0)
19  { execlp("ps", "ps", NULL);
20    perror("execlp");
21    exit(1);
22  }
23  execlp("ls", "ls", "-l", NULL);
24  perror("execlp");
25  exit(1);
26 }
```

exec1.c (1,1) | ft:c | unix | utf-8 Alt-g: bindings, CtrlG: help


```
bahaj@bahaj-virtual-machine: ~  
1 #include<stdio.h>  
2 #include <stdlib.h>  
3 #include <sys/types.h>  
4 #include <sys/wait.h>  
5 #include <unistd.h>  
6 int main(void)  
7 { pid_t pid;  
8   if ((pid = fork()) == -1)  
9   { perror("fork"); exit(1);  
10  }  
11  if (pid == 0)  
12  { execlp("who", "who", NULL);  
13    perror("execlp");  
14    exit(1);  
15  }  
16  wait(NULL);  
17  if ((pid = fork()) == -1)  
18  { perror("fork"); exit(1);  
19  }  
20  if (pid == 0)  
21  { execlp("ps", "ps", NULL);  
22    perror("execlp");  
23    exit(1);  
24  }  
25  wait(NULL);  
26  execlp("ls", "ls", "-l", NULL);  
27  perror("execlp");  
28  exit(1);  
29 }
```

exec2.c (1,1) | ft:c | unix | utf-8 Alt-g: bindings, CtrlG: help

```
bahaj@bahaj-virtual-machine: ~  
bahaj@bahaj-virtual-machine:~$ micro exec2.c  
bahaj@bahaj-virtual-machine:~$ gcc -o exec2 exec2.c  
bahaj@bahaj-virtual-machine:~$ ./exec2  
bahaj :0 2020-12-24 08:34 (:0)  
PID TTY TIME CMD  
122977 pts/0 00:00:00 bash  
129754 pts/0 00:00:00 xclip  
129792 pts/0 00:00:00 exec2  
129794 pts/0 00:00:00 ps  
total 1004  
-rw-rw-r-- 1 bahaj bahaj 5 déc. 26 18:03 11  
-rw-r--r-- 1 bahaj bahaj 31 janv. 2 17:21 affichageFunction.sh  
-rwxrwxr-x 1 bahaj bahaj 16744 déc. 31 08:27 a.out  
-rwxr-xr-x 1 bahaj bahaj 548 déc. 27 12:35 bash4  
drwxr-xr-x 2 bahaj bahaj 4096 déc. 22 07:08 Bureau  
-rwxr-xr-x 1 bahaj bahaj 191 janv. 2 17:09 case1  
-rwxr-xr-x 1 bahaj bahaj 342 déc. 28 19:36 comptefichiers.sh  
drwxr-xr-x 2 bahaj bahaj 4096 déc. 22 07:08 Documents  
-rwxr-xr-x 1 bahaj bahaj 145 déc. 26 23:37 echanger  
-rwxrwxr-x 1 bahaj bahaj 16744 déc. 31 08:28 exec0  
-rw-r--r-- 1 bahaj bahaj 151 déc. 31 08:27 exec0.c  
-rwxrwxr-x 1 bahaj bahaj 16824 janv. 5 16:32 exec1  
-rwxrwxr-x 1 bahaj bahaj 16744 janv. 2 13:10 exec11  
-rw-r--r-- 1 bahaj bahaj 147 janv. 2 13:05 exec11.c
```


La communication entre processus peut être réalisée par :

- des signaux
- des tubes (pipes)
- de la mémoire partagée
- des sockets

Signaux

- Un signal est
 - envoyé par un processus,
 - reçu par un autre processus,
 - véhiculé par le noyau.
- Signal: mécanisme de communication inter-processus
- Signal : mécanisme de notification d'événements/erreurs et de réactions à d'événements/erreurs
- Ordre de réception aléatoire (différent de l'ordre d'émission)
- Une routine de réception est automatiquement invoquée chez le récepteur dès que le signal arrive

Signal

- Chaque signal a un nom ou numéro
- Un gestionnaire (handler), et généralement associé à un événement/erreur
- Lorsqu'un processus se termine, il informe son père en lui envoyant le signal SIGCHILD (17). Par défaut le processus père ignore ce signal.

tout processus a un processus parent sauf le processus initial

- processus initial : init (pid 1)
- arrêter la machine: demander à init d'arrêter tous ses processus fils.

Signaux sont des moyens de communication entre processus
Par exemple, changer l'état statut /STAT d'un processus.

Les cas les plus courants de **STAT** sont!:

R En cours d'exécution (ie, dans la "run queue" - pas en attente)

S Le processus est dormant ("sleeping") depuis moins de 20s

I Le processus est dormant ("idle") depuis plus de 20s

```
bahaj@bahaj-virtual-machine: ~  
bahaj@bahaj-virtual-machine:~$ ps -aux | head  
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND  
root         1  0.1  0.1 169632 11984 ?        Ss   02:50    0:11 /sbin/init splash  
root         2  0.0  0.0      0     0 ?        S    02:50    0:00 [kthreadd]  
root         3  0.0  0.0      0     0 ?        I<   02:50    0:00 [rcu_gp]  
root         4  0.0  0.0      0     0 ?        I<   02:50    0:00 [rcu_par_gp]  
root         6  0.0  0.0      0     0 ?        I<   02:50    0:00 [kworker/0:0H-kblockd]  
root         9  0.0  0.0      0     0 ?        I<   02:50    0:00 [mm_percpu_wq]  
root        10  0.0  0.0      0     0 ?        S    02:50    0:00 [ksoftirqd/0]  
root        11  0.1  0.0      0     0 ?        I    02:50    0:12 [rcu_sched]  
root        12  0.0  0.0      0     0 ?        S    02:50    0:00 [migration/0]  
bahaj@bahaj-virtual-machine:~$
```

Lister les types des différents signaux: `kill -l`
`NSIG` c'est une constante 64 (le nombre des signaux systèmes).

```
bahaj@bahaj-virtual-machine: ~  
bahaj@bahaj-virtual-machine:~$ ps -aux | head  
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND  
root         1  0.1  0.1 169632 11984 ?        Ss   02:50   0:11 /sbin/init splash  
root         2  0.0  0.0      0     0 ?        S    02:50   0:00 [kthreadd]  
root         3  0.0  0.0      0     0 ?        I<   02:50   0:00 [rcu_gp]  
root         4  0.0  0.0      0     0 ?        I<   02:50   0:00 [rcu_par_gp]  
root         6  0.0  0.0      0     0 ?        I<   02:50   0:00 [kworker/0:0H-kblockd]  
root         9  0.0  0.0      0     0 ?        I<   02:50   0:00 [mm_percpu_wq]  
root        10  0.0  0.0      0     0 ?        S    02:50   0:00 [ksoftirqd/0]  
root        11  0.1  0.0      0     0 ?        I    02:50   0:12 [rcu_sched]  
root        12  0.0  0.0      0     0 ?        S    02:50   0:00 [migration/0]  
bahaj@bahaj-virtual-machine:~$ kill -l  
1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL      5) SIGTRAP  
6) SIGABRT     7) SIGBUS      8) SIGFPE      9) SIGKILL     10) SIGUSR1  
11) SIGSEGV    12) SIGUSR2    13) SIGPIPE    14) SIGALRM     15) SIGTERM  
16) SIGSTKFLT  17) SIGCHLD    18) SIGCONT    19) SIGSTOP     20) SIGTSTP  
21) SIGTTIN    22) SIGTTOU    23) SIGURG     24) SIGXCPU     25) SIGXFSZ  
26) SIGVTALRM  27) SIGPROF    28) SIGWINCH   29) SIGIO       30) SIGPWR  
31) SIGSYS     34) SIGRTMIN   35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3  
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8  
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13  
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12  
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7  
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2  
63) SIGRTMAX-1 64) SIGRTMAX  
bahaj@bahaj-virtual-machine:~$
```

Signaux

Les signaux permettent au système de communiquer avec les processus

signaux utiles

- SIGSTP (19): pause le processus immédiatement [CTRL+Z]
- SIGCONT: reprendre
- SIGHUP (1): hang up demande au processus de se terminer (coupure de ligne)
- SIGKILL (9): Ce signal permet de forcer brutalement la fin d'un processus
- SIGINT (2): Ce signal est envoyé par le shell lorsque l'utilisateur tape [CTRL+C] pendant l'exécution d'un programme. Il provoque normalement la terminaison du processus.
- SIGALRM. Ce signal survient lorsqu'une alarme définie par la fonction alarm ou l'appel système a expiré. Par défaut, la réception de ce signal provoque la terminaison du processus.

Envoyer un signal à un processus

- `int kill(pid_t pid, int n[uo]msignal)`
 - kill envoie le signal `n[uo]msignal` au processus `pid`
 - Quelle valeur pour `n[uo]msignal`?
 - valeur entière (par ex: 9):
 - constante (par ex: SIGKILL) définie dans `signal.h`
- `$ kill -9 1345` /* Cette commande envoie le signal 9 au processus d'identificateur 1345*/.
- Pour informer un processus de s'arrêter temporairement, puis de redémarrer, il faut envoyer les signaux SIGSTP pour l'arrêter, et SIGCONT pour lui dire de reprendre :

`kill (pid ,SIGSTP) # Stop [CTRL+Z]`
`kill (pid,SIGCONT) # Start`
- Le processus suspend temporairement son traitement en cours, réalise celui associé au signal, et reprend le traitement suspendu.

TP

- La fonction `kill()` Ecrire un programme qui crée un processus fils qui affiche à chaque seconde le nombre de secondes écoulées. Le processus père arrête le processus fils au bout de 10 secondes.

```
bahaj@bahaj-virtual-machine: ~  
1 #include<stdio.h>  
2 #include<unistd.h>  
3 #include<signal.h>  
4 int main() {  
5     int i=0;  
6     int pidfiles=fork();  
7     if(pidfiles!=0) {  
8         sleep(10);  
9         kill(pidfiles,SIGKILL);  
10    }  
11    else  
12    {  
13        while(1) {  
14            sleep(1); i++;  
15            printf("%d \n",i);  
16        }  
17    }  
18 }  
19
```

```
kill3.c (1,1) | ft:c | unix | utf-8Alt-g: bindings, Ctr
```

```
bahaj@bahaj-virtual-machine: ~  
bahaj@bahaj-virtual-machine:~$ gcc -o kill3 kill3.c  
bahaj@bahaj-virtual-machine:~$ ./kill3  
1  
2  
3  
4  
5  
6  
7  
8  
9  
bahaj@bahaj-virtual-machine:~$
```

Terminaison d'un processus fils: kill

```
bahaj@bahaj-virtual-machine: ~  
1 #include <unistd.h> // Pour l'instruction fork();  
2 #include <stdio.h>  
3 #include <signal.h> // Pour l'instru  
4 int main ()  
5 {  
6     int i=0;  
7     int pid;  
8  
9     pid =fork();  
10    if (pid==0)  
11    {  
12        printf("Je suis le fils, mon pid est %d\n", getpid());  
13        printf("Mon père son pid est %d il va me tuer\n", getppid());  
14        while(1)  
15            printf("Je suis encore vivant");  
16    }  
17    else  
18    {  
19        sleep(1);  
20        printf("Je suis le père, mon pid est %d\n", getpid());  
21        printf("Je vais tuer mon fils %d\n", pid);  
22        kill(pid,SIGKILL);  
23        printf("Mon fils est mort, je vais mourir moi aussi %d\n", getpid());  
24    }  
25 }
```

kill5.c (1,1) | ft:c | unix | utf-8

```
bahaj@bahaj-virtual-machine: ~  
bahaj@bahaj-virtual-machine:~$ gcc -o kill5 kill5.c  
bahaj@bahaj-virtual-machine:~$ ./kill5 > kill50  
bahaj@bahaj-virtual-machine:~$ micro kill50  
bahaj@bahaj-virtual-machine:~$
```

```
bahaj@bahaj-virtual-machine: ~  
1 Je suis le fils, mon pid est 147258  
2 Mon père son pid est 147257 il va me tuer  
3 Je suis encore vivantJe suis encore vivantJe suis encore vivantJe suis enc  
4 Je vais tuer mon fils 147258  
5 Mon fils est mort, je vais mourir moi aussi 147257  
6
```

Alt-g: bindings, CtrlG: help

Recevoir un signal: Handler

- Le processus qui reçoit le signal :
 - il a un comportement par défaut,
 - il peut modifier le comportement par défaut.
 - il peut ignorer le signal.
- A chaque type de signal est associé dans le système un handler par défaut désigné par SIG_DFL.
- Ce handler définit le comportement par défaut pour chaque type de signal.
 - Terminaison du processus
 - Signal ignoré
 - Suspension du processus
 - Reprise d'un signal stoppé
 - ..
- Tout processus peut installer, pour chaque type de signal (excepté certains SIGKILL, SIGSTOP, SIGCONT) un nouveau handler.

A l'arrivée du signal le processus se comportera en fonction du handler fourni

Nom	Action
SIG_IGN	Le processus ignorera l'interruption
SIG_DFL	Le processus rétablira son comportement par défaut
handler	Le processus exécutera la fonction handler définie par l'utilisateur

La fonction signal:

signal(Num/Nom_du_signal, Nom_Fonction)

- L'appel de la fonction signal défini dans la bibliothèque <signal.h> permet de modifier le traitement par défaut associé au signal, par celui qu'on va définir dans la fonction (Nom_Fonction: handler), qu'on a défini comme paramètre dans la fonction signal.
- Il y a 4 signaux systèmes dont on ne peut pas changer leur traitement, qui sont 9,19, 32,33.
- sleep(n); endort un processus pendant n secondes.

Handler

- Chaque signal a un traitement par défaut mais un processus peut associer un autre traitement.
 - sighandler_t `signal`(int signum, sighandler_t handler)
 - int `sigaction`(int signum, const struct sigaction *act, struct sigaction *oldact);

Attendre un signal

```
#include<unistd.h>  
int pause(void);
```

Suspend le processus appelant jusqu'à l'arrivée d'un signal quelconque.
Retourne -1 et errno = EINTR si un signal a été capté et que le gestionnaire du signal s'est terminé.

Exemple

Soit le programme :

- void **siginhandler**(int num){
- printf(signal SIGINT reçu par %d\n, getpid()); }
- int main(){
- printf(assigner un gestionnaire personnalisé à SIGINT pour le processus %d\n, getpid());
- signal(SIGINT, **siginhandler**);
- printf(C'est fait\n);
- printf(Appuyez sur CTRL+C\n);
- pause();
- Return 0;
- }

```
bahaj@bahaj-virtual-machine: ~  
1 #include <stdio.h>  
2 #include <signal.h>  
3 #include <unistd.h>  
4 #include <sys/types.h>  
5 void siginhandler(int num){  
6     printf("signal SIGINT reçu par %d\n", getpid());}  
7 int main()  
8 {  
9     printf("assigner un gestionnaire personnalisé à SIGINT pour le processus %d\n", getpid());  
10    signal(SIGINT, siginhandler);  
11    printf("C'est fait\n");  
12    printf("Appuyez sur CTRL+c\n");  
13    pause();  
14    return 0;  
15 }  
  
signal0.c (1,1) | ft:c | unix | utf-8
```

```
bahaj@bahaj-virtual-machine: ~  
bahaj@bahaj-virtual-machine:~$ micro signal0.c  
bahaj@bahaj-virtual-machine:~$ ./signal0  
assigner un gestionnaire personnalisé à SIGINT pour le processus 158115  
C'est fait  
Appuyez sur CTRL+c  
^Csignal SIGINT reçu par 158115  
bahaj@bahaj-virtual-machine:~$
```

```

bahaj@bahaj-virtual-machine: ~
1 #include <stdio.h>
2 #include <stdio.h>
3 #include <signal.h>
4 int compteur; int captation=0;
5 void traiter_signal (int sig) {
6 printf ("\nGestionnaire\tCompteur:\t\t\t%d\n", compteur);
7 captation=1; return;
8 }
9 void main(){ signal(SIGUSR1, traiter_signal); signal(SIGUSR2, traiter_signal);
10 for (;;) {compteur++;
11 if (captation) {printf("Main\t\tCompteur après captation\t%d\n", compteur); captation=0;}
12 }
13 }

```

Installation d'un gestionnaire utilisateur avec
signal (signal capté)

- 1) Le signal SIGUSR1 ou SIGUSR2 est capté
- 2) Le gestionnaire traiter_signal s'exécute puis retourne
- 3) le programme reprend sa boucle là où il à été interrompu

signal6.c (1,1) | ft:c | unix | utf-8

```

bahaj@bahaj-virtual-machine:~$ micro signal6.c
bahaj@bahaj-virtual-machine:~$ gcc -o signal6 signal6.c
bahaj@bahaj-virtual-machine:~$ ./signal6 &
[1] 144727
bahaj@bahaj-virtual-machine:~$ kill -10 144727
bahaj@bahaj-virtual-machine:~$
Gestionnaire    Compteur:                1282432734
Main           Compteur après captation 1282432734

bahaj@bahaj-virtual-machine:~$ kill -12 144727

Gestionnaire    Compteur:                -843101383
Main           Compteur après captation -843101383
bahaj@bahaj-virtual-machine:~$

```

TP

Le programme client

Le but de ce programme est d'envoyer les signaux attendus par le programme *serveur*. Dans un premier temps, c'est le signal **SIGUSR1** qui est envoyé et après une pause (avec **sleep**), c'est le signal **SIGUSR2** qui est envoyé. Le PID du programme serveur doit être spécifié en argument (il est affiché à l'écran lors de l'exécution du programme serveur).

Le programme serveur

Dans ce programme, un gestionnaire est placé pour les signaux **SIGUSR1** et **SIGUSR2**. La seule action de ce gestionnaire est de marquer chaque signal reçu. Dans le main, le programme se contente d'attendre que les deux types de signal soient reçus. En attendant, il se met en pause (avec **sleep**).

Tester les programmes.

```

1 #include <signal.h> /* Pour kill */
2 #include <stdlib.h> /* Pour exit, EXIT_SUCCESS, EXIT_FAILURE */
3 #include <stdio.h> /* Pour printf */
4 #include <unistd.h> /* Pour sleep */
5 #include <sys/types.h> /* Pour pid_t */
6
7 int main(int argc, char *argv[]) {
8     pid_t pidServeur;
9
10    /* Recuperation des arguments */
11    if(argc != 2) {
12        fprintf(stderr, "Tapez %s pid\n", argv[0]);
13        fprintf(stderr, "Ou\n\tpid : pid du serveur\n");
14        exit(EXIT_FAILURE);
15    }
16    pidServeur = atoi(argv[1]);
17    /* Envoi du premier signal */
18    printf("Attente avant envoi premier signal\n");
19    sleep(1);
20    printf("Envoi premier signal\n");
21    if(kill(pidServeur, SIGUSR1) == -1) {
22        perror("Erreur lors de l'envoi du signal ");
23        exit(EXIT_FAILURE);
24    }
25    /* Envoi du deuxieme signal */
26    printf("Attente avant envoi deuxieme signal\n");
27    sleep(2);
28    printf("Envoi deuxieme signal\n");
29    if(kill(pidServeur, SIGUSR2) == -1) {
30        perror("Erreur lors de l'envoi du signal ");
31        exit(EXIT_FAILURE);
32    }
33    return EXIT_SUCCESS;
34 }
35

```


13 janv. 16:09

bahaj@bahaj-virtual-machine: ~

```
1 #include <stdlib.h> /* Pour exit, EXIT_SUCCESS, EXIT_FAILURE */
2 #include <signal.h> /* Pour signal */
3 #include <stdio.h> /* Pour printf */
4 #include <unistd.h> /* Pour sleep */
5
6 int cpt = 0; /* Permet de marquer les s
7
8 /**
9  * Ceci est le gestionnaire qui est associ
10  * @param signum le numero du signal recu
11  */
12 void handler(int signum) {
13     if(signum == SIGUSR1) {
14         printf("Signal 1 recu\n");
15         cpt = cpt | 1;
16     }
17     if(signum == SIGUSR2) {
18         printf("Signal 2 recu\n");
19         cpt = cpt | 2;
20     }
21 }
22
23 int main() {
24     struct sigaction action;
25
26     /* Positionnement du gestionnaire pour SIGUSR1 */
27     action.sa_handler = handler;
28     sigemptyset(&action.sa_mask);
29     action.sa_flags = 0;
30     if(sigaction(SIGUSR1, &action, NULL) == -1) {
31         perror("Erreur lors du positionnement ");
32         exit(EXIT_FAILURE);
33     }
34
35     /* Positionnement du gestionnaire pour SIGUSR2 */
36     if(sigaction(SIGUSR2, &action, NULL) == -1) {
37         perror("Erreur lors du positionnement ");
38         exit(EXIT_FAILURE);
39     }
40
41     printf("Pret a recevoir des signaux. Mon PID : %d\n", getpid());
42
43     /* Mise en attente jusqu'a recevoir au moins un signal SIGUSR1 et SIGUSR2 */
44     while(cpt != 3) {
45         sleep(1);
46     }
47     return EXIT_SUCCESS;
48 }
49
```

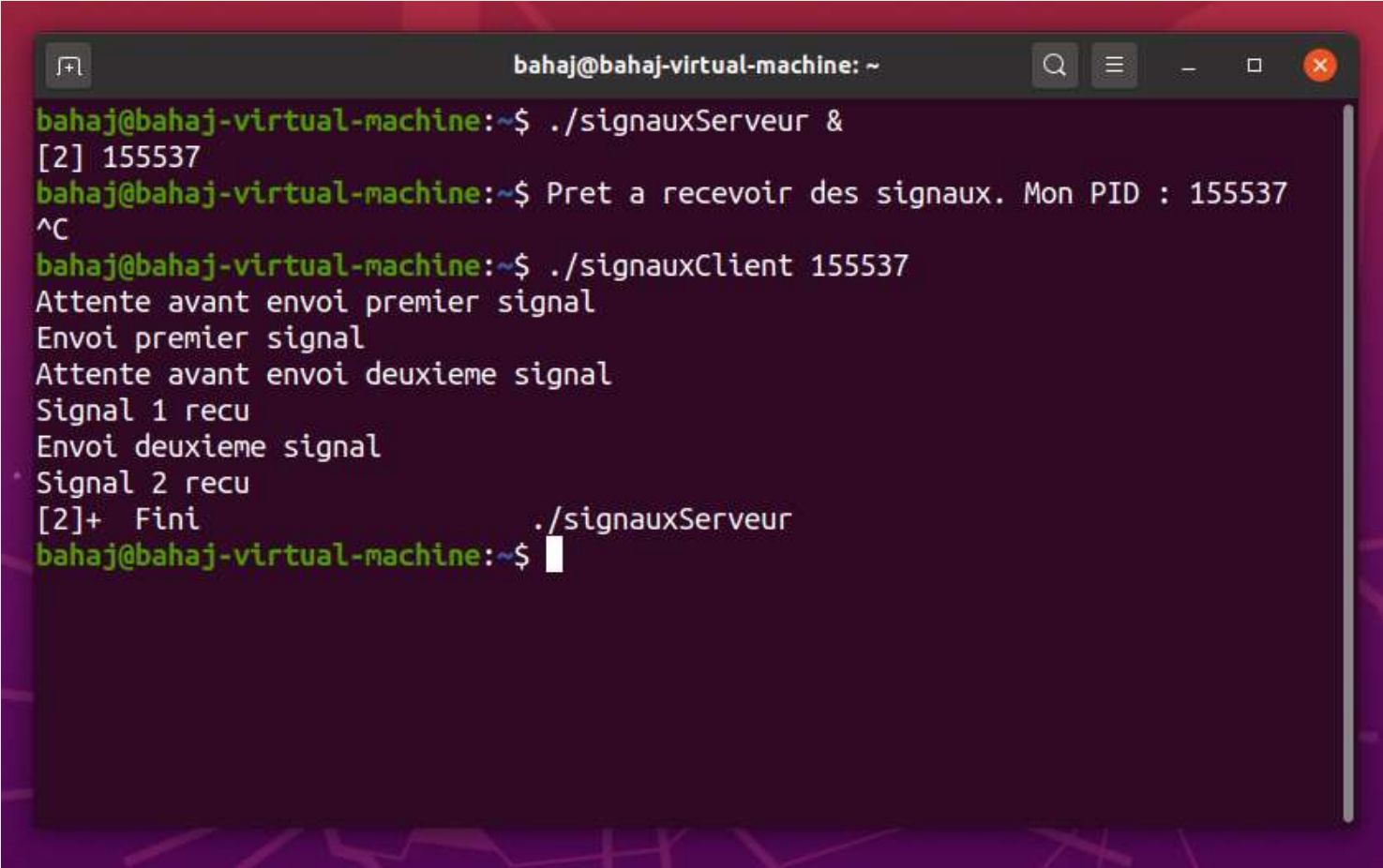
| ou bit à bit
Retourne 1 si l'un ou l'autre
des deux bits de même
poids est à 1 (ou les deux)

signauxServeur.c (49,1) | ft:c | unix | utf-8

Alt-g: bind

Pour tester le programme, il faut d'abord lancer le serveur (on le lance en tâche de fond avec le `&`). Il affiche son PID à l'écran. On lance ensuite le client en spécifiant en argument le PID affiché :

```
./signauxServeur &  
./signauxClient PID
```



```
bahaj@bahaj-virtual-machine: ~  
bahaj@bahaj-virtual-machine:~$ ./signauxServeur &  
[2] 155537  
bahaj@bahaj-virtual-machine:~$ Pret a recevoir des signaux. Mon PID : 155537  
^C  
bahaj@bahaj-virtual-machine:~$ ./signauxClient 155537  
Attente avant envoi premier signal  
Envoi premier signal  
Attente avant envoi deuxieme signal  
Signal 1 reçu  
Envoi deuxieme signal  
Signal 2 reçu  
[2]+  Fini                  ./signauxServeur  
bahaj@bahaj-virtual-machine:~$
```


TP

- Création et Synchronisation d'un processus parent avec ses processus fils
- Ecrire un programme C permettant de saisir par l'utilisateur 4 entiers x, y, z, t et qui crée deux fils, l'un return $x*y$, l'autre $z*t$. Le processus parent doit attendre les résultats des deux fils pour afficher $x*y+z*t$.

TP

- Ecrire un programme qui permet de parcourir l'ensemble des signaux systèmes et modifie leur traitement en affichant un message « le traitement par défaut de ce signal a été changé ». En cas d 'erreur, appeler la fonction handler.
- Soit la fonction handler(traitement) qui affiche le nombre de fois le signal est invoqué.