Chapitre 2 : Etude de Processus UNIX/Linux

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

int main(){
        printf("avant fork");
        fflush(stdout);
        fork();
        printf("apres fork\n");
        return 0;
}
```

Listing 1 – Création de processus par fork()

Output :
avant forkapres fork
apres fork

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

int main(){
        pid_t id;
        id = fork();
        printf("id = %id pid = %id, ppid = %id \n", id, getpid(), getppid()
            );
        return 0;
}
```

Listing 2 – Différenciation du processus pere et du processus fils

Output
id = 67877d pid = 67876d, ppid = 64085d
id = 0d pid = 67877d, ppid = 67876d

```c
#include <stdlib.h>
int main(){
pid_t status;
        status = fork();
        switch (status){
                case -1:
        perror("Creation de processus");
                return -1;
                case0:
                printf("[%d] Je viens de naitre\n", getpid());
                printf("[%d] Mon pere %d\n", getpid(), getppid());
                break;
                default:
                printf("[%d] Jai engendre\n", getpid());
                printf("[%d] Mon fils est %d\n", getpid(), status);
        }
        printf("[%d] Je termine\n", getpid());
        exit(EXIT_SUCCESS);
```

```
}
```

Listing 3 – Utilisation typique de fork()

Output

```
[68275] Jai engendre
[68275] Mon fils est 68276
[68275] Je termine
[68276] Jai engendre
[68276] Mon fils est 0
[68276] Je termine
```

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

int glob = 1;
int main(){
        int loc = 1;
        switch (fork()){
                case -1:
                        perror("Creation de processus");
                        return -1;
                case 0:
                        glob++; loc++;
                        printf("Fils : (%d, %d)\n", glob, loc);
                        break;
                default:
                        sleep(1);
                        printf("Pere : (%d, %d)\n", glob, loc);
        }
        printf("[%d], Je termine\n", getpid());
        return 0;
}
```

Listing 4 – Duplication de la mémoire du processus père

Output

```
Fils : (2, 2)
[73399], Je termine
Pere : (1, 1)
[73398], Je termine
```

```c
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>
#include<stdlib.h>
int main (){
        pid_t id = 0;
        printf ("Processus Pere [%d]\n",getpid());
        if (fork() == 0){
                printf("Processus Enfant [%d] : monpere est %d\n",getpid(),
                    getp>
                exit (0);
        }
```

```
12          id = wait(NULL);
13          printf("Processus␣pere␣[%d]␣:␣mon␣Enfant␣[%d]␣est␣mort␣\n␣", getpid
                ()),i>
14          return 0;
15  }
```

Listing 5 – Exemple d'attente de terminaison d'un processus

Output
Processus Pere [74195]
Processus Enfant [74196] : monpere est 74195
Processus pere [74195] : mon Enfant [74196] est mort

```
1   #include<stdio.h>
2   #include<sys/wait.h>
3   #include<unistd.h>
4   #include<stdlib.h>
5   int main(void){
6           pid_t pid;
7           int status;
8           pid = fork();
9           switch(pid){
10                  case -1:
11                  perror("Error␣dans␣ l appel ␣fork");
12                  exit(1);
13                  case 0 : /*le fils*/
14                  printf("Processus␣fils␣[%d]␣:␣mon␣pere␣est␣[%d]\n",getpid()
                        ,get>
15                  exit(2);
16                  default : /*le pere*/
17                  printf("Pere␣[%d]␣:␣a␣cree␣processus␣[%d]\n",getpid(),pid);
18                  wait(& status);
19                  if(WIFEXITED(status))
20                  printf("Le␣fils␣termine␣normalement␣:␣status␣=␣%d\n",
21                          WEXITSTATUS(status));
22                  else
23                          printf("fils␣termine␣anormalement␣\n");
24          }
25  }
```

Listing 6 – Exemple d'utilisation de macros de wait

Output
Pere [74595] : a cree processus [74596]
Processus fils [74596] : mon pere est [74595]
Le fils termine normalement : status = 2

```
1   #include<stdlib.h>
2   #include<stdio.h>
3   void bilan(void){
4           printf("Vous␣avez␣fait␣1h␣\n");
5   }
6   void paiement(void){
7           printf("vous␣devez␣payer␣10␣frs␣\n");
8   }
9   int main(void){
```

```
        void goodbye(void);
        if (atexit(goodbye) != 0) perror("Error in atexit");
        if (atexit(paiement) != 0) perror("Error in atexit");
        if(atexit(bilan) != 0) perror("Error in atexit");
        exit (0) ;
}
void goodbye(void){
        printf("Goodbye \n");
}
```

Listing 7 – Exemple d'utilisation de atexit

Output
Vous avez fait 1h
vous devez payer 10 frs
Goodbye

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

int main(int argc, char ** argv){
        printf("Processus [%d] \n",getpid());
        switch(fork()){
                case -1 : perror("Creation de processus"); return 1;
                case 0 :
                printf("Processus [%d]    Mon est Pere [%d] \n",getpid(),
                    getppid)
                sleep(5);
                printf("Processus [%d]    Mon est Pere [%d] \n",getpid(),
                    getppid)
                exit(0);
                default :
                sleep(1);
                printf("Processus pere [%d] : je  termine \n",getpid());
                exit(0);
        }
}
```

Listing 8 – Processus orphelins

Output
Processus [75394]
Processus [75395] Mon est Pere [75394]
Processus pere [75394] : je termine

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

int main(){
        pid_t status;
        status = fork();
        switch(status){
```

```
11                  case -1: perror("Erreur de creation de processus"); return
                        1;
12                  case 0 : // Code du fils
13                          switch(fork()){
14                          case -1: perror("Erreur de creation de processus
                               interm>
15                                  return 1;
16                          case 0 :
17                                  printf("Processus [%d] , Mon Pere : [%d] \n
                        ",ge>
18                                  sleep(5);
19                                  printf("Processus [%d] , Mon Pere : [%d] \n
                        ",ge>
20                                  break;
21                          default : return 0;
22                          };
23                          break ;
24                  default :
25                          wait(&status); break;
26          };
27          exit(EXIT_SUCCESS);
28  }
```

Listing 9 – Double fork

Output

Processus [76193] , Mon Pere : [76192]

```
1  #include<stdio.h>
2  #include <unistd.h>
3
4  int main(){
5          execl("/bin/ls","ls",NULL);
6          printf(" je ne suis pas mort \n");
7          return 0;
8  }
```

Listing 10 – Recouvrement

Output

ex10.out ex2.out ex6.out example10.c example2.c example6.c
ex11.out ex3.out ex7.out example11.c example3.c example7.c
ex12.out ex4.out ex8.out example12.c example4.c example8.c
ex1.out ex5.out ex9.out example1.c example5.c example9.c

```
1  #include<stdio.h>
2  #include<unistd.h>
3  #include<stdlib.h>
4  int main (){
5          if(fork() == 0) execl ("/bin/ls","ls",NULL);
6          else{
7                  sleep(2);
8                  printf("Je suis le pere et je continue mon travail");
9          }
10 }
```

Listing 11 – Recouvrement exec() avec de la primitive fork()

Output

ex10.out ex2.out ex6.out example10.c example2.c example6.c
ex11.out ex3.out ex7.out example11.c example3.c example7.c
ex12.out ex4.out ex8.out example12.c example4.c example8.c
ex1.out ex5.out ex9.out example1.c example5.c example9.c
Je suis le pere et je continue mon travail

```c
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
int main(){
        if(fork() ==0 ) execl("/bin/ls","ls",NULL);
        else{
                sleep(2);
                printf("Je␣suis␣le␣pere␣et␣je␣continue␣mon␣travail");
        }
}
```

Listing 12 – Recouvrement : exec() avec de la primitive fork()

Output

ex10.out ex2.out ex6.out example10.c example2.c example6.c
ex11.out ex3.out ex7.out example11.c example3.c example7.c
ex12.out ex4.out ex8.out example12.c example4.c example8.c
ex1.out ex5.out ex9.out example1.c example5.c example9.c
Je suis le pere et je continue mon travail