Les Processus sous Unix

Les fonctions fondamentales

Duplication (Création) de Processus

pid_t fork(void)

Tout processus a un seul père.

Tout processus peut avoir zéro ou plusieurs processus fils.

Recouvrement (Chargement) de Processus

int execl(char *path, char *arg0, char *arg1, ... , char *argn, (char *)0)

int execv(char *path, char *argv[])

int execle(char *path, char *arg0, char *arg1, ... , char *argn, (char *)0, char **envp)

int execlp(char *file, char *arg0, char *arg1, ... , char *argn, (char *)0)

int execvp(char *file, char *argv[])

int execve(char *path, char *argv[], char **envp)

```
// int execv(char *path, char *argv[])
   ...
            argv[0] = "ls";
            argv[1] = "-l";
            argv[2] = NULL;
            execv("/bin/ls", argv);

// int execlp(char *file, char *arg0, char *arg1, ... , char *argn, (char *)0)

            execlp("ls","ls","-l",NULL);
```

Il n'est pas nécessaire ici de spécifier le chemin d'accès (/bin/ls) pour l'exécutable.

Pour recouvrir un processus avec la commande **ps -aux**,
nous pouvons utiliser le code suivant :

```
#include <unistd.h>
#include <stdlib.h>
int main( ) {
        char *arg[3];
        arg[0] = "ps";
        arg[1] = "-aux";
        arg[2] = NULL;
        execv("/bin/ps", arg);  /* l'éxécutable ps se trouve dans /bin */
        fprintf(stderr, "erreur dans execv\n");
        return 0;
}
```

Un processus se termine lorsqu'il n'a plus d'instructions
ou lorsqu'il exécute la fonction
void exit(int statut)

L'élimination d'un processus terminé de la table
ne peut se faire que par son père, grâce à la fonction :
int wait(int *code_de_sortie)

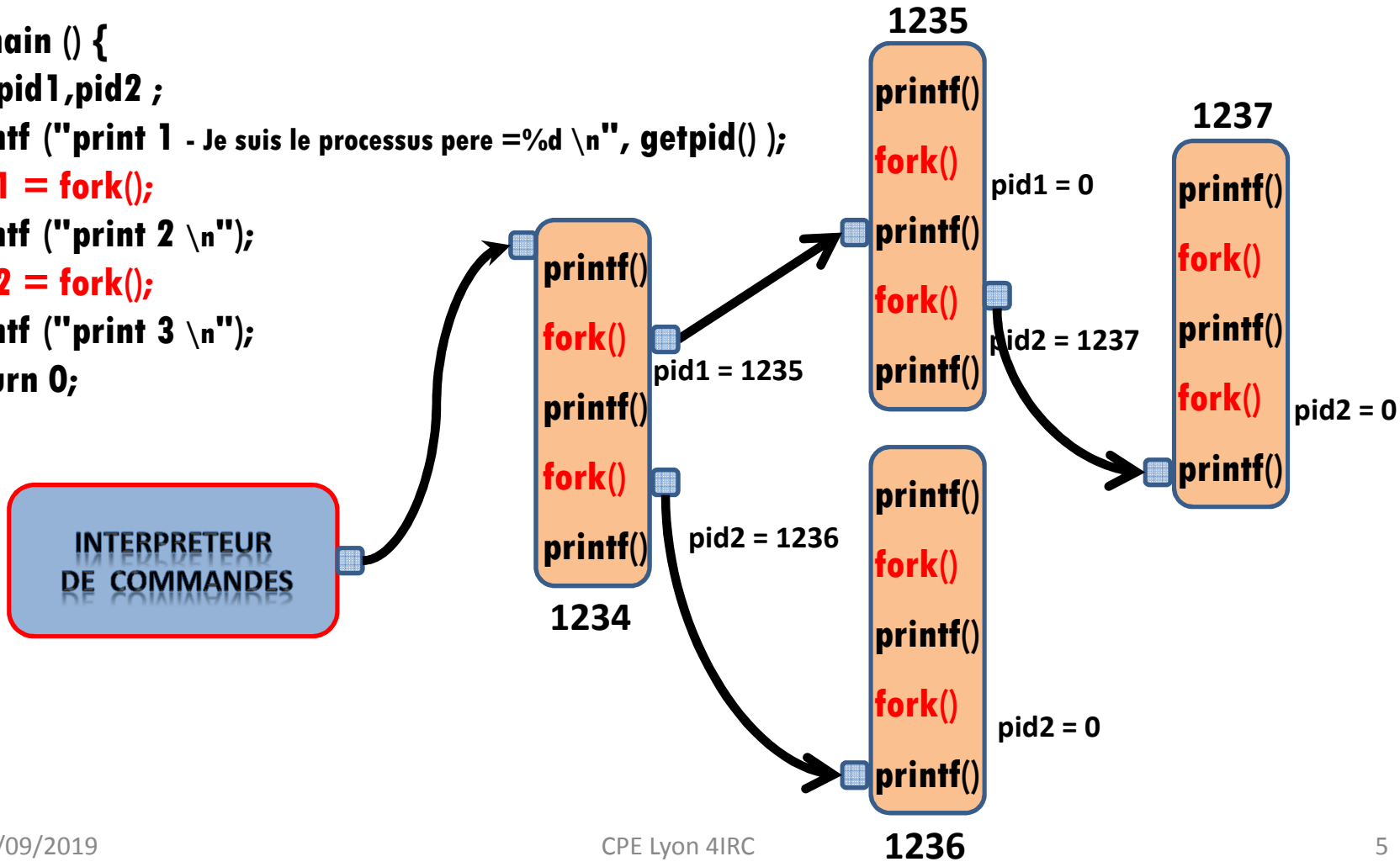
Grâce aux 3 instructions, fork( ), exec( ), et wait( )

on peut écrire un interpréteur de commandes simplifié.
Il prend la forme suivante :

```
while(1) {
        lire_commande(commande, paramètres);        /* attend commande*/
        if (fork() != 0) wait(&statut);             /* processus père */
        else execv(commande, paramètres);           /* processus fils */
}
```

Exemple – fork()

```
int main () {
  int pid1,pid2 ;
  printf ("print 1 - Je suis le processus pere =%d \n", getpid() );
  pid1 = fork();
  printf ("print 2 \n");
  pid2 = fork();
  printf ("print 3 \n");
  return 0;
}
```



exit(i)

termine un processus,

i est un octet (valeurs possibles : **0 à 255**)

retourné dans une variable du type int au processus père.

Wait(&Etat)

met le processus en **attente** de la **fin** de l'un de **ses processus fils**.

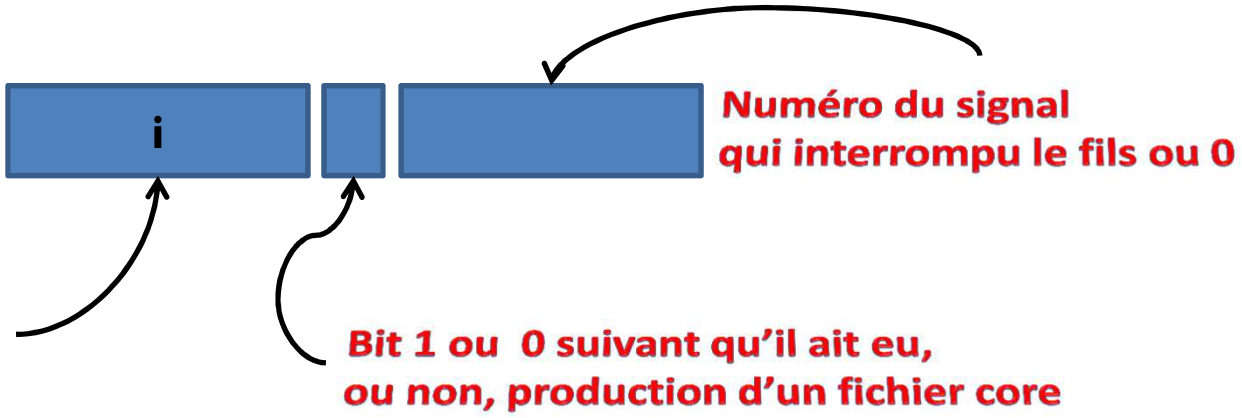
Quand un processus se termine, le signal **SIGCHILD** est envoyé à son père.

La réception de ce signal fait passer le processus père de l'état **bloqué** à l'état **prêt**.

Le processus père sort donc de la fonction wait().

Etat

est un pointeur
sur un mot de 2 octets



Le père attend son fils

```
int main( ) {
        int pid, status ;
        if (! fork() )
        {
            printf(''Le processus fils %d \n'', getpid());
            exit(10);
        }
        pid = wait (&status) ;
        printf(''Le processus père %d \n'', getpid());
        printf(''Sortie du wait \n'');
        sleep(15) ;
        printf(''pid = %d statud = %d \n'', pid,status);
        return 0;
}
```

Le père n' attend pas son fils et est toujours en vie après la terminaison de son fils

```
int main() {
        int pid, status ;
        if (! fork() )
        {
            printf(''Le processus fils %d \n'', getpid());
            exit(10);
        }
        printf(''Le processus père %d \n'', getpid());
        for( ; ;) ;
        return 0;
}
```

Le père reçoit le signal de terminaison de son fils et n'exécute le wait() qu'après Le fils reste zombie momentanément

```
int main() {
  int pid, status ;
  if (! fork() )
  {
     printf(''Le processus fils %d \n'', getpid());
     exit(10);
  }
  printf(''Le processus père %d \n'', getpid());
  sleep(15) ;
  printf(''sortie du premier sleep \n'');
  pid = wait(&status) ;
  printf(''sortie du wait \n'');
  sleep(15) ;
  printf(''pid = %d status = %d \n'', pid,status);
  return 0;
}
```

Le père n'attend pas son fils mais se termine avant celui ci. Le fils devient orphelin

```
int main() {
  int pid, status ;
  if (! fork() )
  {
    printf(''Le processus fils %d \n'', getpid());
    sleep(60);
    exit(10);
  }
  printf(''Le processus père terminé %d \n'', getpid());
  return 0;
}
```

Commenter ce programme
(préciser la fonctionnalité réalisée par ce programme).

```
int main (int argc, char* argv[]) {
  char *f[20];
  int status, i, j = 0;
  for (i=1 ; i<=argc; i++) {
     if ( fork()==0) {
          execlp (argv[i], argv[i], NULL);
          exit(3);
     }
     wait(&status);
     if (WIFEXITED(status)) {
               printf("Terminaison normale du processus fils.\n");
               if( WEXITSTATUS(status) == 3) {   //récupérer le code de retour
                    f[j] = (char *)malloc(strlen(argv[i])+1);
                    strcpy(f[j],argv[i]);     // strcpy(s1, s2) : copie la chaîne de caractère s2 dans s1
                    j++;
               }
    }
  }
  if(j != 0) for (i=0; i<j; i++) printf("%s\n", f[i]) ;
  return 0;
}
```

Considérer le programme suivant :

```
int main ( ) {
  int n=0;
  pid_t pid;
  int status ;
  if ( ( pid = fork ()) == -1 ) exit ( -1 );
 if( pid != 0) {
       n++;
       wait (& status );
       if ( WIFEXITED ( status )) n = n + WEXITSTATUS (status);
   }
 else { n -- ; }
       printf ("[% d ] : valeur de n est %d\n", getpid (),-n);
       exit (n);
}
```

Détailler (ligne par ligne)
le comportement de ce programme.

EXERCICE 2 [/2 points] Étant donné le programme suivant :

```
int main() {
   int i=10;
   int s;
   if(fork()==0)
   {
      i=20;
      exit(i);
      i=1;
   }
   wait(&s);
   printf("%d",i);
   return 0;
}
```

Qu'affiche ce programme ? cocher la (les) bonne(s) réponse(s) et justifier votre réponse.

10 ……

20 ……

1 ……

0 ……

N'affiche rien ……

Dessiner l'arbre généalogique des processus engendrés par ce programme suivant :

```
int main( ) {

   fork( ) && ( fork() || fork() ) ;

   return 0;

}
```

On considère le programme suivant :

```
int main( ) {
   int i=0;
   fork( );
   fork( );
   fork( );
   i++;
   printf("%d\n",i);
   return 0 ;
}
```

Combien de lignes sont affichées ?
Combien de processus sont créés ?
Quelles sont les valeurs affichées ?

Combien le programme suivant affichera d'étoiles ?

Pourquoi ?

```
int main(){
    int i ;
    for(i=0 ; i<5 ; i++) {
            putchar('*') ;
            fork();
    }
    return 0;
}
```

Programme1.c

```
int main( ) {
   int p=1 ;
   while(p>0) p=fork()
   execlp("prog", "prog", NULL) ;
   return 0 ;
}
```

Programme2.c

```
int main( ) {
   int p;
   int i=2;
   while(i-- &&  p=fork() );
   if (p<0) exit(1) ;
   return 0 ;
}
```

Programme3.c

```
int main ( ) {
   int p ;
   int i=2 ;
    j=10;
    while(i-- && p = fork()) if(p<0) exit(1) ;
    j += 2;
    if (p == 0) {   i *= 3; j *= 3; }
    else { i *= 2;  j *= 2; }
    printf(« i=%d, j=%d », i,j) ;
    return 0 ;
}
```

Programme4.c

```
int i=4, j=10;
int main ( ) {
  int p ;
  p = fork();
  if(p<0) exit(1) ;
  j += 2;
  if (p == 0){ i *= 3; j *= 3; }
  else { i *= 2; j *= 2;  }
  printf("i=%d, j=%d", i,j) ;
  return 0 ;
}
```

Programme5.c

```
int main ( ) {
   int p=1 ;
   for(int i=0 ; i<=4 ; i++)
       if (p>0)  p=fork( ) ;
   if(p !=-1) execlp("prog", "prog", NULL) ;
   else exit(1) ;
   while( wait(NULL) !=-1) ;
  return 0 ;
}
```

```
main(int argc, char *argv[] () {
 int i,n,m,p;
 n = atoi(argv[1]);
 p = atoi(argv[2]);
 if(p==0) {
            sleep(3) ;
            printf(‘‘sortie 1\n’’);
            exit(0);
 }
 for (i=0 ; i<n ; i++)
            if (fork() == 0) {
                        sprintf(argv[2], ’’%d’’ , p-1);
                        main(argc , argv);
                        printf(‘’sortie 2\n’’) ;
                        exit(0);
            }
 while(wait(0) != -1);
 printf‘(‘‘sortie 3\n’’);
 return 0;
}
```

```
int main( ) {
        int n = 100 ;
        printf(''Bonjour →'');
        n *= 2;
        if (fork() == 0) {
                sleep(1) ;
                printf(''dans le fils – adresse de n = %p\n'',&n);
                n += 10 ; sleep(1) ; printf(''n = %d\n'', n);
        }
        else {
                 printf(''dans le père – adresse de n = %p\n'',&n);
                n += 10 ; sleep(3) ; printf(''n = %d\n'', n);
        }
        return 0
}
```

```
$ prog
Bonjour → dans le père – adresse de n = 142e8
Bonjour → dans le fils – adresse de n = 23200
N = 210
N = 220
$
```