Systèmes et Réseaux : Réalisations d'un mini shell

Grand Maxence, Muller Lucie

17/02/2017

Table des matières

Introduction

Durant le projet, nous avons créé un interprêteur de commande shell mais d'une façon simplifiée. Il a fallut en comprendre la structure pour la refaire, et permettre a notre programme d'accèder à des commandes simples puis à des commandes plus complexes gèrant les redirections d'entrées-sorties ainsi que les séquences de commandes pipées.
Dans un second temps, nous avons amélioré notre shell pour lui permettant la gestion des différents plans, des signaux de terminaison et de suspension, des zombis et des jobs.

Chapitre 1

Partie 1

1.1 Commandes simples

Dans cette partie nous devions implémenté une fonction permettant d'exécuter une commande simple en shell. Pour tester noôtre fonction nous avons tester plusieurs commande n'utilisant ni les pipes ni les redirections dans un shell et dans nôtre programme shell.
Code :

```
/*
Cette fonction permet d'executer une commande simple sans
    redirection et sans tube
*/
void commande_simple(struct cmdline *l){
  //Processus pere cree un processus fils qui devra executer la
      commande
  int pid = Fork();
  int status;
  if(pid == 0){
    /*Si on est le fils alors
      on execute la commande passe en paramettre
      on s'arrette
    */
    execvp(l->seq[0][0], l->seq[0]);
    exit(0);
  }else{
    /*
    Le pere attend la fin de son fils
    */
    while(waitpid(pid, &status, 0) != pid);
  }
}
```

Test
SHELL

```
maxence@Sybil:~ ls
```

```
csapp.c  Makefile     rapport.log  rapport.tex  readcmd.c  README.
    md  shell.c    test   test3  tst    tube_simple.c
csapp.h  rapport.aux  rapport.pdf  rapport.toc  readcmd.h  shell
         sujet.pdf  test2  text   tst.c  waitpid1.c
maxence@Sybil:~ ls -l
total 2688
-rwx------ 1 maxence maxence  20259 dÃ©c.  25  2014 csapp.c
-rwx------ 1 maxence maxence   6105 mars  16  2014 csapp.h
-rwx------ 1 maxence maxence    136 oct.   6  2009 Makefile
-rwx------ 1 maxence maxence   1499 fÃ©vr.  8 15:58 rapport.aux
-rwx------ 1 maxence maxence  30949 fÃ©vr.  8 15:58 rapport.log
-rwx------ 1 maxence maxence 128494 fÃ©vr.  8 15:58 rapport.pdf
-rwx------ 1 maxence maxence   2558 fÃ©vr.  8 16:02 rapport.tex
-rwx------ 1 maxence maxence    933 fÃ©vr.  8 15:58 rapport.toc
-rwx------ 1 maxence maxence   4699 fÃ©vr.  8 14:07 readcmd.c
-rwx------ 1 maxence maxence   1029 fÃ©vr.  8 14:02 readcmd.h
-rwx------ 1 maxence maxence     14 fÃ©vr.  7 18:02 README.md
-rwx------ 1 maxence maxence  41024 fÃ©vr.  8 15:45 shell
-rwx------ 1 maxence maxence   4338 fÃ©vr.  8 16:03 shell.c
-rwx------ 1 maxence maxence  94332 fÃ©vr.  7 12:01 sujet.pdf
-rwx------ 1 maxence maxence    150 fÃ©vr.  8 13:57 test
-rwx------ 1 maxence maxence      3 fÃ©vr.  8 13:58 test2
-rwx------ 1 maxence maxence      3 fÃ©vr.  7 15:51 test3
-rwx------ 1 maxence maxence      0 fÃ©vr.  7 15:36 text
-rwx------ 1 maxence maxence  18008 fÃ©vr.  8 12:08 tst
-rwx------ 1 maxence maxence    723 fÃ©vr.  7 13:55 tst.c
-rwx------ 1 maxence maxence    815 janv. 31 15:18 tube_simple.c
-rwx------ 1 maxence maxence    657 janv. 20  2009 waitpid1.c
maxence@Sybil:~ cat test
19
pp.c
csapp.h
Makefile
readcmd.c
readcmd.h
README.md
shell
shell.c
shell.o
sujet.pdf
test
test2
test3
text
tst
tst.c
tst.o
tube_simple.c
waitpid1.c
```

shell.c

```
shell> ls
csapp.c  Makefile     rapport.log  rapport.tex  readcmd.c  README.
    md  shell.c  test test3  tst    tube_simple.c
csapp.h  rapport.aux  rapport.pdf  rapport.toc  readcmd.h  shell
         sujet.pdf  test2  text   tst.c  waitpid1.c
shell> ls -l
```

```
total 2688
-rwx------ 1 maxence maxence  20259 dÃ©c.  25  2014 csapp.c
-rwx------ 1 maxence maxence   6105 mars  16  2014 csapp.h
-rwx------ 1 maxence maxence    136 oct.   6  2009 Makefile
-rwx------ 1 maxence maxence   1499 fÃ©vr.  8 15:58 rapport.aux
-rwx------ 1 maxence maxence  30949 fÃ©vr.  8 15:58 rapport.log
-rwx------ 1 maxence maxence 128494 fÃ©vr.  8 15:58 rapport.pdf
-rwx------ 1 maxence maxence   4558 fÃ©vr.  8 16:05 rapport.tex
-rwx------ 1 maxence maxence    933 fÃ©vr.  8 15:58 rapport.toc
-rwx------ 1 maxence maxence   4699 fÃ©vr.  8 14:07 readcmd.c
-rwx------ 1 maxence maxence   1029 fÃ©vr.  8 14:02 readcmd.h
-rwx------ 1 maxence maxence     14 fÃ©vr.  7 18:02 README.md
-rwx------ 1 maxence maxence  41024 fÃ©vr.  8 16:06 shell
-rwx------ 1 maxence maxence   4338 fÃ©vr.  8 16:03 shell.c
-rwx------ 1 maxence maxence  94332 fÃ©vr.  7 12:01 sujet.pdf
-rwx------ 1 maxence maxence    150 fÃ©vr.  8 13:57 test
-rwx------ 1 maxence maxence      3 fÃ©vr.  8 13:58 test2
-rwx------ 1 maxence maxence      3 fÃ©vr.  7 15:51 test3
-rwx------ 1 maxence maxence      0 fÃ©vr.  7 15:36 text
-rwx------ 1 maxence maxence  18008 fÃ©vr.  8 12:08 tst
-rwx------ 1 maxence maxence    723 fÃ©vr.  7 13:55 tst.c
-rwx------ 1 maxence maxence    815 janv. 31 15:18 tube_simple.c
-rwx------ 1 maxence maxence    657 janv. 20  2009 waitpid1.c
shell> cat test
19
pp.c
csapp.h
Makefile
readcmd.c
readcmd.h
README.md
shell
shell.c
shell.o
sujet.pdf
test
test2
test3
text
tst
tst.c
tst.o
tube_simple.c
waitpid1.c
shell>  echo "test"
"test"
shell>
```

1.2 Commande avec redirections d'entrée ou de sortie

Dans cette section nous devions implenter une fonction permettant de gerÍes redirections d'entrée ou de sortie. Nos test ont suivi les même principes que

l'étape précédente à l'exception que seul les commandes avec pipes ne seront pas testées ici.
Code

```
/*
Cette fonction permet d'executer une commande avec redirection des
    flux sans pipe
*/
void commande_redirection(struct cmdline *l){
  /*
  fIn : le descripteur de fichier d'entrer, initialiser a 1, l'
      entree standard
  fOut : le descripteur de fichier de sortie, initialiser a 1, le
      sortie standard
  */
  int fOut = 1; int fIn = 0;

  if(l->in != NULL)
    /*
    Si, ete present, '> <fic>' alors fIn prend la valeur du
        descripteur de <fic>
    */
    fIn = open(l->in, O_RDONLY,0);
  if(l->out != NULL)
    fOut = open(l->out, O_WRONLY | O_CREAT, 0700);
    /*
    Si, ete present, '< <fic>' alors fOut prend la valeur du
        descripteur de <fic>
    */
  int pid = Fork(); int status;
  if(pid == 0){
    if(fOut != 1){
      /*
      Si fOut diffrenrent de 1 alors
        On ferme la sortie standard
        La sortie de l'execution devient le fichier descrit par
            fOut
      */
      close(1);
      dup2(fOut, 1);
    }

    /*
      Si fIn diffrenrent de i alors
        On ferme l'entree standard
        L'entree de l'execution devient le fichier descrit par fIn
        On execute
        On s'arrete
      Sinon
        On execute
        On s'arrete
      */
    if(fIn == 0){
      execvp(l->seq[0][0], l->seq[0]);
      exit(0);
    }else{
      close(0);
```

```
      dup2(fIn, 0);
      execvp(l->seq[0][0], l->seq[0]);
      exit(0);
    }

  }
  else{
    /*
    Le pere attend la fin d'execution du fils
    */
    if(l->bg){
        ajout_job(l, pid, 0);
      }else{
        child = pid;
        Signal(SIGINT, stop);
        Signal(SIGTSTP, suspend);
        ajout_job(l, pid, 1);
        //printf("creer\n");
        while(waitpid(pid, &status, WUNTRACED) != pid);
        if(getStatus(pid) != 2 && getStatus(pid)!=-1){
          supprime_job(pid);
        }
        //printf("supprimer\n");
      }
  }
}
```

Test
SHELL

```
maxence@Sybil:~ ls -l > test
maxence@Sybil:~ cat test
total 2688
-rwx------ 1 maxence maxence  20259 dÃ©c.  25  2014 csapp.c
-rwx------ 1 maxence maxence   6105 mars  16  2014 csapp.h
-rwx------ 1 maxence maxence    136 oct.   6  2009 Makefile
-rwx------ 1 maxence maxence   1499 fÃ©vr.  8 16:10 rapport.aux
-rwx------ 1 maxence maxence  31866 fÃ©vr.  8 16:10 rapport.log
-rwx------ 1 maxence maxence 218712 fÃ©vr.  8 16:10 rapport.pdf
-rwx------ 1 maxence maxence   7408 fÃ©vr.  8 16:16 rapport.tex
-rwx------ 1 maxence maxence    933 fÃ©vr.  8 16:10 rapport.toc
-rwx------ 1 maxence maxence   4699 fÃ©vr.  8 14:07 readcmd.c
-rwx------ 1 maxence maxence   1029 fÃ©vr.  8 14:02 readcmd.h
-rwx------ 1 maxence maxence     14 fÃ©vr.  7 18:02 README.md
-rwx------ 1 maxence maxence  41024 fÃ©vr.  8 16:06 shell
-rwx------ 1 maxence maxence   4338 fÃ©vr.  8 16:03 shell.c
-rwx------ 1 maxence maxence  94332 fÃ©vr.  7 12:01 sujet.pdf
-rwx------ 1 maxence maxence      0 fÃ©vr.  8 16:16 test
-rwx------ 1 maxence maxence      3 fÃ©vr.  8 13:58 test2
-rwx------ 1 maxence maxence      3 fÃ©vr.  7 15:51 test3
-rwx------ 1 maxence maxence      0 fÃ©vr.  7 15:36 text
-rwx------ 1 maxence maxence  18008 fÃ©vr.  8 12:08 tst
-rwx------ 1 maxence maxence    723 fÃ©vr.  7 13:55 tst.c
-rwx------ 1 maxence maxence    815 janv. 31 15:18 tube_simple.c
-rwx------ 1 maxence maxence    657 janv. 20  2009 waitpid1.c
maxence@Sybil:~ wc -l < test
23
```

```
maxence@Sybil:~ wc -c < test
1347
maxence@Sybil:~ wc - < test > test2
maxence@Sybil:~ cat test2
  23   200 1347 -
```

shell.c

```
shell> ls -l > test
shell> cat test
total 2816
-rwx------ 1 maxence maxence  20259 dÃ©c.  25  2014 csapp.c
-rwx------ 1 maxence maxence   6105 mars  16  2014 csapp.h
-rwx------ 1 maxence maxence    136 oct.   6  2009 Makefile
-rwx------ 1 maxence maxence   1499 fÃ©vr.  8 16:10 rapport.aux
-rwx------ 1 maxence maxence  31866 fÃ©vr.  8 16:10 rapport.log
-rwx------ 1 maxence maxence 218712 fÃ©vr.  8 16:10 rapport.pdf
-rwx------ 1 maxence maxence   9314 fÃ©vr.  8 16:22 rapport.tex
-rwx------ 1 maxence maxence    933 fÃ©vr.  8 16:10 rapport.toc
-rwx------ 1 maxence maxence   4699 fÃ©vr.  8 14:07 readcmd.c
-rwx------ 1 maxence maxence   1029 fÃ©vr.  8 14:02 readcmd.h
-rwx------ 1 maxence maxence     14 fÃ©vr.  7 18:02 README.md
-rwx------ 1 maxence maxence  41024 fÃ©vr.  8 16:23 shell
-rwx------ 1 maxence maxence   4343 fÃ©vr.  8 16:23 shell.c
-rwx------ 1 maxence maxence  94332 fÃ©vr.  7 12:01 sujet.pdf
-rwx------ 1 maxence maxence      1 fÃ©vr.  8 16:17 test
-rwx------ 1 maxence maxence      1 fÃ©vr.  8 16:17 test2
-rwx------ 1 maxence maxence      3 fÃ©vr.  7 15:51 test3
-rwx------ 1 maxence maxence      0 fÃ©vr.  7 15:36 text
-rwx------ 1 maxence maxence  18008 fÃ©vr.  8 12:08 tst
-rwx------ 1 maxence maxence    723 fÃ©vr.  7 13:55 tst.c
-rwx------ 1 maxence maxence    815 janv. 31 15:18 tube_simple.c
-rwx------ 1 maxence maxence    657 janv. 20  2009 waitpid1.c
shell> wc -l < test
23
shell> wc -c < test
1347
shell> wc - < test > test2
shell> cat test2
  23   200 1347 -
```

1.3 Séquence de commandes composée de deux commandes reliés par un tube

Dans cette section nous devions implanter une fonction permettant de gérer les commandes reliés par un ou plusieurs tubes. Nos test ont suivi les même principes que l'étape précédente à l'exception que seul les commandes avec redirections ne seront pas testées ici.
Code

```
/*
Cette fonction permet d'executer une suite d'instruction pipee sans
     redirection de flus d'entrees/sorties
```

```
*/
void commande_pipe(struct cmdline *l){
  int tailleSeq; int tmp;
  //On calcul le nb de commande pipee
  for(tailleSeq=0; l->seq[tailleSeq+1]!=0; tailleSeq++);
  int pid = Fork(); int status; int i=0; int desc[2];
  if(pid == 0){
    /*
    Le premier fils creer pipe qu'il partageras avec son futur fil
    Il ferme son entrer standard et recupere comme entree la sortie
         du pipe
    Attend la fin de son fils
    s'execute
    s'arrete
    */
    pipe(desc);
    pid=Fork();
    if(pid != 0){
      dup2(desc[0], 0);
      close(desc[1]);
      while( (tmp = waitpid(pid, &status, WNOHANG|WUNTRACED)) !=
          pid);
      execvp(l->seq[1][0], l->seq[tailleSeq]);
      close(desc[0]);
      exit(0);
    }else{
      /*
      Tant qu'il reste des commandes suivantes
        Si nous sommes la derniere commande
          On ferme sa sortie standard et recupere comme sortie l'
              entree du pipe partage avec le pere
          on cree un pipe
          On cree un processus fils
          Si on est le pere
            On ferme  son entrer standard et recupere comme entree
                la sortie du pipe
            on attend la fin de son fils
            On s'execute
            On s'arrete
        Sinon
          On ferme sa sortie standard et recupere comme sortie l'
              entree du pipe partage avec le pere
          on cree un pipe
          On s'execute
          On s'arrete
      */
      for(i = 1; i<=tailleSeq; i++){
        if(i+1 <= tailleSeq){
          dup2(desc[1], 1);
          close(desc[0]);
          pipe(desc);
          pid = Fork();
          if(pid != 0){
            dup2(desc[0], 0);
            close(desc[1]);
            while(waitpid(pid, &status, 0) != pid);
            execvp(l->seq[tailleSeq - i][0], l->seq[tailleSeq-1]);
```

```
            exit(0);
          }
        } else{
          dup2(desc[1], 1);
          close(desc[0]);
          execvp(l->seq[0][0], l->seq[0]);
          exit(0);
        }
      }
    }
  }else{
    // le processus pere attend la fin de tous ses descendants.
    while(waitpid(pid, &status, 0) != pid);
  }
}
```

Test
SHELL

```
shell> ls -l | wc -l
23
shell> ls -l | wc -l | wc -c
3
```

shell.c

```
maxence@Sybil:~ ls -l | wc -l
23
maxence@Sybil:~ ls -l | wc -l | wc -c
3
```

1.4 Séquence de commandes composée de plusieurs commandes et plusieurs tubes

Dans cette section nous devions implanter une fonction permettant de gérer les commandes reliés par un ou plusieurs tubes et avec redirections. Nos test ont suivi les même principes que l'étape précédentetous les types de commandes sont testées ici.
Code

```
/*
Cette fontion permet d'executer une sequence de commandes pipees
    avec redirection de flus d'entrees/sorties
*/
void commande1_final(struct cmdline *l){
  if(l->seq[1]!=0){
    /*
    SI nous avons une sequences de commande pipe alors
      On gere les les flux d'entree/sortie comme pour la fonction
          commande_redirection
      La suite est gerer de la meme facon que dans fonction
          commande_pipe
    */
```

```
    int fOut = 1; int fIn = 0;
      if(l->in != NULL)
      fIn = open(l->in, O_RDONLY,0);
    if(l->out != NULL)
      fOut = open(l->out, O_WRONLY | O_CREAT, 0700) ;
    commande_pipe(l,fOut, fIn );
  }else{
    /*Si nous n'avons pas de commande pipees alors
      nous appelons la fonction commande_redirection
    */
    commande_redirection(l);
  }
}
```

Test
SHELL

```
maxence@Sybil:~ head -5 < test | tail -2 | sort > tmp2
maxence@Sybil:~ cat tmp2
-rwx------ 1 maxence maxence    136 oct.   6  2009 Makefile
-rwx------ 1 maxence maxence   1499 fÃ©vr.  8 16:10 rapport.aux
```

shell.c

```
shell>  head -5 < test | tail -2 | sort > tmp
shell> cat tmp
-rwx------ 1 maxence maxence    136 oct.   6  2009 Makefile
-rwx------ 1 maxence maxence   1499 fÃ©vr.  8 16:10 rapport.aux
```

Chapitre 2

Partie Bonus

2.1 Exécution de commandes en arrière plan

Dans cette section, nous avons implémenté une fonction permettant de faire fonctionner des processus en arrière plan : ils continuent de fonctionner sans bloquer le shell, ce qui permet a l'utilisateur de continuer de travailler et taper d'autres commandes pendant que le processus continue.
Code :

```
/*
Cette fonction permet de gerer les commandes avec &
*/
void commande_bg(struct cmdline *l){
  /*
  Si la commande est suivi d'un & alors
    Alors la commande est en second plan et le pere n'attend pas la
         fin de l'execution de son fils
  SInon
    Le pere attend la fin de l'execution de son fils
  */
  int pid, status;

  pid = Fork();

  if(pid == 0){
    execvp(l->seq[0][0], l->seq[0]);
    exit(0);
  }else{
    if(l->bg);
    else{
      waitpid(pid, &status, 0);
    }
  }
}
```

Test
SHELL :

```
maxence@Sybil:~ kate &
[1] 7381
maxence@Sybil:~ kate
```

shell.c

```
shell> kate &
7347
shell> kate
7362
```

2.2 Changer l'état du processus au premier plan

Dans cette section, on a modifié notre interprêteur afin qu'il reconnaisse certains signaux agissant sur les processus en premier plan qui sont les signaux de terminaison et de suspension respectivement envoyés par la pression des touches Ctrl-C et Ctrl-Z.
Code :

```
void stop(int sig){
  printf("Fini\n");
  kill(child, SIGKILL);
}
void suspend(int sig){
  printf("Suspendu\n");
  kill(child, SIGSTOP);
}
/*
Cette fonction permet de gerer les sinaux SIGINT et SIGSTSTP
*/
void commande_signaux(struct cmdline *l){

  int pid = Fork();
  int status;
  Signal(SIGINT, stop);
  Signal(SIGTSTP, suspend);

  if(pid == 0){
    execvp(l->seq[0][0], l->seq[0]);
    exit(0);
  }else{
    child = pid;
    while(waitpid(pid, &status, WUNTRACED) != pid);
    //Le pere attend un signal tant que son fils n'a pas fini son
        execution
  }
}
```

Test
SHELL :

```
maxence@Sybil:~ gedit
^C
[2]+  Fini                    okular
```

```
maxence@Sybil:~ gedit
^Z
[1]+  ArrÃªtÃ©                 gedit
```

shell.c

```
shell> kate
7498
^Z
Suspendu 7498
shell> kate
7514
^Z
Suspendu 7514
```

2.3 Gestion des zombis

Dans cette section nous devons gèrer les cas de prosessus fils qui n'ont pas terminés lors de la terminaison du shell, appelés zombis. Pour cela nous avons ajouté un signal qui le fait attendre jusqu'à la terminaison de chacun de ces fils, avant de se terminer lui-même.
Code :

```
void zombi(int sig)
{
  /*
  Attend la fin des processus zombis
  */
    waitpid(-1, NULL, WNOHANG|WUNTRACED);
}

void commande_zombi(struct cmdline *l){
  Signal( SIGCHLD, zombi);
  int pid = Fork();
  int status;
  if(pid == 0){
    execvp(l->seq[0][0], l->seq[0]);
    exit(0);
  }
}
```

2.4 Commande integrée jobs

Dans cette section, nous avons implémenter la fonction Jobs du shell permettant d'afficher sur la sortie standard les differents job démarrés durant la session courante de ce shell. Quand jobs indique la terminaison d'un job, il l'efface de la liste et donc ce job n'est plus dans la liste à l'appel suivant de jobs.
Code :

```
/*
Structure representant les jobs
*/
struct job{
  char* cmd; //Commande du processus
  int pid; //pid du processus
  int status; //status du job (bg, fg suspend)
};

/*
fonction permettant de gerer l'ajout de processus dans les jobs
*/
void ajout_job(struct cmdline *l, int pid, int status){
  /*
  Alloue ou realloue la memoire pour notre structure job
  */
  if(nbEnCours == 0){
    enCours = malloc(sizeof(struct job*)*(nbEnCours+1));
    enCours[nbEnCours] = malloc(sizeof(struct job)*1);
  }else{
    enCours = realloc(enCours, (sizeof(struct job*)*(nbEnCours+1)))
        ;
    enCours[nbEnCours] = malloc(sizeof(struct job)*1);
  }
  /*Copie du nom de la commande du processus*/
  enCours[nbEnCours]->cmd = malloc(sizeof(char)*strlen(l->seq
      [0][0]));
  memcpy(enCours[nbEnCours]->cmd, l->seq[0][0], strlen(l->seq
      [0][0]));
  /*
  Copie du numÃ©ro de pid et du status
  */
  enCours[nbEnCours]->pid = pid;
  enCours[nbEnCours]->status = status;
  nbEnCours++;
}

/*
Affiche les jobs en cours
*/
void affiche_job(){
    int i;
    for(i=0; i<nbEnCours; i++){
      switch(enCours[i]->status){
        case 0:
          printf("[%d] \t En cours d'execution en arriere plan \t%d
               \t%s\n", i+1,enCours[i]->pid,  enCours[i]->cmd);
          break;
        case 1:
          printf("[%d] \t En cours d'execution au premier plan\t%d\
              t %s\n", i+1,enCours[i]->pid, enCours[i]->cmd);
          break;
        default:
          printf("[%d] \t Suspendu \t%d\t %s\n", i+1, enCours[i]->
              pid, enCours[i]->cmd);
      }
    }
```

```
}

/*
Fonction permettant de gerer l'ajout et l'affichage de job
*/
void commande_job(struct cmdline *l){
  int pid, status;

  pid = Fork();
  Signal(SIGINT, stop);
  Signal(SIGTSTP, suspend);
  if(pid == 0){
    execvp(l->seq[0][0], l->seq[0]);
    exit(0);
  }else{
    if(l->bg){
      ajout_job(l, pid, 0);
    }
    else{
      child = pid;
      ajout_job(l, pid, 1);
      while(waitpid(pid, &status, WUNTRACED) != pid);
      supprime_job(pid);
    }
  }
}
```

Test
SHELL :

```
maxence@Sybil:~$ jobs
maxence@Sybil:~$ kate &
[1] 7769
maxence@Sybil:~$ jobs
[1]+  En cours d'exÃ©cution    kate &
maxence@Sybil:~$
```

shell.c :

```
shell> jobs
shell> kate &
shell>
jobs
[1]    En cours d'execution en arriere plan   7871  kate
shell>
```

2.5 Agir sur les commandes en arrière plan

Dans cette section, il fallait que le programme agisse sur les jobs en arriére plan. Pour cela il y avait trois commandes a implémenter qui sont fg pour ramener un jobs au premier plan, bg pour mettre un jobs en arrière plan et stop pour arrêter le jobs.
Code :

```
/*
Fonction permettant de supprimer de Pid pid
*/
void supprime_job(int pid){
  printf("delete %d\n",pid);
  /*
  Si il y a au moins 2 jobs, alors on modifie enCours pour qu'il n'
      y plus le job de Pid pid
  dans la structure enCours
  */
  if(nbEnCours>1){
    struct job** tmp = (struct job**)malloc(sizeof(struct job *)*(
        nbEnCours-1));
    int i;
    int j = 0;
    for(i=0; i<nbEnCours; i++){
      if(enCours[i]->pid != pid){
        tmp[j] = malloc(sizeof(struct job)*1);
        memcpy(tmp[j], enCours[i],sizeof(struct job));
        tmp[j]->cmd = malloc(sizeof(char)*strlen(enCours[i]->cmd));
        memcpy(tmp[j]->cmd,enCours[i]->cmd,sizeof(char)*strlen(
            enCours[i]->cmd));
        j++;
      }
    }
    if(j==nbEnCours-1){
      nbEnCours--;
      free_job(enCours, nbEnCours);
      enCours = malloc(sizeof(struct job *)*nbEnCours );
      memcpy(enCours,tmp,sizeof(struct job *)*nbEnCours );
    }else{
      free_job(tmp, j);
    }
  /*
  Si il n'y a qu'un seul job dans enCours
  Alors ont libere la memoire allouee pour enCours
  */
  }else if(nbEnCours == 1){
    free_job(enCours, nbEnCours);
    nbEnCours = 0;
  }else;
}

/*
Fonction permettant de mettre a jours le status des jobs
*/
void maj_job(int pid, int action){
  if(nbEnCours == 0)
    printf("Aucun job en cours\n");
  else{
    int k;
    switch(action){
      case 0: //bg
        /*
        Si la commande bg a ete utilisee alors
          le processus doit s'executer au second plan
          on envoie doonc un signal SIGCONT pour reveiller le
```

```
        prcessus
      Le processus n'attend pas la fin du processus pour
        reprendre la main
    */
    for(k = 0; k<nbEnCours && enCours[k]->pid != pid; k++);
    if(k<nbEnCours){
      printf("processus %d mis au second plan\n", pid);
      kill(pid, SIGCONT);
      enCours[k]->status = action;
    }
    else
      printf("Il n'y pas de processus dont le pid est : %d\n",
        pid);
    break;
  case 1: //fg
    /*
    Si la commande fg a ete utilisee alors
      le processus doit s'executer au premier plan
      on envoie doonc un signal SIGCONT pour reveiller le
        prcessus
      Le processus pere attend pas la fin du processus pour
        reprendre la main
    */
    for(k = 0; k<nbEnCours && enCours[k]->pid != pid; k++);
    if(k<nbEnCours){
      child = pid;
      printf("processus %d mis au premier plan\n", pid);
      kill(pid, SIGCONT);
      enCours[k]->status = action;
      affiche_job();
      int tmp = waitpid(pid, NULL, WUNTRACED);
      printf("test\n");
      while(tmp != pid){
        tmp = waitpid(pid, NULL, WUNTRACED);
      }
      if(getStatus(pid) != 2 && getStatus(pid) != -1)
        supprime_job(pid);
    }
    else
      printf("Il n'y pas de processus dont le pid est : %d\n",
        pid);
    break;
  case 2: //stop
  /*
    Si la commande stop a ete utilisee alors
      le processus doit etre suspendu
      on envoie doonc un signal SIGSTOP pour suspendre le
        prcessus
    */
    for(k = 0; k<nbEnCours && enCours[k]->pid != pid; k++);
    if(k<nbEnCours){
      printf("processus %d Suspendu\n", pid);
      kill(pid, SIGSTOP);
      enCours[k]->status = action;
    }
    else
      printf("Il n'y pas de processus dont le pid est : %d\n",
```

```
            pid);
        break;
      default:
        printf("Action impossible\n");
    }
  }
}
void maj_job2(int idx, int action){
  if(nbEnCours == 0)
    printf("Aucun job en cours\n");
  else
    if(idx <=0 || idx > nbEnCours)
      printf("Action impossible, l'indice du pid doit etre en 1 et
          %d\n",nbEnCours);
    else{
      int pid = enCours[idx-1]->pid;
      maj_job(pid, action);
    }
}

/*
Fonction permettant de gerer les fonctions en premier/second plan
    ou suspendu, et les jobs.
*/
void commande(struct cmdline *l){
  /*
  Attente de signaux
  */
  Signal(SIGCHLD, zombi);
  Signal(SIGINT, stop);
  Signal(SIGTSTP, suspend);
  /*
  Permet de gerer les commandes fg, bg, stop.
  C'est le preocessus pere qui s'occupe de mettre a jour les jobs
  */
  if((strcmp(l->seq[0][0], "bg") == 0) || (strcmp(l->seq[0][0], "fg
      ") == 0) || (strcmp(l->seq[0][0], "stop") == 0)){
    if(l->seq[0][1][0] == '%'){
      char tmp[strlen(l->seq[0][1])];
      memcpy(tmp, l->seq[0][1]+1, 4);
      int idx = atoi(tmp);
      if(strcmp(l->seq[0][0], "bg") == 0){
        maj_job2(idx, 0);
      }else if(strcmp(l->seq[0][0], "fg") == 0){
        maj_job2(idx, 1);
      }else{
        maj_job2(idx, 2);
      }
    }else{
      int pid = atoi(l->seq[0][1]);
      if(strcmp(l->seq[0][0], "bg") == 0){
        maj_job(pid, 0);
      }else if(strcmp(l->seq[0][0], "fg") == 0){
        maj_job(pid, 1);
      }else{
        maj_job(pid, 2);
      }
```

```
    }
  }else if(l->seq[1]!=0){
    /*
    Si nous avons une sequences de commande pipe alors
    On utilise la fonction commande_pipe modifiÃ© pour erer l'ajout
        /suppression de job
    */

    int fOut = 1; int fIn = 0;
      if(l->in != NULL)
      fIn = open(l->in, O_RDONLY,0);
    if(l->out != NULL)
      fOut = open(l->out, O_WRONLY | O_CREAT, 0700) ;

    commande_pipe(l,fOut, fIn );

  }
  else{
    /*Si nous n'avons pas de commande pipees alors
      nous appelons la fonction commande_redirection modifiÃ© pour
          pouvoir gerer l'jout/suppression de jobs
    */
    commande_redirection(l);
  }
}
```

Test
SHELL :

```
maxence@Sybil:~$ jobs
maxence@Sybil:~$ kate
^Z
[1]+  ArrÃªtÃ©                 kate
maxence@Sybil:~$ jobs
[1]+  ArrÃªtÃ©                 kate
maxence@Sybil:~$ fg %1
kate
^C
maxence@Sybil:~$ jobs
maxence@Sybil:~$ kate &
[1] 8367
maxence@Sybil:~$ okular &
[2] 8381
maxence@Sybil:~$ jobs
[1]-  En cours d'exÃ©cution   kate &
[2]+  En cours d'exÃ©cution   okular &
maxence@Sybil:~$ jobs
[1]-  En cours d'exÃ©cution   kate &
[2]+  En cours d'exÃ©cution   okular &
maxence@Sybil:~$ fg %2
okular
^C
maxence@Sybil:~$ jobs
[1]+  En cours d'exÃ©cution   kate &
```

shell.c :

```
shell> kate
^Zprocessus 8797 Suspendu
shell> jobs
[1]    Suspendu   8797   kate
shell> fg %1
processus 8797 mis au premier plan
[1]    En cours d'execution au premier plan 8797   kate
^C
Fini 8797
delete 8797
shell> jobs
shell> kate &
shell> jobs
[1]    En cours d'execution en arriere plan   8809  kate
shell> stop %1
processus 8809 Suspendu
shell> jobs
[1]    Suspendu   8809   kate
shell> exit
```