Réponse 1-7 : Jean-Léo DARY

1_SN_I

Question 1) j'ai implanter une simple boucle while (stop != 0){}. Ensuite en lisant attentivement le module readcmd.[c\h]. Il était facile d'extraire les commandes des touches tapées par l'utilisateur il suffisait d'écrire les quelques lignes suivantes pour avoir les informations.

```
        /* lecture Commande */

        commande = readcmd();  /* Lecture de la commande */
        seq = commande->seq;
        backgrounded = commande->backgrounded;
```

Ensuite pour exécuter les commandes grâce à :

```
      error = execvp(seq[0][0], seq[0]);
```

Il fallait que cette ligne de code ne recouvre pas notre programme principal d'où la nécessité d'utilisée des processus fils grâce à la ligne :

```
      pid = fork();
```

Question 2) En effet sans :

```
      fflush(stdout);
```

les fils récupèrent le tampon non vide ce qui peut entrainer des incohérences au niveau de l'affichage chronologique des informations.

Question 3) pour cela il nous faut rajouter un waitpid pour attendre que le processus fils finis la commande. Pour cela j'ai crée une fonction nommé forground() pour désigner les processus d'avant plan (ceux qu'on attend forcément donc) en voici le code :

```
void foreground(int pid) {
    int status;
    if (waitpid(pid, &status, 0) < 0)
    {
        printf("Erreur waitpid");
    }
    if (WIFEXITED(status))
    {
        supprimer(pid,&tab_jobs);
    }
    else
    {
    }

}
```

Édit : Certaines lignes on changé au vu des questions d'après j'ai les ai mise à jour.

Question 4)

-la commande cd a eu droit a son propre module comportant 2 fonctions majeures : l'une pour changer le dossier courant l'autre pour le récupérer sous forme de caractère. Le code est commenté voir le module cd.h cd .c pour plus de détail.

-La commande exit fut facile a codé puisqu'il faut régler dans se cas la variable stop à 1 pour quitter la boucle.

Question 5)

Pour mettre en tâche de fond il suffit d'introduire un if sur le waitpid précédent et donc ne plus attendre la fin du fils. Cette méthode n'est pas sans contrainte puisqu'on a toujours le retour du fils sur l'écran ce qui n'est pas optimal.

Question 6)

- La commande listée à demandé à faire appel à un module gérant des listes,au vu de sa structure spéciale contenant id,pid,cmd, j'ai préfère l'appel processus.[h\c]. Une fois ce module fonctionnel il suffit d'ajouter à une liste le processus quand il est lancé. Cependant pour le supprimer il a fallu créer un handler sur les signaux SIGHLD pour déterminer quand il se terminait (et donc les enlever de la liste). Ceci a pu être efficacement fait grâce aux ressources mises sur moodle. Ci-dessous la structure de la liste et le handler :

```
void suivi_fils (int sig) {

    sig ++;
    int etat_fils, pid_fils;

    do {

        pid_fils = (int) waitpid(-
1, &etat_fils, WNOHANG | WUNTRACED | WCONTINUED);

        if ((pid_fils == -1) && (errno != ECHILD)) {

            perror("waitpid");

            exit(EXIT_FAILURE);

        } else if (pid_fils > 0) {

            if (WIFSTOPPED(etat_fils)) {

                chgtEtat(pid_fils,SUSPENDU,&tab_jobs);
```

```
            } else if (WIFCONTINUED(etat_fils)) {

                chgtEtat(pid_fils,ACTIF,&tab_jobs);

            } else if (WIFEXITED(etat_fils)) {

                supprimer(pid_fils,&tab_jobs);

            } else if (WIFSIGNALED(etat_fils)) {

                /* traiter signal */

            }

        }

    } while (pid_fils > 0);
    /* autres actions après le suivi des changements d'état */
}
```

Mise en place grâce à :

```
    signal(SIGCHLD, suivi_fils);
```

Structure de la liste :

```
struct processus {
    int id;
    int pid;
    etat e;
    char * cmd;
    proc suivant;
};
```

- La commande stop nécessite de définir l'état d'un processus. Pour cela j'ai du définir une énumération contenant : SUSPENDUE, ACTIF. Il suffit de lui envoyer le signal SIGSTOP. Grâce à la ligne :

```
        kill(get_pid(id,tab_jobs),SIGSTOP);
```

- Ensuite grâce au handler définit précédemment il était facile de mettre à jours jobs.
- La commande bg n'est pas très compliquée puisqu'il faut envoyé là même ligne que précédemment, mais avec SIGCONT
- La commande fg est pareille que bg, mais on rappel forground() définit plus haut.

Pour mettre en place ctrlZ j'ai décidé d'opter pour masquer les signaux SIGTSTP pour les fils. Et de les traduire pour le minishell en SIGSTOP. Plus de détail dans la fonction void ctrlZ.

Question 7) pour le ctrlC il c'est le même principe que pour ctrlZ, mais en remplaçant SIGINT par SIGKILL cette fois.