TINAZZI Giovanni VANSON Nathan

RAPPORT TP6

Objectifs

L'objectif principal de ce TP etait de creer son propre shell avec lequel on pourra executer des programmes du systeme. Ce shell devait etre capable de gerer les processus (et d'eviter les zombies), et de gerer les signaux envoyes au shell

PARTIE BUILTIN

Les commandes builtin sont des commandes implementes directement dans le shell. Ici, notre but etait d implementer la commande "cd", qui allait nous permettre de changer de repertoire courant, et puis la commande "exit" qui etait charge de quitter le shell.

Dans ce module, on y trouve 3 fonctions

```
void cd_func(char* path);
```

Cette fonction va prendre le path, et avec la fonction `chidir()` va changer le repertoire. Cette fonction retourne -1 en cas d'echec et 0 si tout s est bien passe.

```
void exit_shell(void);
```

Cette fonction va nous permettre de quitter notre shell. Pour que le shell se termine, etant un processus, il faut envoyer un signal au pid pour qu'il se termine. On envoie un signal avec la ligne

```
kill(pid, SIGHUP)
```

Ou `SIGHUP` va terminer tous les processus, meme ceux en background, pour eviter les zombies

```
void task_background_config();
```

est une fonction qui ne vas pas s'occuper d'implementer des commandes, mais plutot de configurer le mask de l'enfant. Cette fonction vas nous etre utile lorsque nous sommes dans un processus enfant et que ce processus va devoir ignorer `SIGINT` pour eviter de fermer tout le shell.

PARTIE JOBS

Ce module, qui s occupe surtout de la partie des signaux, va nous permettre de pouvoir gerer les differents signaux qui vont etre envoyes

```
void handler(int signum);
```

cette fonction s'occupe de gerer `SIGINT` et ```SIGHUP``

Le premier va etre utilise lorsque l'on veux permettre a l utilisateur d'utiliser CTRL+C pour arreter un job principal sans stopper le shell

Le deuxieme lui va s'occuper de terminer les processus en background et en foreground (c'est pour cela que l'on check les deux types de pid)

```
void child_handler(int signum, siginfo_t *siginfo, void* unused);
```

cette fois ci, cette fonction s'occupe des processus lorsque ceux la sont des processus dit "enfant". On va devoir etre sur que ce processus enfant (en background) va etre terminer avec succes, pour eviter les zombies

```
void set_mask(void)
```

Enfin, cette derniere fonction va s'occuper de configurer tous les mask, en fonction du signal envoye

PARTIE SHELL

Dans cette derniere partie, qui est en quelque sorte notre main, on va utiliser les deux modules precedents pour que notre shell puisse marcher.

Tout d'abord, on va faire une boucle `while` qui vas nous permettre de garder actif notre shell a chaque fois que la touche entree est tape. Dans cette boucle, les lignes 30 à 48 sont essentielles puisque elle permettent de "parse" l'input, c'est a dire de diviser par mot le texte tapee par l utilisateur, pour que le shell puisse reconnaitre une eventuelle commande.

Apres avoir parse l'input, on va d'abord regarder si l'input utilisateur est une commande de type builtin ("cd" et "exit"). Si ce n'est pas le cas, c'est que l'on doit executer soit une commande en background soit un job

On va d'abord check si on se trouve dans un processus enfant ou parent. Si on se trouve dans un processus enfant, on va d'abord check si c'est un processus en background ou pas (un processus est en background avec la commande "&" a la fin)

Sinon on se retrouve dans la partie job Pour la partie job, lorsqu l'on tape par example "ls" notre shell doit etre capable d'aller chercher le path.

```

```

```
int res = execvp(_argv[0], _argv);
```

la fonction `execvp` va nous permettre de lancer n'importe quel programme qui n'est pas un builtin

Si c'est un processus parent, on va check d'abord si c'est un processus en background ou pas

Comment utiliser le programme

```
./shell
```

sera la commande a taper pour que notre shell s'active

Ensuite si on veux changer de repertoire, on utilise la commande:

```
cd
```

avec le repertoire que l'on veux, attention a ne pas mettre de "/" a la fin ni d'utiliser un tab pour completer comme dans un linux

pour mettre un processus en background, on fait:

```
[commande] &
```

& va mettre le processus en background et on pourra continuer a utiliser notre shell.