Nathan
Foucher

Rapport Systèmes d'exploitation centralisés

Questions :

- Questions 1 à 10 : traitées
- Question 11 : non traitée (manque de temps)

Architecture et choix :

Le mini Shell est divisé en plusieurs parties. Le programme principal (minishell.c) utilise en effet les codes présents dans :

- intcommand.c : gère les commandes internes (lj, bg, etc)
- processus.c : gère la liste des processus
- readcmd : donné avec le sujet

Dans son fonctionnement général le minishell affiche « >> » quand il est en attente d'une commande. Il va ensuite d'abord regarder s'il s'agit d'une commande interne. Si tel est le cas il réalise directement les opérations nécessaires.

Dans le cas d'une commande externe, le minishell va créer un fils où la commande sera exécutée. Dans le cas d'une commande au 1er plan le processus père (le minishell) va attendre la fin d'exécution du fils à moins que l'utilisateur appuie sur ctrl-c ou ctrl-z. Pour une commande en arrière-plan le père continu de s'exécuter normalement.

A chaque nouvelle commande externe le minishell enregistre dans la liste des processus le processus lancé. A chaque signal envoyé par un fils (SIGCHLD) le processus père va exécuter les opérations selon les cas associées. Par exemple lors de la fin d'un fils, le signal SIGCHLD est reçu par le père qui va ensuite savoir à l'aide de waitpid dans quel état est le fils. Dans le cas d'une fin du fils, « WIFEXITED » sera vérifié et le minishell va retirer le processus de la liste des processus.

Pour la gestion des ctrl-z et ctrl-c le choix a été fait premièrement de dire aux fils d'ignorer les signaux SIGINT et SIGTSTP grâce à sigprocmask. Ensuite pour que le minishell sache s'il faut arrêter un processus ou non lors de l'appuie sur les touches d'interruption, il a été fait le choix d'avoir une variable pid_fg qui contient soit le pid du processus au 1er plan soit -1 si aucun processus n'est au premier plan. Si pid_fg est différent de -1 alors on interrompt le processus au 1er plan.

Les tubes ont étés mis en place avec la fonction pipe pour rediriger les entrées et sorties standards. Malheureusement je n'ai pu réaliser les redirections qu'entre deux processus n'ayant pas eu le temps d'aller plus loin. Pour plus de processus je pense qu'il

aurait suffit de créer plusieurs fils pour chaque processus différent et rediriger successivement les entrées et sorties.

Problèmes rencontrés et résolutions

Le plus gros problème que j'ai pu rencontrer était le fait que certains processus en arrière-plan était « perdu » par le père. Je n'arrivais pas à les gérer correctement. Depuis le rendu intermédiaire j'ai pu comprendre après conseils de mon professeur que cela était dû à l'utilisation de plusieurs waitpid. Ainsi certains waitpid attendais toujours un fils qui avait déjà été pris en charge par d'autres waitpid. J'ai résolu ce problème en ne mettant qu'un waitpid dans le child_handler (gère les interruptions des fils).

Tests

Pour tester les processus en arrière-plan j'ai tout simplement utilisé la commande « sleep & » et regardé si le minishell se mettait en pause ou non. Comme on s'y attendait le minishell ne se met pas en pause.

Pour tester la redirection vers des fichier la commande « cat < f1 > f2 » copie bien le contenu de f1 dans f2 comme prévu.

Enfin pour l'utilisation des tubes simples la commande « ls | wc -l » donne bien le nombre de fichiers dans le répertoire comme on s'y attend.

Après chaque modification majeure j'ai bien évidement re-testé les éléments précédents pour voir si les modifications n'ont pas eu d'influence sur les choses déjà fonctionnelles.