DUQUENNOY Antoine
GONTARD Benjamin

Groupe de TD numéro 2

RAPPORT DE PROJET :
MINI SHELL

Structure globale du programme :

Notre projet se découpe en plusieurs sous-fichiers :

-readcmd.c/.h :

Contient une structure permettant de récupérer les commandes que l'utilisateur souhaite piper ainsi que les fonctions permettant de remplir cette structure (fonctions de parsing).
Ces fichiers nous étaient fournis, cependant nous avons apporté une modification au niveau de la structure afin de savoir si l'utilisateur souhaite que la commande se réalise en background (arrière plan) ou si il souhaite qu'elle s'exécute en foreground (premier plan). Ce champ se nomme bg (pour background) et est de type entier (int).
Nous avons aussi modifié la fonction readcmd afin de détecter le caractère spécial '&' pour savoir si il faut lancer la commande en premier plan ou en arrière plan.

-jobs.h :

Contient des constantes STOPPED et RUNNING qui facilitent la lecture du code (plus explicite que des 0 ou des 1).
Contient un tableau de chaînes de caractères pour éviter de stocker une chaîne de caractère ne pouvant contenir que deux valeurs distinctes (Stopped et Running).
Contient une structure permettant de connaître différentes informations sur job en cours d'exécution, qu'il soit en pause ou non. Pour cela, on stocke son pid, le nom de la commande et l'état dans lequel celui-ci est (« Stoppé » ou « en cours d'exécution »).

-miniShell.c :
Contient le programme principal et les fonctions nous permettant de gérer la création de tubes, la communication entre les différentes commandes pipées et la gestion des signaux.

Réalisations :

Quelques précisions sont à apporter quand à l'organisation générale de notre code afin de faciliter la compréhension de la partie suivante :
-Nous avons en variable globale une table des "jobs" qui nous permet de stocker tout les processus qui sont actuellement en pause ou en arrière plan.
-Nous gardons dans une variable globale les informations concernant le processus en premier plan, notamment son pid, si il y en a un.
-Une variable globale permet de savoir à quel indice de la table des jobs doit être inséré le processus à mettre en arrière plan suivant, celui ci fonctionne de la manière suivante :

* Lorsque l'on insère un job dans la table, alors on incrémente son indice de 1.
* Si lorsque l'on supprime un job et que celui-ci était le dernier inséré dans la table de jobs, alors on décrémente son indice de 1.
* Si lorsque l'on supprime un job et que celui ci n'était pas le dernier dans la table des jobs, alors l'indice ne bouge pas. Nous avons recrée le comportement du shell bash tel qu'il est sur les machines de l'IM2AG.

-Le comportement des commandes spéciales telles que « jobs », « bg », « fg » et « stop » est spécial : lorsque l'utilisateur entre une commande sur le shell, nous examinons le nom de la première commande afin de savoir si elle fait partie des mots clés spécifiques précédents afin de réaliser un traitement spécifique.

-Gestion de commandes jointes par des tubes :

Il est possible de d'exécuter plusieurs commandes en parallèle. Ces commandes communiquent via des pipes.
Le processus principal s'occupe de gérer les commandes qui seront lancées en parallèle et qui communiqueront via des tubes.
Pour chaque commande, le processus principal crée un fils qui va exécuter sa commande avec la fonction « execvp ».

-Gestion de la redirection d'entrée/sortie :

Grâce au caractère clé '<', il est possible de rediriger l'entrée standard par le fichier suivant le caractère clé '<'.
De manière similaire, la sortie standard sera redirigée dans le fichier dont le nom suit le caractère spécial '>'.
A noter que seule la dernière commande pipée redirige sa sortie dans le fichier de sortie et que seule la première commande pipée lit son entrée dans le fichier reçu. La première commande pipée écrit dans le tube et la dernière commande prend son entrée dans le tube. Si il y a plus de deux commandes pipées, alors les autres commandes prennent leur entrée dans le tube et écrivent dans le tube.

-Exécution de commandes en arrière plan :

Si le caractère spécial '&' est spécifié à la fin de la commande que le shell doit exécuter, alors celui-ci lance cette tâche en arrière plan.
Pour cela, nous avons modifié la structure cmdline, qui nous avait été fournie.
Nous avons ajouté un champ de type entier nommé bg (pour background). bg aura pour valeur 0 si l'utilisateur ne souhaite pas lancer sa commande en arrière plan et vaudra 1 si l'utilisateur souhaite

lancer sa commande en arrière plan. Dans le cas où l'utilisateur souhaite lancer sa commande en premier plan, le programme attends la fin de l'exécution de cette commande via la fonction waitpid avec pour paramètres (pid,0,UNTRACED).

-Changer l'état du processus en premier plan :

Afin de gérer les signaux que l'utilisateur envoi au shell via les commandes Control+C et Control+Z.
Lorsque l'utilisateur effectue Control+C, alors la tâche exécutée en premier plan se termine.
Lorsque l'utilisateur effectue Control+Z, alors la tâche exécutée en premier plan se met en pause.
Pour réaliser cette étape, nous disposons d'une variable globale nommée fg nous permettant de connaître le pid du processus en premier plan.
Nous avons donc définis des handlers qui nous permettent de surcharger les traitants par défaut des signaux SIGINT(envoyé par un Control+C) et SIGTSTP (envoyé par un Control+Z). Ainsi, notre shell ne se ferme plus ni ne se met en pause lorsque l'utilisateur entre un Control+C ou un Control+Z.
Le programme principal traite les signaux SIGINT et SIGTSTP d'une manière particulière, là où les fils gardent le traitant par défaut.
Lorsque le programme principal reçoit un signal SIGINT, il transmet celui-ci au processus qui s'exécute en premier plan si il y en a effectivement un (via kill(pid,SIGINT)), ce qui aura pour effet de fermer ce processus.
La même méthode est utilisée pour le handler su signal SIGTSTP, à la différence qu'un traitement supplémentaire est effectué afin de mettre à jour la table des jobs (mettre le processus en premier plan dans la table des jobs et signifie que son état est en pause [STOPPED]).

-Gestion des zombis :

Pour éviter que les fichiers en arrière plan soient attendus par le miniShell, si nous savons qu'un processus doit s'exécuter en arrière plan(quand l'utilisateur le spécifie explicitement via le caractère clé '&') alors nous ajoutons ce processus dans la table des jobs afin de pouvoir le récupérer plus tard.
De ce fait, nous avons implémenté un traitant du signal SIGCHLD afin que lorsqu'un des processus fils meurt, nous puissions récupérer son pid afin de le retirer de la table des processus en arrière plan si il était effectivement en arrière plan ou de ne rien faire si il était en premier plan.
Pour cela, nous utilisons l'instruction waitpid(-1, NULL, WNOHANG)) afin de récupérer le fils qui est sur le point de se terminer, afin de ne pas générer de zombis.

-Commande intégrée : jobs :

Nous avons implémenté la commande jobs de la manière suivante : si l'utilisateur entre la commande "jobs" dans le miniShell, alors nous affichons la liste des processus en arrière plan, qu'ils soient en pause ou en cours d'exécution.
Grâce à la structure spécifiée dans le fichier jobs.h, nous gardons dans une table des jobs, en variable globale, toutes les informations nécessaires à leur affichage :
-PID.
-Etat courant (Pause ou actif).
-nom.

-Agir sur les commandes en arrière plan :

Lorsque l'utilisateur entre le mot clé « bg » suivi soit du pid du processus qu'il souhaite passer en

arrière plan, soit du numéro de job de ce processus, alors le programme envoi un signal SIGCONT à ce processus afin qu'il reprenne son exécution mais en arrière plan. Ceci change aussi son état courant dans la table des jobs (passe en RUNNING).

Lorsque l'utilisateur entre le mot clé « fg » seulement, alors le dernier processus qui était en premier plan, si celui ci a été basculé en arrière plan et est toujours en train de s'exécuter, repasse en premier plan.
Si l'utilisateur spécifie lui même le numéro du job qu'il souhaite faire passer en premier plan (les numéros de jobs sont consultables via la commande spéciale "jobs"), alors il repassera effectivement en premier plan.
Pour cela, le programme effectue un waitpid(pid, NULL, WUNTRACED), comme lorsque nous souhaitions lancer une commande en premier plan.

Lorsque l'utilisateur entre le mot clé « stop » suivi soit du pid du processus qu'il souhaite mettre en pause, soit du numéro de job de ce processus, alors le programme envoi un signal SIGSTOP à ce processus afin qu'il se mette en pause. Ceci change aussi son état courant dans la table des jobs (passe en STOPPED).

Tests réalisés :

Étape numéro 1 : Gestion de commandes jointes par des tubes :

Afin de s'assurer de la validation de cette étape, nous avons réalisé différents tests qui vont de l'exécution d'une commande simple, jusqu'à l'exécution de plusieurs commandes connectées via des tubes avec des redirection d'entrée et sorties.

Commande testée	Résultat attendu	Résultat correct ?
ls -l	Liste les fichiers présents dans le répertoire courant.	Correct
ls -l | wc -l	Compte le nombre de fichiers présents dans le répertoire courant.	Correct
ls – l | wc -l | wc -l	Compte le nombre de lignes (1) du résultat du comptage du nombre de fichiers présents dans le répertoire courant.	Correct
cat < toto	Affiche le contenu du fichier toto.	Correct
ls -l > tata	Écrit dans le fichier tata le résultat de la commande ls -l.	Correct
cat < titi > tutu	Écrit dans le fichier tutu le contenu du fichier titi.	Correct
cat | wc -l | wc -m < toto > tata	Écrit dans le fichier tata le nombre de caractères(+ 1 pour le caractère de fin de ligne) renvoyé par le comptage du nombre de lignes de l'affichage du fichier toto.	Correct

Étape numéro 2 : Pour aller plus loin :

Pour s'assurer de la validation de chacune des étapes, nous avons effectué les tests suivants :

-Exécution de commandes en arrière plan :

Il suffit de lancer une commande suivie du caractère spécial '&' afin de s'assurer que celle-ci se lance effectivement en arrière en plan : celle-ci est alors insérée dans la table des processus en arrière plan. De plus, on remarque que lorsque l'on lance une commande en premier plan, alors le prompt du miniShell ne revient pas et il est impossible de lancer une nouvelle commande avant que le processus lancé soit mis en pause ou terminé. Tandis que lorsque l'on souhaite lancer cette commande en premier plan, le prompt revient immédiatement et il est immédiatement possible de lancer une nouvelle commande, que ce soit en arrière plan ou en premier plan.
Pour s'assurer de la validation de cette étape, il est possible de lancer la commande « xterm & » afin de s'assurer que celle-ci se lance effectivement en arrière plan. De plus, si on lance la commande « xterm », alors on remarque bien que le terminal se lance bien en premier plan.
Le résultat attendu est donc correct.

-Changer l' état du processus en premier plan :

Pour s'assurer de la validation de cette étape, il suffit de lancer une commande en premier plan et d'effectuer un Control+C afin de voir que le processus lancé se termine bien (se ferme) ou d'effectuer un Control+Z afin de voir que le processus lancé se met bien en pause.
Dans tout les cas, l'utilisateur reprend la main sur le shell et peut entrer à nouveau de nouvelles commandes. De plus, l'utilisateur peut constater que le processus mis en pause est bien présent dans la tables des jobs en tapant la commande spéciale « jobs » : il verra bien que son processus est effectivement en pause.
Lancer une commande « xterm » faire un Control+C, s'assurer de la fermeture de celui-ci.
Lancer une commande « xterm » faire un Control+Z, s'assurer que celui-ci est effectivement en pause, lancer la commande jobs afin de constater que le processus est dans l'état « stoppé ».
Le résultat attendu est donc correct.

-Gestion des zombis :

Pour s'assurer de la validation de cette étape, le test à réaliser est le suivant :
Il faut lancer une commande en premier plan et fermer le processus lancé à l'aide d'un Control+C (voir l'étape précédente). Puis il suffit de lancer la commande ps afin de s'assurer que le processus précédemment lancé n'est effectivement plus en train de s'exécuter et n'est pas dans un état zombi.
De plus, il est aussi possible de lancer plusieurs processus en arrière plan et de quitter le minishell puis de lancer la commande ps afin de s'assurer qu'aucun processus n'est effectivement dans l'état zombi.
Le résultat attendu est donc correct.

-Commande intégrée : jobs :

Pour s'assurer de la validation de cette étape, le test à réaliser est le suivant :
Lancer plusieurs processus en arrière plan et s'assurer qu'ils sont tous effectivement présents dans la table des jobs via la commande spéciale « jobs ».
Il est possible d'en fermer quelque-uns et de relancer la commande spéciale jobs afin de constater qu'ils ne sont effectivement plus présents dans la table des jobs.
De plus, il est possible de connaître l'état courant du processus (« en cours d'exécution » ou « stoppé »).
Pour avoir un processus en état « stoppé », il suffit de lancer un processus en premier plan puis de faire un Control+Z. On constate que celui-ci est effectivement dans l'état « stoppé » après l'appel à « jobs ». Il est aussi possible d'utiliser la commande spéciale « stop » pour passer un job de l'état « en cours d'exécution » à l'état « stoppé », ou « bg » pour passer un job de l'état « stoppé » à l'état « en cours d'exécution ».
Toutes les modifications d'état peuvent être constatées via la commande spéciale jobs.
Le résultat attendu est donc correct.

-Agir sur les commandes en arrière plan :

Pour s'assurer de la validation de cette étape, les tests à réaliser est le suivant :
Lancer un processus en arrière plan (xterm par exemple), puis le stopper en utilisant la commande spéciale « stop » (+ numéro du job que l'on souhaite mettre en pause).
A partir de là, on peut constater que le processus est effectivement mis en pause et qu'il est impossible d'interagir avec lui. De plus, via la commande spéciale jobs, on peut constater qu'il est effectivement présent dans la table des jobs dans l'état « stoppé ».
Maintenant, on souhaite remettre ce processus dans l'état « en cours d'exécution ». Pour cela, on utilise la commande spéciale « bg » (+numéro du job que l'on souhaite mettre en arrière plan). On peut constater via la commande « jobs » qu'il est effectivement remis dans l'état « en cours d'exécution » et qu'il est à nouveau possible d'interagir avec le processus.
Enfin, nous allons passer ce processus en premier plan via la commande spéciale « fg » (+numéro du job que l'on souhaite mettre en premier plan). On remarque qu'il est désormais impossible de lancer une nouvelle commande car il y a un processus en premier plan.
Pour pousser le test un peu plus loin, il est possible de lancer un processus en premier plan, puis de faire un Control+Z afin de le mettre en pause, puis de lancer la commande spéciale « fg » sans argument, ce qui aura pour effet de remettre en premier plan le dernier processus en arrière plan qui a été en premier plan (sous réserve que ce processus soit toujours en cours d'exécution, qu'il soit en pause ou non).
Le résultat attendu est donc correct.