Master CCI

Introduction aux systèmes d'exploitation

Sujet de travail pratique "Triangle de Pascal"

La note de contrôle continu portera sur ce travail pratique qui a pour thème la génération et l'affichage d'un code Postscript qui dessine un triangle de Pascal.

La progression se fait par étapes adaptées au rythme de progression du cours et des travaux dirigés.

Etape	Prérequis
1	aucun (utilisation de makefile)
2	tp processus/fork/exec (exec)
3	tp processus/fork/exec (fork)
4	td API fichiers
5	td redirections et tubes
6	tp processus/fork/exec

Les deux séances de travaux pratiques encadrées planifiées pour ce sujet de TP suffiront peut-être pour réaliser les deux premières étapes, mais vous devrez probablement y consacrer du temps en dehors des heures encadrées.

Seuls les plus motivés pouvant y consacrer du temps viendront à bout de l'étape 6. Terminer l'étape 3 est un prérequis pour obtenir 10 au tp.

Recommandation importante : avant de passer à une étape suivante, faites une sauvegarde de tout votre travail (fichiers sources + fichiers de traces) avant de modifier vos fichiers pour traiter l'étape suivante.

1 Compilation et génération du fichier exécutable

Le tp est copnçu pour être effectué sur le serveur mandelbrot. Il est cependant probable qu'il fonctionne aussi sur un pc linux.

Extraire le contenu de l'archive triangle.tar.

```
mandelbrot> tar xvf triangle.tar
mandelbrot> cd  programme_triangle
mandelbrot> make
mandelbrot> make demo_gs
mandelbrot> make demo_gv
mandelbrot> make demo_creer
```

1.1 Compilation

Les paramètres de lancement du fichier exécutable triangle sont la taille de la police de caractères à utiliser pour représenter les nombres et la taille (nombre de lignes) du triangle à générer.

Lancez l'exécution du programme avec une police de taille 12 et 4 lignes. Le programme affiche une suite d'instructions en langage postscript. Ce langage est destiné aux imprimantes et traceurs. Il permet de décrire un document à afficher ou imprimer.

```
mandelbrot> ./triangle 12 4
```

1.2 Redirection de la sortie dans un fichier

La fonction C **printf** écrit sur la sortie standard. Par défaut la sortie standard est associée à l'écran du poste de travail ou (en mode graphique) à la fenêtre dans laquelle est lancée l'exécution du programme.

Le caractère "supérieur" (>) permet d'associer la sortie standard à un fichier, auquel cas les messages générés par le programme qui s'exécute seront stockés dans le fichier au lieu d'apparaître à l'écran.

Redirigez la sortie standard sur le fichier, et vérifiez ensuite (avec cat, more, less ou nedit) que ce fichier contient bien ce qui s'affichait auparavant à l'écran. Lancez ensuite l'interprète de langage postscript gv sur ce fichier.

```
mandelbrot> ./triangle 12 4 > triangle_pascal_12_4.ps
mandelbrot>  cat triangle_pascal_12_4.ps
mandelbrot>  gv triangle_pascal_12_4.ps
```

1.3 Redirection de l'entrée standard

La fonction C **scanf** lit sur l'entrée standard. Par défaut l'entrée standard est associée au clavier du poste de travail. Dans un environnement graphique, le clavier sert d'entrée standard à la fenêtre active.

Un filtre est un programme qui lit des caractères sur l'entrée standard et les réécrit sur la sortie standard après leur avoir appliqué un certain traitement. Par exemple, la commande **tr "bc" "BC"** lit des caractères au clavier et les réécrit à l'écran en mettant en majuscule toutes les lettres b et c.

Le caractère "inférieur" (<) permet d'associer l'entrée standard à un fichier, auquel cas les caractères lus par le programme qui s'exécute seront pris dans le fichier au lieu d'être lus au clavier.

Appliquez par exemple la commande au contenu du fichier Makefile :

```
mandelbrot>  tr "bc" "BC" < Makefile
```

1.4 Tubes et chaînage de commandes

Exécuter la suite de commandes suivante qui affiche la date deux fois. Dans le deuxième affichage, les heures, minutes et secondes sont séparées par un caractère souligné.

```
mandelbrot>  date
mandelbrot>  date > d
mandelbrot>  tr ":" "_" < d
```

On peut éviter de passer par un fichier en associant directement la sortie standard de la première commande à l'entrée standard de la deuxième commande par un tube (pipe en anglais). La syntaxe de création d'un tube est la barre verticale.

```
mandelbrot>  date | tr ":" "_"
```

On peut chaîner le programme de génération de triangle et un interprète de langage Postscript (gs). Ce dernier lit alors directement les messages générés par le programme triangle. La commande gv lit également l'entrée standard lorsqu'on lui passe "-" comme nom de fichier :

```
mandelbrot>  ./triangle 11 5 | gs -sDEVICE=x11 -q
mandelbrot>  ./triangle 11 5 | gv -
```

2 Lancement de l'interprète Postscript par exec

La commande **which** permet de déterminer dans quel répertoire[1] se trouve le fichier exécutable d'une commande. Vérifier dans quel(s) répertoire(s) se trouvent les interprètes Postscritpt **gv** et **gs**. Dans la suite, nous supposerons qu'ils sont dans /usr/bin.

```
mandelbrot> which gv
mandelbrot> which gs
```

Avec la redirection, on peut afficher le triangle des deux manières suivantes (il faut saisir la commande quit dans la fenêtre terminal pour terminer l'exécution de gs) :

```
mandelbrot> ./triangle 12 4 > triangle.ps
mandelbrot>  /usr/bin/gv triangle.ps
mandelbrot>  /usr/bin/gs -q -sDEVICE=x11 triangle.ps
GS> quit
```

1. du chemin d'accès aux commandes défini par la variable d'environnement **PATH**

On veut maintenant que la première commande exécute automatiquement **gv** ou **gs** sur le fichier triangle.ps pour afficher le résultat.

```
mandelbrot> ./triangle 12 4 > triangle.ps
```

Pour celà, il suffit d'utiliser la primitive exec à la fin de la procédure main pour lancer gv. Inspirez-vous de l'exemple d'utilisation de exec du tp sur les processus (notament dans forkecho, forkps).

Modifier le code, recompiler, tester (ne pas oublier de taper quit pour terminer l'exécution de gs ou q pour terminer celle de gv).

Attention : Pensez à la sortie standard d'erreur (qui n'est pas redirigée) pour vos messages de trace (sinon ils iront se mélanger aux ordres postscript dans le fichier triangle.ps).

3 Lancement de gs par fork et exec

Avec la commande ci-dessous, on veut maintenant que le programme genère le Postscript, lance l'exécution de l'interprète **gs**, puis écrive un message "Génération et affichage du triangle de Pascal terminés".

```
mandelbrot> ./triangle 12 4 > triangle.ps
Génération et affichage du triangle de Pascal terminés
mandelbrot>
```

Modifier le code de affiche_triangle.c, recompiler, tester. Il faut que le programme

1. **crée** un processus fils par **fork** pour exécuter **gs**
2. attende sa terminaison par l'appel **wait**
3. affiche enfin le message de terminaison.

La commande **killall** *nom* recherche et tue tous les processus qui vous appartiennent et exécutent *nom*. Elle est particulièrement utile en cas d'erreur se traduisant par une création récursive de processus : le temps de trouver le numéro du processus et de le tuer, ce dernier a eu le temps de créer un autre fils (qui va en créer à son tour un autre ...).

4 Génération du Postscript dans un fichier

Regénérer le fichier triangle.ps par une redirection et observer sa taille (notons T1 cette taille).

```
mandelbrot> ./triangle 10 8 > triangle.ps
mandelbrot>  gv triangle.ps
mandelbrot> ls -l triangle.ps
```

Dans boite_chiffre.c, le code postcript peut être écrit :

— sur un fichier (à ouvrir puis refermer), dont le nom est contenu dans le tableau sortie ou

— sur la sortie standard préouverte, si le nom est "stdout"

Dans l'affectation de sortie (procédure main), remplacer "stdout" par "triangle.ps" de telle sorte que le programme écrive directement dans le fichier triangle.ps sans que l'on ait à rediriger la sortie lors du lancement de l'exécution.

```
mandelbrot>  /bin/rm -f triangle.ps
mandelbrot> ./triangle 10 8
mandelbrot> ls -l triangle.ps
```

Quelle est maintenant la taille (T2) du fichier triangle.ps ?
Si le programme n'affiche qu'une seule case, déterminer pourquoi et corriger le probleme.

Vous aurez probablement créé un nouveau problème observable en lançant une suite d'exécutions : faire varier la taille de police et observer l'évolution de la taille du fichier postscript.

Après correction du problème, la taille doit rester égale à T1 et l'affichage correct.

5 Création d'un tube entre triangle et gs

Repartez de l'étape 3 dans laquelle le programme triangle écrit le Posctscript sur sa sortie standard. Modifier le programme de telle sorte que le processus père :

1. Crée un tube
2. Crée un processus fils (gauche) qui associe l'entrée du tube à sa sortie standard et appelle la procédure postscript_triangle pour genérer le postscript.
3. Crée un processus fils (droit) qui associe la sortie du tube à son entrée standard et utilise la primitive **exec** pour exécuter gs.
4. Attend la terminaison des deux fils et affiche un message de fin.

A présent, la commande ci-dessous doit afficher le triangle.

```
mandelbrot> ./triangle 12 4
```

Pour vérifier le fonctionnement correct du tube, vous pouvez lancer /bin/cat à la place d'un interprète de postcript : le postcript devrait s'afficher à l'écran.

Le nom de fichier à passer à gv pour qu'il lise l'entrée standard semble être le caractère moins (-), mais je n'en ai trouvé confirmation nulle part dans la documentaztion de gv ...

Attention : l'utilisation de deux appels à fork sans précaution aboutit généralement à la création de trois nouveaux processus au lieu de deux, le processus fils issu du premier fork exécutant lui aussi le deuxième fork.

Voici un squelette correct de création de deux processus fils :

```
gauche = fork ();
if (gauche < 0) /* erreur */
if (gauche == 0) {
   code_gauche ();
   exit (1);  /* empecher le fils gauche de continuer avec 2eme fork */
   }
/* seul le pere continue ici */
droit = fork ();
if (droit <0) /* erreur */
if (droit == 0) {
   code_droit ();
   exit (1);  /* empecher le fils droit de continuer avec code_pere */
   }
/* Seul le pere continue ici */
code_pere ();
```

6 Utilisation d'un processus par boîte

L'exécutable **creer_boite** permet d'appeler la procédure boîte_chiffre en passant les paramètres sous forme de chaînes de caractères de la ligne de commande.

Essayer de créer une case de triangle avec creer_boite :

```
mandelbrot> ./creer_boite 12 stdout 24 110 14 15 1 1 > triangle.ps
mandelbrot> gv triangle.ps
```

Un mécanisme de trace des appels à boite_chiffre est inclus dans le fichier affiche_triangle.c. Pour l'activer, ajouter la définition de la macro TRACE_BOITE dans le fichier affiche_triangle.h et recompiler. Avec csh ou tcsh, la redirection par >& redirige les deux sorties (standard et standard d'erreur). La redirection par > ne redirige que la sortie standard.

Exécuter le programme triangle en ne redirigeant que la sortie standard dans le fichier triangle.ps. Les messages de trace apparaissent à l'écran. Effacer le fichier triangle.ps.

Avec un copier/coller à la souris de ces messages, exécuter les commandes de lancement de creer_boite pour créer toutes les cases du triangle, puis vérifier que le postscript du triangle complet est bien créé.

```
mandelbrot> ./triangle 12 3 > triangle.ps
mandelbrot> gv triangle.ps
mandelbrot> rm triangle.ps
mandelbrot> ./creer_boite 12 stdout ...   >> triangle.ps
mandelbrot>  ....
mandelbrot> ./creer_boite 12 stdout ...   >> triangle.ps
mandelbrot> gv triangle.ps
```

Modifier maintenant le programme de telle sorte que pour chaque case du triangle un processus fils soit créé pour exécuter le fichier binaire exécutable creer_boite , le programme principal attendant la fin de tous les fils créateurs de cases pour lancer l'exécution de gs. Afficher les numéros de tous les processus crées pour la génération du triangle.