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 client/serveur réseau
7	td création de threads

L'utilisation des threads sera vue ou pas selon l'avancement en td. L'étape 7 qui en dépend est destinée aux plus motivés qui souhaitent aller un peu plus loin.

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.

A rendre sur caseine au plus tard le jour de l'examen : une archive tar ou zip contenant les sources la dernière étape traitée et un compte-rendu des étapes 2 et suivantes détaillant le problème éventuel et l'extrait de code modifié ou ajouté.

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
```

1.1 Compilation

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

Lancez l'exécution du programme avec une police de taille 10 et 4 lignes. Le programme affiche une suite d'instructions en langage postscript.

```
mandelbrot> ./triangle 4 10 -
```

Le dernier paramètre (-) indique que le postscript doit être généré sur la sortie standard.

Ce langage postscript est destiné aux imprimantes et traceurs. Il permet de décrire un document à afficher ou imprimer. Le format pdf (format portable de documents) consiste esentiellement en une forme compressée de postscript.

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 4 12 - > triangle_pascal_4_12.ps
mandelbrot>  cat triangle_pascal_4_12.ps
mandelbrot>  gv triangle_pascal_4_12.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 4 12 - | gs -sDEVICE=x11 -q
mandelbrot>  ./triangle 4 12 - | 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 Postscript **gv** et **gs**. Dans la suite, nous supposerons qu'ils sont dans /usr/bin.

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

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

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

```
mandelbrot> ./triangle 4 12 - > 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 q pour terminer l'exécution 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 4 12 - > 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 4 12 - > triangle.ps
mandelbrot>  gv triangle.ps
mandelbrot> ls -l triangle.ps
```

Lorsqu'on passe en troisième paramètre un nom de fichier (autre que -), le programme triangle écrit directement dans celui-ci.

```
mandelbrot>  /bin/rm -f triangle.ps
mandelbrot> ./triangle 4 12 triangle.ps
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 (fichiers main.c et write_file.c).

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.

```
mandelbrot> ./triangle 5 10 triangle.ps
mandelbrot>  gv triangle.ps
mandelbrot> ls -l triangle.ps
mandelbrot> ./triangle 6 8 triangle.ps
mandelbrot>  gv triangle.ps
mandelbrot> ls -l triangle.ps
```

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 documentation 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.

6 Générateur de triangle en serveur html

6.1 Découverte (observation uniquement)

L'exécutable serveur attend une requête HTTP (GET) sur le port local 8080, affiche le message de requête du client, répond en envoyant le html d'un triangle de Pascal et se termine.

Dans votre fenêtre terminal, lancer le serveur de triangle : **./triangle** puis dans votre navigateur firefox saisir l'URL

```
localhost:8080
```

un triangle de Pascal doit s'afficher. Dans la fenêtre terminal, le serveur doit afficher le message de requête reçu du client. Noter le / entre GET et HTTP.

Relancer le serveur dans votre fenêtre terminal. Ouvrir une autre fenêtre terminal et dans celle-ci saisir la commande suivante :

```
wget localhost:8080
```

Wget stocke la réponse du serveur dans le fichier index.html, dont vous pouvez observer le contenu avec votre éditeur de fichiers préféré (par exemple nedit ou gedit). Vous pouvez aussi l'afficher avec la commande suivante :

```
firefox file:index.html
```

dans le terminal dans lequel vous avez lancé wget.

6.2 Requête paramétrée

La syntaxe d'une URL est généralement la suivante :

```
nom_hote:port/param
```

le port étant optionel et param un paramètre de nature quelconque, par exemple un entier (dans ce tp la taille du triangle à tracer) ou un nom de fichier.

Relancer le serveur et s'y connecter via l'URL

```
localhost:8080/12
```

et observer que le dans le début du message du client, entre GET et HTTP, / est devenu /12.

Modifier le code de create_answer dans le fichier server.c pour dessiner un triangle de x lignes en réponse à l'URL de la forme

```
localhost:8080/x
```

Recompiler et tester.

7 Etape 7 : parallélisation de génération du postscript

Paralléliser compute_postscript en créant autant de threads que de cases du tableau à traiter (ne pas espérer gagner en temps d'exécution : la création de thread est probablement au moins aussi coûteuse qu'un calcul de postscript d'une cellule).

Ne pas oublier d'attendre leurs terminaisons dans le corps de compute_postscript.

Chaque case du tableau contient déjà un champ permettant de stocker l'id du thread qui traite la case.