TP n°3 – processus

Ce TP consiste à comprendre ce que sont les processus et comment on les gère pour optimiser l'utilisation de l'ordinateur en mode ligne de commande. Ensuite, il y a le concept universel de redirection des entrées et sorties des commandes.

1) Processus en arrière-plan

Ce TP propose d'utiliser un logiciel de synthèse d'images appelé `povray` (Persistence of Vision Ray Tracer). C'est une sorte de compilateur : il fabrique une image à partir d'un fichier source contenant la description d'une scène dans une syntaxe particulière. La scène est spécifiée dans un fichier d'extension .pov et l'image est produite dans le format .png.

NB : ce logiciel a été choisi parce qu'il fait des calculs pendant assez longtemps, il se prête bien aux manipulations des processus. On peut aussi l'utiliser pour les loisirs, mais ce n'est pas notre but.

a) installation du logiciel (à ne faire qu'une fois par séance)

Mais auparavant, pour pouvoir utiliser `povray` chez vous, voici ce qu'il faut taper (une fois seulement par connexion) :

`cd /usr/loc*/ano*/SYS/povray-3.6` remarquer les jokers pour taper moins de choses

`source installpov.bash` cela permet de positionner une variable nécessaire pour ce tp, c'est dû au fait que povray n'est pas vraiment « installé » dans le système.

Puis recopier les fichiers `*.pov` chez vous (dans `~/systeme/tp3`). Revenez chez vous.

b) lancement du logiciel

Pour calculer une image 2048x1536 : `povray isocacti.pov -w2048 -h1536`

Voici la commande pour afficher l'image : `display isocacti.png`

Ce qui nous intéresse ici, c'est le temps que prennent les calculs graphiques (et non pas l'image obtenue). Le calcul de l'image proposée peut durer largement plus d'un quart d'heure en fonction de ce que font les autres utilisateurs au même moment. Pour ne pas rester bloqué, il faut apprendre à arrêter le programme ou à le lancer en arrière-plan, et c'est le but de ce TP.

c) Interruption et relance du programme au premier-plan

Relisez le cours d'amphi partie 3, sur les processus et les jobs.

Lancer `povray` comme indiqué plus haut et immédiatement après essayez de taper des commandes : `ls`, `ps`, `date`... Constatez que vous n'avez pas la main : les commandes ne sont pas exécutées même si on les voit à l'écran. Noter que povray affiche beaucoup de messages qui se superposent à vos touches.

Relancez `povray` sur un fichier qui demande beaucoup de temps de calcul, comme isocacti.pov. Appuyez sur les touches CTRL et C en même temps (on les note `^C`). Ça termine immédiatement les calculs. Le programme est fini de force, mais sans finir son travail. il faudra reprendre à zéro. Vous avez à nouveau la main dans le shell.

Relancez `povray` puis appuyez sur les touches CTRL et Z en même temps (`^Z`). Ça fige le programme, ce qui vous redonne la main dans le shell et permet de poursuivre les calculs quand on voudra. Tapez la commande `ps` pour observer la colonne `TIME`, elle indique le temps (minutes:secondes) déjà consacré au processus (remarque, vous verrez que le programme s'appelle en fait x-povray). Tapez la commande `jobs` et voyez que votre programme est figé (*stopped*). Tapez la commande `fg` et le processus repart (de là où il en était). Vous pouvez retaper sur `^Z` puis à nouveau lancer la commande `ps` pour voir combien de temps le CPU lui a consacré puis `fg` pour qu'il reparte et ainsi de suite.

d) passage premier-plan vers arrière-plan

Comme précédemment, lancez `povray` puis appuyez sur les touches CTRL et Z en même temps (`^Z`). Ensuite tapez la commande `bg`. Le programme va repartir mais en arrière-plan. Constatez que vous pouvez taper des commandes (ls, ps), mais quasiment en aveugle car vos touches se mélangent aux affichages de povray. Le shell et povray se partagent l'écran et tous les affichages sont entrecroisés.

Utiliser toutes les deux secondes la commande `ps` pour observer la colonne `TIME`. Le nombre de secondes augmente peu à peu, si ce n'est pas le cas, c'est que le processus est stoppé.

e) lancement direct en arrière-plan

Taper `povray isocacti.pov -w2048 -h1536 &` Constatez que vous avez la main, vous pouvez taper des commandes : par exemple `ls -l` pour voir la taille du fichier `isocacti.png`. Par contre, remarquez au début que le prompt a été mélangé aux messages de povray, il apparaît si vous tapez entrée. Vous voyez aussi que les résultats de vos commandes sont mélangés à ceux de povray. Tapez plusieurs fois `ps` et `ls -l` pour constater la progression des calculs. Le but du TP3 consiste à lancer povray sans qu'il ne gène votre travail par la suite, voir le §4.

f) tentative de lecture clavier en arrière-plan

Taper `povray &` (lancement direct en arrière-plan et sans aucun paramètre). Constater 1) qu'il semble demander un nom de fichier, 2) qu'il se retrouve en réalité bloqué (*stopped*), 3) si vous tapez un nom de fichier, ça fait une erreur. Que s'est-il passé ? Analyser l'affichage : on voit à la fois le prompt et la demande de paramètres de povray. Lequel a la main ? Réfléchissez bien à ce qui s'est passé.

Faire passer ce job au premier plan en tapant `fg` et lui donner le nom qu'il attend (notez qu'il ne ré-affiche pas son message, il reste simplement à attendre). Le replacer immédiatement en arrière-plan pour qu'il achève son travail sans bloquer le clavier. L'observer avec `ps`.

g) suppression d'un processus

Lancer `povray` avec tous les paramètres et au premier plan (pas de &). Le supprimer avec `^C`.

Lancer `povray` avec tous les paramètres et en arrière-plan. Le supprimer en le tuant : repérer son `PID` avec la commande `ps` et utiliser la commande `kill`. Lancer plusieurs processus `povray` en arrière-plan et les visualiser, tuer ceux qui n'ont pas fini au bout d'une minute.

h) Question : à quoi sert la mise en arrière-plan ?

- Pouvoir lancer plusieurs programmes de calcul en même temps.

- Pouvoir travailler avec emacs et le compilateur en même temps : lancer emacs en arrière-plan, modifier le fichier, enregistrer sans quitter l'éditeur, compiler et tester dans la fenêtre xterm.

- Pouvoir supprimer un processus qui boucle indéfiniment, sans devoir se reconnecter : taper ^Z pour figer puis détruire par `kill -9` le processus en question.

2) Redirection des entrées et des sorties d'un processus

Relisez le cours d'amphi concernant les redirections (partie 4).

Les exercices suivants proposent de tester la même commande sans redirection puis avec redirection.

a) redirection de la sortie standard avec écrasement `> nomfichier`

Essayer les commandes suivantes (une case = une commande à taper) en **analysant** les résultats (à quel moment le contenu du fichier texte change-t-il et comment ? Décrivez ce qui se passe en une phrase sur votre feuille.

`echo bonjour`	`more texte`	`echo bonjour >texte`	`more texte`
`ls -l`	`more texte`	`ls -l >texte`	`more texte`
`expr 2 + 3`	`more texte`	`expr 2 + 3 >texte`	`more texte`
`pwd`	`more texte`	`pwd >texte`	`more texte`
`whoami`	`more texte`	`whoami >texte`	`more texte`
`cal`	`more texte`	`cal >texte`	`more texte`

En conclure : une redirection de sortie > renvoie les affichages de la commande dans le fichier qui est préalablement effacé ou créé par cette opération. Presque toutes les commandes peuvent être redirigées.

b) redirection de la sortie standard avec concaténation `>> nomfichier`

Taper les commandes suivantes (une case = une ligne à taper) et pour chacune, à chaque étape, contrôler le contenu du fichier:

`cal 9 2015 > texte`	`cal 10 2015 >> texte`	`date >> texte`
`echo ligne 1 > texte`	`echo ligne 2 >> texte`	`echo ligne 3 >> texte`

`ls -l > texte`	`file * >> texte`	`wc texte >> texte`

En conclure : une redirection avec concaténation >> ajoute tous les affichages à la fin du fichier.

c) *redirection de toutes les sorties* `&> nomfichier`

Cette redirection redirige tous les messages d'erreur des commandes ainsi que les affichages normaux. Taper les séquences suivantes (une case = une ligne à taper) qui provoquent toutes une erreur (volontaire) :

`date -t`	`date -t > texte`	`date -t &> texte`
`lol`	`lol > texte`	`lol &> texte`
`cd dd`	`cd dd > texte`	`cd dd &> texte`
`cc`	`cc > texte`	`cc &> texte`

d) *redirection de l'entrée* `< nomfichier`

Avec un éditeur, créez le fichier **`data`** avec ces 4 lignes, attention ce sont des p, pas des P

```
2 3 + p
8 5 6 * - p
75 5 - 7 / p
q
```

Ensuite lancer les commandes suivantes(une case = une ligne à taper) et surtout, essayez de comprendre ce qui résulte à l'écran (posez des questions, lisez le début de la doc histoire d'avoir une idée) :

`wc < data`	`dc < data`	`tail -n 2 < data`
`fmt -w 16 < data`	`cat -n < data`	`shuf < data`
`column -t < data`	`tr '+p5' '@=#' < data`	`sed 's/ /_/g' < data`

Consultez la documentation en ligne pour les commandes que vous ne connaissez pas, ou demandez des explications à votre enseignant.

Que pensez-vous de `who < data` ? Est-ce que c'est utile de rediriger l'entrée de cette commande ?

3) Qu'est-ce finalement qu'une redirection ?

Cela fait-il une différence pour une commande d'être redirigée ou non ? Sans redirection, elle écrit sur l'écran ; avec redirection, elle écrit dans un fichier. On va voir que c'est pareil du point de vue de la commande.

Ouvrez une seconde fenêtre « terminal ».

Tapez la commande `tty`. Elle vous affiche un nom complet, par exemple /dev/pts/13 (chaque fenêtre « terminal » possède son propre numéro, par exemple pts/0 et pts/1).

Tapez `who` : Le résultat sort à l'écran.

Tapez `who > liste` : le résultat est écrit dans le fichier liste qui est créé s'il n'existe pas.

Tapez `who > /dev/pts/`*`n°`* (c'est à dire le nom complet indiqué par `tty`) : le résultat est arrivé à l'écran. Que comprendre à part qu'en fait, l'écran, c'est ce fichier /dev/pts/n°. Ce fichier ne correspond pas à un stockage sur disque dur. Il s'agit d'un fichier spécial : tout ce qu'on écrit dedans est affiché directement sur l'écran, et si vous tentez d'afficher son contenu, c'est tout ce que vous tapez au clavier que vous allez voir (^C pour arrêter).

En somme quand on lance une commande, par défaut, sa sortie stdout est redirigée vers /dev/pts/n° qui est vu comme un fichier par la commande.

Question : et si on écrivait sur un autre /dev/pts/n° que celui que tty nous envoie ? Remarquez que la commande who affiche le pts/n° correspondant à chaque fenêtre terminal....

4) Arrière-plan et redirections sur povray

L'exercice 1)e a montré que la commande lancée en arrière-plan affichait ses informations à l'écran par dessus les commandes tapées en direct. Il s'agit maintenant d'utiliser des fichiers pour fournir et recevoir les informations en combinant redirections et arrière-plan.

a) redirection des sorties

Lancer `povray` en arrière-plan avec une redirection de sa sortie normale sur un fichier (en tapant `povray arches > stats &`). Constater que quelques infos sont mises dans le fichier mais qu'il reste encore des affichages à l'écran. Rediriger aussi les « erreurs » (`&> stats`) pour tout envoyer dans le même fichier.

b) redirection de l'entrée

Créer avec `vi` un nouveau fichier (par exemple appelé `params`) contenant une seule ligne : le nom du fichier d'image et les options (`arches.pov -w2048 -h1536`), c'est à dire exactement ce qu'il faut taper au clavier quand on lance povray sans paramètres. Lancer `povray` en arrière-plan avec une redirection de son entrée sur ce fichier `params`. Constater que la demande de nom de fichier apparaît quand même mais povray n'attend plus de réponse, il va la chercher dans le fichier `params`. Faire de même avec en plus une redirection des sorties de façon que la commande n'utilise plus du tout le terminal. Qu'en pensez-vous, est-ce pratique ou pas ?

c) lancement et déconnexion

Lancer `povray` en arrière-plan avec redirection des entrées et des sorties comme précédemment avec une taille d'image plutôt importante (2048x1536) afin qu'il dure longtemps. Normalement, si vous vous déconnectez, ce jobs est supprimé. Il y a moyen de le faire continuer en votre absence. Il suffit de connaître son PID et de taper `disown` *`PID`* (mettez le nombre en paramètre) avant de vous déconnecter.

Ensuite, déconnectez-vous en fermant la fenêtre PuTTY ou avec `logout`. Reconnectez-vous, vous vérifierez avec `ps -fu` *`votrelogin`* que le processus a continué ses calculs, par contre, il n'est plus listé avec la commande `jobs` : il a été détaché de votre ancien terminal. Il tournera jusqu'à la fin des calculs ou sera supprimé par le système s'il dure beaucoup trop longtemps.

Rappel : si vous voulez relancer povray, il faut le réinstaller à chaque connexion, cf 1)a.

5) Redirections sur un programme C

Taper (copiez-collez) le programme `nbrinf10.c` suivant :

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
     int compte = 0;
     int valeur = 0;

     printf("Veuillez saisir des entiers :\n");
     while ( scanf("%d", &valeur) == 1 ) {
          if (valeur < 10) {
               compte = compte + 1;
          }
     }
     printf("nbrinf10: il y a eu %d nombres < 10\n", compte);

     return EXIT_SUCCESS;
}
```

RQ : si vous copiez-collez ce texte, alors revérifiez les signes + ' " et - car mon traitement de texte emploie des codes qui ne sont pas les bons pour le compilateur (ex : différence entre ' et ’)

Compilez et testez-le avec quelques valeurs. Notez qu'il s'arrête dès qu'on saisit autre chose qu'un entier et qu'il affiche alors le nombre d'entiers plus petits que 10 qui ont été tapés.

Maintenant, créez un fichier texte contenant des entiers, un par ligne ou côte à côte, et fournissez-le en entrée du programme : `nbrinf10 < maliste`. Vous pouvez vérifier qu'il compte les entiers plus petits que 10 du fichier. On voit quand même le message du printf car la sortie n'est pas redirigée.

Essayez aussi la commande `maliste > nbrinf10`. Que s'est-il passé ? Est-ce que votre programme `nbrinf10` marche encore ? Sinon, ouf ! vous avez seulement à le recompiler... Retenez bien la leçon : le nom du programme doit toujours être en premier et les signes **<** ou **>** avant les noms des fichiers de données.

Voici un second programme `aleat.c` :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
     int nb = 0;
     fprintf(stderr, "Nombre d'entiers à générer :\n");
     scanf("%d", &nb);       // combien de nombres faut-il produire ?
     srand(getpid());        // initialisation du générateur aléatoire
     while (nb > 0) {        // produire nb nombres
          printf("%d\n", rand()%100);
          nb = nb - 1;
     }
     printf("fini\n");       // sentinelle pour signaler la fin

     return EXIT_SUCCESS;
}
```

Compilez pour obtenir l'exécutable `aleat`. Il faut taper un nombre positif au tout début, et il va afficher autant de nombres aléatoires que demandé, par exemple tapez `aleat` puis 10 et il affichera dix nombres aléatoires. Faire `man 3 rand` pour avoir de la documentation si ça vous intéresse.

L'idée est de faire produire un fichier de nombres par `aleat` et de le fournir à `nbrinf10`. Question : est-il nécessaire qu' `aleat` affiche le mot fini à la fin de la liste ? Est-ce que la fonction aléatoire fournit bien environ N/10 nombres plus petits que 10 ?

Maintenant, en option mais également intéressant : pouvez-vous écrire un programme `somme.c`, en adaptant `nbrinf10.c`, qui fait la somme des nombres qu'on lui entre : sa boucle consiste à lire un nombre qu'on additionne avec le total précédent. Couplez `aleat` avec votre programme afin de calculer la somme de 10 nombres aléatoires. Constatez que cette somme est voisine de N*49,5 quand N est très grand. Pourquoi ?

On peut imaginer d'autres programmes C faciles à écrire et dont la combinaison réalise des choses puissantes. Ces exemples montrent la généralité et la simplicité des redirections Unix. Tout programme qui utilise le clavier et l'écran peut être redirigé vers des fichiers.