

TRAVAUX PRATIQUES 2

Chaîne de compilation et Processus sous Linux

L'objectif de ce TP est de revenir sur les outils et fonctions proposées par un système d'exploitation de type UNIX pour la compilation et l'exécution de programmes.

Une fois votre session ouverte sous un système d'exploitation de type UNIX, ouvrez un terminal pour entrer les commandes décrites dans l'énoncé. Dans la suite, le prompt du terminal sera symbolisé par « \$> ».

Vous aurez à utiliser plusieurs fichiers tout au long de ce TP qu'il faudra télécharger à partir d'une adresse qui vous sera indiquée en séance.

L'exécution des programmes est manipulée par le système d'exploitation à travers la notion de processus. Nous rappelons qu'un processus est la représentation de la dynamique d'exécution d'un programme (ou partie d'un programme si celui-ci créé durant son exécution plusieurs autres processus).

EXERCICE 1

Le système d'exploitation ne peut exécuter que des fichiers binaires contenant du code machine (c'est-à-dire un ensemble d'instructions compréhensibles par le processeur de la machine). Ainsi avant d'exécuter un programme, il faut au préalable générer le code machine à destination de l'architecture processeur de la machine utilisée. Pour cela, nous devons utiliser un compilateur qui a la charge de traduire les codes sources d'un programme (développé à l'aide d'un langage haut niveau comme C/C++, Java, ...) en du code machine comprise par le processeur.

Dans la suite, vous utiliserez le compilateur **gcc** (pour GNU Compiler Collection), qui fait partie de l'ensemble des logiciels libres. Pour répondre aux questions ci-dessous, vous devrez utiliser les fichiers `prog_ex01.c` et `lib_func.c`. Le premier contient le programme principale tandis que le second contient le code de fonctions supplémentaires appelées dans le fichier `prog_ex01.c`.

Question 1 – Utilisez le compilateur **gcc** et donnez la commande pour générer le code objet associé au fichier `prog_ex01.c`. Qu'obtenez-vous comme résultat ?

Correction : Il faut utiliser commande suivante `$> gcc -c prog_ex01.c`

Le fichier `prog_ex01.o` est généré dans le même répertoire que `prog_ex01.c`

Question 2 – Toujours à l'aide du compilateur **gcc**, donnez la commande pour générer le code exécutable associé au fichier `prog_ex01.c`.

Qu'obtenez-vous et expliquez ce que cela signifie ?

Correction : Il faut utiliser commande suivante

```
$> gcc -o prog_exo1 prog_exo1.o
```

On obtient une erreur similaire à celle-ci-dessous :

```
prog_exo1.o: In function `main':
prog_exo1.c:(.text+0x1c): undefined reference to `fonction_fils'
collect2: error: ld returned 1 exit status
```

Le problème vient du fait que la référence de la fonction « fonction_fils » n'a pas été trouvée par le compilateur gcc alors que celle-ci est utilisée dans le programme prog_exo1.c.

Question 3 – Indiquez ce qu'il faut faire et les commandes à taper pour corriger cela.

Correction : Il est nécessaire pour corriger l'erreur de compiler et ajouter le fichier lib_func.c dans la liste des fichiers objets que gcc doit utiliser pour générer l'exécutable.

Pour cela il faut exécuter les commandes suivantes :

```
$> gcc -c prog_exo1.c
$> gcc -c lib_func.c
$> gcc -o prog_exo1 prog_exo1.o lib_func.o
```

Le compilateur **gcc** dispose d'un grand nombre d'options afin de paramétrer son utilisation.

Il est par exemple possible d'indiquer le nom de bibliothèques partagées afin que l'éditeur de lien puisse les inclure lors de l'édition. Pour cela, vous pouvez utiliser l'option `-l<nom_lib>` et l'éditeur cherchera la bibliothèque `<nom_lib>` parmi les répertoires du système contenant les bibliothèques.

Une autre option très utilisée concerne l'affichage de mises en garde (*warning*, en anglais) sur l'écriture du programme. Pour cela, vous pouvez utiliser l'option `-Wall` (à utiliser comme suit `gcc -Wall -o prog prog.o`) qui demande au compilateur d'afficher toutes les mises en garde.

Question 4 – Essayez de recompiler les fichiers sources. Dites si certaines mises en garde sont indiquées et si oui lesquelles.

Correction : Il y a une mise en garde lors de la compilation du fichier source lib_func.c. En effet une variable a été déclarée dans le programme sans être utilisée dans celui-ci comme indiqué ci-dessous :

```
$> gcc -Wall -c lib_func.c
lib_func.c: In function `fonction_fils':
lib_func.c:7:10: warning: unused variable `k' [-Wunused-variable]
```

Pour demander au système d'exploitation d'exécuter le programme exécutable obtenu associé aux fichiers `prog_exo1.c` et `lib_func.c`, il faut utiliser la syntaxe ci-dessous à partir du répertoire contenant l'exécutable `prog_exo` : `$> ./prog_exo1`

Si le programme exécutable se trouve dans un autre répertoire, il faut indiquer le chemin d'accès vers l'exécutable après les symboles `./`.

Question 5 – Exécutez le programme exécutable comme indiqué ci-dessus. Qu'est ce qui s'affiche dans le terminal ?

Correction : On voit des messages affichés venant des processus père et fils.

Question 6 – On peut observer qu'il n'est plus possible d'entrer de nouvelles commandes dans le terminal avant que le programme lancé n'ait terminé son exécution. Expliquez pourquoi.

Correction : Cela vient du fait que le programme a été exécuté au premier plan. Le terminal attend que le programme termine son exécution avant d'autoriser l'utilisateur à entrer une nouvelle commande.

Il est possible de demander à tout moment au terminal d'arrêter l'exécution du programme lancé au premier plan, cela est réalisé à l'aide de la combinaison de touches <CTRL>+c (appui des touches CTRL et c simultanément).

Question 7 – Essayez d'exécuter à nouveau le programme `prog_exo1` et arrêtez-le durant son exécution.

Il est également possible de demander au terminal de passer l'exécution du programme lancé en tâche de fond afin de pouvoir entrer de nouvelles commandes. Pour cela, il faut taper la combinaison de touches <CTRL>+z le terminal arrête temporairement la commande qui avait été lancée, puis il faut entrer la commande `bg` (pour *background*, en anglais) permettant de placer la dernière commande stoppée en tâche de fond.

La commande `fg` (pour *foreground*, en anglais) peut être utilisée et a l'effet contraire de la commande `bg`.

Un programme peut être exécuté directement en tâche de fond avec le symbole `&` comme indiqué ci-dessous : `$> ./prog_exo1&`

Question 8 – Essayez d'exécuter à nouveau le programme `prog_exo1` directement en tâche de fond.

Correction : Il faut utiliser commande suivante `$> ./prog_exo1&`

EXERCICE 2

Dans cet exercice nous allons nous intéresser à la visualisation d'informations fournies par le système sur l'exécution des programmes en cours d'exécution (c'est-à-dire des processus).

Nous rappelons que le système maintient une arborescence des processus en présents dans le système et chaque processus est caractérisé par plusieurs informations listées ci-dessous :

- ❖ **utilisateur (UID)** : nom ou identifiant de l'utilisateur ayant lancé le programme,
- ❖ **pid (Processus Identifier)** : correspond à l'entier attribué par le système à un processus pour l'identifier,

- ❖ **ppid (Parent Processus Identifier)** : correspond à l'identifiant du processus parent ayant engendré le processus en question,
- ❖ **état (S, STA ou STATE)** : définit par l'ordonnanceur du système, plusieurs valeurs sont possibles :
 - S pour *Stopped* si le processus est en sommeil,
 - R pour *Running* si le processus est en exécution ou prêt à s'exécuter,
 - T arrêté temporairement à la demande d'un signal comme CTRL+Z,
 - Z processus dans l'état zombie,
 - +, l, < indications additionnels avec des états ci-dessus (associé aux processus en premier plan, possède des processus légers, haute priorité).
- ❖ **utilisation processeur (c ou %CPU)** : indique le pourcentage utilisation du processeur par rapport à la durée de vie du processus,
- ❖ **terminal (TTY)** : terminal auquel est rattaché le processus (pour affichages et contrôle ...),
- ❖ **durée d'exécution (TIME)** : temps cumulé passé à exécuter le processus,
- ❖ **commande (CMD)** : correspond au nom du programme exécutable.
- ❖ **priorité d'exécution statique (PRI)** : priorité statique allant égale 0 pour les applications non temps réelle et allant de 1 à 99 pour les applications temps réelle (une valeur important implique une priorité plus forte),
- ❖ **priorité d'exécution dynamique (NI)** : priorité dynamique allant de -20 à 19 (une faible valeur implique une plus forte priorité) la priorité dynamique est utilisée entre les processus avec une priorité statique égale,
- ❖ **ordonnement (CLS)** : classe d'ordonnement utilisé pour le processus (toutes les classes sont préemptives), plusieurs valeurs sont possibles :
 - - ou ? : non signalé,
 - TS (SCHED_OTHER) : ordonnement classique en temps partagé (politique tourniquet appliquée pour tous les processus de cette classe) utilisé par la plupart des processus d'application non temps réel (priorité statique égale à 0),
 - B (SCHED_BATCH) : politique similaire à SCHED_OTHER utilisée pour les applications non-interactive car de priorité toujours plus faible que SCHED_OTHER,
 - FF (SCHED_FIFO) : ordonnement pour les applications temps réelle (priorité statique > 0) à base de tranches de temps, exécution du processus suivant la politique FIFO (premier arrivé premier servi) jusqu'à ce qu'il soit bloqué par une opération d'entrée/sortie ou préempté par un processus de priorité supérieure,
 - RR (SCHED_RR) : politique d'exécution tourniquet dans chaque file d'attente de même priorité où chaque processus s'exécute durant un quantum de temps et est ensuite placé à la fin de la liste de sa priorité (s'il a été préempté par un processus de priorité supérieure alors il terminera sa tranche de temps lorsqu'il reprendra son exécution).

L'ordonnanceur est la partie du noyau qui décide quel processus prêt va être exécuté. L'ordonnanceur de Linux propose trois grandes politiques différentes (une pour les processus classiques et deux pour les applications à vocation temps-réel).

Une valeur de priorité statique est assignée à chaque processus. L'ordonnanceur dispose d'une liste de tous les processus prêts pour chaque valeur possible de priorité statique (allant de 0 à

99). Afin de déterminer quel processus doit s'exécuter, l'ordonnanceur de Linux recherche la liste non-vide de plus haute priorité statique et prend le processus en tête de cette liste. La politique d'ordonnement détermine pour chaque processus l'emplacement où il sera inséré dans la liste contenant les processus de même priorité statique, et comment il se déplacera dans cette liste.

La commande `ps` (pour *Processus*) permet d'obtenir les informations décrites ci-dessus pour l'ensemble des processus en cours d'exécution.

Par défaut, cette commande n'affiche que les processus associés aux applications lancées explicitement par un utilisateur.

L'option `-u utilisateur` permet d'avoir accès à la liste de tous les processus associés à l'utilisateur `utilisateur` : `$> ps -u utilisateur`

L'option `-l` permet d'afficher plus d'informations sur les processus listés par exemple pour un utilisateur donné :

```
$> ps -lu utilisateur
```

Ci-dessous est donné un exemple d'informations obtenues à l'aide de ces paramètres :

F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
100	S	0	467	1	0	60	0	-	256	read_c	tty5	00:00:00	mingetty
100	S	0	468	1	0	60	0	-	256	read_c	tty6	00:00:00	mingetty
100	S	0	576	570	0	60	0	-	498	read_c	pts/0	00:00:00	cat
100	S	0	580	578	0	70	0	-	576	wait4	pts/1	00:00:00	bash
100	S	0	581	579	0	60	0	-	77	wait4	pts/2	00:00:00	bash
000	S	0	592	581	2	61	0	-	253	down_f	pts/2	00:00:01	essai
040	S	0	593	592	2	61	0	-	253	write_	pts/2	00:00:01	essai
100	R	0	599	580	0	73	0	-	652	-	pts/1	00:00:00	ps

Les champs `S`, `PID`, `PPID` et `CMD` codent respectivement l'état du processus, la valeur du `PID` et du `PPID` pour le processus et le nom du programme exécuté.

Plus généralement, la commande suivante permet d'afficher toutes les informations associées à tous les processus en cours d'exécution dans le système : `$> ps aux`

Afin de répondre aux questions ci-dessous, ouvrez un second terminal pour vous permettre d'afficher en parallèle de l'exécution des programmes demandés les informations fournies par la commande `ps`.

Question 1 – Exécutez dans un premier terminal le programme `prog_exo1` et indiquez à l'aide de la commande `ps` les informations pour le processus associé au programme `prog_exo1` (`PID`, `PPID`, état, priorité statique `PRI`, durée d'exécution `TIME`).

Qu'observez-vous pour le processus représentant l'exécution du programme `prog_exo1` ?

Correction : On peut constater qu'il y a deux processus exécutant le programme `prog_exo1`. L'un étant le père du second comme indiqué par les champs `PID` et `PPID` de ces deux processus.

De plus, les deux processus sont dans l'état `Stopped` bien qu'en cours d'affichage, cela est dû au fait que le code est simple et cours à exécuter, idem pour la durée d'exécution. Les deux processus ont la même priorité égale à 80.

Question 2 – Donnez l’option associée à la commande `ps` permettant d’afficher l’arborescence des processus, aidez vous de la page de manuel de la commande `ps`.

Correction : Il faut utiliser soit commande suivante `$> ps -ejH`
ou `$> ps axjf`

Question 3 – Par défaut la commande `ps` n’affiche pas la classe de politique d’ordonnancement utilisée pour chaque processus, il faut indiquer les informations désirées à afficher avec l’option `-o` comme suit :

```
$> ps -u utilisateur -o pid,ppid,class,ni,pri,pcpu,stat,comm
```

Exécutez le programme `prog_exo1` et utilisez la commande `ps` avec les options ci-dessus. Indiquez quelle politique d’ordonnancement est utilisée pour exécuter les processus associés au programme `prog_exo1`.

Correction : C’est la politique `SCHED_OTHER` qui est utilisée (politique par défaut).

Question 4 – Le processus `init` est l’ancêtre de tous les processus en cours d’exécution dans le système. Est-ce qu’il apparaît dans la liste obtenu à la question précédente ? Si ce n’est pas le cas, donnez la commande à taper pour que le processus `init` apparaisse dans la liste. Quel est le PID associé à ce processus ?

Correction : Il faut utiliser commande suivante `$> ps aux`
Le PID associé au processus `init` est égal à 1.

La commande `top` permet d’obtenir des informations similaires à la commande `ps`, mais contrairement à cette dernière ces informations sont mises à jour en temps réel.

Question 5 – Réitérez la procédure décrite en Question 1 mais en utilisant la commande `top`. Vous pouvez faire défiler l’affichage en utilisant les touches `<PageUp>` et `<PageDown>` (pour sortir de `top` vous pouvez utiliser la touche `q` ou `<CTRL>+c`).

Question 6 – A l’aide de la commande `top`, indiquez le nombre de processus en cours d’exécution, ainsi que ceux dans chacun des états : *running*, *sleeping*, *stopped* et *zombie*. Indiquez également le pourcentage d’utilisation processeur et mémoire physique.

Correction : Il faut relever les valeurs indiquées respectivement dans les champs : *Tasks total*, *Tasks running*, *Tasks sleeping*, *Tasks stopped* et *Tasks zombie*.
De même pour le processeur et la mémoire physique, avec les champs : *%Cpu us* (processus utilisateur), *%Cpu sy* (processus système), *KiB Mem total*, *KiB Mem used* et *KiB Mem free*.

Le système donne la possibilité à un utilisateur de demander l’arrêt de l’exécution d’un processus qu’il a lancé. Pour cela il faut utiliser la commande `kill` en indiquant le PID du processus dont on désire demander l’arrêt au système.

L’exemple ci-dessous demande l’arrêt du processus de PID 25446 :

```
$> kill -9 25446
```

Question 7 – Lancez le programme `prog_exo1` dans un premier terminal et utilisez la commande `kill` pour demander la terminaison du processus fils. Vous utiliserez la commande `ps` avant et après la commande `kill` dans le même terminal afin de visualiser les processus en exécution.

Est-ce que le processus fils s'est bien terminé ? Observez le PPID du processus parent.

Correction : Le processus fils a bien été arrêté par le système, et le PPID du processus parent n'a pas changé de valeur.

Question 8 – Même procédure que pour la Question 7, mais vous demanderez la terminaison du processus parent au lieu du processus fils.

Est-ce que le processus parent s'est bien terminé ? Quel est la valeur du PPID du processus fils après exécution de la commande `kill` ?

Expliquez ce que représente la valeur observée et pourquoi ?

Correction : Le processus parent a bien été arrêté par le système. On observe cette fois-ci que le PPID du processus fils a changé et est égal à 1. Ce PID correspond à celui du processus `init`. Le processus devient orphelin car son parent a été terminé, comme tout processus doit posséder un parent le système rattache tout processus orphelin au processus `init` qui est toujours présent durant tout le temps où le système est en fonctionnement.

Le système permet à chaque utilisateur de changer la priorité dynamique associé à l'exécution des processus associé à un programme qu'il désire exécuter à l'aide de la commande `nice`. Pour cela il suffit d'indiquer la priorité dynamique à utiliser et le programme à exécuter. Dans l'exemple ci-dessous on exécute un programme `toto` avec une priorité dynamique de 15 :

```
$> nice -n 15 ./toto
```

Il est également possible de changer la priorité dynamique d'un processus en cours d'exécution. Pour cela il faut utiliser la commande `renice` à la place de `nice` et indiquer le PID du processus avec la nouvelle priorité dynamique comme illustré ci-dessous pour le processus 24546 avec une nouvelle priorité de 15 :

```
$> renice -n 15 -p 24546
```

Question 9 – Utilisez la commande `nice` et donnez la commande permettant d'exécuter le programme `prog_exo1` avec une priorité dynamique de -5.

Indiquez en utilisant la commande `ps` dans un second terminal si la priorité des processus associés au programme `prog_exo1` a bien été prise en compte.

Correction : Il faut utiliser commande suivante `$> nice -n -5 ./prog_exo1`

Question 10 – Lancez le programme `prog_exo1` dans un premier terminal et utilisez la commande `renice` dans un second terminal pour affecter la priorité dynamique -5 au processus parent ou fils.

Quelle commande à utiliser ? Indiquez en utilisant la commande `ps` dans le second terminal si la nouvelle priorité du processus a été prise en compte.

Correction : Il faut utiliser commande suivante `$> renice -n -5 -p 24561`

EXERCICE 3

Dans cet exercice, nous allons nous intéresser aux redirections de flux d'informations des processus. Comme vu en cours, chaque processus possède 3 types de flux d'informations :

- STDIN : flux d'information en entrée (par défaut provenant du clavier),
- STDOUT : flux d'information en sortie pour l'affichage (par défaut vers l'écran),
- STDERR : flux d'information en sortie pour l'affichage des messages d'erreur (par défaut vers l'écran).

Le système d'exploitation Linux permet de rediriger ces flux vers d'autres destinations, par exemple vers un fichier au lieu de l'écran.

Pour contrôler les redirections, plusieurs symboles (ou opérateur) sont disponibles afin d'indiquer au système ce que l'on désire réaliser :

- `>` : redirige la sortie standard d'un processus vers un fichier, si celui-ci n'existe pas alors il est créé sinon son contenu est écrasé (supprimé),
- `>>` : redirige la sortie standard d'un processus vers un fichier, si celui-ci n'existe pas alors il est créé sinon la sortie est ajoutée en fin du fichier,
- `<` : redirige le contenu d'un fichier vers l'entrée standard d'un processus,
- `|` : outil appelé « tube » permettant de rediriger la sortie standard d'un processus vers l'entrée standard d'un autre processus, permettant ainsi d'exécuter un enchaînement de programmes où chaque programme traite en entrée les informations fournies en sortie par son prédécesseur dans la chaîne.

Question 1 – Soit la commande `$> echo "Ceci est une phrase de test !"` permettant d'afficher le message "Ceci est une phrase de test !" dans le terminal dans lequel la commande est lancée.

Donnez la commande permettant d'ajouter ce message dans le fichier `test.txt` qui doit être créé.

Exécutez cette commande et éditez le fichier `test.txt` et indiquez quel est son contenu.

Correction : Il faut utiliser commande suivante

```
$> echo "Ceci est une phrase de test !" > test.txt
```

Le fichier `test.txt` contient le message passé en paramètre de la commande `echo`.

Question 2 – Même question en ajoutant la sortie fournie par la commande `ls -al` dans le fichier `test.txt`.

Editez le fichier `test.txt` et indiquez ce qu'il contient.

Est-ce que le fichier `test.txt` contient les deux messages ? Pourquoi ?

Correction : Il faut utiliser commande suivante

```
$> ls -al > test.txt
```

Le fichier test.txt ne contient que la sortie de la commande ls -al. Cela vient du fait avec l'opérateur > le contenu du fichier est écrasé.

Question 3 – Indiquez la suite de commandes à exécuter afin que le fichier test.txt contienne les deux messages l'un à la suite de l'autre.

Correction : Il faut utiliser la suite de commandes suivante

```
$> echo "Ceci est une phrase de test !" > test.txt  
$> ls -al >> test.txt
```

Question 4 – La commande \$> wc -c permet d'obtenir le nombre de caractères (c'est-à-dire indirectement la taille) du fichier passé en paramètre.

Indiquez comment associer le contenu du fichier test.txt à l'entrée standard de cette commande.

Exécutez cette commande et indiquez le résultat obtenu.

Correction : Il faut utiliser la commande suivante

```
$> wc -c < test.txt
```

Question 5 – La commande wc -w nom_fich compte le nombre de mots contenu dans le fichier nom_fich. Nous désirons réaliser la même opération à l'aide de l'opérateur pipe (« | ») à partir d'une chaîne de caractères fournie par la commande echo.

Indiquez comment réaliser cela en un seul appel à l'aide des commandes indiquées ci-dessus.

Correction : Il faut utiliser la commande suivante

```
$> echo "Coucou, ceci est un message" | wc -w
```

Question 6 – On désire trier les lignes du fichier texte poeme.txt par ordre alphabétique de la première lettre de chaque ligne et les afficher page par page sur le terminal.

Pour cela, vous utiliserez la commande sort permettant de trier par ordre alphabétique (ordre par défaut) les lignes passées en entrée.

Indiquez comment réaliser cela en un seul appel.

Pour vous aider : vous devez tout d'abord éditer le fichier, puis trier les lignes et enfin afficher le résultat page par page avec une dernière commande.

Correction : Il faut utiliser la commande suivante

```
$> cat poeme.txt | sort | more
```

EXERCICE 4

Nous allons nous intéresser ici aux variables utilisées par un terminal pour stocker des informations importantes nécessaires à la bonne exécution des commandes lancées par les

utilisateurs sous les systèmes Linux. Par exemple, cela permet au terminal de déterminer le répertoire courant de l'utilisateur.

On distingue deux types de variables : les *variables d'environnement* ou les *variables du terminal*. Le premier type contient des variables affectées par le système ou le terminal lors de son lancement, celles-ci sont transmises aux processus fils engendrés par le terminal. Tandis que le deuxième type de variables ne sont utilisables que par terminal courant (les processus fils ne peuvent en hériter). Ces dernières sont essentiellement utilisées pour maintenir une information temporaire nécessaire au traitement réalisé par une commande.

Les variables (d'environnement ou du terminal) sont un ensemble de paires clé-valeur et une variable peut contenir une suite de valeurs séparées par le symbole « : ». Le nom d'une variable est composé de lettres, chiffres et de certains symboles.

Par convention, des lettres majuscules sont utilisées pour les noms des variables d'environnement, tandis que pour les noms des variables du terminal des lettres minuscules peuvent être utilisées pour ne pas les confondre avec des variables d'environnement. Les noms des variables sont sensibles à la casse, par exemple `home`, `HOME`, `HoMe` et `HomE` sont vu comme des variables différentes.

La valeur des variables peut être un nombre ou une chaîne de caractères. Dans le cas où une variable contient des espaces, il faut encadrer la valeur par des guillemets (comme suit `VAR="valeur avec des espaces"`), dans le cas où le symbole « ! » est contenu dans la chaîne il faut utiliser des cotes (') afin que le symbole « ! » ayant une signification pour le terminal ne soit pas interprété.

Voici ci-dessous l'exemple d'une variable `VAR` contenant une liste de valeurs :
`VAR=val1:val2:val3.`

Nous listons ci-dessous (de façon non exhaustive) quelques variables d'environnement en indiquant leur utilisation :

- **DISPLAY** : l'écran sur lequel les programmes possédant une interface graphique réalise leur affichage (si *vide* il n'y a pas d'affichage graphique, `:0` indique l'écran standard local, `@ip:0` permet un affichage distant),
- **SHELL** : contient le chemin d'accès vers le terminal utilisé (en général `/bin/bash`),
- **HOME** : contient le chemin d'accès vers la racine de votre compte utilisateur,
- **USER** : contient le nom de votre login,
- **PATH** : contient une liste de répertoires dans lesquels votre terminal va chercher les commandes que vous lui demandez d'exécuter,
- **PWD** : contient votre répertoire courant,
- **LANG** : contient la langue utilisée par le système.

Note : le terminal utilise un fichier lors de son lancement pour charger les variables d'environnement qu'il devra utiliser, le fichier `.bashrc`. Ce fichier se trouve à la racine du compte de chaque utilisateur. Il est ainsi possible d'ajouter/modifier des variables d'environnement afin que celles-ci soient automatiquement déclarées au prochain lancement du terminal.

Il est possible de lister toutes les variables d'environnement du système et leur contenu à l'aide des commandes `env` ou `printenv`.

Attention : ces deux commandes ne considèrent que les variables d'environnement et non les variables du terminal.

Question 1 – Exécutez la commande dans un terminal. Donnez les variables d'environnement déclarées dans le système sur votre machine, dites si les variables listées ci-dessus sont présentes et quelle est leur valeur (notamment pour les variables HOME, USER, PATH et PWD).

Correction : Il faut utiliser la commande suivante avec un exemple de résultat

```
$> env
DM_CONTROL=/var/run/xdmctl
TERM=xterm
SHELL=/bin/bash
SHELL_SESSION_ID=bf954bdad96f4000937c861db4828ee1
USER=steph
SESSION_MANAGER=local/debian:@/tmp/.ICE-unix/3179,unix/debian:/tmp/.ICE-unix/3179
PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
DESKTOP_SESSION=default
PWD=/home/steph/Travail/TP_DUT
LANG=fr_FR.UTF-8
KDE_SESSION_UID=1000
HOME=/home/steph
XCURSOR_THEME=oxy-steel
LOGNAME=steph
DISPLAY=:0
OLDPWD=/home/steph
```

Les commandes `env` et `printenv` permettent de réaliser des tâches différentes. La commande seconde permet contrairement à la première d'afficher le contenu de la variable indiquée en paramètre (par exemple, `printenv HOME` pour le contenu de la variable `HOME`). Alors que la commande `env` permet d'exécuter une commande passée en paramètre dans un environnement modifié dans lequel une liste de variables d'environnement sont déclarées. Dans l'exemple ci-dessous, les variables d'environnement `TEST1` et `TEST2` sont déclarées avec le contenu "toto" et "titi", puis affichée dans le processus fils engendré par le processus parent :

```
$> env TEST1="toto" TEST2="titi" printenv
```

Question 2 – Exécutez tout d'abord la commande : `$> env TEST1="toto" TEST2="titi" printenv` puis la commande `$> printenv`.

Dites si la variable `TEST1` est toujours reconnu, expliquez pourquoi.

Correction : La première commande permet d'exécuter la commande `printenv` dans une nouvelle instance du terminal dans lequel les variables `TEST1` et `TEST2` ont été déclarées. Tandis que le deuxième appel à `printenv` est réalisé dans l'environnement initial (ne contenant pas la déclaration des deux variables) donc le terminal ne peut afficher leur contenu.

Il est également possible d'afficher le contenu des variables du terminal à l'aide de la commande `set`, comme beaucoup de variables sont déclarées en général utilisez plutôt la combinaison : `$> set | less`.

Question 3 – Utilisez la commande ci-dessus et listez les variables du terminal sur votre machine.

Il est également possible de déclarer des variables du terminal dans la session courante, pour cela il suffit de suivre la syntaxe suivante : `VAR=val`.

Question 4 – Exécutez la commande suivante `$> set | grep TEST` et dites si le terminal affiche un résultat. Exécutez à la suite les deux commandes suivantes puis dites si le terminal affiche en résultat le contenu déclaré dans la variable `TEST` :

```
$> TEST="ceci est un test"  
$> set | grep TEST
```

Question 5 – Exécutez la commande `printenv` ou `env` et dites si la variable `TEST` apparaît dans la liste des variables d'environnement.

Correction : Non, car les commandes `printenv` ou `env` ne permettent d'afficher que les variables du terminal, or `TEST` est une variable du terminal.

Il est également possible d'accéder au contenu des variables d'environnement ou du terminal en utilisant le symbole `$`. Par exemple, la commande suivante permet d'afficher la variable d'environnement `HOME` : `$> echo $HOME`.

Note : pour affecter un contenu à une variable on utilise le nom de la variable **sans le symbole \$**, tandis que pour récupérer le contenu d'une variable on indique le nom de la variable **avec le symbole \$** devant.

Question 6 – Essayez d'afficher le contenu de la variable d'environnement `PWD` et la variable du terminal `TEST` que vous venez de déclarer avec la commande `echo`.

Dites si le contenu de ces deux variables est bien affiché.

Question 7 – Tapez la commande `$> bash` pour créer une nouvelle instance du terminal (processus fils du terminal parent), puis déclarez la nouvelle variable du terminal :

```
$> VAR_LOC="nouvelle variable", enfin tapez les commandes $> echo $VAR_LOC  
et $> exit pour afficher le contenu de VAR_LOC et quitter la nouvelle instance du terminal.  
Tapez à nouveau la commande $> echo $VAR_LOC et dites si le contenu de VAR_LOC est toujours accessible. Pourquoi ?
```

Correction : Non, le contenu de la variable n'est plus accessible car c'est une variable du terminal propre à l'instance fils et non transmise à l'instance parent du terminal.

De même que pour les variables du terminal, il est aussi possible de créer de nouvelles variables d'environnement. Pour cela il faut utiliser la commande `export` comme suit :

```
$> export VAR=val.
```

Question 8 – Créez une nouvelle variable d’environnement à l’aide de la commande `export`, puis démarrez une nouvelle instance du terminal avec la commande `bash` et dites si vous avez accès à la variable d’environnement créée dans l’instance parent du terminal en affichant son contenu avec la commande `echo`.

Enfin, quittez l’instance fils du terminal avec la commande `exit` pour revenir à l’instance parent et dites si vous avez toujours accès au contenu de la variable que vous avez créé.

Question 9 – Essayez de suivre la même procédure mais cette fois-ci en déclarant la variable d’environnement dans l’instance fils du terminal (au lieu de l’instance parent).

Dites si vous pouvez afficher le contenu de la variable d’environnement dans l’instance parent du terminal. Pourquoi ?

Correction : Non, les variables d’environnement ne sont héritées que par les fils. Un processus parent ne pourra donc pas en hériter de l’un de ses processus fils.

Question 10 – Dites comment modifier la variable d’environnement `PATH` pour ajouter le répertoire contenant l’exécutable du programme `prog_exo1` réaliser à l’Exercice 1.

Placez-vous à la racine de votre compte utilisateur et essayez d’exécuter à nouveau le programme. Dites ce que cela change.

Correction : Pour modifier la variable `PATH` il faut exécuter la commande ci-dessous où `$HOME/TP` est le répertoire contenant l’exécutable `prog_exo1`

```
$> export $PATH=$PATH:$HOME/TP
```

L’effet de ce changement est de ne plus avoir à fournir le chemin d’accès vers le programme `prog_exo1`, le terminal détermine tout seul où se trouve le programme.

Note : une variable d’environnement peut être changée en une variable du terminal à l’aide de la commande `export`, comme suit pour une variable d’environnement `VAR`

```
$> export -n VAR
```

Il est bien évidemment possible de supprimer une variable d’environnement ou une variable du terminal en indiquant le nom de la variable à la commande `unset`.

Question 11 – Déclarez une nouvelle variable puis supprimez la.

Dites si vous arrivez à afficher son contenu après l’avoir supprimer.