

8INF342 – Systèmes d'exploitation

Laboratoire #1

Familiarisation avec linux et gestion des processus

Automne 2022

Professeure : Sara Séguin

Instructions générales

- Devoir à remettre au plus tard le **6 octobre 2022 à 16h**.
- Travail à réaliser en équipe de 2 étudiants.
- Un seul rapport à remettre par équipe.
- Ce travail compte pour 12% de la note finale.

Objectifs du devoir

Partie I : Familiarisation avec le système d'exploitation linux	2
Partie II : Compilation d'un programme en ligne de commande.....	5
Partie III : Créer un projet en C++ avec Visual Studio Code.....	6
Partie A : Compilation et exécution.....	6
Partie B : Configuration du mode debug.....	9
Partie IV : Gestion des processus sous linux	12
Partie V : Historique des commandes	16

Mise en place

Avant d'effectuer votre travail, vous devez d'abord mettre en place les outils qui seront utilisés au courant de la session.

Une machine virtuelle est créée pour le cours. Le système d'exploitation est Linux Ubuntu 18.04 et Visual Studio Code est installé. Tous les programmes seront développés en C++.

Vous devriez avoir reçu un **courriel avec un lien** vous permettant d'installer VMWare. Noter qu'il se retrouve souvent dans les courriels indésirables...

1. Téléchargez VMware Workstation 16.x Player à l'adresse suivante: <http://tiny.cc/dim-vmware>.
2. Téléchargez la machine virtuelle à l'adresse suivante : <http://tiny.cc/8inf341vm>

3. Ouvrez VMWare, puis importez la machine virtuelle. Le username : etudiant et le password : etudiant.
4. NOTE : Ce ne sont pas tous les CPU qui permettent la virtualisation. Aussi, sur certains systèmes, il faut activer l'option directement dans le BIOS. Si vous obtenez un message d'erreur lorsque vous tentez d'ouvrir la machine sur votre ordinateur portable, veuillez le signaler à l'assistant de laboratoire.
5. Les fichiers nécessaires pour le devoir sont disponibles sur le site MOODLE du cours.

NOTE : LE NOMBRE DE CORES PEUT ÊTRE CHANGÉ DANS LES PARAMÈTRES DE LA VM. Par exemple si un ordinateur a un dual core mais que la VM utilise 4 threads ça ne fonctionnera pas.

Énoncé du devoir

Partie I : Familiarisation avec le système d'exploitation linux

Cette première partie vise à vous familiariser avec Linux et les commandes habituelles de terminal. Lorsque vous exécutez la machine virtuelle, vous verrez le bureau tel qu'illustré à la Figure 1.

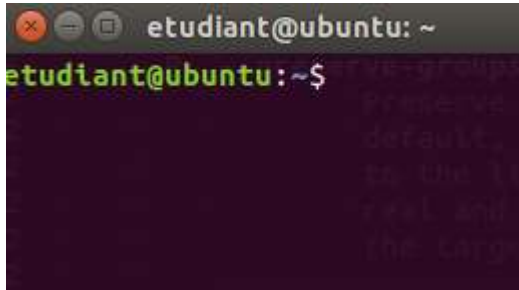


Le tableau 1 présente les commandes de base du terminal. Cette section vise simplement à vous familiariser avec le terminal et ses commandes usuelles.

Sous linux, le shell (terminal) est un interpréteur de commandes. Ce que vous voyez à l'écran permet d'entrer des commandes qui sont ensuite exécutées par le système d'exploitation.

Suivez les instructions suivantes :

1. Cliquez droit sur le bureau et choisissez : **Open terminal**.



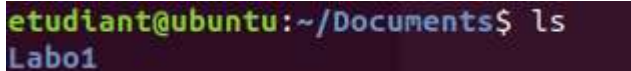
2. Entrer `ls` sur le terminal puis faire « enter ». Vous devriez voir une liste à l'écran. En bleu ce sont les répertoires et en blanc les fichiers.
 3. Entrer `cd Documents` sur le terminal puis faire « enter ».
 4. Entrer `pwd`. Cette commande affiche le répertoire courant, et devrait retourner : `/home/etudiant/Documents`.
 5. Entrer `ls` sur le terminal. Puisqu'il n'y a pas de fichiers ni de dossiers sous le répertoire « Documents », il ne devrait pas y avoir d'affichage.
 6. Entrer `mkdir Labo1`.
 7. Entrer `ls`. Vous devriez voir votre nouveau répertoire Labo1
- 
8. Entrer `echo jello`. Jello s'affiche à l'écran
 9. Entrer `echo jello >> jello.txt`.
 10. Entrer `ls`. Un fichier jello.txt contenant jello est créé.
 11. Entrer `cat jello.txt`. Afficher le contenu du fichier, soit jello.
 12. Entrer `cd ..`. Cette commande change le répertoire courant pour le répertoire parent.
 13. Entrer `exit`. Le terminal ferme.

Tableau 1 : Commandes de base du terminal

ls	afficher le contenu d'un répertoire
pwd	afficher le répertoire courant
cd	changer de répertoire
cd ..	changer le répertoire courant pour le répertoire parent
man <nom de la commande>	afficher le manuel utilisateur de la commande

mkdir	créer un répertoire
rmdir	supprimer un répertoire
mv	déplacer un fichier
sudo	mode administrateur
exit	fermer le terminal
echo	afficher une ligne de texte à l'écran
>>	redirection du terminal vers un fichier. Si le fichier n'existe pas il est créé et la redirection est ajoutée à la fin du fichier.
>	redirection du terminal vers un fichier. Le fichier est remplacé s'il existe.
cat	concaténer et afficher à l'écran. Cette commande a plusieurs utilisations, je vous invite à utiliser man.

Finalement, si vous cliquez sur Fichiers, vous devriez normalement voir le répertoire *Labo1* et le fichier *jello.txt*.

Fichiers



Partie II : Compilation d'un programme en ligne de commande

Cette partie vise à vous familiariser avec la compilation en ligne de commande sous Linux.

Renommer le fichier jello.txt par jello.cpp. Remplacer son contenu par ceci :

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
int main()
{
    vector<string> msg {"Hello", "Jello", "Mon", "premier", "Code"};
    for (const string& word : msg)
    {
        cout << word << " ";
    }
    cout << endl;
}
```

Ouvrir le terminal. Utiliser la commande `cd` afin de positionner le répertoire courant à l'emplacement du fichier jello.cpp.

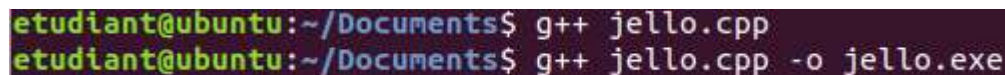
Afin d'exécuter votre code, il doit d'abord être compilé. Le compilateur peut être invoqué directement en ligne de commande.

Toutes les commandes ont la structure suivante :

Commande – options

Je vous invite à lire les manuels utilisateurs (`man <nom de la commande>`) pour plus de détails sur les commandes vues précédemment.

Invoquer `g++ jello.cpp` crée automatiquement un fichier `a.out`, qui se trouve à être un exécutable du programme passé en argument. Cela signifie que le code a été compilé et assemblé et qu'il est prêt à l'exécution.



```
etudiant@ubuntu:~/Documents$ g++ jello.cpp
etudiant@ubuntu:~/Documents$ g++ jello.cpp -o jello.exe
```

Si vous invoquez la deuxième commande ci-haut, le `-o` spécifie que le fichier de sortie se nommera `jello.exe`. Compilez votre code et créez l'exécutable `jello.exe`.

Pour exécuter un programme par le terminal, la commande suivante est utilisée :

./nomDuProgramme

Lancer votre programme et valider que le message « Hello Jello Mon Premier Code » s'affiche dans le terminal.

Partie III : Créer un projet en C++ avec Visual Studio Code

Afin de vous familiariser avec Visual Studio Code, veuillez compléter le tutoriel suivant. Dans un premier temps, vous pourrez compiler et exécuter un code dans Visual Studio Code. La seconde partie vous explique comment configurer le mode debug.

Partie A : Compilation et exécution

Précédemment, vous avez créé un répertoire *Labo1*. Créez un nouveau répertoire *PremierProjet* dont le parent est *Labo1*. Faites un clic droit, ouvrez le terminal et entrez `code .` (avec le point!), cela ouvrira visual studio code dans le répertoire courant.

Lorsque Visual Code sera ouvert, cliquez sur New file, tel qu'illustré à la Figure 2.

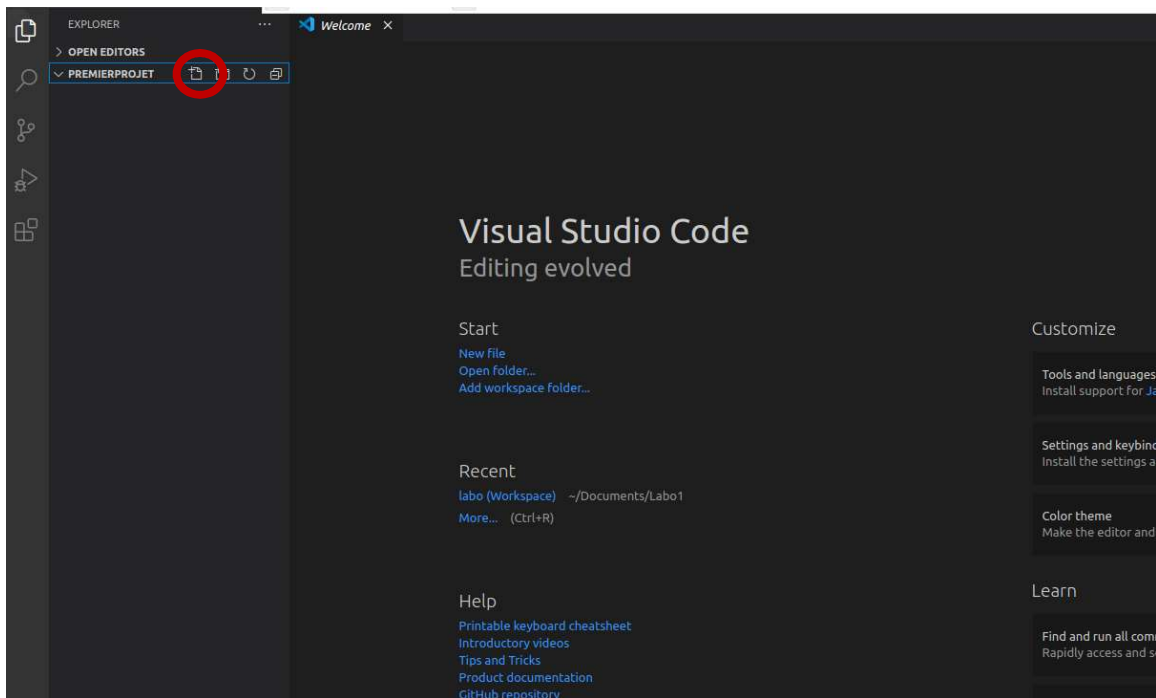


Figure 2. Visual Studio Code

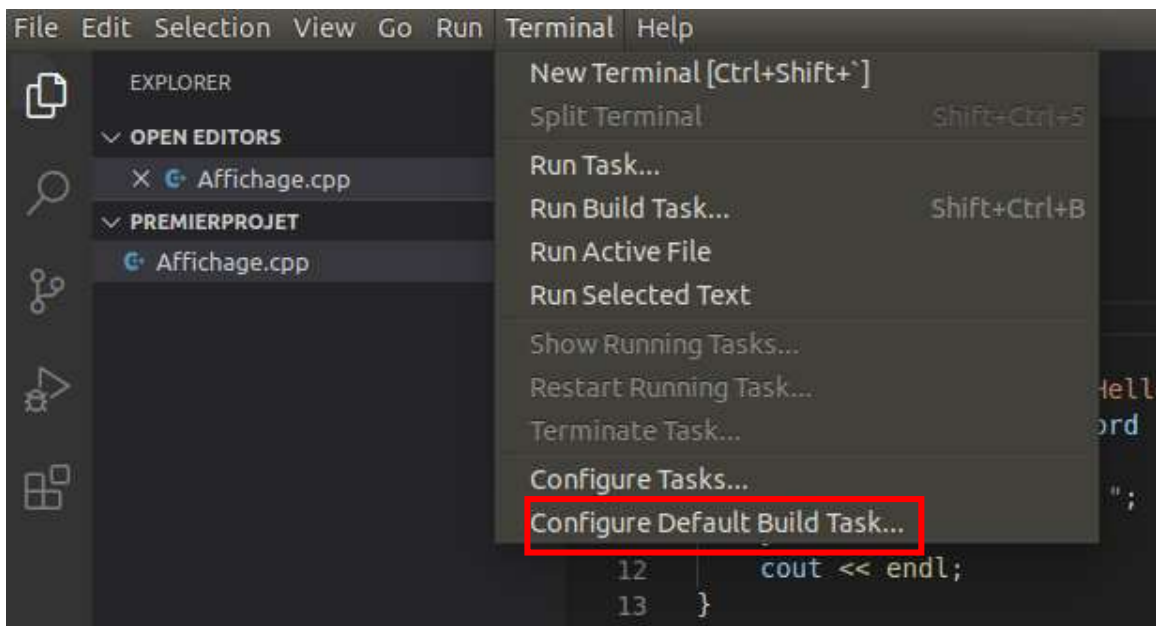
Nommer le nouveau fichier *Affichage.cpp* et y inscrire ceci. Ensuite, utiliser Ctrl+S pour sauvegarder.

```

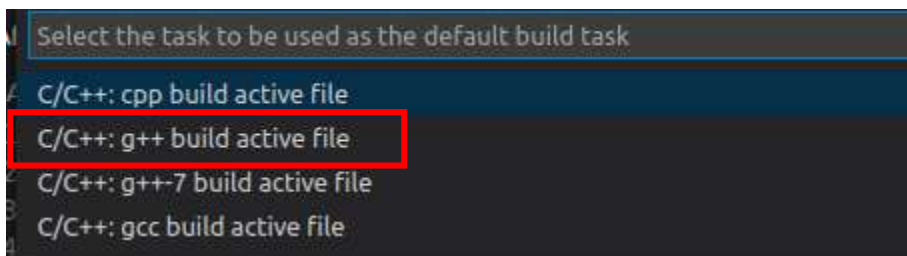
#include <iostream>
#include <vector>
#include <string>
using namespace std;
int main()
{
    vector<string> msg {"Hello", "Jello", "Mon", "premier", "Code"};
    for (const string& word : msg)
    {
        cout << word << " ";
    }
    cout << endl;
}

```

Dans le menu, cliquer sur *Terminal*, puis sur *Configure Default Build Task*.



Choisir ensuite C/C++ : g++ build active file.

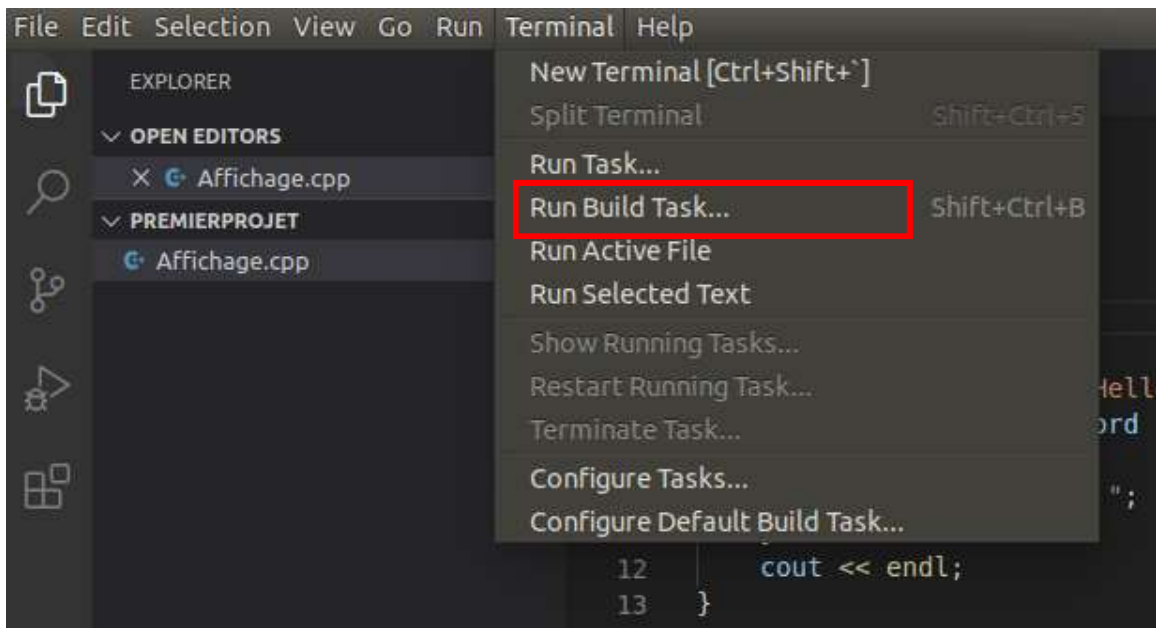


Cela créé un fichier *tasks.json* qui devrait ressembler à ceci :

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "type": "shell",
      "label": "C/C++: g++ build active file",
      "command": "/usr/bin/g++",
      "args": [
        "-g",
        "${file}",
        "-o",
        "${fileDirname}/${fileBasenameNoExtension}"
      ],
      "options": {
        "cwd": "${workspaceFolder}"
      },
      "problemMatcher": [
        "$gcc"
      ],
      "group": {
        "kind": "build",
        "isDefault": true
      }
    }
  ]
}
```

Ce fichier permet de lancer la compilation de votre programme. D'abord, assurez-vous que *Affichage.cpp* est ouvert (et non *tasks.json*). Cliquer à nouveau sur

Terminal -> Run Build task.



Le message suivant s'affichera dans le terminal de Visual Studio Code :

```
> Executing task: /usr/bin/g++ -g /home/etudiant/Documents/Lab01/PremierProjet/Affichage.cpp -o /home/etudiant/Documents/Lab01/PremierProjet/Affichage <

Terminal will be reused by tasks, press any key to close it.
```

Appuyer sur une touche et entrer `./Affichage` dans le terminal de Visual Studio Code.

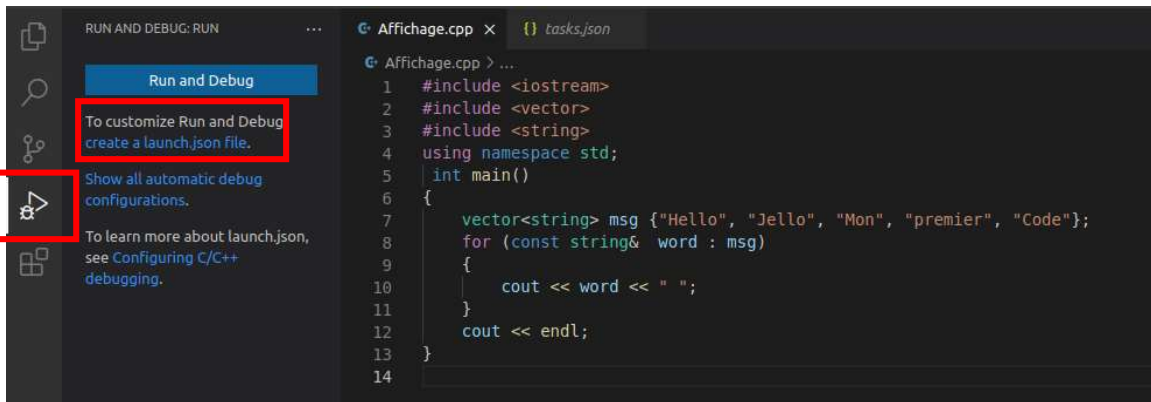
```
etudiant@ubuntu:~/Documents/Lab01/PremierProjet$ ./Affichage
Hello Jello Mon premier Code
```

Voilà, vous avez compilé et exécuté votre premier programme.

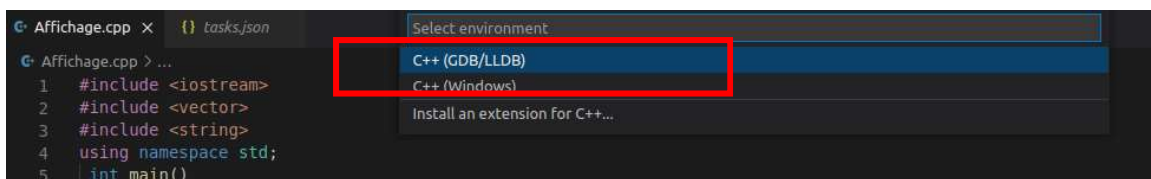
Partie B : Configuration du mode debug

Cette partie permet de configurer le mode debug, sinon vous ne pourrez pas ajouter de breakpoints à votre programme.

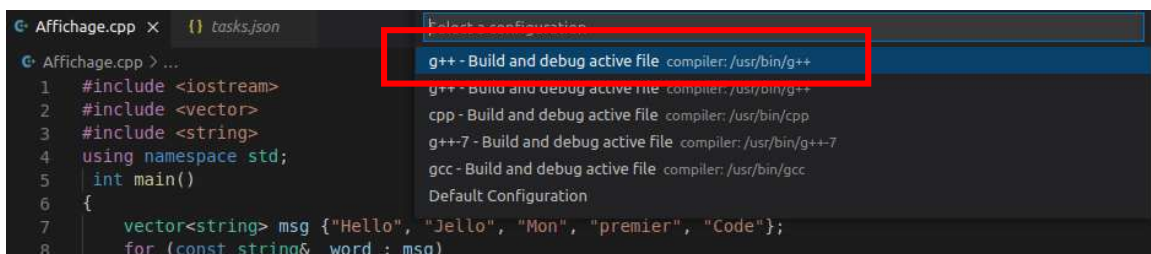
Cliquer sur le menu de gauche de VSCode (icône debug avec play) puis sur *create a launch.json file*.



Sélectionner ensuite C++ (GDB/LLDB).



Puis



Un fichier *launch.json* est créé automatiquement. Valider que le paramètre « stopAtEntry » est égal à « true », sinon le modifier.

```
{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
  "version": "0.2.0",
  "configurations": [
    {
      "name": "g++ - Build and debug active file",
      "type": "cppdbg",
      "request": "launch",
      "program": "${fileDirname}/${fileBasenameNoExtension}",
      "args": [],
      "stopAtEntry": true,
      "cwd": "${workspaceFolder}",
      "environment": [],
      "externalConsole": false,
      "MIMode": "gdb",
      "setupCommands": [
```

```
{
  "description": "Enable pretty-printing for gdb",
  "text": "-enable-pretty-printing",
  "ignoreFailures": true
},
"preLaunchTask": "C/C++: g++ build active file",
"miDebuggerPath": "/usr/bin/gdb"
}
]
```

Vous pouvez maintenant ajouter des breakpoints à votre programme et le débogger pas à pas.

Partie IV : Gestion des processus sous linux

La quatrième partie de ce laboratoire vise à vous familiariser avec la gestion des processus sous Linux. Deux programmes sont nécessaires pour compléter cette partie du laboratoire. Ils sont disponibles sur le site moodle du cours.

- Téléchargez les fichiers programme1 et programme2 sur le site moodle du cours. Vous pouvez simplement glisser les fichiers de votre ordinateur vers la machine virtuelle. Renommer par programme1.exe et programme2.exe.

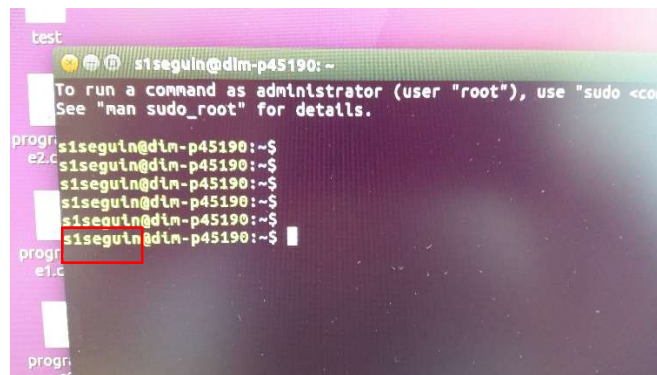
Ces programmes C++ sont des exécutables. Pour lancer un programme dans le terminal linux, la commande suivante est utilisée :

`./nomDuProgramme`

En utilisant cette commande, le programme est lancé en avant-plan (foreground).

Q1. Quel est le comportement d'un processus en avant-plan? Définir un processus en avant-plan dans vos mots.

- Ouvrez un terminal. Noter votre nom d'utilisateur. Par exemple,



dans le terminal ci-haut, le nom d'utilisateur est s1seguin.

Voici une commande :

`ps -u <nomUtilisateur> -o pid, ppid, pri, state, tty, stime,args`

Q2. À l'aide de man et sans lancer la commande, expliquer cette commande dans vos mots, ainsi que les options déclarées.

En tant qu'utilisateur, les actions pouvant être posées en passant par le terminal pour gérer les processus sont limitées. Les commandes disponibles sont un outil de gestion, mais permettent tout de même de modifier indirectement les processus, car le système d'exploitation insérera toujours les commandes demandées par l'utilisateur au moment opportun dans sa gestion des processus.

Noter que les quatre états de processus importants pour ce laboratoire sont : R, S, Z et T. Puisque vous les avez définis à la question 2, des explications ne sont pas nécessaires ici.

Avant de débiter les manipulations, ouvrez trois terminaux sur votre poste de travail. Le premier terminal (T1) servira à visualiser les processus à l'aide de la commande `ps`, le second (T2) servira à lancer le programme1 et le troisième (T3) servira à lancer le programme2.

- Lancez la commande `ps -u <nomUtilisateur> -o pid, ppid, pri, state, tty, stime,args` dans T1. **Pour la suite de ce laboratoire, une instruction spécifiant de lancer la commande `ps` consiste à lancer toute la commande ci-haut.** Notez que si vous lancez simplement la commande `ps` dans un terminal, la commande affichera les processus liés à ce terminal seulement et non à l'utilisateur.

Q3. Faites une capture d'écran du résultat de la commande.

- Lancez la commande `./programme1` dans T2.

Q4. Que se passe-t-il dans le terminal T2?

***À partir d'ici, noter que « Lancez la commande `ps` » signifie**
`ps -u <nomUtilisateur> -o pid, ppid, pri, state, tty, stime,args`

- Lancez la commande **ps** dans T1.

Q5. Vous devriez voir ./programme1. Notez le numéro du processus ainsi que son état. Expliquez l'état du processus.

- Appuyez sur la combinaison de touches **ctrl-z** dans T2.

Q6. Que s'est-il produit dans T2?

- Lancez la commande **ps** dans T1.

Q7. Repérez le numéro du processus pour programme1 et notez son état. Est-ce que l'état est différent de la question 5? Si non, refaire les manipulations. Si oui, expliquez la fonction de la combinaison de touches ctrl-z. Est-ce que le programme 1 consomme du temps CPU?

- Tapez, dans T2, la commande **fg**.

Q8. Que s'est-il produit dans T2?

- Lancez la commande **ps** dans T1.

Q9. Repérez à nouveau le numéro du processus pour programme 1 et notez son état. Est-il différent de la Question 7? Si non, refaire les manipulations. Si oui, expliquez la fonction de la commande fg.

- Lancez la commande **./programme2 &** dans T3.

Q10. Que s'est-il produit dans T3?

- Lancez la commande **ps** dans T1. Vous devriez voir deux lignes affichant ./programme2. Regardez le code derrière programme2 :

```
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <string>
#include <sys/types.h>
#include <unistd.h>

using namespace std;
int main(void){
    pid_t processus = fork();
    sleep(86400);
}
```

Q11. Pourquoi y a-t-il deux processus nommés programme2? Est-ce qu'il s'agit du même processus? Regardez aussi les pid et ppid pour vous aider dans votre réponse.

- Dans T3, tapez kill <numero>, mais remplacez <numero> par le pid du processus enfant programme2. Ensuite, lancez la fonction ps dans T1.

Q12. Est-ce que le processus enfant programme2 a changé d'état? Si oui, quel est-il maintenant? Expliquez ce que cet état signifie.

- Maintenant, dans T3, tapez kill <numero>, mais remplacez <numero> par le pid du programme2 parent. Ensuite lancez la commande ps dans T1.

Q13. Est-ce que programme2 est toujours visible? Si non, quelle est la fonction de la commande kill?

- Utilisez la bonne commande pour terminer le processus lié à programme1, puis assurez-vous qu'il n'y ait plus d'instances de programme1 ou programme2 lorsque vous lancez la commande ps dans T1.

Q14. Quelle est la commande que vous avez tapée au terminal pour terminer le programme1? Donner des détails sur cette commande, en vous aidant de man.

Partie V : Historique des commandes

Vous devez programmer, en C++, un environnement qui réplique un terminal.

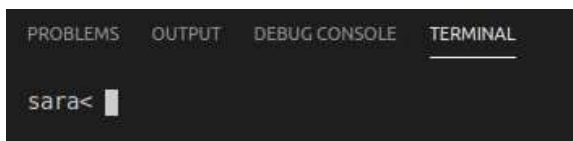
Les seules commandes que vous pouvez entrer en ligne de commande sont celles au Tableau 2. Évidemment, vous pouvez aussi entrer des options pour chaque commande. Utiliser le manuel de la commande pour avoir plus de détails.

Tableau 2 : Commandes possibles

ls	afficher le contenu d'un répertoire
pwd	afficher le répertoire courant
man <nom de la commande>	afficher le manuel utilisateur de la commande
mkdir	créer un répertoire
rmdir	supprimer un répertoire
mv	déplacer un fichier
cat	concaténer et afficher à l'écran. Cette commande a plusieurs utilisations, je vous invite à utiliser man. Pour les besoins du laboratoire, cat affichera le contenu d'un fichier.

Attention : Certaines commandes ne fonctionnent pas avec execvp. Celles du tableau 2 devraient fonctionner, si ce n'est pas le cas, ne pas l'utiliser.

Lorsque vous lancez le programme, l'invite de commande est votre nom, par exemple :



Utilisez les 2 noms de famille des étudiants pour votre invite de commande.

L'utilisateur entre une commande suivie d'options. Afin de simplifier le tout, seulement 1 option peut être entrée, donc un seul tiret (-).

Par exemple, vous pourriez entrer `ls -l`. Les commandes et les options doivent être placées dans un array :

Argument[0] = « ls »

Argument[1] = « -l »

Argument[2] = NULL

Noter que le dernier argument NULL est nécessaire (voir la définition de `execvp`).

Lorsque la touche « entrée » est appuyée, un processus est créé et exécute la commande entrée, ce qui donnerait :

```
sara< ls -l
total 100
-rwxr-xr-x 1 etudiant etudiant 97624 Aug 31 14:14 Affichage
-rw-r--r-- 1 etudiant etudiant 1595 Aug 31 14:14 Affichage.cpp
```

Les fonctions suivantes doivent être utilisées :

`fork()` : créer un processus enfant

`execvp` : exécuter la commande entrée dans votre terminal

Le symbole & placé à la fin d'une commande signifie que la commande sera exécutée dans le background par le processus enfant, sinon, le parent doit attendre que le processus enfant se termine.

Votre programme doit conserver un historique des 100 dernières commandes dans un fichier texte et un historique des 8 dernières commandes à l'écran.

Par exemple, si vous entrez 10 commandes une à la suite de l'autre dans votre terminal et que vous tapez **historique** (la commande qui sera utilisée pour l'historique!), alors les 10 commandes seront visibles dans le fichier *historique.txt* et seules les commandes 13 à 20 seront visibles à l'écran. La numérotation des commandes doit évoluer. Par exemple, si vous avez entré moins de 100 commandes, elles seront toutes affichées et numérotées de 1 à 100. Si vous avez, par exemple entré 110 commandes, alors les 100 dernières seront affichées 10 à 110. Même chose pour l'écran. Si vous entrez moins de 8 commandes alors elles sont numérotées 1 à 8, mais si vous en entrez 10 alors elles sont numérotées 3 à 10.

Lorsque le processus enfant est créé afin d'exécuter la commande entrée, son pid doit aussi être conservé dans le fichier *historique.txt*.

Voici un exemple de *historique.txt* (délimiter vos colonnes avec un \t):

20	ls -l	25000
19	ls	12685
18	pwd	63325
17	mv	16789
16	cat	36154
15	ls -l	26458
14	man	69445
13	mkdir	65977
12	pwd	94856
11	ls	14598

À l'écran, vous verriez :

```
20    ls -l    25000
19    ls      12685
18    pwd     63325
17    mv      16789
16    cat     36154
15    ls -l    26458
14    man     69445
13    mkdir   65977
```

Lorsque vous entrez la commande **stop** alors le terminal se ferme et le programme se termine.

Afin de valider votre code, vous devez lancer votre code 3 fois :

- Instance 1 : 100 commandes, lancer **historique** à 50 et à 100 commandes entrées
- Instance 2 : 500 commandes, lancer **historique** à toutes les 100 commandes
- Instance 3 : 1000 commandes, lancer **historique** à toutes les 150 commandes

Le comportement de **historique** est décrit plus haut.

Attention : ne pas générer les instances manuellement, vous pouvez vous créer un script pour les générer automatiquement de façon aléatoire!!!

Seules les commandes au Tableau 2 sont acceptées. Utilisez les options! Veuillez conserver vos fichiers de sortie en les nommant : historique1_50.txt, historique 1_100.txt, historique2_100.txt, historique2_200.txt, etc... (historique<#instance>_<#commande>). De plus, vous devez générer un fichier pour chaque instance qui contient toutes les commandes (1 fichier de 100 commandes, 1 fichier de 500 commandes et 1 fichier de 1000 commandes).

Évidemment, vous devez aussi afficher à l'écran les 8 dernières commandes (et votre fichier historique devrait contenir au maximum 100 entrées).

Faites des captures d'écran de l'affichage au terminal afin d'insérer le tout dans votre rapport.

Rapportez le temps d'exécution de vos trois instances. N'oubliez pas de bien expliquer vos instances dans le rapport.

Remise et barème

Les parties 1 à 3 doivent être complétées, et les réponses aux questions données dans le rapport.

Les parties 4 et 5 doivent être remises sous forme de rapport et en démonstration au chargé de travaux dirigés. Un gabarit de laboratoire est disponible sur le site du cours. Vous devez aussi remettre une copie de votre code.

À remettre (voir template sur Moodle)

Vous devez préparer un rapport de devoir comportant :

- Page-titre!
- Introduction générale du laboratoire #1. Expliquer le but du laboratoire et ses deux parties dans vos mots, et formulez une introduction claire et concise.
- Vous devez répondre à chacune des questions numérotées Q1 à Q14 de la partie I en donnant le plus de détails possibles. Je ne veux pas juste les réponses!
- Vous devez expliquer votre démarche pour la réalisation de la partie II (logiciels utilisés, librairies, synchronisation, etc.) dans une section méthodologie.
- Une section résultats et discussion, qui détaille les générations d'instance, les résultats obtenus ainsi que les difficultés rencontrées.
- Une conclusion.
- Le code doit être commenté.
- Vous devez remettre une copie électronique de votre travail.

Le rapport est corrigé sur 114 points.

Partie 1 : 14 points

Partie 2 : 60 points

- Terminal avec noms des étudiants
- Lecture des commandes
- Fork() et exécution avec `execvp`
- & pris en charge
- Historique pris en charge
- Fonctionnement de l'historique
- Stop pris en charge et arrêt du programme
- Gestion des erreurs

Rapport : 10 points

Commentaires dans le code : 10 points

Exécution du programme devant le chargé de TD: 20 points