Parallel and Concurrent Programming - CS3D666 - Hour 1

- Part 1: Recap/Getting started with GNU/Linux
  - firstly install vmware with GNU/Linux or the Raspberry Pi-4 version of GNU/Linux see the GM Material link for support videos on how to install GNU/Linux.

Part 2: Investigating processes

- launch a terminal window, which can be found in the applications, you may need to move your cursor to the top right corner to activate the search bar. Or on a raspberry pi 4 it should be visible.
- In terminal window:
- Type `ls` to display contents of current directory
- `cd directory_name` to navigate to directory (note tab can be used to autocomplete).
- `cd ..` to navigate up a level from the current directory
- `ps -ef` to display all processes, along with PID, PPID (this will make sense later).
- `ps -ef | grep username` to display all processes for the specified username (more on this later).

Part 2: Writing and Compiling Code

- To open a file type: `gedit filename`. For example:
-

```
$ gedit hello_world.cpp
```


Part 2: Writing and Compiling Code

- Type in your code to the editor and save. Example:
- **hello_world.cpp**

```
#include<iostream>
#include<unistd.h>

int main()
{
   std::cout << "Hello World" << std::endl;
   return 0;
}
```

Part 2: Writing and Compiling Code

- In the above example we need iostream to be able to output to the terminal window and unistd.h for access to POSIX operating API.
- This header will generally be used in all work we do as it allows access to the Linux OS commands we need to use processes, threads etc.
- Note that in the above example we donât actually need to include it.

Part 2: Writing and Compiling Code

- Save the file and open a new terminal window and type: `g++ -o executableName FileName`, for example:
-
```
$ g++ -o HelloWorld hello_world.cpp
```


Part 2: Writing and Compiling Code

- Make sure you remember to save your file in gedit before attempting to compile it!
- To run the executable type `./executableName`, for example:
-
```
$ ./HelloWorld
```


Part 3: Getting Started

- 1. Research what processes are in Linux, what represents a process? How is a process identified? How does a process get created?
- 2. Given your understanding of processes type in `ps`âef what does `pid` and `ppid` represent? Can you draw the process tree for your system.
- 3. Investigate the fork command. What arguments does it return? What does it do?
- 4. Write a simple program that uses the fork command and examines the returned result to either print:
  - `I am the parent process.` or `I am the child process.`

Part 3: Getting Started

- 5. Write a program that calls the fork command twice in a row. Modify the program in 4 to print out what relative each process is. First start by identifying what relatives you are expecting. Hint: you will need to store the returned value from each call to fork.

- 6. If you call fork n times in a row, what information can you programmatically print about each thread?