ECU178 Computer Science:
207SE - Operating Systems, Security and Networks
Coursework

Due on March 16th 2015

Robert Rigler : 4939377

Contents

Week 12: Multitasking vs Multiprogramming

In this task I am going to be comparing two different types of process scheduling: Multitasking, and Multiprogramming. I will look into what they are, their differences and their similarities.

Multiprogramming

Definition: A way of scheduling processes to maximise CPU usage by switching processes that are 'waiting' for I/O, it ensures that the CPU is never idle.

Much older systems, unlike modern computers were very expensive and slow and often, when a process needed to use a peripheral device It often meant that the CPU was sitting idle for a long period of time. The solution to this is 'batch processing'.
Multiprogramming allows a computer to do several tasks at the same time. When a group of processes are marked 'Ready' for execution they are placed in a queue in main memory. The first process from this queue is then loaded into the CPU and is executed. There may come a time when this process is interrupted because It needs I/O to continue. At this point the process changed to a 'waiting' state. The process is then swapped out of the CPU into the I/O queue, and the next process in the 'Ready Queue' is swapped into the CPU. When the I/O request of the first process is completed, it is then placed back into the 'Ready queue'. This cycle continues until there are no jobs to be processed.

Multitasking

Definition: A logical extension of Multiprogramming, it involves rapidly switching between processed in the 'Ready state' to give the impression that they are all running simultaneously.

In Multiprogramming, processes are executing one at a time, in the order that they are placed into the ready queue. This means that only one process can be actively used at a time. Similarly in multitasking, processes are executed individually, but ther is also a certain level of concurrency; Because once a process has used it allotted processing time, It is swapped back into main memory.
This is beneficial, because with multiprogramming, a process has complete control over the CPU until an interrupt is called. There may be a situation where a process does not call an interrupt and takes a long time to finish processing. This will cause shorter, more time efficient or more important processes to be delayed until the first process is finished.

Week 14: Process Manipulation & Nohup

Process Manipulation

For this task I will look into the different ways to manipulate a process, and show examples of how to use each command.

Command	**Description**
command	Type the name of the process to start it
command &	Start the process in the background (symbolised by the & symbol)
ps -au	Shows all the processes currently running on the machine
ps -ux	Shows all the processes currently running owned by the current user
jobs	Shows the processs that are currently suspended.
CTRL - C	Kills the process running in the foreground
kill -9 x	Kills the process with the PID *x*
kill %1	Kills the process with job number *1*
CTRL - Z	Susoends the process curently running in the foreground.
kill -cont %1	Continues the execution of suspended job %1
bg %1	Pushes job number 1 to to the background
fg %1	Pushes job number 1 to to the foreground

In the pages below, I will show two scenarios in which I use all of these commands. You will find a snippet of terminal code and an explanation of each step that was taken.

Listing 1: Scenario 1

```
Script started on Thu 12 Mar 2015 14:58:19 GMT
rob@rob-HP-ProBook-6470b:$ xclock
^Z
[1]+  Stopped                 xclock
rob@rob-HP-ProBook-6470b:$ jobs
[1]+  Stopped                 xclock
rob@rob-HP-ProBook-6470b:$ fg %1
xclock
^C
rob@rob-HP-ProBook-6470b:$ exit
exit

Script done on Thu 12 Mar 2015 14:59:23 GMT
```

This typescript recording shows how I:

1. Starting the process *xclock* in the foreground,
2. Suspending *xclock* via CTRL-z,
3. Bringing *xclock* back to the foreground using *fg %1*
4. Finally Killing the process with CTRL-C

Listing 2: Scenario 2

```
1
2  rob@rob-HP-ProBook-6470b:$ xclock &
3  [1] 21811
4
5  rob@rob-HP-ProBook-6470b:$ xclock
6  ^Z
7  [2]+  Stopped                 xclock
8
9  rob@rob-HP-ProBook-6470b:$ jobs
10 [1]-  Running                 xclock &
11 [2]+  Stopped                 xclock
12
13 rob@rob-HP-ProBook-6470b:$ kill %1
14 rob@rob-HP-ProBook-6470b:$ jobs
15 [1]-  Terminated              xclock
16 [2]+  Stopped                 xclock
17
18 rob@rob-HP-ProBook-6470b:$ kill -cont %2
19 rob@rob-HP-ProBook-6470b:$ jobs
20 [2]+  Running                 xclock &
21
22 rob@rob-HP-ProBook-6470b:$ ps au |grep rob
23 USER     PID    %CPU %MEM VSZ    RSS  TTY      STAT START   TIME COMMAND
24 rob      16721  0.0  0.0  27336  4448 pts/5    Ss   14:26   0:00 /bin/bash
25 rob      21794  0.0  0.0  21892   960 pts/5    S+   15:47   0:00 script -a com2.txt
26 rob      21795  0.0  0.0  21896   396 pts/5    S+   15:47   0:00 script -a com2.txt
27 rob      21796  0.0  0.0  27224  4180 pts/15   Ss   15:47   0:00 bash -i
28 rob      21813  0.0  0.0  70556  4840 pts/15   S    15:47   0:00 xclock
29 rob      21872  0.0  0.0  22648  1320 pts/15   R+   15:48   0:00 ps au
30
31 rob@rob-HP-ProBook-6470b:$ kill -9 21813
32 rob@rob-HP-ProBook-6470b:$ jobs
33 [2]+  Killed                  xclock
34
35 rob@rob-HP-ProBook-6470b:$ exit
36 exit
37
38 Script done on Thu 12 Mar 2015 15:48:38 GMT
```

1. Starting the *xclock* process in the background,

2. Starting another *xclock* process in the foreground,

3. Suspend the *xclock* foreground process using CTRL-Z,

4. Use the Jobs Keyword to show the two *xclock* processes,

5. Kill the first *xclock* job using kill %1

6. Continue the second *xclock* process in the foreground using kill -cont %2

7. Show a list of my running processes using ps -au — grep rob

8. Finally Kill the remaining *xclock* process by using kill -9 21813

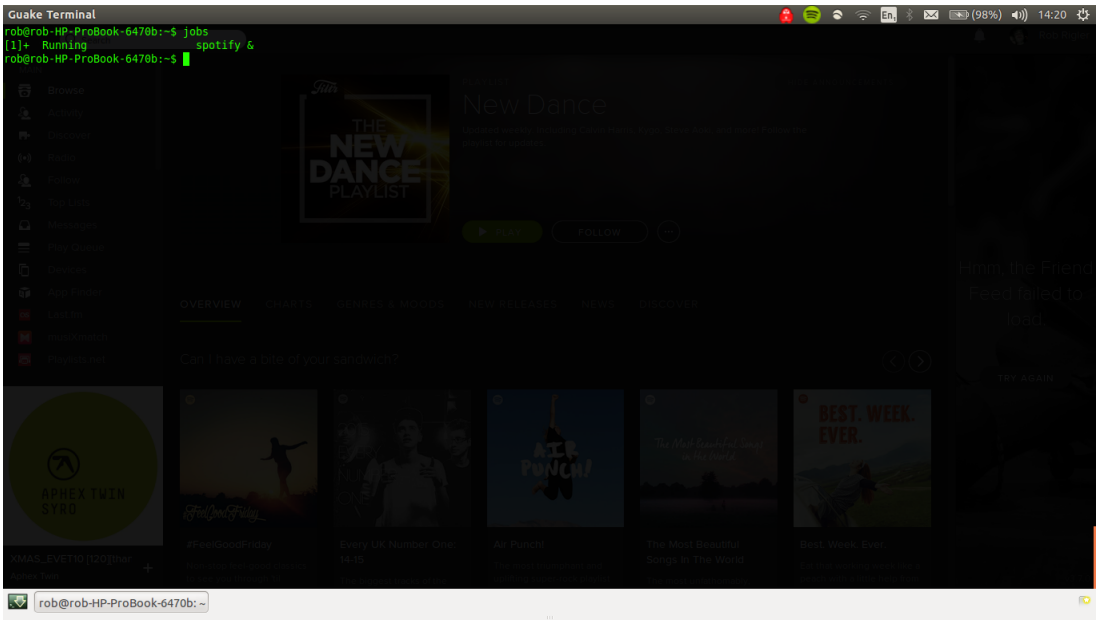
Nohup

Definition : A command which allows a process to continue executing after the parent process has been stopped.

'Nohup' means 'No Hang Up'. Commands that are executed with 'nohup', ignore hang up signals, so that the user can log out of the terminal and the process will still be running in the background. When a process is run in the foreground (no &), it effectively blocks the use of the shell whilst that process is being executed. When a process is run in the background (with &), it is placed into the list of background jobs that the shell is managing, but it is still connected to that shell, so if the shell closes, the process is terminated. NOHUP effictively separates the command from the shell, allowing it to close and the process to continue.

In this example, I am going to use the 'spotify' process as an example.

If I type 'spotify &', the spotify application is started and is run in the background, allowing to continue using the shell. When I use the 'jobs' command , we can see that the spotify process is running in the background.



When I exit the shell, the 'spotify' application also closes.

Now, if I type 'nohup spotify &' and look at the terminal jobs. It shows 'nohup spotify & ' , but now if I type exit, the application will stay open regardless of the terminal.