**Nyeinchan Kyaw**

**Homework 4.**

**CS-340**

**Part 1**

**UNIX-PROCESSES**

Processes and Job Control

1. **From the UNIX handout, read and cover the examples given in subchapters:**

**18. Foreground and Background Processes (fg, bg, suspend)**

[kyny1670@venus ~]$ ls -l >dirlist &

[1] 32733

**Foreground**

The fg command allows us to bring a background process to the foreground.

Syntax: fg [%jobid]

For example:

[kyny1670@venus ~]$ fg %1

**Suspend**

Suspend command allows us to go back to shell, by suspending the process and to return back to the suspended process.

Typing the command PS and VI , in edit mode, typing the following..

#include<stdio.h>

#define SIZE 100

main (int argc, char \*argv[])

{

...

[kyny1670@venus ~]$ ps

PID TTY TIME CMD

1030 pts/12 00:00:00 tcsh

1058 pts/12 00:00:00 ps

[kyny1670@venus ~]$ vi

(when Esc key and ctrl Z are pressed, the process gets suspened)

Suspended

Checking the processes and list of suspended jobs…

[kyny1670@venus ~]$ ps

PID TTY TIME CMD

1030 pts/12 00:00:00 tcsh

1063 pts/12 00:00:00 vim

1212 pts/12 00:00:00 ps

[kyny1670@venus ~]$ jobs

[1] + Suspended vim

**Background**

bg command allows us to send a job (that is suspended) in the background.

For example, if I type bg command now, it will show that the suspended job that just got sent to the background.

[kyny1670@venus ~]$ bg

[1] vim &

A daemon is a system process running in the background. Daemons are used in UNIX to offer

various types of services to users and handle system administration tasks. The print, e-mail, and

finger services are provided via daemons.

**19. nice command**

\*To understand more about the nice command, look at the man page. Give the default value for adjustments.

[kyny1670@venus ~]$ man nice

NICE(1) User Commands NICE(1)

NAME

nice - run a program with modified scheduling priority

SYNOPSIS

nice [OPTION] [COMMAND [ARG]...]

DESCRIPTION

Run COMMAND with an adjusted niceness, which affects process schedul-

ing. With no COMMAND, print the current niceness. Nicenesses range

from -20 (most favorable scheduling) to 19 (least favorable).

-n, --adjustment=N

add integer N to the niceness (default 10)

--help display this help and exit

We can use nice command used to start process with modified scheduling priority / nicenesses. renice command is used to change the priority of a process that's already running.

**The default value for adjustments is +10.**

For example: we can run like this -> [kyny1670@venus ~]$ nice +10 pico vim

If we want to change the priority of process vim, we can set it by using renice as follows.

[kyny1670@venus ~]$ ps

PID TTY TIME CMD

1030 pts/12 00:00:00 tcsh

1063 pts/12 00:00:00 vim

2537 pts/12 00:00:00 ps

[kyny1670@venus ~]$ renice 13 1063

1063: old priority 10, new priority 13

Superuser can use the nice to increase the priority of a command by using a negative value.

We can type: ‘su’ command and give a password to change niceness to negative value if we are superusers.

**20. Abnormal termination of processes**

We can terminate a foreground process by pressing CTRL +C.

We can terminate a background process by bringing the process to the foreground by using the fg command and then pressing CTRL + C by using the kill command.

Here is the list of possible signals.

[kyny1670@venus ~]$ kill -l

HUP INT QUIT ILL TRAP ABRT BUS FPE KILL USR1 SEGV USR2 PIPE ALRM TERM STKFLT CHLD CONT STOP TSTP TTIN TTOU URG XCPU XFSZ VTALRM PROF WINCH POLL PWR SYS RTMIN RTMIN+1 RTMIN+2 RTMIN+3 RTMAX-3 RTMAX-2 RTMAX-1 RTMAX

Using a single kill command we can terminate a list of processes.

For example, we can use sure kill to terminate a process like this.

[kyny1670@venus ~]$ kill -9 1063 //1063 is PID of vim

[kyny1670@venus ~]$ ps

PID TTY TIME CMD

1030 pts/12 00:00:00 tcsh

3140 pts/12 00:00:00 ps

[1] + Killed vim

**B. More on these topics:**

**Sequential Execution**

Syntax: cmd1; cmd2;…;cmdN

Type: date; echo Hello, World!

**Parallel Execution**

Syntax: cmd1; cmd2;…;cmdN&

Type: date& echo Hello, World! & uname; who

Execution of date& who; whoami; uname; echo Hello, World!&

[kyny1670@venus ~]$ date& who

[1] 32091

Wed Mar 21 17:50:47 EDT 2012

peca4541 pts/1 2012-03-21 03:05 (pool-96-232-133-53.nycmny.fios.verizon.n

et)

siha5327 pts/2 2012-03-21 15:03 (cpe-98-14-56-130.nyc.res.rr.com)

maaa3649 pts/4 2012-03-21 16:48 (cpe-24-90-241-27.nyc.res.rr.com)

haji5602 pts/5 2012-03-21 15:40 (bsc.qc.cuny.edu)

siha5327 pts/6 2012-03-21 15:22 (cpe-98-14-56-130.nyc.res.rr.com)

fuda4432 pts/7 2012-03-21 16:59 (bsc.qc.cuny.edu)

jowi0675 pts/8 2012-03-21 17:14 (cpe-24-193-136-237.nyc.res.rr.com)

smma8111 pts/9 2012-03-21 16:15 (pool-96-224-226-175.nycmny.fios.verizon.

net)

zhde5284 pts/10 2012-03-21 17:33 (oct-ib200-x4.qc.ads)

kyny1670 pts/11 2012-03-21 17:44 (bsc.qc.cuny.edu)

brde1342 pts/12 2012-03-21 17:00 (ciscoasa.hewlett-woodmere.net)

lixu5937 pts/13 2012-03-21 17:33 (149.4.104.27)

argr4869 pts/15 2012-03-21 17:45 (pool-108-46-63-7.nycmny.fios.verizon.net

)

[1] + Done date

[kyny1670@venus ~]$ whoami

kyny1670

[kyny1670@venus ~]$ uname

Linux

[kyny1670@venus ~]$ echo hello

hello

[kyny1670@venus ~]$ echo Hello, World!&

Hello, World!

[1] 32108

[1] Done echo Hello, World!

**PART 2**

**Give an outline of the scheduling algorithms currently used in Windows XP, Unix (Linux) and MacOS.**

Windows Xp schedules threads using a priority-based, preemptive scheduling algorithm with Windows XP scheduler to ensure that the highest-priority thread will always run. The dispatcher which handles scheduling in Windows XP, uses a 32-level priority scheme to determine the order of thread execution. Priorities are divided into two classes such as variable class (which contains threads with priorities from 1 to 15) and real-time class (which contains threads with priorities from 16 to 31. The dispatcher also uses a queue for each scheduling priority and traverses the set of queues from highest to lowest until it finds a thread that is ready to run. Idle thread will be executed if no ready thread is found. There is a relationship between the numeric priorities of the Windows XP kernel and the Win32 API which identifies several priority classes to which a process can belong. The values for relative priority include 1. Time-critical 2. Highest 3. Above normal 4. Normal 5. Below normal 6. Lowest 7. Idle. Processes are typically members of the Normal priority class. A process will belong to this class unless the parent of the process was of the Idle priority class or unless another class was specified when the process was created. Windows Xp has a special scheduling rule for processes in the Normal priority class.

**Unix (Linux)**

In Linux version 2.5, the scheduler was overhauled, and the kernel now provides a scheduling algorithm that runs in constant time O (1). The new scheduler also provides increased support for SMP, including processor affinity and load balancing, as well as proving fairness and support for interactive tasks. Linux scheduler is a preemptive, priority-based algorithm with two separate priority ranges such as 1. Real-time (ranging from 0 to 99) and 2. Nice value (ranging from 100 to 140); these both ranges map into a global priority scheme whereby numerically lower values indicate higher priorities. Unlike Windows XP scheduling algorithm, Linux assigns higher-priority tasks longer time quanta and lover-priority tasks shorter time quanta. The kernel maintains a list of all runnable tasks in a runqueue data structure. Each runqueue contains two priority arrays such as 1. Active and 2. Expired. The active array contains all tasks with time remaining in their time slices, and the expired array contains all expired slices. Real-time tasks are assigned static priorities and all other tasks have dynamic priorities based on their nice values plus or minus the value 5. A task’s interactivity is determined by how long it has been sleeping while waiting for I/O. Tasks that are more interactive have longer sleep times and are more likely to have adjustments closer to -5, as the scheduler favors interactive tasks which will be higher priorities. Tasks with shorter sleep times are often more CPU-bound and thus will have their priorities lowered.

**MacOS**

Mac OS 9, is the final major release of Apple's Mac OS before the launch of OS X, uses cooperative scheduling for threads, where one process controls multiple cooperative threads, and also provides preemptive scheduling for MP tasks. The kernel schedules MP tasks using a preemptive scheduling algorithm. However, scheduling algorithms currently used in Mac OSX is different from Mac OS 9. Mach threads represent the lowest level threading on the system. Mac OS X uses a multilevel feedback queue, with four priority bands of threads such as 1. Normal, 2. System high priority, 3. Kernel mode only, and 4. Real-time. Threads are scheduled preemptively. CFS scheduler has a scheduling complexity of O(log N), where N is the number of tasks in the runqueue. Choosing a task can be done in constant time, but reinserting a task after it has run requires O(log N) operations, because the run queue is implemented as a red-black tree. CFS is the first implementation of a fair queuing process scheduler widely used in a general-purpose operating system.