ICS 2305 : SYSTEMS PROGRAMMING

PROCESS MANAGEMENT

Karanja Mwangi

Lesson objective

- **At the end of this class you will**
  - Appreciate the concepts of process management as used in Operating systems
  - Identify various commands and constructs used in Process Management in Linux
  - Explore the shell programming aspects in Process Management

Recap: Processes

- **Main Role of OS in processes :** is the *creation*, *management*, and *termination* of processes
- process?
  - a program under execution,
  - an identifiable entity executed on a processor by the operating system.
- In OS terms:
  - as an executable instance of a program,
  - as the associated data operated upon by the program (variables, temporary results, external (file) storage, ...), and as the program's *execution context*.

Recap: Processes (View Points)

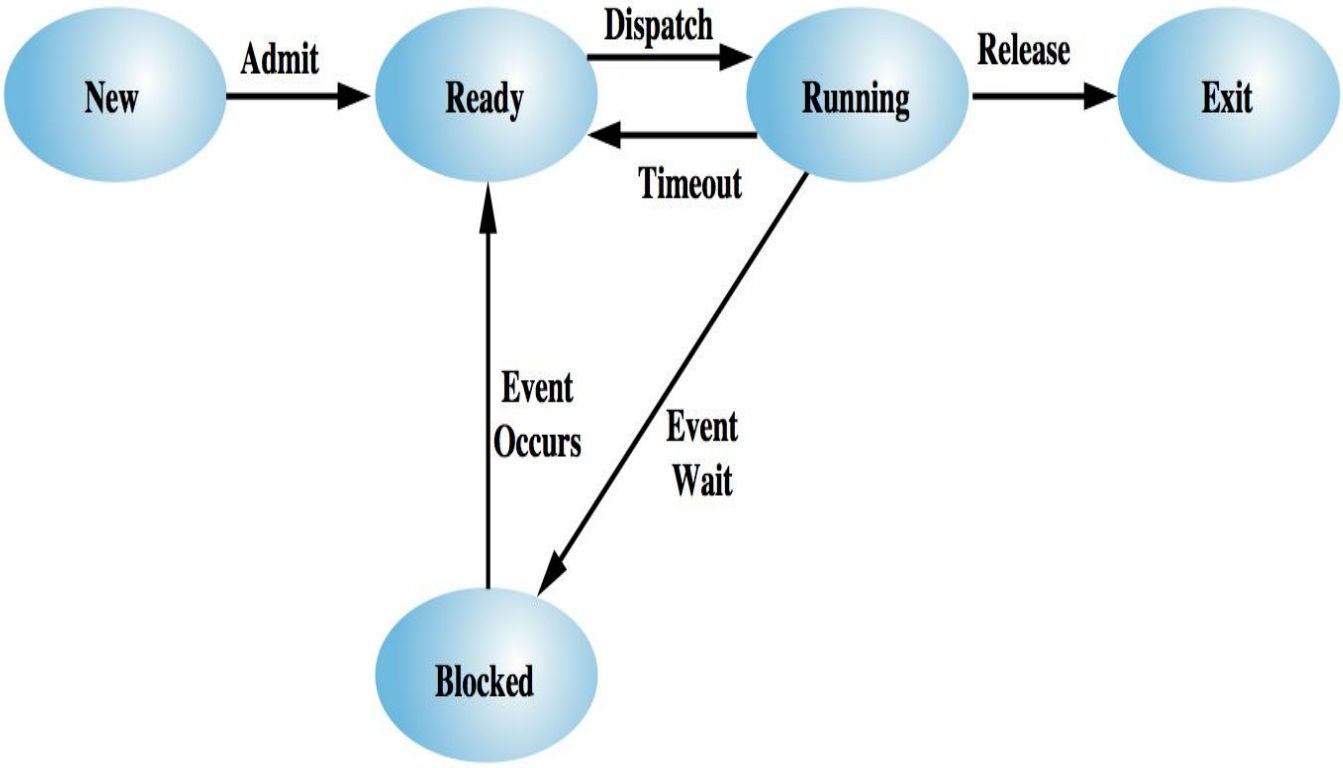
- The process from two points of view: that of the **processor**, and that of **the process itself**.
- **The processor's view** : the processor's only role is to execute machine instructions from main memory. Over time, the processor continually executes the sequence of instructions indicated by the program counter (PC). The processor is unaware that different sequences of instructions have a logical existence, that different sequences under execution are referred to as processes or tasks.
- **process's point of view**, it is either being executed by the processor, or it is waiting to be executed

Recap:
Process Transitions{Process Creation}

- For a **new process**, the OS allocate resources both for the process and the OS itself.
- Ways to create new processes
  - OS taking new request from Batch queue
  - a user logging on at a terminal usually creates an interactive control or encapsulating process (shell or command interpreter).
  - an existing process may request a new process,
  - the operating system itself may create a process after an indirect request for service (to support networking, printing, ...) etc
- Different operating systems support process creation in different ways. (**fork()** call in macOS and Linux);by instantiating a process's image from a named location, the **CreateProcess()** call in Windows) etc

Recap:
The 5-State Model of Process Execution

- The William Stallings 5-state model



Recap:
Process Transitions{Five States 1 ….}

- Why Moving from **Ready to Running states**:
  - Mainly for fairness amongst processes and maximal use of resources
  - This is achieved via **Timer Interrupts**
    - A *hardware timer* will periodically generate an interrupt (say, every 5 milliseconds). Between the execution of any two instructions, the processor will "look for" interrupts. When the *timer interrupt* occurs, the processor will begin execution of the *interrupt handler*.
    - The handler will increment and examine the accumulated time of the currently executing process, and eventually move it from **Running** to **Ready**.
    - **A processes have a** *time quantum* **(Max time a process is allowed to execute)**

Recap:
Process Transitions{Five States -2 ….}

The **Blocked state ?**

- It is hard or rare to have **compute-bound processes(**processes which continually execute to the end of their time quanta) ….
- E.g. If a process request an input or output from slow read and write source it has to get to state which is **not ready or running** waiting for I/O requests to be satisfied
- the process is moved from Running to Blocked and when I/O interrupt occurs the process moves from blocked to ready

Recap:
Process Transitions{Five States -3 ....}

The **Termination ?**

(William Stallings book)

- normal termination,
- execution time-limit exceeded,
- a resource requested is unavailable,
- an arithmetic error (division by zero),
- a memory access violation,
- an invalid request of memory or a held resource,
- an operating system or parent process request, or
- its parent process has terminated.

Swapping? resources such as memory are finite

When none of the processes in main memory is **Ready**, the operating system swaps the memory of some of the **Blocked** processes to disk to recover some memory space. Such processes are moved to a new state: the **Suspend** state, a queue of processes that have been "kicked out" of main memory:

Recap:
Process Transitions{Summary of states}

Null → New	a new process is requested.
New → Ready	resources are allocated for the new process.
Ready → Running	a process is given a time quantum.
Running → Ready	a process's execution time quantum expires.
Running → Blocked	a process requests slow I/O.
Blocked → Ready	an I/O interrupt signals that I/O is ready.
Running → Exit	normal or abnormal process termination.
Ready or Blocked → Exit	external process termination requested.

Process Management in Linux

Linux basics -1

- In Linux/unix a program is identified by its process ID (PID) as well as it's parent processes ID (PPID). There are two types of processes
  - Parent processes – these are processes that create other processes during run-time.
  - Child processes – these processes are created by other processes during run-time.
- **The init Process** mother (parent) of all processes on the system, it's the first program that is executed when the boots up; it manages all other processes on the system. It is started by the kernel itself, so in principle it does not have a parent process.

  - The init process always has process ID of 1. It functions as an adoptive parent for all orphaned processes.

  Daemons : type of background processes that start at system startup and keep running forever as a service; they don't die. They are started as system tasks (run as services), spontaneously. However, they can be controlled by a user via the init process

  States of a Process in Linux (Running , waiting, stopped and Zombie{dead or halted process})

Linux basics -2

- **#** - for root privileges or use of **sudo** command
- **$** - regular non-privileged user

Few command to exercise on (Lets do this now)

$ ps - shows the snapshot of all programs in execution

sudo ps –a ps –A
sudo ps -U karanja

There are too many outputs so we can do more or less

ps -aux | more
sudo ps -aux | less

pidof mysqld will get the pid of the mysqld program
pgrep mysql

ps aux | grep mysqld …..better output of the same

Linux basics -3

- There are Foreground and Background Processes in **Linux**
- **Foreground like user apps such as Libre office etc , background such as antivirus etc**
- **To start a foreground process just type in the terminal or search it on the dashboard**

bg	To send a process to the background
fg	To run a stopped process in the foreground

…..Open a program say Vlc from the terminal …..bg vlc
(**of course it will be a foreground process make it background)**
.

Linux basics -4 Controlling processes

- using Commands such as kill, pkill, pgrep and killall

$ kill –l

- SIGHUP 1 – sent to a process when its controlling terminal is closed.
- SIGINT 2 – sent to a process by its controlling terminal when a user interrupts the process by pressing [Ctrl+C].
- SIGQUIT 3 – sent to a process if the user sends a quit signal [Ctrl+D].
- SIGKILL 9 – this signal immediately terminates (kills) a process and the process will not perform any clean-up operations.
- SIGTERM 15 – this a program termination signal (kill will send this by default).
- SIGTSTP 20 – sent to a process by its controlling terminal to request it to stop (terminal stop); initiated by the user pressing [Ctrl+Z].
- Run this to see all options **$ kill -l**

Kill using process ID

$ kill 9 5717 where 9 is the SIGKILL 9 and 5717 is the Process ID

$ kill PID1 PID2 PID3 to kill many processes

Kill using process name

$ pkill firefox

$ killall firefox

Linux basics -5 Controlling processes

- using Commands such as kill, pkill, pgrep and killall notes

- A user can kill all his process.
- A user can not kill another user's process.
- A user can not kill processes System is using.
- A root user can kill System-level-process and the process of any user.

- **$ pgrep process**

Such as **$ pgrep vlc**

Linux basics -6 Controlling processes

- Changing linux programs priorities
- **We know that -** higher priority will normally get more CPU time than lower priority processes
- the process have to wait for other processes to free resources before it can access them
- **The less the value, the highest the priority of the process**. Values ussually between -20 to 19

nice and renice commands are used

First we run $ top ....its variants are

$htop
$sudo htop
$sudo htop [options]

Then .....

$ nice -n 19 ./script.sh

$ renice +8 5717

Good tools to have : dstat or sysstat

- For process Monitoring
- **Install it :** sudo apt-get install dstat
- Use the tool **glances …**# **glances**

- **Better if you get** sysstat
  - # wget http://pagesperso-orange.fr/sebastien.godard/sysstat-11.0.0.tar.gz

  Configure it
  - # tar -xvf sysstat-11.0.0.tar.gz
  - # cd sysstat-11.0.0/

  - **For more** http://sebastien.godard.pagesperso-orange.fr/download.html

Read on

- **Using C programming**
  - **Creating a new process using *fork() system call***
  - **CREATING multiple processes using fork () and pipe()**
  - **Demonstrate the concept of Orphan Process.**
  - **Using execv and execvp system calls**

  - **Good source :**https://www.usna.edu/Users/cs/aviv/classes/ic221/s16/lec/14/lec.html#orgheadline7
- **You can use this to get a basic appreciation**
  - https://home.deib.polimi.it/fornacia/lib/exe/fetch.php?media=teaching:aos:2016:aos201617_multiprocess_programming_updated20161223.pdf
  - https://linuxhint.com/fork_linux_system_call_c/
  - https://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/fork/create.html
  - https://www.geeksforgeeks.org/fork-system-call/
  - https://www.geeksforgeeks.org/zombie-and-orphan-processes-in-c/