horizontal line

**Assignment 04 | Operating System**

**CE-092**

Assignment submission for Operating System subject week 4.

nevilparmar24@gmail.com

**─**

**PS Command -** report a snapshot of the current process

PS is a command line utility that is used to display or view information related to processes running in a linux system.

The version of ps accepts several kind of options :

1. UNIX - which may be grouped and must be preceded by a dash
2. BSD - which may be grouped and must not be used with a dash
3. GNU - which are preceded by two dashes

It retrieves information about the processes from the virtual files which are located in the /proc file system.

The output consists of 4 columns :

PID – This is the unique process ID

TTY – This is the typeof terminal that the user is logged in to

TIME – This is the time in minutes and seconds that the process has been running

CMD – The command that launched the process

**Several Options :**

**Ps -a :** displays all the running processes in the system

**Ps -e :** Selects all the processes , identical to -a

**Ps aux :** used to see every processes in the system using BSD syntax

**Ps -ejH :** to see process tree of the current running processes

**Ps -elF :** to get info about the threads

**Parked Process State:**

When the kernel cannot (for some reason) service the request , then the process goes into the parked queue. Say, for instance, the process asked the kernel to read a particular disk block (as in a file read), but the disk controller has failed. The kernel cannot release the process back to it's runnable state until the read completes, and the read *never* completes. So, the process stays in the 'parked' queue until the read request completes, and sometimes this means that it stays there until the hardware or software error that prevented the request completion is repaired.

A process residing inside the parking queue is not runnable. Thus, a parked process cannot receive signals. Until you clear the problem that’s keeping the process in the parked state, the process can not receive the signals. The only way is to clear the condition that's keeping the process in the parked state. With that, the process can then move to ‘runnable’ and onwards to actually be run.

**Wake Kill Process State:**

It is designed to wake the process on receipt of fatal signals.

In other words, TASK\_UNINTERRUPTIBLE + TASK\_WAKEKILL = TASK\_KILLABLE .

**Waking Process State:**

We mark some of the processes as waking state. It guarantees that nobody will actually run it, and a signal or other external event cannot wake it up and insert it on the run queue either.

**Task 1-2:**

Write a program to print process id and process name of all the current processes in the system.

Extend the above program to read and display other fields from the stat file.

Code:

/\*

*\* @Author: nevil*

*\* @Date: 2020-07-31 14:25:06*

*\* @Last Modified by: nevil*

*\* @Last Modified time: 2020-07-31 14:28:46*

\*/

#*include*<stdio.h>

#*include*<dirent.h>

#*include*<sys/types.h>

#*include*<string.h>

#*include*<stdlib.h>

void *ps\_command*(int argc, char \*argv*[]*)

{

char mainPath[100];

DIR \*dirp = *opendir*("/proc/");

struct dirent \*d;

int pid;

char cmd[100];

char state;

int ppid;

char state1[20];

*if* (dirp == *NULL*)

{

*fprintf*(*stderr*, "can not open directory : %s\n", mainPath);

*return*;

}

// *formatting condition according to provided arguments*

*if* (argc == 2)

{

*if* (!*strcmp*(argv[1], "--more"))

*printf*("%6s %30s %10s %10s\n", "PID", "COMMAND", "PPID", "STATE");

}

*else*

*printf*("%6s %30s\n", "PID", "COMMAND");

*while* (d = *readdir*(dirp))

{

*if* (*strcmp*(d->d\_name, ".") != 0 && *strcmp*(d->d\_name, "..") != 0)

{

*if* (d->d\_type == *DT\_DIR* && *atoi*(d->d\_name) > 0)

{

// *path generation*

*strcpy*(mainPath, "/proc/");

*strcat*(mainPath, d->d\_name);

*strcat*(mainPath, "/stat");

FILE \*fp = *fopen*(mainPath, "r");

*if* (fp == *NULL*)

*continue*;

*switch* (state)

{

*case* 'R': *strcpy*(state1, "Running"); // *Running state*

*break*;

*case* 'S': *strcpy*(state1, "Intr Wait"); // *Interruptible wait state*

*break*;

*case* 'D': *strcpy*(state1, "Uninter wait"); // *Uninterruptible state*

*break*;

*case* 'Z': *strcpy*(state1, "Zombie"); // *Zombie state*

*break*;

*case* 'T': *strcpy*(state1, "Traced"); // *Traced State*

*break*;

*default*: *strcpy*(state1, "Paging"); // *Paging State*

}

*fscanf*(fp, "%d\t%s\t%c\t%d", &pid, cmd, &state, &ppid);

*printf*("%6d %30s ", pid, cmd);

*if* (argc == 2)

{

*if* (!*strcmp*(argv[1], "--more"))

*printf*("%6d %13s", ppid, state1);

}

*printf*("\n");

*fclose*(fp);

}

}

}

*closedir*(dirp);

}

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

{

*ps\_command*(argc, argv);

*return* 0;

}

/\*

*This program simulates the behaviour of PS command in linux.*

*You can run this program in two diff ways.*

*1) ./a.out*

*-> prints PID , and COMMAND NAME*

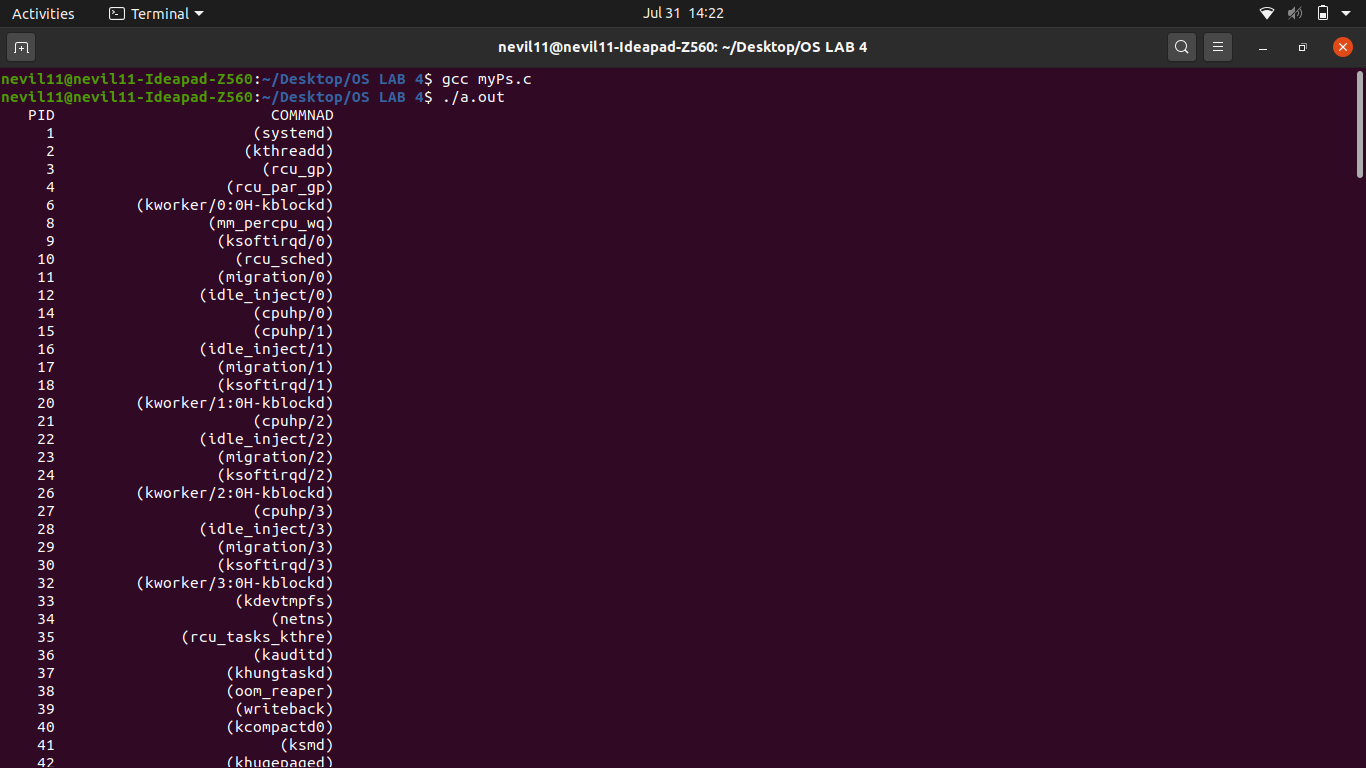
*2) ./a.out --more*

*-> prints PID, COMMAND, STATE AND PPID.*

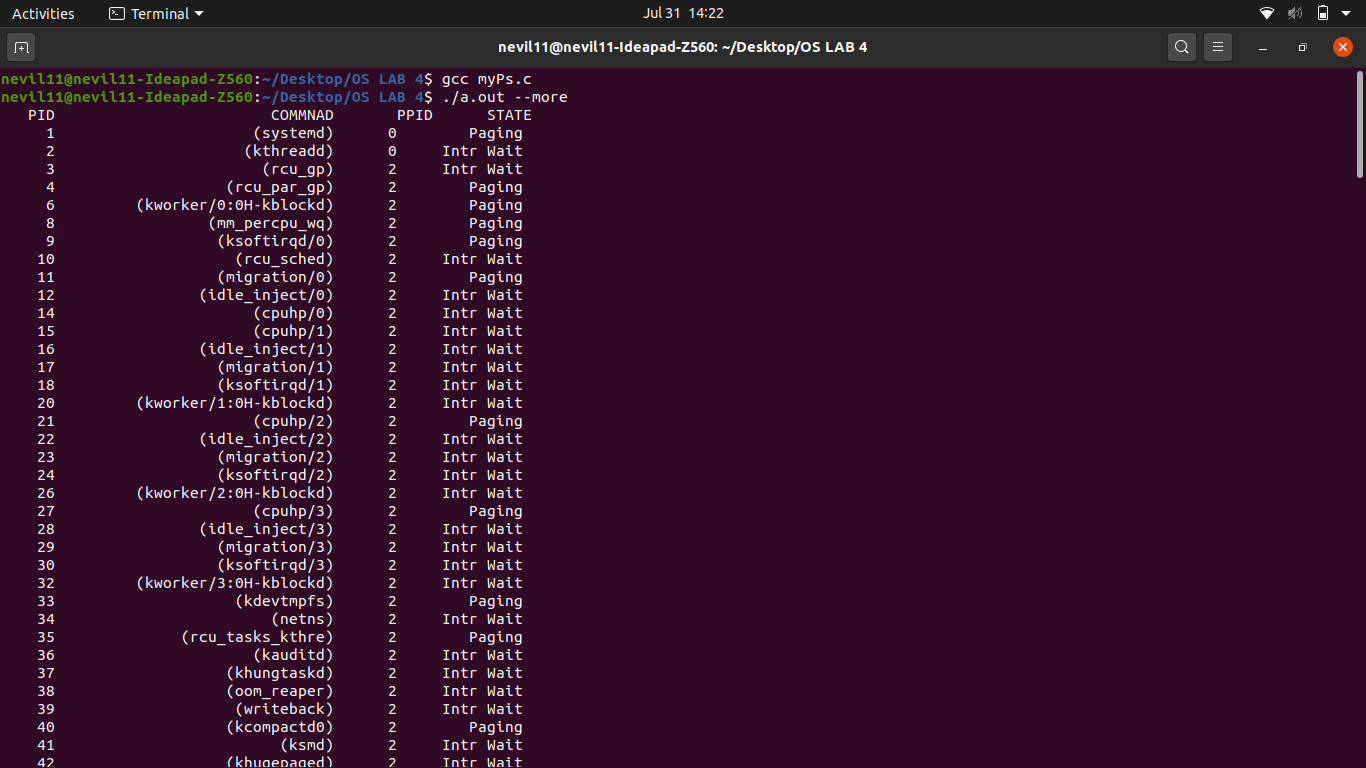
\*/

Output:

Without any command line arguments.



With argument --more.



Nevil Parmar

CE-092

https://nevilparmar.me