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
```

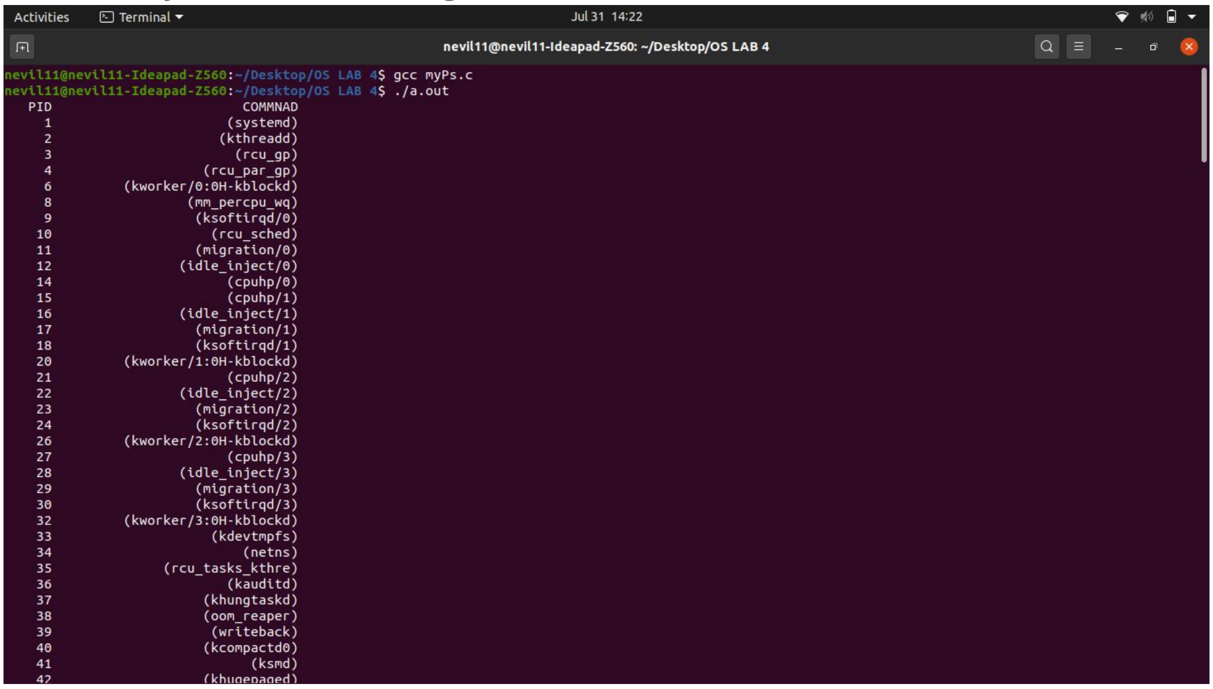
```
*/
#include<stdio.h>
#include<dirent.h>
#include<sys/types.h>
#include<string.h>
#include<stdlib.h>

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.
*/
```

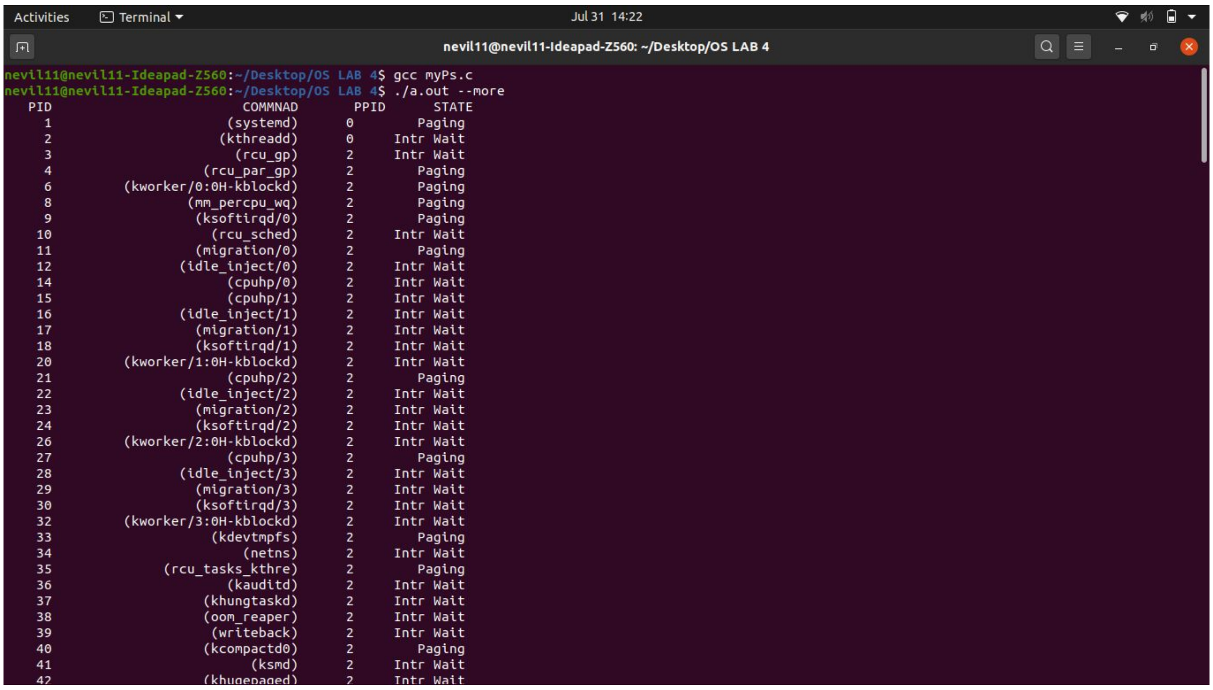
Output:

Without any command line arguments.



```
nevil11@nevil11-Ideapad-Z560:~/Desktop/OS LAB 4$ gcc myPs.c
nevil11@nevil11-Ideapad-Z560:~/Desktop/OS LAB 4$ ./a.out
   PID                        COMMNAD
     1                      (systemd)
     2                     (kthreadd)
     3                       (rcu_gp)
     4                   (rcu_par_gp)
     6          (kworker/0:0H-kblockd)
     8                 (mm_percpu_wq)
     9                  (ksoftirqd/0)
    10                    (rcu_sched)
    11                  (migration/0)
    12                (idle_inject/0)
    14                      (cpuhp/0)
    15                      (cpuhp/1)
    16                (idle_inject/1)
    17                  (migration/1)
    18                  (ksoftirqd/1)
    20          (kworker/1:0H-kblockd)
    21                      (cpuhp/2)
    22                (idle_inject/2)
    23                  (migration/2)
    24                  (ksoftirqd/2)
    26          (kworker/2:0H-kblockd)
    27                      (cpuhp/3)
    28                (idle_inject/3)
    29                  (migration/3)
    30                  (ksoftirqd/3)
    32          (kworker/3:0H-kblockd)
    33                    (kdevtmpfs)
    34                        (netns)
    35              (rcu_tasks_kthre)
    36                      (kauditd)
    37                   (khungtaskd)
    38                   (oom_reaper)
    39                    (writeback)
    40                   (kcompactd0)
    41                         (ksmd)
    42                   (khugepaged)
```

With argument --more.



```
nevil11@nevil11-Ideapad-Z560:~/Desktop/OS LAB 4$ gcc myPs.c
nevil11@nevil11-Ideapad-Z560:~/Desktop/OS LAB 4$ ./a.out --more
   PID                        COMMNAD       PPID      STATE
     1                      (systemd)      0        Paging
     2                     (kthreadd)      0     Intr Wait
     3                       (rcu_gp)      2     Intr Wait
     4                   (rcu_par_gp)      2        Paging
     6          (kworker/0:0H-kblockd)      2        Paging
     8                 (mm_percpu_wq)      2        Paging
     9                  (ksoftirqd/0)      2        Paging
    10                    (rcu_sched)      2     Intr Wait
    11                  (migration/0)      2        Paging
    12                (idle_inject/0)      2     Intr Wait
    14                      (cpuhp/0)      2     Intr Wait
    15                      (cpuhp/1)      2     Intr Wait
    16                (idle_inject/1)      2     Intr Wait
    17                  (migration/1)      2     Intr Wait
    18                  (ksoftirqd/1)      2     Intr Wait
    20          (kworker/1:0H-kblockd)      2     Intr Wait
    21                      (cpuhp/2)      2        Paging
    22                (idle_inject/2)      2     Intr Wait
    23                  (migration/2)      2     Intr Wait
    24                  (ksoftirqd/2)      2     Intr Wait
    26          (kworker/2:0H-kblockd)      2     Intr Wait
    27                      (cpuhp/3)      2        Paging
    28                (idle_inject/3)      2     Intr Wait
    29                  (migration/3)      2     Intr Wait
    30                  (ksoftirqd/3)      2     Intr Wait
    32          (kworker/3:0H-kblockd)      2     Intr Wait
    33                    (kdevtmpfs)      2        Paging
    34                        (netns)      2     Intr Wait
    35              (rcu_tasks_kthre)      2        Paging
    36                      (kauditd)      2     Intr Wait
    37                   (khungtaskd)      2     Intr Wait
    38                   (oom_reaper)      2     Intr Wait
    39                    (writeback)      2     Intr Wait
    40                   (kcompactd0)      2        Paging
    41                         (ksmd)      2     Intr Wait
    42                   (khugepaged)      2     Intr Wait
```

Nevil Parmar
CE-092
https://nevilparmar.me