

# Homework 2 - Operating Systems (ICS431)

Alfaifi, Ammar

## 1 Commands

### 1.1 ps

This shows processes status. without any options, it will show you a list of processes running in your current terminal session.

```
ammar-faifi@ammarf:~$ ps
  PID TTY          TIME CMD
 26494 pts/0        00:00:00 bash
 26684 pts/0        00:00:00 ps
ammar-faifi@ammarf:~$
```

### 1.2 ps -e

show all processes, not just those related to the current terminal session

```
ammar-faifi@ammarf:~$ ps -e
  PID TTY          TIME CMD
    1 ?           00:00:02 systemd
    2 ?           00:00:00 kthreadd
    3 ?           00:00:00 rcu_gp
    4 ?           00:00:00 rcu_par_gp
    5 ?           00:00:00 slub_flushwq
    6 ?           00:00:00 netns
    8 ?           00:00:00 kworker/0:0H-events_highpri
   10 ?           00:00:00 mm_percpu_wq
   11 ?           00:00:00 rcu_tasks_kthread
   12 ?           00:00:00 rcu_tasks_rude_kthread
   13 ?           00:00:00 rcu_tasks_trace_kthread
   14 ?           00:00:00 ksoftirqd/0
```

### 1.3 ps -u

will show only the processes started by the current user.

```
ammar-faifi@ammarf:~$ ps -u
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
ammar-f+  1471  0.0  0.1 159588  5456 tty2    Ssl+  08:53   0:00 /usr/libexec/gdm-wayland
ammar-f+  1474  0.0  0.3 222844 13788 tty2    Sl+   08:53   0:00 /usr/libexec/gnome-sess
ammar-f+  26494  0.0  0.1   8656  5396 pts/0    Ss   13:48   0:00 -bash
ammar-f+  26957  0.0  0.0   9808  1652 pts/0    R+   15:47   0:00 ps -u
```

## 1.4 ps -el

command will show the complete list of processes in a long format. It will include the process ID, parent process ID, user ID, group ID, virtual memory size, resident set size, CPU usage, start time, terminal, and command.

```
ammam-faifi@ammamf:~$ ps -el
F S  UID      PID     PPID  C  PRI  NI ADDR SZ WCHAN  TTY      TIME CMD
4 S   0         1         0  0   80   0 - 42097 -    ?      00:00:02 systemd
1 S   0         2         0  0   80   0 -    0 -    ?      00:00:00 kthreadd
1 I   0         3         2  0   60  -20 -    0 -    ?      00:00:00 rcu_gp
1 I   0         4         2  0   60  -20 -    0 -    ?      00:00:00 rcu_par_gp
1 I   0         5         2  0   60  -20 -    0 -    ?      00:00:00 slub_flushwq
1 I   0         6         2  0   60  -20 -    0 -    ?      00:00:00 netns
1 I   0         8         2  0   60  -20 -    0 -    ?      00:00:00 kworker/0:0H-events_highpri
1 I   0        10         2  0   60  -20 -    0 -    ?      00:00:00 mm_percpu_wq
1 I   0        11         2  0   80   0 -    0 -    ?      00:00:00 rcu_tasks_kthread
1 I   0        12         2  0   80   0 -    0 -    ?      00:00:00 rcu_tasks_rude_kthread
1 I   0        13         2  0   80   0 -    0 -    ?      00:00:00 rcu_tasks_trace_kthread
1 S   0        14         2  0   80   0 -    0 -    ?      00:00:00 ksoftirqd/0
1 I   0        15         2  0   80   0 -    0 -    ?      00:00:07 rcu_preempt
```

## 2 Zombie Processes

This code is to simulate a zombie process.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>

int main() {
    pid_t pid;
    pid = fork();

    if (pid < 0) {
        fprintf(stderr, "Fork failed.\n");
        exit(1);
    } else if (pid == 0) {
        printf("This is the child process.\n");
        exit(0);
    } else {
        printf("This is the parent process. Child's PID %d\n", pid);
        sleep(10);
        waitpid(pid, NULL, 0); // wait for child process to exit
        printf("Parent process exiting.\n");
        exit(0);
    }

    return 0;
}
```

From the following screenshots, the process with PID of 29296 is the **bash** itself. The running C program has PID of 33017, its parent is the **bash**. When **fork()** is called, a duplicate process from the parent (PID:33017) is created with PID of 33020. This child's PID is returned in the parent process, as seen in the first line in the second figure. When the child is finished but the parent is not, the child's process (33020) becomes a zombie process (see the status Z).

```

ammar-faifi@ammarf:~/Documents$ ps -l
F S  UID      PID      PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000      29296     29295  0  80   0 -  2262 do_wai pts/0        00:00:00 bash
0 R  1000      32943     29296  0  80   0 -  2452 -          pts/0        00:00:00 ps
ammar-faifi@ammarf:~/Documents$ ./a.out &
[1] 33017
This is the parent process. Child's PID 33020
This is the child process.
ammar-faifi@ammarf:~/Documents$ ps -l
F S  UID      PID      PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000      29296     29295  0  80   0 -  2262 do_wai pts/0        00:00:00 bash
0 S  1000      33017     29296  0  80   0 -   547 hrtimr pts/0        00:00:00 a.out
1 Z  1000      33020     33017  0  80   0 -    0 -          pts/0        00:00:00 a.out <defunct>
0 R  1000      33092     29296  0  80   0 -  2452 -          pts/0        00:00:00 ps
ammar-faifi@ammarf:~/Documents$ Parent process exiting.
ps -l
F S  UID      PID      PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000      29296     29295  0  80   0 -  2262 do_wai pts/0        00:00:00 bash
0 R  1000      33166     29296  0  80   0 -  2452 -          pts/0        00:00:00 ps
[1]+  Done                  ./a.out

```

### 3 Multithreaded Program

The codes in C is

```

#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

void *print_primes(void *arg) {
    int num = *(int *)arg;
    int i, j, flag;
    for (i = 2; i <= num; i++) {
        flag = 1;
        for (j = 2; j <= i / 2; j++) {
            if (i % j == 0) {
                flag = 0;
                break;
            }
        }
        if (flag == 1) {
            printf("%d ", i);
        }
    }
    printf("\n");
    pthread_exit(NULL);
}

int main(int argc, char *argv[]) {
    int num = atoi(argv[1]);

    pthread_t tid;
    pthread_attr_t attr;
    pthread_attr_init(&attr);

    pthread_create(&tid, &attr, print_primes, &num);
    pthread_join(tid, NULL);

    return 0;
}

```

In the following screenshot, I ran the program with an input value of 100. Next I ran it with very large number so I can watch its thread using `ps -lfl` command. I use `>` to pipe the `stdout` to the virtual device `/dev/null` to discard any output, then I use `&` to run it in background.

From the output of `ps`, there are two processes with same PID which indicated it's a multithreaded process. We see also the value of the thread ID (LWP). Also the CPU utilization (C) is large in the thread with value of 93.

```

ammar-faifi@ammarf:~/Documents$ ./prime 100
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
ammar-faifi@ammarf:~/Documents$ ./prime 1000000000 > /dev/null &
[1] 38065
ammar-faifi@ammarf:~/Documents$ ps -lfl
F S UID          PID     PPID    LWP  C  NLWP  PRI   NI  ADDR  SZ  WCHAN   STIME TTY          TIME CMD
0 S ammar-f+    29296    29295   29296  0    1   80    0 - 2262 do_wai 16:08 pts/0    00:00:00 -bash
0 S ammar-f+    38065    29296   38065  0    2   80    0 - 18995 futex_ 18:55 pts/0    00:00:00 ./prime 1000000000
1 R ammar-f+    38065    29296   38067 93    2   80    0 - 18995 -      18:55 pts/0    00:00:02 ./prime 1000000000
0 R ammar-f+    38140    29296   38140  0    1   80    0 - 2452 -      18:55 pts/0    00:00:00 ps -lfl
ammar-faifi@ammarf:~/Documents$ kill %1
[1]+  Terminated                  ./prime 1000000000 > /dev/null
ammar-faifi@ammarf:~/Documents$ ps
    PID TTY          TIME CMD
  29296 pts/0    00:00:00 bash
  38287 pts/0    00:00:00 ps

```