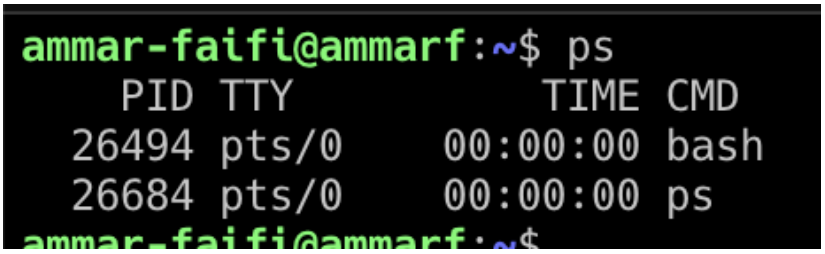
Homework 2 - Operating Systems (ICS431)

Alfaifi, Ammar

1 Commands

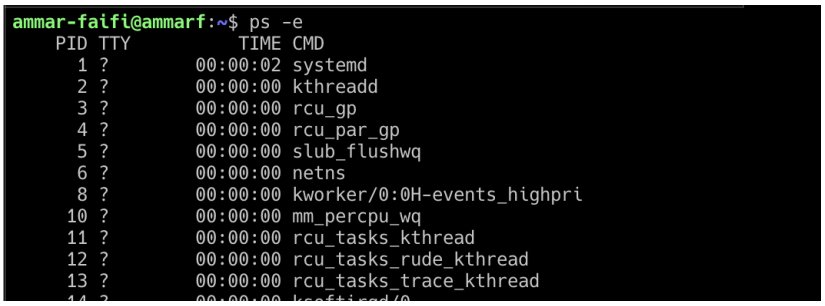
1.1 `ps`

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



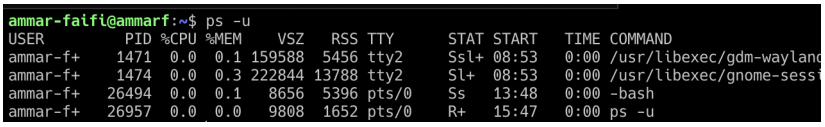
1.2 `ps -e`

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



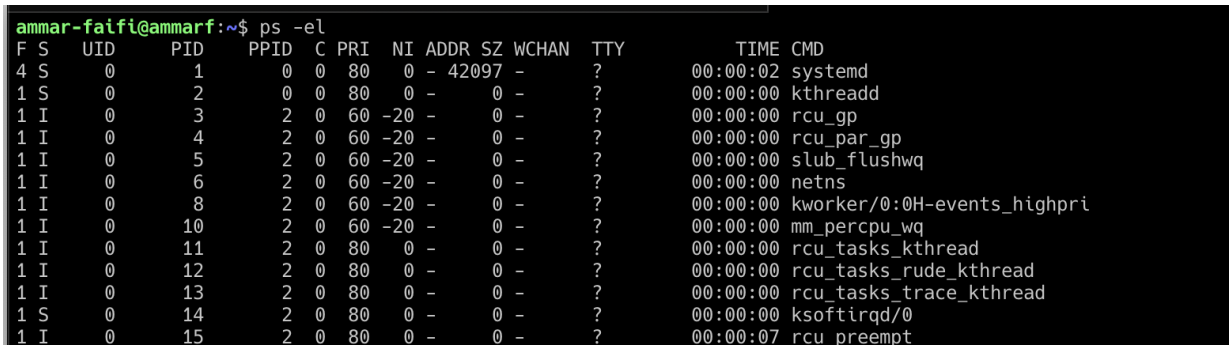
1.3 `ps -u`

will show only the processes started by the current user.



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.



```
ammar-faifi@ammarf:~$ 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 follwing 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 hrtime 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 to 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.