Corneille Ngoy

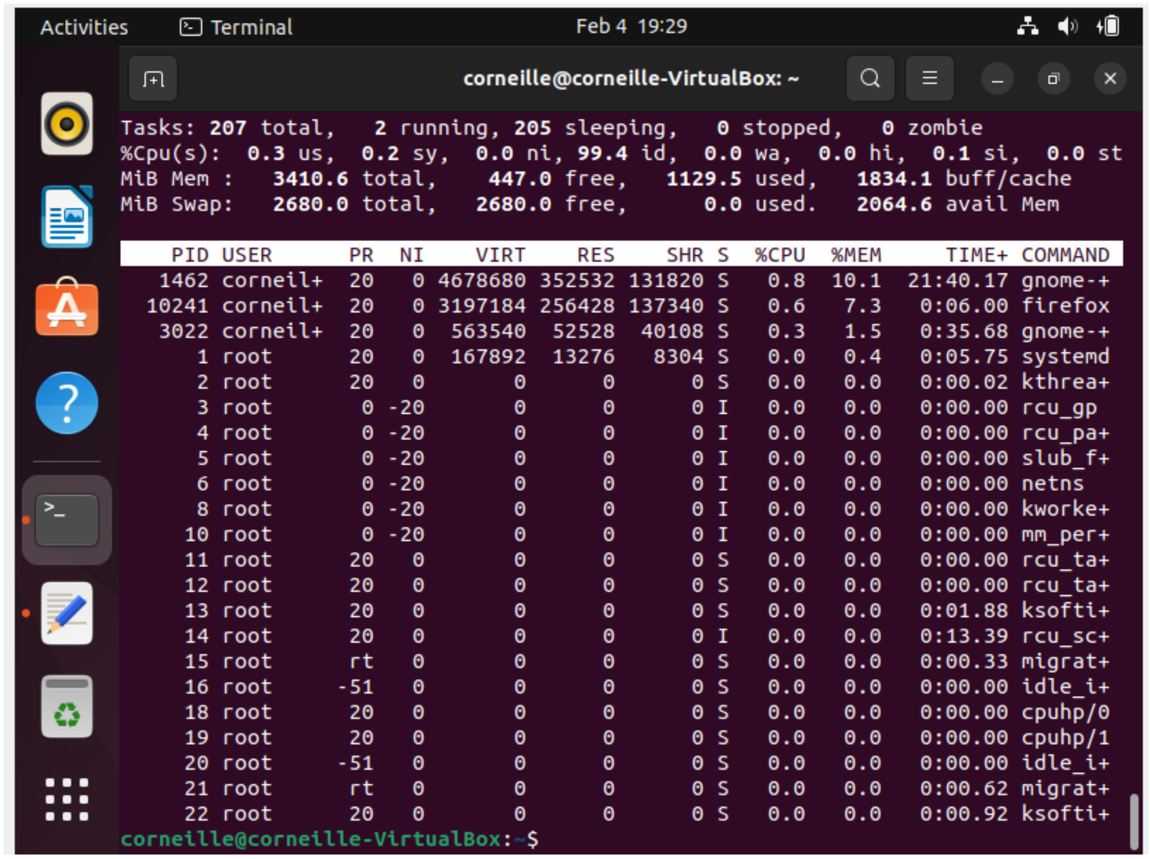
Lab3-Instructions for Running a C Program in Ubuntu and System Calls

1. How many child processes are created upon execution of this program?

   3 processes are executed

2. When you start a browser, you will notice the browser process appear in the top display. What does it consume?

   It consumes Memory and CPU time. Here is the screenshot of the command top running

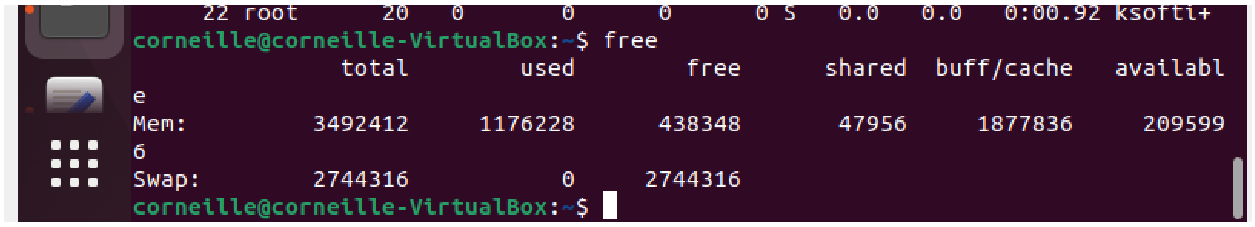


```
Tasks: 207 total,   2 running, 205 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.3 us,  0.2 sy,  0.0 ni, 99.4 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
MiB Mem :  3410.6 total,   447.0 free,  1129.5 used,  1834.1 buff/cache
MiB Swap:  2680.0 total,  2680.0 free,     0.0 used.  2064.6 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   1462 corneil+  20   0 4678680 352532 131820 S   0.8  10.1  21:40.17 gnome-+
  10241 corneil+  20   0 3197184 256428 137340 S   0.6   7.3   0:06.00 firefox
   3022 corneil+  20   0  563540  52528  40108 S   0.3   1.5   0:35.68 gnome-+
      1 root      20   0  167892  13276   8304 S   0.0   0.4   0:05.75 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.02 kthrea+
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_pa+
      5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 slub_f+
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 netns
      8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworke+
     10 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_per+
     11 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_ta+
     12 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_ta+
     13 root      20   0       0      0      0 S   0.0   0.0   0:01.88 ksofti+
     14 root      20   0       0      0      0 I   0.0   0.0   0:13.39 rcu_sc+
     15 root      rt   0       0      0      0 S   0.0   0.0   0:00.33 migrat+
     16 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_i+
     18 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/0
     19 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/1
     20 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_i+
     21 root      rt   0       0      0      0 S   0.0   0.0   0:00.62 migrat+
     22 root      20   0       0      0      0 S   0.0   0.0   0:00.92 ksofti+
corneille@corneille-VirtualBox:~$
```

3. How much memory is available in the system?

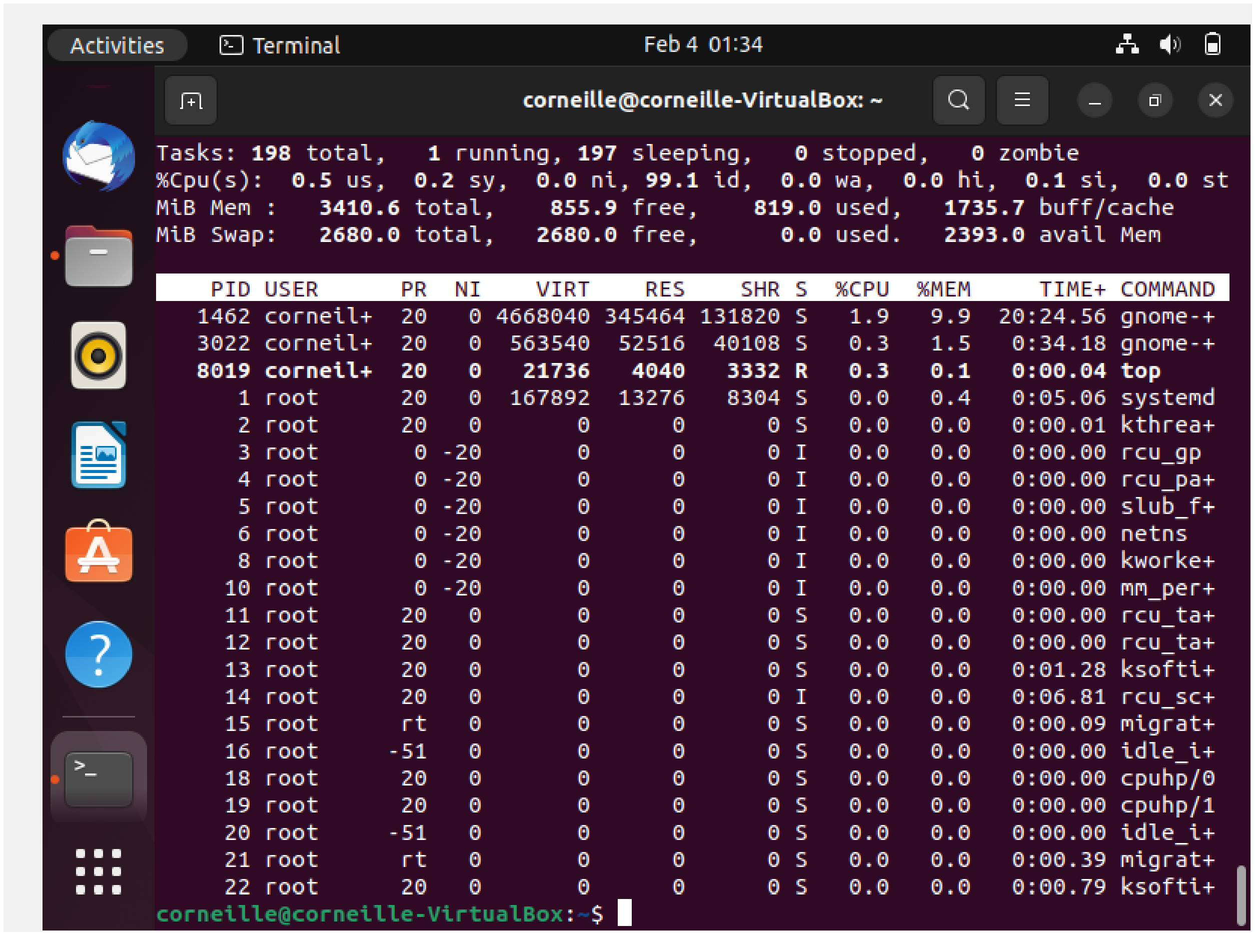
   In this case below, there is 438348 Kilobytes that are free.



```
     22 root      20   0       0      0      0 S   0.0   0.0   0:00.92 ksofti+
corneille@corneille-VirtualBox:~$ free
               total        used        free      shared  buff/cache   availabl
e
Mem:         3492412     1176228      438348       47956     1877836      209599
6
Swap:        2744316           0     2744316
corneille@corneille-VirtualBox:~$ 
```

4. Which process consumes the most CPU?

In this case below, it's the PID 1462 that's consuming the most %CPU at about 1.9%.

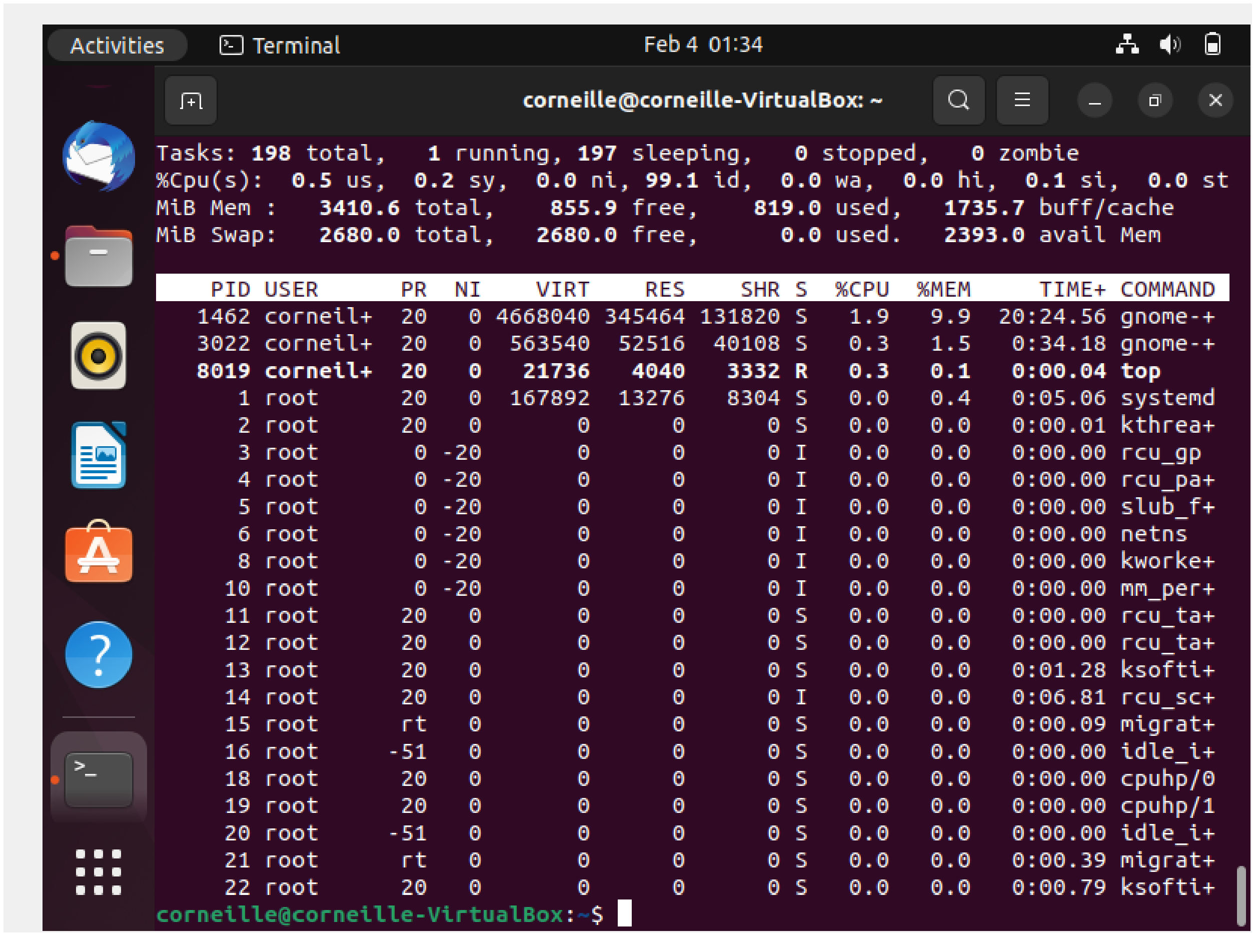


```
Tasks: 198 total,   1 running, 197 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.5 us,  0.2 sy,  0.0 ni, 99.1 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
MiB Mem :  3410.6 total,   855.9 free,   819.0 used,  1735.7 buff/cache
MiB Swap:  2680.0 total,  2680.0 free,     0.0 used.  2393.0 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   1462 corneil+  20   0 4668040 345464 131820 S   1.9   9.9  20:24.56 gnome-+
   3022 corneil+  20   0  563540  52516  40108 S   0.3   1.5   0:34.18 gnome-+
   8019 corneil+  20   0   21736   4040   3332 R   0.3   0.1   0:00.04 top
      1 root      20   0  167892  13276   8304 S   0.0   0.4   0:05.06 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.01 kthrea+
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_pa+
      5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 slub_f+
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 netns
      8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworke+
     10 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_per+
     11 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_ta+
     12 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_ta+
     13 root      20   0       0      0      0 S   0.0   0.0   0:01.28 ksofti+
     14 root      20   0       0      0      0 I   0.0   0.0   0:06.81 rcu_sc+
     15 root      rt   0       0      0      0 S   0.0   0.0   0:00.09 migrat+
     16 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_i+
     18 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/0
     19 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/1
     20 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_i+
     21 root      rt   0       0      0      0 S   0.0   0.0   0:00.39 migrat+
     22 root      20   0       0      0      0 S   0.0   0.0   0:00.79 ksofti+
corneille@corneille-VirtualBox:~$ 
```

5. Which process has the most memory?

In this case below, the PID 1462 uses the most memory. It uses 9.9% of the memory.



```
Tasks: 198 total,   1 running, 197 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.5 us,  0.2 sy,  0.0 ni, 99.1 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
MiB Mem :  3410.6 total,   855.9 free,   819.0 used,  1735.7 buff/cache
MiB Swap:  2680.0 total,  2680.0 free,     0.0 used.  2393.0 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   1462 corneil+  20   0 4668040 345464 131820 S   1.9   9.9  20:24.56 gnome-+
   3022 corneil+  20   0  563540  52516  40108 S   0.3   1.5   0:34.18 gnome-+
   8019 corneil+  20   0   21736   4040   3332 R   0.3   0.1   0:00.04 top
      1 root      20   0  167892  13276   8304 S   0.0   0.4   0:05.06 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.01 kthrea+
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_pa+
      5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 slub_f+
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 netns
      8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworke+
     10 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_per+
     11 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_ta+
     12 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_ta+
     13 root      20   0       0      0      0 S   0.0   0.0   0:01.28 ksofti+
     14 root      20   0       0      0      0 I   0.0   0.0   0:06.81 rcu_sc+
     15 root      rt   0       0      0      0 S   0.0   0.0   0:00.09 migrat+
     16 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_i+
     18 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/0
     19 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/1
     20 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_i+
     21 root      rt   0       0      0      0 S   0.0   0.0   0:00.39 migrat+
     22 root      20   0       0      0      0 S   0.0   0.0   0:00.79 ksofti+
corneille@corneille-VirtualBox:~$
```

6. Could you please explain the following commands?

apt-get: **apt-get** is a package manager for Debian-based systems, such as Ubuntu and Debian. It allows you to easily install, update, and remove packages and their dependencies.

yum: **yum** is a package manager for Red Hat-based systems, such as Fedora and CentOS. It allows you to easily install, update, and remove packages and their dependencies.

wget: **wget** is a command-line utility that allows you to download files from the internet. You can use **wget** to download files from websites, FTP servers, and other sources.

gzip: **gzip** is a file compression tool. It can be used to compress and decompress files in the **.gz** format. This format is commonly used for compressing log files, backups, and other large files.

tar: **tar** is a file archiving tool. It can be used to create and extract tarballs, which are archive files that contain multiple files or directories. Tarballs are often used for backing up data or for distributing software.

rar: **rar** is a file compression and archiving tool. It can be used to create and extract RAR files, which are archive files that contain multiple files or directories. RAR files are commonly used for compressing and archiving data.

7. Write a program that will generate a child process. In a loop, the child process writes “I am a child process" 200 times and the parent process repeatedly prints "I am a parent process" in a loop.

Here is the program that will a generate a child process written in C.

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

#define CHILD_MESSAGE "I am a child process\n"
#define PARENT_MESSAGE "I am a parent process\n"
#define LOOP_COUNT 200

int main(void) {
    pid_t pid;
    int i;

    pid = fork();
    if (pid == -1) {
        perror("fork");
        return 1;
    }

    if (pid == 0) {
        /* Child process */
        for (i = 0; i < LOOP_COUNT; i++) {
            write(1, CHILD_MESSAGE, sizeof(CHILD_MESSAGE) - 1);
        }
    } else {
        /* Parent process */
        for (i = 0; i < LOOP_COUNT; i++) {
            write(1, PARENT_MESSAGE, sizeof(PARENT_MESSAGE) - 1);
        }
        wait(NULL);
    }

    return 0;
}
```

8. Write a program that create a child process with the fork () system call. The parent process waits for the child process to finish before printing the contents of the current directory.

Here is the program written in C that creates a child process with the fork () system call.

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

int main(void) {
    pid_t pid;
    int status;

    pid = fork();
    if (pid == -1) {
        perror("fork");
        exit(1);
    }
    if (pid == 0) {
        /* Child process */
        printf("Child process running...\n");
        sleep(5);
    } else {
        /* Parent process */
        waitpid(pid, &status, 0);
        printf("Child process finished.\n");
        printf("Contents of the current directory:\n");
        system("ls");
    }

    return 0;
}
```

9. Write a program that create a child process with the fork () system call and print its PID. Following a fork () system call, both parent and child processes print their process type and PID. Additionally, the parent process prints the PID of its child, and the child process prints the PID of its parent.

Here is the program written in C that create a child process with the fork () system call and print its PID

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main(void) {
    pid_t pid;

    pid = fork();
    if (pid == -1) {
        perror("fork");
        return 1;
    }

    if (pid == 0) {
        /* Child process */
        printf("Child process: PID = %d, PPID = %d\n", getpid(), getppid());
    } else {
        /* Parent process */
        printf("Parent process: PID = %d, Child PID = %d\n", getpid(), pid);
    }

    return 0;
}
```