

Name: Jimmy Battistoni

Date: 02/01/2023

Course: CS470 Operating System

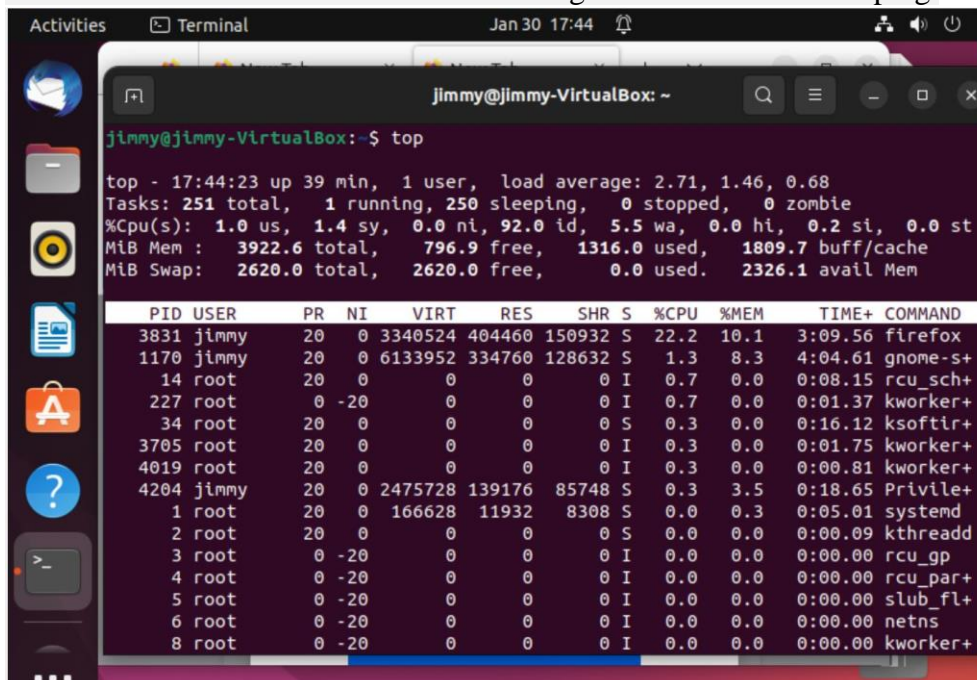
Assignment: Lab 3

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

The amount of process that created upon execution of this program is 3 child process. We know this from using $2^n - 1$ and the code has two `fork()`; meaning that n is 2. So now we have $2^2 - 1$ which equals = 3 child processes are created.

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

When opening Firefox browser and using the `top` command to see what the process is consuming. Looking at the screenshot below Firefox browser consume 1.0% CPU, along with 1316 MiB used memory, followed by 1809.7 MiB in buff/cache. Looking at the Tasks line we can see that there are 251 tasks with on 1 running and the other 251 sleeping.



```
jimmy@jimmy-VirtualBox:~$ top
top - 17:44:23 up 39 min, 1 user, load average: 2.71, 1.46, 0.68
Tasks: 251 total, 1 running, 250 sleeping, 0 stopped, 0 zombie
%Cpu(s): 1.0 us, 1.4 sy, 0.0 ni, 92.0 id, 5.5 wa, 0.0 hi, 0.2 si, 0.0 st
MiB Mem : 3922.6 total, 796.9 free, 1316.0 used, 1809.7 buff/cache
MiB Swap: 2620.0 total, 2620.0 free, 0.0 used. 2326.1 avail Mem

  PID USER      PR  NI   VIRT    RES    SHR S  %CPU  %MEM    TIME+  COMMAND
 3831 jimmy     20   0 3340524 404460 150932 S   22.2  10.1   3:09.56 firefox
 1170 jimmy     20   0 6133952 334760 128632 S    1.3   8.3   4:04.61 gnome-s+
    14 root       20   0        0        0        0 I    0.7   0.0   0:08.15 rcu_sch+
   227 root      -20   0        0        0        0 I    0.7   0.0   0:01.37 kworker+
    34 root       20   0        0        0        0 S    0.3   0.0   0:16.12 ksoftir+
  3705 root       20   0        0        0        0 I    0.3   0.0   0:01.75 kworker+
  4019 root       20   0        0        0        0 I    0.3   0.0   0:00.81 kworker+
  4204 jimmy     20   0 2475728 139176  85748 S    0.3   3.5   0:18.65 Privile+
    1 root       20   0  166628  11932   8308 S    0.0   0.3   0:05.01 systemd
    2 root       20   0        0        0        0 S    0.0   0.0   0:00.09 kthreadd
    3 root      -20   0        0        0        0 I    0.0   0.0   0:00.00 rcu_gp
    4 root      -20   0        0        0        0 I    0.0   0.0   0:00.00 rcu_par+
    5 root      -20   0        0        0        0 I    0.0   0.0   0:00.00 slub_fl+
    6 root      -20   0        0        0        0 I    0.0   0.0   0:00.00 netns
    8 root      -20   0        0        0        0 I    0.0   0.0   0:00.00 kworker+
```

3. How much memory is available in the system?

The available memory in the system is 3922.6 MiB in total. Followed by 796.9 MiB in memory that is free as seen below in the screenshot below.

```
MiB Mem : 3922.6 total, 796.9 free, 1316.0 used, 1809.7 buff/cache
```

4. Which process consumes the most CPU?

The process that consumes the most CPU is PID 1170 with the user jimmy and which uses 0.7% CPU. As seen in the screenshot below.

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1170	jimmy	20	0	6139092	341548	130200	S	0.7	8.5	4:36.62	gnome-shell
670	systemd+	20	0	14824	6148	5356	S	0.3	0.2	0:04.13	systemd-oomd
1	root	20	0	166628	11932	8308	S	0.0	0.3	0:05.05	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.09	kthreadd
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_par_gp

5. Which process has the most memory?

The process that has the most memory is PID 1170 with the user jimmy and uses 8.5% of memory with the virtual size of 6139092. As seen in the screenshot below.

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1170	jimmy	20	0	6139092	341548	130200	S	0.7	8.5	4:36.62	gnome-shell
670	systemd+	20	0	14824	6148	5356	S	0.3	0.2	0:04.13	systemd-oomd
1	root	20	0	166628	11932	8308	S	0.0	0.3	0:05.05	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.09	kthreadd
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_par_gp

6. Could you please explain the following commands?

apt-get – a Linux command line tool that is used to manage packages by installing, updating, and removing different software packages on the system. It also can be used to upgrade the system and make sure the system is up to date with the latest.

yum – a Linux command line tool that allows that users or system admin search, install, update, and remove different software packages that are on the system.

wget – a Linux command line tool that allows the system to download files from the internet, best with a URL and does not require the user to be logged in to download files.

gzip – a Linux command line tool that allows for the decompressing and compressing of files. This is done by reducing the size of the files utilizing the GZIP algorithm to compress the files taking less space on the system, in return smaller files also allow for faster transfer.

tar – a Linux command line tool that allows extraction and creation of archives that are used to compress many files to a single archive, that can have the files extracted from the single archive.

rar – is a Linux command line tool that allows for the creation of a rar file that compress the files that can than be used to extract the files from the compressed rar file.

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.

Screenshots of the program and the code are below:

[illegible]A screenshot of a terminal window from a Virtual Machine. The title bar at the top reads "jimmy@jimmy-VirtualBox: ~". On the left side of the title bar are standard Linux window controls (minimize, maximize, close). On the right side are search, menu, and window management icons. The terminal area has a dark purple background with white text. It contains 20 identical lines stacked vertically, each reading "I am the parent process". A vertical scrollbar is visible on the far right edge of the terminal window.

[illegible]

[illegible][illegible][illegible]

A screenshot of a terminal window from a Virtual Machine. The title bar at the top reads "jimmy@jimmy-VirtualBox: ~". On the right side of the title bar are standard window controls: a search icon, a menu icon (three horizontal lines), and three system icons (minimize, maximize, close). The main area of the terminal has a dark purple background and shows 20 lines of white text, each reading "I am the child process". A vertical scrollbar is visible on the far right edge of the terminal window.

A screenshot of a terminal window from a Virtual Machine. The title bar at the top reads "jimmy@jimmy-VirtualBox: ~". On the right side of the title bar are standard window controls: a search icon, a menu icon (three horizontal lines), a minimize button (-), a maximize button (square), and a close button (X). The main area of the terminal has a dark purple background and shows 20 identical lines of white text stacked vertically: "I am the child process". At the bottom of the terminal, there is a prompt character followed by some partially visible command-line options like "-c 200 -t 1000000".[illegible]A screenshot of a terminal window from a Virtual Machine. The title bar at the top reads "jimmy@jimmy-VirtualBox: ~". On the right side of the title bar are standard window controls: a search icon, a menu icon (three horizontal lines), and three window management icons (minimize, maximize, close). The main area of the terminal has a dark purple background and shows 20 lines of white text, each reading "I am the child process". A vertical scrollbar is visible on the far right edge of the terminal window.

[illegible][illegible]

```

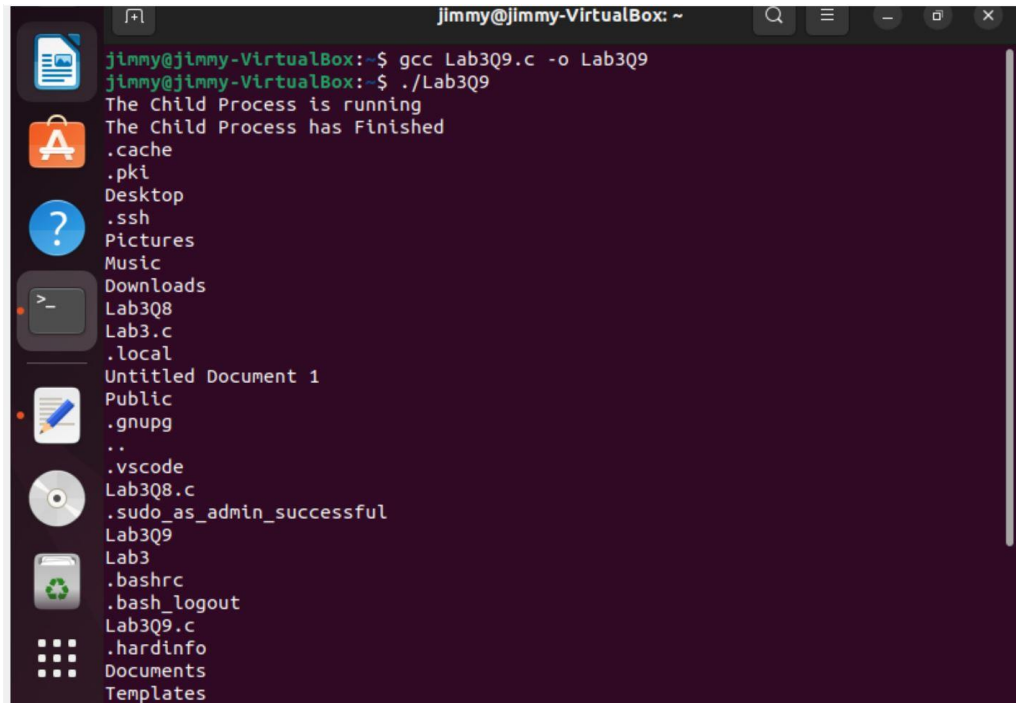
1 // #include <stdio.h>
2 // James Battistoni CS 470 Question 8
3 int main(){
4     int pid;
5
6     pid = fork();
7
8     // Check to see if the Fork Failed
9     if(pid < 0) {
10         printf("Fork failed\n");
11         return 1;
12     }
13     // Child Process
14 } else if(pid == 0) {
15     // For loop to print "I am the child process\n" 200 times
16     for (int i = 0; i < 200; i++) {
17         printf("I am the child process\n");
18     }
19     // Parent Process
20 } else {
21     // For loop to print "I am the parent process\n" 200 times
22     for (int i = 0; i < 200; i++) {
23         printf("I am the parent process\n");
24     }
25 }
26 return 0;

```

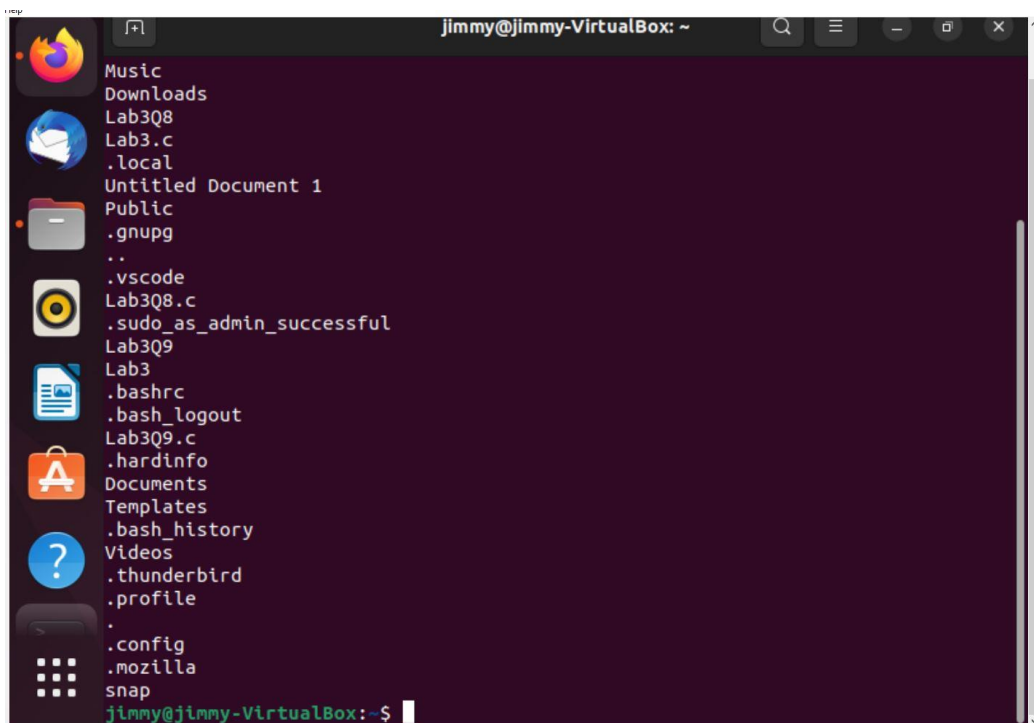
8. Write a program that create a child process with the fork () system call. The

parent process waits for the child process to finish before printing the contents of the current directory.

Screenshots of program running below, followed by code of program:



```
jimmy@jimmy-VirtualBox: ~  
jimmy@jimmy-VirtualBox:~$ gcc Lab3Q9.c -o Lab3Q9  
jimmy@jimmy-VirtualBox:~$ ./Lab3Q9  
The Child Process is running  
The Child Process has Finished  
.cache  
.pki  
Desktop  
.ssh  
Pictures  
Music  
Downloads  
Lab3Q8  
Lab3.c  
.local  
Untitled Document 1  
Public  
.gnupg  
..  
.vscode  
Lab3Q8.c  
.sudo_as_admin_successful  
Lab3Q9  
Lab3  
.bashrc  
.bash_logout  
Lab3Q9.c  
.hardinfo  
Documents  
Templates
```



```
jimmy@jimmy-VirtualBox: ~  
Music  
Downloads  
Lab3Q8  
Lab3.c  
.local  
Untitled Document 1  
Public  
.gnupg  
..  
.vscode  
Lab3Q8.c  
.sudo_as_admin_successful  
Lab3Q9  
Lab3  
.bashrc  
.bash_logout  
Lab3Q9.c  
.hardinfo  
Documents  
Templates  
.bash_history  
Videos  
.thunderbird  
.profile  
..  
.config  
.mozilla  
snap  
jimmy@jimmy-VirtualBox:~$
```

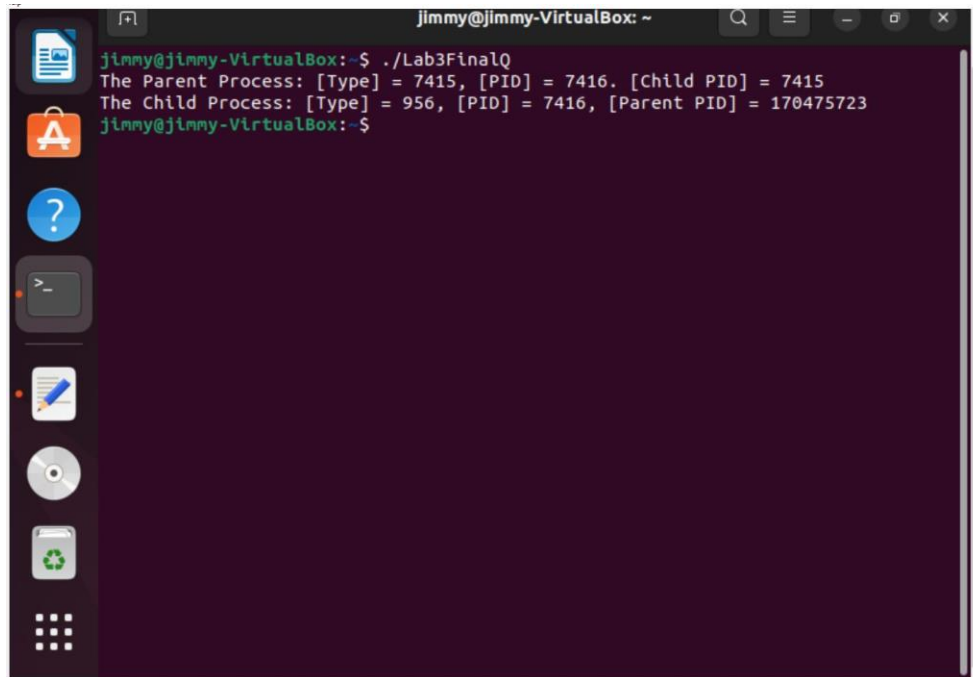
```
1 #include <unistd.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <sys/wait.h>
5 #include <dirent.h>
6
7 // James Battistoni CS 470 Question 9
8 int main(){
9     pid_t pid;
10
11     pid = fork();
12
13     // Check to see if the Fork Failed
14     if(pid < 0) {
15         printf("Fork failed\n");
16         return -1;
17     }
18     // Child Process
19     else if(pid == 0) {
20         printf("The Child Process is running\n");
21         sleep(10); // wait
22     }
23     // Parent Process
24     else {
25         // integer for the current status
26         int currStat;
27         waitpid(pid, &currStat, 0);
28         // Print out that the child process has finished
29     }
30 }
```

C Tab Width: 4 Ln 16, Col 19 INS

```
24     int currStat;
25     waitpid(pid, &currStat, 0);
26     // Print out that the child process has finished
27     printf("The Child Process has Finished\n");
28
29     DIR *dir;
30     struct dirent *ent;
31     // Open the Directory
32     if((dir = opendir(".")) != NULL) {
33         while((ent = readdir(dir)) != NULL) {
34             // Print out content
35             printf("%s\n", ent->d_name);
36         }
37         closedir(dir); // Close the Directory
38     } else {
39         perror("");
40         return EXIT_FAILURE;
41     }
42 }
43 return 0;
44 }
```

C Tab Width: 4 Ln 16, Col 19 INS

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.



```
jimmy@jimmy-VirtualBox: ~  
jimmy@jimmy-VirtualBox:~$ ./Lab3FinalQ  
The Parent Process: [Type] = 7415, [PID] = 7416. [Child PID] = 7415  
The Child Process: [Type] = 956, [PID] = 7416, [Parent PID] = 170475723  
jimmy@jimmy-VirtualBox:~$
```

```
1 #include <sys/types.h>  
2 #include <stdio.h>  
3 #include <unistd.h>  
4 // James Battistoni Lab3 Question 9  
5  
6 int main()  
7 {  
8     pid_t child_PID;  
9     child_PID = fork();  
10  
11     if (child_PID == 0) {  
12         printf("The Child Process: [Type] = %d, [PID] = %d, [Parent PID] =  
13 %d\n", getppid(), getpid());  
14     } else {  
15         printf("The Parent Process: [Type] = %d, [PID] = %d. [Child PID] =  
16 %d\n", getpid(), child_PID);  
17     }  
18     return 0;  
19 }
```

<https://github.com/JimmyBattis/CS-OS-470-Labs>