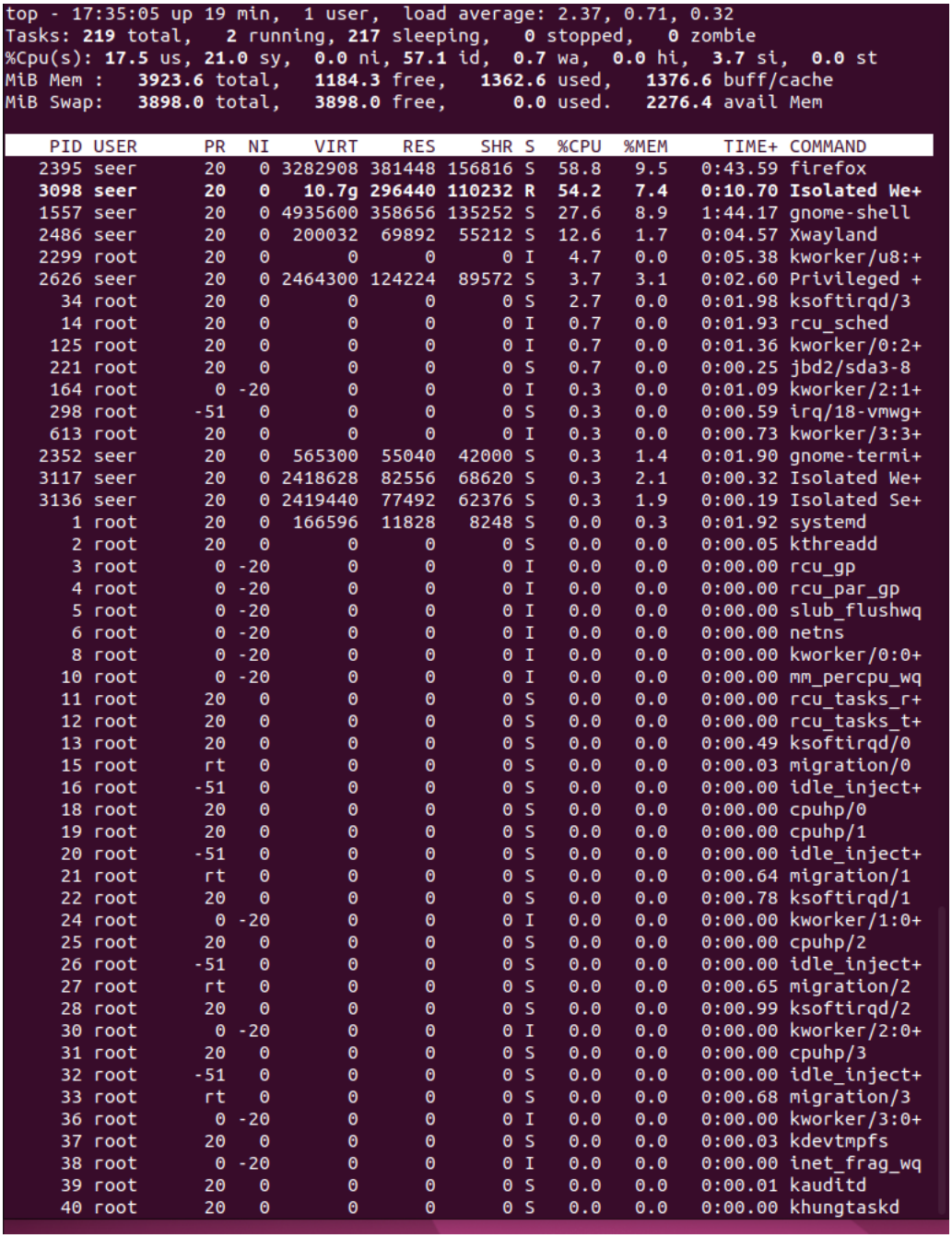
1. How many child processes are created upon execution of this program?

2^n – 1 child processes are created, with n = 2 (the number forks), 3 processes are created.

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



```
top - 17:35:05 up 19 min,  1 user,  load average: 2.37, 0.71, 0.32
Tasks: 219 total,   2 running, 217 sleeping,   0 stopped,   0 zombie
%Cpu(s): 17.5 us, 21.0 sy,  0.0 ni, 57.1 id,  0.7 wa,  0.0 hi,  3.7 si,  0.0 st
MiB Mem :   3923.6 total,   1184.3 free,   1362.6 used,   1376.6 buff/cache
MiB Swap:   3898.0 total,   3898.0 free,      0.0 used.   2276.4 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   2395 seer      20   0 3282908 381448 156816 S  58.8   9.5   0:43.59 firefox
   3098 seer      20   0   10.7g 296440 110232 R  54.2   7.4   0:10.70 Isolated We+
   1557 seer      20   0 4935600 358656 135252 S  27.6   8.9   1:44.17 gnome-shell
   2486 seer      20   0  200032  69892  55212 S  12.6   1.7   0:04.57 Xwayland
   2299 root      20   0       0      0      0 I   4.7   0.0   0:05.38 kworker/u8:+
   2626 seer      20   0 2464300 124224  89572 S   3.7   3.1   0:02.60 Privileged +
     34 root      20   0       0      0      0 S   2.7   0.0   0:01.98 ksoftirqd/3
     14 root      20   0       0      0      0 I   0.7   0.0   0:01.93 rcu_sched
    125 root      20   0       0      0      0 I   0.7   0.0   0:01.36 kworker/0:2+
    221 root      20   0       0      0      0 S   0.7   0.0   0:00.25 jbd2/sda3-8
    164 root       0 -20       0      0      0 I   0.3   0.0   0:01.09 kworker/2:1+
    298 root     -51   0       0      0      0 S   0.3   0.0   0:00.59 irq/18-vmwg+
    613 root      20   0       0      0      0 I   0.3   0.0   0:00.73 kworker/3:3+
   2352 seer      20   0  565300  55040  42000 S   0.3   1.4   0:01.90 gnome-termi+
   3117 seer      20   0 2418628  82556  68620 S   0.3   2.1   0:00.32 Isolated We+
   3136 seer      20   0 2419440  77492  62376 S   0.3   1.9   0:00.19 Isolated Se+
      1 root      20   0  166596  11828   8248 S   0.0   0.3   0:01.92 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.05 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 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 slub_flushwq
      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 kworker/0:0+
     10 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
     11 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_tasks_r+
     12 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_tasks_t+
     13 root      20   0       0      0      0 S   0.0   0.0   0:00.49 ksoftirqd/0
     15 root      rt   0       0      0      0 S   0.0   0.0   0:00.03 migration/0
     16 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject+
     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_inject+
     21 root      rt   0       0      0      0 S   0.0   0.0   0:00.64 migration/1
     22 root      20   0       0      0      0 S   0.0   0.0   0:00.78 ksoftirqd/1
     24 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/1:0+
     25 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/2
     26 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject+
     27 root      rt   0       0      0      0 S   0.0   0.0   0:00.65 migration/2
     28 root      20   0       0      0      0 S   0.0   0.0   0:00.99 ksoftirqd/2
     30 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/2:0+
     31 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/3
     32 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject+
     33 root      rt   0       0      0      0 S   0.0   0.0   0:00.68 migration/3
     36 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/3:0+
     37 root      20   0       0      0      0 S   0.0   0.0   0:00.03 kdevtmpfs
     38 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 inet_frag_wq
     39 root      20   0       0      0      0 S   0.0   0.0   0:00.01 kauditd
     40 root      20   0       0      0      0 S   0.0   0.0   0:00.00 khungtaskd
```

381,488KB (381.4MB) physical memory, 3,282,902KB (3.28GB) virtual memory.

3. How much memory is available in the system?

3923.6 MiB (MB in JEDEC standards), or 3.9GB Memory.

4. Which process consumes the most CPU?

Firefox, at 58.8% CPU utilization.

5. Which process has the most memory?

Firefox, at 9.5% Total memory utilization.

6. Could you please explain the following commands?

6.1 apt-get

“apt-get” is a tool that invokes Debian’s package manager, Aptitude. The Aptitude package manager is used for almost all of Debian’s derivative distributions, including Ubuntu and Linux Mint. A package manager is capable of downloading, installing, updating, and uninstalling packages on an operating system, while also managing the dependencies of packages that are being installed.

6.2 yum

“yum” is a tool to invoke Red-Hat system’s package manager, Yellowdog Updater, Modified. Similar to “apt-get”, yum is capable of all the regular tasks required of a package manager, but the packages provided specifically target Red-Hat distributions, which include Red-Hat enterprise and Fedora.

6.3 wget

“wget”, an abbreviation for “world-wide-web get”, downloads a file or web page from the internet given a URL. It can be used to download files from FTP servers, or a file from any publicly available website.

6.4 gzip

“gzip”, or “GNU zip”, is a file compression utility that uses the DEFLATE algorithm to compress files into .gz files. Its compression is not quite as good on average as LZMA compression, because it is packaged with all Linux distributions, it is a widely supported tool for the compression of files.

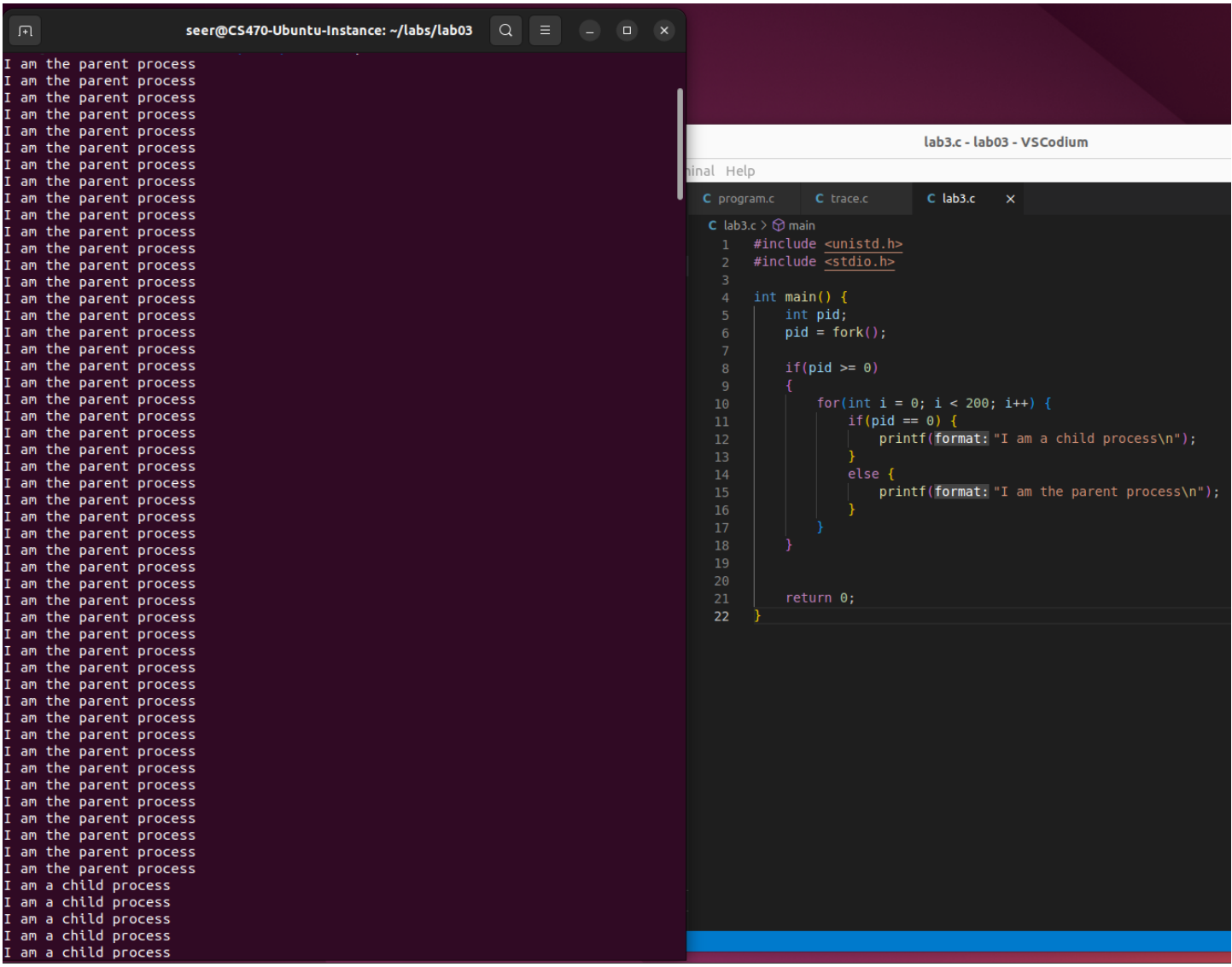
6.5 tar

“tar”, short for “Tape ARchiver”, is an *non-compressing* archival tool for Linux. “tar” archives multiple files into a .tar file, which can then be compressed using a tool like gzip. Although some tools combine the two steps into one, such as 7zip, WinZIP, or WinRAR, gzip and tar are provided individually as to help enforce a “single-responsibility principle” across its programs, where one program accomplishes one specific task, so if a certain aspect of the operating system is misbehaving, it can be easier to debug it and find the cause of the problem.

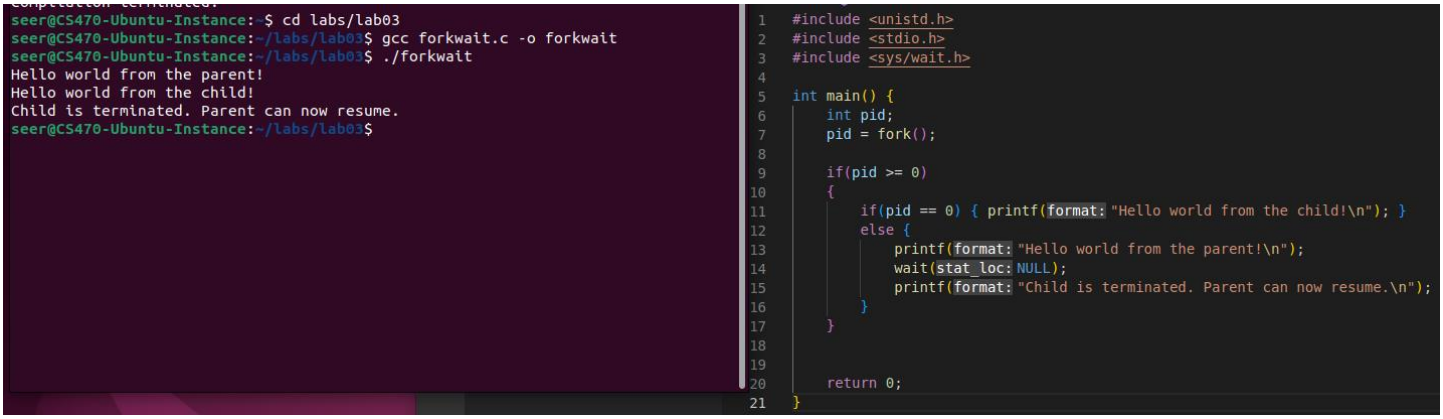
6.6 rar

“rar” is a compression and archival utility that primarily utilizes the proprietary “.rar” format. “rar” itself stands for “Roshal ARchive", as it was developed by the Russian software engineer Eugene Roshal. It was initially developed for Windows NT and MS-DOS systems and was later ported to Linux.

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



8. Write a program that creates 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



```
seer@CS470-Ubuntu-Instance:~$ cd labs/lab03
seer@CS470-Ubuntu-Instance:~/labs/lab03$ gcc forkwait.c -o forkwait
seer@CS470-Ubuntu-Instance:~/labs/lab03$ ./forkwait
Hello world from the parent!
Hello world from the child!
Child is terminated. Parent can now resume.
seer@CS470-Ubuntu-Instance:~/labs/lab03$
```

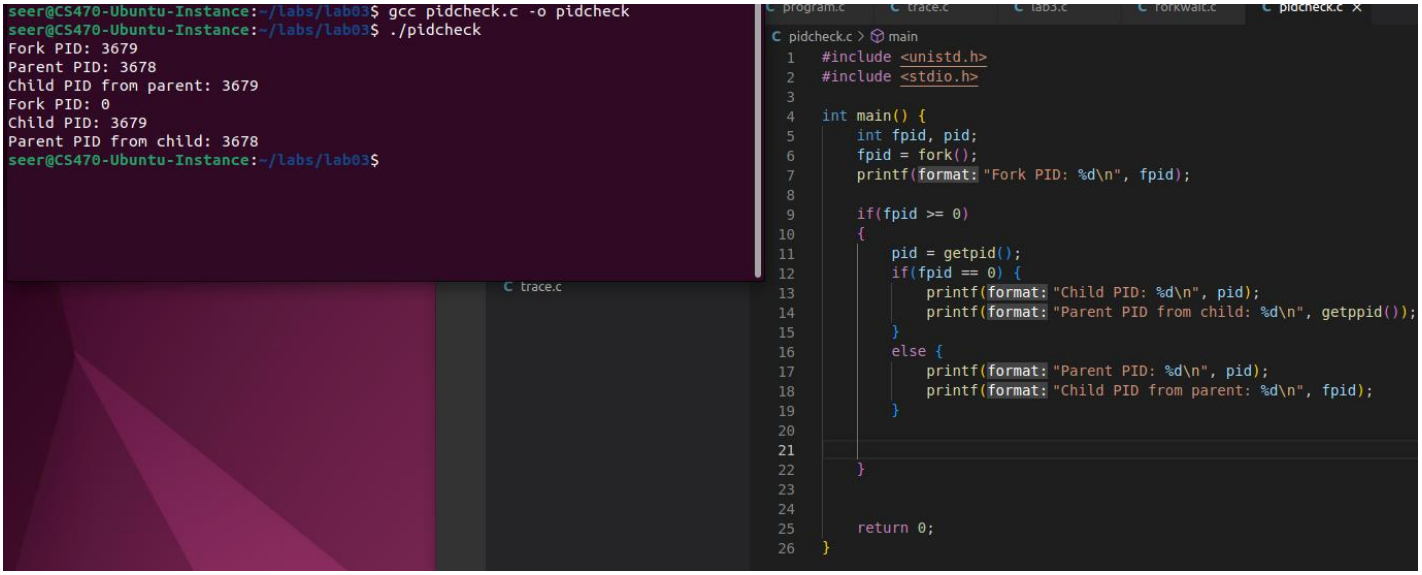
```c
#include <unistd.h>
#include <stdio.h>
#include <sys/wait.h>

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

    if(pid >= 0)
    {
        if(pid == 0) { printf(format: "Hello world from the child!\n"); }
        else {
            printf(format: "Hello world from the parent!\n");
            wait(stat_loc: NULL);
            printf(format: "Child is terminated. Parent can now resume.\n");
        }
    }


    return 0;
}
```

9. Write a program that create a child process with the fork() system call and print its PID. Following the 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.



```
seer@CS470-Ubuntu-Instance:~/labs/lab03$ gcc pidcheck.c -o pidcheck
seer@CS470-Ubuntu-Instance:~/labs/lab03$ ./pidcheck
Fork PID: 3679
Parent PID: 3678
Child PID from parent: 3679
Fork PID: 0
Child PID: 3679
Parent PID from child: 3678
seer@CS470-Ubuntu-Instance:~/labs/lab03$
```

```c
#include <unistd.h>
#include <stdio.h>

int main() {
    int fpid, pid;
    fpid = fork();
    printf(format: "Fork PID: %d\n", fpid);

    if(fpid >= 0)
    {
        pid = getpid();
        if(fpid == 0) {
            printf(format: "Child PID: %d\n", pid);
            printf(format: "Parent PID from child: %d\n", getppid());
        }
        else {
            printf(format: "Parent PID: %d\n", pid);
            printf(format: "Child PID from parent: %d\n", fpid);
        }


    }


    return 0;
}
```