Lab 2

Jarred Parr and Alexander Fountain

1. There are 3 lines printed
2. The program prints "before fork", then, it executes a fork which calls a subprocess. From here, the child executes the remainder of the function call then exits. Afterward, the parent function executes the remaining lines following the fork then exits, leading to the double print.
3. The status of the parent process is s when using the ps tool. This means that the parent process is suspended until the child process completes execution. This is because typically when forking, there is a pause of execution until the forked process has completed running.
4.
```
                                   Init
                                     -
                              Main (PID: 4918)
                              -              -
                 Fork() (PID: 4919)              Fork() (PID: 4920)
            -                      -                          -
(Runs the printf and for loop)  Fork() (PID: 4921)   (Runs the printf and for loop)
                                 -
                  (Runs the printf and for loop)
```
5. The process begins by making a `fork()` call, since the fork runs the remaining code in the function (unless otherwise specified), the forked code also forks itself in the subprocess. Because the child process finished in a staggered manner, it created a race condition to the output stream, which is why the output can be mangled in the `printf()` statements, since they do not block on access to `STDOUT`.

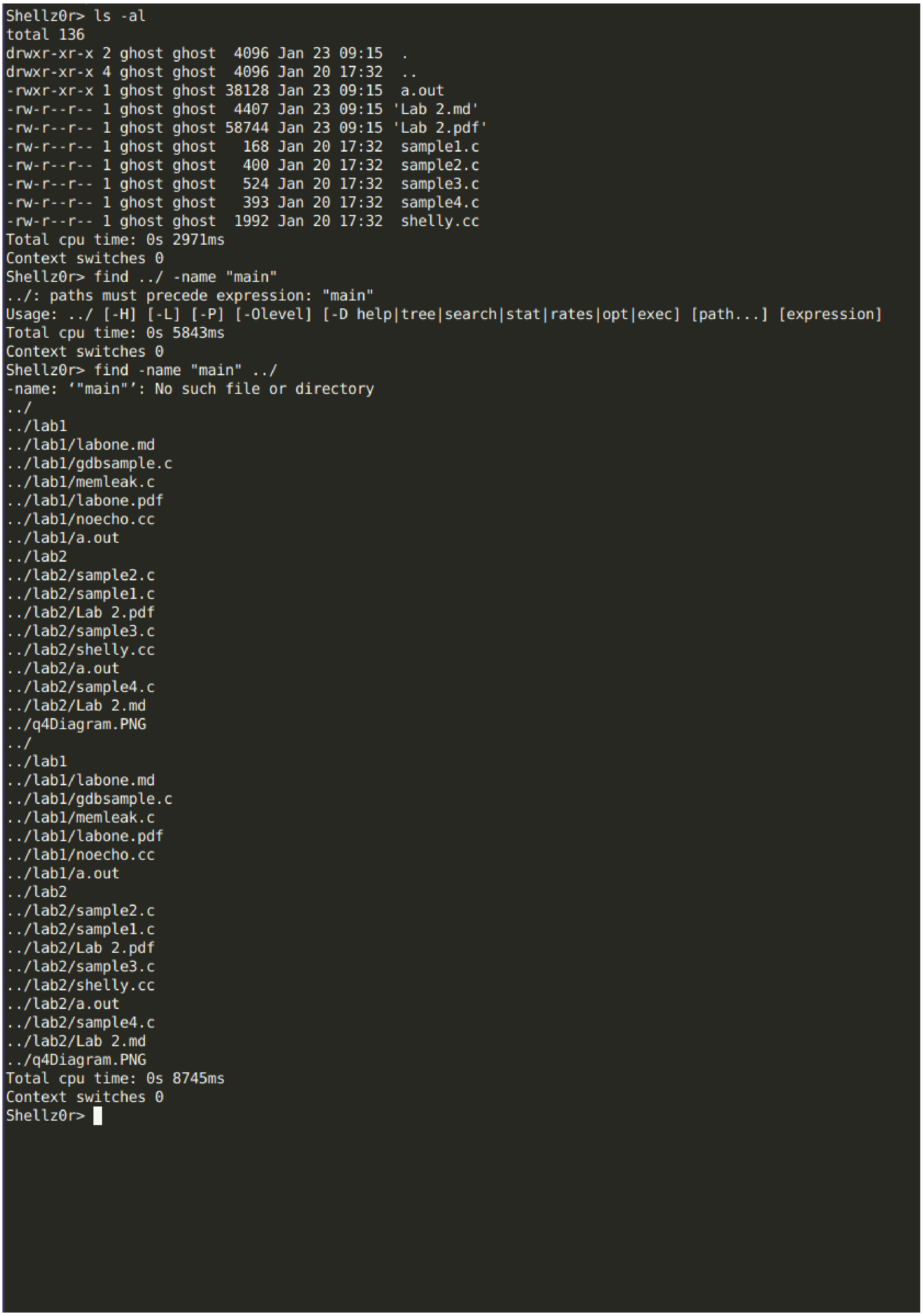
```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main()
{

    // use these variables

    pid_t pid, child;
    int status;

    if ((pid = fork()) < 0) {
        perror("fork failure");
        exit(1); // Proper exit code inserted here
    }
    else if (pid == 0) {
        printf("I am child PID %ld\n", (long) getpid());
        exit(0);

    }
    else {
        wait(&status); // Inserted here

        printf("Child PID %ld terminated with return status %d\n", (long) child,
status);
    }
    return 0;
}
```

7. The parent process thread prints first because, when forking, the child process run while the parent process was waiting.
8. The `wait()` call captures the returned status code of the forked process and stores it in the `pid_t` typed variable
9. Since `execvp()` overwrites the executable, it does not run the remaining code since, in most cases, when calling `execvp()` without a handler, it will turn whatever executable is running the code into the new executable as a result of the call.
10. The second argument contains the flags that were passed to the command and it handles those flags.

The Shell Code

```
#include <algorithm>
#include <iostream>
#include <cstdlib>
#include <sys/wait.h>
#include <unistd.h>
#include <cstring>
#include <sys/time.h>
#include <vector>
#include <sys/resource.h>

int main(int argc, char** argv) {
  std::string instruction;
  struct rusage resources_used;

  for (;;) {
    std::cout << "Shellz0r> ";
    std::getline(std::cin, instruction);
    int instruction_size = instruction.length();

    if (instruction_size <= 0) {
      fprintf(stderr, "Invalid command sequence");
      return EXIT_FAILURE;
    } else if (instruction[instruction_size - 1] == '\n') {
      instruction[instruction_size - 1] = '\0';
    }

    if (instruction == "quit") {
      return EXIT_SUCCESS;
    }

    char* temp = strtok(const_cast<char*>(instruction.c_str()), " ");
    char* command = temp;
    std::vector<char*> flags;
    while (temp != NULL) {
      flags.push_back(temp);
      temp = strtok(NULL, " ");
    }
    flags.erase(flags.begin());

    pid_t pid;
    int status;
    long time_s;
    long time_ms;
    long last_context_switch;
    pid = fork();

    if (pid < 0) {
      std::cerr << "Fork machine broke" << std::endl;
      exit(EXIT_FAILURE);
    } else if (pid == 0) {
      if (execvp(command, flags.data()) < 0) {
        std::cerr << "Exec machine broke" << std::endl;
        exit(EXIT_FAILURE);
      } else {
        exit(EXIT_SUCCESS);
      }
    } else {
      wait(&status);

      if (getrusage(RUSAGE_CHILDREN, &resources_used) < 0) {
        std::cout << "Failed to get process stats" << std::endl;
      } else {
        time_s = resources_used.ru_utime.tv_sec;
        time_ms = resources_used.ru_utime.tv_usec;
        last_context_switch = resources_used.ru_nivcsw;
        std::cout << "Total cpu time: " << resources_used.ru_utime.tv_sec - time_s << "s
" << time_ms << "ms " << std::endl;
        std::cout << "Context switches " << resources_used.ru_nivcsw -
last_context_switch << std::endl;
      }
    }
  }

  return EXIT_SUCCESS;
}
```

Sample Output



```
Shellz0r> ls -al
total 136
drwxr-xr-x 2 ghost ghost  4096 Jan 23 09:15  .
drwxr-xr-x 4 ghost ghost  4096 Jan 20 17:32  ..
-rwxr-xr-x 1 ghost ghost 38128 Jan 23 09:15  a.out
-rw-r--r-- 1 ghost ghost  4407 Jan 23 09:15 'Lab 2.md'
-rw-r--r-- 1 ghost ghost 58744 Jan 23 09:15 'Lab 2.pdf'
-rw-r--r-- 1 ghost ghost   168 Jan 20 17:32  sample1.c
-rw-r--r-- 1 ghost ghost   400 Jan 20 17:32  sample2.c
-rw-r--r-- 1 ghost ghost   524 Jan 20 17:32  sample3.c
-rw-r--r-- 1 ghost ghost   393 Jan 20 17:32  sample4.c
-rw-r--r-- 1 ghost ghost  1992 Jan 20 17:32  shelly.cc
Total cpu time: 0s 2971ms
Context switches 0
Shellz0r> find ../ -name "main"
../: paths must precede expression: "main"
Usage: ../ [-H] [-L] [-P] [-Olevel] [-D help|tree|search|stat|rates|opt|exec] [path...] [expression]
Total cpu time: 0s 5843ms
Context switches 0
Shellz0r> find -name "main" ../
-name: ‘"main"’: No such file or directory
../
../lab1
../lab1/labone.md
../lab1/gdbsample.c
../lab1/memleak.c
../lab1/labone.pdf
../lab1/noecho.cc
../lab1/a.out
../lab2
../lab2/sample2.c
../lab2/sample1.c
../lab2/Lab 2.pdf
../lab2/sample3.c
../lab2/shelly.cc
../lab2/a.out
../lab2/sample4.c
../lab2/Lab 2.md
../q4Diagram.PNG
../
../lab1
../lab1/labone.md
../lab1/gdbsample.c
../lab1/memleak.c
../lab1/labone.pdf
../lab1/noecho.cc
../lab1/a.out
../lab2
../lab2/sample2.c
../lab2/sample1.c
../lab2/Lab 2.pdf
../lab2/sample3.c
../lab2/shelly.cc
../lab2/a.out
../lab2/sample4.c
../lab2/Lab 2.md
../q4Diagram.PNG
Total cpu time: 0s 8745ms
Context switches 0
Shellz0r>
```