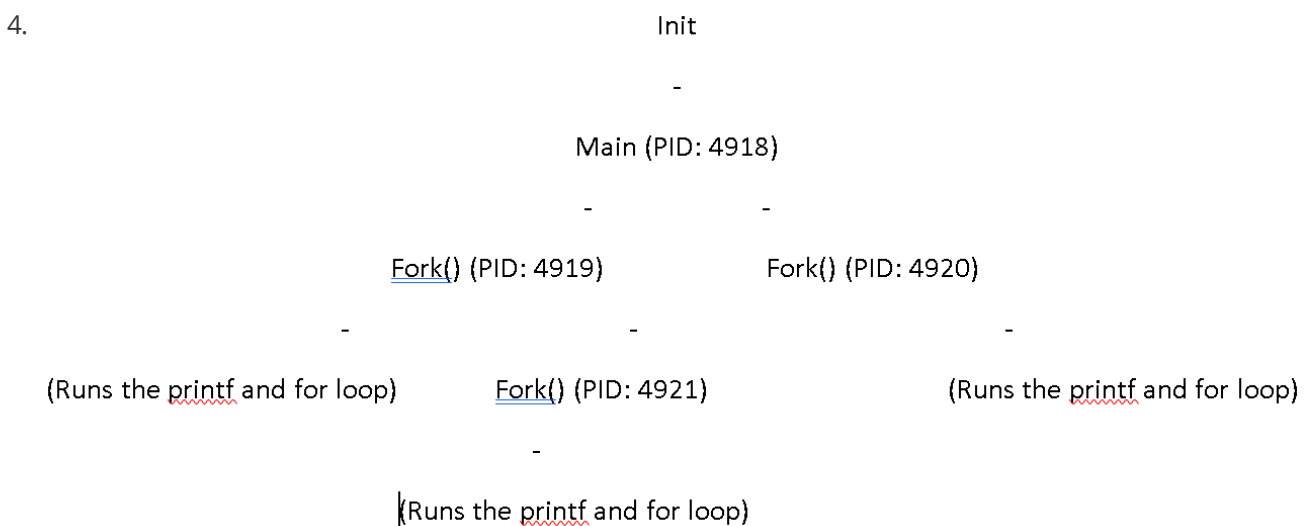


Lab 2

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.



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`.

```
#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.