EECS 678, W 4:00
Rachel J. Morris

1. Consider the following code segment:

```
        ...
        pid = fork();

        if (pid == 0) {
                /* Child Process */
                char    *cmd = (char *)malloc (128);
                memset (cmd, 0, 128);
                sprintf (cmd, “%s %s”, “ls”, “ll”);
                execl (“/bin/bash”, “bash”, “-c”, cmd, (char *)NULL);

                /* Free the memory allocated for command buffer */
                free (cmd);
        }
        ...
```

Is there a memory leak in the above program? Why or why not? (5-Points)
If there is a memory leak, is the leak dangerous? (10-Points)

The free(…) function won’t be called because execl will take over this child process. Therefore, any time this type of child is spun off, more memory will be allocated without being released.

2. Consider the following two ways of executing an “ls” job in a process:

```
        …
        execl (“/bin/bash”, “bash”, “-c”, “ls -ll”, (char *)NULL);
        execl (“/bin/ls”, "ls", "-ll", (char *)NULL);
        …
```

Do the above two commands produce the same output? Try running them in a sample program to verify your answer. (5-Points)
Are the two commands above equivalent in terms of the processes which get invoked after executing each of them? (5-Points)

```
Test 1      test 1 or test 2? 1

             Test 1: execl (“/bin/bash”, "bash", "-c", "ls -ll", (char *)NULL);
            Hi I'm the parent


            ------------------
            (program exited with code: 0)
            Press return to continue
            total 16
            -rwxr-xr-x 1 rayechell rayechell 8992 Feb 15 10:04 blorp
            -rw-r--r-- 1 rayechell rayechell  778 Feb 15 10:04 blorp.c
```

```
Test 2      test 1 or test 2? 2

             Test 2: execl (“/bin/ls”, "ls", "-ll", (char *)NULL);
            Hi I'm the parent


            ------------------
            (program exited with code: 0)
            Press return to continue
            total 16
            -rwxr-xr-x 1 rayechell rayechell 8992 Feb 15 10:05 blorp
            -rw-r--r-- 1 rayechell rayechell  781 Feb 15 10:05 blorp.c
```

Visually, the output appears the same.

With using execl on /bin/bash, it will execute the bash program, which will execute “ls -ll”. By doing it this way, environment variables are available to the “ls” command? According to docs, it also assigns the arguments as “positional parameters”, though the docs don’t really describe very well what this means or what it implies in the running process. (http://www.tldp.org/LDP/abs/html/internalvariables.html#POSPARAMREF)

3. While performing the "exec" command to run a c-program, why is the first argument always the same as the program name? (5-Points)

It isn't required that argv[0] contains the program name, but it is pretty standard. However, the name stored in argv[0] is what the process manager would display as the process name? (References: http://unix.stackexchange.com/questions/187666/why-do-we-have-to-pass-the-file-name-twice-in-exec-functions/187673#187673 , http://unix.stackexchange.com/questions/315812/why-does-argv-include-the-program-name)

4. What would happen if a program tries to read from an empty pipe? (5-Points)

If I create a pipe between a child and parent, but never write anything from the child's end, when I call

```
int byteCount = read( childToParent[ READ_PIPE ], readBuffer,
      sizeof( readBuffer ) );
```

the parent just says the byteCount is 0.

5. Consider a producer and a consumer program connected to each other through a named pipe in Linux. Assume that the producer is pushing data into the pipe at a much faster rate than the consumer's rate of removing data from the pipe. Is there a limit to how much data can be buffered in the pipe by the producer without affecting the producer's operation? (5-Points)

On https://linux.die.net/man/7/pipe it says "the pipe capacity is 65536 bytes."