Dawit Woldegiorgis
COP 4600
Section: 0467
September 10, 2015
Homework #2

Learning Experience:

Reading the *man* page of commands is usually difficult for beginners, and that was the case when I attempted to try and understand what was specified. After spending hours trying to read and re-read the page, I was able to extract bits of relevant information and piece them together to have a better, yet not complete, understanding of the *exec* function. The *man* page is not worded like any other works of literature or books, and therefore requires getting used to – which I believe I now am a litter more familiar with after completing this exercise.

Suggestion:
The code displayed on the course website (http://www.cise.ufl.edu/~dts/cop4600/hw02.html) does not compile without modification as it is missing some headers. This would have saved some time spent trying to debug the code.

On my honor, I have neither given nor received unauthorized aid in doing this assignment.

[illegible]

Dawit Woldegiorgis
COP 4600
Section: 0467
September 10, 2015
Homework #2

Source Code: *hw02-forkls.c*

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

main (int argc, char** argv)
{
	int pid = 0;

	printf("Parent about to fork\n");
	int i = 1;

	pid = fork();

	if (pid < 0) /* error forking */
	{
		fprintf(stderr, "Fork failed\n" );
		exit(-1);
	}
	else if (pid == 0) /* the child process will execute this branch */
	{
		/* on next line, the third arg is dash-ell-tee, NOT dash-one-tee */
		execlp( "/bin/ls", "ls", "-lt", NULL);
	}
	else /* the parent process will execute this branch */
	{
		printf( "parent waiting for child [PID = %d] to complete\n", pid);
		wait(NULL);
		printf("Child has completed\n");
	}
	return 0;
}
```

Output: *output.txt*

```
total 43
-rw-r--r--   1 dawit    stdnt          0 Sep 10 06:28 output.txt
-rwxr-xr-x   1 dawit    stdnt       7884 Sep 10 06:27 out
-rw-r--r--   1 dawit    stdnt        636 Sep 10 06:27 hw02-forkls.c
-rwxr-xr-x   1 dawit    stdnt       7896 Sep  9 12:14 test
Parent about to fork
parent waiting for child [PID = 9948] to complete
Child has completed
```

Dawit Woldegiorgis
COP 4600
Section: 0467
September 10, 2015
Homework #2

Answers:

1. The obvious difference between *execlp ( )* and the other members of the *exec* family is the format of the arguments it they accept. The functions that begin with *execl* ( )* – i.e. *execl, execlp, execle* -- take the arguments *arg0 – argn* as a comma separated list, whereas the *execv*( )* functions – i.e. *execv, execvp, execve* – take the arguments *arg0 – argn* as an array of strings terminated by a *null* character. For the functions *execlp ( ) and execvp ( )*, the first argument is a *file* name, and the path of the *file* is constructed via search through the environment of the caller. If the *file* is specified as the path to the *file*, then the *file* is treated as the path and no search would be necessary. This might be the reason as to why only *execlp ( )* and *execvp ( )* will not fail when "the new process image file has the appropriate access permission but is not in the proper format." Therefore, *execlp ( )* is the only function from the family that takes in a *file* name and a comma separated list of arguments used when *file* is executed.

2. Having the "ls" argument in the function doesn't seem to provide any extra functionality to the execution. Attempting to run the program by replacing "ls" with an empty string, or even a random nonsensical string, does not change how the program is executed. On the other hand, removing the "ls" argument completely does change the execution as the next argument "-lt" is seen as if it was *arg0* instead of *arg1* (the output suggest that an "ls" was executed with default settings rather than "ls –lt", which is what the original code performed). Therefore, the need for having some string for *arg0* is important although what the argument should be is not clear.

3. The parent process executed the *return 0;* command, because the child was "dead" at the time when *return* was executed. The program execution started with the parent, and it forked off to create the child. The parent then waited for the child to die as the *wait* command was invoked in the parent. When the child dies, the parent began executing it's last command *printf* before it escaped the *if – else* statement, whereupon the *return* command was used. Additionally, the *execlp* command replaced the child process (a C- program) with the *ls* process, therefore the *return* statement could not have been executed by the child process.