COSC 301: Operating Systems

Lab 4: The process API

You can write/paste your answers (text and/or code) into the `lab04.txt` file in the lab04 repo and just submit that file to Moodle.

1. Consider the following program (also in the git repo, named `fork01.c`). Compile and run it, then explain its behavior:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char **argv) {

    char *s = "I am a process!";
    int value = 100;

    pid_t pid = fork();
    if (pid == 0) {
        s = "Or am I?";
        value -= 50;
    } else {
        s = "Who are you?";
        value += 50;
    }

    printf("%s %d\n", s, value);
    return 0;
}
```

2. Write an ill-tempered program that uses the `fork` system call in an effort to make the operating system crash (or run very slowly). Careful with this one --- only run it if you have all your work saved.

3. The following code (also in the repo as `fork03.c`) uses the `execv` system call to start and run the `ps` program.

Your task is to modify the program so that it first forks a child process, which then does the `execv`. The parent process should wait for the child process to finish running (using the `waitpid` system call), print a message like `"Child process finished"`, then exit.

The `waitpid` system call takes three parameters: the process ID of the child to wait for, a pointer to an int which is filled with the return value of the child process, and an options bitmask (an int). You should be able to call it similar to the following: `waitpid(pid, &childrv, 0);` where `childrv` is declared as an int. See the `man` page for more on `waitpid`.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <errno.h>
```

```
int main(int argc, char **argv)
{

    char *cmd[] = { "/bin/ls", "-ltr", ".", NULL };

    if (execv(cmd[0], cmd) < 0) {
        fprintf(stderr, "execv failed: %s\n", strerror(errno));
    }

    return 0;
}
```