

Programming Assignment I

Robert Maloy

3 September 2018

System Specifications of Development Machine

These three programs were developed on a MacBook Pro laptop running macOS Mojave Developer Beta 10 (Build 18A377A); the C code was written using Apple's *Xcode* integrated development environment as simple files, and was compiled via Terminal using GNU C compiler.

Program A (Warm-Up App)

Program A (or *program_one.c*) was composed with four pre-processor definitions—**stdio** and **stdlib**, which provide the standard gauntlet of C utilities—but also **sys/types.h** and **unistd.h** which are UNIX/POSIX standard libraries that do not exist under Microsoft Windows. As a result, development had to move from my Windows system over to my MacBook Pro in order to complete the assignment.

In the process of creating the program, due to the linear nature of it—and the fact it did not require extensive dynamic allocation or nested functions, it is written linearly, with a sole main function that performs all activities. In order to pass data to the forked children, I utilized UNIX pipelines through `pipe()`—this allows me to send and receive data through the children processes when they calculate data.

After creating the process fork, I run a simple if, else if and else statement to direct data where it needs to go, as this requires only one nested child to complete.

Program A Result

```
±|master x| OSCAssignment1
% ./program_one
I'm the parent process, my pid is 33359.
-----
I'm the child process, my pid is 33360.
Array sum is 27, and I will now be sending it to my parent.
-----
I'm the parent process, my pid is 33359.
27 is the final array sum.
±|master x| OSCAssignment1
% _
```

As you can see, when executing the program, it provides the parent PID, executes the child process which calculates the array sum and sends it back to it's parent via pipelining, which then prints out the final array sum. A very simple program that does not do very much at all.

Program B

Program B (or *program_two.c*) took Program A and *expanded it*. While the original program still exists in the file as a separate command, Program B takes any input from integer value 1 to 3, and passes it to the program which then converts the argument variable through `atoi` and checks to make sure it is either `<2` or `>3`, at which time it will direct to either `ProgramOne()` or `SyntaxError()`.

Should it succeed, it pipes itself through multiple pipes (done for redundancy purposes to ensure information integrity between parent and child and prevent any data corruption or desync).

It then goes through the typical process of splitting the array up through an integer called “`div_index`” which serves as a marker to indicate where the array is being split. The split array is then dumped into a 3D array which is arranged in the following way for a two-ways, one for a two-way split, and the other for a three-way split.

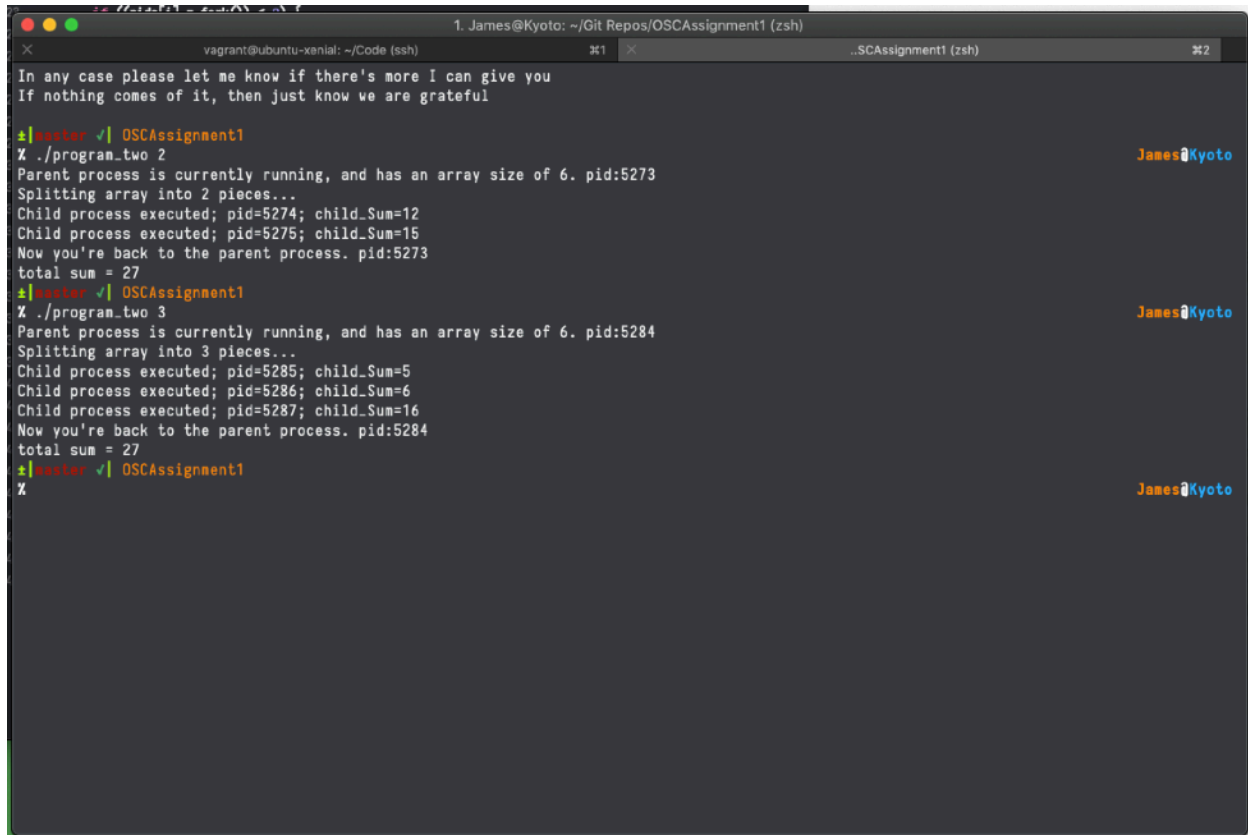
Child 1	2	3	7
Child 2	-1	10	6

Child 1	2	3
Child 2	7	-1
Child 3	10	6

By doing this, it allows us to use a for loop to acquire the values in the secondary array and quickly sum them for each child process—at which point the child sum is pushed back to the parent and updated. For each time it runs, it continues to add up towards a total value of 27.

Once it is complete, it prints the parent ID and the array sum.

Program B Results



```
1. James@Kyoto: ~/Git Repos/OSCAssignment1 (zsh)
vagrant@ubuntu-xenial: ~/Code (ssh)
In any case please let me know if there's more I can give you
If nothing comes of it, then just know we are grateful

$|baxter ✓| OSCAssignment1
% ./program_two 2
Parent process is currently running, and has an array size of 6. pid:5273
Splitting array into 2 pieces...
Child process executed; pid=5274; child_sum=12
Child process executed; pid=5275; child_sum=15
Now you're back to the parent process. pid:5273
total sum = 27

$|baxter ✓| OSCAssignment1
% ./program_two 3
Parent process is currently running, and has an array size of 6. pid:5284
Splitting array into 3 pieces...
Child process executed; pid=5285; child_sum=5
Child process executed; pid=5286; child_sum=6
Child process executed; pid=5287; child_sum=16
Now you're back to the parent process. pid:5284
total sum = 27

$|baxter ✓| OSCAssignment1
%
```

As you can see, it works as intended—the two and three partition numbers successfully print the data out as necessary, and return it to the parent.