Cody Carroll
Programming Assignment 1
Operating Systems Concepts
Fall 2018

Part A

I approached this code very linearly because it didn't require very much dynamic allocation or nested functions. It was written with a sole main function that performs all of the activities. To pass data to the forked child I utilized UNIX pipelines through pipe() and this allowed me to both send and receive data through the child process when they calculate data. After the fork a if, else if and else statement was used so that the data could be directed to where it needs to go since it only requires one child to complete. When executing this program, the parent process ID is provided, the child process is executed which calculates the sum of the array and then sends it back to the parent through pipelining, which then prints the final array sum.

```
Codys-MacBook-Pro:desktop cody$ ./PartA
Parent process: Process ID = 10256.

Child process: Process ID = 10257.
Sum = 24, sending back to parent process

Parent Process: Process ID = 10256.
24 is the sum of the final array.
Codys-MacBook-Pro:desktop cody$
```

Cody Carroll
Programming Assignment 1
Operating Systems Concepts
Fall 2018

Part B

Part B is just a variation of Part A where it has multiple children that each have a part of the overall array and then later sum them all up to give the array back to the parent process. Program B uses an input of an integer from 1 to 3 children. It starts by piping itself through five different pipes. It will then split up the array into different parts based on the integer chosen. The split-up array is then dumped into an array where it is arranged for the following ways: two-ways, one for a two-way split, and the other for a three-way split. After the array is split-up it allows for the use of a for loop so that it can acquire the values in the secondary array and then sum them up for each child process. After this the child sum is then pushed to the parent and then updated to show the new outcome in the final array. Every time it is ran it adds up to a total of 24. It then finalizes by printing the parent process ID and the sum of the final array.

```
Last login: Fri Sep  7 14:42:13 on ttys000
Codys-MacBook-Pro:~ cody$ cd desktop
Codys-MacBook-Pro:desktop cody$ make PartB
cc     PartB.c   -o PartB
Codys-MacBook-Pro:desktop cody$ ./PartB
The parent process is running, and has an array size of 6. pid:10096
The array is in 3 pieces...
di: 0
sArray[0][0] = 3
di: 1
sArray[0][1] = 6
2 | 2
di: 0
sArray[1][0] = 2
di: 1
sArray[1][1] = 7
4 | 2
di: 0
sArray[2][0] = 1
di: 1
sArray[2][1] = 5
0-0 --> 3
0-1 --> 6
9
Child Process: pid=10097; Child_Sum=9
1-0 --> 2
1-1 --> 7
9
Child Process: pid=10098; Child_Sum=9
2-0 --> 1
2-1 --> 5
6
Child Process: pid=10099; Child_Sum=6
Final sum of the parent process: Process ID:10096
Total sum = 24
```