**Problem 10 (remove Roman enumerations later)**

1. Beginning with the two functions declared in the beginning of the program, void h() reads a number from the pipe and stores it in the variable num. void \*f() receives arg[k], which has the kth k value from the for loop in main() that deals with arg[]. This function adds k to res, so res changes cumulatively by k each time in the arg[] loop in main(). Specifically, res is initially 0, then res = 0+0 = 0, then res = 0+1 = 1, then res = 1+2 = 3.

The main function of the program begins by initializing integers, a void pointer, a semaphore and a pipe. It also sets up the parent process to listen for when one of its children terminate to run function h. The program then enters a loop in which it forks a process in each iteration.

In each child process, num is incremented from 2 to 3, then written to the pipe. If num is ever greater than 3, then num is not written to the pipe and the child becomes a zombie until the parent process either waits on it or sends a kill signal.

Additionally, each child process creates threads that run function f. This uses the arg[] array to add to the child’s copy of res. After the threads are done running, the child terminates them in the following for loop. Since num should be less than 4 anyways in the child process, the child then writes its copy of num to the pipe and exits, passing its value of res to the parent.

In the parent process, the parent first writes its own value of num to the pipe, which is 2. This always happens before the last child writes its own value of num to the pipe, so the parent’s value of num is always overwritten by one of the child’s value of num in the pipe. The parent waits until all of the child processes have terminated before reading from the pipe, so the final result of num is always 3.

The last for loop that modifies the value of res runs two times because at that point in the code, the parent has not yet read from the pipe, so the value of num is unchanged and is still 2. In the loop, the parent waits on a child until that child terminates normally. After a child terminates normally, the parent takes the argument that the child passed to exit(), which is the child’s copy of res, and adds that to its own copy of res. This occurs once more in the loop, so the final value of res is 6.

1. See image below
2. See image below
3. See image below
4. Line 54 prints “Final result1: 6” and line 56 prints “Final result2: 3”

