1. **What does the program print, and in what order? (3 points)**

The program prints in the following order:

1. “waiting…\n”

2. “ received an interrupt\n”

3. “time to exit\n”

1. **Describe exactly what is happening to produce the answer observed for the above question. (3 points)**

When the program starts the signal function associates the function sigHandler(int) with the interrupt signal the user sends through the terminal when CTRL+C is pressed. The print statement “waiting…\n” is written to the terminal. The program then pauses which makes it wait until a signal has been received. Once the signal has been received the function sigHandler(int) runs which prints “ received an interrupt\n”. It then waits for 1 second with the sleep function and prints “time to exit\n”. It then exits the program.

1. **Where does the standard output of the child process go? Explain your answer. (2 points)**

The child process would continue to print to the file named temp. This is because dup2 is changing the output for the program to that file. Since the child has the same information as the parent it would continue to print to the temp file. This behavior would continue after the exec() system call because the child process would continue to treat that as it’s file output location. The new program executed as a result of exec() would then inherit the file descriptors of the child process which is why it does not change back to the original file descriptor output.

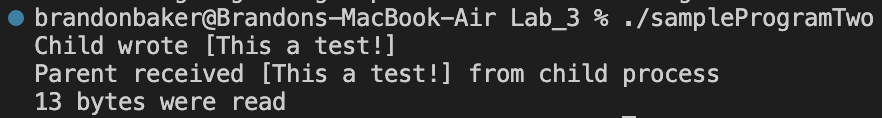
1. **Where does the standard output of the child process go? Explain your answer. (2 points)**

The standard output of the child would continue to be in the initial output of the program. This is because the dup2() system call is executed after the fork() and only for the parent. So the child would have no information about changing it’s standard output to another file at this time.

1. **What exactly does the program do (i.e., describe its high-level functionality)? (4 points)**

This program starts by making the file descriptors array, and other variables for the pipe creation and pid that are used for error checking and control flow. The pipe(fd) command is creating a communication tunnel to be used by two processes. A child process is created. The parent process waits for the child process to write to the file descriptor. The child process writes the int “3” into the output file descriptor, then prints “Child wrote 3\n” before terminating. Now that there is something within the communication channel the parent process reads “3” from the file descriptor and prints what it read, “Parent received 3 from child process\n”.

Screenshot from Modified sampleProgramTwo:



**Lab Programming assignment Communicating Processes:**

The code is in communicatingProcess.c

