Sebastian Torrejon Alonzo
Fabian Kirberg

CIS 452 Lab: Week #3
InterProcess Communication (IPC)

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

```
waiting...
 received an interrupt.
time to exit
```

2. **Describe exactly what is happening to produce the answer observed for the above question. (3 points)**
   The program calls the signal(SIGINT, sigHandler) to handle the signal 'SIGINT' which is (Control-C), and when the program receives that signal, it will execute the function 'sigHandler(int)'.
   1. While the program is waiting for that interrupt signal, it will print "waiting…" and then it will pause the program until it receives the signal.
   2. Once the program receives that signal, it will finally run the 'sigHandler(int)'.
   3. It will print " received an interrupt.".
   4. Then, it will sleep for a second (simulating shutting down any necessary processes, etc.).
   5. Finally, it will print "time to exit" and exit the program.

3. **Where does the standard output of the child process go? Explain your answer. (2 points)**
   The standard output of the child process goes to the same "temp" file as it did in the parent process.

4. **Where does the standard output of the child process go? Explain your answer. (2 points)**
   The standard output of the child process goes to the output specified by the program being executed via the exec() system call of the child process, if the program run by exec() does not change the output, the child process will default to the original standard output. This is because the fork() that creates the child process occurs before the dup2() call changes the standard output of the parent process.

Sebastian Torrejon Alonzo
Fabian Kirberg

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

   The program creates a pipe and forks a child process. If the process running is the child process, it writes the value of output to the write end of the pipe which is then sent to the parent process through the pipe and prints a message that it has done so. If the process running is the parent process, it reads from the read end of the pipe, stores the value in input, and then prints the value stored in input in a message.

   Steps:

   1. It declares an array ‘fd[2]’ to store the file descriptors for a pipe and two integer variables `pipeCreationResult` and ‘pid’.
   2. It creates a pipe using the ‘pipe()’ function and stores the result in ‘pipeCreationResult’.
   3. It then forks a child process using the ‘fork()’ function and stores the result in ‘pid’.
   4. In the child process:
      a. It writes an integer value (‘output’) to the write end of the pipe (‘fd[1]’).
      b. It prints the value that it has written along with a message.
   5. In the parent process:
      a. It reads an integer value from the read end of the pipe (‘fd[0]’) and stores it in the ‘input’ variable.
      b. It prints the received integer value along with a message.