|  |
| --- |
| **t1.c** |
| 1. What are the pid and ppid of the process that executes a.out?   **6279 and 2257, respectively.**   1. What are the pid and ppid of the CHILD process?   **6280 and 6279, respectively.**   1. Run a.out several times: which pid does NOT change and WHY?   **The parent of the process running a.out (2257) because it is the sh (shell) process.** |

|  |
| --- |
| **t2.c** |
| 1. What's the value of status in pid=wait(&status)?   What's the relation between the exitValue in exit(exitValue) and status in wait(&status)?  **0200 (Exit status: 2, Termination Signal: 0)**  **The process reaches exit(exitValue)**  **The left half of status is the hex equivalent to whatever exitValue I enter.**   1. Insert \*p = 1234; to HERE:   Run the program again, and answer (1) again:  **008b (Exit status: 0, Termination Signal: 11)**  **The process gets terminated because of the null pointer assignment error.**  **The process never reaches exit(exitValue)**   1. Insert { int a,b; a = 1; b = 0; a = a/ b; } to HERE:   Run the program again, and answer (1) again:  **0088 (Exit status: 0, Termination Signal: 8)**  **The process gets terminated because of the division by zero error.**  **The process never reaches exit(exitValue)** |

|  |
| --- |
| **a.c & b.c** |
| 1. Which process executes a.out? **2718** 2. Which process executes b.out? **2718, the same process**      1. What are the argv[] strings in b.out?   **The argv[] in b.out is equivalent to myargv[] created in a.out**   1. HOW TO PASS env[ ] strings to main(int argc, char \*argv[ ], char \*env[ ]);   **The same way as with argv[].**  **Just pass in an array of char\* with the last element in the array equal to NULL.**  **execve(command, myargv, myenv)** |

|  |
| --- |
| **io.c** |
| At L1: the printed line will show up on the screen.   1. At L2, L3: where do the printed lines go? Why?   **The printed lines go to the file myfile.**  **In the proc struct, fd[1] normally points to stdout.**  **close(1) caused fd[1] to point to nothing (NULL), now an empty slot in fd[]**  **dup(fd) filled the first empty slot, fd[1], with the file descriptor for myfile.**  **Now when printf() sends everything to fd[1] it is myfile instead of stdout.** |

|  |
| --- |
| **pipe.c** |
| 1. What's a pipe?   **A pipe is the concept of piping the output from one process into another process as its input.**   1. The parent is the pipe WRITER. How does it replace its fd=1 with pd[1]?   **close(1);**  **dup(pd[1]);**   1. The child is the pipe READER. How does it replace its fd=0 with pd[0]?   **close(0);**  **dup(pd[0]);**   1. MODIFY the code to let the parent be the READER and the child the WRITER.   Test run the program again.  **Just swapped the child/parent code and changed the printf’s accordingly.**  **parent 3127 read from pipe**  **child 3128 write to pipe**  **CHILD WRITES LINE 0 TO STDOUT**  **CHILD WRITES LINE 1 TO STDOUT**  **…**  **CHILD WRITES LINE 8 TO STDOUT**  **CHILD WRITES LINE 9 TO STDOUT**  **CHILD WRITES LINES TO PIPE**  **this is line 0 from child**  **this is line 1 from child**  **…**  **this is line 8 from child**  **this is line 9 from child** |