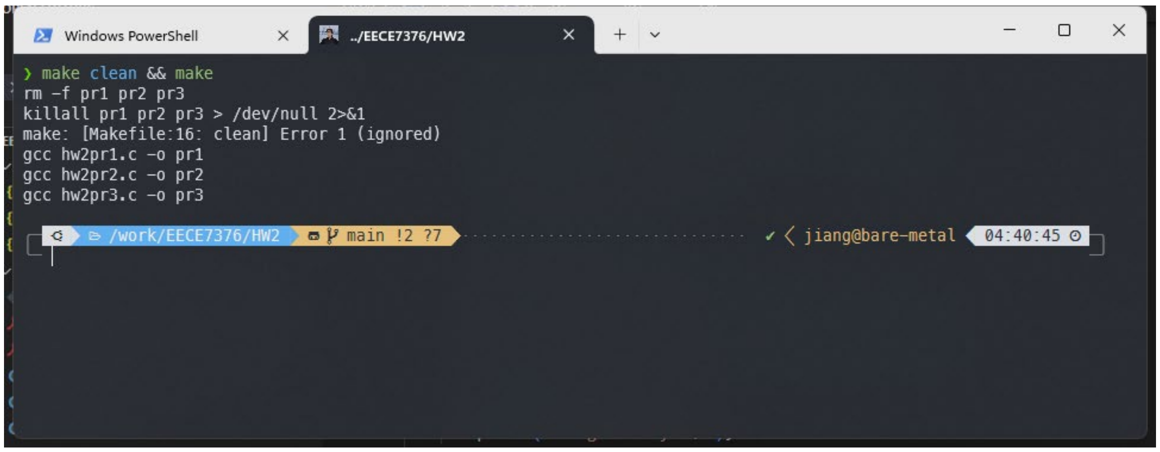
Northeastern University
College of Engineering
Department of Electrical & Computer Engineering

EECE7376: Operating Systems: Interface and Implementation

Homework 2



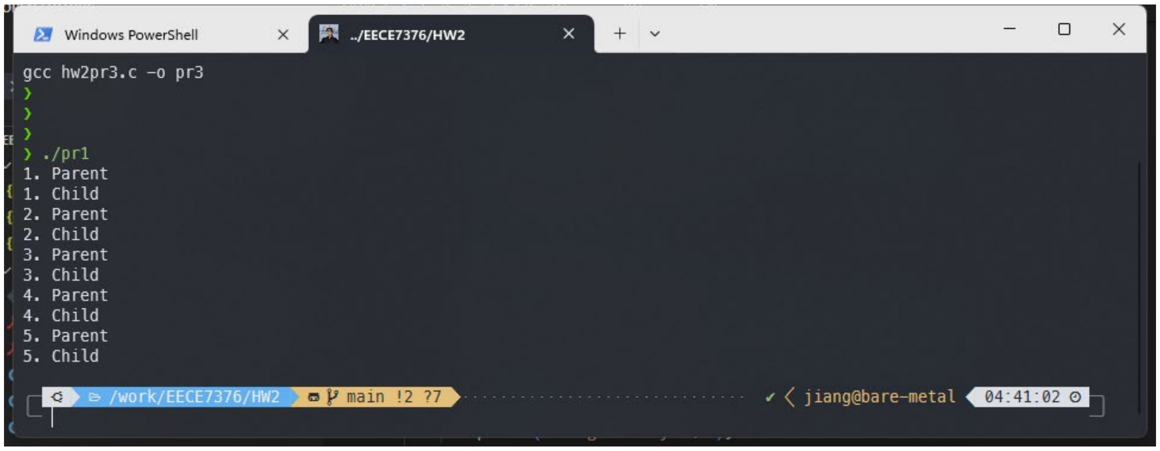
```
❯ make clean && make
rm -f pr1 pr2 pr3
killall pr1 pr2 pr3 > /dev/null 2>&1
make: [Makefile:16: clean] Error 1 (ignored)
gcc hw2pr1.c -o pr1
gcc hw2pr2.c -o pr2
gcc hw2pr3.c -o pr3
 /work/EECE7376/HW2  main !2 ?7 ······························ ✔ jiang@bare-metal  04:40:45
```

Problem 1

Approach Summary:

Two pipes were created using the pipe() system call, one for communication from parent to child and another from child to parent. A child process was created using fork(). The parent and child processes were set to use the pipes for synchronization. In a loop, each process writes a message to the standard output, then signals the other process by writing a character to its pipe. It then waits for a signal (a character read) from the other pipe before printing the next message. This ensures that the messages are alternated between the parent and child. Each process closes the ends of the pipes that it does not use to ensure proper signal flow.

Screenshot:

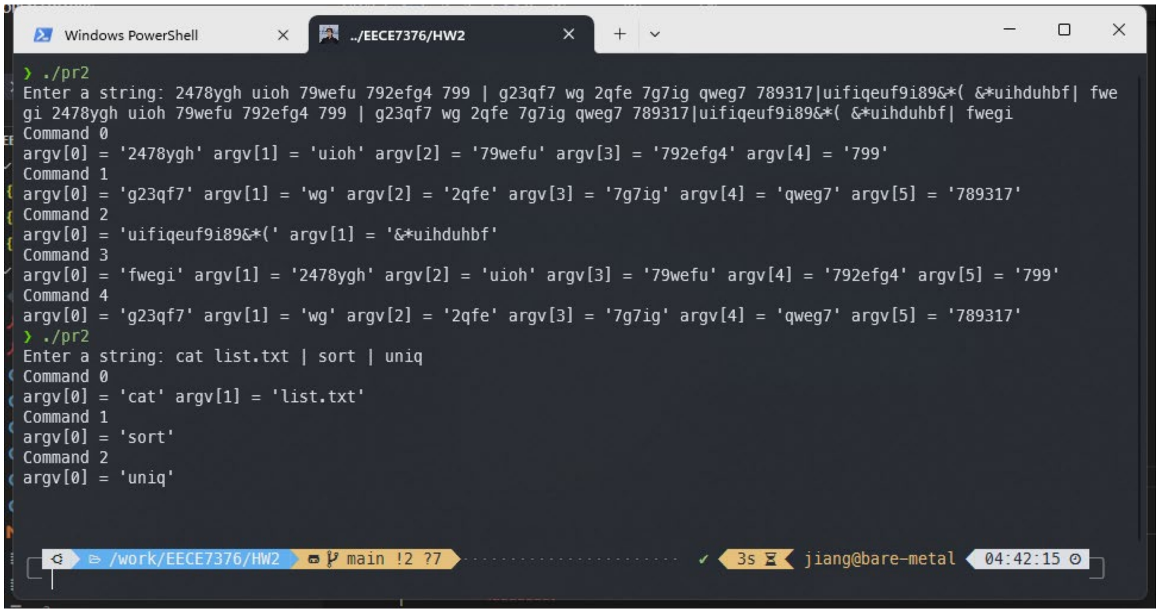

```
gcc hw2pr3.c -o pr3
❯
❯
❯
❯ ./pr1
1. Parent
1. Child
2. Parent
2. Child
3. Parent
3. Child
4. Parent
4. Child
5. Parent
5. Child
```

Problem 2

Approach Summary:

The input string from the user was read, and then split into sub-commands based on the pipe delimiter using strtok(). Each sub-command was further parsed into its arguments using a modified ReadArgs function, which now handles null-terminated arrays. Each parsed sub-command and its arguments were stored in the Command structure. Care was taken to ensure arrays were null-terminated.

Screenshot:

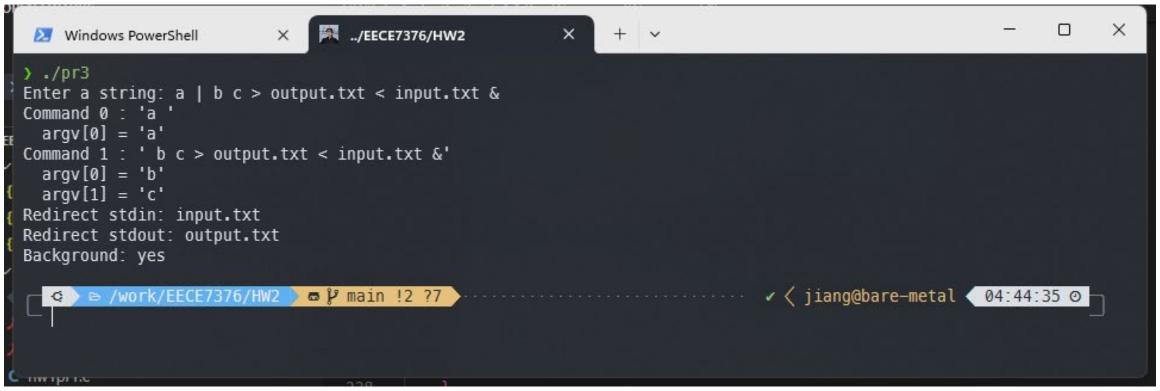

```
❯ ./pr2
Enter a string: 2478ygh uioh 79wefu 792efg4 799 | g23qf7 wg 2qfe 7g7ig qweg7 789317|uifiqeuf9i89&*( &*uihduhbf| fwe
gi 2478ygh uioh 79wefu 792efg4 799 | g23qf7 wg 2qfe 7g7ig qweg7 789317|uifiqeuf9i89&*( &*uihduhbf| fwegi
Command 0
argv[0] = '2478ygh' argv[1] = 'uioh' argv[2] = '79wefu' argv[3] = '792efg4' argv[4] = '799'
Command 1
argv[0] = 'g23qf7' argv[1] = 'wg' argv[2] = '2qfe' argv[3] = '7g7ig' argv[4] = 'qweg7' argv[5] = '789317'
Command 2
argv[0] = 'uifiqeuf9i89&*(' argv[1] = '&*uihduhbf'
Command 3
argv[0] = 'fwegi' argv[1] = '2478ygh' argv[2] = 'uioh' argv[3] = '79wefu' argv[4] = '792efg4' argv[5] = '799'
Command 4
argv[0] = 'g23qf7' argv[1] = 'wg' argv[2] = '2qfe' argv[3] = '7g7ig' argv[4] = 'qweg7' argv[5] = '789317'
❯ ./pr2
Enter a string: cat list.txt | sort | uniq
Command 0
argv[0] = 'cat' argv[1] = 'list.txt'
Command 1
argv[0] = 'sort'
Command 2
argv[0] = 'uniq'
```

Problem 3

Approach Summary:

New fields (stdin_redirect, stdout_redirect, background) were added to the Command structure to store redirection files and background execution status. A new function, ReadRedirectsAndBackground, was implemented to parse the last arguments of the last sub-command for input/output redirection and background execution symbols. The logic in ReadCommand was kept for splitting the input into sub-commands and arguments. After this, ReadRedirectsAndBackground was called to handle redirection and background execution. The PrintCommand function was extended to also print information about input/output redirection and whether the command should run in the background.

Screenshot:



```
❯ ./pr3
Enter a string: a | b c > output.txt < input.txt &
Command 0 : 'a '
  argv[0] = 'a'
Command 1 : ' b c > output.txt < input.txt &'
  argv[0] = 'b'
  argv[1] = 'c'
Redirect stdin: input.txt
Redirect stdout: output.txt
Background: yes

 /work/EECE7376/HW2   main !2 ?7 ·····························  ✓ jiang@bare-metal  04:44:35
```