CS307 Programming Assignment 2 Report

İsmet Ayvazoğlu 29493

Handling Special Characters

In my program, I did not identify any special characters and tried to escape them.

Implementing Redirectioning

For each command my program checks the redirectioning. If the redirectioning is '>', in the child process my program opens the output file to write using open() call and redirects standart output to the output file using dup2() call, then calls the execlp() call.

If the redirectioning is not '>' my program pipe(). In the child process, if the redirectioning is not '>' and it is '<' my program a) opens the input file, redirects standart input to the input1 file using dup2() call, b) redirects standart output to the write end of the pipe using dup2() call. If there is no redirection it only does the b part. Both calls execlp() call at the end. In both scenerio, in the parent process my program creates a thread and passes the pipe's read end as argument to the thread function. Thread function is wrapped with lock to provide mutual exclusion. Inside my program reads from the pipe's read end and prints to the terminal.

Implementing Background Jobs

At the beginning of the parent process my program checks if the command includes background jobs or not. If it does not include the parent process waits for the specific child process created for that command using waitpid() call. Because we use sibling hierarchy, we need to specify the the process which we will wait for (not wait(NULL)).
If the current program is wait, my program does not create a pipe or a child process. It waits all the child processes and threads that are running at that moment. The child processes running at that moment must be executing background commands since we wait the commands which are not background commands

```
pthread_t listenerThreads[MAX_LINE_LENGTH];
pid_t childPids[MAX_LINE_LENGTH];
```

I have this global variable arrays in my program. After my program creates a thread it also adds that thread's thread ID into listenerThreads array. At the end of parent process, it also adds into array the process ID of each child of its. When wait command comes up, in a for loop it joins for threads that are running, in another for loop it waits for all child processes running at that moment.

Use of Pipes, Use of Mutex

```
typedef struct {
    int fd[2];
} ppipe;


//in the main function
ppipe pipes[i]; //i: number of commands
```

I used a pipe for each command and my implementation looks like the above. I used mutex as a single lock which wraps the whole thread function.