CS307 PA2 Report

Tansylu Akhmetova 30517

My implementation consists of the following functions:

```
int executeForeground(char * in_file, char * out_file, char cmd_path[], char * argv[]);
int executeBackground(char * in_file, char * out_file, char cmd_path[], char * argv[]);
void * readPipe(void * arg);
void parse_line(char * line, char * argv[], char ** in_file, char ** out_file, bool * background);
```

main - managing the flow of the program via a “while able to read from commands.txt” loop. I also declare BackgroundInfo struct and an array of background jobs here, which I will traverse after the loop in order to join remaining jobs, wait on their children and clean up file descriptors.

```
typedef struct {
  pthread_t thread_id;
  pid_t child_pid;
  int * fd;
} BackgroundInfo;
```

parse_line - updates argv, in_file, out_file and background flag with information from commands.txt through parsing it. Also, handles the output to parse.txt

Depending on three conditions, main performs 3 different functions:

- if the command is “wait”, I join all background jobs currently present in the background jobs array
- else, if the background flag is on I call executeBackground
- if background flag is false, I call executeForeground

In both execute functions I create an array for the pipe from heap, so that each thread has its own set of file descriptors. Then, I initiate a thread with readPipe function as argument.

Inside the readPipe I set up the thread to listen to the read end of the pipe and then obtain lock on the mutex to ensure no other thread interleaves while the current one is trying to print to the console

Then I fork to create a child process that will run the parsed command, while also using dup2 calls to redirect input and output

In the foreground functions parent I wait on the child and join the thread. In the background function I do not wait or join immediately, I just create an entry in the background jobs array so that I can wait, join and cleanup at the end of main’s execution