CS307 Operating Systems

Instructor: Süha Orhun Mutluergil

Programming Assignment 1 Report

Göktuğ Gençkaya 27888

Implementing Redirectioning

My program was able to identify if there is a redirection or not and keep this redirection file's name in a specific string. If there's an > sign, meaning that program should redirect output of a command to a file, file name is stored in "outfile" variable. Standard output of terminal is closed and given outpu file is opened so that the printing ocur in that file and not to the console. If there's < sing, meaning that program should redirect input of a file to a command, input file name is stored in "inpfile" variable. I seacrhed the internet and found that we should first open a desired, given input file and then create a copy of given file descriptor and assign a new integer to it. Similar as outputs, we change the standard input of terminal this time.

```
if (outfile != NULL){
    close(STDOUT_FILENO);
    open(outfile, O_CREAT|O_WRONLY|O_TRUNC, S_IRWXU);
}
if (inpfile != NULL){
    int fd0 = open(inpfile, O_RDONLY, 0);
    dup2(fd0, STDIN_FILENO);
    close(fd0);
```

Background Jobs

Whenever there is an ampersand symbol in the command, my program were able to store it in a variable called "symbol". If this symbol is NULL, this means that the current command doesn't include an ampersand in it. I created an array of pids in which I collect pid's of the commands that are supposed to work in background, child processes that are working in the background. Whenever a wait command is entered, these processes in the background should be waited. So, if the current command is "wait" (which I checked by checking the first and second letter of it), it'll go over every process and wait each of them to end.

```
if(symbol != NULL){
    arrcmd[imm]= (int)getpid();
    imm++;
```

```
int is= 0;
while (is<imm) {
    waitpid(arrcmd[is], NULL, 0);
    is++;
}
```