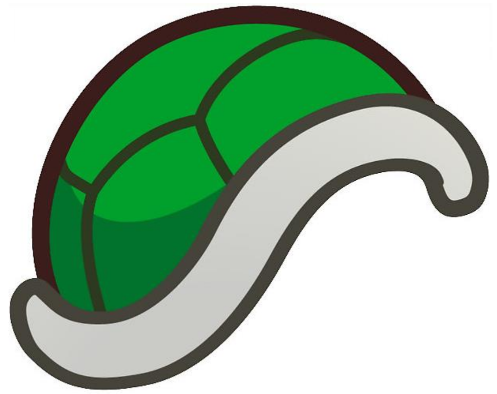
Sarah's Shell

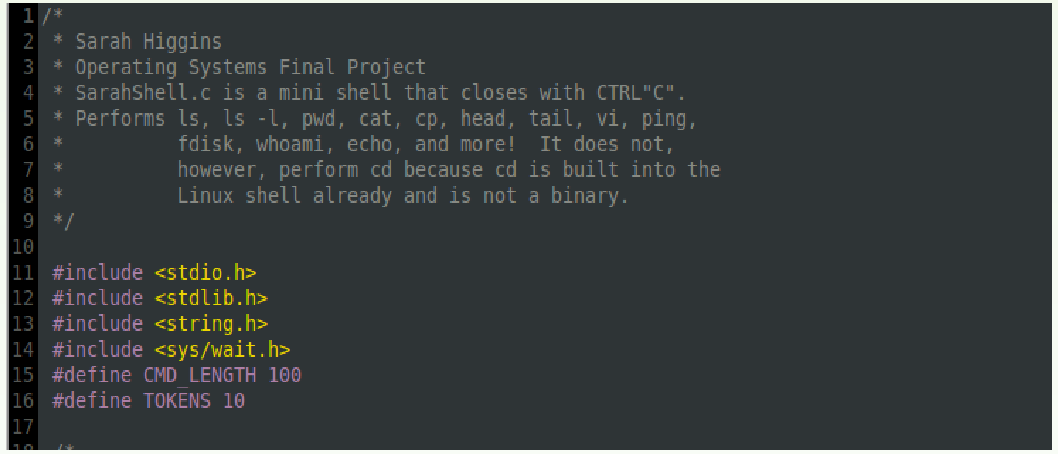
SARAH HIGGINS
OPERATING SYSTEMS
FINAL PROJECT PRESENTATION



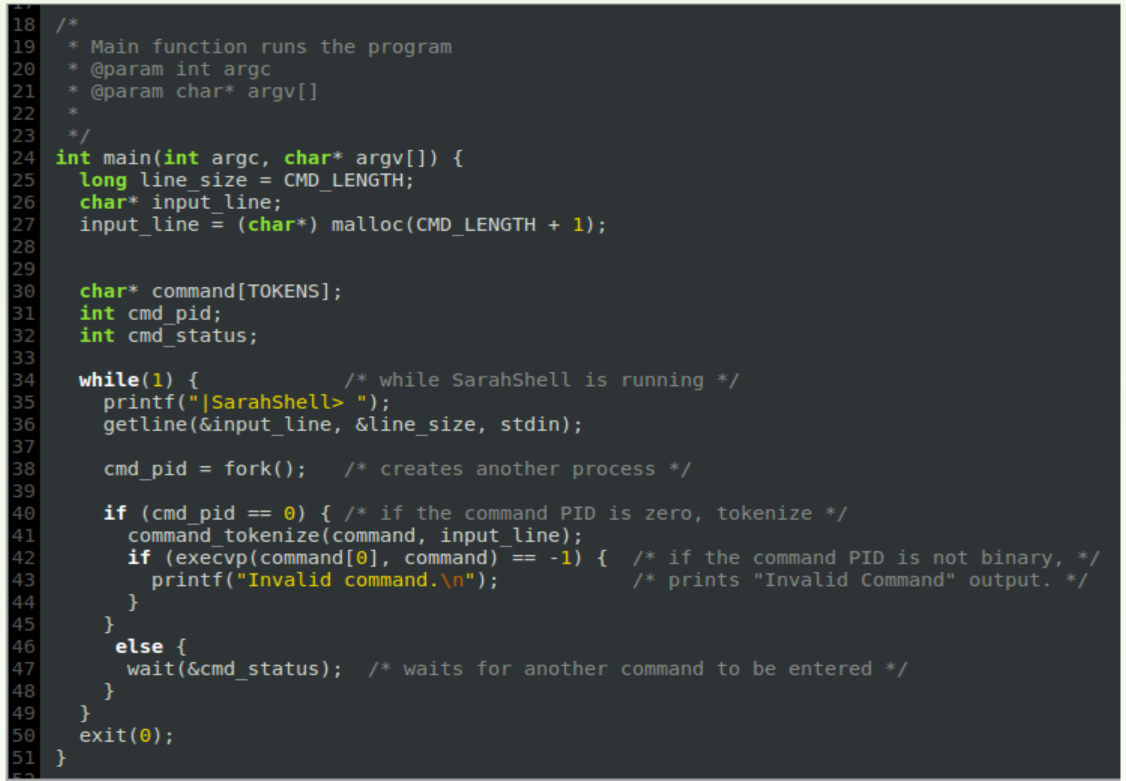
The Linux Shell

- The Shell is a program that takes commands from the keyboard from the user and sends them to the Operating System to perform.
- The Shell is not a part of the Kernel, but uses the kernel system to execute the programs and create files.
- The user can access the Shell through the command line, or Terminal.

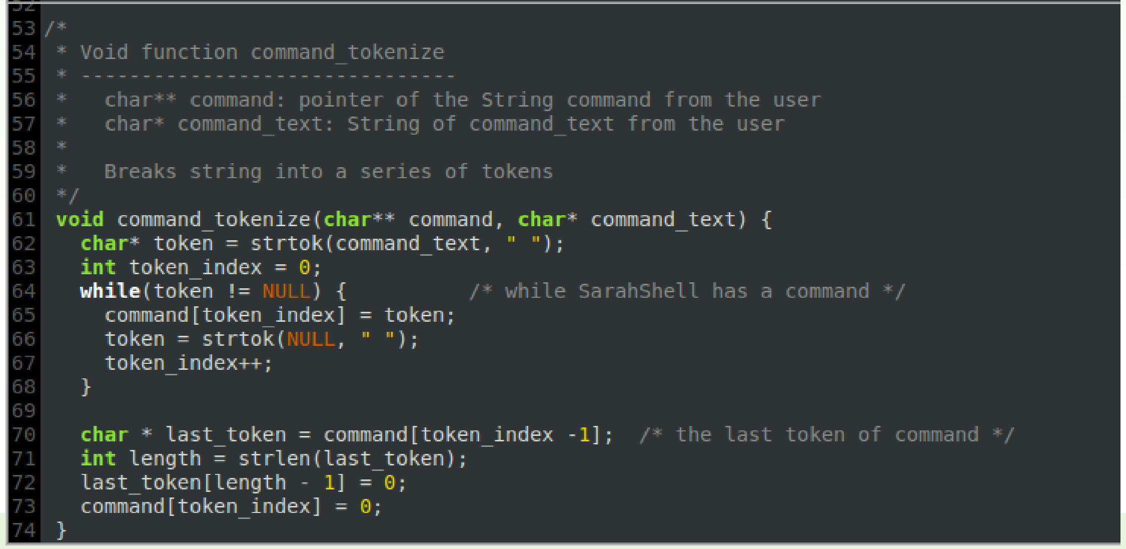
The Shell



```c
/*
 * Sarah Higgins
 * Operating Systems Final Project
 * SarahShell.c is a mini shell that closes with CTRL"C".
 * Performs ls, ls -l, pwd, cat, cp, head, tail, vi, ping,
 *          fdisk, whoami, echo, and more!  It does not,
 *          however, perform cd because cd is built into the
 *          Linux shell already and is not a binary.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>
#define CMD_LENGTH 100
#define TOKENS 10

```



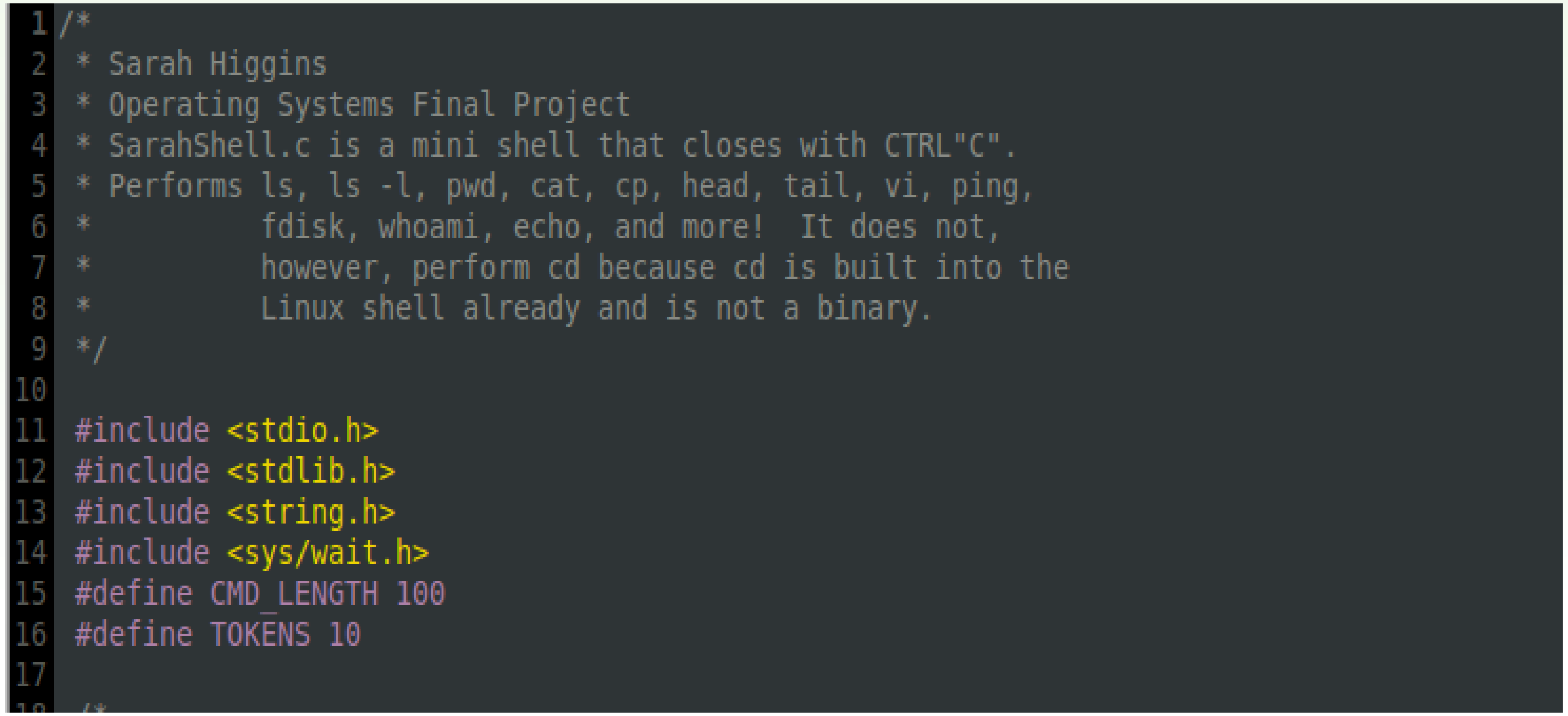
```c
/*
 * Main function runs the program
 * @param int argc
 * @param char* argv[]
 *
 */
int main(int argc, char* argv[]) {
  long line_size = CMD_LENGTH;
  char* input_line;
  input_line = (char*) malloc(CMD_LENGTH + 1);


  char* command[TOKENS];
  int cmd_pid;
  int cmd_status;

  while(1) {            /* while SarahShell is running */
    printf("|SarahShell> ");
    getline(&input_line, &line_size, stdin);

    cmd_pid = fork();   /* creates another process */

    if (cmd_pid == 0) { /* if the command PID is zero, tokenize */
      command_tokenize(command, input_line);
      if (execvp(command[0], command) == -1) {  /* if the command PID is not binary, */
        printf("Invalid command.\n");           /* prints "Invalid Command" output. */
      }
    }
     else {
      wait(&cmd_status);  /* waits for another command to be entered */
    }
  }
  exit(0);
}
```



```c
/*
 * Void function command_tokenize
 * -------------------------------
 *   char** command: pointer of the String command from the user
 *   char* command_text: String of command_text from the user
 *
 *   Breaks string into a series of tokens
 */
void command_tokenize(char** command, char* command_text) {
  char* token = strtok(command_text, " ");
  int token_index = 0;
  while(token != NULL) {          /* while SarahShell has a command */
    command[token_index] = token;
    token = strtok(NULL, " ");
    token_index++;
  }

  char * last_token = command[token_index -1];  /* the last token of command */
  int length = strlen(last_token);
  last_token[length - 1] = 0;
  command[token_index] = 0;
}
```

The Shell Components

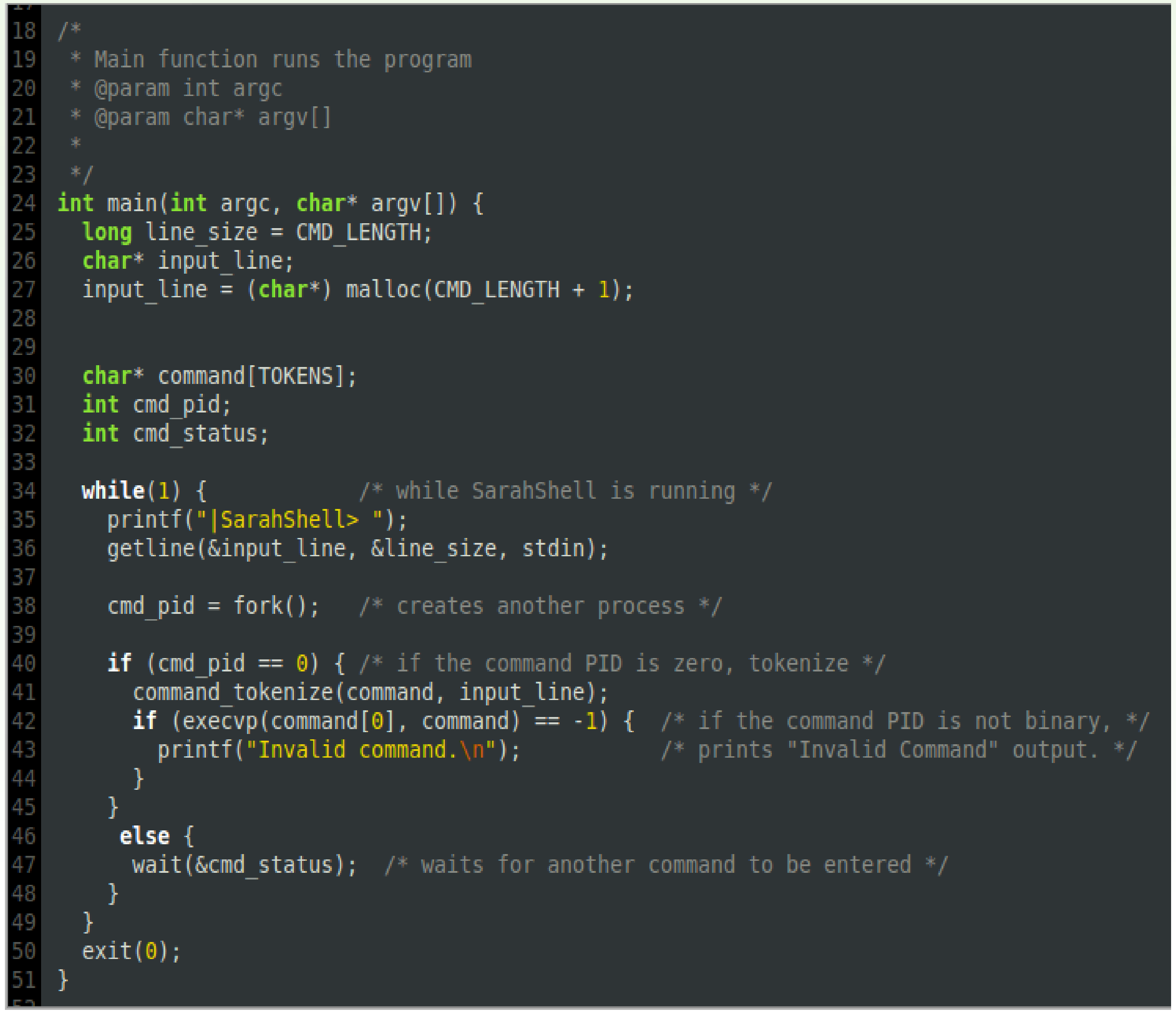
- Header comments and include statements



```
/*
 * Sarah Higgins
 * Operating Systems Final Project
 * SarahShell.c is a mini shell that closes with CTRL"C".
 * Performs ls, ls -l, pwd, cat, cp, head, tail, vi, ping,
 *          fdisk, whoami, echo, and more!  It does not,
 *          however, perform cd because cd is built into the
 *          Linux shell already and is not a binary.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>
#define CMD_LENGTH 100
#define TOKENS 10

```

The Shell Components

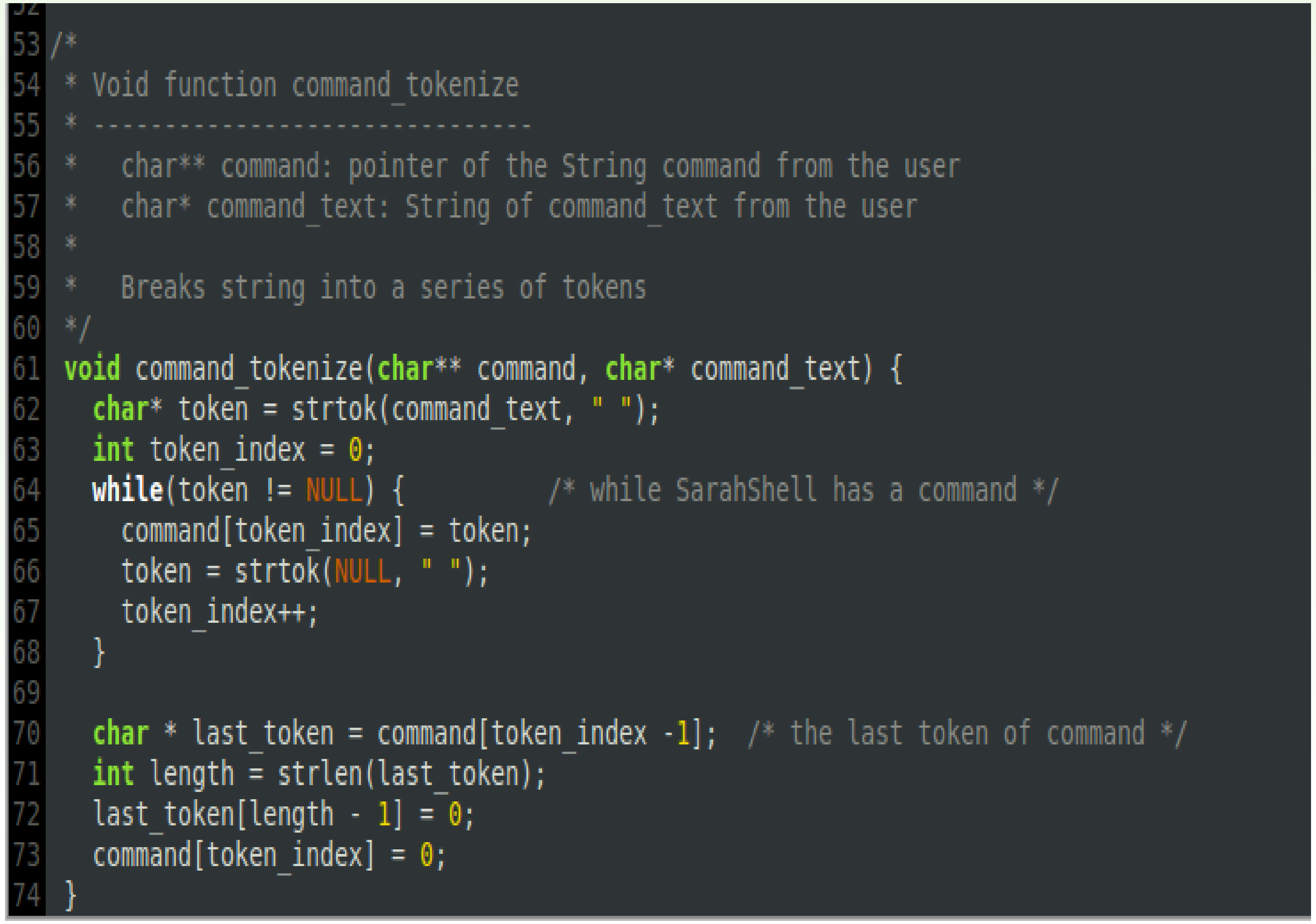
- Main function which runs the shell
- While the shell is running, it prints “SarahShell>” to the terminal for the user to type in their command.
- If the command PID == 0, it will tokenize the command. Else, will print “Invalid Command”.



```
/*
 * Main function runs the program
 * @param int argc
 * @param char* argv[]
 *
 */
int main(int argc, char* argv[]) {
  long line_size = CMD_LENGTH;
  char* input_line;
  input_line = (char*) malloc(CMD_LENGTH + 1);


  char* command[TOKENS];
  int cmd_pid;
  int cmd_status;

  while(1) {            /* while SarahShell is running */
    printf("|SarahShell> ");
    getline(&input_line, &line_size, stdin);

    cmd_pid = fork();   /* creates another process */

    if (cmd_pid == 0) { /* if the command PID is zero, tokenize */
      command_tokenize(command, input_line);
      if (execvp(command[0], command) == -1) {  /* if the command PID is not binary, */
        printf("Invalid command.\n");           /* prints "Invalid Command" output. */
      }
    }
     else {
      wait(&cmd_status);  /* waits for another command to be entered */
    }
  }
  exit(0);
}
```

The Shell Components

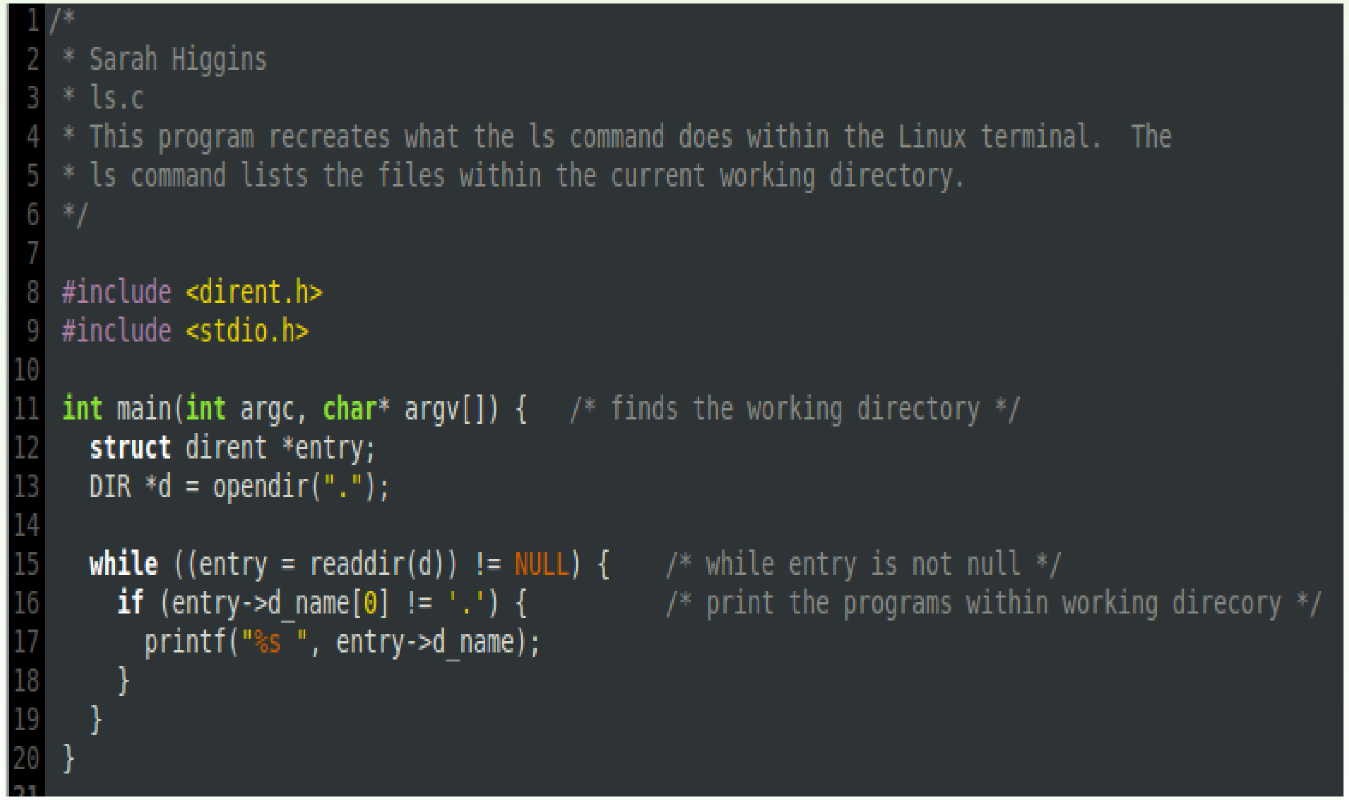
- Command_tokenize() breaks the command String typed in by the user into tokens.
- Command_tokenize() does this while SarahShell has a command
- Else, the last token of the command has a token_index of 0.



```
/*
 * Void function command_tokenize
 * ------------------------------
 *   char** command: pointer of the String command from the user
 *   char* command_text: String of command_text from the user
 *
 *   Breaks string into a series of tokens
 */
void command_tokenize(char** command, char* command_text) {
  char* token = strtok(command_text, " ");
  int token_index = 0;
  while(token != NULL) {          /* while SarahShell has a command */
    command[token_index] = token;
    token = strtok(NULL, " ");
    token_index++;
  }

  char * last_token = command[token_index -1];  /* the last token of command */
  int length = strlen(last_token);
  last_token[length - 1] = 0;
  command[token_index] = 0;
}
```

Commands: ls

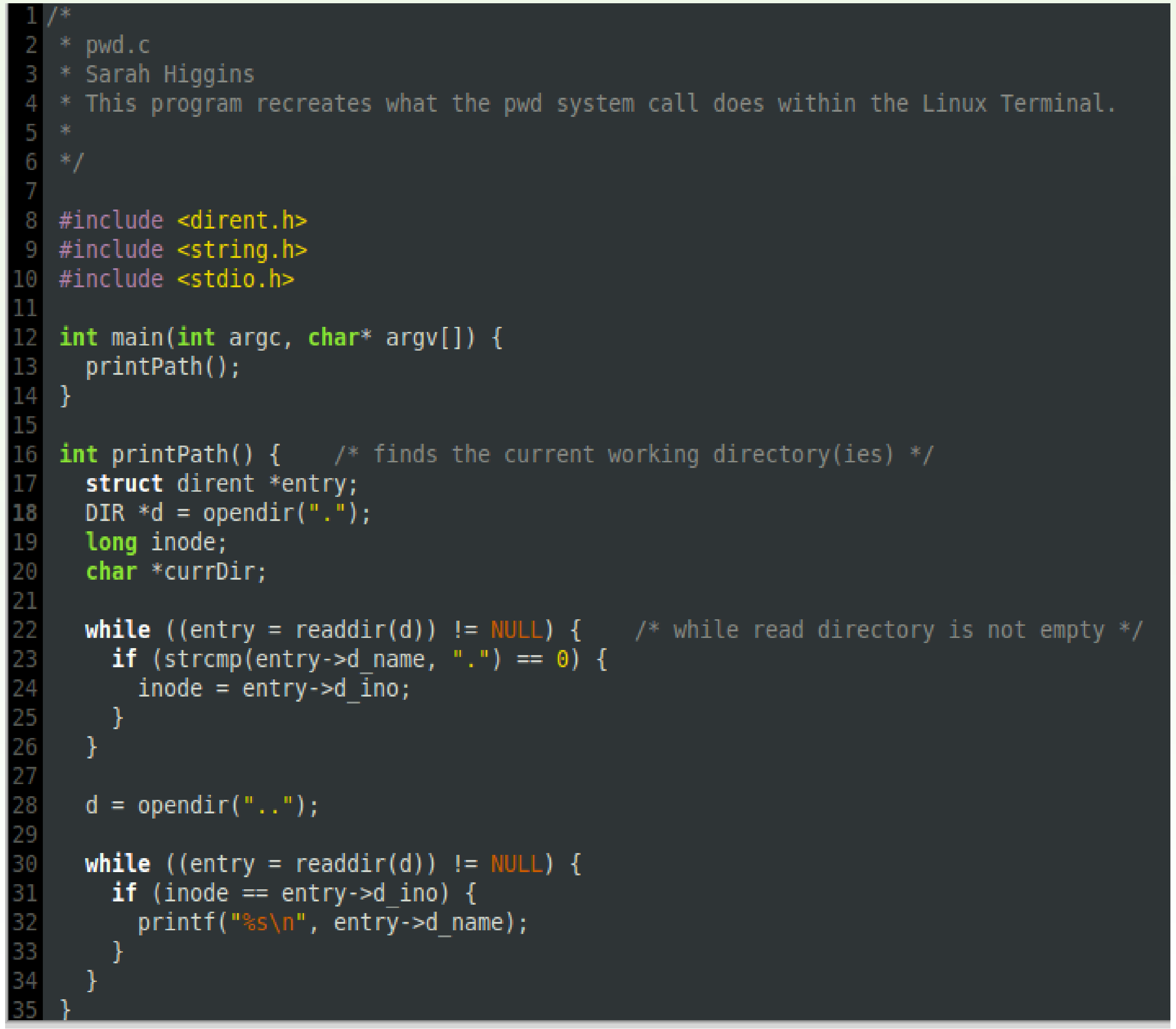
- #include <dirent.h> includes the directory stream from the current user into the code file for ls to properly work.
- Main() function locates the current working directory.
- While the command entry is not null, show the names of the programs and files within the working directory.



```
/*
 * Sarah Higgins
 * ls.c
 * This program recreates what the ls command does within the Linux terminal.  The
 * ls command lists the files within the current working directory.
 */

 #include <dirent.h>
 #include <stdio.h>

 int main(int argc, char* argv[]) {   /* finds the working directory */
   struct dirent *entry;
   DIR *d = opendir(".");

   while ((entry = readdir(d)) != NULL) {    /* while entry is not null */
     if (entry->d_name[0] != '.') {          /* print the programs within working direcory */
       printf("%s ", entry->d_name);
     }
   }
 }
```

Commands: pwd

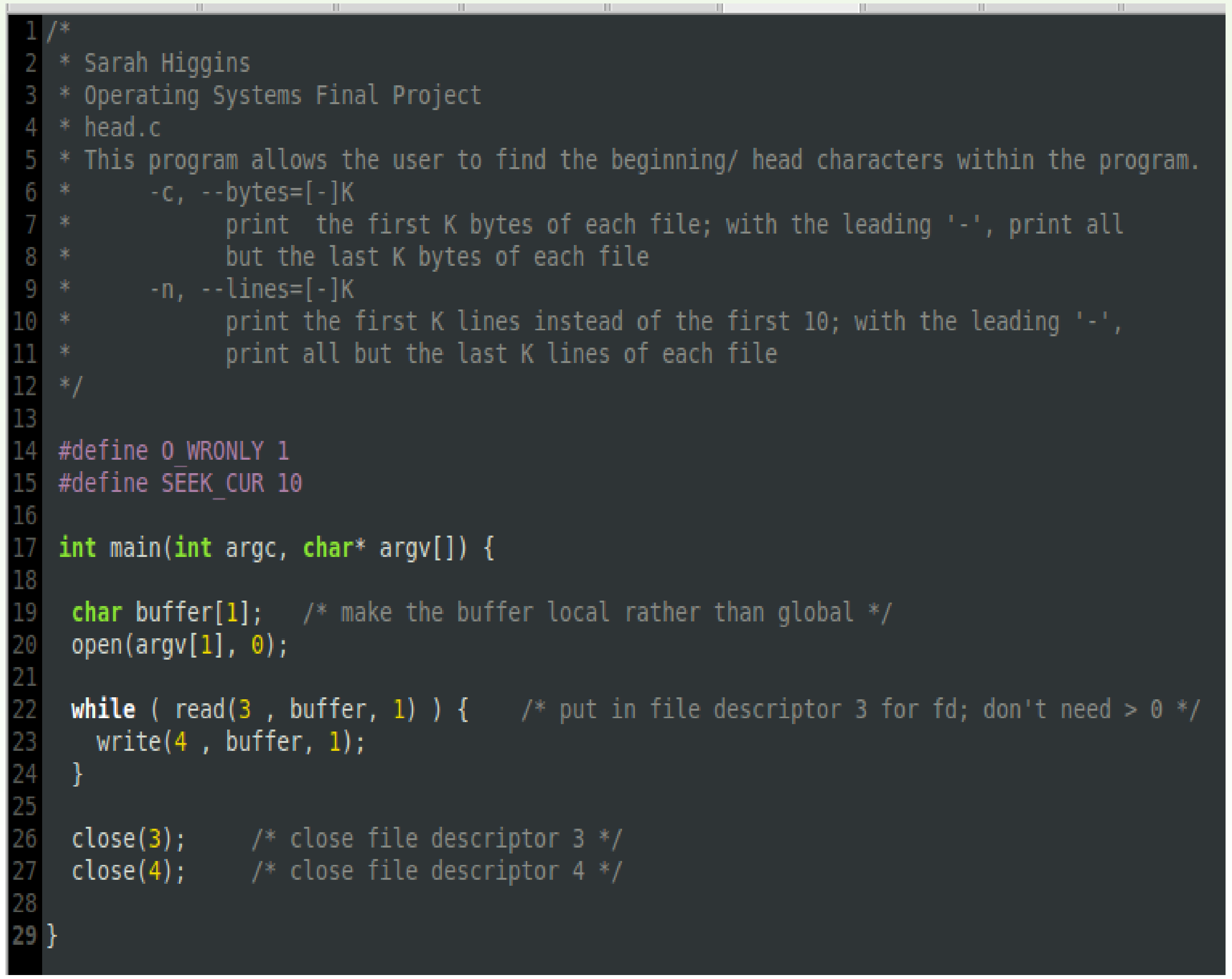
- Main() function prints the path of current working directories
- printPath() function opens the current working directory.
- While reading the directory is not null, inode = the directory components.
- While reading the directory is not null, print the inodes of the working directory.



```c
/*
 * pwd.c
 * Sarah Higgins
 * This program recreates what the pwd system call does within the Linux Terminal.
 *
 */

#include <dirent.h>
#include <string.h>
#include <stdio.h>

int main(int argc, char* argv[]) {
  printPath();
}

int printPath() {    /* finds the current working directory(ies) */
  struct dirent *entry;
  DIR *d = opendir(".");
  long inode;
  char *currDir;

  while ((entry = readdir(d)) != NULL) {    /* while read directory is not empty */
    if (strcmp(entry->d_name, ".") == 0) {
      inode = entry->d_ino;
    }
  }

  d = opendir("..");

  while ((entry = readdir(d)) != NULL) {
    if (inode == entry->d_ino) {
      printf("%s\n", entry->d_name);
    }
  }
}
```

Commands: head

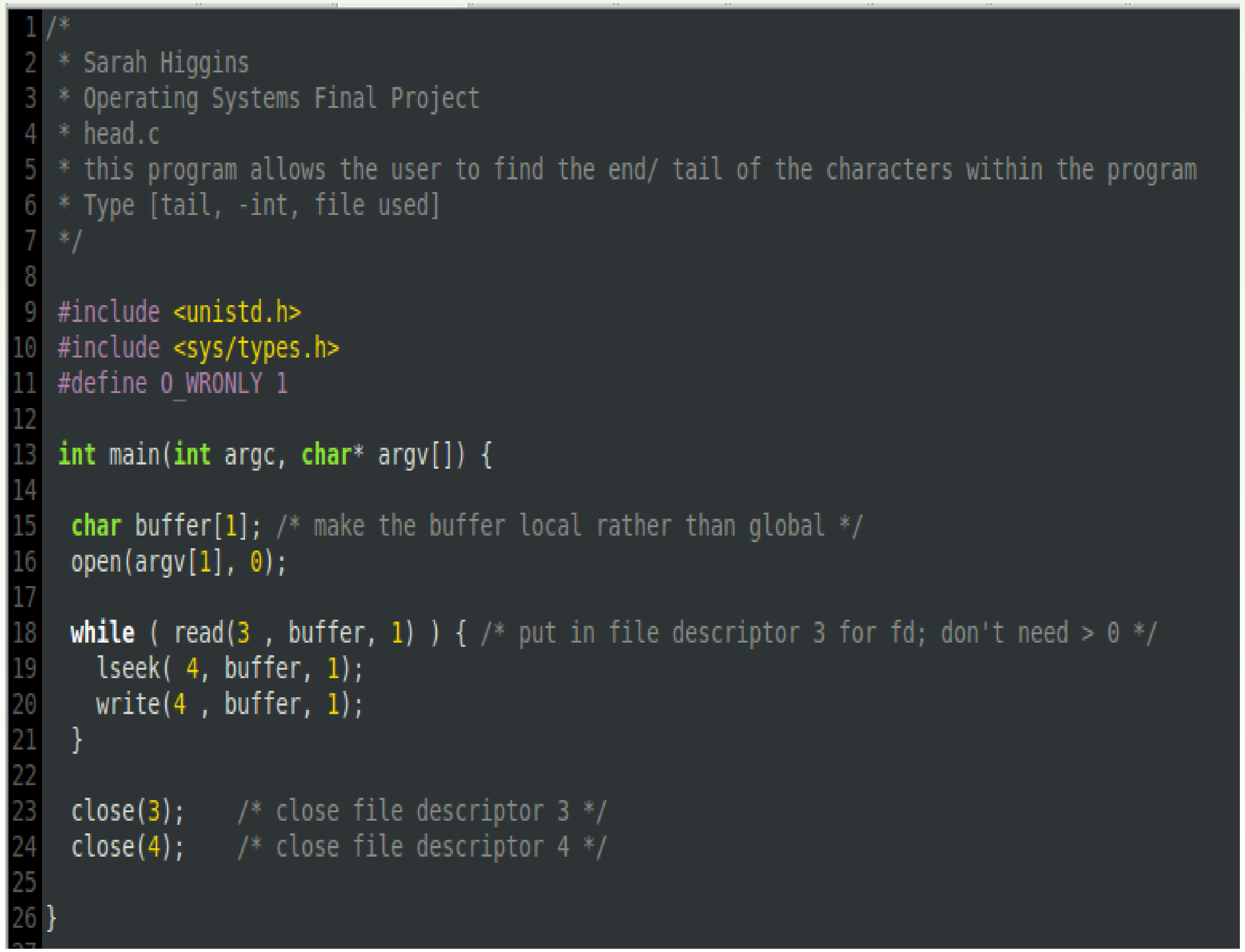
- #define O_WRONLY 1 makes the command with argument of File Descriptor 1.
- Main() function creates a local buffer and opens File Descriptor 1.
- While the program reads from the open file, it reads to File Descriptor 3 from the buffer and File Descriptor 1, and writes to File Descriptor 4 the contents of what is open.
- Closes File Descriptors 3 and 4.



```
/*
 * Sarah Higgins
 * Operating Systems Final Project
 * head.c
 * This program allows the user to find the beginning/ head characters within the program.
 *      -c, --bytes=[-]K
 *            print  the first K bytes of each file; with the leading '-', print all
 *            but the last K bytes of each file
 *      -n, --lines=[-]K
 *            print the first K lines instead of the first 10; with the leading '-',
 *            print all but the last K lines of each file
 */

 #define O_WRONLY 1
 #define SEEK_CUR 10

 int main(int argc, char* argv[]) {

  char buffer[1];   /* make the buffer local rather than global */
  open(argv[1], 0);

  while ( read(3 , buffer, 1) ) {    /* put in file descriptor 3 for fd; don't need > 0 */
    write(4 , buffer, 1);
  }

  close(3);     /* close file descriptor 3 */
  close(4);     /* close file descriptor 4 */

}
```

Commands: tail

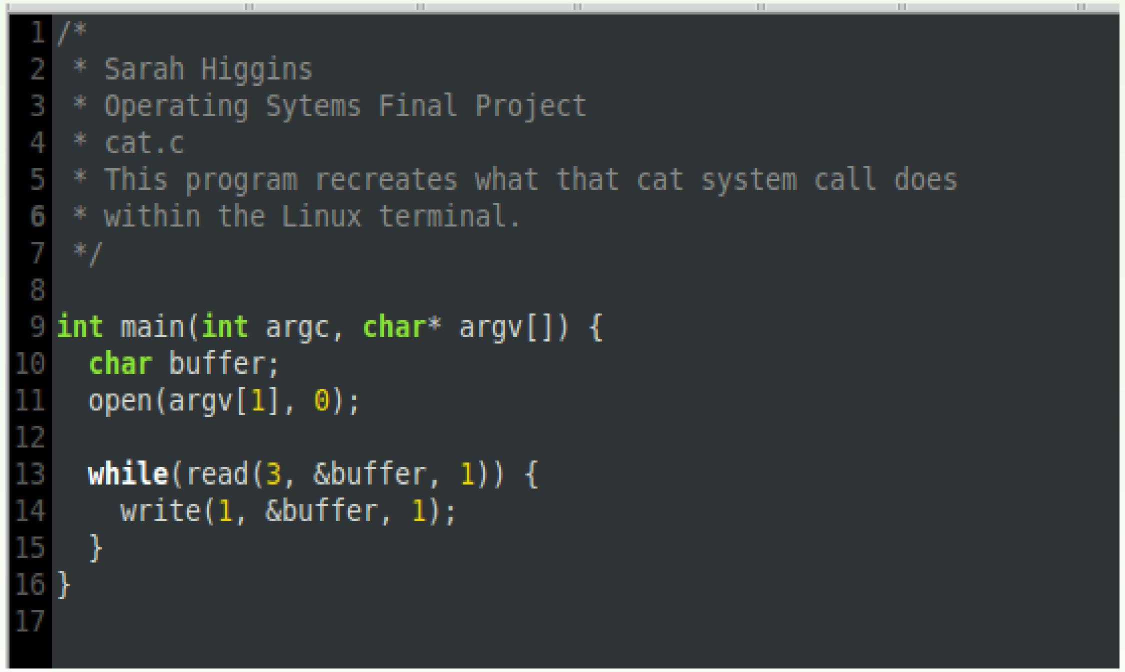
- While reading from File Descriptor 3 of the chosen file, lseek() reads from the bottom/ tail of the file rather than the front/ head of it.
- Write to File Descriptor 4 what the lseek() finds at the end of the file.
- Close File Descriptors 3 and 4.



```
/*
 * Sarah Higgins
 * Operating Systems Final Project
 * head.c
 * this program allows the user to find the end/ tail of the characters within the program
 * Type [tail, -int, file used]
 */

 #include <unistd.h>
 #include <sys/types.h>
 #define O_WRONLY 1

 int main(int argc, char* argv[]) {

  char buffer[1]; /* make the buffer local rather than global */
  open(argv[1], 0);

  while ( read(3 , buffer, 1) ) { /* put in file descriptor 3 for fd; don't need > 0 */
    lseek( 4, buffer, 1);
    write(4 , buffer, 1);
  }

  close(3);    /* close file descriptor 3 */
  close(4);    /* close file descriptor 4 */

}
```

Commands: cat

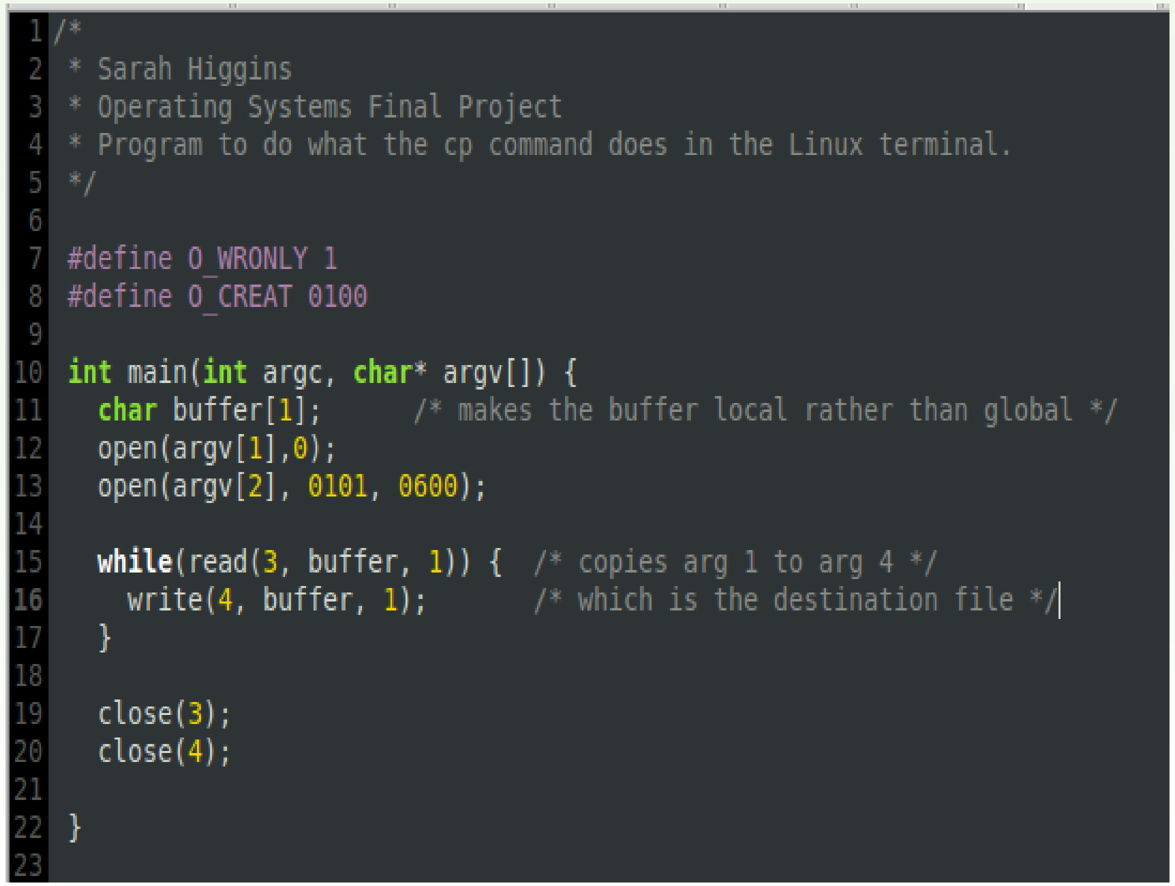
- Main() function creates a buffer character that reads from the open File Descriptor 3 (the file), and writes to File Descriptor 1 what is in the file to the user.



```
/*
 * Sarah Higgins
 * Operating Sytems Final Project
 * cat.c
 * This program recreates what that cat system call does
 * within the Linux terminal.
 */

int main(int argc, char* argv[]) {
  char buffer;
  open(argv[1], 0);

  while(read(3, &buffer, 1)) {
    write(1, &buffer, 1);
  }
}

```

Commands: cp

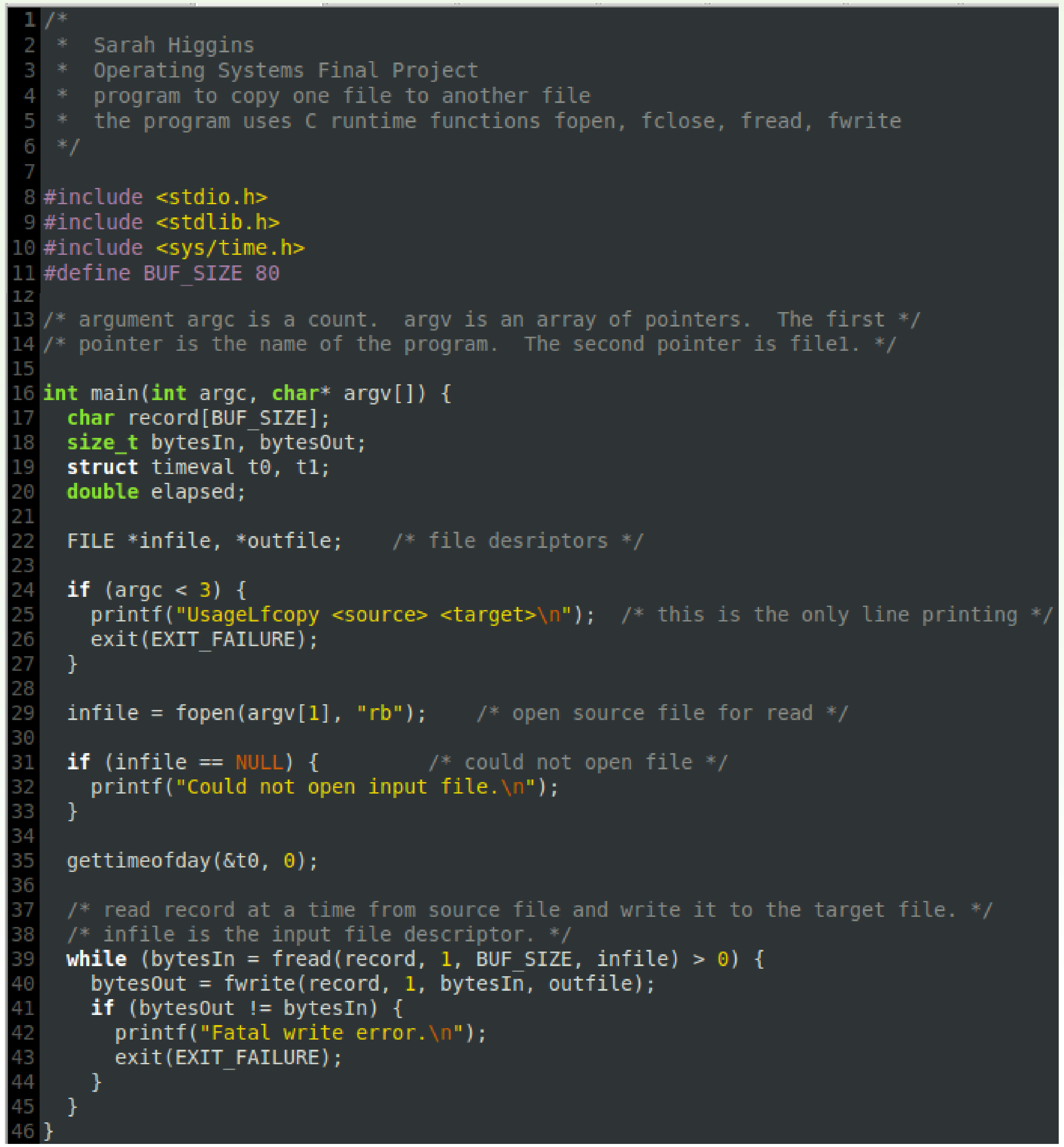
- Main() function creates a local buffer character
- Opens the file, copies the contents of what is in File Descriptor 3 (the file) to File Descriptor 4 (the copy of the file).



```
/*
 * Sarah Higgins
 * Operating Systems Final Project
 * Program to do what the cp command does in the Linux terminal.
 */

#define O_WRONLY 1
#define O_CREAT 0100

int main(int argc, char* argv[]) {
  char buffer[1];      /* makes the buffer local rather than global */
  open(argv[1],0);
  open(argv[2], 0101, 0600);

  while(read(3, buffer, 1)) {  /* copies arg 1 to arg 4 */
    write(4, buffer, 1);       /* which is the destination file */
  }

  close(3);
  close(4);

}
```

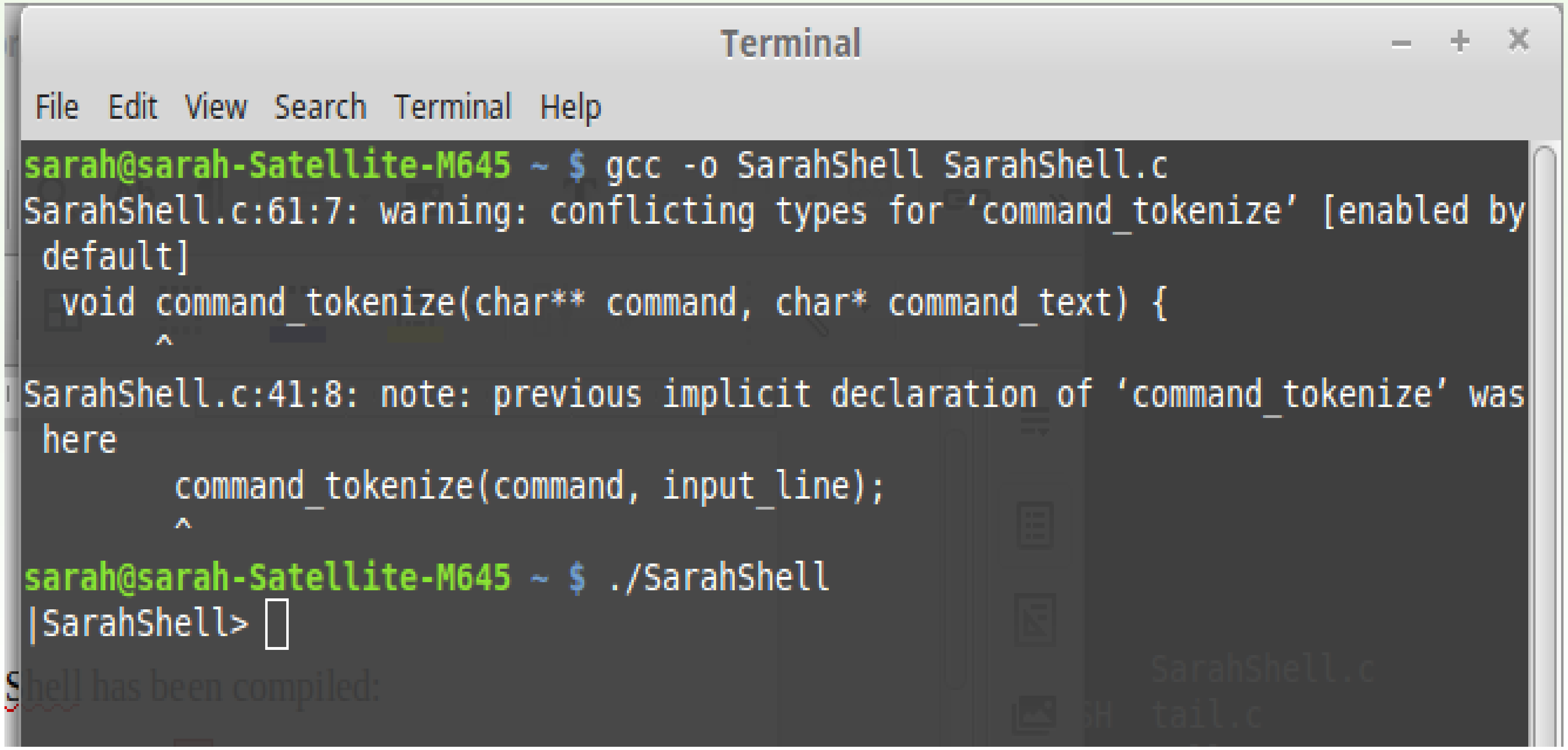
Commands: cp (lab2)

- Main() function creates a record buffer character to read the contents from file1 to file2.
- If the argument is less than 3, copies the contents of file1 (source) to the destination (target).



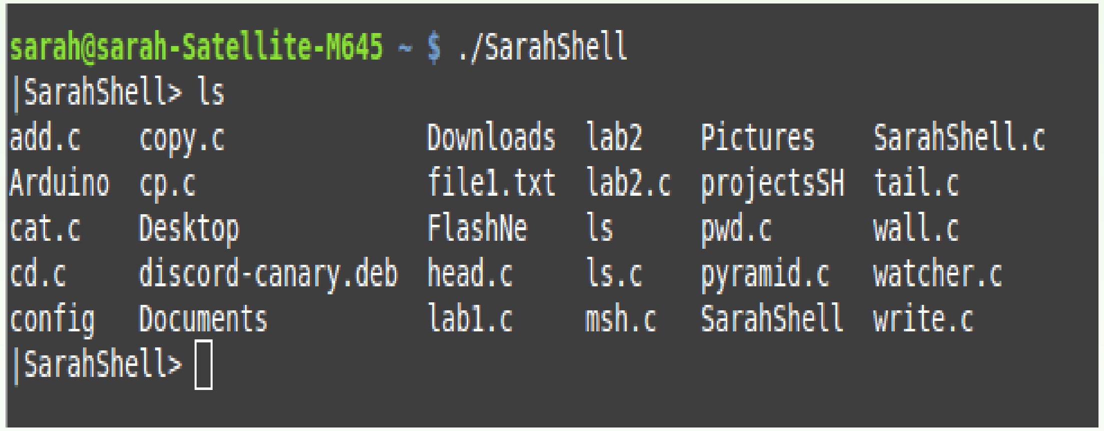
```c
/*
 *  Sarah Higgins
 *  Operating Systems Final Project
 *  program to copy one file to another file
 *  the program uses C runtime functions fopen, fclose, fread, fwrite
 */

#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#define BUF_SIZE 80

/* argument argc is a count.  argv is an array of pointers.  The first */
/* pointer is the name of the program.  The second pointer is file1. */

int main(int argc, char* argv[]) {
  char record[BUF_SIZE];
  size_t bytesIn, bytesOut;
  struct timeval t0, t1;
  double elapsed;

  FILE *infile, *outfile;    /* file desriptors */

  if (argc < 3) {
    printf("UsageLfcopy <source> <target>\n");  /* this is the only line printing */
    exit(EXIT_FAILURE);
  }

  infile = fopen(argv[1], "rb");    /* open source file for read */

  if (infile == NULL) {         /* could not open file */
    printf("Could not open input file.\n");
  }

  gettimeofday(&t0, 0);

  /* read record at a time from source file and write it to the target file. */
  /* infile is the input file descriptor. */
  while (bytesIn = fread(record, 1, BUF_SIZE, infile) > 0) {
    bytesOut = fwrite(record, 1, bytesIn, outfile);
    if (bytesOut != bytesIn) {
      printf("Fatal write error.\n");
      exit(EXIT_FAILURE);
    }
  }
}
```

Command Output: SarahShell

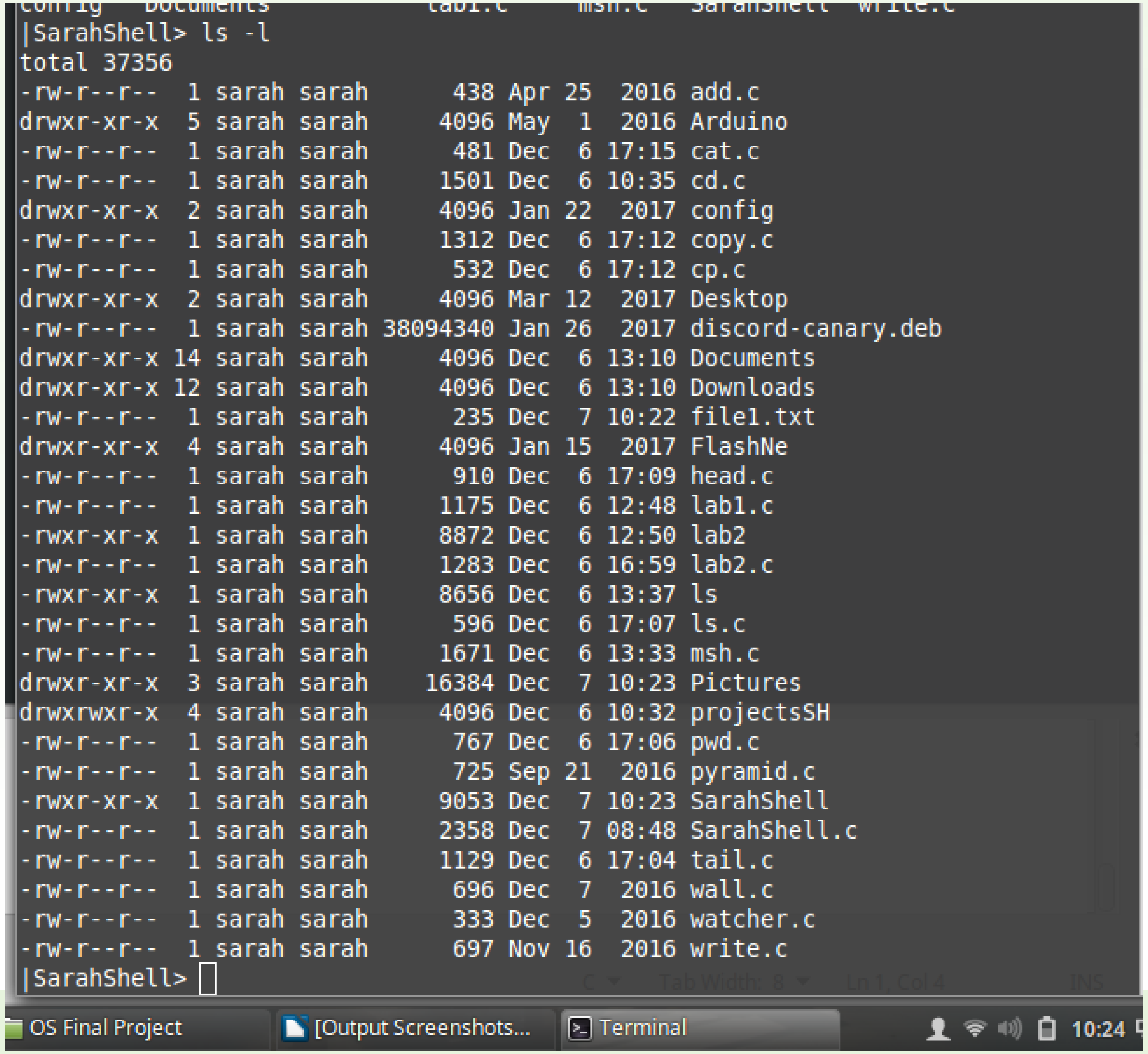


```
sarah@sarah-Satellite-M645 ~ $ gcc -o SarahShell SarahShell.c
SarahShell.c:61:7: warning: conflicting types for ‘command_tokenize’ [enabled by
 default]
  void command_tokenize(char** command, char* command_text) {
       ^
SarahShell.c:41:8: note: previous implicit declaration of ‘command_tokenize’ was
 here
        command_tokenize(command, input_line);
        ^
sarah@sarah-Satellite-M645 ~ $ ./SarahShell
|SarahShell> 
```

Command Output: ls and ls -l

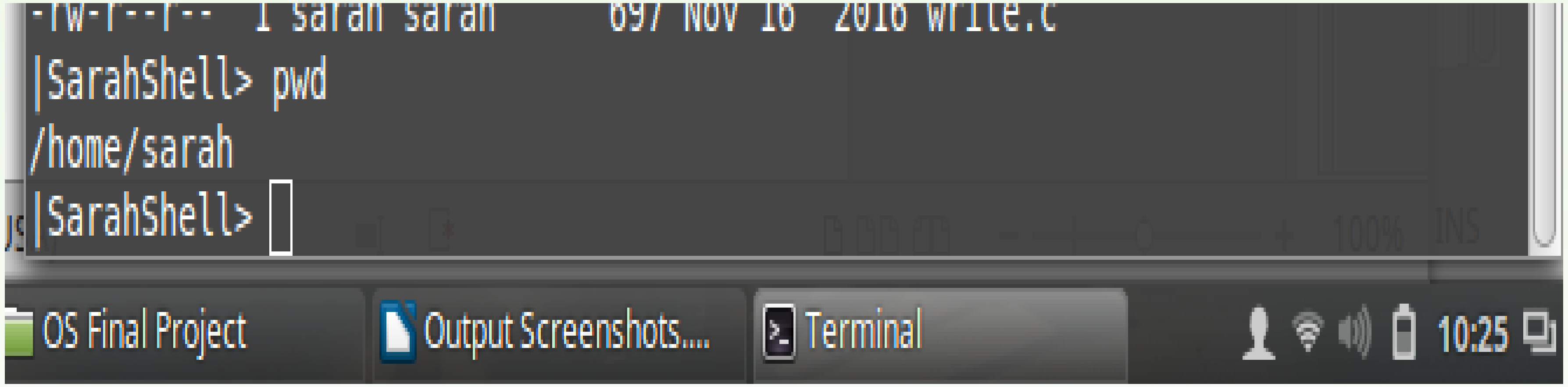


```
sarah@sarah-Satellite-M645 ~ $ ./SarahShell
|SarahShell> ls
add.c    copy.c            Downloads  lab2    Pictures    SarahShell.c
Arduino  cp.c              file1.txt  lab2.c  projectsSH  tail.c
cat.c    Desktop           FlashNe    ls      pwd.c       wall.c
cd.c     discord-canary.deb head.c    ls.c    pyramid.c   watcher.c
config   Documents         lab1.c     msh.c   SarahShell  write.c
|SarahShell> 
```



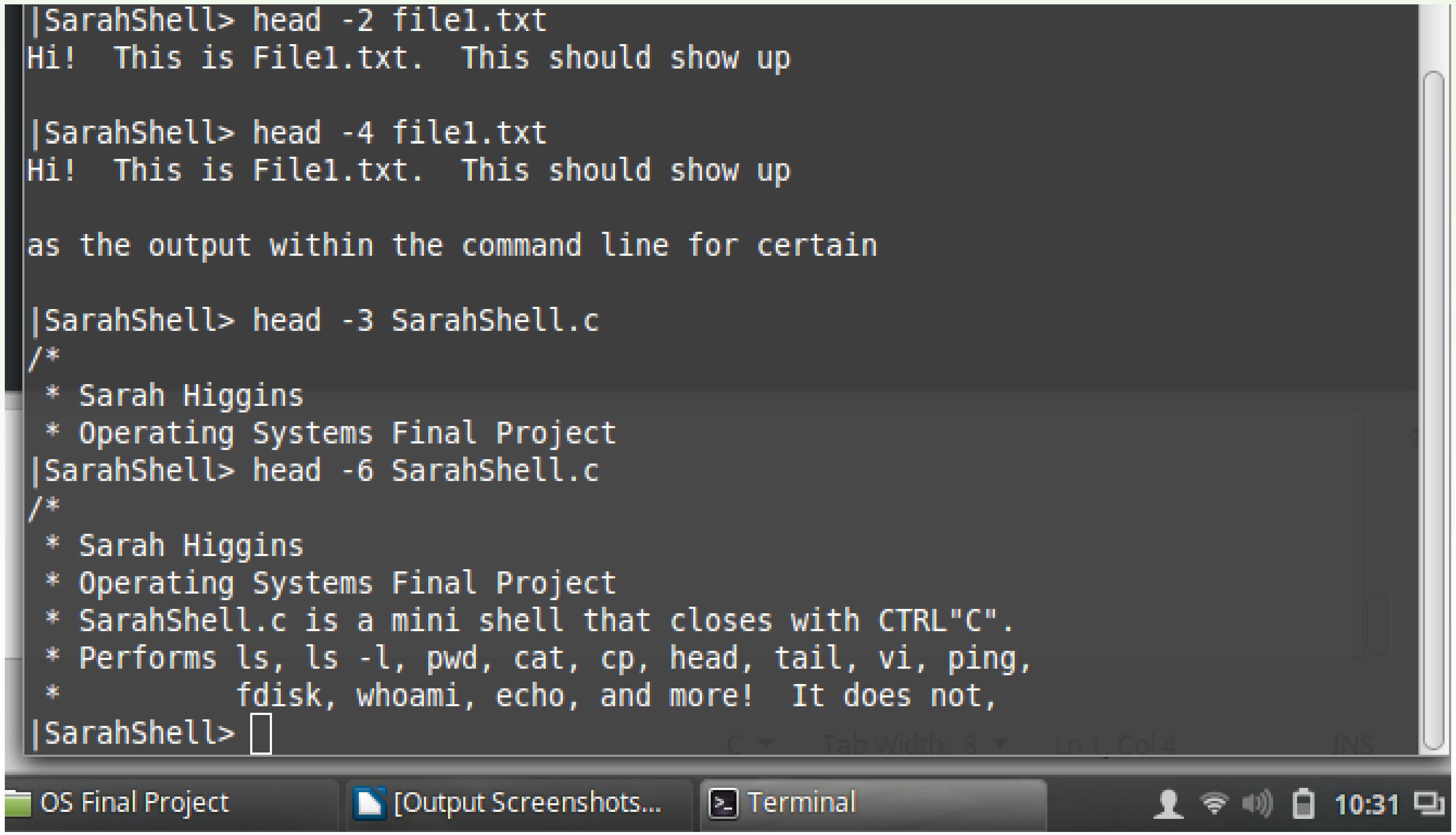
```
config   Documents          lab1.c    msh.c   SarahShell  write.c
|SarahShell> ls -l
total 37356
-rw-r--r--  1 sarah sarah      438 Apr 25  2016 add.c
drwxr-xr-x  5 sarah sarah     4096 May  1  2016 Arduino
-rw-r--r--  1 sarah sarah      481 Dec  6 17:15 cat.c
-rw-r--r--  1 sarah sarah     1501 Dec  6 10:35 cd.c
drwxr-xr-x  2 sarah sarah     4096 Jan 22  2017 config
-rw-r--r--  1 sarah sarah     1312 Dec  6 17:12 copy.c
-rw-r--r--  1 sarah sarah      532 Dec  6 17:12 cp.c
drwxr-xr-x  2 sarah sarah     4096 Mar 12  2017 Desktop
-rw-r--r--  1 sarah sarah 38094340 Jan 26  2017 discord-canary.deb
drwxr-xr-x 14 sarah sarah     4096 Dec  6 13:10 Documents
drwxr-xr-x 12 sarah sarah     4096 Dec  6 13:10 Downloads
-rw-r--r--  1 sarah sarah      235 Dec  7 10:22 file1.txt
drwxr-xr-x  4 sarah sarah     4096 Jan 15  2017 FlashNe
-rw-r--r--  1 sarah sarah      910 Dec  6 17:09 head.c
-rw-r--r--  1 sarah sarah     1175 Dec  6 12:48 lab1.c
-rwxr-xr-x  1 sarah sarah     8872 Dec  6 12:50 lab2
-rw-r--r--  1 sarah sarah     1283 Dec  6 16:59 lab2.c
-rwxr-xr-x  1 sarah sarah     8656 Dec  6 13:37 ls
-rw-r--r--  1 sarah sarah      596 Dec  6 17:07 ls.c
-rw-r--r--  1 sarah sarah     1671 Dec  6 13:33 msh.c
drwxr-xr-x  3 sarah sarah    16384 Dec  7 10:23 Pictures
drwxrwxr-x  4 sarah sarah     4096 Dec  6 10:32 projectsSH
-rw-r--r--  1 sarah sarah      767 Dec  6 17:06 pwd.c
-rw-r--r--  1 sarah sarah      725 Sep 21  2016 pyramid.c
-rwxr-xr-x  1 sarah sarah     9053 Dec  7 10:23 SarahShell
-rw-r--r--  1 sarah sarah     2358 Dec  7 08:48 SarahShell.c
-rw-r--r--  1 sarah sarah     1129 Dec  6 17:04 tail.c
-rw-r--r--  1 sarah sarah      696 Dec  7  2016 wall.c
-rw-r--r--  1 sarah sarah      333 Dec  5  2016 watcher.c
-rw-r--r--  1 sarah sarah      697 Nov 16  2016 write.c
|SarahShell> 
```

Command Output: pwd



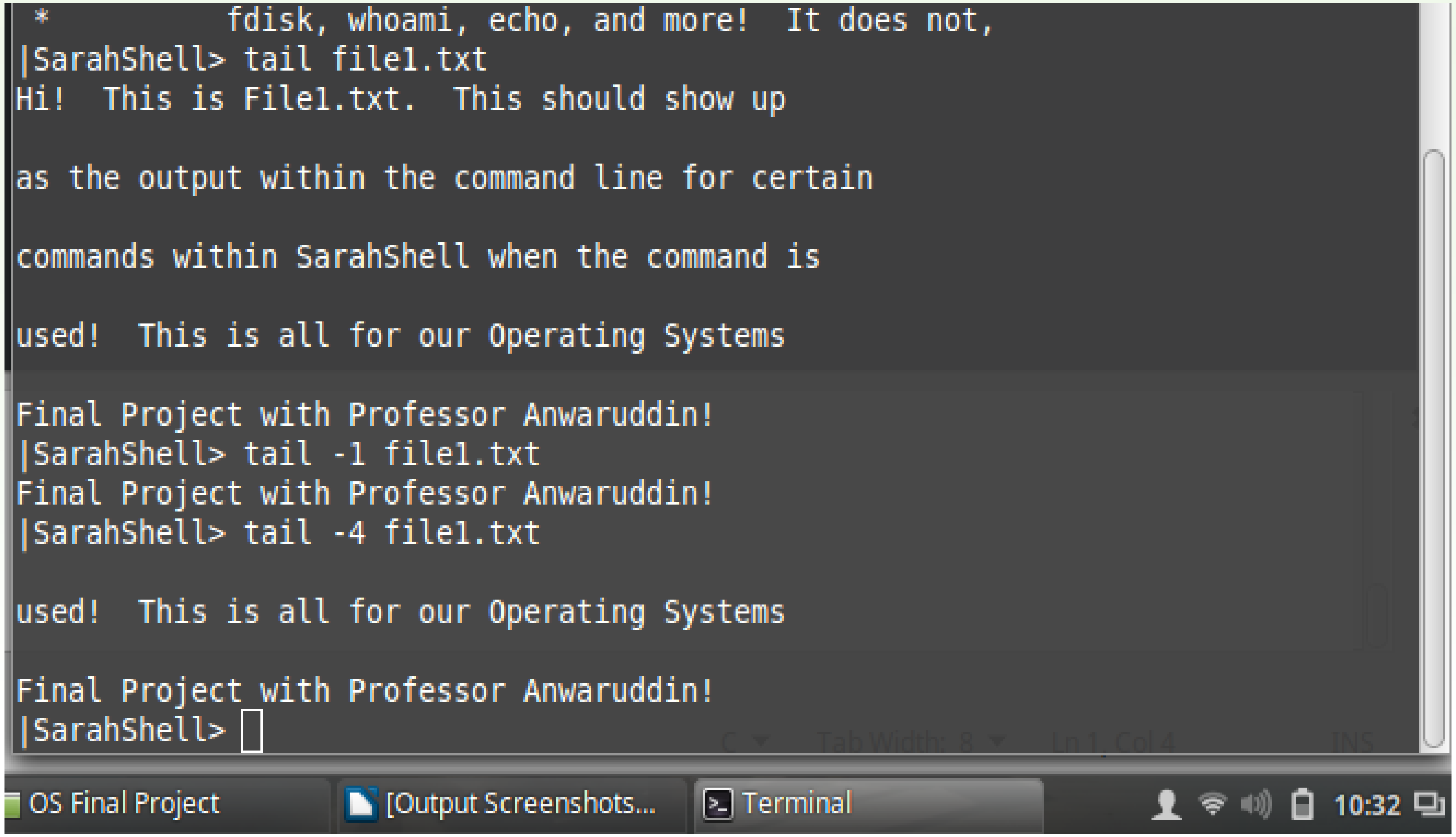
```
-rw-r--r--  1 sarah sarah      697 Nov 16  2016 write.c
SarahShell> pwd
/home/sarah
SarahShell> 
```

Command Output: head



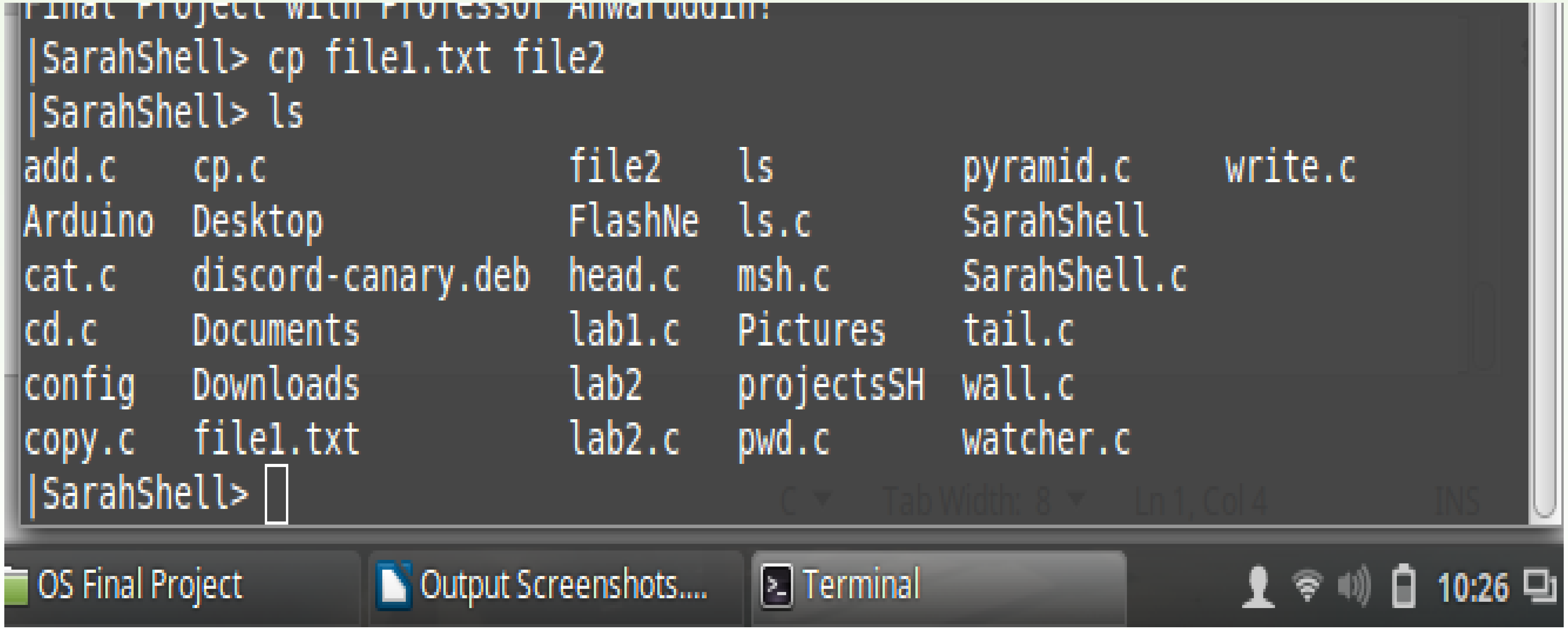
```
|SarahShell> head -2 file1.txt
Hi!  This is File1.txt.  This should show up

|SarahShell> head -4 file1.txt
Hi!  This is File1.txt.  This should show up

as the output within the command line for certain

|SarahShell> head -3 SarahShell.c
/*
 * Sarah Higgins
 * Operating Systems Final Project
|SarahShell> head -6 SarahShell.c
/*
 * Sarah Higgins
 * Operating Systems Final Project
 * SarahShell.c is a mini shell that closes with CTRL"C".
 * Performs ls, ls -l, pwd, cat, cp, head, tail, vi, ping,
 *         fdisk, whoami, echo, and more!  It does not,
|SarahShell> 
```

Command Output: tail



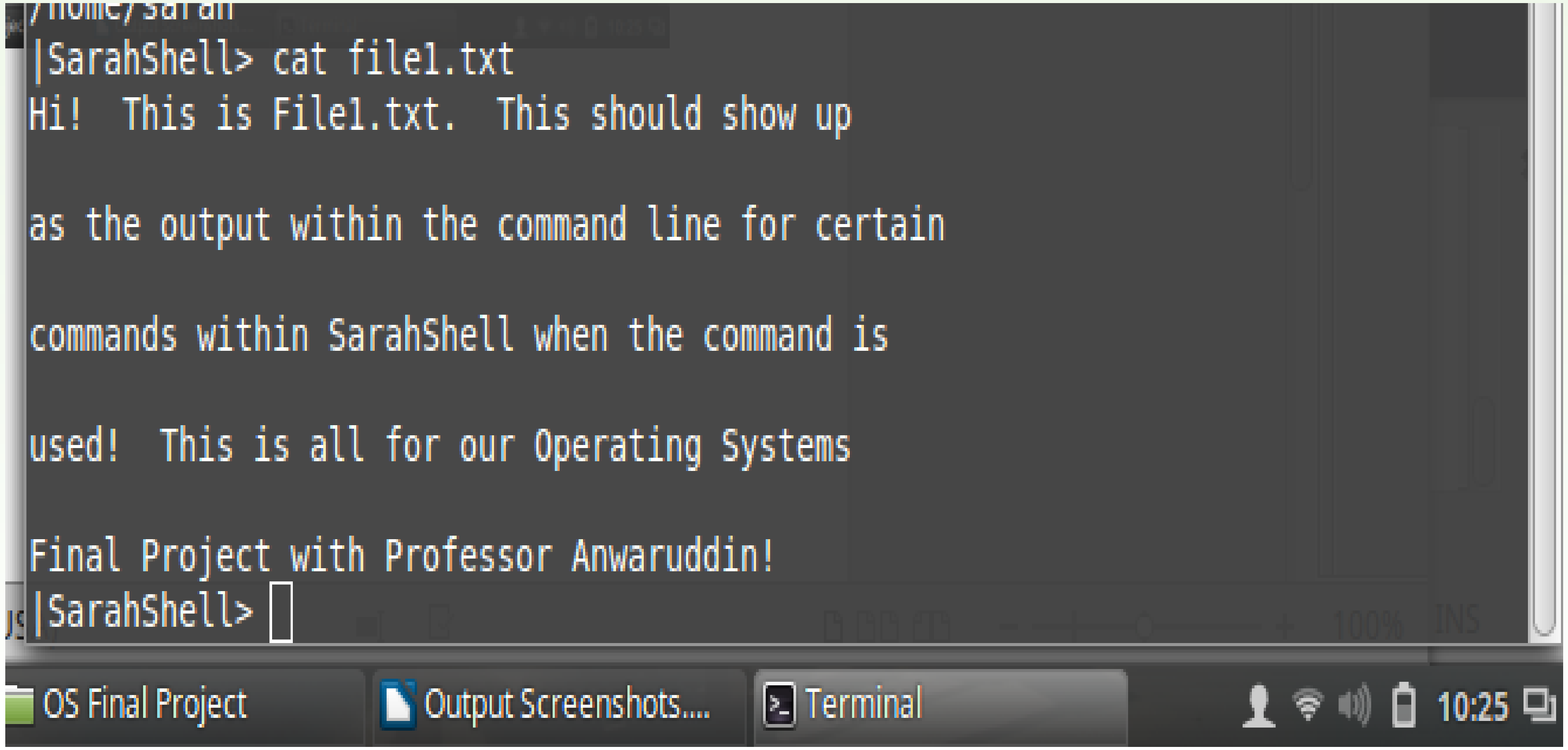
```
 *          fdisk, whoami, echo, and more!  It does not,
|SarahShell> tail file1.txt
Hi!  This is File1.txt.  This should show up

as the output within the command line for certain

commands within SarahShell when the command is

used!  This is all for our Operating Systems

Final Project with Professor Anwaruddin!
|SarahShell> tail -1 file1.txt
Final Project with Professor Anwaruddin!
|SarahShell> tail -4 file1.txt

used!  This is all for our Operating Systems

Final Project with Professor Anwaruddin!
|SarahShell> 
```

Command Output: cp



```
Final Project with Professor Anwaruddin!
|SarahShell> cp file1.txt file2
|SarahShell> ls
add.c    cp.c                file2    ls          pyramid.c     write.c
Arduino  Desktop             FlashNe  ls.c        SarahShell
cat.c    discord-canary.deb  head.c   msh.c       SarahShell.c
cd.c     Documents           lab1.c   Pictures    tail.c
config   Downloads           lab2     projectsSH  wall.c
copy.c   file1.txt           lab2.c   pwd.c       watcher.c
|SarahShell> 
```

Command Output: cat



```
|SarahShell> cat file1.txt
Hi!  This is File1.txt.  This should show up

as the output within the command line for certain

commands within SarahShell when the command is

used!  This is all for our Operating Systems

Final Project with Professor Anwaruddin!
|SarahShell> 
```

Questions