CSc 3320: Systems Programming

Spring 2021

Final/Project: Total points = 100

THIS FINAL IS OPTIONAL

Assigned: 23th Apr 2021, Friday Noon

Submission Deadline (if attempting): 2nd May 2021, Sunday, 11.59 PM

(No extensions. If your submission is not received by this time then it will NOT be accepted.)

Submission instructions:

1. Create a Google doc for your submission.
2. Start your responses from page 2 of the document and copy these instructions on page 1.
3. Fill in your name, campus ID and panther # in the fields provided. If this information is missing TWO POINTS WILL BE DEDUCTED.
4. Keep this page 1 intact. If this *submissions instructions* page is missing in your submission TWO POINTS WILL BE DEDUCTED.
5. Start your responses to each QUESTION on a new page.
6. If you are being asked to write code copy the code into a separate txt file and submit that as well. The code should be executable. E.g. if asked for a C script then provide myfile.c so that we can execute that script. In your answer to the specific question, provide the steps on how to execute your file (like a ReadMe).
7. If you are being asked to test code or run specific commands or scripts, provide the evidence of your outputs through a screenshot and/or screen video-recordings and copy the same into the document.
8. Upon completion, download a .PDF version of the google doc document and submit the same along with all the supplementary files (videos, pictures, scripts etc).

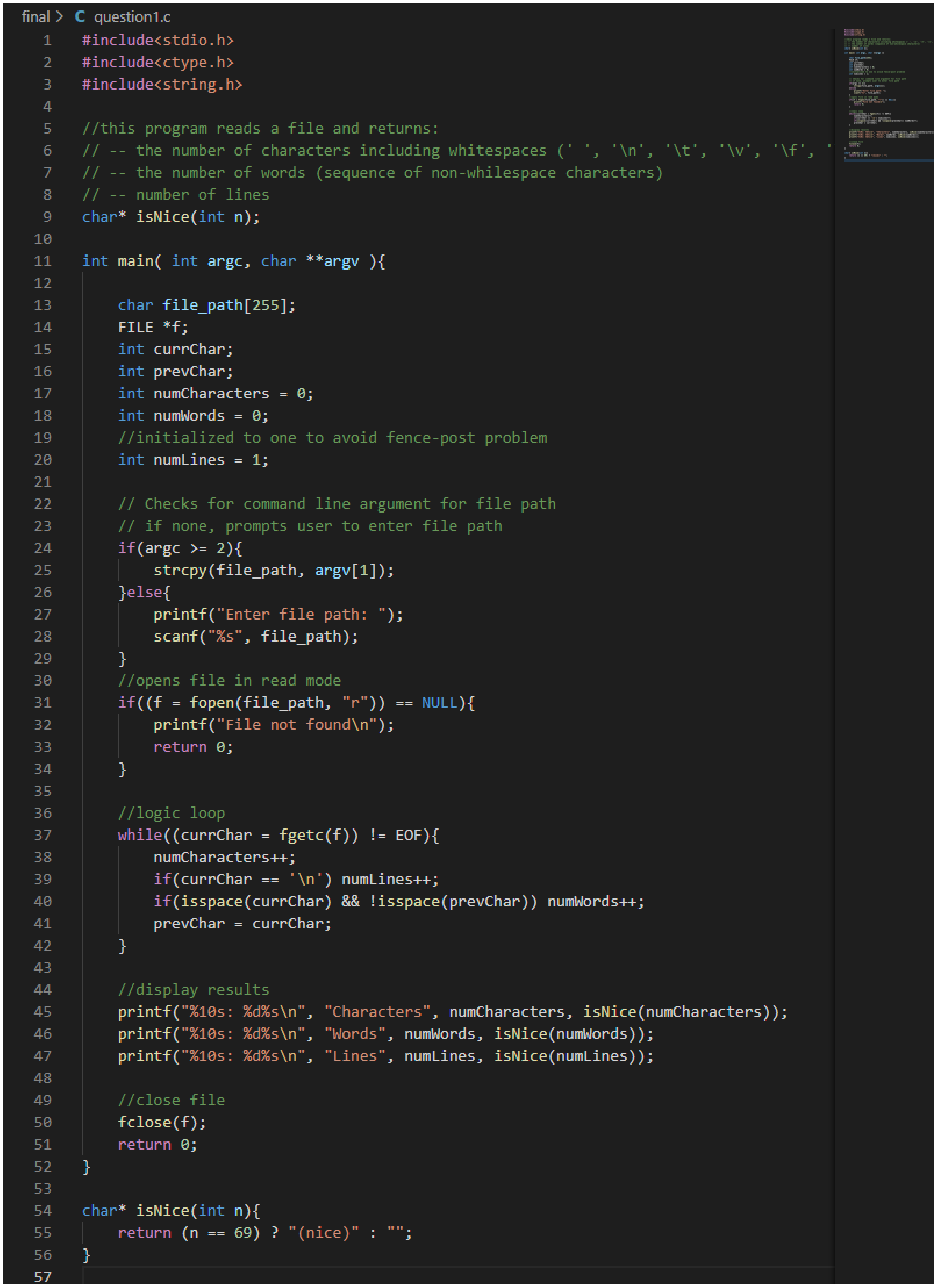
Full Name: Tracy Michaels

Campus ID: tmichaels1

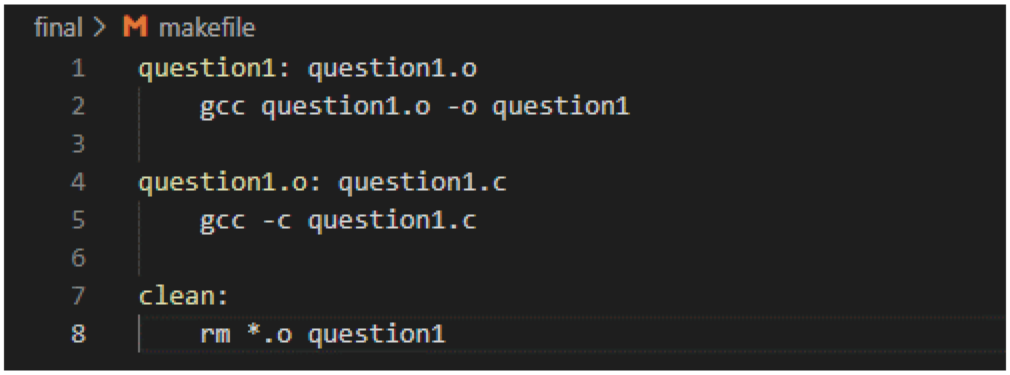
Panther #: 002430918

All programs have to be well commented. Non commented programs will receive 0 points. Comments have to be easily comprehensible and concise.

1. [30pts] Copy the contents of this document into a text file. Make sure the spacings and indentations are included.
   Write a C program that reads the text file and then outputs
   -- the number of characters (space is to be considered a character),
   -- number of words (a word is any sequence of non-white-space characters), -
   - number of lines.
   Write a makefile as well.

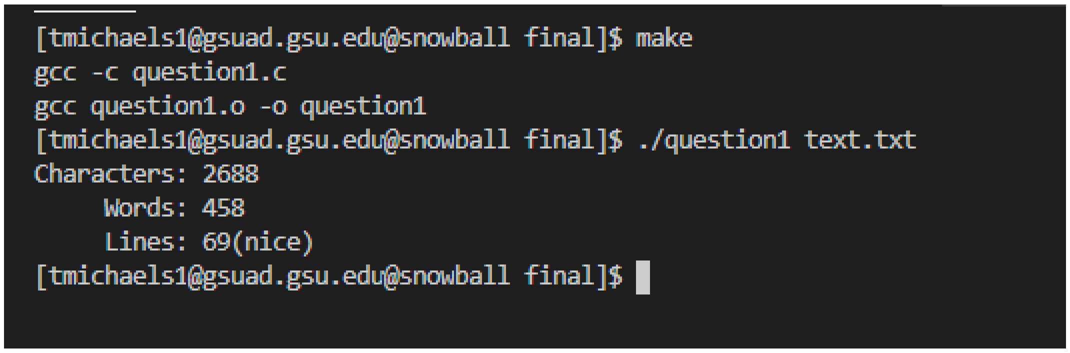

final > C question1.c

```c
#include<stdio.h>
#include<ctype.h>
#include<string.h>

//this program reads a file and returns:
// -- the number of characters including whitespaces (' ', '\n', '\t', '\v', '\f', '
// -- the number of words (sequence of non-whilespace characters)
// -- number of lines
char* isNice(int n);

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

    char file_path[255];
    FILE *f;
    int currChar;
    int prevChar;
    int numCharacters = 0;
    int numWords = 0;
    //initialized to one to avoid fence-post problem
    int numLines = 1;

    // Checks for command line argument for file path
    // if none, prompts user to enter file path
    if(argc >= 2){
        strcpy(file_path, argv[1]);
    }else{
        printf("Enter file path: ");
        scanf("%s", file_path);
    }
    //opens file in read mode
    if((f = fopen(file_path, "r")) == NULL){
        printf("File not found\n");
        return 0;
    }

    //logic loop
    while((currChar = fgetc(f)) != EOF){
        numCharacters++;
        if(currChar == '\n') numLines++;
        if(isspace(currChar) && !isspace(prevChar)) numWords++;
        prevChar = currChar;
    }

    //display results
    printf("%10s: %d%s\n", "Characters", numCharacters, isNice(numCharacters));
    printf("%10s: %d%s\n", "Words", numWords, isNice(numWords));
    printf("%10s: %d%s\n", "Lines", numLines, isNice(numLines));

    //close file
    fclose(f);
    return 0;
}

char* isNice(int n){
    return (n == 69) ? "(nice)" : "";
}

```



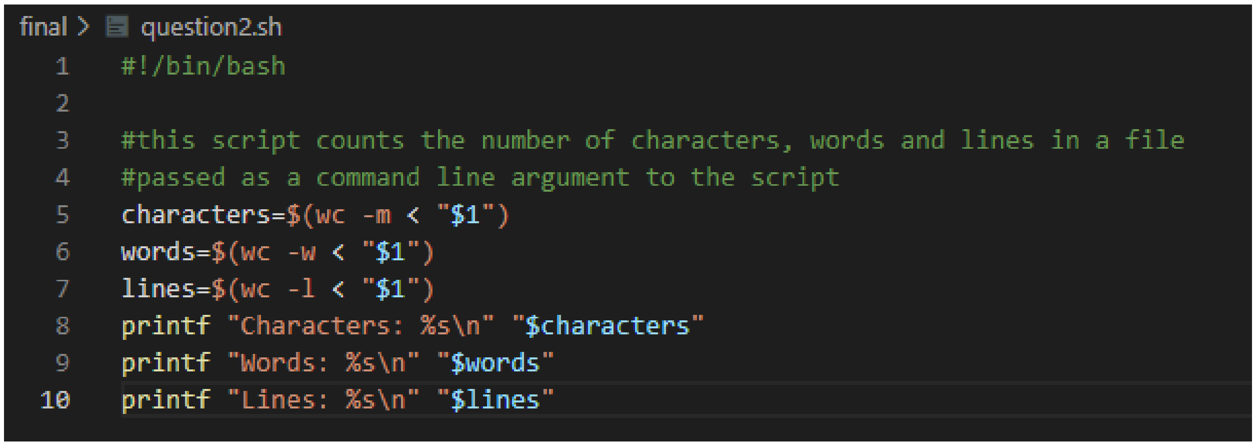
final > makefile

```
question1: question1.o
	gcc question1.o -o question1

question1.o: question1.c
	gcc -c question1.c

clean:
	rm *.o question1
```



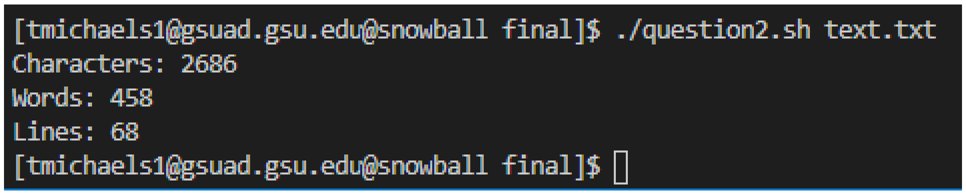
```
[tmichaels1@gsuad.gsu.edu@snowball final]$ make
gcc -c question1.c
gcc question1.o -o question1
[tmichaels1@gsuad.gsu.edu@snowball final]$ ./question1 text.txt
Characters: 2688
     Words: 458
     Lines: 69(nice)
[tmichaels1@gsuad.gsu.edu@snowball final]$
```

2. Repeat question 1, but write a shell script instead of C. Makefile not necessary. [30pts]



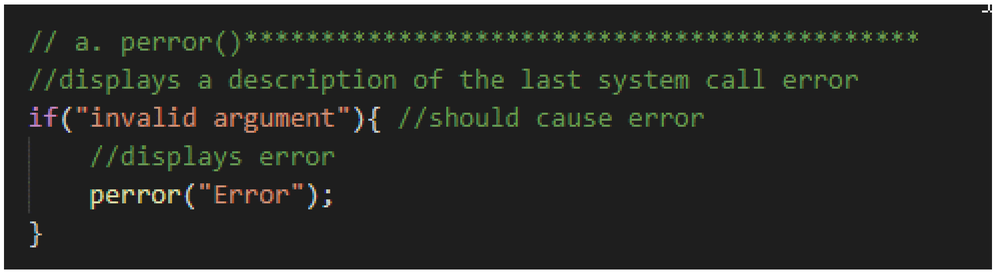
final > question2.sh

```bash
#!/bin/bash

#this script counts the number of characters, words and lines in a file
#passed as a command line argument to the script
characters=$(wc -m < "$1")
words=$(wc -w < "$1")
lines=$(wc -l < "$1")
printf "Characters: %s\n" "$characters"
printf "Words: %s\n" "$words"
printf "Lines: %s\n" "$lines"
```

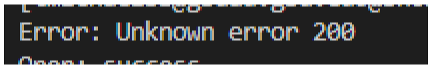


```
[tmichaels1@gsuad.gsu.edu@snowball final]$ ./question2.sh text.txt
Characters: 2686
Words: 458
Lines: 68
[tmichaels1@gsuad.gsu.edu@snowball final]$ 
```

3. [40pts] Describe (briefly in 1-2 sentences) the following unix utility functions and provide 1 example of it's usage. You can refer to Chapter 13 in the Unix textbook. You must NOT provide the same example from the textbook:
   a. perror()
      - displays a description of the last system call error

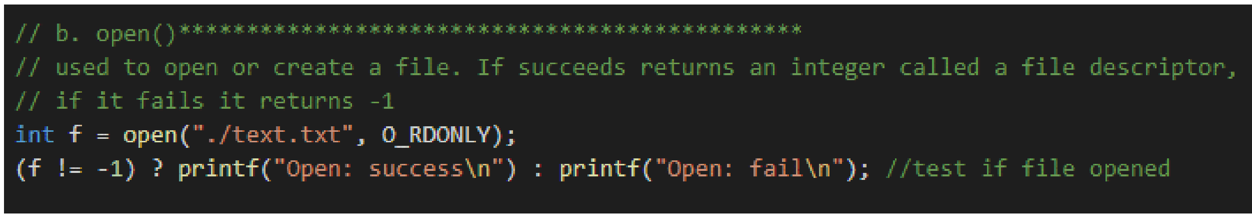


```
// a. perror()*********************************************
//displays a description of the last system call error
if("invalid argument"){ //should cause error
    //displays error
    perror("Error");
}
```

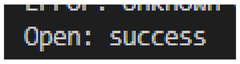


```
Error: Unknown error 200
```

   b. open()

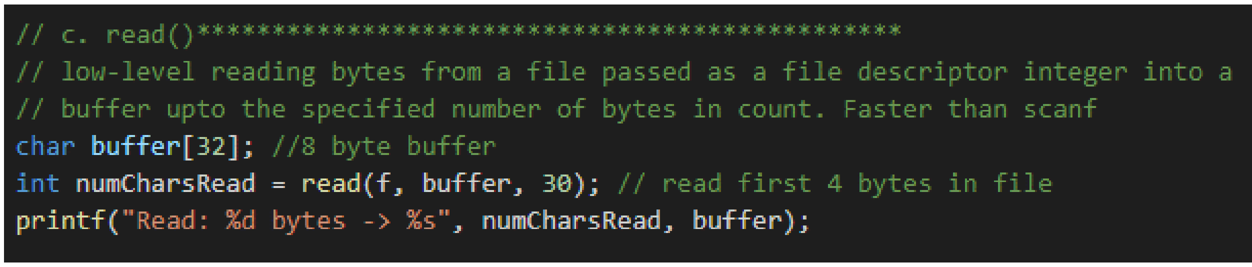


```
// b. open()***********************************************
// used to open or create a file. If succeeds returns an integer called a file descriptor,
// if it fails it returns -1
int f = open("./text.txt", O_RDONLY);
(f != -1) ? printf("Open: success\n") : printf("Open: fail\n"); //test if file opened
```

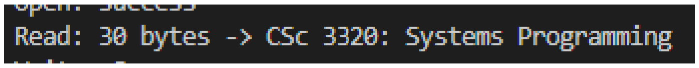


```
Open: success
```

   c. read()

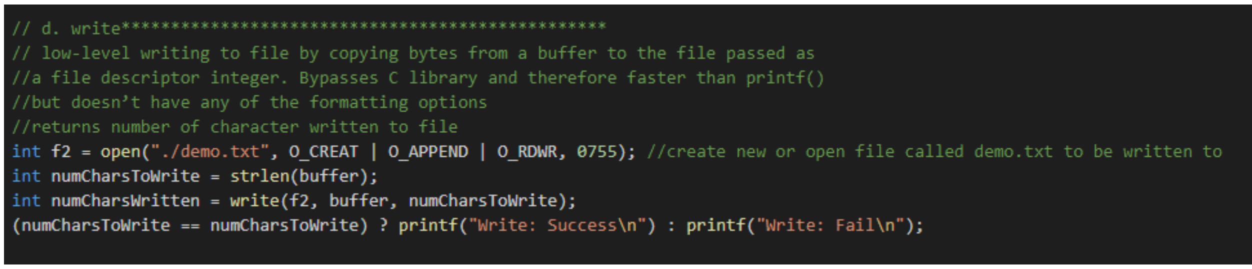


```
// c. read()***********************************************
// low-level reading bytes from a file passed as a file descriptor integer into a
// buffer upto the specified number of bytes in count. Faster than scanf
char buffer[32]; //8 byte buffer
int numCharsRead = read(f, buffer, 30); // read first 4 bytes in file
printf("Read: %d bytes -> %s", numCharsRead, buffer);
```

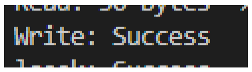


```
Read: 30 bytes -> CSc 3320: Systems Programming
```

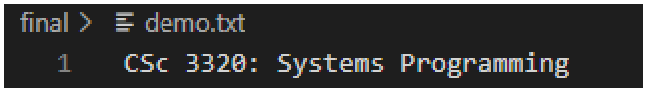
d. write()



```
// d. write**************************************************
// low-level writing to file by copying bytes from a buffer to the file passed as
//a file descriptor integer. Bypasses C library and therefore faster than printf()
//but doesn't have any of the formatting options
//returns number of character written to file
int f2 = open("./demo.txt", O_CREAT | O_APPEND | O_RDWR, 0755); //create new or open file called demo.txt to be written to
int numCharsToWrite = strlen(buffer);
int numCharsWritten = write(f2, buffer, numCharsToWrite);
(numCharsToWrite == numCharsToWrite) ? printf("Write: Success\n") : printf("Write: Fail\n");
```

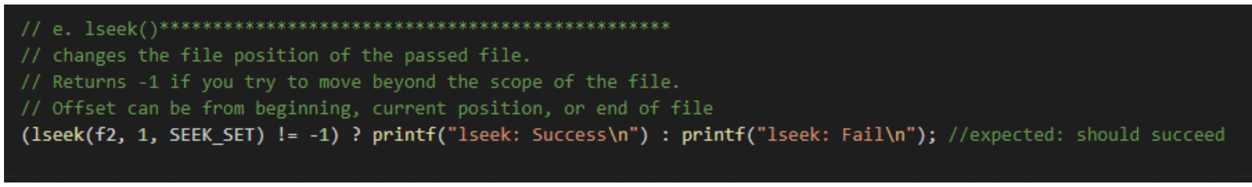


```
Write: Success
```

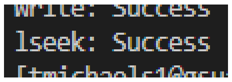


```
final > demo.txt
CSc 3320: Systems Programming
```

e. lseek()



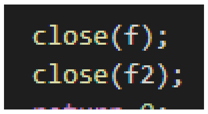
```
// e. lseek()**************************************************
// changes the file position of the passed file.
// Returns -1 if you try to move beyond the scope of the file.
// Offset can be from beginning, current position, or end of file
(lseek(f2, 1, SEEK_SET) != -1) ? printf("lseek: Success\n") : printf("lseek: Fail\n"); //expected: should succeed
```



```
lseek: Success
```

f. close()

- closes the files that was opened by freeing the file descriptor

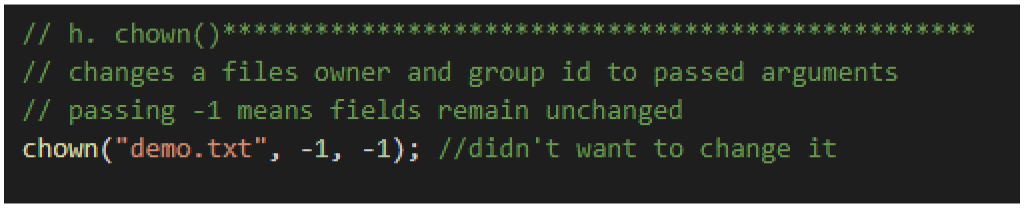


```
close(f);
close(f2);
```

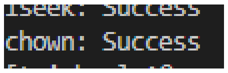
g. monitor()

- perodically scans and displays information about a file
- $ monitor text.txt

h. chown()

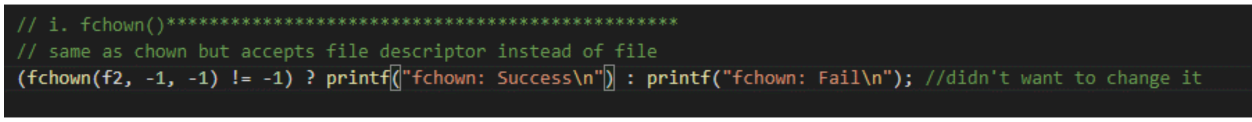


```
// h. chown()**************************************************
// changes a files owner and group id to passed arguments
// passing -1 means fields remain unchanged
chown("demo.txt", -1, -1); //didn't want to change it
```

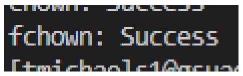


```
chown: Success
```

i. fchown()

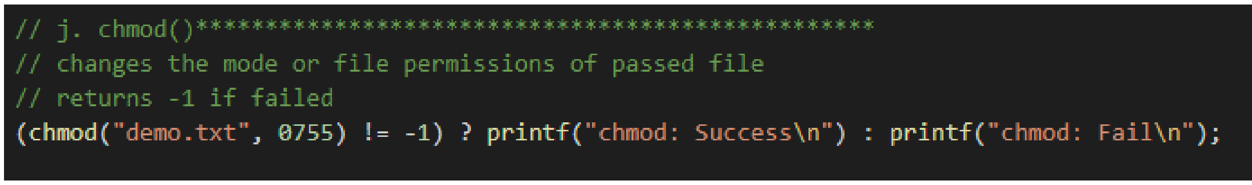


```
// i. fchown()**************************************************
// same as chown but accepts file descriptor instead of file
(fchown(f2, -1, -1) != -1) ? printf("fchown: Success\n") : printf("fchown: Fail\n"); //didn't want to change it
```

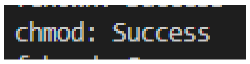


```
fchown: Success
```

j. chmod()

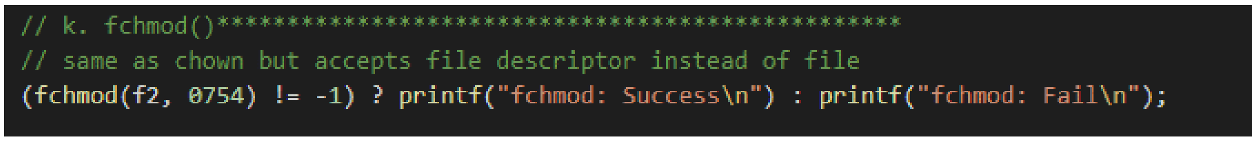


```
// j. chmod()****************************************************
// changes the mode or file permissions of passed file
// returns -1 if failed
(chmod("demo.txt", 0755) != -1) ? printf("chmod: Success\n") : printf("chmod: Fail\n");
```

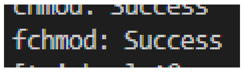


```
chmod: Success
```

k. fchmod()

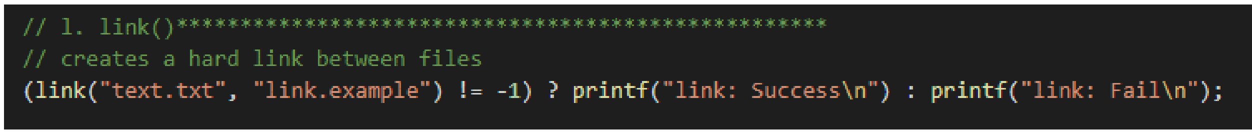


```
// k. fchmod()***************************************************
// same as chown but accepts file descriptor instead of file
(fchmod(f2, 0754) != -1) ? printf("fchmod: Success\n") : printf("fchmod: Fail\n");
```



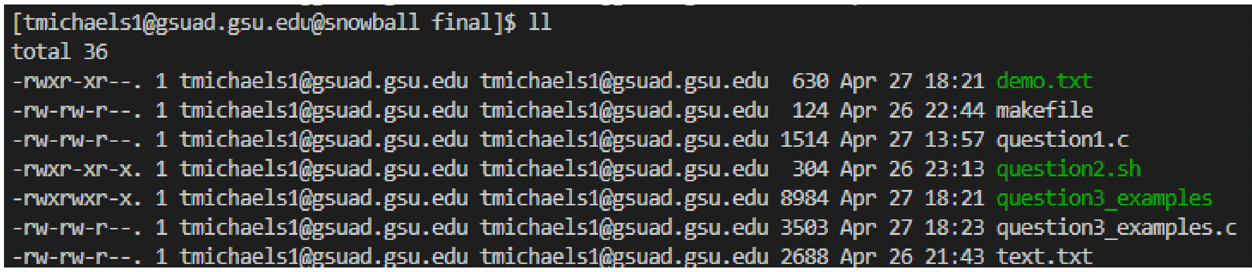
```
fchmod: Success
```

I. link()



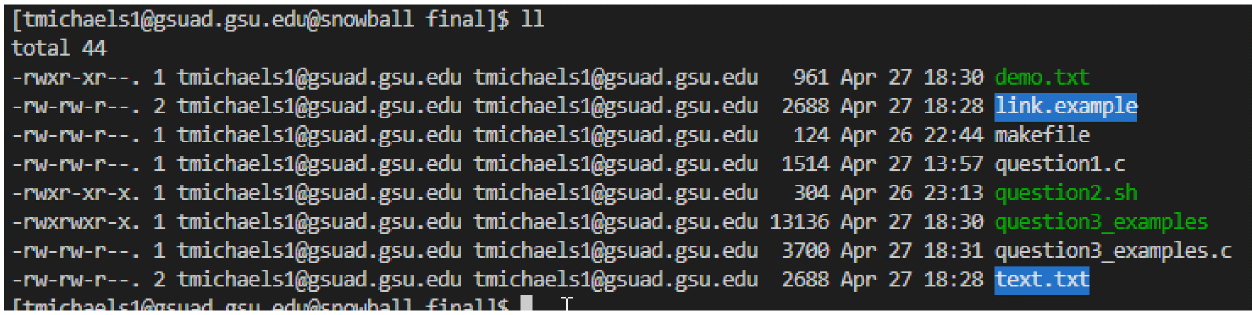
```
// 1. link()*******************************************************
// creates a hard link between files
(link("text.txt", "link.example") != -1) ? printf("link: Success\n") : printf("link: Fail\n");
```

Before:

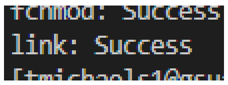


```
[tmichaels1@gsuad.gsu.edu@snowball final]$ ll
total 36
-rwxr-xr--. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu  630 Apr 27 18:21 demo.txt
-rw-rw-r--. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu  124 Apr 26 22:44 makefile
-rw-rw-r--. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu 1514 Apr 27 13:57 question1.c
-rwxr-xr-x. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu  304 Apr 26 23:13 question2.sh
-rwxrwxr-x. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu 8984 Apr 27 18:21 question3_examples
-rw-rw-r--. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu 3503 Apr 27 18:23 question3_examples.c
-rw-rw-r--. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu 2688 Apr 26 21:43 text.txt
```

After:

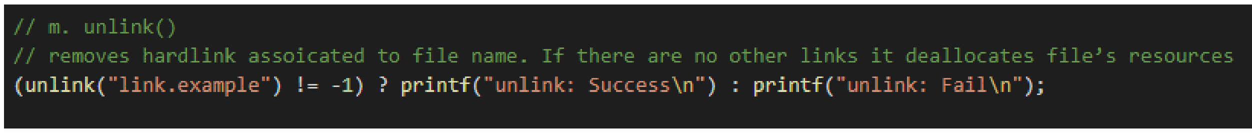


```
[tmichaels1@gsuad.gsu.edu@snowball final]$ ll
total 44
-rwxr-xr--. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu   961 Apr 27 18:30 demo.txt
-rw-rw-r--. 2 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu  2688 Apr 27 18:28 link.example
-rw-rw-r--. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu   124 Apr 26 22:44 makefile
-rw-rw-r--. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu  1514 Apr 27 13:57 question1.c
-rwxr-xr-x. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu   304 Apr 26 23:13 question2.sh
-rwxrwxr-x. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu 13136 Apr 27 18:30 question3_examples
-rw-rw-r--. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu  3700 Apr 27 18:31 question3_examples.c
-rw-rw-r--. 2 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu  2688 Apr 27 18:28 text.txt
[tmichaels1@gsuad.gsu.edu@snowball final]$
```



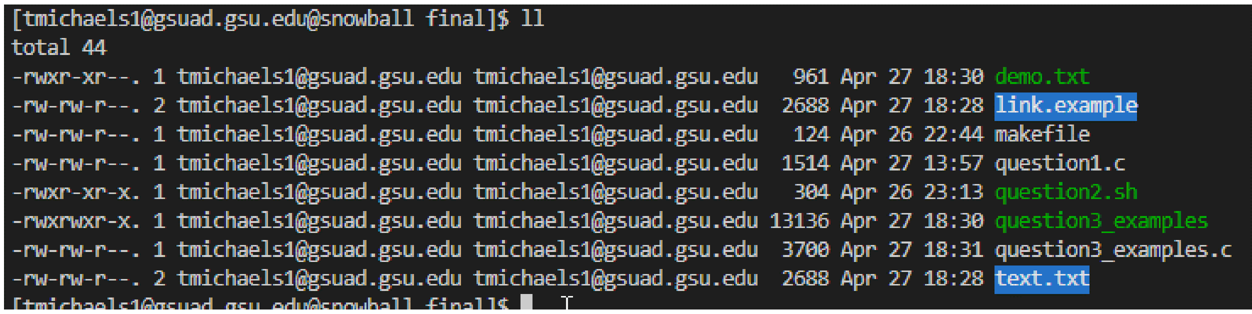
```
fchmod: Success
link: Success
[tmichaels1@gsu
```

m. unlink()



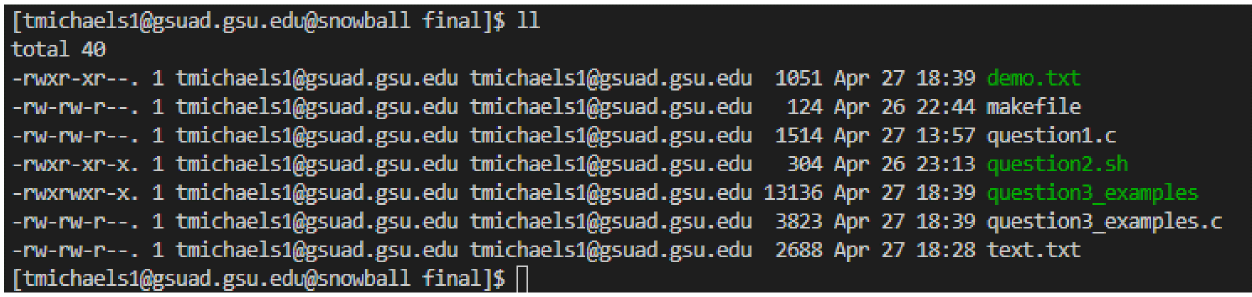
```
// m. unlink()
// removes hardlink assoicated to file name. If there are no other links it deallocates file's resources
(unlink("link.example") != -1) ? printf("unlink: Success\n") : printf("unlink: Fail\n");
```

before:



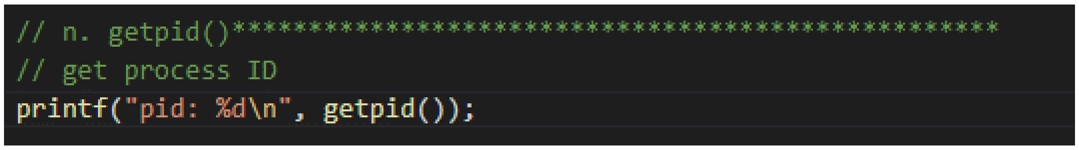
```
[tmichaels1@gsuad.gsu.edu@snowball final]$ ll
total 44
-rwxr-xr--. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu   961 Apr 27 18:30 demo.txt
-rw-rw-r--. 2 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu  2688 Apr 27 18:28 link.example
-rw-rw-r--. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu   124 Apr 26 22:44 makefile
-rw-rw-r--. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu  1514 Apr 27 13:57 question1.c
-rwxr-xr-x. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu   304 Apr 26 23:13 question2.sh
-rwxrwxr-x. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu 13136 Apr 27 18:30 question3_examples
-rw-rw-r--. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu  3700 Apr 27 18:31 question3_examples.c
-rw-rw-r--. 2 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu  2688 Apr 27 18:28 text.txt
[tmichaels1@gsuad.gsu.edu@snowball final]$
```

After:

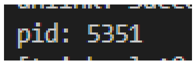


```
[tmichaels1@gsuad.gsu.edu@snowball final]$ ll
total 40
-rwxr-xr--. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu  1051 Apr 27 18:39 demo.txt
-rw-rw-r--. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu   124 Apr 26 22:44 makefile
-rw-rw-r--. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu  1514 Apr 27 13:57 question1.c
-rwxr-xr-x. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu   304 Apr 26 23:13 question2.sh
-rwxrwxr-x. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu 13136 Apr 27 18:39 question3_examples
-rw-rw-r--. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu  3823 Apr 27 18:39 question3_examples.c
-rw-rw-r--. 1 tmichaels1@gsuad.gsu.edu tmichaels1@gsuad.gsu.edu  2688 Apr 27 18:28 text.txt
[tmichaels1@gsuad.gsu.edu@snowball final]$
```

n. getpid()

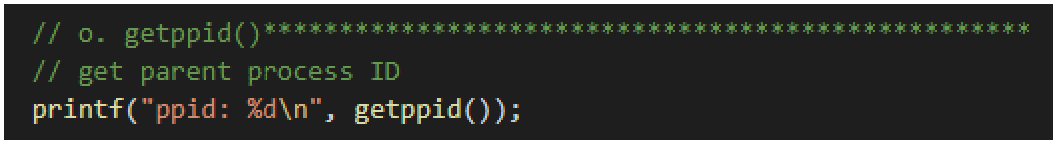


```
// n. getpid()*****************************************************
// get process ID
printf("pid: %d\n", getpid());
```

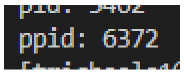


```
pid: 5351
```

o. getppid()

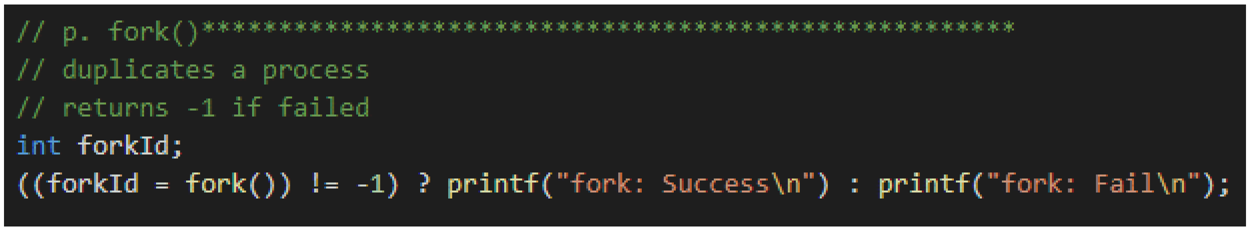


```
// o. getppid()****************************************************
// get parent process ID
printf("ppid: %d\n", getppid());
```

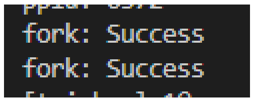


```
ppid: 6372
```

p. fork()

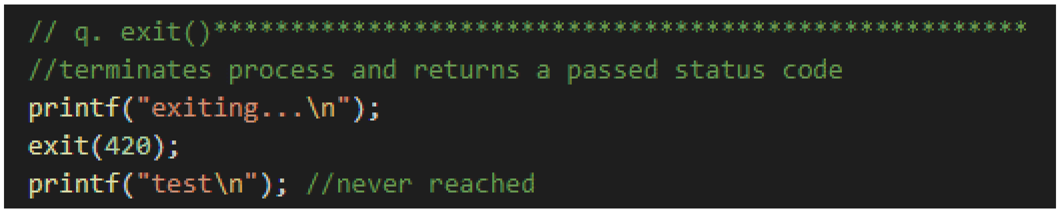


```c
// p. fork()*****************************************************
// duplicates a process
// returns -1 if failed
int forkId;
((forkId = fork()) != -1) ? printf("fork: Success\n") : printf("fork: Fail\n");
```

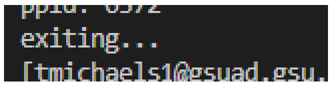


```
fork: Success
fork: Success
```

q. exit()

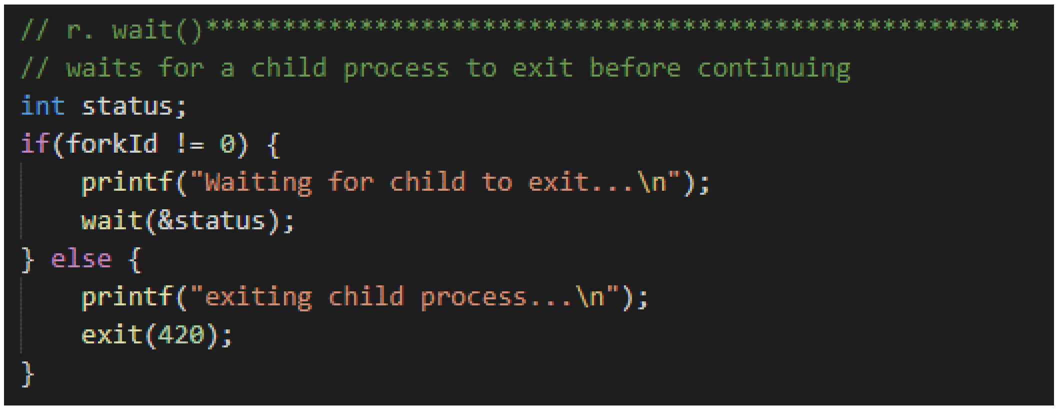


```c
// q. exit()*****************************************************
//terminates process and returns a passed status code
printf("exiting...\n");
exit(420);
printf("test\n"); //never reached
```

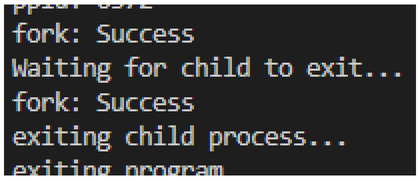


```
exiting...
[tmichaels1@gsuad.gsu.
```

r. wait()

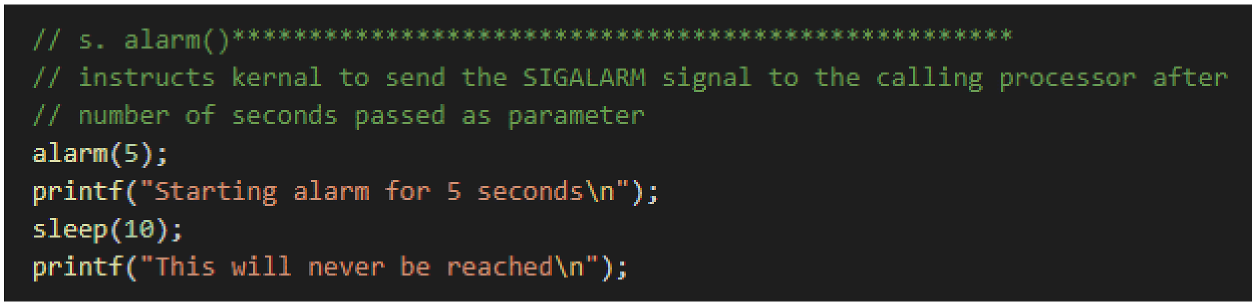


```c
// r. wait()*****************************************************
// waits for a child process to exit before continuing
int status;
if(forkId != 0) {
    printf("Waiting for child to exit...\n");
    wait(&status);
} else {
    printf("exiting child process...\n");
    exit(420);
}
```

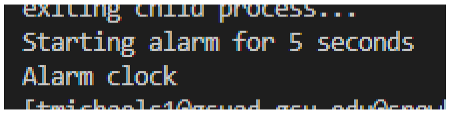


```
fork: Success
Waiting for child to exit...
fork: Success
exiting child process...
exiting program
```

s. alarm()

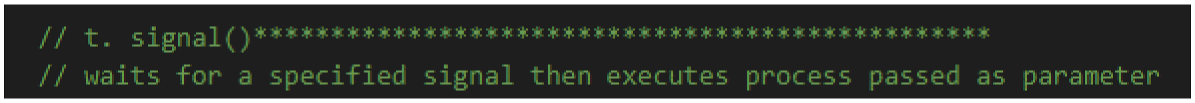


```
// s. alarm()****************************************************
// instructs kernal to send the SIGALARM signal to the calling processor after
// number of seconds passed as parameter
alarm(5);
printf("Starting alarm for 5 seconds\n");
sleep(10);
printf("This will never be reached\n");
```

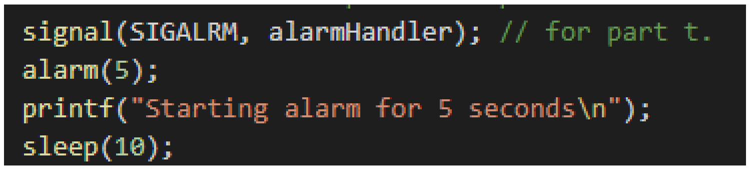


```
exiting child process...
Starting alarm for 5 seconds
Alarm clock
[tmichaels10@csuad.gsu.edu@snow
```

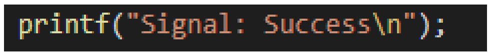
t. signal()



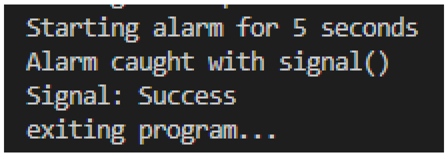
```
// t. signal()***************************************************
// waits for a specified signal then executes process passed as parameter
```



```
signal(SIGALRM, alarmHandler); // for part t.
alarm(5);
printf("Starting alarm for 5 seconds\n");
sleep(10);
```

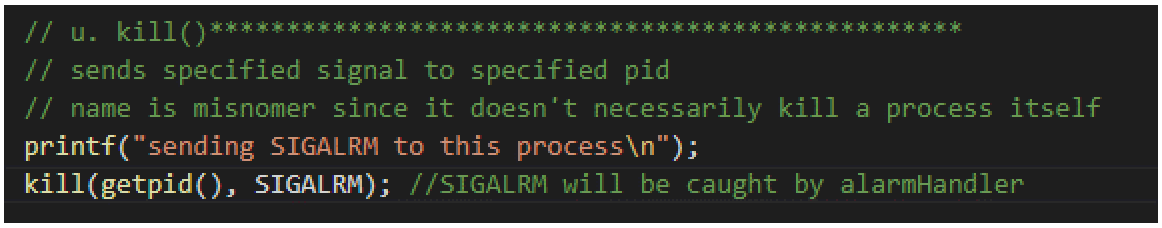


```
printf("Signal: Success\n");
```

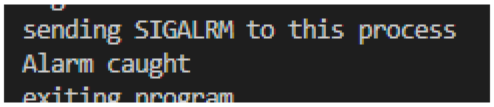


```
Starting alarm for 5 seconds
Alarm caught with signal()
Signal: Success
exiting program...
```

u. kill()

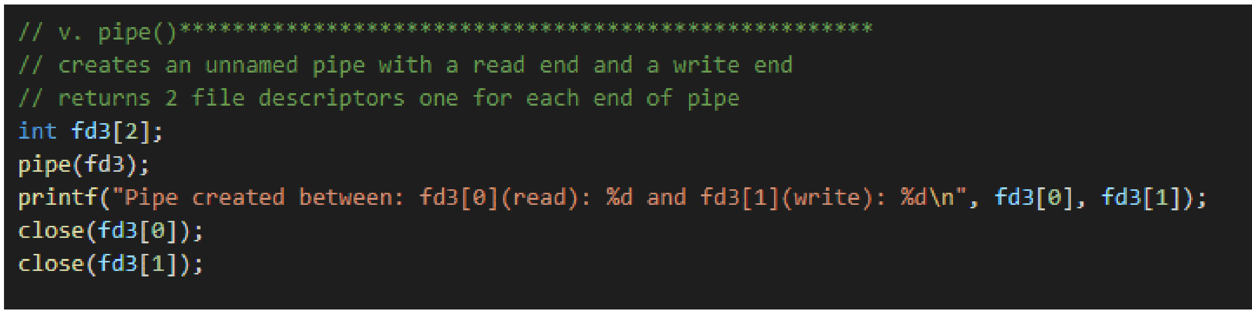


```
// u. kill()*****************************************************
// sends specified signal to specified pid
// name is misnomer since it doesn't necessarily kill a process itself
printf("sending SIGALRM to this process\n");
kill(getpid(), SIGALRM); //SIGALRM will be caught by alarmHandler
```

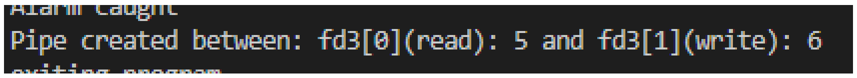


```
sending SIGALRM to this process
Alarm caught
exiting program
```

v. pipe()



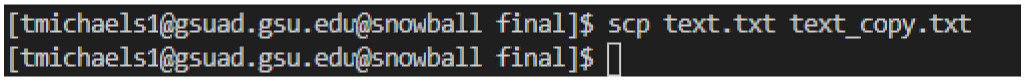
```c
// v. pipe()*****************************************************
// creates an unnamed pipe with a read end and a write end
// returns 2 file descriptors one for each end of pipe
int fd3[2];
pipe(fd3);
printf("Pipe created between: fd3[0](read): %d and fd3[1](write): %d\n", fd3[0], fd3[1]);
close(fd3[0]);
close(fd3[1]);
```



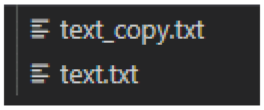
```
Alarm caught
Pipe created between: fd3[0](read): 5 and fd3[1](write): 6
exiting program
```

w. scp() (also referred to as secure copy)

- stands for secure copy protocol
- securely copies files between hosts on a network
- ex: $ scp text.txt text_copy.txt (this example copies inplace, but with the right flags can be done over network)



```
[tmichaels1@gsuad.gsu.edu@snowball final]$ scp text.txt text_copy.txt
[tmichaels1@gsuad.gsu.edu@snowball final]$ 
```



```
text_copy.txt
text.txt
```