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: Hakan Gunerli

Campus ID: hgunerli1

Panther #: 002504797

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.

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

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()
   b. open()
   c. read()
   d. write()
   e. lseek()
   f. close()
   g. monitor()
   h. chown()
   i. fchown()
   j. chmod()
   k. fchmod()
   l. link()
   m. unlink()
   n. getpid()
   o. getppid()
   p. fork()

q. exit()
r. wait()
s. alarm()
t. signal()
u. kill()
v. pipe()
w. scp() (also referred to as secure copy)

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{

    FILE *fp;
    char ch, file_name[100]; // create space for the name our file as well as our
 characters from th efile each time.
    int num_char = 0, word_count = 0, line_count = 0, in_word = 0;

    fp = fopen("test.txt", "r");

    while ((ch = fgetc(fp)) != EOF) // if for each character from our file is not
 equal to the end of file, keep running the inner code.
    {

        if (ch == ' ' || ch == '\0' || ch == '\n'|| ch == '\t')
        {
            if (in_word)
            {
                in_word = 0;  // used for counting the amount of words
                word_count++; //  iterate to count the amount of words.
            }

            if (ch = '\0' || ch == '\n')
                line_count++; // this is for calculating how many lines, if it's
a newline or an empty one.
        }
        else
        {
            in_word = 1;
        }
        num_char++; // for the number of characters.
    }

    printf("Number of characters: %d \n", num_char);
    printf("Number of words: %d \n", word_count);
    printf("Number of lines: %d \n", line_count);

    return 0;
}
```

MAKEFILE

```
# this will compile when you call make, there is only one file which is named lines.c

all : lines.o

lines.o: lines.c
    gcc lines.c -o output

clean:
    rm -rf *o output
```

PART 2:

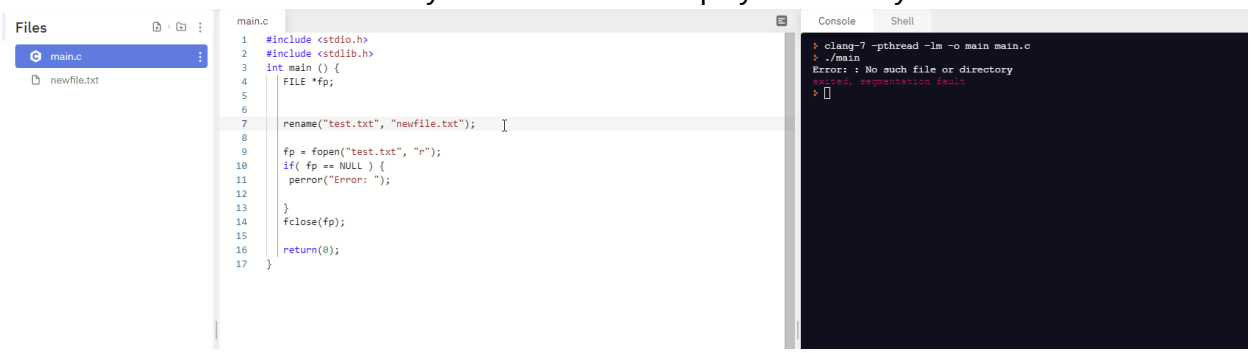
```
input="./test.txt"

echo "number of lines in my text file "
wc -l < $input # word count number of lines

echo "number of words in my text file"
wc -w < $input # word count word

echo "number of characters in my text file"
wc -c < $input # this is word count character
```

HOW TO RUN:
gcc lines.c && ./a.out
./main.sh

PART 3:

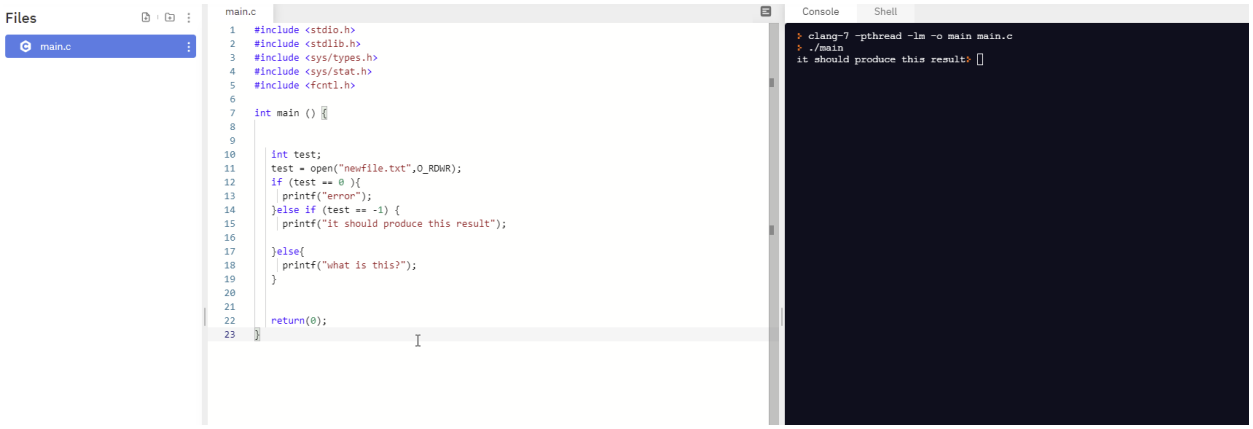
a. perror()

a subroutine that describes the system call errors. Displays the last system call error.



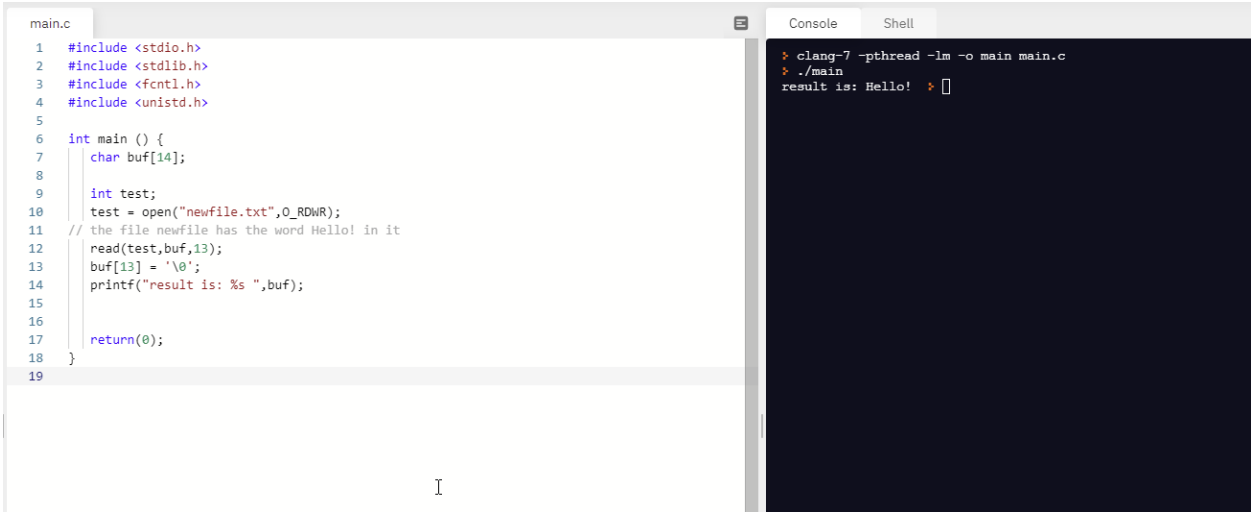
b. open()

opens or creates a file.



c. read()

reads the byte representation of a file into a buffer.



d. write()

writes byte from a buffer to the file.

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

int main () {
    char buf[14];

    int test;
    test = open("newfile.txt",O_RDWR);
// the file newfile has the word Hello! in it
    read(test,buf,13);
    buf[13] = '\0';
    printf("result is: %s ",buf);


    return(0);
}
```

Console Shell

```
> clang-7 -pthread -lm -o main main.c
> ./main
result is: Hello!  >
```

e. lseek()

moves a offset that is particular into a file

main.c

```c
#include <fcntl.h>
#include <stdio.h>
int main()
// This is a sample application that demonstrates lseek ==   testfile.txt
{
        int file;
        file = open("testfile.txt",O_RDONLY);
        if((file) < -1) // if it's not readonly, get me an error code
                return 1;

        char buffer[19];
        if(read(file,buffer,19) != 19) { // get me an error code if it does not work, print
          return -1;
        }
        printf("%s\n",buffer);

        if(lseek(file,10,SEEK_SET) < 0){ // lseek capability offset
return 1;
        }

        if(read(file,buffer,19) != 19){ // get me an error code if it does not work, print
 return -1;
        }
        printf("%s\n",buffer);

        return 0;
}
```

Console Shell

```
> clang-7 -pthread -lm -o main main.c
> ./main
This is a sample ap
sample application
>
```

f. close()

close the file.

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

int main () {
    char buf[14];

    int test;
    test = open("newfile.txt",O_RDWR);
// the file newfile has the word Hello! in it
    read(test,buf,13);
    buf[13] = '\0';
    printf("result is: %s ",buf);


    return(0);
}
```

```
clang-7 -pthread -lm -o main main.c
./main
result is: Hello!
```

g. monitor()

scans the files specified, displays information about the modification of those files.
// THIS DOES NOT EXIST WITHIN MY SHELL OR GNU C

```
hakancangunerli@DELL-CAN:/mnt/c/Users/hakan$ man monitor
No manual entry for monitor
hakancangunerli@DELL-CAN:/mnt/c/Users/hakan$ monitor
-bash: monitor: command not found
hakancangunerli@DELL-CAN:/mnt/c/Users/hakan$
```

h. chown()

changing the owner status or the group status of a file

```
hakancangunerli@DELL-CAN:/mnt/d/Users/hakan/Desktop/tet$ ls -la lec2_branch_loops.py
-rwxrwxrwx 1 hakancangunerli hakancangunerli 2890 May  2 01:30 lec2_branch_loops.py
hakancangunerli@DELL-CAN:/mnt/d/Users/hakan/Desktop/tet$ sudo chown -v :docker lec2_branch_loops.py
changed ownership of 'lec2_branch_loops.py' from hakancangunerli:hakancangunerli to :docker
```

i. fchown()

pretty much like chown, except that it needs to be the file descriptor definition as supposed to the path.

j. chmod()

changing the access permissions of a file

```c
#include <sys/stat.h>
#include <stdio.h>



int main () {
int flag;
flag = chmod ("test.txt", 0600);
}
```

```
~/final-1$ ls -la test.txt
-rw-r--r--- 1 runner runner 52 May  3 10:16 test.txt
~/final-1$ ls -la test.txt
-rw------- 1 runner runner 52 May  3 10:16 test.txt
~/final-1$
```

k. fchmod()

pretty much like chmod, except that it needs to be the file descriptor definition as supposed to the path.

```c
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <stdlib.h>
#include <stdio.h>

int main() {
  char file_name[]="test.txt";
  FILE *file;

  struct stat info;
int fd = creat(file_name, S_IWUSR);

  if ((fd) < 0)
    perror("creat() error");
  else {
    stat(file_name, &info);
    printf("original owner was %d and group was %d\n", info.st_uid,info.st_gid);
    if (fchown(fd, 100, 0) != 0){
printf("error");
      exit(-1);

    }
      else {
      stat(file_name, &info);
      printf("after fchown(), owner is %d and group is %d\n", info.st_uid, info.st_gid);
    }
```

```
hakancangunerli@DELL-CAN:/mnt/d/Users/hakan/Desktop/tet$ gcc main.c && ./a.out
original owner was 1000 and group was 1000
errorhakancangunerli@DELL-CAN:/mnt/d/Users/hakan/Desktop/tet$ sudo su
[sudo] password for hakancangunerli:
root@DELL-CAN:/mnt/d/Users/hakan/Desktop/tet# gcc main.c && ./a.out
original owner was 1000 and group was 1000
after fchown(), owner is 1000 and group is 1000
root@DELL-CAN:/mnt/d/Users/hakan/Desktop/tet#
```

l. link()

creating a link between files.

```c
#include <unistd.h>
#include <stdio.h>
int main(){

int val;
val =link ("test.txt", "textexample.txt");
if (val == 0){
  printf("when link works, it returns 0\n");
}
int unval;
unval = unlink("textexample.txt");

if (unval == 0){

  printf("otherwise, the code is correct!");
}else{
printf("if this is right, i probably messed up.");
}


}
```

```
> clang-7 -pthread -lm -o main main.c
> ./main
when link works, it returns 0
otherwise, the code is correct!>
```

m. unlink()

killing the link between files.

main.c

```c
#include <unistd.h>
#include <stdio.h>
int main(){

int val;
val =link ("test.txt", "textexample.txt");
if (val == 0){
  printf("when link works, it returns 0\n");
}
int unval;
unval = unlink("textexample.txt");

if (unval == 0){

  printf("otherwise, the code is correct!");
}else{
printf("if this is right, i probably messed up.");
}


}
```

Console Shell

```
> clang-7 -pthread -lm -o main main.c
> ./main
when link works, it returns 0
otherwise, the code is correct!>
```

n. getpid()

get the id of a process. This is for calling function.

main.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>


int main() {
    pid_t c_pid = fork(); // create new process.
    if (c_pid == 0) {
        printf("printed from child process %d", getpid());
    } else {
        printf("printed from parent process %d\n", getpid());
        wait(NULL);
    }


}
```

Console Shell

```
> clang-7 -pthread -lm -o main main.c
> ./main
printed from parent process 232
printed from child process 233>
```

o. getppid()

return the process id of the parent.

main.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>


int main() {
    pid_t c_pid = fork(); // create
    if (c_pid == 0) {
        printf("printed from child process %d", getppid());
    } else {
        printf("printed from parent process %d\n", getppid());
        wait(NULL);
    }


}
```

Console Shell

```
> clang-7 -pthread -lm -o main main.c
> ./main
printed from parent process 1
printed from child process 775>
```

p. fork()

calls a process child. Duplicates the calling process

main.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>


int main() {
    pid_t c_pid = fork(); // create new process.
    if (c_pid == 0) {
        printf("printed from child process %d", getpid());
    } else {
        printf("printed from parent process %d\n", getpid());
        wait(NULL);
    }


}
```

Console Shell

```
> clang-7 -pthread -lm -o main main.c
> ./main
printed from parent process 232
printed from child process 233>
```

p. exit()

exit. Terminate the process.

main.c

```c
#include <stdio.h>
#include <stdlib.h>

int main () {


  printf("Exiting  program....\n");
  exit(0);

  printf("error program....\n");
exit(-1);
  return(0);
}
```

Console Shell

```
> clang-7 -pthread -lm -o main main.c
> ./main
Exiting  program....
>
```

q. wait()

wait for a process to change.

main.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>


int main() {
    pid_t c_pid = fork(); // create new process.
    if (c_pid == 0) {
        printf("printed from child process %d", getpid());
    } else {
        printf("printed from parent process %d\n", getpid());
        wait(NULL);
    }


}
```

Console Shell

```
> clang-7 -pthread -lm -o main main.c
> ./main
printed from parent process 232
printed from child process 233>
```

r. alarm()

set an alarm clock for a signal.

main.c

```c
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

int breakflag = 3;
void handle(int);

int main() {
    signal(SIGALRM, handle);
    alarm(1);
    while(breakflag) {
      sleep(1);
      }
    printf("done\n");
    return 0;
}

void handle(int sig) {
    printf("Test\n");
    breakflag--;
    alarm(1);
}
```

Console Shell

```
> clang-7 -pthread -lm -o main main.c
> ./main
Test
Test
Test
done
>
```

s. signal()

signal handler.

Welcome test.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
void signalhandle(int);
int main()
{
    signal(SIGINT, signalhandle);
    while (1)
    {
        printf("testing\n");
        sleep(1);
    }
    return 0;
}
void signalhandle(int sig)
{
    printf("Caught signal %d\n", sig);
}
```

PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL

```
  11 |         sleep(1);
     |         ^~~~~~
testing
testing
testing
^CCaught signal 2
testing
testing
testing
testing
testing
testing
testing
testing
^CCaught signal 2
testing
^CCaught signal 2
testing
^CCaught signal 2
testing
```

t. kill()

killing a process.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

int main(void){
    pid_t returnval;
    returnval = fork();

    if(returnval > 0){
        int i = 0;
        while(i++ < 3){ // do this process about 3 times each
            printf("parent process.\n");
            sleep(1);
        }

        kill(returnval, SIGKILL); //kill the child process

    } else if (returnval == 0){
        int i = 0;

        do{
        printf("child process.\n");
            sleep(1);
        }
        while(i++ < 10); // it'll never reach this number since the parent process will
        kill it



    }

    return 0;

}
```

```
> clang-7 -pthread -lm -o main main.c
> ./main
parent process.
child process.
parent process.
child process.
parent process.
child process.
>
```

u. pipe()

connection between two processes.

```c

#include <stdio.h>
#include <unistd.h>


int main()
{
    int n;
    int filedes[0];
    char buffer[100];
    char *message = "testing value";
  int i;
    for (i = 0; message[i] != '\0'; ++i);
    pipe(filedes);
    write(filedes[1], message, i);
    char test= read ( filedes[0], buffer, 100 );

    if ((n = test  ) >= 0) { // comparison
        buffer[n] = 0;  //terminate the string
        printf(" pipe received : %s", buffer); // show pipe result
    }
return 0;
}
```

```
> clang-7 -pthread -lm -o main main.c
> ./main
 pipe received : testing value>
```

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

openSSH file copy. Copy between localhost and remote host.
I'm not able to implement this due to host issues.