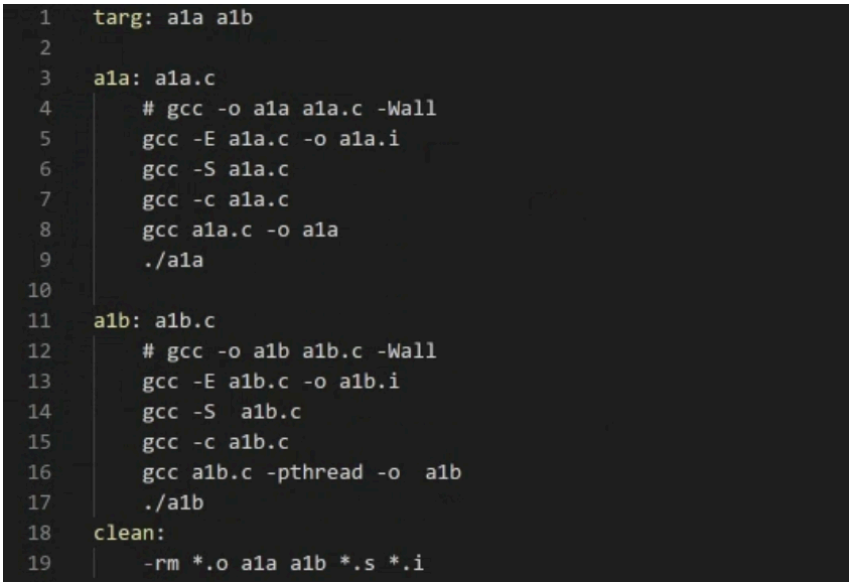
CSE 231: Operating Systems
Assignment 1, Question 1

Name: Abhik S Basu
Roll No: 2020165
Section: A



```
targ: a1a a1b

a1a: a1a.c
    # gcc -o a1a a1a.c -Wall
    gcc -E a1a.c -o a1a.i
    gcc -S a1a.c
    gcc -c a1a.c
    gcc a1a.c -o a1a
    ./a1a

a1b: a1b.c
    # gcc -o a1b a1b.c -Wall
    gcc -E a1b.c -o a1b.i
    gcc -S  a1b.c
    gcc -c a1b.c
    gcc a1b.c -pthread -o  a1b
    ./a1b
clean:
    -rm *.o a1a a1b *.s *.i
```

To run the program, run the following make command in the terminal -

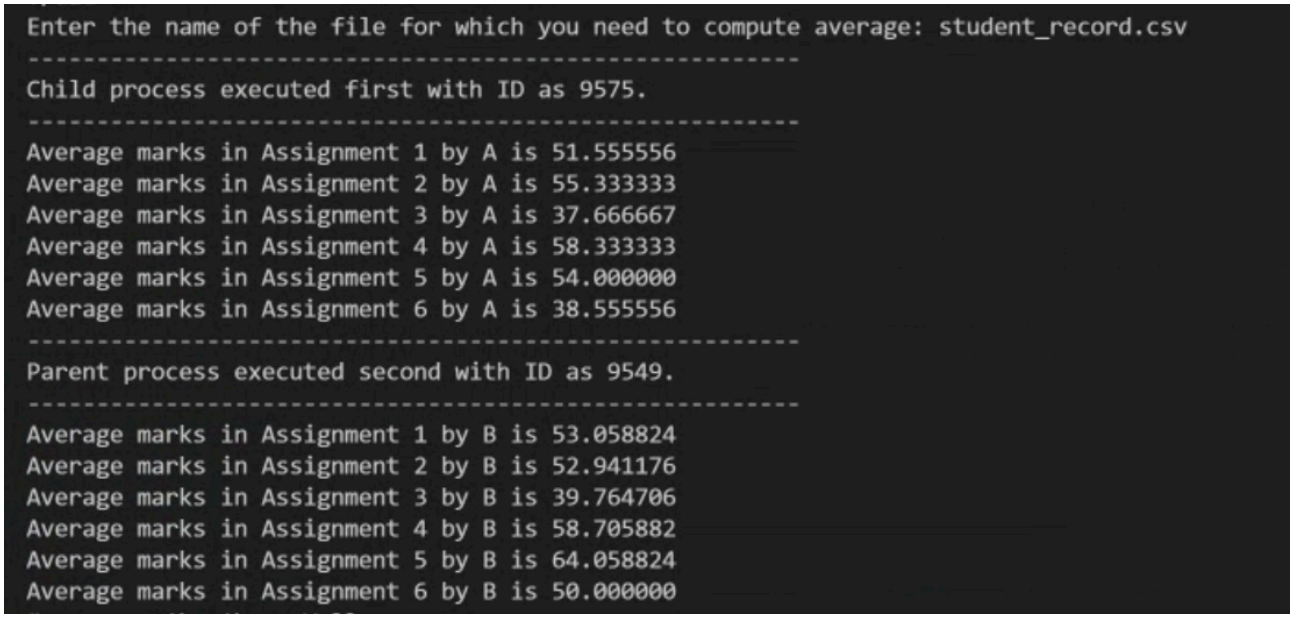
```
make
```

To clear out the files generated and reset the programme for a new run, run the following command in the terminal -

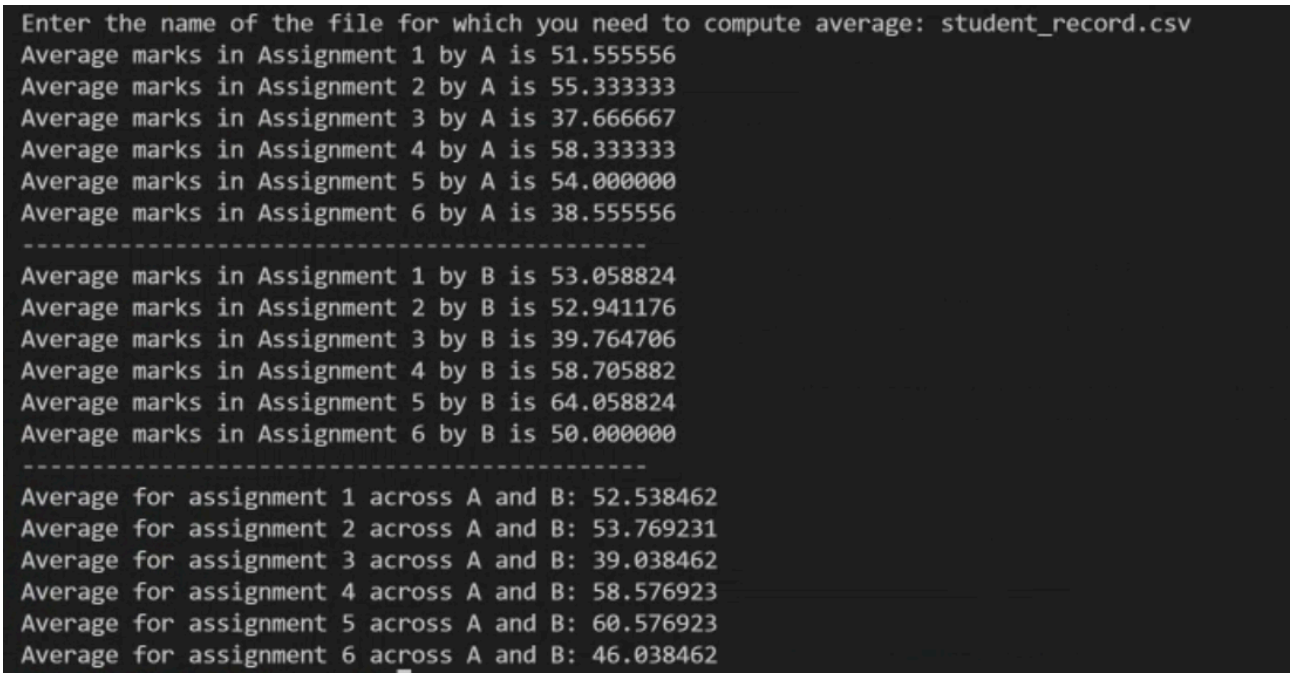
```
make clean
```

On running make, we will find the output of both the codes in the following flow -

1. We will first be prompted to enter the name of the CSV file, which in the question given to us is student_record.csv. However it may vary during test cases being run so we need to make sure the .csv file is located in the same folder as the codes.
2. We will then find the output of the first part of question 1 as follows:
3. We then repeat the step 1. for the second part of Q1 and receive the output as follows:



Task 1 using processes



Task 2 using threads

How the programme works (Processes) -

As soon as the fork() system call is encountered, a new child process is created. The process ID 0 is alloted to the child process and then an if case if run where if it encounters a zero the process is run for the child process.

Inside the function computeAverages(), the file is fist opened and it is accessed only via the read() command. We basically maintain a 2D array in order to store for the data for each row and the

columns accordingly. A while loop is run over read and whenever a “\n” is encountered we terminate the parameters of the 2d arrays and start traversing the next row.

Now after the reading part is done, we use a string tokeniser function with , as the separator and assign it to variables and sum it accordingly. A counter also gets incremented which maintains the count of the total no. of students in one particular section.

Once we are done traversing we compute and print the averages and the data type used is double for the same. We then proceed to close the file. And end the process with exit(0).

Now the parent process waits for the child process to terminate with the help of the wait() function. Once the child process terminates we begin executing the parent process. It repeats what was done for section A but does it with Section B this time. The comparisons for section is carried out using the strcmp() function.

System Calls used and Error Handling

1) fork() – creates a child process which runs along side the process that makes for its call, which is known as the parent process. It takes no parameters and returns an integer, which gives the status of the process generation –
   - Negative Value – operation has been unsuccessful
   - Zero – given to the child process
   - Positive Value – process id of child process

   Error Handling –
   the program checks whether the pid_id generated is negative, in which case it prints and error message and exits from the process.

2) wait() – used to halt the parent process until the child process exits successfully. After the termination of the child process successfully, the parent is executed.. It returns the value -1 if there are no child processes.
   Error handling –
   The parent process only runs after the child process has successfully exited, thus taking care of anomalous behaviour.

3) exit() – this sys call terminates the process which it is running, disregarding the code that follows the command call.

It is used to exit from the child process and signal the parent's wait command that it can resume its process.
Error handling –
This command is used to depict errors so no error handling for this as such.

4) open() – This system call is used to open a file for reading, writing or doing both at the same time. The mode can be specified using one of the flag specifiers which is O_RDONLY in this case. The path is also passed for the file to be opened, on which operations will be performed afterwards. It returns an integer value which is used for error handling.
Error handling -
The open command returns a -1 value if it runs unsuccessfully due to any reason. This has been handled in the code.

5) read() – This system call is used to read given amount of bytes from the opened file into an array defined by user. It takes in the file descriptor value, array to read into and length of array to read as parameters.
Error handling –
A white loop has been used which terminated as soon as the file reading is done. No other error handling done since in this case, the file name has to be correct and error free to reach till this segment of the code.

6) close() – This system call is used to tell an OS that we have used a particular file descriptor and hence close the file that we had opened using open(). It takes in the file descriptor as its only parameter, and returns 0 on success and -1 on failure.
Error Handling –
In case of a -1 return value observed, an error is observed.

How the programme works (Threads) -

First we initialise two processes using the pthread_t. Following that we create two processes which run parallel. The processes are created using the pthread_create(). They are then used to wait for the other thread to finish executing by using the pthread_join() sys call which will wait till the thread which started executing first does not finish its termination. The function is more or less the same as in the previous part of this question however

there are a few global variables used and the overall average is computed in the main() class once all data is available. The threads are terminated by using the pthread_exit().

Additional Functions used and Error Handling

1) pthread_create() - this functions is used to start a new thread in the calling process. It starts executing by immediately going to the area where its start routine pointer is located in the function. This is the 3rd parameter.The fourth parameter is the arguments which are passed which in our case is the name of the file.The first parameters tells the id of the thread which has to start executing, and the second parameter is the attributes which by default is NULL in our case since we don't need any.
   Error Handling -
   In case of an error an error value is returned, otherwise in successful execution, 0 is returned. So if the value is non zero then an error is generated otherwise it goes on.

2) pthread_join() - this makes the calling thread wait for the termination of the thread which is passed in as the first parameter in this function. The second parameter is the return value which is returned when the thread is terminated, however in our case we simply state NULL since we do not need this return value in any case in our code at all.
   Error Handling -
   In case of an error an error value is returned, otherwise in successful execution, 0 is returned. So if the value is non zero then an error is generated otherwise it goes on.

3) pthread_exit() - this functions is used to terminate the thread which is currently undergoing.It takes in a parameter which is basically the return value however in our code we do not use this return value so it is stated as null in the code
   Error Handling -
   In case of an error this function will almost always be called so that it does not disrupt the functioning of the other thread. This function is used for error validation in a way