Name: Rohan Kulkarni
Roll No: 2020537

Question 1

In order to run the programs, first you need to type make(so that it compiles both Q1a(the first part of question 1) and Q1b(second part of question 1) in one go) incase you want to compile them step wise(or you want to pause at each step) you need to enter make assemble if you only want to assemble, then after that make compile if you only want to compile without linking and then make link to link.If you want to do it step wise for only Q1a then you can do make assemble_Q1a to assemble for just Q1a, make compile_Q1a to compile without linking for Q1a, make link_Q1a for linking just for Q1a. The steps are the same if you want to do it only for Q1b except instead of Q1a you need to write Q1b.

open(): Used to open a file. Helps to read/write to it. Returns -1 if error occurs else returns the file descriptor(int).

read(): Helps to read a file(using file descriptor) and store the contents of a file into a buffer. Return -1 if error occurs else returns the number of bytes read.

write(): Helps to write upto a specified number of bytes, into a buffer. Returns -1 if error occurs else returns the number of bytes written.

fork(): Used to create a child process(or a new process) by duplicating the parent process. Returns -1 if error occurs.

pthread_create(): Helps in starting a new thread in the process from which this is called. Return 0 if error occurs.

pthread_join(): Helps to wait till the specified thread(in the first argument) terminates. Returns 0 if error occurs.

waitpid(): Described in the explanation below. Returns -1 if error occurs.

First I used the fork system call and stored the value returned by it in a variable called ret_value of type pid_t. If the value returned is -1 then I did error handling by printing “Error occurred” and then perror(“fork”) and then exited by exit(errno).
Now for the child process the value returned is 0, so I wrote my child process code here, and if value returned is positive then it is in the parent process, so I wrote my parent process code here.
The following is what I did for both:
First I opened the .csv file using open() system call and passed the name of the .csv file(student_records.csv) as the first parameter and because I only want to read from that file I chose the O_RDONLY as the second parameter of the open() system call.
I used an int variable called fp to store the file descriptor returned by open().
If fp==-1 then I did error handling by using perror(“open”) and then exited by using exit(errno).

Then I read the student_records.csv file and stored the contents of it into a char array called line, using read() system call.
The first argument passed into it was the file descriptor fp returned by open, then I passed the address of the char array line by setting &line as the second argument, and then in the third argument, I chose a large buffer size 1024(I have assigned the value of sz as 1024).
I used variable called nread to store the value returned by read(). If nread==-1 then I I did error handling by using perror("read") and then exited by using exit(errno).

Then using strtok_r I broke the content of the file first linewise and then inside every line I broke it after every comma. I used the second column's value to identify the section, and then I used the rest of the columns to store the marks obtained in each assignment in an array named marks(in the first part, in the second part I used arrays marksA and marksB to store marks of each assignment of section A and section B respectively) and then finally after coming out the two while loops I calculated the average marks in the ith assignment by dividing the marks of the ith assignment by a_cnt(number of students in section A) and b_cnt(number of students in section B) to obtain the averages of each assignment for section A and section B.

Also for the parent to wait till the child process completes, I used waitpid() which helps us to wait for the child process to finish before executing the the parent process. I passed the value return by fork() as the first argument(ret_value) and the second parameter I passed is NULL because my child process is not returning anything so I want to ignore the return status of the child process and the third parameter I passed is 0 because I do not want to pass in a parameter telling it how to wait. If it return -1 then I have handled that by perror("waitpid") and exited by exit(errno).

In the second part of the question 1 of the assignment, I used pthread_create to create a thread using reference of variables of type pthread_t to specify the threads(ret_value1 and ret_value2) as the first argument, since I wanted to use default attributes, I passed NULL as the second argument, in the third argument I passed the reference to the function that this thread needs to work on(&AverageA and &AverageB respectively for A and B functions respectively), and since my functions didn't need any parameters to be passed for them to work so the fourth argument was chosen as NULL. For error handling, if pthread_create() returns a non zero number then I coded perror("pthread_create") and then exited by exit(errno).

But before using pthread_create for the second time, I used pthread_join to wait for the first thread to finish its execution, in pthread_join i passed reference to ret_value1(first thread) and NULL as the second argument since my function doesn't return anything. For error handling, if pthread_join() returns a non zero number then I coded perror("pthread_join") and then exited by exit(errno).

Then I followed the same exact steps for the second thread(corresponding to the function AverageB) by using the pthread_t type variable ret_value2.

To use the results from the two threads to calculate the overall average I first made two global int arrays, marksA and marksB of size 6(as there are 6 assignments) and then after the two threads were finished executing, these arrays contained the total marks in each assignment, for section A and section B students. To count the total number of students I made two global int variables numStudentsA and numStudentsB, and then reused their values while finally printing the overall averages(I added up the total marks of the ith assignment from each section and divided by the total number of students(numStudentsA + numStudentsB), to get the overall average.

To print, I used the write() system call, So whenever I wanted to print, I first created a char array called buff, buff1, buff2 etc of size 1024(a large size) and then used the sprintf() and snprintf() (only for the overall average) function to store the string that I wanted to print in the buff array by passing the name of array as the first argument and the formatted string as the second argument in the sprintf() function.
Then I used write() system call to print the contents of the buff array, I gave 1 as the first argument(for standard output), then name of the array in which the contents to be printed are stored(for ex buff1) and then the third argument was the size of the content for which I used strlen(buff1)(for buff1). I used variable called nwrite to store the value returned by write(). If nwrite==-1 then I did error handling by using perror("write") and then exited by using exit(errno).

Question 2

Steps to compile and run:

When in the Q2 folder, write make in the command line and press enter to compile the program(as its just one program written in 3 different files).
Now to run the program write ./Q2 and press enter.

Details about the makefile:
First I did `nasm -f elf64 B.asm -o B.o` to generate the object file for B.asm(i.e B.o in this case) and then I compiled and linked all of the three files (A.c, B.o and C.c) and created an executable file called Q2 by using `gcc -no-pie A.c B.o C.c -o Q2.`

In the A.c file, in the main function I have called the A() function.
In the A function I have used int64_t data type to store a 64 bit number into a variable of type int_64 called num. I have stored this number

```
491495383406
```

This basically will print the ascii characters corresponding to my first name(rohan)
I have declared an external void function called _B() in the A() function which takes in a 64 bit number as parameter(input type- int64_t).

The _B function is implemented in a file named B.asm. This function interpretes the integer passed as an 8 byte ascii string and then prints the corresponding ascii characters.
In another .c file called C.c I have made another function called C() which when called just prints that “You are in C” and then exits. I have called this function from _B by pushing the address of C before ret so that when ret is executed the return address is overwritten by the address of C(rbp has been popped dearlier) because ret pops the address at the top of the stack into rip register(intruction pointer which stores the address of the next instruction to be executed). So instead of returning back to A it now returns to C because return address of _B has been overwritten by the address of C.
Also to print the ascii characters I first did mov rax, rdi so that the 64 bit value passed in function A gets into rax register. To print the ascii characters I first divided the number in rax register by 256(i.e 2^8)(using rbx register) so that the remainder is the value corresponding to the last 8 bits and the quotient is the value corresponding to to the rest of the bits, the remainder get stored in rdx register so I stored the content of the rdx register using rcx register.To do so I first declared a memory space called text in section .bss and made rcx point to that memory location(the memory address of text), then I stored the rdx value into memory address pointed by rcx and then always increased the rcx register so that it points to next memory location/address everytime to store the incoming rdx value. I did this under a label called _iterate(which I called from _B) and continued doing this till the quotient(or the value of the rax register doesn’t become zero. When it became 0, I called _printLoop, which basically called the label _printNum which printed the character present inside the memory address pointed to by the rcx register using the write() system call, I passed first parameter as 1(using rax) for setting ID of sys_write(), then I passed 1 as the file descriptor using rdi register, then I passed the address of the character to be printed using rsi register(I passed rcx as the address) then finally I passed in the count of the number of bytes to be printed as 1 using rdx register. But always first in the _printNum label I decreased the value of rcx so that we are printing the characters in the reverse order.
I continued this process till the memory address pointed by rcx is greater than or equal to the memory address of text, so that all the characters that were stored starting from the memory address of text are printed(In reverse order because we are reaching the memory address of text at the end). Then I returned from the printLoop label into _B again and called C as explained above.
Also In B.asm I have printed the line “You are in B” and “The ascii characters are: “ using sys_write() only as explained above.