Assignment 1
Question 1 Documentation

Q1_a) USING FORK:

How the program works:

1. The program at the start declares the variables and the arrays to be used in the program.
2. After that, it calls the fork() command.
3. In my machine, the parent is preferred to the child, so when the command goes to the parent, there it encounters the waitpid command.
4. At the same time, the child is also being executed. In the child, it the first parses the given CSV file using the open() and read() commands. Then the data is divided into arrays for ease of use.
5. Since the child has to handle the section A students, it correspondingly accesses the data from the arrays and computes the averages of each assignment.
6. Then using the write() system call, it prints the data on the screen.
7. After this, the child exits using the exit() command.
8. After this, the parent's waiting period is over and it executes the above steps for the students of section B

System Calls Used:

1. **fork():** This command is used to create the child process. No arguments are passed in it. It returns ◦ 0 if returned to the newly created child process. ◦ Negative Value: the creation of a child process was unsuccessful. ◦ Positive value: Returned to parent or caller. The value contains the process ID of the newly created child process.
2. **open():** Used to Open the file for reading, writing or both. It takes 2 arguments: ◦ Path ◦ Flags. It returns: ◦ file descriptor being used for the file. ◦ -1 on failure
3. **read():** From the file indicated by the file descriptor fd, the read() function reads cnt bytes of input into the memory area indicated by buf. It takes 3 arguments: ◦ fd: file descriptor ◦ buf: buffer to read data from ◦ cnt: length of buffer It returns: ◦ return Number of bytes read on success ◦ return 0 on reaching the end of file ◦ return -1 on error ◦ return -1 on signal interrupt
4. **write():** Writes cnt bytes from buf to the file or socket associated with fd. It takes 3 arguments: ◦ fd: file descriptor ◦ buf: buffer to write data to ◦ cnt: length of buffer It returns: ◦ return Number of bytes written on success ◦ return 0 on reaching the end of file ◦ return -1 on error ◦ return -1 on signal interrupt
5. **close():** Used to close the file being pointed by the file descriptor. It takes 1 argument: ◦ file descriptor of the file to be closed It returns: ◦ 0 on success ◦ -1 on failure
6. **waitpid():** If we want to reap any specific child process, waitpid is used it returns ◦ pid of the child if the child has exited ◦ 0 if the child hasn't exited

Q1_b) USING THREADS:

How the program works:

1. This subpart has more or less the same working. Here instead of fork(), I used the pthread_create() command to create a new thread.
2. Then the second thread is created for simultaneous processing. Then both the threads are joined one by one to end execution in the simultaneous threads.
3. The first and second threads do exactly what the child and parent processes did respectively in the first part.
4. To calculate the Assignment average overall sections, there is a struct to be passed as args to the thread, which stores the total marks for each assignment and then the average is calculated.

System Calls Used (apart from part **(a)**):

1. **pthread_create():** function starts a new thread in the calling process. On success, it returns 0; on error, it returns an error number, and the contents of *thread are undefined.
2. **pthread_join():** function waits for the thread specified by a thread to terminate. If that thread has already terminated, then pthread_join() returns immediately. The thread specified by the thread must be joinable.