Method of running the program:
From the terminal go to the directory where the folder is located. Once inside the directory, enter the command ‘make’. After make is completed, run the desired program by ./{program name} in the same directory. For the names of the program, you can use ‘ls’ command in the same directory.

System calls and potential errors:

1. fork(): It is used to create a child class and returns the PID of the child process. Error handled: If PID returned is not 0 (-ve number returned) , that means it has failed to create a child process and the system exits.
2. waitpid(pid_t pid, int *status, int options): Used for waiting for the child process to complete. The parameters are child pid, exit status of the child process(NULL here), and flags(0 here).
3. open(char *path, int flag): Opens the file according to the flag inputted. Error handled: If file cannot be opened, throws error.
4. read(int fd, void *buffer, size_t count): Reads the file uptil count number of bytes and stores it in buffer for use.
5. close(int fd): Closes the file. Error handled: throws error if file cannot be closed.
6. exit(int num): terminates the programme.
7. pthread_create(pthread_t * thread, const pthread_attr_t * attr, void * (*start_routine)(void *), void *arg): It is used to create a new thread. Parameters that go thread address, attribute (NULL here), void function, NULL(here). Error handled: If pthread_create doesn’t return 0, then system exit call is made along with an error message.
8. pthread_join(pthread_t thread, void **thread_return): Used to wait for the termination of thread. Parameters passed are thread, return address(NULL here). Error Handling: Throws error and exits if doesn’t return 0.

Program Runthrough:
Q1a) We first fork the process. If the fork returns a negative value, programme terminates with an error message, else it calls function averageout(“A”) for computing the averages of section A. The waitpid() is used to wait for the child process to terminate after which the parent process averageout(“B”) is called for computing the averages of section B.

Inside the averageout() function, we make an array to store the total marks scored in each assignment by the section students. The file is opened in read-only mode since there is nothing to write on the file. If the file fails to open, error is printed and programme terminates. The programme moves to the next part where it starts reading the contents of the file byte by byte.
It stores these contents (line) in an array until it arrives at “\n”, from where it goes into the part where the line is divided into tokens using strtok, giving us the marks and section. The section is validated and then marks are added to their particular cell in an array using atof(). This happens

until the while condition does not break, i.e. it keeps finding characters. Also, when the section is validated 'n' is incremented by 1, for keeping check of number students required for calculating average.

Lastly, the averages are calculated and results are displayed. After this, the file is closed. If the file is not closing, it displays an error and exits.

Q1b) The process is completely the same for threads but instead of fork(), we create two threads using pthread_create and use pthread_join is used to wait for the created thread to terminate. The functions used were averageoutA() and averageoutB() which were hardcoded for sections and were required to be void for the pthread_create command.
However, the total marks of each assignment of each section were stored in global variables and their number of students were also stored in it, which helped in calculating the entire average.