Use ‘make’ to preprocess, compile, assemble the source file.
Then use ‘make run’ or ./main to run the executable file.
Part 1 -> fork()
Part 2-> threads
If want to do
The program is forked using fork()
System calls used:

1) fork()

This command is used to create the child process.
No arguments are passed in it.
It returns ->
0 if return to new created child process,
Positive value if returned to the parent process(value contains process ID of newly created child process.)
Negative value if error occurred in creation of child process.
All three cases are handled in main by if else, both the processes start continue from if else statement which is in main.

2) open()
Used to open files for read or write operations,
Arguments -> two (path of the file, flags)
Returns -> file descriptor being used for the file, -1 on failure, when file is not found or file cannot be opened.
3) read()
Read function reads ‘buffer_size’ bytes of data, into the memory indicated by ’buffer_str’, from the file depicted by file descriptor ‘fd’
Arguments: fd, buffer_str, buffer_size
fd -> file descriptor
buffer_str -> buffer to read data from
buffer_size->size/length of the buffer

Returns Number of bytes read,
Returns -1 when error or signal interrupt.

4)
close()
Arguments -> fd (file descriptor)
Used to close the file being pointed by the file descriptor.
Returns 0 on success
Returns -1 on failure

5)

waitpid()
Used to wait for a specific child process to end.
Returns pid of child, if child process exited
Returns 0 if child process is still going on.
Arguments -> (pid_t pid, int *status_ptr, int options)
pid -> Specifies the child processes the caller wants to wait for
*status_ptr -> here in this program it's set to NULL.
So waitpid() ignores the child's return status.
Options -> Specifies additional information for waitpid()

6)
write()
Writes buff_size bytes from buffer to the file associated with file descriptor ‘fd’.
Arguments: fd: file descriptor, buffer, buffer_size: length of buffer

Returns Number of bytes written, return 0 on reaching end of filereturn -1 on error or signal interrupt.

Question 1 part 1:

Some variables and arrays are declared which are used by the program.
Program is forked by fork() system call which creates a new copy of the process.

Fork returns:

0 if return to new created child process,
Positive value if returned to the parent process(value contains process ID of newly created child process.)
Negative value if error occurred in creation of child process.
All three cases are handled in main by if else, both the processes start continue from if else statement which is in main.

In the child process, we read the student data present in the csv file, the data read is stored in the buffer and only the students which belong to section A are printed on stdout by child process(). In case of any error in reading the file or the file cannot be accessed all these errors are checked, and the error is printed.

By waitpid() the parent process waits till the child process is completed and does not interfere when the child process is writing to stdout.

After completion of child process , parent process starts writing details of assignment of section B students,

Question 1 part 2:
In this part threads are used, one thread is created ThreadFuncForSectionA()
To handle the writing to stdout for section A. all the printing for section B is done by the main thread.
Here, as both threads are run parallely.
All the errors which are checked in part 1 all those are implemented here too, also pthread_join() is used, this function waits for the thread specified by the main thread to terminate.