1_1b

Command to Run

```
@make
```

Make File Break Down

This line creates the output generated by the preprocessor which has a .i extension. This file has all headers included, comments removed and macros replaced

```
@gcc -E 1b.c -o 1b.i
```

This line generates the output assembly file. Note this is pure assembly which is denoted by the lower case s in the extension '.s'. A capital '.S' indicates an assembly code which still needs to go through a preprocessor.

```
@gcc -S 1b.i -o 1b.s
```

The subsequent 3 lines generate and link the object file (.o) which then produces the executable which is run. The -pthread denotes linking with the pthread library

```
@gcc -c 1b.s -o 1b.o
@gcc 1b.o -pthread -o 1b
@./1b
```

General Code Breakdown

- Inside the main two p_thread objects are created.
- pthread_create() is then used with the parameters denoting the first thread, The attr parameter is set to Null, the function processDatasecA is passed as third parameter and is typecasted to void * while the 4th parameter which denotes the processDatasecA function's parameters is again null;
- A sleep(1) call is then used to wait for the current thread for 1 second
- Then pthread_create() is again used this time the first parameter is the second thread and the 3rd parameter is the processDataSecB function. Rest remains the same.
- p_threadjoin() is then used backtoback with the 2 threads. The first one would return immediately since the sleep(1) provided sufficient time for thread 1 process to complete on almost any modern day processor while the second one might take time and wait incase the second thread is still running.
- Then using a for loop the average for the 2 sections is computed and printed to the screen. The values are taken from 2 global arrays. They will not cause any problems with threads as both threads write in

different preallocated sections of the memory.

Function Specific Breakdown (Including Error Handling) -

- processDatasecA(void* args) and processDataSecB(void* args)

> The process for the 2 functions is almost identical except for the occasional differences in some places due to the section being different, apart from the code remains largely identical.

  - Open syscall is used to open the csv. The parameters are the file name and the oflag. From the documentation -

    > Values for oflag are constructed by a bitwise-inclusive OR of flags from the following list, defined in <fcntl.h> O_RDONLY specifies the file is Read Only since we do need to do any editing this is the perfect choice.

  - The return value is stored in a int variable file descriptor which is then used to check for errors. A -1 value stored indicates error while reading and hence an perror is raised.
  - Some variables are initialized for the process of reading and a read syscall is made with the arguments filedescriptor denoting the file which needs to be read, a buffer size which counts which specifies how many bytes from the file should be read and the count denotes the count parameter.
  - The errors are checked if the returned value of read is less than 0 a perror is raised.
  - Subsequently the program counts the lines using a while loop
  - Appropiately sized arrays are made using the line count
  - The file is reopened and errors are handled again as described above.
  - The csv file is then read line by line in the data matrix
  - Then using strtok() and atoi() tokens are generated and numeric tokens are converted to number from originally being strings. The scores are stored in the marks matrix and sections are stored in the sections array.
  - Finally the close() syscall is used with the filedescriptor to close reading the file since the data is already extracted. Incase it has an error which is when it returns -1 a perror is raised denoting the same.
  - 6 Variables are initialized to store the different collective scores of the assignments. A counter is mainted to count how many submissions were there in a given section.
  - The for loop then runs and at any place if the section in the sectionTracker array matches the required section parameter it increments counter and adds the value in the variables.
  - Finally the section size and total marks for individual assignments are stored inside the section_size variable and the marks_store structure respectively.
  - The marks_store is a marks_structure struct which has the following form and hence all assignment scores are stored within it-

```
struct marks_structure
    {
        int total_assignment1;
        int total_assignment2;
        int total_assignment3;
        int total_assignment4;
```

```
        int total_assignment5;
        int total_assignment6;
    };
```

- The values are also stored in the appropriate global array depending on the section.
- Finally the average is computed and stored in avg variables for respective assignments. The strings are then passed inside snprintf with the required formatting of what needs to be printed which are then sent to the stringPrinter function. snprintf takes the String where the value is to be stored, the size of the buffer and a C string which denotes the format.
- write() syscall is used with the parameter 1, str and length of the string str. The 1 denotes the filedescriptor where it has to be written i.e. standard output in this case. str denotes the string which needs to be sent to stdout and the length tells the syscall how many bytes will be there.
- Incase of an error a -1 is returned by write which is handled by the if block. Subsequently if errno saved by write is EINTR it means "The call was interrupted by a signal before any data was written" and hence an appropriate perror is raised.