Name: Jatin Kumar Sharma
Roll No. : 2020563

---

1. **Makefile**: It contains commands to compile and the both file with the threading and fork process and produce the executable file along with .s, .i, .o file (part of break the compilation process) and a txt file which stores the output printed on console.

2. **readline**: this function bytewise reads the .csv file and stores that into the data using **read** syscall inside a while loop which restricts the function to a single line only.

3. **cleansum**: this function accepts the pointer of a line of .csv file using **strtok_r** which returns the pointer of the first token and uses buffer_context introduced in the function to store the left string. We tokenize the string until the whole string tokenizer and checks for the section (A for child and B for Parent) then proceeds for other token containing marks and add them to corresponding position of the marks array (i.e. add to value present at $(i-1)^{th}$ position for i^{th} assignment) and hence this function stores the sum of marks of all student in the marks array for a particular section assigned as $(i-1)^{th}$ position contains sum for i^{th} assignment.

4. **printAndDumb**: this function uses snprintf to store the formatted output in a char array(i.e a string formation) and then write it to the console using the **stdout file** and to the file descriptor of a pre opened file using **write** syscall.

student_fork.c: This file has 4 functions viz readline, cleansum, printAndDumb as described above and a main.

The main function of this file first opens a file named fork_Dumb.txt to store the console output using **open** syscall. Now the program uses **fork()** command to create a branch of the same process called as child process. which returns pid value as -1 for unsuccessful execution of fork which is handled using if statement and 0(Child Process) and 1(Parent Process) on successful execution. We uses **waitkey** in parent process which accept the pid of the process for wait is required(here child process pid = 0) and after this both of the process works similarly they first read every line sequentially of this csv file using readline function and a while loop in which we have used cleansum to store the grades in marks array as per the condition for that process and thenn uses printAndDumb function to print and dumb the average to the console and dumbfile.

Student_pthread.c: This file has 4 functions viz readline, cleansum, printAndDumb as described above other than this a mychild, my parent and a main. Mychild and my parent functions does the same work but for different threads, i.e. thread1 and thread2 described as follows:

These functions first open the csv file and after that they first sequentially read every line of this csv file using readline function and a while loop in which we have used cleansum to store the grades in marks array as per the condition for that process and thenn uses printAndDumb function to print and dumb the average to the console and dumbfile than perform pthread_exit.

The main function of this .c file has 2 arrays for storing marks of both sections and total no student in that section as value present at last location. This function creates two threads one by one to

ensure parallel threading along with passing the arguments asmarksA array, mychild function to one of them and marksB, my parent to another one. These functions do the work as explained before and then join these threads one by one. Now using a for loop main computes the total marks for each section and total no students and then perform printAndDumb function with the argument for average across section.

Different SYSCALL used And their Handing.

1. open:

```
int temp = open("filename.txt",  O_WRONLY | O_CREAT, 0644);
   if(temp < 0){
       perror("Error while opening the file that is used to dump!\n");

       exit(0);
   }
```

2. read:

```
int file_size = read(filedesc, &data[i], 1);
       if(file_size < 0){
           perror("Line:20-error while reading the file");
           return 0;
       }
```

3. write:

```
int wr = write(STDOUT_FILENO, print1, strlen(print1));
       if(wr < 0){
           perror("Error while writing to stdout");
           exit(0);
       }
```

4. close:

```
int cl = close(filedesc);
       if(cl < 0) {
           perror("Error while closing the file!");
           exit(0);
       }
```

5. fork:

```
pid_t pid = fork();

   if(pid < 0) {
       perror("Failure in forking!\n");
       exit(1);
   }
```

6. waitpid:

```
 waitpid(0, NULL, 0);
```

7. Pthread_create:

```
int thread = pthread_create(&child, NULL, myChild, (void*)marksA);
   if(thread != 0){
       perror("Error while creating thread");
```

```
    }

void *myChild(void *vargp) {
        //code elided
}
```

8. pthread_join:

```
int thread = pthread_join(child, NULL);
    if(thread != 0){
        perror("Line:207-Error while creating thread");
    }
```

Outputs: