Faculty of Engineering and Applied Science

SOFE 3950U / CSCI 3020U: Operating Systems

Tutorial 4

Group Member 1

Name: Alexander Campbell

Student ID: 100703650

Group Member 2

Name: Atharshan Kennedy

Student ID: 100590243

Group Member 3

Name: Joey Villafuerte

Student ID: 100759003

Date: Mar 13, 2022

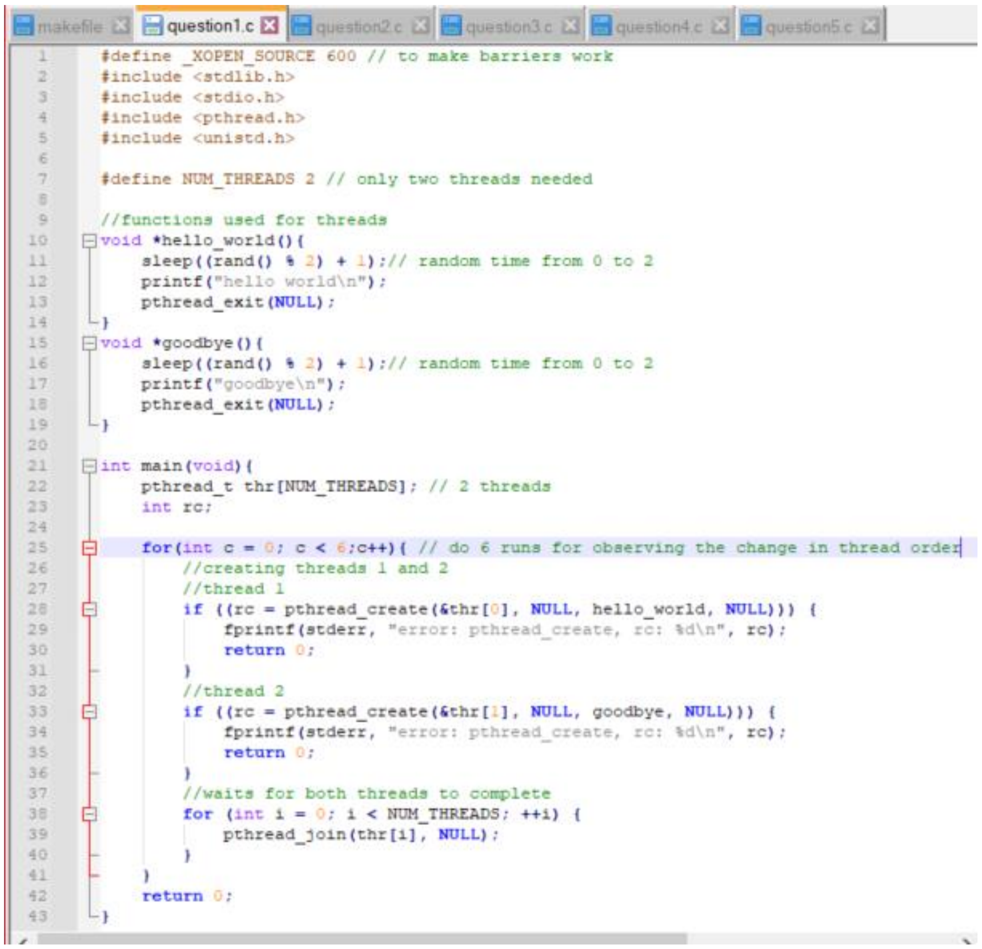
Conceptual Questions

1. - pthread_create() is the function that creates the new thread.
   - pthread_join() will wait for the thread that's specified in the thread argument to terminate before continuing execution of the program.
   - pthread_exit will terminate the current thread that is running and return it with a return val specified by the retval argument.
2. Processes do not generally share the same memory and to do so requires additional memory schemes. Threads can more easily share the same memory as they are given the same memory state as the process, unless specified otherwise.
3. Threads consume a lot less resources than processes, this makes multithreading a much better choice for most applications. Another difference is the ease with which there is inter-thread or inter-process communication. Threads are much easier to communicate with each other, while processes are much more difficult and take system calls in order to do so. Processes have their own memory space, while threads share memory unless otherwise specified. This makes it so heavyweight; memory intensive processes are generally better in multiprocessing, and it is also an advantage when you want separate memory.
4. Mutual exclusion is a technique in multithreading that is used to prevent race conditions between threads, meaning that one thread may access a resource before it's ready and cause undefined behaviors. This means that it cannot access a critical section before it's ready. The critical section is a section of the code that is a shared resource between threads or processes.
5. - pthread_cond_wait will make a thread wait until a certain condition is met
   - pthread_cond_timedwait will make a thread wait for a specified time
   - pthread_rwlock_init will lock a pthread until the thread is unlocked by another thread or process
   - pthread_mutex_init will create a mutex, or a mutually exclusive flag, which is an object that allows threads to take turns accessing a certain resource

   All of these functions can be used to control what the thread is working on and ensure that it will not access a critical section.

Code Explanation

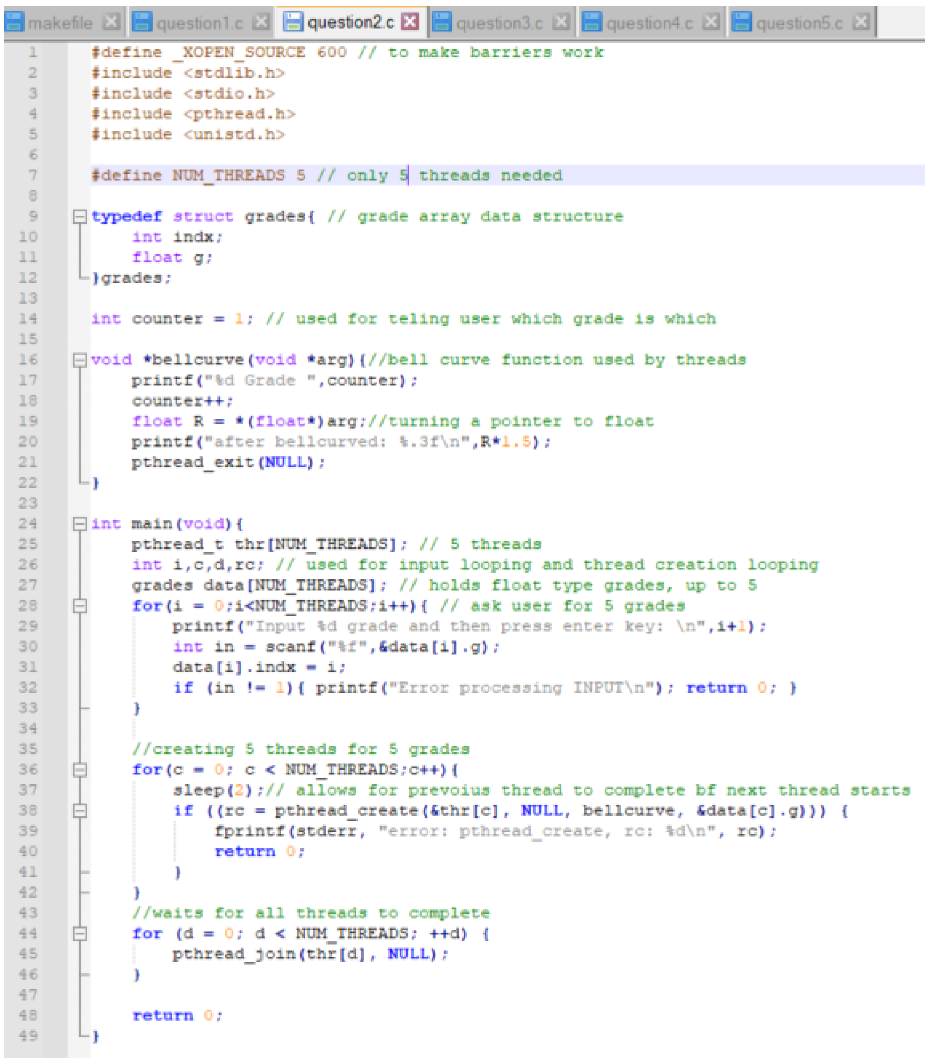
Q1:


makefile question1.c question2.c question3.c question4.c question5.c

```c
#define _XOPEN_SOURCE 600 // to make barriers work
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

#define NUM_THREADS 2 // only two threads needed

//functions used for threads
void *hello_world(){
    sleep((rand() % 2) + 1);// random time from 0 to 2
    printf("hello world\n");
    pthread_exit(NULL);
}
void *goodbye(){
    sleep((rand() % 2) + 1);// random time from 0 to 2
    printf("goodbye\n");
    pthread_exit(NULL);
}

int main(void){
    pthread_t thr[NUM_THREADS]; // 2 threads
    int rc;

    for(int c = 0; c < 6;c++){ // do 6 runs for observing the change in thread order
        //creating threads 1 and 2
        //thread 1
        if ((rc = pthread_create(&thr[0], NULL, hello_world, NULL))) {
            fprintf(stderr, "error: pthread_create, rc: %d\n", rc);
            return 0;
        }
        //thread 2
        if ((rc = pthread_create(&thr[1], NULL, goodbye, NULL))) {
            fprintf(stderr, "error: pthread_create, rc: %d\n", rc);
            return 0;
        }
        //waits for both threads to complete
        for (int i = 0; i < NUM_THREADS; ++i) {
            pthread_join(thr[i], NULL);
        }
    }
    return 0;
}
```

The first program creates two threads one after another with one exit before the next thread. The first thread will print ‘hello world’ then next thread will print ‘goodbye’. Th sleep random from 0 to 2 sec is used to change the order of the which will print first which is done randomly for both prints.

Q2:

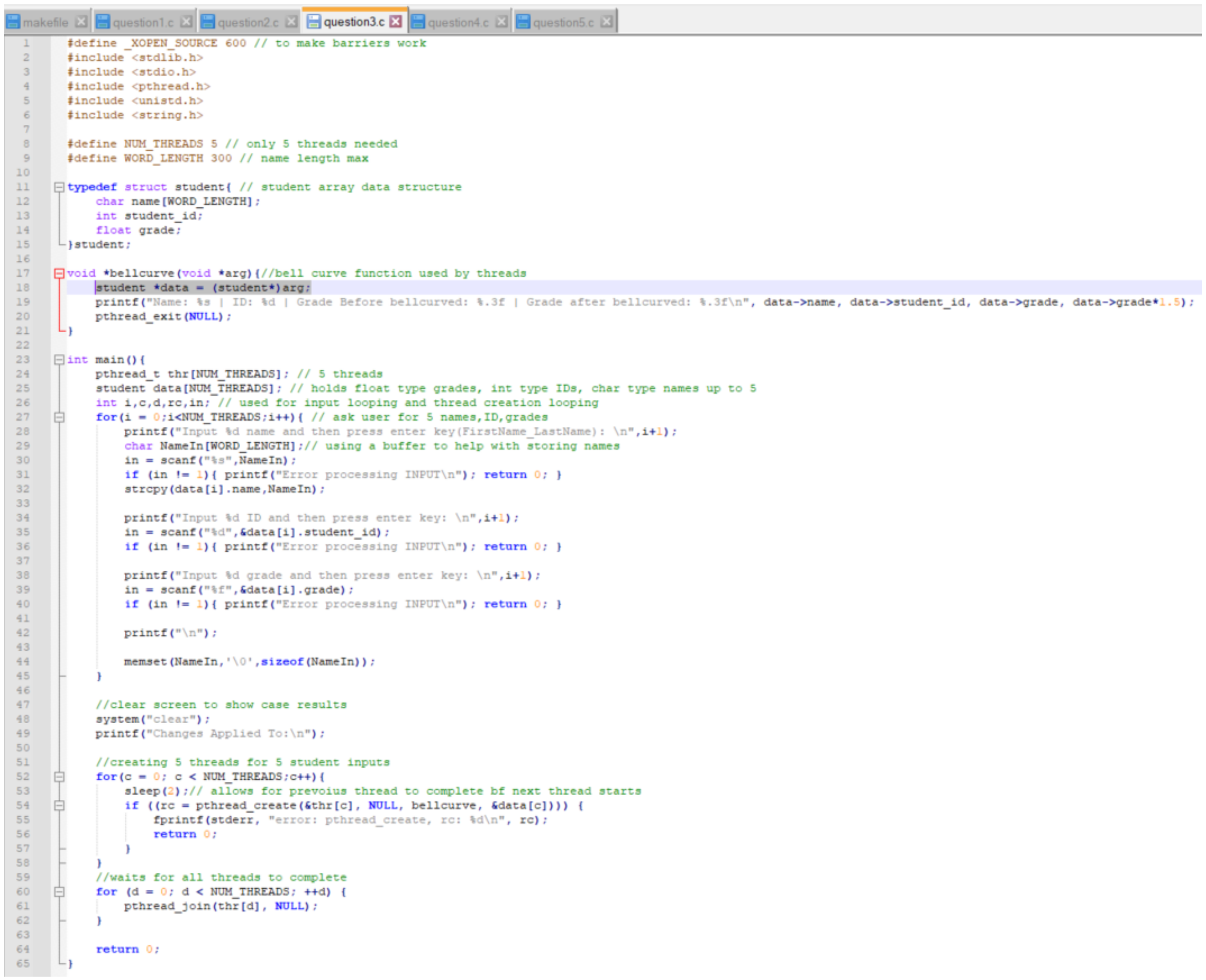


makefile | question1.c | question2.c | question3.c | question4.c | question5.c

```c
#define _XOPEN_SOURCE 600 // to make barriers work
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

#define NUM_THREADS 5 // only 5 threads needed

typedef struct grades{ // grade array data structure
    int indx;
    float g;
}grades;

int counter = 1; // used for teling user which grade is which

void *bellcurve(void *arg){//bell curve function used by threads
    printf("%d Grade ",counter);
    counter++;
    float R = *(float*)arg;//turning a pointer to float
    printf("after bellcurved: %.3f\n",R*1.5);
    pthread_exit(NULL);
}

int main(void){
    pthread_t thr[NUM_THREADS]; // 5 threads
    int i,c,d,rc; // used for input looping and thread creation looping
    grades data[NUM_THREADS]; // holds float type grades, up to 5
    for(i = 0;i<NUM_THREADS;i++){ // ask user for 5 grades
        printf("Input %d grade and then press enter key: \n",i+1);
        int in = scanf("%f",&data[i].g);
        data[i].indx = i;
        if (in != 1){ printf("Error processing INPUT\n"); return 0; }
    }

    //creating 5 threads for 5 grades
    for(c = 0; c < NUM_THREADS;c++){
        sleep(2);// allows for prevoius thread to complete bf next thread starts
        if ((rc = pthread_create(&thr[c], NULL, bellcurve, &data[c].g))) {
            fprintf(stderr, "error: pthread_create, rc: %d\n", rc);
            return 0;
        }
    }
    //waits for all threads to complete
    for (d = 0; d < NUM_THREADS; ++d) {
        pthread_join(thr[d], NULL);
    }

    return 0;
}
```

Th second program will take 5 grade inputs. The grades are first stored into an array that use the struct called grades as it's framework. Afterwards the 5 threads are created one after another with the second for loop. Within the thread function a counter before bell curved grade is printed and then changes. Then the bell curved grade is printed before the thread exits. Each grade is multiplied by 1.5 before being printed. The counter is used to help list the grades. And sleep of two secs is used before thread creation to make sure the previous thread prints and exit before the next thread.

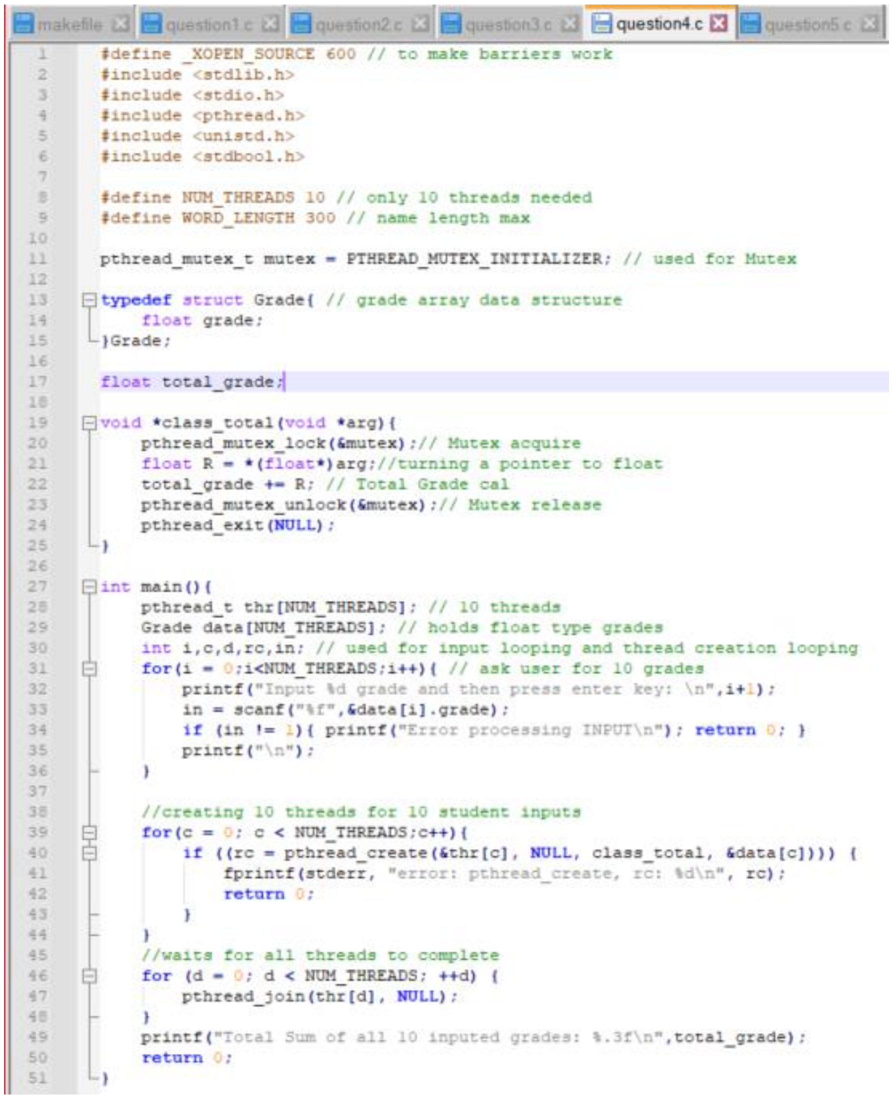
Q3:



```c
#define _XOPEN_SOURCE 600 // to make barriers work
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <string.h>

#define NUM_THREADS 5 // only 5 threads needed
#define WORD_LENGTH 300 // name length max

typedef struct student{ // student array data structure
    char name[WORD_LENGTH];
    int student_id;
    float grade;
}student;

void *bellcurve(void *arg){//bell curve function used by threads
    student *data = (student*)arg;
    printf("Name: %s | ID: %d | Grade Before bellcurved: %.3f | Grade after bellcurved: %.3f\n", data->name, data->student_id, data->grade, data->grade*1.5);
    pthread_exit(NULL);
}

int main(){
    pthread_t thr[NUM_THREADS]; // 5 threads
    student data[NUM_THREADS]; // holds float type grades, int type IDs, char type names up to 5
    int i,c,d,rc,in; // used for input looping and thread creation looping
    for(i = 0;i<NUM_THREADS;i++){ // ask user for 5 names,ID,grades
        printf("Input %d name and then press enter key(FirstName_LastName): \n",i+1);
        char NameIn[WORD_LENGTH];// using a buffer to help with storing names
        in = scanf("%s",NameIn);
        if (in != 1){ printf("Error processing INPUT\n"); return 0; }
        strcpy(data[i].name,NameIn);

        printf("Input %d ID and then press enter key: \n",i+1);
        in = scanf("%d",&data[i].student_id);
        if (in != 1){ printf("Error processing INPUT\n"); return 0; }

        printf("Input %d grade and then press enter key: \n",i+1);
        in = scanf("%f",&data[i].grade);
        if (in != 1){ printf("Error processing INPUT\n"); return 0; }

        printf("\n");

        memset(NameIn,'\0',sizeof(NameIn));
    }

    //clear screen to show case results
    system("clear");
    printf("Changes Applied To:\n");

    //creating 5 threads for 5 student inputs
    for(c = 0; c < NUM_THREADS;c++){
        sleep(2);// allows for prevoius thread to complete bf next thread starts
        if ((rc = pthread_create(&thr[c], NULL, bellcurve, &data[c]))) {
            fprintf(stderr, "error: pthread_create, rc: %d\n", rc);
            return 0;
        }
    }
    //waits for all threads to complete
    for (d = 0; d < NUM_THREADS; ++d) {
        pthread_join(thr[d], NULL);
    }

    return 0;
}
```

The third question takes 5 names, ID, grades as input and saves the data to an array that use the struct called student as its framework. Afterwards the screen will clear and will start creating 5 threads with the second for loop. Sleep with two seconds is used again to make sure that the previous thread is finished and terminated before the next thread. And each thread will print out the name of the student followed by the student ID which is then followed by the grade before bell curve and grade after bell curve, where 1.5 is multiplied to the current grade input that is to be outputted.

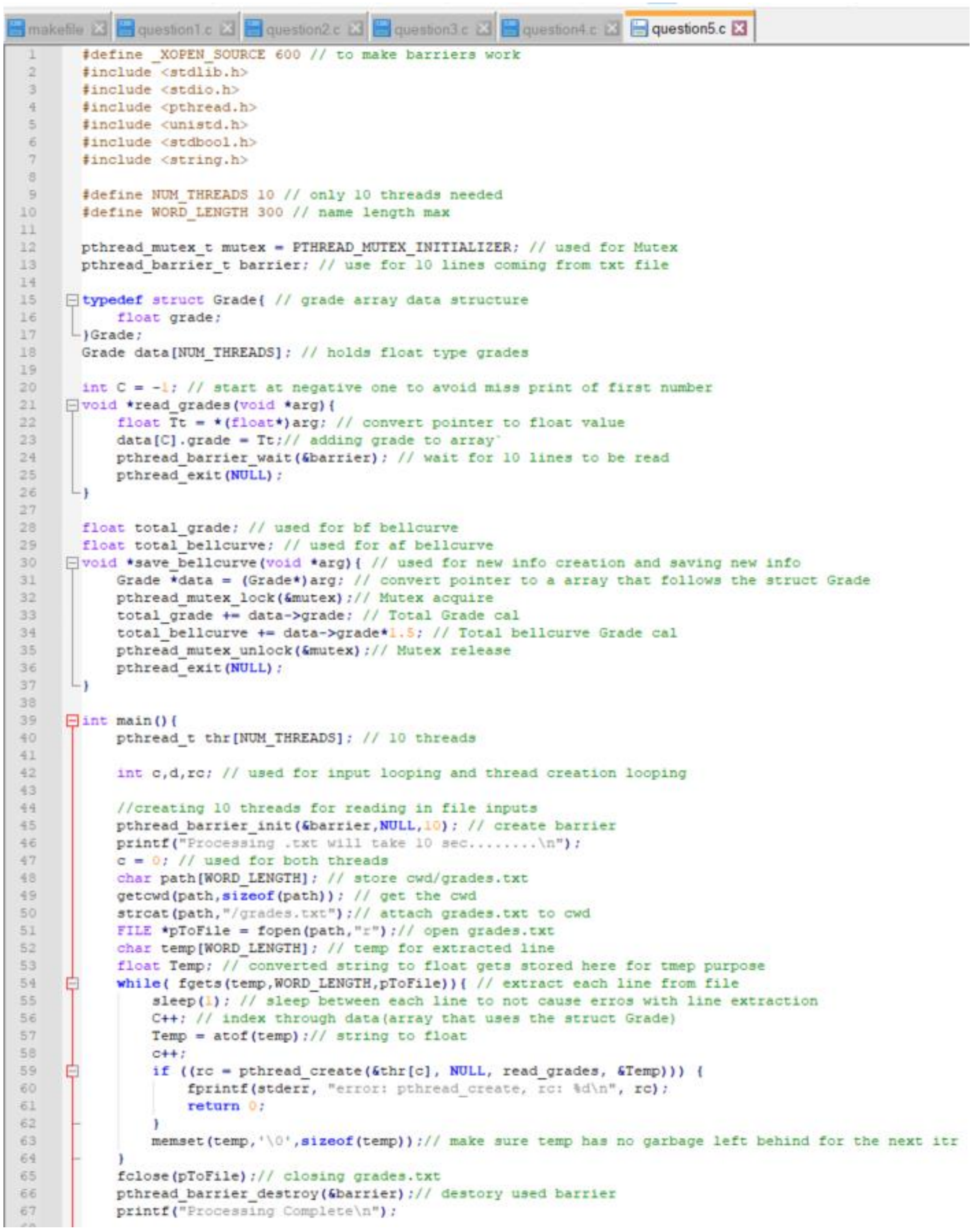
Q4:



```c
#define _XOPEN_SOURCE 600 // to make barriers work
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <stdbool.h>

#define NUM_THREADS 10 // only 10 threads needed
#define WORD_LENGTH 300 // name length max

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; // used for Mutex

typedef struct Grade{ // grade array data structure
    float grade;
}Grade;

float total_grade;

void *class_total(void *arg){
    pthread_mutex_lock(&mutex);// Mutex acquire
    float R = *(float*)arg;//turning a pointer to float
    total_grade += R; // Total Grade cal
    pthread_mutex_unlock(&mutex);// Mutex release
    pthread_exit(NULL);
}

int main(){
    pthread_t thr[NUM_THREADS]; // 10 threads
    Grade data[NUM_THREADS]; // holds float type grades
    int i,c,d,rc,in; // used for input looping and thread creation looping
    for(i = 0;i<NUM_THREADS;i++){ // ask user for 10 grades
        printf("Input %d grade and then press enter key: \n",i+1);
        in = scanf("%f",&data[i].grade);
        if (in != 1){ printf("Error processing INPUT\n"); return 0; }
        printf("\n");
    }

    //creating 10 threads for 10 student inputs
    for(c = 0; c < NUM_THREADS;c++){
        if ((rc = pthread_create(&thr[c], NULL, class_total, &data[c]))) {
            fprintf(stderr, "error: pthread_create, rc: %d\n", rc);
            return 0;
        }
    }
    //waits for all threads to complete
    for (d = 0; d < NUM_THREADS; ++d) {
        pthread_join(thr[d], NULL);
    }
    printf("Total Sum of all 10 inputed grades: %.3f\n",total_grade);
    return 0;
}
```

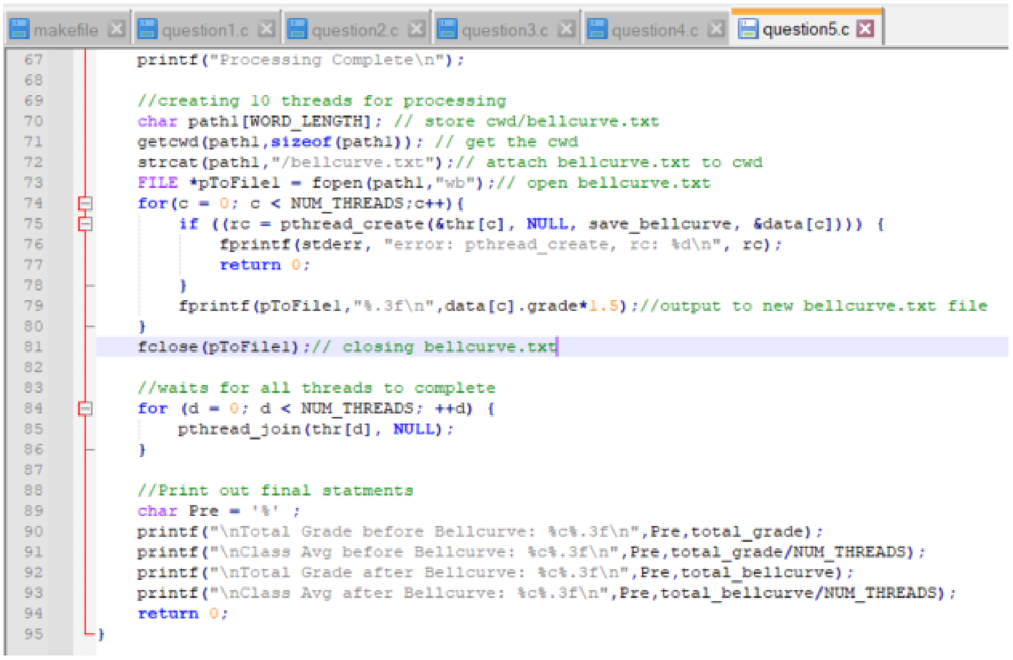
The fourth question will take 10 grades as input. The grades are stored in an array that uses the struct Grade as its framework. Once that is done the second for loop will create 10 threads with each thread adding the grade to the total grade global variable. Here, in order prevent two or more threads from accessing the same global variable, Mutex is used to lock other threads out while still allowing one thread to go in. After the total grade has been calculated, it is finally printed out.

Q5:


makefile question1.c question2.c question3.c question4.c question5.c

```c
#define _XOPEN_SOURCE 600 // to make barriers work
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <stdbool.h>
#include <string.h>

#define NUM_THREADS 10 // only 10 threads needed
#define WORD_LENGTH 300 // name length max

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; // used for Mutex
pthread_barrier_t barrier; // use for 10 lines coming from txt file

typedef struct Grade{ // grade array data structure
    float grade;
}Grade;
Grade data[NUM_THREADS]; // holds float type grades

int C = -1; // start at negative one to avoid miss print of first number
void *read_grades(void *arg){
    float Tt = *(float*)arg; // convert pointer to float value
    data[C].grade = Tt;// adding grade to array`
    pthread_barrier_wait(&barrier); // wait for 10 lines to be read
    pthread_exit(NULL);
}

float total_grade; // used for bf bellcurve
float total_bellcurve; // used for af bellcurve
void *save_bellcurve(void *arg){ // used for new info creation and saving new info
    Grade *data = (Grade*)arg; // convert pointer to a array that follows the struct Grade
    pthread_mutex_lock(&mutex);// Mutex acquire
    total_grade += data->grade; // Total Grade cal
    total_bellcurve += data->grade*1.5; // Total bellcurve Grade cal
    pthread_mutex_unlock(&mutex);// Mutex release
    pthread_exit(NULL);
}

int main(){
    pthread_t thr[NUM_THREADS]; // 10 threads

    int c,d,rc; // used for input looping and thread creation looping

    //creating 10 threads for reading in file inputs
    pthread_barrier_init(&barrier,NULL,10); // create barrier
    printf("Processing .txt will take 10 sec........\n");
    c = 0; // used for both threads
    char path[WORD_LENGTH]; // store cwd/grades.txt
    getcwd(path,sizeof(path)); // get the cwd
    strcat(path,"/grades.txt");// attach grades.txt to cwd
    FILE *pToFile = fopen(path,"r");// open grades.txt
    char temp[WORD_LENGTH]; // temp for extracted line
    float Temp; // converted string to float gets stored here for tmep purpose
    while( fgets(temp,WORD_LENGTH,pToFile)){ // extract each line from file
        sleep(1); // sleep between each line to not cause erros with line extraction
        C++; // index through data(array that uses the struct Grade)
        Temp = atof(temp);// string to float
        c++;
        if ((rc = pthread_create(&thr[c], NULL, read_grades, &Temp))) {
            fprintf(stderr, "error: pthread_create, rc: %d\n", rc);
            return 0;
        }
        memset(temp,'\0',sizeof(temp));// make sure temp has no garbage left behind for the next itr
    }
    fclose(pToFile);// closing grades.txt
    pthread_barrier_destroy(&barrier);// destory used barrier
    printf("Processing Complete\n");
```

In the final program a thread barrier is first initiated before the while loop. This will be used mainly with a thread function read_grades(). Then a file called grade.txt is opened. The file contains 10 grades that must be read line by line. After a line is extracted with the while loop fgets it is sent to read_grades() thread function. In read_grades() the grade value is saved into a global array that uses the Grade struct as it's framework. After each thread saves a grade value to the global array it waits at the barrier in read_grades() until 10 threads as arrived. After 10 threads as arrived the barrier opens allowing for all threads to flow out and exit synchronously. After the while loop has finished the file is closed and the barrier is destroyed.

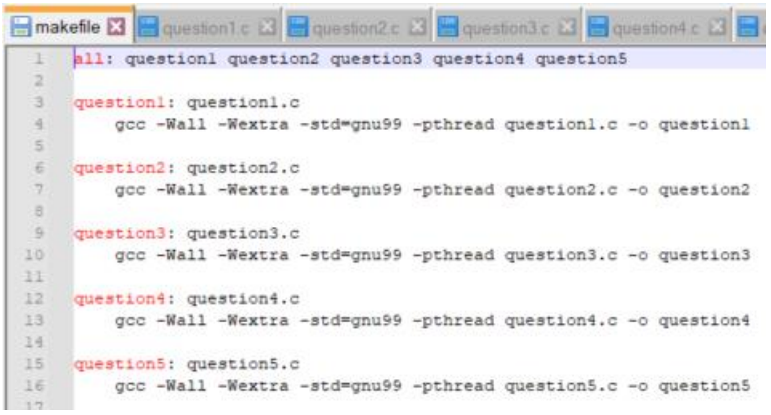


makefile | question1.c | question2.c | question3.c | question4.c | question5.c

```c
    printf("Processing Complete\n");

    //creating 10 threads for processing
    char path1[WORD_LENGTH]; // store cwd/bellcurve.txt
    getcwd(path1,sizeof(path1)); // get the cwd
    strcat(path1,"/bellcurve.txt");// attach bellcurve.txt to cwd
    FILE *pToFile1 = fopen(path1,"wb");// open bellcurve.txt
    for(c = 0; c < NUM_THREADS;c++){
        if ((rc = pthread_create(&thr[c], NULL, save_bellcurve, &data[c]))) {
            fprintf(stderr, "error: pthread_create, rc: %d\n", rc);
            return 0;
        }
        fprintf(pToFile1,"%.3f\n",data[c].grade*1.5);//output to new bellcurve.txt file
    }
    fclose(pToFile1);// closing bellcurve.txt

    //waits for all threads to complete
    for (d = 0; d < NUM_THREADS; ++d) {
        pthread_join(thr[d], NULL);
    }

    //Print out final statments
    char Pre = '%' ;
    printf("\nTotal Grade before Bellcurve: %c%.3f\n",Pre,total_grade);
    printf("\nClass Avg before Bellcurve: %c%.3f\n",Pre,total_grade/NUM_THREADS);
    printf("\nTotal Grade after Bellcurve: %c%.3f\n",Pre,total_bellcurve);
    printf("\nClass Avg after Bellcurve: %c%.3f\n",Pre,total_bellcurve/NUM_THREADS);
    return 0;
}
```

The next part of question 5 will create a file and write to bellcurve.txt. The for loop will be used with save_bellcurve() thread function when creating the threads. Each grade from the global array is sent to this thread function to be added to both total_grade and total_bullcurve global variables. As a thread comes in it locks others out until it has finished with Mutex lock. Going back to the first for loop in int mian(), after a creation of thread is done the new bell curved grade is saved to the bellcurve.txt file. After for loop has finished the file bellcurver.txt is closed. Finally, the Total grade and Avg of before and after the bell curve is outputted.

Make File:

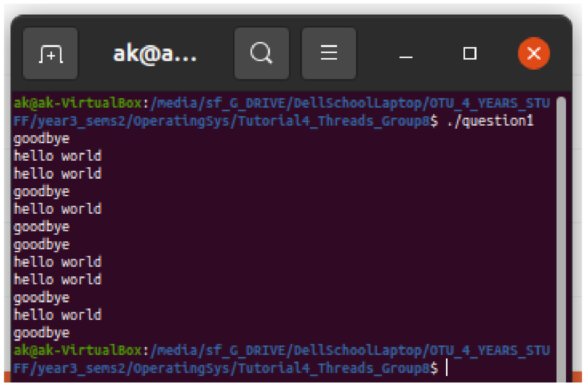


makefile | question1.c | question2.c | question3.c | question4.c

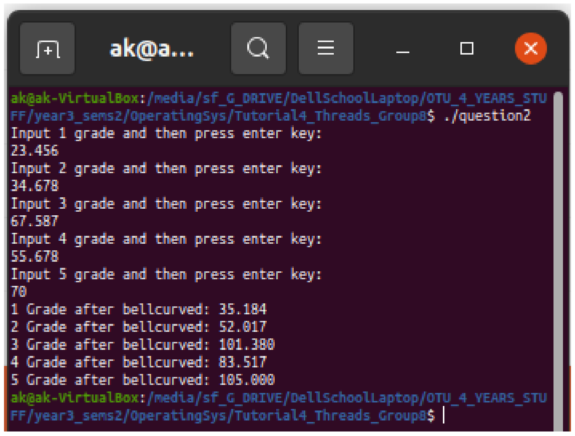
```makefile
all: question1 question2 question3 question4 question5

question1: question1.c
	gcc -Wall -Wextra -std=gnu99 -pthread question1.c -o question1

question2: question2.c
	gcc -Wall -Wextra -std=gnu99 -pthread question2.c -o question2

question3: question3.c
	gcc -Wall -Wextra -std=gnu99 -pthread question3.c -o question3

question4: question4.c
	gcc -Wall -Wextra -std=gnu99 -pthread question4.c -o question4

question5: question5.c
	gcc -Wall -Wextra -std=gnu99 -pthread question5.c -o question5
```

Make file used for compiling questions 1 to 5.

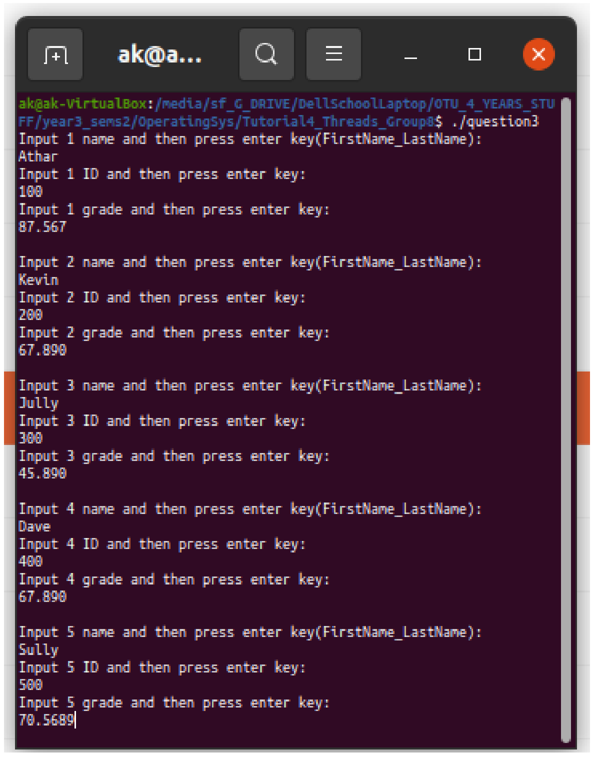
Sample Runs



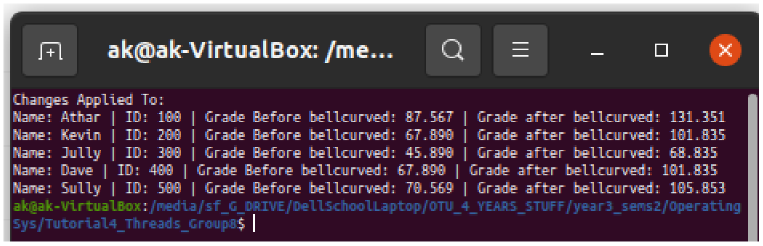
```
ak@ak-VirtualBox:/media/sf_G_DRIVE/DellSchoolLaptop/OTU_4_YEARS_STU
FF/year3_sems2/OperatingSys/Tutorial4_Threads_Group8$ ./question1
goodbye
hello world
hello world
goodbye
hello world
goodbye
goodbye
hello world
hello world
goodbye
hello world
goodbye
ak@ak-VirtualBox:/media/sf_G_DRIVE/DellSchoolLaptop/OTU_4_YEARS_STU
FF/year3_sems2/OperatingSys/Tutorial4_Threads_Group8$
```



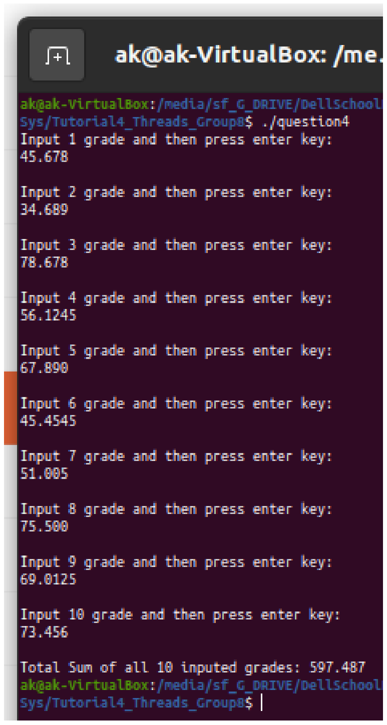
```
ak@ak-VirtualBox:/media/sf_G_DRIVE/DellSchoolLaptop/OTU_4_YEARS_STU
FF/year3_sems2/OperatingSys/Tutorial4_Threads_Group8$ ./question2
Input 1 grade and then press enter key:
23.456
Input 2 grade and then press enter key:
34.678
Input 3 grade and then press enter key:
67.587
Input 4 grade and then press enter key:
55.678
Input 5 grade and then press enter key:
70
1 Grade after bellcurved: 35.184
2 Grade after bellcurved: 52.017
3 Grade after bellcurved: 101.380
4 Grade after bellcurved: 83.517
5 Grade after bellcurved: 105.000
ak@ak-VirtualBox:/media/sf_G_DRIVE/DellSchoolLaptop/OTU_4_YEARS_STU
FF/year3_sems2/OperatingSys/Tutorial4_Threads_Group8$
```



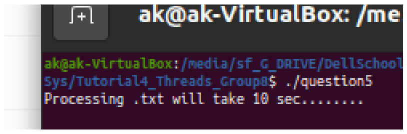
```
ak@ak-VirtualBox:/media/sf_G_DRIVE/DellSchoolLaptop/OTU_4_YEARS_STU
FF/year3_sems2/OperatingSys/Tutorial4_Threads_Group8$ ./question3
Input 1 name and then press enter key(FirstName_LastName):
Athar
Input 1 ID and then press enter key:
100
Input 1 grade and then press enter key:
87.567

Input 2 name and then press enter key(FirstName_LastName):
Kevin
Input 2 ID and then press enter key:
200
Input 2 grade and then press enter key:
67.890

Input 3 name and then press enter key(FirstName_LastName):
Jully
Input 3 ID and then press enter key:
300
Input 3 grade and then press enter key:
45.890

Input 4 name and then press enter key(FirstName_LastName):
Dave
Input 4 ID and then press enter key:
400
Input 4 grade and then press enter key:
67.890

Input 5 name and then press enter key(FirstName_LastName):
Sully
Input 5 ID and then press enter key:
500
Input 5 grade and then press enter key:
70.5689
```



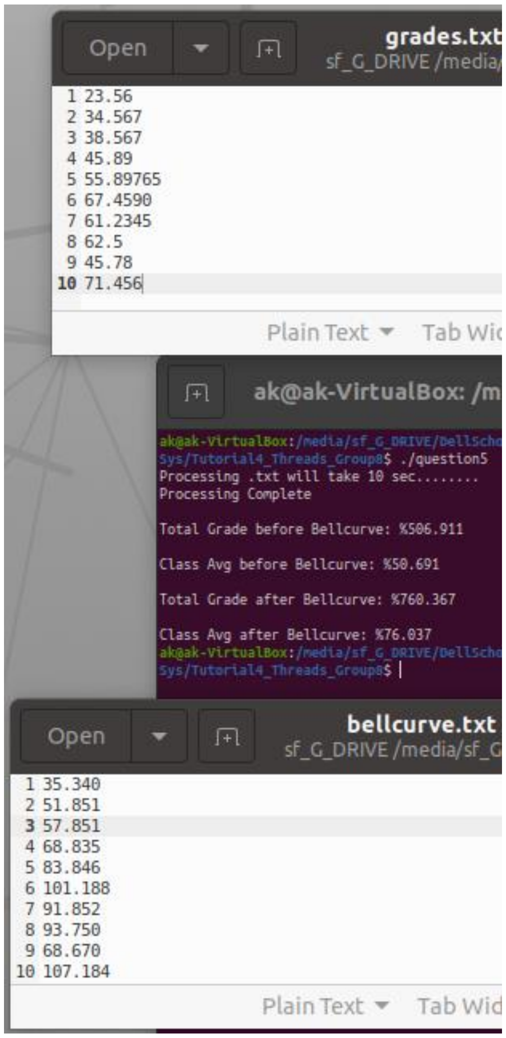
```
Changes Applied To:
Name: Athar | ID: 100 | Grade Before bellcurved: 87.567 | Grade after bellcurved: 131.351
Name: Kevin | ID: 200 | Grade Before bellcurved: 67.890 | Grade after bellcurved: 101.835
Name: Jully | ID: 300 | Grade Before bellcurved: 45.890 | Grade after bellcurved: 68.835
Name: Dave | ID: 400 | Grade Before bellcurved: 67.890 | Grade after bellcurved: 101.835
Name: Sully | ID: 500 | Grade Before bellcurved: 70.569 | Grade after bellcurved: 105.853
ak@ak-VirtualBox:/media/sf_G_DRIVE/DellSchoolLaptop/OTU_4_YEARS_STUFF/year3_sems2/Operating
Sys/Tutorial4_Threads_Group8$
```



```
ak@ak-VirtualBox:/media/sf_G_DRIVE/DellSchool
Sys/Tutorial4_Threads_Group8$ ./question4
Input 1 grade and then press enter key:
45.678

Input 2 grade and then press enter key:
34.689

Input 3 grade and then press enter key:
78.678

Input 4 grade and then press enter key:
56.1245

Input 5 grade and then press enter key:
67.890

Input 6 grade and then press enter key:
45.4545

Input 7 grade and then press enter key:
51.005

Input 8 grade and then press enter key:
75.500

Input 9 grade and then press enter key:
69.0125

Input 10 grade and then press enter key:
73.456

Total Sum of all 10 inputed grades: 597.487
ak@ak-VirtualBox:/media/sf_G_DRIVE/DellSchool
Sys/Tutorial4_Threads_Group8$
```



```
ak@ak-VirtualBox:/media/sf_G_DRIVE/DellSchool
Sys/Tutorial4_Threads_Group8$ ./question5
Processing .txt will take 10 sec........
```



```
23.56
34.567
38.567
45.89
55.89765
67.4590
61.2345
62.5
45.78
71.456
```

```
ak@ak-VirtualBox:/media/sf_G_DRIVE/DellScho
Sys/Tutorial4_Threads_Group8$ ./question5
Processing .txt will take 10 sec........
Processing Complete

Total Grade before Bellcurve: %506.911

Class Avg before Bellcurve: %50.691

Total Grade after Bellcurve: %760.367

Class Avg after Bellcurve: %76.037
ak@ak-VirtualBox:/media/sf_G_DRIVE/DellScho
Sys/Tutorial4_Threads_Group8$
```

```
35.340
51.851
57.851
68.835
83.846
101.188
91.852
93.750
68.670
107.184
```