Huỳnh Tấn Dương 3122410061

Hoàn thành 2/2 Lab 6

Bài 1/

bubble.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/time.h>

#define SEED  100
#define LENGTH 10000
#define UPPER_LIM 1000
#define LOWER_LIM  1
#define NUM_THREADS 2

const int NUMBERS_PER_THREAD = LENGTH / NUM_THREADS;
const int OFFSET = LENGTH % NUM_THREADS;
int arr[LENGTH];//biến toàn cục

int generate_random_number(unsigned int lower_limit, unsigned int upper_limit);
void bubblesort(int arr[], int low, int high);
void* thread_bubble_sort(void* pi);
void test_array_is_in_order(int arr[]);
struct data{
        int low;
        int high;
};
/*-------------------------------------------------------------------------
int main(int argc, const char * argv[]) {
    srand(SEED);
    struct timeval  start, end;
    double time_spent;

        int i1;
        for (i1 = 0; i1 < LENGTH; i1 ++)
        arr[i1] = generate_random_number(LOWER_LIM, UPPER_LIM);

        for (i1 = 0; i1 < LENGTH; i1 ++) {
        printf("%d ",arr[i1]);}
        printf("\n ");//in ra giá trị sinh ngẫu nhiên trc khi sắp xếp
```

```
    pthread_t threads[NUM_THREADS];
    gettimeofday(&start, NULL);    /* begin timing */
    int pivot=partition(arr,0,LENGTH);
    struct data pi[NUM_THREADS];
        pi[0].low=0;
        pi[0].high=pivot-1;
        pi[1].low=pivot+1;
        pi[1].high=LENGTH;

    /* create threads */
        int i;
     for (i = 0; i < NUM_THREADS; i ++) {
        int rc = pthread_create(&threads[i], NULL, thread_bubble_sort, (void*)&pi[i]);
        if (rc){
            printf("ERROR; return code from pthread_create() is %d\n", rc);
            exit(-1);
        }
    }
    for( i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }
        for (i1 = 1; i1 <= LENGTH; i1 ++) {
        printf("%d ",arr[i1]);}//in ra các giá trị sinh ngẫu nhiên sau khi sắp xếp

    gettimeofday(&end, NULL);/* end timing */
    time_spent = ((double) ((double) (end.tv_usec - start.tv_usec) / 1000000 +(double) (end.tv_sec - start.tv_sec)));
    printf("\nTime taken for execution: %f seconds\n", time_spent);
    test_array_is_in_order(arr);
    return 0;
}
/*-------------------------------------------------------------------------------------------------------*/
int generate_random_number(unsigned int lower_limit, unsigned int upper_limit) {
    return lower_limit + (upper_limit - lower_limit) * ((double)rand() / RAND_MAX);
}
```

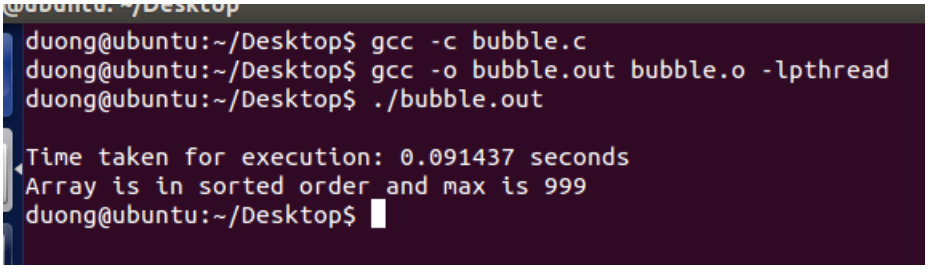
```c
void *thread_bubble_sort(void* pi){/* assigns work to each thread to perform merge sort */
    struct data *temp= (struct data*) pi;
    int low = temp->low;
    int high=temp->high;

    if (low < high){
        /* pi là chỉ số nơi phần tử này đã đứng đúng vị trí pivot
         và là phần tử chia mảng làm 2 mảng con trái & phải */
        int pi = partition(arr, low, high);
        // Gọi đệ quy sắp xếp 2 mảng con trái và phải
        bubblesort(arr, low, pi);
        bubblesort(arr, pi + 1, high);
    }
}
void test_array_is_in_order(int arr[]) {
    int max = 0,i;
    for (i = 1; i < LENGTH; i ++) {
        if (arr[i] >= arr[i - 1]) {
            max = arr[i];
        } else
            printf("Error. Out of order sequence: %d found\n", arr[i]);
    }
    printf("Array is in sorted order and max is %d\n",max);
}

void bubblesort(int arr[], int low, int high) {
    int i, j;
    for (i = low; i < high - 1; i++) {
        for (j = high - 1; j > i; j--) {
            if (arr[j] < arr[j - 1]) {
                int t = arr[j];
                arr[j] = arr[j - 1];
                arr[j - 1] = t;
            }
        }
    }
}
int partition (int arr[], int low, int high){//partition function
    int pivot = arr[high];    // pivot
    int left = low;
    int right = high - 1;
    while(1){
        while(left <= right && arr[left] < pivot) left++;
        while(right >= left && arr[right] > pivot) right--;
        if (left >= right) break;
        int t=arr[left];
        arr[left]=arr[right];
        arr[right]=t;

        left++;
        right--;
    }
        int t=arr[left];//swap pivot với left để đẩy pivot vào giữa chia ra 2 đoạn > < pivot
        arr[left]=arr[high];
        arr[high]=t;

    return left;//left swap với pivot giá trị để chia 2 bên lớn hơn pivot và nhỏ hơn pivot nên
                //khi return left là return pivot
}
```


duong@ubuntu:~/Desktop$ gcc -c bubble.c
duong@ubuntu:~/Desktop$ gcc -o bubble.out bubble.o -lpthread
duong@ubuntu:~/Desktop$ ./bubble.out
Time taken for execution: 0.091437 seconds
Array is in sorted order and max is 999
duong@ubuntu:~/Desktop$

Bài 2/

quick.c ×

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/time.h>

/* define variables for the problem */
#define SEED  100
#define LENGTH 1000000
#define UPPER_LIM 1000000
#define LOWER_LIM  1
#define NUM_THREADS 2

/* define derived values from the variables */
const int NUMBERS_PER_THREAD = LENGTH / NUM_THREADS;
const int OFFSET = LENGTH % NUM_THREADS;
int arr[LENGTH];//biến toàn cục

int generate_random_number(unsigned int lower_limit, unsigned int upper_limit);
void quick_sort(int arr[], int low, int high);
int partition (int arr[], int low, int high);
void merge(int arr[], int left, int middle, int right);
void* thread_quick_sort(void* pi);
void merge_sections_of_array(int arr[], int number, int aggregation);
void test_array_is_in_order(int arr[]);
struct data{
        int low;
        int high;
};
int main(int argc, const char * argv[]) {
    srand(SEED);
    struct timeval  start, end;
    double time_spent;

        int i1;
        for (i1 = 0; i1 < LENGTH; i1 ++) {
        arr[i1] = generate_random_number(LOWER_LIM, UPPER_LIM); /* initialize array with random numbers */
    }
        /*for (i1 = 0; i1 < LENGTH; i1 ++) {
        printf("%d ",arr[i1]);}
        printf("\n ");//in ra giá trị sinh ngẫu nhiên trc khi sắp xếp*/

    pthread_t threads[NUM_THREADS];
    gettimeofday(&start, NULL);    /* begin timing */
    int pivot=partition(arr,0,LENGTH);
    struct data pi[NUM_THREADS];
        pi[0].low=0;
        pi[0].high=pivot-1;
        pi[1].low=pivot+1;
        pi[1].high=LENGTH;

    /* create threads */
        int i;
     for (i = 0; i < NUM_THREADS; i ++) {
        int rc = pthread_create(&threads[i], NULL, thread_quick_sort, (void*)&pi[i]);
        if (rc){
            printf("ERROR; return code from pthread_create() is %d\n", rc);
            exit(-1);
        }
    }
    for( i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }
    merge_sections_of_array(arr, NUM_THREADS, 1);
        /*for (i1 = 0; i1 < LENGTH; i1 ++) {
        printf("%d ",arr[i1]);}//in ra các giá trị sinh ngẫu nhiên sau khi sắp xếp*/

    gettimeofday(&end, NULL);/* end timing */
    time_spent = ((double) ((double) (end.tv_usec - start.tv_usec) / 1000000 +(double) (end.tv_sec - start.tv_sec)));
    printf("Time taken for execution: %f seconds\n", time_spent);
    /* test to ensure that the array is in sorted order */
     test_array_is_in_order(arr);
    return 0;
}

int generate_random_number(unsigned int lower_limit, unsigned int upper_limit) {/* generate random numbers within the specified limit */
    return lower_limit + (upper_limit - lower_limit) * ((double)rand() / RAND_MAX);
}
```

```c
void merge_sections_of_array(int arr[], int number, int aggregation) {/* merge locally sorted sections */
        int i;
    for(i = 0; i < number; i = i + 2) {
        int left = i * (NUMBERS_PER_THREAD * aggregation);
        int right = ((i + 2) * NUMBERS_PER_THREAD * aggregation) - 1;
        int middle = left + (NUMBERS_PER_THREAD * aggregation) - 1;
        if (right >= LENGTH) {
            right = LENGTH - 1;
        }
        /*lần for thứ i=0 cho ra left=o,midle=499999,right=999999,lần gọi đệ quy thi cho ra left=0,midle=999999,right=1999999
        nhưng vì right>lenght nên right=lenght-1=999999*/
        merge(arr, left, middle, right);
        /*có 2 lần gọi merge(arr,0,499999,999999)và merge(arr,0,999999,999999)*/
    }
    if (number / 2 >= 1) {
        merge_sections_of_array(arr, number / 2, aggregation * 2);
    }
}
void *thread_quick_sort(void* pi){/* assigns work to each thread to perform merge sort */
    struct data *temp= (struct data*) pi;
    int low = temp->low;
    int high=temp->high;

    if (low < high){
        /* pi là chỉ số nơi phần tử này đã đứng đúng vị trí pivot
         và là phần tử chia mảng làm 2 mảng con trái & phải */
        int pi = partition(arr, low, high);
        // Gọi đệ quy sắp xếp 2 mảng con trái và phải
        quick_sort(arr, low, pi - 1);
        quick_sort(arr, pi + 1, high);
    }
}
```

```c
void test_array_is_in_order(int arr[]) {/* test to ensure that the array is in sorted order */
    int max = 0,i;
    for (i = 1; i < LENGTH; i ++) {
        if (arr[i] >= arr[i - 1]) {
            max = arr[i];
        } else
            printf("Error. Out of order sequence: %d found\n", arr[i]);
    }
    printf("Array is in sorted order\n");
}
void quick_sort(int arr[], int low, int high) {/* perform quick sort */
    if (low < high){
        /* pi là chỉ số nơi phần tử này đã đứng đúng vị trí
         và là phần tử chia mảng làm 2 mảng con trái & phải */
        int pi = partition(arr, low, high);

        // Gọi đệ quy sắp xếp 2 mảng con trái và phải
        quick_sort(arr, low, pi - 1);
        quick_sort(arr, pi + 1, high);
    }
}
int partition (int arr[], int low, int high){/* partition function */
    int pivot = arr[high];    // pivot
    int left = low;
    int right = high - 1;
    while(1){
        while(left <= right && arr[left] < pivot) left++;
        while(right >= left && arr[right] > pivot) right--;
        if (left >= right) break;
        int t=arr[left];
        arr[left]=arr[right];
        arr[right]=t;

        left++;
        right--;
    }
        int t=arr[left];//swap pivot với left để đẩy pivot vào giữa chia ra 2 đoạn > < pivot
```

```c
        int t=arr[left];//swap pivot với left để đẩy pivot vào giữa chia ra 2 đoạn > < pivot
        arr[left]=arr[high];
        arr[high]=t;

    return left;/*left swap với pivot giá trị để chia 2 bên lớn hơn pivot và nhỏ hơn pivot nên khi return left là return pivot*/
}
/* merge function */
void merge(int arr[], int left, int middle, int right) {
    int i = 0;
    int j = 0;
    int k = 0;
    int left_length = middle - left + 1;
    int right_length = right - middle;
    int left_array[left_length];
    int right_array[right_length];

    /* copy values to left array */

        for (i = 0; i < left_length; i ++) {
        left_array[i] = arr[left + i];
    }

    /* copy values to right array */
        for (j = 0; j < right_length; j ++) {
        right_array[j] = arr[middle + 1 + j];
    }

    i = 0;
    j = 0;
    /** chose from right and left arrays and copy */
    while (i < left_length && j < right_length) {
        if (left_array[i] <= right_array[j]) {
            arr[left + k] = left_array[i];
            i ++;
        } else {
            arr[left + k] = right_array[j];
            j ++;
        }
```

```c
    while (i < left_length && j < right_length) {
        if (left_array[i] <= right_array[j]) {
            arr[left + k] = left_array[i];
            i ++;
        } else {
            arr[left + k] = right_array[j];
            j ++;
        }
        k ++;
    }

    /* copy the remaining values to the array */
    while (i < left_length) {
        arr[left + k] = left_array[i];
        k ++;
        i ++;
    }
    while (j < right_length) {
        arr[left + k] = right_array[j];
        k ++;
        j ++;
    }
}
```

```
duong@ubuntu:~/Desktop$ gcc -c quick.c
duong@ubuntu:~/Desktop$ gcc -o quick.out quick.o -lpthread
duong@ubuntu:~/Desktop$ ./quick.out
Time taken for execution: 0.178349 seconds
Array is in sorted order
duong@ubuntu:~/Desktop$
```

merge.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/time.h>

/* define variables for the problem */
#define SEED  100
#define LENGTH 1000000
#define UPPER_LIM 1000000
#define LOWER_LIM  1
#define NUM_THREADS 6

/* define derived values from the variables */
const int NUMBERS_PER_THREAD = LENGTH / NUM_THREADS;
const int OFFSET = LENGTH % NUM_THREADS;
int arr[LENGTH];

/* function definitions */
int generate_random_number(unsigned int lower_limit, unsigned int upper_limit);
void merge_sort(int arr[], int left, int right);
void merge(int arr[], int left, int middle, int right);
void* thread_merge_sort(void* arg);
void merge_sections_of_array(int arr[], int number, int aggregation);
void test_array_is_in_order(int arr[]);

int main(int argc, const char * argv[]) {
    srand(SEED);
    struct timeval  start, end;
    double time_spent;
        int i1;
        for (i1 = 0; i1 < LENGTH; i1 ++) {/* initialize array with random numbers */
        arr[i1] = generate_random_number(LOWER_LIM, UPPER_LIM);
    }
        /*for (i1 = 0; i1 < LENGTH; i1 ++) {
        printf("%d ",arr[i1]);}
        printf("\n ");*/

    pthread_t threads[NUM_THREADS];
    gettimeofday(&start, NULL);/* begin timing */

        long i;
    for (i = 0; i < NUM_THREADS; i ++) {/* create threads */
```

```
    for (i = 0; i < NUM_THREADS; i ++) {/* create threads */
        int rc = pthread_create(&threads[i], NULL, thread_merge_sort, (void *)i);
        if (rc){
            printf("ERROR; return code from pthread_create() is %d\n", rc);
            exit(-1);
        }
    }
    for( i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }
    merge_sections_of_array(arr, NUM_THREADS, 1);
        /*for (i1 = 0; i1 < LENGTH; i1 ++) {
        printf("%d ",arr[i1]);}*/

    gettimeofday(&end, NULL);/* end timing */
    time_spent = ((double) ((double) (end.tv_usec - start.tv_usec) / 1000000 +(double) (end.tv_sec - start.tv_sec)));
    printf("Time taken for execution: %f seconds\n", time_spent);
    /* test to ensure that the array is in sorted order */
    /* test_array_is_in_order(arr); */
    return 0;
}

int generate_random_number(unsigned int lower_limit, unsigned int upper_limit) {/* generate random numbers within the specified limit */
    return lower_limit + (upper_limit - lower_limit) * ((double)rand() / RAND_MAX);
}

void merge_sections_of_array(int arr[], int number, int aggregation) {/* merge locally sorted sections */
        int i;
    for(i = 0; i < number; i = i + 2) {
        int left = i * (NUMBERS_PER_THREAD * aggregation);
        int right = ((i + 2) * NUMBERS_PER_THREAD * aggregation) - 1;
        int middle = left + (NUMBERS_PER_THREAD * aggregation) - 1;
        if (right >= LENGTH) {
            right = LENGTH - 1;
        }
        merge(arr, left, middle, right);
    }
    if (number / 2 >= 1) {
        merge_sections_of_array(arr, number / 2, aggregation * 2);
    }
}

void *thread_merge_sort(void* arg){/* assigns work to each thread to perform merge sort */
    int thread_id = (long)arg;
    int left = thread_id * (NUMBERS_PER_THREAD);
    int right = (thread_id + 1) * (NUMBERS_PER_THREAD) - 1;
    if (thread_id == NUM_THREADS - 1) {/*nếu là thread cuối thì xử lý nốt mấy số còn bị dư khi chia lenght cho numthread*/
        right += OFFSET;
    }
    int middle = left + (right - left) / 2;
    if (left < right) {
        merge_sort(arr, left, middle);
        merge_sort(arr, middle + 1, right);
        merge(arr, left, middle, right);
    }
}

void test_array_is_in_order(int arr[]) {/* test to ensure that the array is in sorted order */
    int max = 0,i;
    for (i = 1; i < LENGTH; i ++) {
        if (arr[i] >= arr[i - 1]) {
            max = arr[i];
        } else
            printf("Error. Out of order sequence: %d found\n", arr[i]);
    }
    printf("Array is in sorted order\n");
}

void merge_sort(int arr[], int left, int right) {/* perform merge sort */
    if (left < right) {
        int middle = left + (right - left)/2;
        merge_sort(arr, left, middle);
        merge_sort(arr, middle + 1, right);
        merge(arr, left, middle, right);
    }
```

```c
void merge(int arr[], int left, int middle, int right) {/* merge function */
    int i = 0;
    int j = 0;
    int k = 0;
    int left_length = middle - left + 1;
    int right_length = right - middle;
    int left_array[left_length];
    int right_array[right_length];
    /* copy values to left array */
        for (i = 0; i < left_length; i ++) {
        left_array[i] = arr[left + i];
    }
    /* copy values to right array */
        for (j = 0; j < right_length; j ++) {
        right_array[j] = arr[middle + 1 + j];
    }
    i = 0;
    j = 0;
    /* chose from right and left arrays and copy */
    while (i < left_length && j < right_length) {
        if (left_array[i] <= right_array[j]) {
            arr[left + k] = left_array[i];
            i ++;
        } else {
            arr[left + k] = right_array[j];
            j ++;
        }
        k ++;
    }
    /* copy the remaining values to the array */
    while (i < left_length) {
        arr[left + k] = left_array[i];
        k ++;
        i ++;
    }
    while (j < right_length) {
        arr[left + k] = right_array[j];
        k ++;
        j ++;
    }
}
```

```
duong@ubuntu:~/Desktop$ gcc -c merge.c
duong@ubuntu:~/Desktop$ gcc -o merge.out merge.o -lpthread
duong@ubuntu:~/Desktop$ ./merge.out
Time taken for execution: 0.164412 seconds
Array is in sorted order
duong@ubuntu:~/Desktop$
```

LAB 7

fcfs.c

```c
#include<stdio.h>
void main(){
        int n,i,j,sum=0;
        int arrv[10], ser[10], start[10], finish[10],wait[10], turn[10];
        float avgturn=0.0,avgwait=0.0;
        start[0]=0;
        system("clear");
                printf("\n ENTER THE NO. OF PROCESSES:");
                scanf("%d",&n);
        for(i=0;i<n;i++){
                printf("\n ENTER THE ARRIVAL TIME AND SERVICE TIME OF PROCESS %d: ",i+1);
                scanf("%d%d",&arrv[i],&ser[i]);
        }
        for(i=0;i<n;i++){
                sum=0;
                for(j=0;j<i;j++)
                        sum=sum+ser[j];
                        start[i]=sum;
        }
        for(i=0;i<n;i++){
                finish[i]=ser[i]+start[i];
                wait[i]=start[i];
                turn[i]=ser[i]+wait[i];
        }

        for(i=0;i<n;i++){
                avgwait+=wait[i] ;
                avgturn+=turn[i];
        }
        avgwait/=n;
        avgturn/=n;

        printf("\n PROCESS ARRIVAL SERVICE START FINISH WAIT TURN \n");
        for(i=0;i<n;i++){
                printf("\n\tP%d\t%d \t %d \t %d \t %d \t %d \t %d \n",i ,arrv[i],
                ser[i], start[i], finish[i],wait[i],turn[i]);
        }
        printf("\n AVERAGE WAITING TIME = %f tu",avgwait);
        printf("\n AVERAGE TURN AROUND TIME = %f tu\n",avgturn);
}
```

```
duong@ubuntu: ~/Desktop/baitap/lab7/fcfs

 ENTER THE NO. OF PROCESSES:3

 ENTER THE ARRIVAL TIME AND SERVICE TIME OF PROCESS 1: 0
3

 ENTER THE ARRIVAL TIME AND SERVICE TIME OF PROCESS 2: 1
3

 ENTER THE ARRIVAL TIME AND SERVICE TIME OF PROCESS 3: 2
2

 PROCESS ARRIVAL SERVICE START FINISH WAIT TURN

        P0      0       3       0       3       0       3

        P1      1       3       3       6       3       6

        P2      2       2       6       8       6       8

 AVERAGE WAITING TIME = 3.000000 tu
 AVERAGE TURN AROUND TIME = 5.666667 tu
duong@ubuntu:~/Desktop/baitap/lab7/fcfs$
```

FCFS									
TIME	0	1	2	3	4	5	6	7	8
CPU	P0	P0	P0	P1	P1	P1	P2	P2	
WAIT	P0	P1	P1,P2	P2	P2	P2			

PROCESS	ARRIVAL	SERVICE	START	FINISH	WAIT	TURN
P0	0	3	0	3	0	3
P1	1	3	3	6	3	6
P2	2	2	6	8	6	8

sjf.c

```c
#include<stdio.h>
void main(){
        int n,i,j,temp1,temp2,sum=0;
        int pro[10],arrv[10],ser[10],start[10],finish[10],wait[10],turn[10];
        float avgturn=0.0,avgwait=0.0;
        start[0]=0;
        printf("\n ENTER THE NO. OF PROCESSES:");
        scanf("%d",&n);
        for(i=0;i<n;i++){
                printf("\n ENTER THE ARRIVAL TIME AND SERVICE TIME OF PROCESS %d:",i+1);
                scanf("%d%d",&arrv[i],&ser[i]);
                pro[i]=i;
        }
        for(i=0;i<n;i++){
                for(j=0;j<n-i-1;j++){//sap xem lai mang ser theo thu tu giam dan  de cai nao ngan nhat se dc thuc hien trc
                        if(ser[j]>ser[j+1]){
                                temp1=ser[j];
                                ser[j]=ser[j+1];
                                ser[j+1]=temp1;
                                temp2=arrv[j];//swap luon mang arrv
                                arrv[j]=arrv[j+1];
                                arrv[j+1]=temp2;
                        }
                }
        }
        for(i=0;i<n;i++){
                sum=0;
                for(j=0;j<i;j++)
                        sum=sum+ser[j];
                        start[i]=sum;
        }
        for(i=0;i<n;i++){
                finish[i]=ser[i]+start[i];
                wait[i]=start[i];
                turn[i]=ser[i]+wait[i];
        }
        for(i=0;i<n;i++){
                avgwait +=wait[i];
                avgturn +=turn[i];
        }
        avgwait/=n;
        avgturn/=n;
```

```
duong@ubuntu: ~/Desktop/baitap/lab7/sjf
duong@ubuntu:~/Desktop/baitap/lab7/sjf$ ./sjf.out

 ENTER THE NO. OF PROCESSES:3

 ENTER THE ARRIVAL TIME AND SERVICE TIME OF PROCESS 1:5
9

 ENTER THE ARRIVAL TIME AND SERVICE TIME OF PROCESS 2:2
4

 ENTER THE ARRIVAL TIME AND SERVICE TIME OF PROCESS 3:5
6

 PROCESS ARRIVAL SERVICE START FINISH WAIT TURN

        P0      2       4       0       4       0       4

        P1      5       6       4       10      4       10

        P2      5       9       10      19      10      19

 AVERAGE WAITING TIME = 4.666667 tu
 AVERAGE TURN AROUND TIME = 11.000000 tu
duong@ubuntu:~/Desktop/baitap/lab7/sjf$
```

```c
avgwait/=n;
avgturn/=n;
printf("\n PROCESS ARRIVAL SERVICE START FINISH WAIT TURN \n");
for(i=0;i<n;i++){
        printf("\n\tP%d\t%d \t %d \t %d \t %d \t %d \t %d \n",
        pro[i],arrv[i], ser[i], start[i], finish[i],wait[i],turn[i]);
}
printf("\n AVERAGE WAITING TIME = %f tu",avgwait);
printf("\n AVERAGE TURN AROUND TIME = %f tu\n" ,avgturn) ;
```

SJF

TIME	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CPU	P0	P0	P0	P0	P1	P1	P1	P1	P1	P1	P2	P2	P2	P2	P2	P2	P2	P2	P2	
WAIT	P0			P1,P2	P2	P2	P2	P2	P2	P2										

PROCESS	ARRIVAL	SERVICE	START	FINISH	WAIT	TURN
P0	2	4	0	4	0	4
P1	5	6	4	10	4	10
P2	5	9	10	19	10	19

srtf.c

```c
#include<stdio.h>

void main(){
        int n,i,j,temp1,temp2,sum=0;
        int pro[10],ser[10],start[10],finish[10],wait[10],prior[10], turn[10];
        float avgturn=0.0,avgwait=0.0;
        start[0]=0;
        printf("\n ENTER THE NO. OF PROCESSES:");
        scanf("%d",&n);
        for(i=0;i<n;i++){
                printf("\n ENTER THE SERVICE TIME AND PRIORITY OF PROCESS%d:",i+1);
                scanf("%d%d",&ser[i],&prior[i]);//mang prior mang tinh chat muc do uu tien cua tien trinh
                pro[i]=i;
        }
        for(i=0;i<n;i++){
                for(j=0;j<n-i-1;j++){
                        if(prior[j]>prior[j+1]){
                                temp1=ser[j];//swap service theo thu tu nho dan
                                ser[j]=ser[j+1];
                                ser[j+1]=temp1;

                                temp2=prior[j];//swap do uu tien theo service
                                prior[j]=prior[j+1];
                                prior[j+1]=temp2;
                        }
                }
        }
        for(i=0;i<n;i++){
                sum=0;
                for(j=0;j<i;j++)
                        sum=sum+ser[j];
                start[i]=sum;
        }
        for(i=0;i<n;i++){
                finish[i]=ser[i]+start[i];
                wait[i]=start[i];
                turn[i]=ser[i]+wait[i];
        }
        for(i=0;i<n;i++){
        avgwait+=wait[i] ;
        avgturn+=turn[i];
        }
        for(i=0;i<n;i++){
        avgwait+=wait[i] ;
        avgturn+=turn[i];
        }
        avgwait/=n;
        avgturn/=n;
        printf("\n PROCESS SERVICE PRIORITY START FINISH WAIT TURN \n");
        for(i=0;i<n;i++){
                printf("\n\tP%d\t%d \t %d \t %d \t %d \t %d \t %d \n", pro[i],ser[i], prior[i],
                start[i],finish[i],wait[i],turn[i]);
        }
        printf("\n AVERAGE WAITING TIME = %f tu",avgwait);
        printf("\n AVERAGE TURN AROUND TIME = %f tu \n", avgturn);
}
```

```
duong@ubuntu: ~/Desktop/baitap/lab7/srtf
duong@ubuntu:~/Desktop/baitap/lab7/srtf$ ./srtf.out

 ENTER THE NO. OF PROCESSES:3

 ENTER THE SERVICE TIME AND PRIORITY OF PROCESS1:4
2

 ENTER THE SERVICE TIME AND PRIORITY OF PROCESS2:1
5

 ENTER THE SERVICE TIME AND PRIORITY OF PROCESS3:3
4

 PROCESS SERVICE PRIORITY START FINISH WAIT TURN

        P0      4       2       0       4       0       4

        P1      3       4       4       7       4       7

        P2      1       5       7       8       7       8

 AVERAGE WAITING TIME = 3.666667 tu
 AVERAGE TURN AROUND TIME = 6.333333 tu
duong@ubuntu:~/Desktop/baitap/lab7/srtf$
```

SRTF									
TIME	0	1	2	3	4	5	6	7	8
CPU	P0	P0	P0	P0	P1	P1	P1	P2	

PROCESS	SERVICE	PRIORITY	START	FINISH	WAIT	TURN
P0	4	2	0	4	0	4
P1	3	4	4	7	4	7
P2	1	5	7	8	7	8

rr.c

```c
#include<stdio.h>

void main(){
        int count=0,swt=0,stat=0,i,temp,sq=0;
        int pro[10],st[10],bt[10],wt[10],tat[10],n,tq;
        float atat=0.0,awt=0.0;
/*swt:sum wait time
stat :sum turn around time
sq:sum quantum
st:service time
bt:burt time thgian can de hoan thanh tien trinh
wt:wait time thgian doi tung tien trinh
tat:turn around time thgian turn around tung tien trinh
tq:time quantum
*/
        printf("\n ENTER THE NO. OF PROCESSES:");
        scanf("%d",&n);
        for(i=0;i<n;i++){
                printf("\n ENTER THE SERVICE TIME OF PROCESS %d:",i);
                scanf("%d",&bt[i]);
                st[i]=bt[i];
                pro[i]=i;//luu so thu tu process
        }
        printf("\n ENTER THE TIME QUANTUM:");
        scanf("%d",&tq);
        while(1){//chay het vong for r chay nua den khi du process thi moi break trong for
                for(i=0,count=0;i<n;i++){
                        temp=tq;
                        if(st[i]==0){
                                count++;//dem so proces hoan thanh khi st[i]==0 nghia la het service time
                                continue;
                        }
                        if(st[i]>tq)
                                st[i]=st[i]-tq;
                        else if(st[i]>=0){
                                temp=st[i];
                                st[i]=0;
                        }
                        sq=sq+temp;//thgian chay +time quantum
                        tat[i]=sq;//;luu lai time aroud tung process
                }
                if(count==n)//chay du so process
                        break;
        }
        for(i=0;i<n;i++){
                wt[i]=tat[i]-bt[i];
                stat=stat+tat[i];
                swt=swt+wt[i];
        }
        awt=(float)swt/n;
        atat=(float)stat/n;
        printf("\n PROCESS BURST WAIT TURN \n");
        for(i=0;i<n;i++){
                printf("\n\tP%d\t%d \t %d \t%d\n" ,
                pro[i],bt[i],wt[i],tat[i]);
        }
        printf("\n AVERAGE WAITING TIME = %f tu",awt);
        printf("\n AVERAGE TURN AROUND TIME = %f tu",atat);
}
```

```
duong@ubuntu: ~/Desktop/baitap/lab7/roundrobin
duong@ubuntu:~/Desktop/baitap/lab7/roundrobin$ ./rr.out

 ENTER THE NO. OF PROCESSES:3

 ENTER THE SERVICE TIME OF PROCESS 0:30

 ENTER THE SERVICE TIME OF PROCESS 1:40

 ENTER THE SERVICE TIME OF PROCESS 2:20

 ENTER THE TIME QUANTUM:10

 PROCESS BURST WAIT TURN

        P0      30      40      70

        P1      40      50      90

        P2      20      40      60

 AVERAGE WAITING TIME = 43.333332 tu
 AVERAGE TURN AROUND TIME = 73.333336 tuduong@ubuntu:~/D
robin$
```

round robin									
time quantum	10	10	10	10	10	10	10	10	10
cpu	P0	P1	P2	P0	P1	P2	P0	P1	P1

PROCESS	BURST	WAIT	TURN
P0	30	40	70
P1	40	50	90
P2	20	40	60