OS PROJECT :

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>

struct process{
	int no;
	int bt1;
	int bt2;
	int at;
	int pri;
}p[100],q[100];
char name[10][100] = {"null","First Come First Serve","Shortest Job First(Preemptive)","Shotest Job first(Non-preemptive)","Round Robin ","Priority Scheduling(Preemptive)","Priority Scheduling(Non - preemptive)"};
int n,n1,x,ct[100],count = 0,c,rear = -1,front = -1,choice,tq;
struct process h[100];

//priority np
void build_heap(){
	int i;
	for(i=count/2;i>=0;i--)
		heapify(i);
}


void heapify(int i){
	int left,right,smallest;
	left = 2*i+1;
	right = 2*i+2;

	if(left < count && h[i].pri > h[left].pri)
		smallest = left;
	else
		smallest = i;

	if(right < count && h[smallest].pri > h[right].pri)
		smallest = right;

	if(smallest != i){
		struct process temp = h[i];
		h[i] = h[smallest];
		h[smallest] = temp;
		heapify(smallest);
	}
}


struct process min(){
		struct process temp = h[0];
		h[0] = h[count-1];
```

```
                h[count-1] = temp;
                count = count - 1;
                build_heap();
                return temp;
}


void prioritynp(){
        int t=0,t0,i;
        struct process temp;
        while(x>0){
                if(count > 0){
                        temp = min();
                        t0 = t;
                        x = x - temp.bt1;
                        t = t + temp.bt1;
                        ct[temp.no] = t;
                        temp.bt1 = 0;
                        printf("Process %d exicuted for %d unit of time.\n-->Process %d
completes its exicution.\n",temp.no + 1,temp.bt2,temp.no+1);
                        sleep(1);
                }

                for(i=0;i<n1;i++){
                        if(t == 0){
                                if(p[i].at == 0){
                                        h[count] = p[i];
                                        count = count + 1;
                                        build_heap();
                                }
                        }
                        else if(p[i].at > t0 && p[i].at <= t){
                                h[count] = p[i];
                                count = count + 1;
                                build_heap();
                        }
                }

                if(count == 0){
                        printf("CPU is idle for 1 unit of time.\n");
                        sleep(1);
                        t0 = t;
                        t = t + 1;
                }
        }
        print();
}

//priority p
void priorityp(){
        int t=0,t0,i;
```

```
        struct process temp;
        while(x>0){
                if(count > 0){
                        temp = min();
                        t0 = t;
                        x = x - 1;
                        t = t + 1;
                        temp.bt1 = temp.bt1 - 1;
                        printf("Process %d exicuted for 1 unit of time.\n",temp.no + 1);
                        sleep(1);

                        if(temp.bt1 == 0){
                                printf("-->Process %d complete its exicution.\n",temp.no+1);
                                ct[temp.no] = t;
                        }
                        else{
                                h[count] = temp;
                                count = count + 1;
                                build_heap();
                        }
                }

                for(i=0;i<n1;i++){
                        if(t == 0){
                                if(p[i].at == 0){
                                        h[count] = p[i];
                                        count = count + 1;
                                        build_heap();
                                }
                        }
                        else if(p[i].at > t0 && p[i].at <= t){
                                h[count] = p[i];
                                count = count + 1;
                                build_heap();
                        }
                }

                if(count == 0){
                        printf("CPU is idle for 1 unit of time.\n");
                        sleep(1);
                        t0 = t;
                        t = t + 1;
                }
        }
        print();
}

//RR
void enqueue(struct process p){
        if(count == n){
                printf("Queue_Overflow\n");
```

```
                return;
        }
        rear = (rear + 1)%n;
        q[rear] = p;
        count = count + 1;
}

struct process dequeue(){
        if(count == 0){
                printf("Queue_underflow\n");
                return;
        }
        front = (front + 1) % n;
        struct process temp = q[front];
        count = count - 1;
        return temp;
}


void roundRobin(){
        int t=0,t0,k,i;
        struct process temp;
        temp.bt1 = 0;
        while(x>0){
                if(count > 0){
                        temp = dequeue();
                        if(temp.bt1 >= tq){
                                temp.bt1 = temp.bt1 - tq;
                                t0 = t;
                                t = t + tq;
                                x = x - tq;
                                printf("Process %d executes for %d unit of time.\n",temp.no+1,t-
t0);
                                sleep(1);
                                if(temp.bt1 == 0){
                                        ct[temp.no] = t;
                                        printf("-->Process %d completes its execution.\
n",temp.no+1);
                                        sleep(1);
                                }
                        }
                        else{
                                k = temp.bt1;
                                temp.bt1 = temp.bt1 - k;
                                t0 = t;
                                t = t + k;
                                x = x - k;
                                ct[temp.no] = t;
                                printf("Process %d executes for %d unit of time.\n",temp.no+1,t-
t0);
                                sleep(1);
```

```
                        printf("-->Process %d completes its execution.\n",temp.no+1);
                        sleep(1);
                    }
                }

                for(i = 0;i < n;i++){
                    if(t == 0){
                        if(p[i].at == 0){
                            enqueue(p[i]);
                        }
                    }
                    else if(p[i].at <= t && p[i].at > t0)
                        enqueue(p[i]);
                }
                if(temp.bt1 != 0)
                    enqueue(temp);

                if(count == 0){
                    t0 = t;
                    t = t + 1;
                    printf("CPU is idle for 1 unit of time.\n");
                    sleep(1);
                }
            }
            print();
}

//fcfs
void fcfs(){
            int t,t1=0,i;
            struct process temp;
            while(x > 0){
                if(count > 0){
                   temp = dequeue();
                   x = x - temp.bt1;
                   t = t1;
                        t1 = t1 + temp.bt1;
                        ct[temp.no] = t1;
                        printf("Process %d exicutes for %d units of time. \n",temp.no+1,temp.bt1);
                        sleep(1);
                        printf("-->Process %d completes its exicution.\n",temp.no+1);
                        sleep(1);
                }
                for(i=0;i<n;i++){
                    if(t1 == 0){
                        if(p[i].at == 0)
                            enqueue(p[i]);
                    }
                    else if(p[i].at <= t1 && p[i].at > t){
                        enqueue(p[i]);
                    }
```

```
                }
                if(count == 0){
                        t = t1;
                        t1 = t1 + 1;
                        printf("CPU is idle for 1 unit time.\n");
                        sleep(1);
                }
        }
        print();
}

//sjfp
struct process h[100];


void build_heap1(){
        int i;
        for(i=count/2;i>=0;i--)
                heapify1(i);
}


void heapify1(int i){
        int left,right,smallest;
        left = 2*i+1;
        right = 2*i+2;

        if(left < count && h[i].bt1 > h[left].bt1)
                smallest = left;
        else
                smallest = i;

        if(right < count && h[smallest].bt1 > h[right].bt1)
                smallest = right;

        if(smallest != i){
                struct process temp = h[i];
                h[i] = h[smallest];
                h[smallest] = temp;
                heapify1(smallest);
        }
}


struct process min1(){
                struct process temp = h[0];
                h[0] = h[count-1];
                h[count-1] = temp;
                count = count - 1;
                build_heap1();
                return temp;
```

```
}


void sjf(){
	int t=0,t0,i;
	struct process temp;
	while(x>0){
		if(count > 0){
			temp = min1();
			t0 = t;
			x = x - 1;
			t = t + 1;
			temp.bt1 = temp.bt1 - 1;
			printf("Process %d exicuted for 1 unit of time.\n",temp.no + 1);
			sleep(1);

			if(temp.bt1 == 0){
				printf("-->Process %d complete its exicution.\n",temp.no+1);
				ct[temp.no] = t;
			}
			else{
				h[count] = temp;
				count = count + 1;
				build_heap1();
			}
		}

		for(i=0;i<n1;i++){
			if(t == 0){
				if(p[i].at == 0){
					h[count] = p[i];
					count = count + 1;
					build_heap1();
				}
			}
			else if(p[i].at > t0 && p[i].at <= t){
				h[count] = p[i];
				count = count + 1;
				build_heap1();
			}
		}

		if(count == 0){
			printf("CPU is idle for 1 unit of time.\n");
			sleep(1);
			t0 = t;
			t = t + 1;
		}
	}
	print();
}
```

```
//sjf np
void sjfnp(){
	int t=0,t0,i;
	struct process temp;
	while(x>0){
		if(count > 0){
			temp = min1();
			t0 = t;
			x = x - temp.bt1;
			t = t + temp.bt1;
			ct[temp.no] = t;
			temp.bt1 = 0;
			printf("Process %d exicuted for %d unit of time.\n-->Process %d
completes its exicution.\n",temp.no + 1,temp.bt2,temp.no+1);
			sleep(1);
		}

		for(i=0;i<n1;i++){
			if(t == 0){
				if(p[i].at == 0){
					h[count] = p[i];
					count = count + 1;
					build_heap();
				}
			}
			else if(p[i].at > t0 && p[i].at <= t){
				h[count] = p[i];
				count = count + 1;
				build_heap();
			}
		}

		if(count == 0){
			printf("CPU is idle for 1 unit of time.\n");
			sleep(1);
			t0 = t;
			t = t + 1;
		}
	}
	print();
}

void initialization(){
	int i,c;
	printf("Hello!\n");
	sleep(1);
	printf("Welcome to %s calculations.\n",name[choice]);
	sleep(1);
	printf("This program will calculate Turn Around Time and Waiting Time.\n\n");
	sleep(1);
```

```
printf("Enter the number of processes : ");
scanf("%d",&n);
n1 = n;
printf("Enter the burst time and the arrival time of the process : \n");
for(i = 0;i < n;i++){
        printf("Process %d\n",i+1);
        printf(">>Arrival time : ");
        scanf("%d",&p[i].at);
        printf(">>Burst time : ");
        scanf("%d",&p[i].bt1);
        if(choice >= 5){
        printf(">>Priority : ");
        scanf("%d",&p[i].pri);
        }
        printf("\n");
        x = x + p[i].bt1;
        p[i].bt2 = p[i].bt1;
        p[i].no = i;
}
printf("The total burst time = %d\n",x);
if(choice == 4){
        printf("Enter the time quantum : ");
        scanf("%d",&tq);
}

printf("\n\nCheck the following information is correct or not.\n");
if(choice >= 5){
        printf("\n\nCheck the following information is correct or not.\n");
        printf("process\t\tPriority\t\tArrival time\t\tBurst time\n");
        for(i=0;i<n;i++){
                printf("P%d\t\t%d\t\t\t%d\t\t\t%d\n",i+1,p[i].pri,p[i].at,p[i].bt1);
        }
}
else{
        printf("process\t\tArrival time\t\tBurst time\n");
        for(i=0;i<n;i++){
                printf("P%d\t\t%d\t\t\t%d\n",i+1,p[i].at,p[i].bt1);
        }
}

if(choice == 4)
        printf("The time quantum is : %d\n",tq);

read:
sleep(1);
printf("\n\nChoose the option\n1.correct\n2.Incorrect\n");
scanf("%d",&c);
if(c==1)
        printf("***Thank You For Confirmation***\n\n");
else if(c==2)
        return initialization();
```

```
	else{
		printf("Invalid input!\n");
		goto read;
	}
}

void print(){
	sleep(1);
	printf("\nThe final result is..\n\n");
	sleep(1);
	int i,tat = 0,wt = 0;
	float atat,awt;
	printf("Process\t\tAT\tBT\tCT\tTAT\tWT\n");
	for(i = 0;i < n;i++){
		printf("P%d\t\t%d\t%d\t%d\t%d\t%d\n",i+1,p[i].at,p[i].bt2,ct[i],ct[i]-p[i].at,(ct[i]-
p[i].at)-p[i].bt2);
		tat = tat + (ct[i]-p[i].at);
		wt = wt + (ct[i]-p[i].at)-p[i].bt2;
	}
	printf("\nHere,\nAT -> Arrival time.\nBT -> Burst Time.\nCT -> Completion Time.\nTAT
-> Turn Around Time.\nWT -> Waiting Time\n");
	atat = (float)tat/n;
	awt = (float)wt/n;
	sleep(1);
	printf("\n\nTotal Turn Around Time is %d\n",tat);
	printf("Total Waiting Time is %d\n",wt);
	printf("Average Turn Around Time is %f\n",atat);
	printf("Average Waiting Time is %f\n",awt);
}

int main(){
	printf("Enter your choice.\n1.First Come First Serve.\n2.Shortest Job First(Preemptive).\
n3.Shotest Job first(Non-preemptive).\n4.Round Robin Algorithm.\n5.Priority
Scheduling(Preemptive).\n6.Priority Scheduling(Non - preemptive).\n");

	scanf("%d",&choice);
	switch(choice){
		case 1:
			initialization();
			fcfs();
			break;
		case 2:
			initialization();
			sjf();
			break;
		case 3:
			initialization();
			sjfnp();
			break;
		case 4:
			initialization();
```

```
                roundRobin();
                break;
        case 5:
                initialization();
                priorityp();
                break;
        case 6:
                initialization();
                prioritynp();
                break;
        default:
                printf("Enter valid input!\n");
                return main();
    }
    read:
    printf("\n\nEnter Your Choice : \n1.I want to contiue .\n2.exit.\n");
    int l;
    scanf("%d",&l);
    switch(l){
        case 1:
                return main();
                break;
        case 2:
                printf("Exitting..\n");
                exit(0);
                break;
        default:
                printf("Invalid Input.\n");
                goto read;
    }
}
```