

**Subject:** Operating System

**Innovative Assignment**

**Group Member:**

21BCE164 – Jiya Nayak

22BCE516 – Riya Mehta

**Methodology Followed:**

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

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

```
#include<conio.h>
```

```
#include<string.h>
```

```
struct node{  
    char n[10];  
    int arr_time;  
    int burst_time;  
    int priority;  
    int final_time;  
    int turnarr_time;  
    int waiting_time;  
    int response_time;  
    int c;  
    struct node *next;  
};
```

```
struct Node{
```

```

    char n[10];
    struct Node *next;
};

struct queue{
    char n[10];
    int arr_time;
    int burst_time;
    int priority;
    int final_time;
    int turnarr_time;
    int waiting_time;
    int response_time;
    struct queue *next;
};

struct Queue{
    struct queue *front,*rear;
};

void readFile(struct node **l){
    FILE *fp;
    fp=fopen("Process.txt","r");
    struct node *t;
    char a[10];
    int b,c,d;
    while((fscanf(fp,"%s %d %d %d\n",a,&b,&c,&d))!=EOF){

```

```

    t=(struct node*)malloc(sizeof(struct node));
    strcpy(t->n,a);
    t->arr_time=b;
    t->burst_time=c;
    t->priority=d;
    t->final_time=t->response_time=t->turnarr_time=t-
>waiting_time=t->c=0;
    t->next=NULL;
    if((*l)==NULL){
        (*l)=t;
    }
    else{
        struct node *r>(*l);
        while(r->next!=NULL){
            r=r->next;
        }
        r->next=t;
    }
}

int len(struct node *l){
    int len=0;
    while(l!=NULL){
        l=l->next;
        len++;
    }
}

```

```
    }  
    return len;  
}
```

```
void swap(struct node *p,struct node *q){  
    char a[10];  
    strcpy(a,p->n);  
    int b=p->arr_time,c=p->burst_time,d=p->priority;  
    strcpy(p->n,q->n);  
    p->arr_time=q->arr_time,p->burst_time=q->burst_time,p->  
>priority=q->priority;  
    strcpy(q->n,a);  
    q->arr_time=b,q->burst_time=c,q->priority=d;  
}
```

```
void sort_Arrival(struct node **l){  
    struct node *p>(*l),*q,*r;  
    while(p!=NULL){  
        q>(*l);  
        while(q!=NULL){  
            if(p->arr_time<q->arr_time){  
                swap(p,q);  
            }  
            q=q->next;  
        }  
        p=p->next;
```

```

    }
}

void printDetails(struct node *l){
    printf("Process Arrival Burst Final Priority Turnarr Waiting\n");
    while(l!=NULL){
        printf("%s %7d %7d %7d %7d %7d %7d\n",l->n,l->arr_time,l-
>burst_time,l->final_time,l->priority,l->turnarr_time,l-
>waiting_time);
        l=l->next;
    }
}

```

```

void printGantt(struct Node *g){
    printf("Here is your Gantt Chart....\n");
    while(g!=NULL){
        printf("%s ",g->n);
        g=g->next;
    }
}

```

```

void clearData(struct node **l,struct Node **g){
    struct node *p=(*l),*u;
    u=p;
    while(p->next!=NULL){
        u=p;
        p=p->next;
    }
}

```

```

    free(u);
}
free(p);
(*l)=NULL;
struct Node *w=(*g),*t;
while(w->next!=NULL){
    t=w;
    w=w->next;
    free(t);
}
free(w);
(*g)=NULL;
}

```

```

void add_node(struct Node **g,char *s){
    struct Node *v;
    v=(struct Node*)malloc(sizeof(struct Node));
    v->next=NULL;
    strcpy(v->n,s);
    if((*g)==NULL){
        (*g)=v;
    }
    else{
        struct Node *u=(*g);
        while(u->next!=NULL){

```

```

        u=u->next;
    }
    u->next=v;
}
}

void fcfs(struct node **l,struct Node **g){
    sort_Arrival(&(*l));
    // printDetails((*l));
    struct node *p=(*l);
    int c=0,f=0;
    float avg_t=0,avg_w=0;
    while(p!=NULL){
        int s=p->burst_time;
        c=0;
        int d=f-p->arr_time;
        struct Node *v;
        if(d<0){
            while(d<0){
                add_node(&(*g),"_");
                d++;
                f++;
            }
        }
        for(int i=0;i<s;i++){
            f++;

```

```

    c++;
    v=(struct Node*)malloc(sizeof(struct Node));
    v->next=NULL;
    char a[10];
    strcpy(a,p->n);
    add_node(&(*g),a);
}
p->final_time=f;
p->turnarr_time=p->final_time-p->arr_time;
avg_t+=p->turnarr_time;
p->waiting_time=p->turnarr_time-p->burst_time;
avg_w+=p->waiting_time;
p=p->next;
}
printf("First Come First Served: \n\n");
printDetails((*l));
printGantt((*g));
float x=len(*l);
avg_t=(float)(avg_t)/x;
avg_w=(float)(avg_w)/x;
printf("\nAverage turn_around Time = %f\nAverage waiting
Time = %f\n",avg_t,avg_w);
printf("\n");
}
struct node *take(struct node **k){

```



```

    if((*k)==NULL){
        return NULL;
    }
    struct node *t=(*k);
    (*k)=(*k)->next;
    return t;
}

void sort_burst(struct node **l){
    struct node *p=(*l),*q,*r;
    while(p!=NULL){
        q=(*l);
        while(q!=NULL){
            if(p->burst_time<q->burst_time){
                swap(p,q);
            }
            q=q->next;
        }
        p=p->next;
    }
}

void add_list(struct node **l,struct node **k,int f){
    struct node *t=(*l),*r;
    while(t!=NULL){
        if(t->arr_time<=f && t->c==0){
            t->c=1;

```

```

    r=(struct node*)malloc(sizeof(struct node));
    strcpy(r->n,t->n);
    r->arr_time=t->arr_time;
    r->burst_time=t->burst_time;
    r->priority=t->priority;
    r->next=NULL;
    if((*k)==NULL){
        (*k)=r;
    }
    else{
        struct node *u=(*k);
        while(u->next!=NULL){
            u=u->next;
        }
        u->next=r;
    }
}

t=t->next;
}

sort_burst(&(*k));
}

void sjf(struct node **l,struct Node **g){
    struct node *k=NULL;
    add_list(&(*l),&k,0);
    struct node *p;

```

```

int f=0,c=0,n=len(*l),m=0;
float avg_t=0,avg_w=0;
while(n!=m){
    p=take(&(k));
    while(p==NULL){
        f++;
        add_list(&(*l),&k,f);
        p=take(&(k));
    }
    int s=p->burst_time;
    c=0;
    int d=f-p->arr_time;
    struct Node *v;
    if(d<0){
        while(d<0){
            add_node(&(*g),"_");
            d++;
            f++;
        }
    }

    for(int i=0;i<s;i++){
        f++;
        c++;
        v=(struct Node*)malloc(sizeof(struct Node));

```

```

        v->next=NULL;
        char a[10];
        strcpy(a,p->n);
        add_node(&(*g),a);
    }
    p->final_time=f;
    p->turnarr_time=p->final_time-p->arr_time;
    avg_t+=p->turnarr_time;
    p->waiting_time=p->turnarr_time-p->burst_time;
    avg_w+=p->waiting_time;
    p->c=1;
    m++;
    add_list(&(*l),&k,f);
}

printf("Shortest Job First: \n\n");
printDetails(*l);
printGantt(*g);


float x=len(*l);
avg_t=(float)(avg_t)/x;
avg_w=(float)(avg_w)/x;

printf("\nAverage turn_around Time = %f\nAverage waiting
Time = %f\n",avg_t,avg_w);

printf("\n");
}

```

```

struct queue *create(char *a,int b,int c,int d){
    struct queue *t=(struct queue*)malloc(sizeof(struct queue));
    strcpy(t->n,a);
    t->arr_time=b,t->burst_time=c,t->priority=d;
    t->next=NULL;
    return t;
}

void add_queue(struct node **l,struct Queue **q,int f){
    struct node *p>(*l);
    while(p!=NULL){
        if(p->c==0 && p->arr_time<=f){
            p->c=1;
            struct queue *t=create(p->n,p->arr_time,p->burst_time,p->priority);
            if((*q)->rear==NULL){
                (*q)->rear=t;
                (*q)->front=t;
            }
            else{
                (*q)->rear->next=t;
                (*q)->rear=t;
            }
        }
        p=p->next;
    }
}

```

```
}
```

```
struct queue *pop(struct Queue **q){  
    if((*q)->front==NULL){  
        return NULL;  
    }  
    struct queue *t=(*q)->front;  
    (*q)->front=(*q)->front->next;  
    if((*q)->front==NULL) (*q)->rear=NULL;  
    return t;  
}
```

```
void push(struct Queue **q,struct queue *t){  
    if((*q)->rear==NULL){  
        (*q)->front=(*q)->rear=t;  
    }  
    (*q)->rear->next=t;  
    (*q)->rear=t;  
}
```

```
void r_r(struct node **l,struct Node **g){  
    struct Queue *q=(struct Queue*)malloc(sizeof(struct Queue));  
    struct node *l1=NULL,*r,*z;  
    (q)->rear=NULL;  
    (q)->front=NULL;  
    add_queue(&(*l),&(q),0);
```

```

int tq,oh;
float avg_t=0,avg_w=0;
printf("Enter the time qaunta: ");
scanf("%d",&tq);
printf("Enter the switch overhead: ");
scanf("%d",&oh);
int e=len(*l),y=0;
int tt=0;
while(e!=y){
    add_queue(&(*l),&q,tt);
    // printf("W\n");
    while((q)->front==NULL){
        tt++;
        add_node(&(*g),"_");
        add_queue(&(*l),&(q),tt);
    }

    struct queue *t=pop(&(q));
    int d=t->burst_time;
    for(int i=1;i<=tq;i++){
        d-=1;
        add_node(&(*g),t->n);
        t->burst_time-=1;
        tt++;
        if(d<=0){

```

```

        break;
    }
}

if(d==0){
    r=(struct node*)malloc(sizeof(struct node));
    strcpy(r->n,t->n);
    r->arr_time=t->arr_time;
    r->burst_time=t->burst_time;
    r->final_time=tt;
    r->turnarr_time=tt-r->arr_time;
    r->waiting_time=r->turnarr_time-r->burst_time;
    r->priority=t->priority;
    r->next=NULL;
    avg_t+=r->turnarr_time;
    avg_w+=r->waiting_time;
    if(l1==NULL){
        l1=r;
        z=l1;
    }
    else{
        z->next=r;
        z=r;
    }
    y++;
}

```



```

else{
    for(int i=0;i<oh;i++){
        add_node(&(*g),"O");
    }
    tt+=oh;
    push(&q,t);
}
}

```

```

printf("Round Robbin: \n\n");
printDetails(l1);
printGantt(*g);

```

```

float x=len(*l);
avg_t=(float)(avg_t)/x;
avg_w=(float)(avg_w)/x;
printf("\nAverage turn_around Time = %f\nAverage waiting
Time = %f\n",avg_t,avg_w);
printf("\n");
}

```

```

void sort_priority(struct node **l,char *pr){
    struct node *p=(*l),*q,*r;
    while(p!=NULL){
        q=(*l);
        while(q!=NULL){

```

```

        if(strcmp(pr,"H")==0){
            if(p->priority>q->priority){
                swap(p,q);
            }
        }
        else if(strcmp(pr,"L")==0){
            if(p->priority<q->priority){
                swap(p,q);
            }
        }
        q=q->next;
    }
    p=p->next;
}

void add_list_p(struct node **l,struct node **k,int f,char *pr){
    struct node *t>(*l),*r;
    while(t!=NULL){
        if(t->arr_time<=f && t->c==0){
            t->c=1;
            r=(struct node*)malloc(sizeof(struct node));
            strcpy(r->n,t->n);
            r->arr_time=t->arr_time;
            r->burst_time=t->burst_time;
            r->priority=t->priority;

```

```

        r->next=NULL;
        if((*k)==NULL){
            (*k)=r;
        }
        else{
            struct node *u=(*k);
            while(u->next!=NULL){
                u=u->next;
            }
            u->next=r;
        }
    }
    t=t->next;
}
sort_priority(&(*k),pr);
}

void priority(struct node **l,struct Node **g){
    struct node *k=NULL,*p,*r,*z,*l1=NULL;
    char c[2];

    printf("Enter the value L (0 as the low priority) and H(Max value as
high priority): ");

    scanf("%s",c);

    add_list_p(&(*l),&k,0,c);

    int f=0,n=len(*l),m=0;

    float avg_t=0,avg_w=0;

```

```

while(n!=m){
    p=take(&(k));
    while(p==NULL){
        f++;
        add_node(&(*g),"_");
        add_list_p(&(*l),&k,f,c);
        p=take(&(k));
    }
    int s=p->burst_time;
    int d=f-p->arr_time;
    struct Node *v;
    if(d<0){
        while(d<0){
            add_node(&(*g),"_");
            d++;
            f++;
        }
    }

    for(int i=0;i<s;i++){
        f++;
        v=(struct Node*)malloc(sizeof(struct Node));
        v->next=NULL;
        char a[10];
        strcpy(a,p->n);
    }
}

```

```

    add_node(&(*g),a);
}

r=(struct node*)malloc(sizeof(struct node));
strcpy(r->n,p->n);
r->arr_time=p->arr_time;
r->burst_time=p->burst_time;
r->final_time=f;
r->turnarr_time=f-r->arr_time;
r->waiting_time=r->turnarr_time-r->burst_time;
r->priority=p->priority;
r->next=NULL;
avg_t+=r->turnarr_time;
avg_w+=r->waiting_time;
if(l1==NULL){
    l1=r;
    z=l1;
}
else{
    z->next=r;
    z=r;
}
p->c=1;
m++;
add_list_p(&(*l),&k,f,c);
}

```

```

printf("Priority Sheduling: \n\n");
printDetails(l1);
printGantt(*g);

float x=len(*l);
avg_t=(float)(avg_t)/x;
avg_w=(float)(avg_w)/x;
printf("\nAverage turn_around Time = %f\nAverage waiting
Time = %f\n",avg_t,avg_w);
printf("\n");
}

struct node *give(struct node **k){
    if((*k)==NULL){
        return NULL;
    }
    struct node *t=(*k);
    return t;
}

int burst(struct node **l,char *s){
    struct node *k=(*l);
    while(k!=NULL){
        if(strcmp(k->n,s)==0){
            return k->burst_time;
        }
        k=k->next;
    }
}

```

```

    }
}

void srlf(struct node **l,struct Node **g){
    struct node *k=NULL,*r,*v,*ll=NULL,*z;
    add_list(&(*l),&k,0);
    int f=0,n=len(*l),m=0;
    float avg_t=0,avg_w=0,x=len(*l);
    while(n!=m){
        struct node *p=give(&k);
        while(p==NULL){
            f++;
            add_list(&(*l),&k,f);
            add_node(&(*g),"_");
            p=give(&k));
        }
        f++;
        p->burst_time-=1;
        add_node(&(*g),p->n);
        if(p->burst_time==0){
            m++;
            r=take(&k);
            r=(struct node*)malloc(sizeof(struct node));
            strcpy(r->n,p->n);
            r->burst_time=burst(&(*l),r->n);
            r->arr_time=p->arr_time;

```

```

    r->final_time=f;
    r->turnarr_time=f-p->arr_time;
    r->priority=p->priority;
    r->next=NULL;
    r->waiting_time=r->turnarr_time-r->burst_time;
    avg_t+=r->turnarr_time;
    avg_w+=r->waiting_time;
    if(l1==NULL){
        l1=r;
        z=l1;
    }
    else{
        z->next=r;
        z=r;
    }
}
add_list(&(*l),&k,f);
}

printf("Shortest Remaining Time First: \n\n");
printDetails(l1);
printGantt(*g);

avg_t=(avg_t)/x;
avg_w=(avg_w)/x;

printf("\nAverage turn_around Time = %f\nAverage waiting
Time = %f\n",avg_t,avg_w);

```



```

    printf("\n");
}

void sort_burst_l(struct node **l){
    struct node *p>(*l),*q,*r;
    while(p!=NULL){
        q>(*l);
        while(q!=NULL){
            if(p->burst_time>q->burst_time){
                swap(p,q);
            }
            q=q->next;
        }
        p=p->next;
    }
}

void add_list_l(struct node **l,struct node **k,int f){
    struct node *t>(*l),*r;
    while(t!=NULL){
        if(t->arr_time<=f && t->c==0){
            t->c=1;
            r=(struct node*)malloc(sizeof(struct node));
            strcpy(r->n,t->n);
            r->arr_time=t->arr_time;
            r->burst_time=t->burst_time;
            r->priority=t->priority;

```

```

        r->next=NULL;
        if((*k)==NULL){
            (*k)=r;
        }
        else{
            struct node *u>(*k);
            while(u->next!=NULL){
                u=u->next;
            }
            u->next=r;
        }
    }
    t=t->next;
}
sort_burst_l(&(*k));
}

void lrtf(struct node **l,struct Node **g){
    struct node *k=NULL,*r,*v,*l1=NULL,*z;
    add_list_l(&(*l),&k,0);
    int f=0,n=len(*l),m=0;
    float avg_t=0,avg_w=0,x=len(*l);
    struct node *p=give(&k);
    while(n!=m){
        while(p==NULL){
            f++;

```

```

    add_list_l(&(*l),&k,f);
    add_node(&(*g),"_");
    p=give(&k);
}
f++;
p->burst_time-=1;
add_node(&(*g),p->n);
if(p->burst_time==0){
    m++;
    r=take(&k);
    r=(struct node*)malloc(sizeof(struct node));
    strcpy(r->n,p->n);
    r->burst_time=burst(&(*l),r->n);
    r->arr_time=p->arr_time;
    r->final_time=f;
    r->turnarr_time=f-p->arr_time;
    r->priority=p->priority;
    r->next=NULL;
    r->waiting_time=r->turnarr_time-r->burst_time;
    avg_t+=r->turnarr_time;
    avg_w+=r->waiting_time;
    if(l1==NULL){
        l1=r;
        z=l1;
    }
}

```

```

        else{
            z->next=r;

            z=r;
        }
    }
    add_list_l(&(*l),&k,f);

    p=k;
}

printf("Longest Remaining Time First: \n\n");
printDetails(l1);
printGantt(*g);

avg_t=(avg_t)/x;
avg_w=(avg_w)/x;

printf("\nAverage turn_around Time = %f\nAverage waiting
Time = %f\n",avg_t,avg_w);

printf("\n");
}

int main()
{
    struct node *l=NULL;
    struct Node *g=NULL;


printf("\n\n=====
=====\\n\\n");

```

```
printf("          Simulator of Sheduling Algorithms\n\n");
```

```
printf("\n\n=====
=====\\n\\n");
```

```
readFile(&l);
```

```
int ch=0;
```

```
while(ch!=7){
```

```
    printf("1. First Come First Searved\\n");
```

```
    printf("2. Shortest Job First\\n");
```

```
    printf("3. Round Robbin\\n");
```

```
    printf("4. Priority Sheduling\\n");
```

```
    printf("5. Shortest Remaining Time First\\n");
```

```
    printf("6. Longest Remaining Time First\\n");
```

```
    printf("7. Exit\\n\\n");
```

```
    printf("Press: ");
```

```
    scanf("%d",&ch);
```

```
    switch (ch)
```

```
    {
```

```
        case 1:
```

```
            fcfs(&l,&g);
```

```
            clearData(&l,&g);
```

```
            readFile(&l);
```

```
            break;
```

```
        case 2:
```

```
sjf(&l,&g);  
clearData(&l,&g);  
readFile(&l);  
    break;
```

```
case 3:  
r_r(&l,&g);  
clearData(&l,&g);  
readFile(&l);  
    break;
```

```
case 4:  
priority(&l,&g);  
clearData(&l,&g);  
readFile(&l);  
    break;
```

```
case 5:  
srtf(&l,&g);  
clearData(&l,&g);  
readFile(&l);  
    break;
```

```
case 6:  
lrtf(&l,&g);
```

```

clearData(&l,&g);
readFile(&l);
    break;

case 7:

printf("\n\n=====
=====\\n\\n");
    printf("          THANK YOU..:)          ");

printf("\n\n=====
=====\\n\\n");
    break;
default:
    printf("Please press valid button..!!!\\n\\n");
    break;
}
}
}

```

## Output:

---

```

P1 0 3 0
P2 1 4 0
P3 0 5 5
P4 3 2 4

```

```
=====
Simulator of Sheduling Algorithms
=====
```

1. First Come First Searved
2. Shortest Job First
3. Round Robbin
4. Priority Sheduling
5. Shortest Remaining Time First
6. Longest Remaining Time First
7. Exit

Press: 1

First Come First Served:

| Process | Arrival | Burst | Final | Priority | Turnarr | Waiting |
|---------|---------|-------|-------|----------|---------|---------|
| P1      | 0       | 3     | 3     | 0        | 3       | 0       |
| P3      | 0       | 5     | 8     | 5        | 8       | 3       |
| P2      | 1       | 4     | 12    | 0        | 11      | 7       |
| P4      | 3       | 2     | 14    | 4        | 11      | 9       |

Here is your Gantt Chart....

P1 P1 P1 P3 P3 P3 P3 P3 P2 P2 P2 P2 P4 P4

Average turn\_around Time = 8.250000

Average waiting Time = 4.750000

1. First Come First Searved
2. Shortest Job First
3. Round Robbin
4. Priority Sheduling
5. Shortest Remaining Time First
6. Longest Remaining Time First
7. Exit



Press: 2

Shortest Job First:

| Process | Arrival | Burst | Final | Priority | Turnarr | Waiting |
|---------|---------|-------|-------|----------|---------|---------|
| P1      | 0       | 3     | 0     | 0        | 0       | 0       |
| P2      | 1       | 4     | 0     | 0        | 0       | 0       |
| P3      | 0       | 5     | 0     | 5        | 0       | 0       |
| P4      | 3       | 2     | 0     | 4        | 0       | 0       |

Here is your Gantt Chart....

P1 P1 P1 P4 P4 P2 P2 P2 P2 P3 P3 P3 P3 P3

Average turn\_around Time = 6.750000

Average waiting Time = 3.250000

1. First Come First Searved
2. Shortest Job First
3. Round Robbin
4. Priority Sheduling
5. Shortest Remaining Time First
6. Longest Remaining Time First
7. Exit

Press: 3

Enter the time qaunta: 2

Enter the switch overhead: 3

Round Robbin:

| Process | Arrival | Burst | Final | Priority | Turnarr | Waiting |
|---------|---------|-------|-------|----------|---------|---------|
| P1      | 0       | 0     | 11    | 0        | 11      | 11      |
| P4      | 3       | 0     | 18    | 4        | 15      | 15      |
| P2      | 1       | 0     | 25    | 0        | 24      | 24      |
| P3      | 0       | 0     | 26    | 5        | 26      | 26      |

Here is your Gantt Chart....

P1 P1 0 0 0 P3 P3 0 0 0 P1 P2 P2 0 0 0 P4 P4 P3 P3 0 0 0 P2 P2 P3

Average turn\_around Time = 19.000000

Average waiting Time = 19.000000

1. First Come First Searved
2. Shortest Job First
3. Round Robbin
4. Priority Sheduling
5. Shortest Remaining Time First
6. Longest Remaining Time First
7. Exit

Press: 4  
 Enter the value L (0 as the low priority) and H(Max value as high priority): H  
 Priority Sheduling:

| Process | Arrival | Burst | Final | Priority | Turnarr | Waiting |
|---------|---------|-------|-------|----------|---------|---------|
| P3      | 0       | 5     | 5     | 5        | 5       | 0       |
| P4      | 3       | 2     | 7     | 4        | 4       | 2       |
| P2      | 1       | 4     | 11    | 0        | 10      | 6       |
| P1      | 0       | 3     | 14    | 0        | 14      | 11      |

Here is your Gantt Chart....

P3 P3 P3 P3 P4 P4 P2 P2 P2 P2 P1 P1 P1

Average turn\_around Time = 8.250000

Average waiting Time = 4.750000

1. First Come First Searved
2. Shortest Job First
3. Round Robbin
4. Priority Sheduling
5. Shortest Remaining Time First
6. Longest Remaining Time First
7. Exit

Press: 5  
 Shortest Remaining Time First:

| Process | Arrival | Burst | Final | Priority | Turnarr | Waiting |
|---------|---------|-------|-------|----------|---------|---------|
| P1      | 0       | 3     | 3     | 0        | 3       | 0       |
| P4      | 3       | 2     | 5     | 4        | 2       | 0       |
| P2      | 1       | 4     | 9     | 0        | 8       | 4       |
| P3      | 0       | 5     | 14    | 5        | 14      | 9       |

Here is your Gantt Chart....

P1 P1 P1 P4 P4 P2 P2 P2 P2 P3 P3 P3 P3 P3

Average turn\_around Time = 6.750000

Average waiting Time = 3.250000

1. First Come First Searved
2. Shortest Job First
3. Round Robbin
4. Priority Sheduling
5. Shortest Remaining Time First
6. Longest Remaining Time First
7. Exit

```

Press: 6
Longest Remaining Time First:

Process Arrival Burst Final Priority Turnarr Waiting
P4      3      2     11      4       8       6
P3      0      5     12      5      12       7
P1      0      3     13      0      13      10
P2      1      4     14      0      13       9
Here is your Gantt Chart....
P3 P3 P2 P3 P1 P2 P2 P1 P3 P4 P4 P3 P1 P2
Average turn_around Time = 11.500000
Average waiting Time = 8.000000

1. First Come First Searved
2. Shortest Job First
3. Round Robbin
4. Priority Sheduling
5. Shortest Remaining Time First
6. Longest Remaining Time First
7. Exit

Press: 7

=====

                THANK YOU..:)

=====

Process returned 0 (0x0)   execution time : 27.279 s
Press any key to continue.

```