PROCESS SCHEDULING PROGRAM

```
/*
* process.h
*
* Created on: 08-Mar-2019
* Author: abhimanyu
*/
#ifndef PROCESS_H_
#define PROCESS_H_
typedef struct process
{
int pid, ps, rq;
float at, bt, ft, tat, wt, rt, pr,rbt;
} Process;
Process *process_create();
void process_delete(void *);
void process_init(Process *);
void process_display(const Process*);
#endif /* PROCESS_H_ */

/*
* node.h
*
* Created on: 08-Mar-2019
* Author: abhimanyu
*/
#include"process.h"
#ifndef NODE_H_
#define NODE_H_
typedef struct node
{
   Process *data;
   struct node *next;
} Node;
Node* node_create();
void node_delete(Node*);
#endif /* NODE_H_ */

/*
* queue.h
*
* Created on: 09-Mar-2019
* Author: abhimanyu
*/
#include "node.h"
#ifndef QUEUE_H_
#define QUEUE_H_
typedef struct queue
{
```

```
   Node *start, *end;
} Queue;
Queue* queue_create();
void queue_delete(Queue*);
void queue_push(Queue* q, Process* p);
Process * queue_pop(Queue* q);
void queue_display(const Queue* q);
Queue* queue_clone(Queue* q);
void queue_sort(Queue* q,char s[5]);
void calc_tat(Queue* q);
void calc_wt(Queue* q);
void gantt_chart(Queue* q);
void cal_avg_tat_wt(Queue* q);
#endif /* QUEUE_H_ */

/*
* main.h
*
* Created on: 09-Mar-2019
* Author: abhimanyu
*/
#include "queue.h"
#ifndef MAIN_H_
#define MAIN_H_
void fcfs(Queue *init);
void sjf(Queue* init);
void srtn(Queue* init);
void rr(Queue *init);
void dpriority_np(Queue* init);
void ipriority_np(Queue* init);
void dpriority(Queue* init);
void ipriority(Queue* init);
#endif /* MAIN_H_ */

/*
* main.c
*
* Created on: 08-Mar-2019
* Author: abhimanyu
*/
#include <stdio.h>
#include <stdlib.h>
#include "process.h"
#include "queue.h"
#include "node.h"
#include "main.h"
int main()
{
 Queue *q = queue_create();
 int i = 0, num, choice;
 char ch = 0;
```

```
 Node* n;
 printf("program to simulate cpu scheduling by ABHIMANYU MAURYA( 1713310003 AKTU )
B.Tech NIET Gr. Noida \n");
 printf("1.it can accept unsorted data\n2.it can calculate cpu idle time\n\n");
 printf("enter number of processes: ");
 scanf("%d",&num);
 if(num <= 0)
 {
  printf("number is invalid\n");
  exit(1);
 }
 for(i =0; i<num; i++)
 {
  Process* p = process_create();
  process_init(p);
  p->pid = i;
  queue_push(q, p);
 }
 printf("do you want to enter priority for processes(y/n)\n");
 scanf(" %c",&ch);
 if(ch == 'y')
 for (n = q->start; n != 0; n = n->next)
  {
   do
    {
    printf ("enter priority of P%d : ", n->data->pid);
       scanf ("%f", &n->data->pr);
    if(n->data->pr < 0.0) printf("priority must be greater or equal to zero\n");
    }while(n->data->pr < 0.0);
  }
 else if(ch != 'n') exit(0);
 do
  {
   printf("\n1. first come first serve\n2. sortest job first\n3. sortest remaining time next\n4.
round robin\n");
   printf("5. decreasing priority (non - preemptive)\n6. increasing priority (non -
preemptive)\n7. decreasing priority ( preemptive )\n8. increasing priority ( preemptive )\n9.
run 1 - 4\n10. run 5 - 8\n99. exit\n");
   printf("enter your choice: ");
   scanf("%d",&choice);
   switch(choice)
   {
    case 1: fcfs(q); break;
    case 2: sjf(q); break;
    case 3: srtn(q); break;
    case 4: rr(q); break;
    case 5: dpriority_np(q); break;
    case 6: ipriority_np(q); break;
    case 7: dpriority(q); break;
    case 8: ipriority(q); break;
    case 9: fcfs(q); sjf(q); srtn(q); rr(q); break;
```

```
    case 10: dpriority_np(q); ipriority_np(q); dpriority(q); ipriority(q); break;
    case 99: exit(0);
    default: printf("invalid choice");

   }
  }while(1);
}



/*
* process.c
*
* Created on: 08-Mar-2019
* Author: abhimanyu
*/
#include"process.h"
#include <stdio.h>
#include <stdlib.h>
#include<assert.h>

Process* process_create()
{
Process *t = (Process*)calloc(1,sizeof(Process));
assert(t);
return t;
}


void process_delete(void *t)
{
   assert(t);
free(t);
}

void process_init(Process *t)
{
float temp;
printf("enter arrival time: ");
scanf("%f",&t->at);
printf("enter bus time: ");
scanf("%f",&temp);
t->bt = temp;
t->rbt = temp;
}

void process_display(const Process* p)
{
   printf("P%-6d: %-6.3f %-6.3f \n", p->pid, p->at, p->bt);
}
```

```c
/*
* node.c
*
* Created on: 08-Mar-2019
* Author: abhimanyu
*/
#include "node.h"
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
Node* node_create()
{
   Node *t = calloc(1,sizeof(Node));
   assert(t);
   return t;
}

void node_delete(Node* t)
{
   assert(t);
   free(t);
}

void node_init(Node* n)
{
   n->data = process_create();
   process_init(n->data);
}

/*
* queue.c
*
* Created on: 09-Mar-2019
* Author: abhimanyu
*/
#include "queue.h"
#include "node.h"
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
Queue* queue_create()
{
   Queue* q = calloc(1,sizeof(Node));
   assert(q);
   return q;
}

void queue_delete(Queue* q)
{
   Node* n = 0, *t = 0;
```

```
   assert(q);
   for(n = q->start; n != 0; )
   {
      assert(n->data);
      process_delete(n->data);
      assert(n);
      t = n;
      n = n->next;
      node_delete(t);
   }
   assert(q);
   free(q);
}

void queue_push(Queue* q, Process* p)
{
   Node* n = node_create();
   n->data = p;
if (q->start == 0) q->start = q->end = n;
else
{
(q->end)->next = n;
q->end = n;
}
}


Process * queue_pop(Queue* q)
{
Node *n = q->start;
Process *t = n->data;
if(q->start->next == 0) q->start = q->end = 0 ;
else q->start = q->start->next;
node_delete(n);
return t;
}

void queue_display(const Queue* q)
{
Node* n = q->start;
printf("\nprocess AT BT\n");
for( ; n != 0; n = n ->next)
process_display(n->data);
}

Queue* queue_clone(Queue* q)
{
   Queue *tq = queue_create();
   Node* tn;
   for(tn = q->start; tn != 0; tn = tn->next)
   {
```

```
        Process* p = process_create();
        *p = *(tn->data) ;
        queue_push(tq, p);
    }
    return tq;
}

void queue_sort(Queue* q,char s[5])
{
Node* temp = q->start, *curr;
int num = 1,swapped=1,i,j;
if(temp == 0) return;
else
{
for(;temp->next != 0; temp = temp -> next)
num = num + 1;
}
void swap()
{
    Process *t = curr->data;
    curr->data = curr->next->data;
    curr->next->data = t;
    swapped = 1;
}
for( i = 0; i < num && swapped == 1; i++)
{
swapped = 0;
curr = q->start;
for( j =0 ; j < num - i -1 && curr != 0; j++)
{
    if(strcmp(s,"at") ==0 && curr->data->at > curr->next->data->at) swap();
    else if(strcmp(s,"bt") ==0 && curr->data->bt > curr->next->data->bt) swap();
    else if(strcmp(s,"rbt") ==0 && curr->data->rbt > curr->next->data->rbt) swap();
    else if(strcmp(s,"dpr") ==0 && curr->data->pr > curr->next->data->pr) swap();
    else if(strcmp(s,"ipr") ==0 && curr->data->pr < curr->next->data->pr) swap();
curr = curr->next;
}
}
}

void calc_tat(Queue* q)
{
    Node* n = q->start;
for(; n != 0; n = n->next)
    n->data->tat = n->data->ft - n->data->at;
}

void calc_wt(Queue* q)
{
    Node* n = q->start;
    for(; n != 0; n = n->next)
```

```
    n->data->wt = n->data->tat - n->data->bt;
}

void gantt_chart(Queue* q)
{
queue_sort(q,"at");
Node* n = q->start;
   printf("\n\ndata after gantt chart:\n");
printf("process AT BT     FT TAT WT RT\n");
for(; n != 0; n = n->next)
{
   Process* t = n->data;
printf("P%-6d: %-6.3f %-6.3f    %-6.3f  %-7.3f  %-6.3f %-6.3f\n", t->pid, t->at, t->bt, t->ft, t-
>tat, t->wt, t->rt);
}
}

void cal_avg_tat_wt(Queue* q)
{
Node* n = q->start;
int num = 0;
float total_tat = 0, total_wt = 0, avg_tat = 0, avg_wt = 0;
   for(; n != 0; n = n->next, num++)
   {
      total_tat = total_tat + n->data->tat;
      total_wt = total_wt + n->data->wt;
   }
avg_tat = total_tat / num;
   avg_wt = total_wt / num;
printf("\naverage turn arround time = %.3f\n",avg_tat);
   printf("average waiting time = %.3f\n",avg_wt);
}


/*
* fcfs.c
*
* Created on: 09-Mar-2019
* Author: abhimanyu
*/
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"
#include "process.h"
#include "node.h"
void fcfs(Queue *init)
{
Node* n;
Queue *rq = queue_clone(init);
int num = 0;    //number of elements in queue
float cpu_time = 0, idle_time = 0, throughput = 0,temp = 0;
```

```
printf("\nfirst come first serve process scheduling\n\n");
queue_sort(rq,"at");
for(n = rq->start; n != 0; n = n->next,num++);
for(n = rq->start; n != 0; n = n->next)
{
   Process* p = n->data;
if(p->at > cpu_time)
{
temp = p->at - cpu_time;
printf("| idle (%.1f-%.1f)|", cpu_time, cpu_time + temp);
cpu_time = cpu_time + temp;
idle_time = idle_time + temp;
}
p->ft = cpu_time + p->bt;
printf("| P%d (%.1f-%.1f)|", p->pid, cpu_time, p->ft);
p->rt = cpu_time;
cpu_time = p->ft;
}
calc_tat(rq);
calc_wt(rq);
gantt_chart(rq);
cal_avg_tat_wt(rq);
   throughput = num / cpu_time;
   printf("throughput of the system = %.3f\n",throughput);
printf("cpu idle time = %.3f\n",idle_time);
queue_delete(rq);
}


/*
* sjf.c
*
* Created on: 09-Mar-2019
* Author: abhimanyu
*/
#include<stdio.h>
#include<stdlib.h>
#include<limits.h>
#include "queue.h"
#include "process.h"
#include "node.h"
void sjf(Queue* init)
{
   Process *p;
Node* n, *add;
Queue *rq = queue_create(), *inpq= queue_clone(init);
float cpu_time = 0, idle_time = 0, throughput = 0, temp = 0;
int num = 0, i;
queue_sort(inpq,"at");
add = inpq->start;
cpu_time = idle_time = inpq->start->data->at;
```

```
for(n = inpq->start; n != 0; n = n->next,num++);
printf("\nsortest job first scheduling\n\n");
for(i = 0; i < num; i++)
{
   for( n = add; n != 0 && n->data->at <= cpu_time; n = n->next)
   {
      queue_push(rq,n->data);
      n->data->rq = 1;
      add = add->next;
   }
   queue_sort(rq,"bt");
   if(rq->start == 0)
   {
      temp = add->data->at - cpu_time;
      printf("| idle (%.1f-%.1f)|", cpu_time, cpu_time + temp);
      idle_time = idle_time + temp;
      cpu_time = cpu_time + temp;
      i = i - 1;
   }
   else
   {
      p = rq->start->data;
      p->ft = cpu_time + p->bt;
      printf("| P%d (%.1f-%.1f)|", p->pid, cpu_time, p->ft);
      p->rt = cpu_time;
      cpu_time = p->ft;
      p->ps = 1;
      queue_pop(rq);
   }
}
calc_tat(inpq);
calc_wt(inpq);
queue_sort(inpq,"at");
gantt_chart(inpq);
cal_avg_tat_wt(inpq);
   throughput = num / cpu_time;
   printf("throughput of the system = %.3f\n",throughput);
printf("cpu idle time = %.3f\n",idle_time);
queue_delete(inpq);
}


/*
* srtn.c
*
* Created on: 12-Mar-2019
* Author: abhimanyu
*/
#include<stdio.h>
#include<stdlib.h>
#include<limits.h>
```

```c
#include "queue.h"
#include "process.h"
#include "node.h"
void srtn(Queue* init)
{
   Process *p;
Node* n, *add;
Queue *rq = queue_create(), *inpq= queue_clone(init);
float cpu_time = 0, idle_time = 0, throughput = 0, temp = 0, max_time =0;
int num = 0, i;
queue_sort(inpq,"at");
add = inpq->start;
cpu_time = idle_time = inpq->start->data->at;
for(n = inpq->start; n != 0; n = n->next,num++);
printf("\nsortest remaining time next scheduling\n");
for(i = 0; i < num; i++)
{
   for( n = add; n != 0 && n->data->at <= cpu_time; n = n->next)
   {
    queue_push(rq,n->data);
    n->data->rq = 1;
    add = add->next;
    if(add != 0) max_time = add->data->at - cpu_time;
    else max_time = LLONG_MAX;
   }
   queue_sort(rq,"rbt");
   if(rq->start == 0)
   {
       temp = add->data->at - cpu_time;
       printf("| idle (%.1f-%.1f)|", cpu_time, cpu_time + temp);
       idle_time = idle_time + temp;
       cpu_time = cpu_time + temp;
       i = i - 1;
   }
   else
   {
      p = rq->start->data;
      if(p->rbt < max_time) temp = p->rbt;
      else temp = max_time;
      if(p->bt == p->rbt) p->rt = cpu_time;
      p->ft = cpu_time + temp;
      p->rbt = p->rbt - temp;
      printf("| P%d (%.1f-%.1f)|", p->pid, cpu_time, p->ft);
      cpu_time = p->ft;
      queue_pop(rq);
      if(p->rbt != 0)
      {
         queue_push(rq, p);
         i--;
      }
   }
```

```
}
calc_tat(inpq);
calc_wt(inpq);
queue_sort(inpq,"at");
gantt_chart(inpq);
cal_avg_tat_wt(inpq);
   throughput = num / cpu_time;
   printf("throughput of the system = %.3f\n",throughput);
printf("cpu idle time = %.3f\n",idle_time);
queue_delete(inpq);
}

/*
* rr.c
*
* Created on: 12-Mar-2019
* Author: abhimanyu
*/
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"
#include "process.h"
#include "node.h"
void rr(Queue *init)
{
Node* n, *add;
Queue *rq = queue_create(), *inpq= queue_clone(init);
Process *p;
int num = 0, i; //number of elements in queue
float cpu_time = 0, idle_time = 0, throughput = 0,temp = 0, quantum = 0;
printf("\nround robbin process scheduling\n\n");
printf("enter time quantum: ");
scanf("%f",&quantum);
queue_sort(inpq,"at");
add = inpq->start;
   for(n = inpq->start; n != 0; n = n->next, num++);
   void add_p()
   {
      for (n = add; n != 0 && n->data->at <= cpu_time; n = n->next)
      {
         queue_push(rq, n->data);
         n->data->rq = 1;
         add = add->next;
      }
   }
   for(i = 0; i < num; i++)
   {
      add_p();
      if(rq->start != 0)
      {
         p = rq->start->data;
```

```
        if (p->rbt < quantum)
           temp = p->rbt;
        else
           temp = quantum;
        if (p->bt == p->rbt)
           p->rt = cpu_time;
        p->ft = cpu_time + temp;
        p->rbt = p->rbt - temp;
        printf("| P%d (%.1f-%.1f)|", p->pid, cpu_time, p->ft);
        cpu_time = p->ft;

        if (p->rbt != 0)
        {
           queue_push(rq, p);
           i--;
        }
        queue_pop(rq);
        add_p();
     }

     if (rq->start == 0 && add != 0)
     {
        temp = add->data->at - cpu_time;
        printf("| idle (%.1f-%.1f)|", cpu_time, cpu_time + temp);
        idle_time = idle_time + temp;
        cpu_time = cpu_time + temp;
        add_p();
        i = i - 1;
     }
  }
calc_tat(inpq);
calc_wt(inpq);
gantt_chart(inpq);
cal_avg_tat_wt(inpq);
   throughput = num / cpu_time;
   printf("throughput of the system = %.3f\n",throughput);
printf("cpu idle time = %.3f\n",idle_time);
queue_delete(inpq);
}

/*
* dpriority_np.c
*
* Created on: 02-Apr-2019
* Author: abhimanyu
*/
#include<stdio.h>
#include<stdlib.h>
#include<limits.h>
#include "queue.h"
#include "process.h"
```

```c
#include "node.h"
void dpriority_np(Queue* init)
{
   Process *p;
   Node* n, *add;
   Queue *rq = queue_create(), *inpq = queue_clone(init);
   float cpu_time = 0, idle_time = 0, throughput = 0, temp = 0;
   int num = 0, i;
   queue_sort(inpq, "at");
   add = inpq->start;
   cpu_time = idle_time = inpq->start->data->at;
   for (n = inpq->start; n != 0; n = n->next, num++);
   printf("\nincreasing priority scheduling ( non - preemptive )\n");
   for (i = 0; i < num; i++)
   {
      for (n = add; n != 0 && n->data->at <= cpu_time; n = n->next)
      {
         queue_push(rq, n->data);
         n->data->rq = 1;
         add = add->next;
      }
      queue_sort(rq, "dpr");
      if (rq->start == 0 && add != 0)
      {
         temp = add->data->at - cpu_time;
         printf("| idle (%.1f-%.1f)|", cpu_time, cpu_time + temp);
         idle_time = idle_time + temp;
         cpu_time = cpu_time + temp;
         i = i - 1;
      }
      else
      {
         p = rq->start->data;
         if (p->bt == p->rbt)
            p->rt = cpu_time;
         p->ft = cpu_time + p->bt;
         p->rbt = p->rbt - temp;
         printf("| P%d (%.1f-%.1f)|", p->pid, cpu_time, p->ft);
         cpu_time = p->ft;
         queue_pop(rq);
      }
   }
   calc_tat(inpq);
   calc_wt(inpq);
   queue_sort(inpq, "at");
   gantt_chart(inpq);
   cal_avg_tat_wt(inpq);
   throughput = num / cpu_time;
   printf("throughput of the system = %.3f\n", throughput);
   printf("cpu idle time = %.3f\n", idle_time);
   queue_delete(inpq);
```

```
}


/*
* ipriority_np.c
*
* Created on: 02-Apr-2019
* Author: abhimanyu
*/
#include<stdio.h>
#include<stdlib.h>
#include<limits.h>
#include "queue.h"
#include "process.h"
#include "node.h"
void ipriority_np(Queue* init)
{
   Process *p;
   Node* n, *add;
   Queue *rq = queue_create(), *inpq = queue_clone(init);
   float cpu_time = 0, idle_time = 0, throughput = 0, temp = 0;
   int num = 0, i;
   queue_sort(inpq, "at");
   add = inpq->start;
   cpu_time = idle_time = inpq->start->data->at;
   for (n = inpq->start; n != 0; n = n->next, num++);
   printf("\nincreasing priority scheduling ( non - preemptive )\n");
   for (i = 0; i < num; i++)
   {
      for (n = add; n != 0 && n->data->at <= cpu_time; n = n->next)
      {
         queue_push(rq, n->data);
         n->data->rq = 1;
         add = add->next;
      }
      queue_sort(rq, "ipr");
      if (rq->start == 0 && add != 0)
      {
         temp = add->data->at - cpu_time;
         printf("| idle (%.1f-%.1f)|", cpu_time, cpu_time + temp);
         idle_time = idle_time + temp;
         cpu_time = cpu_time + temp;
         i = i - 1;
      }
      else
      {
         p = rq->start->data;
         if (p->bt == p->rbt)
            p->rt = cpu_time;
         p->ft = cpu_time + p->bt;
         p->rbt = p->rbt - temp;
```

```
            printf("| P%d (%.1f-%.1f)|", p->pid, cpu_time, p->ft);
            cpu_time = p->ft;
            queue_pop(rq);
        }
    }
    calc_tat(inpq);
    calc_wt(inpq);
    queue_sort(inpq, "at");
    gantt_chart(inpq);
    cal_avg_tat_wt(inpq);
    throughput = num / cpu_time;
    printf("throughput of the system = %.3f\n", throughput);
    printf("cpu idle time = %.3f\n", idle_time);
    queue_delete(inpq);
}


/*
* dpriority.c
*
* Created on: 14-Mar-2019
* Author: abhimanyu
*/
#include<stdio.h>
#include<stdlib.h>
#include<limits.h>
#include "queue.h"
#include "process.h"
#include "node.h"
void dpriority(Queue* init)
{
    Process *p;
    Node* n, *add;
    Queue *rq = queue_create(), *inpq = queue_clone(init);
    float cpu_time = 0, idle_time = 0, throughput = 0, temp = 0, max_time = 0;
    int num = 0, i;
    queue_sort(inpq, "at");
    add = inpq->start;
    cpu_time = idle_time = inpq->start->data->at;
    for (n = inpq->start; n != 0; n = n->next, num++);
    printf("\ndecreasing priority scheduling\n");
    for (i = 0; i < num; i++)
    {
        for (n = add; n != 0 && n->data->at <= cpu_time; n = n->next)
        {
            queue_push(rq, n->data);
            n->data->rq = 1;
            add = add->next;
            if (add != 0)
                max_time = add->data->at - cpu_time;
            else
```

```
            max_time = LLONG_MAX;
        }
        queue_sort(rq, "dpr");
        if (rq->start == 0 && add != 0)
        {
            temp = add->data->at - cpu_time;
            printf("| idle (%.1f-%.1f)|", cpu_time, cpu_time + temp);
            idle_time = idle_time + temp;
            cpu_time = cpu_time + temp;
            i = i - 1;
        }
        else
        {
            p = rq->start->data;
            if (p->rbt < max_time)
                temp = p->rbt;
            else
                temp = max_time;
            if (p->bt == p->rbt)
                p->rt = cpu_time;
            p->ft = cpu_time + temp;
            p->rbt = p->rbt - temp;
            printf("| P%d (%.1f-%.1f)|", p->pid, cpu_time, p->ft);
            cpu_time = p->ft;
            queue_pop(rq);
            if (p->rbt != 0)
            {
                queue_push(rq, p);
                i--;
            }
        }
    }
    calc_tat(inpq);
    calc_wt(inpq);
    queue_sort(inpq, "at");
    gantt_chart(inpq);
    cal_avg_tat_wt(inpq);
    throughput = num / cpu_time;
    printf("throughput of the system = %.3f\n", throughput);
    printf("cpu idle time = %.3f\n", idle_time);
    queue_delete(inpq);
}


/*
* ipriority.c
*
* Created on: 14-Mar-2019
* Author: abhimanyu
*/
#include<stdio.h>
```

```c
#include<stdlib.h>
#include<limits.h>
#include "queue.h"
#include "process.h"
#include "node.h"
void ipriority(Queue* init)
{
   Process *p;
   Node* n, *add;
   Queue *rq = queue_create(), *inpq = queue_clone(init);
   float cpu_time = 0, idle_time = 0, throughput = 0, temp = 0, max_time = 0;
   int num = 0, i;
   queue_sort(inpq, "at");
   add = inpq->start;
   cpu_time = idle_time = inpq->start->data->at;
   for (n = inpq->start; n != 0; n = n->next, num++);
   printf("\nincreasing priority scheduling\n");
   for (i = 0; i < num; i++)
   {
      for (n = add; n != 0 && n->data->at <= cpu_time; n = n->next)
      {
         queue_push(rq, n->data);
         n->data->rq = 1;
         add = add->next;
         if (add != 0)
            max_time = add->data->at - cpu_time;
         else
            max_time = LLONG_MAX;
      }

      queue_sort(rq, "ipr");
      if (rq->start == 0 && add != 0)
      {
         temp = add->data->at - cpu_time;
         printf("| idle (%.1f-%.1f)|", cpu_time, cpu_time + temp);
         idle_time = idle_time + temp;
         cpu_time = cpu_time + temp;
         i = i - 1;
      }
      else
      {
         p = rq->start->data;
         if (p->rbt < max_time)
            temp = p->rbt;
         else
            temp = max_time;
         if (p->bt == p->rbt)
            p->rt = cpu_time;
         p->ft = cpu_time + temp;
         p->rbt = p->rbt - temp;
         printf("| P%d (%.1f-%.1f)|", p->pid, cpu_time, p->ft);
```

```
            cpu_time = p->ft;
            queue_pop(rq);
            if (p->rbt != 0)
            {
               queue_push(rq, p);
               i--;
            }
         }
      }
      calc_tat(inpq);
      calc_wt(inpq);
      queue_sort(inpq, "at");
      gantt_chart(inpq);
      cal_avg_tat_wt(inpq);
      throughput = num / cpu_time;
      printf("throughput of the system = %.3f\n", throughput);
      printf("cpu idle time = %.3f\n", idle_time);
      queue_delete(inpq);
}
```