TY BCS

Operating System - I

Solved Practical Slips

2022-23

IMPORTANT NOTE: To execute OS practical's you must need to know how the algorithm works. You must need to trace the program on notebook along with a problem solution. It is impossible to remember the complete code, so it's better to learn algorithms first.

Solution of Orphan process and myshell search command is not prepared because of time limitation.

Slip 1_1 / Slip11_1:Write the simulation program to implement demand paging and show the page scheduling and total number of page faults according to the LFU page replacement algorithm. Assume the memory of n frames.

Reference String : 3,4,5,4,3,4,7,2,4,5,6,7,2,4,6

```
#include<stdio.h>
#define MAX 20

int frames[MAX],ref[MAX],mem[MAX][MAX],faults,
 sp,m,n,count[MAX];

void accept()
{
 int i;

 printf("Enter no.of frames:");
 scanf("%d", &n);

 printf("Enter no.of references:");
 scanf("%d", &m);

 printf("Enter reference string:\n");
 for(i=0;i<m;i++)
 {
  printf("[%d]=",i);
  scanf("%d",&ref[i]);
 }
}

void disp()
{
 int i,j;

 for(i=0;i<m;i++)
  printf("%3d",ref[i]);

 printf("\n\n");

 for(i=0;i<n;i++)
```

```
 {
  for(j=0;j<m;j++)
  {
   if(mem[i][j])
   printf("%3d",mem[i][j]);
   else
    printf("   ");
  }
  printf("\n");
 }
 printf("Total Page Faults: %d\n",faults);
}
int search(int pno)
{
 int i;
 for(i=0;i<n;i++)
 {
  if(frames[i]==pno)
   return i;
 }
 return -1;
}
int get_lfu(int sp)
{
 int i,min_i,min=9999;

 i=sp;
 do
 {
  if(count[i]<min)
  {
   min = count[i];
   min_i = i;
  }
  i=(i+1)%n;
 }while(i!=sp);

 return min_i;
}
void lfu()
{
 int i,j,k;

 for(i=0;i<m && sp<n;i++)
 {
  k=search(ref[i]);
  if(k==-1)
  {
```

```
   frames[sp]=ref[i];
   count[sp]++;
   faults++;
   sp++;

   for(j=0;j<n;j++)
    mem[j][i]=frames[j];
  }
  else
   count[k]++;

 }

 sp=0;
 for(;i<m;i++)
 {
  k = search(ref[i]);
  if(k==-1)
  {
   sp = get_lfu(sp);
   frames[sp] = ref[i];
   count[sp]=1;
   faults++;
   sp = (sp+1)%n;

   for(j=0;j<n;j++)
    mem[j][i] = frames[j];
  }
  else
   count[k]++;
 }
}


int main()
{
 accept();
 lfu();
 disp();

 return 0;
}
```

Slip 1_2/Slip13_1/Slip14_1/Slip20_1: Write a C program to implement the shell which displays the command prompt “myshell$”. It accepts the command, tokenize the command line and execute it by creating the child process. Also implement the additional command ‘typeline’ as

typeline +n filename :- To print first n lines in the file.
typeline -a filename :- To print all lines in the file.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <dirent.h>
#include <unistd.h>
int make_toks(char *s, char *tok[])
{
        int i = 0;
    char *p;
    p = strtok(s, " ");
while(p != NULL)
{
       tok[i++] = p;
    p = strtok(NULL, " ");
}
tok[i] = NULL;
return i;
}
void typeline(char *op, char *fn)
{
        int fh, i, j, n;
    char c;
fh = open(fn, O_RDONLY);
if(fh == -1)
{
printf("File %s not found.\n", fn);
return;
}
if(strcmp(op, "a") == 0)
{
    while(read(fh, &c, 1) > 0)
            printf("%c", c);
        close(fh);
    return;
}
n = atoi(op);
if(n > 0)
{
        i = 0;
        while(read(fh, &c, 1) > 0)
    {
            printf("%c", c);
```

```
            if(c == '\n')
                i++;
        if(i == n)
            break;
}
}
if(n < 0)
{
        i = 0;
        while(read(fh, &c, 1) > 0)
            {
            if(c == '\n') i++;
        }
        lseek(fh, 0, SEEK_SET);
j = 0;
while(read(fh, &c, 1) > 0)
{
    if(c == '\n')
            j++;
    if(j == i+n+1)
            break;
}
while(read(fh, &c, 1) > 0)
{
        printf("%c", c);
}
}
close(fh);
}
int main()
{
        char buff[80], *args[10];
        while(1)
    {
            printf ("\n");
        printf("\nmyshell$ ");
        fgets(buff, 80, stdin);
            buff[strlen(buff)-1] = '\0';
            int n = make_toks(buff, args);
        switch (n)
        {
                case 1: if(strcmp(args[0], "exit") == 0)
exit(1);
if (!fork())
execlp (args [0], args[0], NULL);
break;
            case 2:
if (!fork ())
```

```
execlp (args [0], args[0], args[1], NULL);
break;
                case 3: if (strcmp(args[0], "typeline") == 0)
                        typeline (args[1], args[2]);
else
{
        if (!fork ())
execlp (args [0], args[0], args[1], args[2], NULL);
}
break;
case 4:
if (!fork ())
execlp (args [0], args [0], args [1], args [2], args [3], NULL);
break;
}
}
return 0;
}
```

Slip2_1/Slip9_1/Slip10_1: Write the simulation program for demand paging and show the page scheduling and total number of page faults according the FIFO page replacement algorithm. Assume the memory of n frames.

Reference String : 3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6

```
#include<stdio.h>
int frame[5][2],nf;
int searchFrames(int sv)
{
    int x;
    for(x=0;x<nf;x++)
    {
        if(sv==frame[x][0])
        {
            return 0;
        }
    }
    return 1;
}
void displayMemory()
{
    int i;
    printf("\n\nFrame Contains |");
    for(i=0;i<nf;i++)
        printf(" %d  | ",frame[i][0]);
}
```

```c
int findFreeFrame()
{
    int i,min=frame[0][1],ri=0;
    for(i=0;i<nf;i++)
    {
        if(frame[i][1]==-1)
        {
            return i;
        }
    }
    //LRU
    for(i=0;i<nf;i++)
    {
        if(min>frame[i][1])
        {
            min=frame[i][1];
            ri=i;
        }
    }
    return ri;
}
main()
{
    int rs[]={3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6};
    int ts=0;
    int n=15,i,j,pf=0,srch,insert_index;
    printf("Enter how many frames");
    scanf("%d",&nf);
    for(i=0;i<nf;i++)
    {
        for(j=0;j<2;j++)
            frame[i][j]=-1;
    }
    displayMemory();
    for(i=0;i<n;i++)
    {
        srch=searchFrames(rs[i]);
        if(srch==1)
        {
            pf++;
            insert_index=findFreeFrame();
            frame[insert_index][0]=rs[i];
            frame[insert_index][1]=ts++;
        }

        displayMemory();
    }
    printf("\n\nTotal Page Faults Occured is %d\n",pf);
```

}

Slip2_2/Slip10_2/Slip11_2/Slip12_2/Slip15_1/Slip19_1: Write a program to implement the shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following ‘list’ commands as

myshell$ list f dirname:- To print names of all the files in current directory.

myshell$ list n dirname :- To print the number of all entries in the current directory

---

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <unistd.h>
void make_toks(char *s, char *tok[])
{
    int i=0;
    char *p;
    p = strtok(s," ");
    while(p!=NULL)
    {   tok[i++]=p;
        p=strtok(NULL," ");
    }

    tok[i]=NULL;
}

void list(char *dn, char op)
{
    DIR *dp;
    struct dirent *entry;
    int dc=0,fc=0;

    dp = opendir(dn);
    if(dp==NULL)
    {
        printf("Dir %s not found.\n",dn);
        return;
    }
```

```
    switch(op)
    {
        case 'f':
            while(entry=readdir(dp))
            {
                if(entry->d_type==DT_REG)
                    printf("%s\n",entry->d_name);
            }   break;
        case 'n':
            while(entry=readdir(dp))
            {
                if(entry->d_type==DT_DIR) dc++;
                if(entry->d_type==DT_REG) fc++;
            }

            printf("%d Dir(s)\t%d File(s)\n",dc,fc);
            break;
        case 'i':
            while(entry=readdir(dp))
            {
                if(entry->d_type==DT_REG)
                    printf("%s\t%lu\n",entry->d_name,entry->d_fileno);
            }
    }

    closedir(dp);
}

int main() {
    char buff[80],*args[10];
    int pid;

    while(1)
    {
        printf("myshell$");
        fflush(stdin);
        fgets(buff,80,stdin);
        buff[strlen(buff)-1]='\0';
        make_toks(buff,args);
        if(strcmp(args[0],"list")==0)
            list(args[2],args[1][0]);
        else
        {
            pid = fork();
            if(pid>0)
                wait();
            else
            {
```

```
                if(execvp(args[0],args)==-1)
                    printf("Bad command.\n");
            }
        }
    }
    return 0;
}
```

Slip3_1/Slip8_1/Slip12_1/Slip18_1: Write the simulation program to implement demand paging and show the page scheduling and total number of page faults according to the LRU (using counter method) page replacement algorithm. Assume the memory of n frames.

Reference String : 3,5,7,2,5,1,2,3,1,3,5,3,1,6,2

```
#include<stdio.h>
int frame[5][2],nf;
int searchFrames(int sv)
{
    int x;
    for(x=0;x<nf;x++)
    {
        if(sv==frame[x][0])
        {
            return x;
        }
    }
    return -2;
}
void displayMemory()
{
    int i;
    printf("\n\nFrame Contains |");
    for(i=0;i<nf;i++)
        printf(" %d  | ",frame[i][0]);
}
int findFreeFrame()
{
    int i,min=frame[0][1],ri=0;
    for(i=0;i<nf;i++)
    {
        if(frame[i][1]==-1)
        {
            return i;
        }
    }

    for(i=0;i<nf;i++)
    {
```

```
        if(min>frame[i][1])
        {
            min=frame[i][1];
            ri=i;
        }
    }
    return ri;
}
main()
{
    int rs[]={3,5,7,2,5,1,2,3,1,3,5,3,1,6,2};
    int ts=0;
    int n=15,i,j,pf=0,srch,insert_index;
    printf("Enter how many frames");
    scanf("%d",&nf);
    for(i=0;i<nf;i++)
    {
        for(j=0;j<2;j++)
            frame[i][j]=-1;
    }
    displayMemory();
    for(i=0;i<n;i++)
    {
        srch=searchFrames(rs[i]);
        if(srch==-2)
        {
            pf++;
            insert_index=findFreeFrame();
            frame[insert_index][0]=rs[i];
            frame[insert_index][1]=ts++;
        }
        else
        {
            frame[srch][1]=ts++;
        }

        displayMemory();
    }
    printf("\n\nTotal Page Faults Occured is %d\n",pf);
}
```

Slip 3_2/Slip16_1/Slip24_2 :Write a programto implement the toy shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following commands.

count c filename :- To print number of characters in the file.

count w filename :- To print number of words in the file.

count l filename :- To print number of lines in the file.

```
#include <sys/types.h>
#include<unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void make_toks(char *s, char *tok[])
{
        int i=0;
    char *p;
    p = strtok(s," ");
while(p!=NULL)
{  tok[i++]=p;
        p=strtok(NULL," ");
}
tok[i]=NULL;
}
void count(char *fn, char op)
{ int fh,cc=0,wc=0,lc=0;
        char c;
fh = open(fn,O_RDONLY);
if(fh==-1) {
 printf("File %s not found.\n",fn);
 return;
}
while(read(fh,&c,1)>0)
{ if(c==' ')
    wc++;
        else if(c=='\n')
 {
 wc++;
 lc++;
 }  cc++;
} close(fh);
switch(op)
{
        case 'c':
 printf("No.of characters:%d\n",cc-1);
 break;
        case  'w':
 printf("No.of words:%d\n",wc);
 break;
    case 'l':
 printf("No.of lines:%d\n",lc+1);
```

```
 break;
}
} int main()
{
    char buff[80],*args[10];
    int pid;
    while(1) { printf("myshell$ ");
        fflush(stdin);
            fgets(buff,80,stdin);
        buff[strlen(buff)-1]='\0';
            make_toks(buff,args);
if(strcmp(args[0],"count")==0)
count(args[2],args[1][0]);
else
 {  pid = fork();
     if(pid>0)
         wait();
         else
 {
 if(execvp(args[0],args)==-1)
 printf("Bad command.\n");
 }
 }
    }
        return 0;
}
```

Slip4_1: Write the simulation program for demand paging and show the page scheduling and total number of page faults according the MFU page replacement algorithm. Assume the memory of n frames.

Reference String : 8, 5, 7, 8, 5, 7, 2, 3, 7, 3, 5, 9, 4, 6, 2

```
#include<stdio.h>
#define MAX 20

int frames[MAX],ref[MAX],mem[MAX][MAX],faults,
 sp,m,n,count[MAX];

void accept()
{
 int i;

 printf("Enter no.of frames:");
 scanf("%d", &n);
```

```
 printf("Enter no.of references:");
 scanf("%d", &m);

 printf("Enter reference string:\n");
 for(i=0;i<m;i++)
 {
  printf("[%d]=",i);
  scanf("%d",&ref[i]);
 }
}

void disp()
{
 int i,j;

 for(i=0;i<m;i++)
  printf("%3d",ref[i]);

 printf("\n\n");

 for(i=0;i<n;i++)
 {
  for(j=0;j<m;j++)
  {
   if(mem[i][j])
   printf("%3d",mem[i][j]);
   else
    printf("   ");
  }
  printf("\n");
 }

 printf("Total Page Faults: %d\n",faults);
}

int search(int pno)
{
 int i;

 for(i=0;i<n;i++)
 {
  if(frames[i]==pno)
   return i;
 }

 return -1;
}
```

```
int get_mfu(int sp)
{
 int i,max_i,max=-9999;

 i=sp;
 do
 {
  if(count[i]>max)
  {
   max = count[i];
   max_i = i;
  }
  i=(i+1)%n;
 }while(i!=sp);

 return max_i;
}

void mfu()
{
 int i,j,k;

 for(i=0;i<m && sp<n;i++)
 {
  k=search(ref[i]);
  if(k==-1)
  {
   frames[sp]=ref[i];
   count[sp]++;
   faults++;
   sp++;

   for(j=0;j<n;j++)
    mem[j][i]=frames[j];
  }
  else
   count[k]++;

 }

 sp=0;
 for(;i<m;i++)
 {
  k = search(ref[i]);
  if(k==-1)
  {
   sp = get_mfu(sp);
```

```
    frames[sp] = ref[i];
    count[sp]=1;
    faults++;
    sp = (sp+1)%n;

    for(j=0;j<n;j++)
     mem[j][i] = frames[j];
   }
   else
    count[k]++;
 }
}


int main()
{
 accept();
 mfu();
 disp();

 return 0;
}
```

Slip5_1/ Slip7_1 / Slip17_1 / Slip23_2(you just need to change the reference string): Write the simulation program for demand paging and show the page scheduling and total number of page faults according the optimal page replacement algorithm. Assume the memory of n frames.

Reference String : 8, 5, 7, 8, 5, 7, 2, 3, 7, 3, 5, 9, 4, 6, 2

```
#include<stdio.h>
int main()
{
  int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1,
flag2, flag3, i, j, k, pos, max, faults = 0;
    printf("Enter number of frames: ");
    scanf("%d", &no_of_frames);

    printf("Enter number of pages: ");
    scanf("%d", &no_of_pages);

    printf("Enter page reference string: ");
```

```
for(i = 0; i < no_of_pages; ++i){
    scanf("%d", &pages[i]);
}

for(i = 0; i < no_of_frames; ++i){
    frames[i] = -1;
}

for(i = 0; i < no_of_pages; ++i){
    flag1 = flag2 = 0;

    for(j = 0; j < no_of_frames; ++j){
        if(frames[j] == pages[i]){
               flag1 = flag2 = 1;
               break;
           }
    }

    if(flag1 == 0){
        for(j = 0; j < no_of_frames; ++j){
            if(frames[j] == -1){
                faults++;
                frames[j] = pages[i];
                flag2 = 1;
                break;
            }
        }
    }

    if(flag2 == 0){
     flag3 =0;

        for(j = 0; j < no_of_frames; ++j){
         temp[j] = -1;

         for(k = i + 1; k < no_of_pages; ++k){
         if(frames[j] == pages[k]){
         temp[j] = k;
         break;
         }
         }
        }

        for(j = 0; j < no_of_frames; ++j){
         if(temp[j] == -1){
         pos = j;
         flag3 = 1;
```

```
                break;
                }
               }

               if(flag3 ==0){
                max = temp[0];
                pos = 0;

                for(j = 1; j < no_of_frames; ++j){
                if(temp[j] > max){
                max = temp[j];
                pos = j;
                }
                }
               }
frames[pos] = pages[i];
faults++;
        }
           printf("\n");
                for(j = 0; j < no_of_frames; ++j){
            printf("%d\t", frames[j]);
        }
    }
    printf("\n\nTotal Page Faults = %d", faults);

    return 0;
}
```

Slip 6_1: Write the simulation program for demand paging and show the page scheduling and total number of page faults according the MRU page replacement algorithm. Assume the memory of n frames.

Reference String : 8, 5, 7, 8, 5, 7, 2, 3, 7, 3, 5, 9, 4, 6, 2

```
#include<stdio.h>
int frame[5][2],nf;
int searchFrames(int sv)
{
    int x;
    for(x=0;x<nf;x++)
    {
        if(sv==frame[x][0])
```

```
        {
            return x;
        }
    }
    return -2;
}
void displayMemory()
{
    int i;
    printf("\n\nFrame Contains |");
    for(i=0;i<nf;i++)
        printf(" %d  | ",frame[i][0]);
}
int findFreeFrame()
{
    int i,max=frame[0][1],ri=0;
    for(i=0;i<nf;i++)
    {
        if(frame[i][1]==-1)
        {
            return i;
        }
    }

    for(i=0;i<nf;i++)
    {
        if(max<frame[i][1])
        {
            max=frame[i][1];
            ri=i;
        }
    }
    return ri;
}
main()
{
    int rs[]={12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8};
    int ts=0;
    int n=16,i,j,pf=0,srch,insert_index;
    printf("Enter how many frames");
    scanf("%d",&nf);
    for(i=0;i<nf;i++)
    {
        for(j=0;j<2;j++)
            frame[i][j]=-1;
    }
    displayMemory();
    for(i=0;i<n;i++)
```

```
    {
        srch=searchFrames(rs[i]);
        if(srch==-2)
        {
            pf++;
            insert_index=findFreeFrame();
            frame[insert_index][0]=rs[i];
            frame[insert_index][1]=ts++;
        }
        else
        {
            frame[srch][1]=ts++;
        }

        displayMemory();
    }
    printf("\n\nTotal Page Faults Occured is %d\n",pf);
}
```

Slip13_2/Slip19_2: Write the simulation program for Round Robin scheduling for given time quantum. The arrival time and first CPU-burst of different jobs should be input to the system. Accept no. of Processes, arrival time and burst time. The output should give the Gantt chart, turnaround time and waiting time for each process. Also display the average turnaround time and average waiting time.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct process_info
{
    char pname[20];
    int at,bt,ct,bt1;
    struct process_info *next;
}NODE;
int n,ts;
NODE *first,*last;
void accept_info()
{
    NODE *p;
    int i;
    printf("Enter no.of process:");
```

```
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        p = (NODE*)malloc(sizeof(NODE));
        printf("Enter process name:");
        scanf("%s",p->pname);
        printf("Enter arrival time:");
        scanf("%d",&p->at);
        printf("Enter first CPU burst time:");
        scanf("%d",&p->bt);
        p->bt1 = p->bt;

        p->next = NULL;
        if(first==NULL)
           first=p;
        else
            last->next=p;
        last = p;
    }
    printf("Enter time slice:");
    scanf("%d",&ts);
}
void print_output()
{
    NODE *p;
    float avg_tat=0,avg_wt=0;
    printf("pname\tat\tbt\tct\ttat\twt\n");
    p = first;
    while(p!=NULL)
    {
        int tat = p->ct-p->at;
        int wt = tat-p->bt;

        avg_tat+=tat;
        avg_wt+=wt;
        printf("%s\t%d\t%d\t%d\t%d\t%d\n",
                p->pname,p->at,p->bt,p->ct,tat,wt);

        p=p->next;
    }
    printf("Avg TAT=%f\tAvg WT=%f\n",
            avg_tat/n,avg_wt/n);
}
void print_input()
{
    NODE *p;
    p = first;
    printf("pname\tat\tbt\n");
```

```
    while(p!=NULL)
    {
        printf("%s\t%d\t%d\n",
                p->pname,p->at,p->bt1);
        p = p->next;
    }
}
void sort()
{
    NODE *p,*q;
    int t;
    char name[20];
    p = first;
    while(p->next!=NULL)
    {
        q=p->next;
        while(q!=NULL)
        {
            if(p->at > q->at)
            {
                strcpy(name,p->pname);
                strcpy(p->pname,q->pname);
                strcpy(q->pname,name);
                t = p->at;
                p->at = q->at;
                q->at = t;

                t = p->bt;
                p->bt = q->bt;
                q->bt = t;
                t = p->ct;
                p->ct = q->ct;
                q->ct = t;
                t = p->bt1;
                p->bt1 = q->bt1;
                q->bt1 = t;
            }
            q=q->next;
        }
        p=p->next;
    }
}
int time;
int is_arrived()
{
    NODE *p;
    p = first;
    while(p!=NULL)
```

```
    {
        if(p->at<=time && p->bt1!=0)
            return 1;
        p=p->next;
    }
    return 0;
}
NODE * delq()
{
    NODE *t;
    t = first;
    first = first->next;
    t->next=NULL;
    return t;
}
void addq(NODE *t)
{
    last->next = t;
    last = t;
}
struct gantt_chart
{
    int start;
    char pname[30];
    int end;
}s[100],s1[100];
int k;
void rr()
{
    int prev=0,n1=0;
    NODE *p;
    while(n1!=n)
    {
        if(!is_arrived())
        {
            time++;
            s[k].start = prev;
            strcpy(s[k].pname,"*");
            s[k].end = time;
            k++;
            prev=time;
        }
        else
        {
            p = first;
            while(1)
            {
                if(p->at<=time && p->bt1!=0)
```

```
                    break;
                p = delq();
                addq(p);
                p = first;
            }
            if(p->bt1<=ts)
            {
                time+=p->bt1;
                p->bt1=0;
            }
            else
            {
                time+=ts;
                p->bt1-=ts;
            }
            p->ct = time;
            s[k].start = prev;
            strcpy(s[k].pname,p->pname);
            s[k].end = time;
            k++;
            prev = time;

            if(p->bt1==0) n1++;
            p = delq();
            addq(p);
        }
        print_input();
    }
}
void print_gantt_chart()
{
    int i,j,m;
    s1[0] = s[0];
    for(i=1,j=0;i<k;i++)
    {
        if(strcmp(s[i].pname,s1[j].pname)==0)
            s1[j].end = s[i].end;
        else
            s1[++j] = s[i];
    }
    printf("%d",s1[0].start);
    for(i=0;i<=j;i++)
    {
        m = (s1[i].end - s1[i].start);
        for(k=0;k<m/2;k++)
            printf("-");
        printf("%s",s1[i].pname);
        for(k=0;k<(m+1)/2;k++)
```

```
            printf("-");
        printf("%d",s1[i].end);
    }
}
int main()
{
    accept_info();
    sort();
    rr();
    print_output();
    print_gantt_chart();
        return 0;
}
```

Slip14_2/Slip15_2/Slip20_2: Write a C program to simulate Non-preemptive Shortest Job First (SJF) – scheduling. The arrival time and first CPU-burst of different jobs should be input to the system. Accept no. of Processes, arrival time and burst time. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time

---

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

typedef struct PROCESS
{
    char name[80] ;
    int at , bt , ct , tat , tbt , wt ;
}PROCESS ;

PROCESS jobs[10] ;
int processCount , totalTat=0 , totalWt = 0 ;
float avgTat , avgWt ;

void sort()
{
    int i , j ;
    PROCESS p ;
    for(i = 0 ; i<processCount ; i++)
    {
        for(j=0 ; j<processCount ; j++)
        {
            if(jobs[i].at < jobs[j].at)
            {
                p = jobs[i] ;
```

```
                jobs[i] = jobs[j] ;
                jobs[j] = p ;
            }
        }
    }
}

int getJob(int time)
{
    int i , job , min = 999 ;
    for(i=0 ; i<processCount ; i++)
    {
        if(jobs[i].at<=time && jobs[i].tbt!=0)
        {
            if(jobs[i].tbt < min)
            {
                min = jobs[i].tbt ;
                job = i ;
            }
        }
    }
    return job ;
}

void getProcess()
{
    int i ;
    printf("\nEnter the number of processes: ") ;
    scanf("%d",&processCount) ;

    for(i=0 ; i<processCount ; i++)
    {
        printf("\nEnter the process name: ") ;
        scanf("%s",jobs[i].name) ;
        printf("Enter the CPU Burst time: ") ;
        scanf("%d",&jobs[i].bt) ;
        printf("Enter the Arrival time: ") ;
        scanf("%d",&jobs[i].at) ;
        jobs[i].tbt = jobs[i].bt ;
    }
    sort() ;
}

void process()
{
    int job , count = 0 ,time = 0 ;
    char currentJob[10] , prevJob[10] = "NULL" ;
    printf("\n\n GanttChart:\n") ;
```

```
    printf("_____\n") ;
    while(1)
    {
        job = getJob(time) ;
        jobs[job].tbt-- ;
        strcpy(currentJob,jobs[job].name) ;
        if(strcmp(currentJob,prevJob) !=0 )
        {
            printf("%d|%d  %s  ",time , time , currentJob) ;
        }
        time++ ;
        if(jobs[job].tbt==0)
        {
            jobs[job].ct = time ;
            jobs[job].tat = time - jobs[job].at ;
            jobs[job].wt = jobs[job].tat-jobs[job].bt ;
            count++ ;
            totalTat += jobs[job].tat ;
            totalWt+=jobs[job].wt ;
        }
        strcpy(prevJob , currentJob) ;
        if(count==processCount)
            break ;
    }
    printf("%d|",time) ;
    printf("\n_____\n") ;
    avgTat = (float)totalTat/processCount ;
    avgWt = (float)totalWt/processCount ;
}

void display()
{
    int i = 0 ;
    printf("\n\n----------
----------\n") ;
    printf("Process name\tArrival Time\tBurst Time\tCPU Time\tTurn Around
time\tWait Time\n");
    printf("----------
----------\n");
    for(i=0 ; i<processCount ; i++)
        printf("   %s    \t   %d       \t   %d     \t   %d        \t     %d
      \t   %d \n",jobs[i].name ,  jobs[i].at , jobs[i].bt , jobs[i].ct ,
jobs[i].tat , jobs[i].wt );
    printf("----------
----------") ;
    printf("\n\nTotal Turn Around Time: %d",totalTat) ;
    printf("\nTotal Waiting Time: %d",totalWt) ;
    printf("\n\nAverage Turn Around Time: %f",avgTat) ;
```

```
    printf("\nAverage Waiting Time: %f\n",avgWt) ;
}


main()
{
    system("clear");
    getProcess() ;
    process() ;
    display() ;
}
```

Slip16_2 / Slip21_2/Slip 22_2: Write the program to simulate Non preemptive priority scheduling. The arrival time and first CPU-burst of different jobs should be input to the system. Accept no. of Processes, arrival time and burst time. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

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

typedef struct PROCESS
{
    char name[80] ;
    int at , bt , ct , tat , tbt , wt , priority ;
}PROCESS ;

PROCESS jobs[10] ;
int time = 0 , processCount , totalTat=0 , totalWt = 0 ;
float avgTat , avgWt ;

void sort()
{
    int i , j ;
    PROCESS p ;
    for(i = 0 ; i<processCount ; i++)
    {
        for(j=0 ; j<processCount ; j++)
        {
            if(jobs[i].at < jobs[j].at)
            {
                p = jobs[i] ;
                jobs[i] = jobs[j] ;
```

```
                jobs[j] = p ;
            }
        }
    }
}

int getJob()
{
    int i , job , min = 999 ;
    for(i=0 ; i<processCount ; i++)
    {
        if(jobs[i].at<=time && jobs[i].tbt!=0)
        {
            if(jobs[i].priority <= min)
            {
                min = jobs[i].tbt ;
                job = i ;
            }
        }
    }
    return job ;
}

void getProcess()
{
    int i ;
    printf("\nEnter the number of processes: ") ;
    scanf("%d",&processCount) ;

    for(i=0 ; i<processCount ; i++)
    {
        printf("\nEnter the process name: ") ;
        scanf("%s",jobs[i].name) ;
        printf("Enter the CPU Burst time: ") ;
        scanf("%d",&jobs[i].bt) ;
        printf("Enter the Arrival time: ") ;
        scanf("%d",&jobs[i].at) ;
        jobs[i].tbt = jobs[i].bt ;
        printf("Enter the process priority: ") ;
        scanf("%d",&jobs[i].priority) ;
    }
    sort() ;
}

void process()
{
    int job , count = 0  ;
    char currentJob[10] , prevJob[10] = "NULL" ;
```

```
    printf("\tGantt Chart\n\n") ;
    printf("_____________________________________________\n") ;
    printf("|") ;
    printf("0") ;
    while(1)
    {
        job = getJob() ;
        jobs[job].tbt-- ;
        strcpy(currentJob,jobs[job].name) ;
        if(strcmp(currentJob,prevJob) !=0 )
        {
            printf("  %s  %d|%d ",currentJob,time , time) ;
        }
        time++ ;
        if(jobs[job].tbt==0)
        {
            jobs[job].ct = time ;
            jobs[job].tat = jobs[job].ct - jobs[job].at ;
            jobs[job].wt = jobs[job].tat-jobs[job].bt ;
            count++ ;
            totalTat += jobs[job].tat ;
            totalWt+=jobs[job].wt ;
        }
        strcpy(prevJob , currentJob) ;
        if(count==processCount)
            break ;
    }
    printf("\n_____________________________________________\n") ;
    avgTat = (float)totalTat/processCount ;
    avgWt = (float)totalWt/processCount ;
}

void display()
{
    int i = 0 ;
    printf("\n\n-----------------------------------------------------------------------------------------------------------------------------------------\n") ;
    printf("Process name\tArrival Time\tBurst Time\tCPU Time\tTurn Around time\tWait Time\n");
    printf("-----------------------------------------------------------------------------------------------------------------------------------------\n");
    for(i=0 ; i<processCount ; i++)
        printf(" %s\t%d\t%d\t%d\t%d\t%d\n",jobs[i].name , jobs[i].at , jobs[i].bt , jobs[i].ct , jobs[i].tat , jobs[i].wt ) ;
    printf("-----------------------------------------------------------------------------------------------------------------------------------------") ;
    printf("\n\nTotal Turn Around Time: %d",totalTat) ;
    printf("\nTotal Waiting Time: %d",totalWt) ;
```

```
    printf("\n\nAverage Turn Around Time: %f",avgTat) ;
    printf("\nAverage Waiting Time: %f\n",avgWt) ;
}

main()
{
    system("clear");
    getProcess() ;
    process() ;
    display() ;
}
```

Slip17_2/Slip18_2 : Write the program to simulate FCFS CPU-scheduling. The arrival time and first CPU-burst of different jobs should be input to the system. Accept no. of Processes, arrival time and burst time. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time

---

```
#include<stdio.h>
void findWaitingTime(int processes[], int n,
int bt[], int wt[])
{
wt[0] = 0;
int i;
for (i = 1; i < n ; i++ ){

wt[i] = bt[i-1] + wt[i-1] ;
}
}
void findTurnAroundTime( int processes[], int n,
int bt[], int wt[], int tat[])
{
int i;
for ( i = 0; i < n ; i++)
tat[i] = bt[i] + wt[i];
}
void findavgTime( int processes[], int n, int bt[])
{
int wt[n], tat[n], total_wt = 0, total_tat = 0;
findWaitingTime(processes, n, bt, wt);
findTurnAroundTime(processes, n, bt, wt, tat);
printf("Processes Burst time Waiting time Turn around time\n");
int i;
for ( i=0; i<n; i++)
{
total_wt = total_wt + wt[i];
total_tat = total_tat + tat[i];
printf(" %d ",(i+1));
printf(" %d ", bt[i] );
```

```
printf(" %d",wt[i] );
printf(" %d\n",tat[i] );
}
int s=(float)total_wt / (float)n;
int t=(float)total_tat / (float)n;
printf("Average waiting time = %d",s);
printf("\n");
printf("Average turn around time = %d ",t);
}
int main()
{
int processes[] = { 1, 2, 3};
int n = sizeof processes / sizeof processes[0];
int burst_time[] = {10, 5, 8};
findavgTime(processes, n, burst_time);
return 0;
}
```