```
//Best Fit memeory allocation
#include<stdio.h>

int main() {
    int p[10], np, b[10], nb, ch, c[10], alloc[10], flag[10], i, j;
    printf("\nEnter the no of process:");
    scanf("%d", &np);
    printf("\nEnter the no of blocks:");
    scanf("%d", &nb);
    printf("\nEnter the size of each process:");
    for (i = 0; i < np; i++) {
        printf("\nProcess %d:", i);
        scanf("%d", &p[i]);
    }
    printf("\nEnter the block sizes:");
    for (j = 0; j < nb; j++) {
        printf("\nBlock %d:", j);
        scanf("%d", &b[j]);
        c[j] = b[j];
    }

    printf("\nBest Fit\n");
    for (i = 0; i < nb; i++) {
        for (j = i + 1; j < nb; j++) {
            if (c[i] > c[j]) {
                int temp = c[i];
                c[i] = c[j];
                c[j] = temp;
            }
        }
    }
    printf("\nAfter sorting block sizes:");
    for (i = 0; i < nb; i++)
        printf("\nBlock %d:%d", i, c[i]);

    for (i = 0; i < np; i++) {
        for (j = 0; j < nb; j++) {
            if (p[i] <= c[j]) {
                alloc[j] = p[i];
                printf("\n\nAlloc[%d]", alloc[j]);
                printf("\n\nProcess %d of size %d is allocated in block %d of size %d", i, p[i], j,
c[j]);
                flag[i] = 0;
                c[j] = 0;
                break;
            } else
                flag[i] = 1;
        }
```

```
    }
    for (i = 0; i < np; i++) {
        if (flag[i] != 0)
            printf("\n\nProcess %d of size %d is not allocated", i, p[i]);
    }

    return 0;
}

//Worst Fit memory allocation
#include<stdio.h>

int main() {
    int p[10], np, b[10], nb, ch, d[10], alloc[10], flag[10], i, j;
    printf("\nEnter the no of process:");
    scanf("%d", &np);
    printf("\nEnter the no of blocks:");
    scanf("%d", &nb);
    printf("\nEnter the size of each process:");
    for (i = 0; i < np; i++) {
        printf("\nProcess %d:", i);
        scanf("%d", &p[i]);
    }
    printf("\nEnter the block sizes:");
    for (j = 0; j < nb; j++) {
        printf("\nBlock %d:", j);
        scanf("%d", &b[j]);
        d[j] = b[j];
    }

    printf("\nWorst Fit\n");
    for (i = 0; i < nb; i++) {
        for (j = i + 1; j < nb; j++) {
            if (d[i] < d[j]) {
                int temp = d[i];
                d[i] = d[j];
                d[j] = temp;
            }
        }
    }
    printf("\nAfter sorting block sizes:");
    for (i = 0; i < nb; i++)
        printf("\nBlock %d:%d", i, d[i]);

    for (i = 0; i < np; i++) {
        for (j = 0; j < nb; j++) {
            if (p[i] <= d[j]) {
                alloc[j] = p[i];
```

```
            printf("\n\nAlloc[%d]", alloc[j]);
            printf("\n\nProcess %d of size %d is allocated in block %d of size %d", i, p[i], j,
d[j]);
            flag[i] = 0;
            d[j] = 0;
            break;
         } else
            flag[i] = 1;
      }
   }
   for (i = 0; i < np; i++) {
      if (flag[i] != 0)
         printf("\n\nProcess %d of size %d is not allocated", i, p[i]);
   }

   return 0;
}

//First Fit memory allocation

#include<stdio.h>

int main() {
   int p[10], np, b[10], nb, ch, alloc[10], flag[10], i, j;
   printf("\nEnter the no of process:");
   scanf("%d", &np);
   printf("\nEnter the no of blocks:");
   scanf("%d", &nb);
   printf("\nEnter the size of each process:");
   for (i = 0; i < np; i++) {
      printf("\nProcess %d:", i);
      scanf("%d", &p[i]);
   }
   printf("\nEnter the block sizes:");
   for (j = 0; j < nb; j++) {
      printf("\nBlock %d:", j);
      scanf("%d", &b[j]);
   }

   printf("\nFirst Fit\n");
   for (i = 0; i < np; i++) {
      for (j = 0; j < nb; j++) {
         if (p[i] <= b[j]) {
            alloc[j] = p[i];
            printf("\n\nAlloc[%d]", alloc[j]);
            printf("\n\nProcess %d of size %d is allocated in block %d of size %d", i, p[i], j,
b[j]);
            flag[i] = 0;
```

```
                b[j] = 0;
                break;
            } else
                flag[i] = 1;
        }
    }
    for (i = 0; i < np; i++) {
        if (flag[i] != 0)
            printf("\n\nProcess %d of size %d is not allocated", i, p[i]);
    }

    return 0;
}

//FCFS
#include<stdio.h>
#include<stdlib.h>
void main ()
{
int n,pid[10],at[10],bt[10],ft[10],wt[10],ta[10],i,j,t,stt=0,totta=0,totwt=0;
float avgta,avgwt;
printf("ENTER THE NO.OF PROCESSES:");
scanf("%d",&n);
for(i=1;i<=n;i++) {
pid[i]=i;
printf("\n ENTER THE ARRIVAL TIME:");
scanf("%d",&at[i]);
printf("\n ENTER THE BURST TIME:");
scanf("%d",&bt[i]);
}
for(i=1;i<=n;i++)
{
for(j=i+1;j<=n;j++)
{
if(at[i]>at[j])
{
 t=pid[i];
 pid[i]=pid[j];
 pid[j]=t;
 t=at[i];
 at[i]=at[j];
 at[j]=t;
 t=bt[i];
 bt[i]=bt[j];
 bt[j]=t;
}
stt=at[1]; }
}
```

```
printf("\nTHE VALUES OF THE ARRIVAL TIME IS %d",stt);
for(i=1;i<=n;i++)
{
ft[i]=stt+bt[i];
wt[i]=stt-at[i];
ta[i]=ft[i]-at[i];
totta=totta+ta[i];
totwt=totwt+wt[i];
stt=ft[i]; }
avgta=(float)totta/n;
avgwt=(float)totwt/n;
printf("\nPNO\tARRIVAL TIME\tBURST TIME\tCOMPLETION TIME\t\tWAIT TIME\tTAT\n");
for(i=1;i<=n;i++)
printf("%d\t\t%d\t\t%d\t\t%d\t\t\t%d\t\t%d\n",pid[i],at[i],bt[i],ft[i],wt[i],ta[i]);
printf("\nAVERAGE TURN AROUND TIME=%f",avgta);
printf("\nAVERAGE WAITING TIME=%f",avgwt);
}

// Shortest Job First
#include<stdio.h>
void main( )
{
int i,j,t,n,stt=0,pid[10],at[10],bt[10],ft[10],att,wt[10],ta[10],totwt=0,totta=0;
float avgwt,avgta;
printf("ENTER THE NUMBER OF PROCESSES:");
scanf("%d",&n);
 printf("\nENTER THE ARRIVAL TIME:");
scanf("%d",&att);
for(i=1;i<=n;i++)
{
pid[i]=i;
at[i]=att;
printf("\nENTER THE BURST TIME:");
scanf("%d",&bt[i]);
}
for(i=1;i<=n;i++)
{
for(j=i+1;j<=n;j++)
{
if(bt[i]>bt[j])
 {
 t=pid[i];
 pid[i]=pid[j];
 pid[j]=t;
 t=bt[i];
 bt[i]=bt[j];
 bt[j]=t;
 }
```

```
stt=att;
} }
for(i=1;i<=n;i++)
{
ft[i]=stt+bt[i];
wt[i]=stt-at[i];
ta[i]=ft[i]-at[i];
totta=totta+ta[i];
totwt=totwt+wt[i];
stt=ft[i];
}
avgwt=(float)totwt/n;
avgta=(float)totta/n;
printf("\nPNO\tARRIVAL TIME\tBURST TIME\tCOMPLETION TIME\tWAIT TIME\tTAT");
for(i=1;i<=n;i++)
{
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",pid[i],at[i],bt[i],ft[i],wt[i],ta[i]);
}
printf("\nAVERAGE TURN AROUND TIME=%f",avgta);
printf("\nAVERAGE WAITING TIME=%f",avgwt);
}
#include<stdio.h>
void main( )
{
int i,j,t,n,stt=0,pid[10],pr[10],at[10],bt[10],ft[10],att;
int wt[10],ta[10],totwt=0,totta=0;
float avgwt,avgta;
printf("ENTER THE NO.OF.PROCESS:");
scanf("%d",&n);
printf("\nENTER THE ARRIVAL TIME:");
scanf("%d",&att);
for(i=1;i<=n;i++)
{
pid[i]=i;
at[i]=att;
printf("\nENTER THE BURST TIME:");
scanf("%d",&bt[i]);
printf("\nENTER THE PRIORITY OF THE PROCESS:");
scanf("%d",&pr[i]);
}
for(i=1;i<=n;i++)
{
for(j=i+1;j<=n;j++)
{
if(pr[i]>pr[j])
{
t=pid[i];
pid[i]=pid[j];
```

```
pid[j]=t;
t=bt[i];
bt[i]=bt[j];
bt[j]=t;
t=pr[i];
pr[i]=pr[j];
pr[j]=t;
} }
}
stt=att;
for(i=1;i<=n;i++)
{
ft[i]=stt+bt[i];
wt[i]=stt-at[i];
ta[i]=ft[i]-at[i];
if(wt[i]<0)
wt[i]=0;
totwt=totwt+wt[i];
totta=totta+ta[i];
stt=ft[i];
}
avgwt=(float)totwt/n;
avgta=(float)totta/n;
printf("PNO\tARR TIME\tBURST TIME\tFINISH TIME\tWAIT TIME\tTURN TIME\n");
for(i=1;i<=n;i++)
{
printf("\n%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d",pid[i],at[i],bt[i],ft[i],wt[i],ta[i]);
}
printf("\n THE AVERAGE WAITING TIME IS:%f\n",avgwt);
printf("\n THE AVERAGE TURN TIME IS:%f\n",avgta);
}

//priority
#include<stdio.h>
void main( )
{
int i,j,t,n,stt=0,pid[10],pr[10],at[10],bt[10],ft[10],att;
int wt[10],ta[10],totwt=0,totta=0;
float avgwt,avgta;
printf("ENTER THE NO.OF.PROCESS:");
scanf("%d",&n);
printf("\nENTER THE ARRIVAL TIME:");
scanf("%d",&att);
for(i=1;i<=n;i++)
{
pid[i]=i;
at[i]=att;
printf("\nENTER THE BURST TIME:");
```

```
scanf("%d",&bt[i]);
printf("\nENTER THE PRIORITY OF THE PROCESS:");
scanf("%d",&pr[i]);
}
for(i=1;i<=n;i++)
{
for(j=i+1;j<=n;j++)
{
if(pr[i]>pr[j])
{
t=pid[i];
pid[i]=pid[j];
pid[j]=t;
t=bt[i];
bt[i]=bt[j];
bt[j]=t;
t=pr[i];
pr[i]=pr[j];
pr[j]=t;
} }
}
stt=att;
for(i=1;i<=n;i++)
{
ft[i]=stt+bt[i];
wt[i]=stt-at[i];
ta[i]=ft[i]-at[i];
if(wt[i]<0)
wt[i]=0;
totwt=totwt+wt[i];
totta=totta+ta[i];
stt=ft[i];
}
avgwt=(float)totwt/n;
avgta=(float)totta/n;
printf("PNO\tARR TIME\tBURST TIME\tFINISH TIME\tWAIT TIME\tTURN TIME\n");
for(i=1;i<=n;i++)
{
printf("\n%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d",pid[i],at[i],bt[i],ft[i],wt[i],ta[i]);
}
printf("\n THE AVERAGE WAITING TIME IS:%f\n",avgwt);
printf("\n THE AVERAGE TURN TIME IS:%f\n",avgta);
}

//Round robin
#include<stdio.h>
main( )
{
```

```
int i,j,n,wt[10],ta[10],at[10],bt[10],tot_wt=0,tot_ta=0,ft[10],t;
int s[10],prid[10],p[10],max=0,temp,stt=0,ts=0,x=0;
float avg_wt,avg_ta;
printf("Enter the no. of process:");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
prid[i]=i;
printf("\n Enter the Arrival time of process %d:",i);
scanf("%d",&at[i]);
printf("\n Enter the Burst time of the process %d:",i);
scanf("%d",&bt[i]);
wt[i]=0;
p[i]=0;
if(at[i]>max)
{
max=at[i];
}
}
printf("\n Enter the time Slice:");
scanf("%d",&ts);
for(i=1;i<n;i++)
{
for(j=i+1;j<=n;j++)
{
if(at[i]>at[j])
{
t=at[i];
at[i]=at[j];
at[j]=t;
t=bt[i];
bt[j]=bt[j];
bt[j]=t;
} } }
for(i=1;i<=n;i++)
{
s[i]=bt[i];
}
i=1;
x=0;
while(x<n)
{
if(p[i]==1)
goto con;
if(at[i]>stt)
{
temp=max;
for(i=1;i<=n;i++)
```

```
{
if(p[i]==0 && at[i]<=temp)
{
temp=at[i];
}
}
if(temp>stt)
{
stt=temp;
}
if(at[i]>stt)
goto con;
} if(s[i]>ts)
{
s[i]=s[i]-ts;
stt=stt+ts;
}
else
{
stt=stt+s[i];
ft[i]=stt;
s[i]=0;
p[i]=1;
x++;
}
con:
i++;
if(i>n)
i=1; }
for(i=1;i<=n;i++)
{
ta[i]=ft[i]-at[i];
wt[i]=ta[i]-bt[i];
tot_ta+=ta[i];
tot_wt+=wt[i];
}
avg_wt=(float)tot_wt/n;
avg_ta=(float)tot_ta/n;
printf("\n PNO\tARR TIME\tBURST TIME\tWAIT TIME\t TURN TIME\t FINISH TIME");
for(i=1;i<=n;i++)
{
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d\t%d",i,at[i],bt[i],wt[i],ta[i],ft[i]);
}
printf("\n The Average Waiting Time is:%0.2f",avg_wt);
printf("\n The Average Turn Time is :%0.2f",avg_ta);
}
```

//Deadlock avoidance(Bankers algorithm)

```
#include<stdio.h>
int max[100][100];
int alloc[100][100];
int need[100][100];
int avail[100];
int n,r;
void input( );
void show( );
void cal( );
int main( )
{
int i,j;
printf("********** Deadlock Avoidance ***********\n");
input( );
show( );
cal( );
return 0;
}
void input( )
{
int i,j;
printf("Enter the no of Processes\t");
scanf("%d",&n);
printf("Enter the no of resource instances\t");
scanf("%d",&r);
printf("Enter the Max Matrix\n");
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
scanf("%d",&max[i][j]);
}
printf("Enter the Allocation Matrix\n");
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
scanf("%d",&alloc[i][j]);
}
printf("Enter the available Resources\n");
for(j=0;j<r;j++)
scanf("%d",&avail[j]);
}
void show( )
{
int i,j;
printf("Process\t Allocation\t Max\t Available\t");
for(i=0;i<n;i++)
{
printf("\nP%d\t ",i+1);
```

```
for(j=0;j<r;j++)
printf("%d ",alloc[i][j]);
printf("\t");
for(j=0;j<r;j++)
printf("%d ",max[i][j]);
printf("\t");
if(i==0)
{
for(j=0;j<r;j++)
printf("%d ",avail[j]);
}
}
}
void cal( )
{
int finish[100],temp,need[100][100],flag=1,k,c1=0;
int dead[100];
int safe[100];
int i,j;
for(i=0;i<n;i++)
finish[i]=0;
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
need[i][j]=max[i][j]-alloc[i][j];
}
while(flag)
{
flag=0;
for(i=0;i<n;i++)
{
int c=0;
for(j=0;j<r;j++)
{
if((finish[i]==0)&&(need[i][j]<=avail[j]))
{
c++;
if(c==r)
{
for(k=0;k<r;k++)
{
avail[k]+=alloc[i][j];
finish[i]=1;
flag=1;
}
printf("p%d->",i);
if(finish[i]==1)
{
```

```
i=n;
}
}
}
} } }
j=0;
flag=0;
for(i=0;i<n;i++)
{
if(finish[i]==0)
{
dead[j]=i;
j++;
flag=1;
} }
if(flag==1)
{
printf("\n\nSystem is in Deadlock and the Deadlock process are\n");
for(i=0;i<n;i++)
printf("P%d\t",dead[i]);
}
else
{
printf("\nNo Deadlock Occur");
} }

//deadlock avoidance
#include<stdio.h>

int max[100][100];
int alloc[100][100];
int avail[100];
int need[100][100];
int n,r;

void input();
void show();
void cal();

int main() {
   printf("********** Deadlock Detection Algorithm ************\n");
   input();
   show();
   cal();
   return 0;
}

void input() {
```

```
    int i,j;
    printf("Enter the number of Processes: ");
    scanf("%d", &n);
    printf("Enter the number of resource instances: ");
    scanf("%d", &r);
    printf("Enter the Max Matrix:\n");
    for(i = 0; i < n; i++) {
        for(j = 0; j < r; j++)
            scanf("%d", &max[i][j]);
    }
    printf("Enter the Allocation Matrix:\n");
    for(i = 0; i < n; i++) {
        for(j = 0; j < r; j++)
            scanf("%d", &alloc[i][j]);
    }
    printf("Enter the available Resources:\n");
    for(j = 0; j < r; j++)
        scanf("%d", &avail[j]);
}

void show() {
    int i,j;
    printf("Process\t Allocation\t Max\t Available\n");
    for(i = 0; i < n; i++) {
        printf("P%d\t ", i+1);
        for(j = 0; j < r; j++)
            printf("%d ", alloc[i][j]);
        printf("\t");
        for(j = 0; j < r; j++)
            printf("%d ", max[i][j]);
        printf("\t");
        if(i == 0) {
            for(j = 0; j < r; j++)
                printf("%d ", avail[j]);
        }
        printf("\n");
    }
}

void cal() {
    int finish[100], temp, need[100][100], flag = 1, k, c1 = 0;
    int dead[100];
    int safe[100];
    int i, j;
    for(i = 0; i < n; i++)
        finish[i] = 0;
    for(i = 0; i < n; i++) {
        for(j = 0; j < r; j++)
```

```
            need[i][j] = max[i][j] - alloc[i][j];
    }
    while(flag) {
        flag = 0;
        for(i = 0; i < n; i++) {
            int c = 0;
            for(j = 0; j < r; j++) {
                if((finish[i] == 0) && (need[i][j] <= avail[j])) {
                    c++;
                    if(c == r) {
                        for(k = 0; k < r; k++) {
                            avail[k] += alloc[i][j];
                            finish[i] = 1;
                            flag = 1;
                        }
                        printf("p%d->", i);
                        if(finish[i] == 1) {
                            i = n;
                        }
                    }
                }
            }
        }
    }
    j = 0;
    flag = 0;
    for(i = 0; i < n; i++) {
        if(finish[i] == 0) {
            dead[j] = i;
            j++;
            flag = 1;
        }
    }
    if(flag == 1) {
        printf("\n\nSystem is in Deadlock and the Deadlock process are:\n");
        for(i = 0; i < n; i++)
            printf("P%d\t", dead[i]);
    } else {
        printf("\nNo Deadlock Occurs\n");
    }
}
//paging
#include<stdio.h>
int main( )
{
int ms, ps, nop, np, rempages, i, j, x, y,pa, offset;
int s[10], fno[10][20];
printf("\nEnter the memory size --");
```

```
scanf("%d",&ms);
printf("\nEnter the page size --");
scanf("%d",&ps);
nop = ms/ps;
printf("\nThe no. of pages available in memory are --%d ",nop);
printf("\nEnter number of processes --");
scanf("%d",&np);
rempages = nop;
for(i=1;i<=np;i++)
{
printf("\nEnter no. of pages required for p[%d]--",i);
scanf("%d",&s[i]);
if(s[i] >rempages)
{
printf("\nMemory is Full");
break;
}
rempages = rempages -s[i];
printf("\nEnter pagetable for p[%d] ---",i);
for(j=0;j<s[i];j++)
scanf("%d",&fno[i][j]); }
printf("\nEnter Logical Address to find Physical Address ");
printf("\nEnter process no. and pagenumber and offset --");
scanf("%d %d %d",&x,&y, &offset);
if(x>np || y>=s[i] || offset>=ps)
printf("\nInvalid Process or Page Number or offset");
else
{
pa=fno[x][y]*ps+offset;
printf("\nThe Physical Address is --%d",pa); }
return (0); }

//FIFO page replacement algo
#include<stdio.h>
int main( )
{
int i,j,n,a[50],frame[10],no,k,avail,count=0;
printf("\n ENTER THE NUMBER OF PAGES:\n");
scanf("%d",&n);
printf("\n ENTER THE PAGE NUMBER :\n");
for(i=1;i<=n;i++)
scanf("%d",&a[i]);
printf("\n ENTER THE NUMBER OF FRAMES :");
scanf("%d",&no);
for(i=0;i<no;i++)
frame[i]= -1;
j=0;
printf("\tREF STRING\t PAGE FRAMES\n");
```

```
for(i=1;i<=n;i++)
{
printf("%d\t\t",a[i]);
avail=0;
for(k=0;k<no;k++)
if(frame[k]==a[i])
avail=1;
if (avail==0)
{
frame[j]=a[i];
j=(j+1)%no;
count++;
for(k=0;k<no;k++)
printf("%d\t",frame[k]);
}
printf("\n");
}
printf("PAGE FAULT IS %d",count);
return 0;
}

//LRU page replacemnet
#include<stdio.h>
int findLRU(int time[], int n)
{
int i, minimum = time[0], pos = 0;
for(i = 1; i < n; ++i)
{
if(time[i] < minimum)
{
minimum = time[i];
pos = i;
}
}
return pos;
}
int main( )
{
int no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j,
pos, faults = 0;
printf("Enter number of frames: ");
scanf("%d", &no_of_frames);
printf("Enter number of pages: ");
scanf("%d", &no_of_pages);
printf("Enter 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])
{
counter++;
time[j] = counter;
flag1 = flag2 = 1;
break;
}
}
if(flag1 == 0)
{
for(j = 0; j < no_of_frames; ++j)
{
if(frames[j] == -1)
{
counter++;
faults++;
frames[j] = pages[i];
time[j] = counter;
flag2 = 1;
break;
}
}
}
if(flag2 == 0)
{
pos = findLRU(time, no_of_frames);
counter++;
faults++;
frames[pos] = pages[i];
time[pos] = counter;
}
printf("\n");
for(j = 0; j < no_of_frames; ++j)
printf("%d\t", frames[j]);
}
printf("\n\nTotal Page Faults = %d", faults);
return 0;
}

//Threading
#include<stdio.h>
#include<string.h>
```

```
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
pthread_t tid[2];
int counter;
void* doSomeThing(void *arg)
{ unsigned long i = 0;
counter += 1;
printf("\n Job %d started\n", counter);
for(i=0; i<(0xFFFFFFFF);i++);
printf("\n Job %d finished\n", counter);
return NULL;
}
int main(void)
{
int i = 0;
int err;
while(i < 2)
{
err = pthread_create(&(tid[i]), NULL, &doSomeThing, NULL);
if (err != 0)
printf ("\ncan't create thread :[%s]", strerror(err));
i++; }
pthread_join(tid[0], NULL);
pthread_join(tid[1], NULL);
return 0;
}
//semaphore
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<time.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/sem.h>
#define num_loops 5
union semun
{
int val;
struct semid_ds *buf;
short *array;
};
int main(int argc,char *argv[]){
int semset_id;
union semun sem_val;
int child_pid;
int i;
struct sembuf sem_op;
```

```
int rc;
struct timespec delay;
semset_id=semget(IPC_PRIVATE,1,0600);
if(semset_id==-1) {
perror("semget");
exit(1); }
printf("SEMAPHORE SET CREATED SEMAPHORE SET ID%d\n",semset_id);
sem_val.val=0;
rc=semctl(semset_id,0,SETVAL,sem_val);
child_pid=fork( );
switch(child_pid) {
case 1:perror("fork");exit(1);
case 0:for(i=0;i<num_loops;i++) {
 sem_op.sem_num=0;
 sem_op.sem_op=-1;
 sem_op.sem_flg=0;
 semop(semset_id,&sem_op,1);
 printf("Consumer consumed item %d\n",i);
 fflush(stdout); }
break;
default:for(i=0;i<num_loops;i++)
{
printf("Producer produced item %d\n",i);
fflush(stdout);
sem_op.sem_num=0;
sem_op.sem_op=1;
sem_op.sem_flg=0;
semop(semset_id,&sem_op,1);
if(rand( )>3*(RAND_MAX/4)) {
delay.tv_sec=0;
delay.tv_sec=10;
nanosleep(&delay,NULL);
} }break;
} return 0;}
```