1

Fork() system call

#include<stdio.h>

#include <unistd.h>

#include<sys/types.h>

int main()

{

int id, childid;

id=getpid();

if((childid=fork())>0)

{

printf(“\n I am in the parent process %d”,id);

printf(“\n I am in the parent process %d”,getpid());

printf(“\n I am in the parent process %d\n”,getppid());

else

{

printf(“\n I am in child process %d”,id);

printf(“\n I am in the child process %d”, getpid());

printf(“\n I am in the child process %d”, getppid());

}

}

return 0;

}

Wait() system call

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

int main()

{

int I, pid;

pid=fork();

if(pid=-1)

{

printf(“fork failed”);

exit(0);

}

else if(pid==0)

{

printf(“\n Child process starts”);

for(i=0; i<5; i++)

{

printf(“\n Child process %d is called”, i);

}

printf(“\n Child process ends”);

}

else

{

wait(0);

printf(“\n Parent process ends”);

exit(0);

}

Exec() system call

#include <stdio.h>

#include <unistd.h>

int main() {

char \*args[] = {“/bin/ls”, “-1”, NULL);

execv(“/bin/ls”, args);

printf(“This line will not be executed\n”);

return 0;

}

2

//First Come First Serve (FCFS) Scheduling Algorithm

#include<stdio.h>

#include<conio.h>

int main()

{

char pn[10][10]; int arr[10],bur[10],star[10],finish[10],tat[10],wt[10],I,n;

float totwt=0,tottat=0;

clrscr();

printf(“Enter the number of processes:”);

scanf(“%d”,&n); for(i=0;i<n;i++)

{

printf(“Enter the Process Name, Arrival Time & Burst Time:”); scanf(“%s%d%d”,&pn[i],&arr[i],&bur[i]);

}

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

{

if(i-0)

{

star[i]=arr[i];

finish[i]=star[i]+bur[i];

tat[i]=finish[i]-arr[i];

wt[i]=tat[i]-bur[i];

}

else

{

star[i]-finish[i-1];

finish[i]=star[i]+bur[i];

tat[i]-finish[i]-arr[i];

wt[i]-tat[i]-bur[i];

}

}

printf(“\nPName\tArrtime \tBurtime\tStart \tTAT\tCompleteTime\tWT”);

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

{

printf(“\n%s\t%6d\t\t%6d\t%6d\t%6d\t%6d\t\t%6d”,pn[i],arr[i],bur[i],star[i],tat[i],finish

[i],wt[i]);

totwt+=wt[i]; tottat+=tat[i];

}

totwt-totwt/n;

tottat tottat/n;

printf(“\nAverage Waiting time:%f”, totwt);

printf(“\nAverage Turn Around Time:%f”, tottat);

getch();

}

b.// Shortest Job First (SJF) Scheduling Algorithm

#include<stdio.h>

int main()

{   
int bt[20],p[20],wt[20],tat[20],I,j,n,total=0,totalT=0,pos,temp;

float avg\_wt,avg\_tat;

printf(“Enter number of process:”);

scanf(“%d”,&n);

printf(“\nEnter Burst Time:\n”);

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

{

Printf(“p%d:”,i+1);

scanf(“%d”,&bt[i]);

p[i]=i+1;

}

//sorting of burst times

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

{

pos=I;

for(j=i+1;j<n;j++)

{

If(bt[j]<bt[pos])

pos=j;

}

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0;

//finding the waiting time of all the processes

for(i=1;i<n;i++)

{

wt[i]=0;

for(j=0;j<I;j++)

//individual WT by adding BT of all previous completed processes

wt[i]+=bt[j];

//total waiting time

total+=wt[i];

}

//average waiting time

avg\_wt=(float)total/n;

printf(“\nProcess\t Burst Time \tWaiting Time\tTurnaround Time”);

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

{

//turnaround time of individual processes

Tat[i]=bt[i]+wt[i];

//total turnaround time

totalT+=tat[i];

printf(“\np%d\t\t%d\t\t%d\t\t%d”,p[i],bt[i],wt[i],tat[i]);

}

//average turnaround time

avg\_tat=(float)totalT/n;

printf(“\n\nAverage Waiting Time=%f”,avg\_wt);

printf(“\nAverage Turnaround Time=%f”,avg\_tat);}

3.

#include <stdio.h>

#include <stdlib.h>

int mutex = 1;

int full = 0;

int empty = 10, x = 0;

void producer()

{

--mutex;

++full;

--empty;

// Item produced

x++;

printf(“\nProducer produces item %d”, x);

++mutex;

}

void consumer()

{

--mutex;

--full;

++empty;

Printf(“\nConsumer consumes item %d”, x);

x--;

++mutex;

}

// Driver Code

int main()

{

int n, i;

printf("\n1. Press 1 for Producer \n2. Press 2 for Consumer \n3. Press 3 for Exit");

for (i=1; i > 0; i++) {

printf("\nEnter your choice:");

scanf("%d", &n);

switch (n) {

case 1:

if ((mutex = 1)

&& (empty != 0)) {

producer();

}

else {

printf("Buffer is full!");

} break;

case 2:

if ((mutex = 1)

&& (full != 0)) {

consumer();

} // Otherwise, print Buffer is empty

else

{

printf(“Buffer is empty”);

}

break;

case 3:

exit(0);

break;

}

}

}

4.

/\*Writer Process\*/

#include <stdio.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

int fd;

char buf[1024];

char\*myfifo = “/tmp/myfifo”;

mkfifo(myfifo, 0666);

printf(“Run Reader process to read the FIFO File\n”);

fd = open(myfifo, O\_WRONLY);

write(fd, “Hi”, sizeof(“Hi”));

close(fd);

unlink(myfifo);

return 0;

}

/\*Reader Process\*/

#include <fcntl.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>

#include <stdio.h>

#define MAX\_BUF 1024

int main()

{

int fd;

char \*myfifo = “/tmp/myfifo”;

char buf[MAX\_BUF];

fd = open(myfifo, O\_RDONLY);

read(fd, buf, MAX\_BUF);

printf(“Writer: %s\n”, buf);

close(fd);

return 0;

}

5.

#include <stdio.h>

#include<conio.h>

int main()

{

int Max[10][10], need[10][10], alloc[10][10], avail [10], completed[10], safeSequence[10]; int p, r, i, j, process, count;

count = 0 ;

clrscr();

printf("Enter the no of processes: ");

scanf("%d", &p);

for( i = 0 i<p; i++)

completed[ i ]=0;

printf("\n\nEnter the no of resources: ");

scanf("%d", &r);

printf("\n\nEnter the Max Matrix for each process: ");

for(i = 0 ; i < pi i++)

{

printf("\nFor process %d: ", i + 1);

for(j = 0 ; ; j < r ;j++)

scanf("%d", &Max[i][j]);

}

printf("\n\nEnter the allocation for each process: ");

for(i = 0 ; i<p; i++)

{

printf("\nFor process %d: ",i + 1);

for( j = 0; j < r ;j++)

scanf(“%d”, &alloc[i][j]);

}

printf("\n\nEnter the Available Resources: ");

for( i = 0; i < r ;i+++)

scanf(“%d”,&avail[i]);

for(i=0; i < p; i++)

forr(j=0; j <r; j++)

need[i][j] = Max[i][j] - alloc[i][j];

do

{ printf("\n Max matrix:\tAllocation matrix:\n");

for( i = 0; i < p ;i++)

{

for(j=0; j <r; j++)

printf("%d ", Max[i][j]);

printf("\t\t");

for(j=0; j <r; j++) printf("%d ", alloc[i][j]);

printf("\n");

}

process = -1;

for( I = 0; I < p ;i++)

{

if(completed[ i]==0// if not completed

{

process = i

for(j=0; j <r; j++)

{

if(avail[j] < need[i][j])

{

process=-1;

break;

}

if(process l = - 1 )

{

printf(“\nProcess %d runs to completion!”, process + 1);

safeSequence[count] = process + 1;

count++;

for(j=0; j <r; j++)

{

avail[j] += alloc[process][j];

alloc[process][j] = 0;

max[process][j] = 0;

completed[process] = 1;

}

}

}

while(count != p && process l = - 1 );

if(count == p)

{

printf(“\nThe system is in a safe state!!\n”);

printf(“Safe Sequence: <”);

for( I = 0; I < p ;i++)

printf(“%d “, safeSequence[i]);

printf(“>\n”); }

else

printf(“\nThe system is in an unsafe state!!”);

getch();

}

6.

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

void main() {

int f[50], p,I, st, len, j, c, k, a;

clrscr();

for(i=0;i<50;i++)

f[i] = 0;

printf(“how many blocks already allocated: “);

scanf(“%d”,&p);

printf(“Enter blocks already allocated: “);

for(i=0;i<p;i++)

{

scanf(“%d”,&a);

} f[a] = 1;

x: printf(“Enter index starting block and length: “);

scanf(“%d%d”, &st,&len);

k = len;

if(f [st] Rightarrow0)

{

for(j=st;j<(st+k);j++)

{

if(f[j] = 0)

{

f[j] = 1

printf(“%d------🡪%d\n”,j,f[j]);

}

else

{  
printf(“%d Block is already allocated \n”,j);

k++;

}

}

printf(“%d Block is already allocated \n”,j); k++;

}

else

printf(“%d starting block is already allocated \n”,st);

printf(“Do you want to enter more file(Yes – 1/No – 0)”);

scanf(“%d”, &c);

if (c==1)

goto x;

else

exit(0);

getch();

}

7.

#include<stdio.h>

#include<conio.h>

void main()

{

int queue[20],n,head,I,j,k,seek=0,max, diff,temp,queuel [20],queue2[20],temp1=0,temp2=0;

float avg;

clrscr();

printf(“Enter the max range of disk\n”);

scanf(“%d”,&max);

printf(“Enter the initial head position\n”);

scanf(“%d”,&head);

printf(“Enter the size of queue request\n”);

scanf(“%d”,&n);

printf(“Enter the queue of disk positions to be read\n”);

for(i=1;i<=n;i++)

{

scanf(“%d”,&temp);

if(temp>=head)

{

queuel [temp1]=temp;

temp1++;

}

else

{

queue2[temp2]=temp;

temp2++;

}

}

for(i=0;i<temp1-1;i++)

{

for(j=i+1;j<temp1;j++)

{

if(queuel [i]>queuel [j])

{

temp-queuel [i];

queuel [i]=queuel [j];

queuel [j]=temp;

}

}

}

for(i=0;i<temp2-1;i++)

{

for(j=i+1;j<temp2;j++)

{

if(queue2[i]<queue2[j])

{

temp=queue2[i];

queue2[i]=queue2[j];

queue2[j]=temp;

}

}

}

for(i=1,j=0;j<temp1;i++j++)

queue[i]=queuel [j];

queue[i]=max;

for(i=temp1+2.j=0;j<temp2;i++,j++)

queue[i]=queue2[j];

queue[i]=0;

queue[0]=head;

for(j=0;j<=n+1;j++)

{

diff-abs(queue[j+1]-queue[j]);

seek+=diff;

printf("Disk head moves from %d to %d with seek %d\n",queue[j],queue[j+1],diff);

}

printf("Total seek time is %d\n", seek);

avg-seek/(float)n;

printf("Average seek time is %f\n",avg);

getch();

}

}