/\*17 bankers algorithm\*/

#include<stdio.h>

#include<stdlib.h>

int main(){

int max[10][10],need[10][10],allocation[10][10],available[10],finished[10],safeseq[10];

int pr\_cnt,res\_cnt,i,j,count=0,process;

system("clear");

printf("\nEnter the system state information\n");

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

scanf("%d",&pr\_cnt);

for(i=0;i<pr\_cnt;i++){

finished[i]=0;

}

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

scanf("%d",&res\_cnt);

printf("Enter the max matrix for all the processes:\n");

for(i=0;i<pr\_cnt;i++){

printf("Enter max info for process[%d]:",i+1);

for(j=0;j<res\_cnt;j++){

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

}

}

for(i=0;i<pr\_cnt;i++){

printf("Enter allocation info for process[%d]\n",i+1);

for(j=0;j<res\_cnt;j++){

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

}

}

printf("Enter the available resources in the system:\n");

for(i=0;i<res\_cnt;i++){

printf("\navailable[%d]=",i);

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

}

for(i=0;i<pr\_cnt;i++){

for(j=0;j<res\_cnt;j++){

need[i][j]=max[i][j]-allocation[i][j];

}

}

do{

printf("\nAvailable resources are: \t");

for(j=0;j<res\_cnt;j++)

printf("%d",available[j]);

printf("\nmax matrix:\tallocation matrix:\n");

for(i=0;i<pr\_cnt;i++){

for(j=0;j<res\_cnt;j++){

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

}

printf("\t");

for(j=0;j<res\_cnt;j++){

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

}

printf("\n");

}

process=-1;

for(i=0;i<pr\_cnt;i++){

if(finished[i]==0){

process=i;

for(j=0;j<res\_cnt;j++){

if(available[j]<need[i][j]){

process=-1;

break;

}}}

if(process!=-1)

break;

}

if(process!=-1){

safeseq[count]=process+1;

count++;

for(j=0;j<res\_cnt;j++){

available[j]+=allocation[process][j];

allocation[process][j]=0;

max[process][j]=0;

finished[process]=1;

}

}

}while(count!=pr\_cnt && process!=-1);

if(count==pr\_cnt){

printf("\nthe system is in a safe state\n");

printf("\nSafe sequence:<");

for(i=0;i<pr\_cnt;i++)

printf("%d",safeseq[i]);

printf(">\n");

}

else{

printf("\nthe system is in unsafe state!!\n");

}

}

/\*message echo\*/

#include<stdio.h>

#include<stdlib.h>

#include<sys/ipc.h>

#include<sys/shm.h>

#include<string.h>

main(){

int pid,msgid;

char str[30],buf[30];

msgid=msgget((key\_t)68,IPC\_CREAT|0600);

pid=fork();

if(pid==0){

strcpy(str,"Message queue demo");

msgsnd(msgid,str,sizeof(str),0);

printf("Message sent: %s\n",str);

msgrcv(msgid,str,sizeof(str),0,0);

printf("Message recieved: %s\n",str);

}

else if(pid>0){

msgrcv(msgid,buf,sizeof(buf),0,0);

printf("Message recieved: %s\n",buf);

strcpy(buf,"Message echo demo");

msgsnd(msgid,buf,sizeof(buf),0);

printf("Message sent: %s\n",buf);

}

else{

perror("Error in creating message queue\n");

}

}

/\*13 message queue\*/

#include<stdio.h>

#include<stdlib.h>

#include<sys/ipc.h>

#include<sys/msg.h>

#include<string.h>

main(){

int pid,msgid;

char str[30],buf[30];

msgid=msgget((key\_t)68,IPC\_CREAT|0600);

pid=fork();

if(pid==0){

strcpy(str,"Message queue demo");

msgsnd(msgid,str,sizeof(str),0);

}

else if(pid>0){

msgrcv(msgid,buf,sizeof(buf),0,0);

printf("Message recieved: %s\n",buf);

}

else{

perror("Error in creating message queue\n");

}

}

/\*1 open file\*/

#include<stdio.h>

#include<fcntl.h>

main(){

int f;

f=open("sample1.txt",O\_RDWR|O\_CREAT,S\_IRGRP|S\_IWGRP);

if(f<0){

perror("Unable to open file");

}

else{

printf("File opened f=%d",f);

}

}

/\*16 dining philosopher\*/

#include<stdio.h>

#include<pthread.h>

#include<semaphore.h>

#define N 5

#define THINKING 2

#define HUNGRY 1

#define EATING 0

#define LEFT (phnum+N-1)%N

#define RIGHT (phnum)%N

int state[N];

int phil[N]={0,1,2,3,4};

sem\_t mutex;

sem\_t S[N];

void test(int phnum){

if(state[phnum]==HUNGRY && state[LEFT]!=EATING && state[RIGHT]!=EATING){

state[phnum]=EATING;

sleep(2);

printf("Philosopher %d takes %d fork and %d fork\n",phnum,LEFT,RIGHT);

printf("%d is eating\n",phnum);

sem\_post(&S[phnum]);

}

}

void takeFork(int phnum){

sem\_wait(&mutex);

state[phnum]=HUNGRY;

printf("%d is hungry\n",phnum);

test(phnum);

sem\_post(&mutex);

sem\_wait(&S[phnum]);

sleep(1);

}

void putFork(int phnum){

sem\_wait(&mutex);

state[phnum]=THINKING;

printf("philosopher %d putting fork %d and %d\n",phnum,LEFT,RIGHT);

printf("philosopher %d is thinking\n",phnum);

test(LEFT);

test(RIGHT);

sem\_post(&mutex);

}

void \*philosopher(void \*num){

while(1){

int \*i=num;

sleep(1);

takeFork(\*i);

sleep(1);

putFork(\*i);

}

}

int main(){

int i;

pthread\_t tid[N];

sem\_init(&mutex,0,1);

for(i=0;i<N;i++){

sem\_init(&S[i],0,0);

}

for(i=0;i<N;i++){

pthread\_create(&tid[i],NULL,philosopher,&phil[i]);

printf("philosopher %d is thinking\n",phil[i]);

}

for(i=0;i<N;i++){

pthread\_join(tid[i],NULL);

}

}

/\*11 ipc using pipe\*/

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

main(){

int a[2],pid;

char str[30],buf[30];

pipe(a);

pid=fork();

if(pid==0){

strcpy(str,"Pipedemo");

write(a[1],str,sizeof(str));

printf("Child sent %s\n",str);

}

else if(pid>0){

read(a[0],buf,sizeof(buf));

printf("Parent recieved msg is %s\n",buf);

}

else{

perror("Error in creating child process\n");

}

}

/\*14 producer consumer\*/

#include<stdio.h>

#include<stdlib.h>

#include<sys/ipc.h>

#include<sys/sem.h>

int mutex=1,full=0,empty=3,x=0;

main(){

int n;

void producer();

void consumer();

int wait();

int signal();

printf("\n1.Producer\t2.Consumer\t3.Exit\n");

while(1){

printf("\nEnter your choice:");

scanf("%d",&n);

switch(n){

case 1:if((mutex==1) && (empty!=0))

producer();

else

printf("\nBuffer is full");

break;

case 2:if((mutex==1) && (full!=0))

consumer();

else

printf("\nBuffer is empty");

break;

case 3:exit(1);

}

}

}

int wait(int s){

return (--s);

}

int signal(int s){

return (++s);

}

void producer(){

mutex=wait(mutex);

full=signal(full);

empty=wait(empty);

x++;

printf("\nProducer produced the item %d\n",x);

mutex=signal(mutex);

}

void consumer(){

mutex=wait(mutex);

full=wait(full);

empty=signal(empty);

printf("\nConsumer consumed the item %d\n",x);

x--;

mutex=signal(mutex);

}

/\*15 reader writer\*/

#include<stdio.h>

#include<pthread.h>

#include<semaphore.h>

sem\_t mutex,writeblock;

int data=0,rcount=0;

void \*reader(void \*arg){

int f;

f=((int)arg);

sem\_wait(&mutex);

rcount+=1;

if(rcount==1){

sem\_wait(&writeblock);

}

sem\_post(&mutex);

printf("data read by the reader %d is %d\n",f,data);

sleep(3);

sem\_wait(&mutex);

rcount-=1;

if(rcount==0){

sem\_post(&writeblock);

}

sem\_post(&mutex);

}

void \*writer(void \*arg){

int f;

f=((int)arg);

sem\_wait(&writeblock);

data++;

printf("data written by writer %d is %d\n",f,data);

sleep(1);

sem\_post(&writeblock);

}

int main(){

int i,b;

pthread\_t rtid[5],wtid[5];

sem\_init(&mutex,0,1);

sem\_init(&writeblock,0,1);

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

pthread\_create(&wtid[i],NULL,writer,(void \*)i);

pthread\_create(&rtid[i],NULL,reader,(void \*)i);

}

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

pthread\_join(wtid[i],NULL);

pthread\_join(rtid[i],NULL);

}

return 0;

}

/\*9 round robin\*/

#include<stdio.h>

#include<stdlib.h>

main()

{

int rt[20],ts=0,t=0,i,n,j,tott=0,tat[20],bt[20],wt[20],p[20],count=0,totwt=0;

float avgwt=0,avgtot=0;

system("clear");

printf("Enter number of processes:\n");

scanf("%d",&n);

printf("Enter the burst time of the processes\n");

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

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

rt[i]=bt[i];

}

printf("Enter time slice");

scanf("%d",&ts);

while(1){

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

if(rt[i]>0){

if(rt[i]>ts){

t+=ts;

rt[i]-=ts;

}

else{

t+=rt[i];

rt[i]=0;

tat[i]=t;

wt[i]=t-bt[i];

count++;

totwt+=wt[i];

tott+=tat[i];

}}}

if(count==n){

break;

}}

avgwt=(float)totwt/n;

avgtot=(float)tott/n;

printf("\nProcess\tbt\twt\ttat\n");

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

printf("p[%d]\t%d\t%d\t%d\n",i+1,bt[i],wt[i],tat[i]);

}

printf("\nAverage waiting time is %.2f",avgwt);

printf("\nAverage turn around time is %.2f",avgtot);

}

/\*12 ipc using shared memory\*/

#include<stdio.h>

#include<string.h>

#include<sys/ipc.h>

#include<sys/shm.h>

main()

{ int pid,shmid;

char \*p;

shmid=shmget((key\_t)67,40,IPC\_CREAT|0600);

p=(char \*)shmat(shmid,0,0);

pid=fork();

if(pid==0){

strcpy(p,"shared memory demo");

}

else if(pid>0){

printf("%s\n",p);

}

else{

perror("error\n");}

}

/\*7 fcfs\*/

#include<stdio.h>

main()

{

int i,n,bt[20],wt[20],p[10],tat[20],totwt=0,avgwt,avgtat,tott=0;

printf("enter the number of processes\n");

scanf("%d",&n);

printf("enter the burst time\n");

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

{

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

}

wt[0]=0;

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

{

wt[i]=wt[i-1]+bt[i-1];

totwt+=wt[i];

}

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

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

tott+=tat[i];

}

avgwt=(float)totwt/n;

avgtat=(float)tott/n;

printf(" p\t bt\t wt\t tat\n");

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

printf("p[%d]\t%d\t%d\t%d\n",i+1,bt[i],wt[i],tat[i]);

}

printf("average wt is %d\n",avgwt);

printf("average tat is %d\n",avgtat);

}

/\*6 evenodd\*/

#include<stdio.h>

#include<pthread.h>

void \*run();

main()

{

int i,n;

pthread\_t tid;

printf("enter the number:\n");

scanf("%d",&n);

pthread\_create(&tid,NULL,(void \*)run,&n);

pthread\_join(tid,NULL);

}

void \*run(int \*n)

{

if(\*n%2==0)

{

printf("%d is an even number\n",\*n);

}

else{

printf("%d is an odd number\n",\*n);

}

pthread\_exit(0);

}

/\*5 threads\*/

#include<stdio.h>

#include<pthread.h>

void \*run();

main()

{

int i;

pthread\_t tid;

printf("demo of thread\n");

pthread\_create(&tid,NULL,run,0);

pthread\_join(tid,NULL);

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

{

printf("inside main\n");

sleep(1);

}

}

void \*run()

{

int i;

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

{

printf("inside thread\n");

sleep(1);

}

pthread\_exit(0);

}

/\*4 sys call\*/

#include<stdio.h>

#include<stdlib.h>

#include<fcntl.h>

#include<sys/stat.h>

#define bufsize 1024

main()

{

int fd,fd1,fd2,n,n1;

char buf[bufsize];

fd1=open("s1.txt",O\_RDONLY);

fd2=open("s2.txt",O\_WRONLY|O\_CREAT|O\_TRUNC,0644);

if((fd1||fd2)<0){

perror("error while opening file");

exit(1);

}

while((n=read(fd1,buf,bufsize))>0){

n1=write(fd2,buf,n);

if(n1<0)

perror("error in writting the file");

}

//printf("file copied successfully");

close(fd1);

close(fd2);

fd=open("s2.txt",O\_RDONLY);

while(1){

n=read(fd2,buf,1);

if(n==0){

break;

}

write(1,buf,1);

}

close(fd);

}

/\*3 child process\*/

#include<stdio.h>

#include<stdlib.h>

#include<fcntl.h>

#include<sys/stat.h>

int main()

{

int pid;

pid=fork();

if(pid<0)

{

perror("enable to create child");

exit(1);

}

else if(pid==0){

printf("this is child process and its id is %d\n",getpid());

printf("child's parent id is %d\n",getppid());

}

else{

printf("this is parent process and its id is %d\n",getppid());

printf("parent's child id is %d\n",pid);

printf("parent's process id is %d\n",getppid());

wait();

}

}

/\* 8 sjf\*/

#include<stdio.h>

main()

{

int i,j,pos,temp,avgtot=0, tott=0, tat[20], bt[20],wt[20],p[20],totwt=0,avgwt,n;

printf("enter the number of processes\n");

scanf("%d",&n);

printf("enter the burst time of process");

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

{

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

p[i]=i+1;

}

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;

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

{

wt[i]=wt[i-1]+bt[i-1];

totwt=totwt+wt[i];

}

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

{

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

tott=tott+tat[i];

}

avgwt=(float)(totwt/n);

avgtot=(float)(tott/n);

printf("Process\t bt\twt\t tat\n");

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

{

printf("p[%d]\t%d\t%d\t%d\n",p[i],bt[i],wt[i],tat[i]);

}

printf("Average wait time is %d\n",avgwt);

printf("Average turn around time is %d",avgtot);

}