**1)Unix System programs using fork(),getpid(),exit(0)**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

main()

{

intpid,pid1,pid2;

pid=fork();

if(pid==-1)

{

printf(“ERRORINPROCESSCREATION\n”);

exit(1);

}

if(pid!=0)

{

pid1=getpid();

printf(“\ntheparentprocessIDis%d\n”,pid1);

}

else

{

pid2=getpid();

printf(“\nthechildprocessIDis%d\n”,pid2);

}

}

**2)FCFS CPU Scheduling algorithms**

#include <stdio.h>

int main()

{

int pid[15];

int bt[15];

int n;

printf("Enter the number of processes: ");

scanf("%d",&n);

printf("Enter process id of all the processes: ");

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

{

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

}

printf("Enter burst time of all the processes: ");

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

{

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

}

int i, wt[n];

wt[0]=0;

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

{

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

}

printf("Process ID Burst Time Waiting Time TurnAround Time\n");

float twt=0.0;

float tat= 0.0;

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

{

printf("%d\t\t", pid[i]);

printf("%d\t\t", bt[i]);

printf("%d\t\t", wt[i]);

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

printf("\n");

twt += wt[i];

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

}

float att,awt;

awt = twt/n;

att = tat/n;

printf("Avg. waiting time= %f\n",awt);

printf("Avg. turnaround time= %f",att);

}

**3)SJF CPU Scheduling algorithms**

#include<stdio.h>(sjf)

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;

}

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]=0;

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

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=(float)total/n;

printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");

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

{

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

totalT+=tat[i];

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

}

avg\_tat=(float)totalT/n;

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

printf("\nAverage Turnaround Time=%f",avg\_tat);

}

**4)PRIORITY CPU Scheduling algorithms**

#include<stdio.h>

int main()

{

int bt[20],p[20],pri[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("Enter Burst time p%d:",i+1);

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

printf("Enter Priority p%d:",i+1);

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

p[i]=i+1;

}

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

{

pos=i;

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

{

if(pri[j]<pri[pos])

pos=j;

}

temp=pri[i];

pri[i]=pri[pos];

pri[pos]=temp;

temp=pri[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]=0;

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

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=(float)total/n;

printf("\nProcess\t Burst Time \tPriority \t Waiting Time\tTurnaround Time");

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

{

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

totalT+=tat[i];

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

}

avg\_tat=(float)totalT/n;

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

printf("\nAverage Turnaround Time=%f",avg\_tat);

}

**5)producer consumer problem using semaphore**

#include<stdio.h>

intmutex=1,full=0,empty=3,x=0; main()

{

int n;

void producer(); voidconsumer();

int wait(int);

int signal(int);

printf("\n1.PRODUCER\n2.CONSUMER\n3.EXIT\n");

while(1)

{

printf("\nENTERYOURCHOICE\n");

scanf("%d",&n);

switch(n)

{

case1:

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

producer();

else

printf("BUFFERISFULL");

break;

case 2:

if((mutex==1)&&(full!=0))

consumer();

else

printf("BUFFERISEMPTY");

break;

case 3:

exit(0);

break;

}

}

}

intwait(ints)

{

return(--s); }

int signal(int s)

{

return(++s); }

voidproducer()

{

mutex=wait(mutex);

full=signal(full);

empty=wait(empty);

x++;

printf("\nproducerproducestheitem%d",x);

mutex=signal(mutex);

}

void consumer()

{

mutex=wait(mutex);

full=wait(full);

empty=signal(empty);

printf("\nconsumerconsumesitem%d",x);

x--;

mutex=signal(mutex);

}

**7)** **First fit memory allocation Paging technique**

#include<stdio.h>(firstfit)

void firstFit(int blockSize[], int m, int processSize[], int n)

{

int i, j;

int allocation[n];

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

{

allocation[i] = -1;

}

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

{

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

if (blockSize[j] >= processSize[i])

{

allocation[i] = j;

blockSize[j] -= processSize[i];

break;

}

}

}

printf("\nProcess No.\tProcess Size\tBlock no.\n");

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

{

printf(" %i\t\t\t", i+1);

printf("%i\t\t\t\t", processSize[i]);

if (allocation[i] != -1)

printf("%i", allocation[i] + 1);

else

printf("Not Allocated");

printf("\n");

}

}

int main()

{

int m;

int n;

int blockSize[] = {100, 500, 200, 300, 600};

int processSize[] = {212, 417, 112, 426};

m = sizeof(blockSize) / sizeof(blockSize[0]);

n = sizeof(processSize) / sizeof(processSize[0]);

firstFit(blockSize, m, processSize, n);

return 0 ;

}

**8)BEST FIT**

#include<stdio.h>(bestfit)

#include<process.h>

void main()

{

int a[20],p[20],i,j,n,m;

printf("Enter no of Blocks.\n");

scanf("%d",&n);

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

{

printf("Enter the %dst Block size:",i);

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

}

printf("Enter no of Process.\n");

scanf("%d",&m);

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

{

printf("Enter the size of %dst Process:",i);

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

}

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

{

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

{

if(a[i]>=p[j])

{

printf("The Process %d allocated to %d\n",j,a[i]);

p[j]=10000;

break;

}

}

}

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

{

if(p[j]!=10000)

{

printf("The Process %d is not allocated\n",j);

}

}

}

**9)Bankers algo**

**// Banker's Algorithm**

**#include <stdio.h>**

**int main()**

**{**

**// P0, P1, P2, P3, P4 are the Process names here**

**int n, m, i, j, k;**

**n = 5; // Number of processes**

**m = 3; // Number of resources**

**int alloc[5][3] = { { 0, 1, 0 }, // P0 // Allocation Matrix**

**{ 2, 0, 0 }, // P1**

**{ 3, 0, 2 }, // P2**

**{ 2, 1, 1 }, // P3**

**{ 0, 0, 2 } }; // P4**

**int max[5][3] = { { 7, 5, 3 }, // P0 // MAX Matrix**

**{ 3, 2, 2 }, // P1**

**{ 9, 0, 2 }, // P2**

**{ 2, 2, 2 }, // P3**

**{ 4, 3, 3 } }; // P4**

**int avail[3] = { 3, 3, 2 }; // Available Resources**

**int f[n], ans[n], ind = 0;**

**for (k = 0; k < n; k++) {**

**f[k] = 0;**

**}**

**int need[n][m];**

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

**for (j = 0; j < m; j++)**

**need[i][j] = max[i][j] - alloc[i][j];**

**}**

**int y = 0;**

**for (k = 0; k < 5; k++) {**

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

**if (f[i] == 0) {**

**int flag = 0;**

**for (j = 0; j < m; j++) {**

**if (need[i][j] > avail[j]){**

**flag = 1;**

**break;**

**}**

**}**

**if (flag == 0) {**

**ans[ind++] = i;**

**for (y = 0; y < m; y++)**

**avail[y] += alloc[i][y];**

**f[i] = 1;**

**}**

**}**

**}**

**}**

**int flag = 1;**

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

**{**

**if(f[i]==0)**

**{**

**flag=0;**

**printf("The following system is not safe");**

**break;**

**}**

**}**

**if(flag==1)**

**{**

**printf("Following is the SAFE Sequence\n");**

**for (i = 0; i < n - 1; i++)**

**printf(" P%d ->", ans[i]);**

**printf(" P%d", ans[n - 1]);**

**}**

**return (0);**

**}**

**10)ipc**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<string.h>

#include<sys/ipc.h>

#include<sys/shm.h>

#include<sys/types.h>

#define SEGSIZE 100

int main(int argc, char \*argv[ ])

{

int shmid,cntr; key\_t key; char \*segptr;

char buff[]="poooda ";

key=ftok(".",'s');

if((shmid=shmget(key, SEGSIZE, IPC\_CREAT | IPC\_EXCL | 0666))== -1)

{

if((shmid=shmget(key,SEGSIZE,0))==-1)

{

perror("shmget"); exit(1);

}

}

else

{

printf("Creating a new shared memory seg \n");

printf("SHMID:%d",shmid);

}

system("ipcs –m");

if((segptr=(char\*)shmat(shmid,0,0))==(char\*)-1)

{

perror("shmat"); exit(1);

}

printf("Writing data to shared memory…\n"); strcpy(segptr,buff);

printf("DONE\n");

printf("Reading data from shared memory…\n"); printf("DATA:-%s\n",segptr);

printf("DONE\n");

printf("Removing shared memory Segment…\n");

if(shmctl(shmid,IPC\_RMID,0)== -1)

printf("Can‟t Remove Shared memory Segment…\n");

else

printf("Removed Successfully");

}