**Slip 1,5,13**

Q1) #include<stdio.h>

#include<sys/types.h>

#include<unistd.h>

int main()

{

pid\_t pid;

pid=fork();

if(pid==0)

{

printf("\nI am child Process,id=%d\n",getpid());

printf("\nPriority %d,id=%d\n",nice(-7),getpid());

}

else

{

printf("\nI am Parent Process,id=%d\n",getpid());

printf("\nPriority %d,id=%d\n",nice(15),getpid());

}

return 0;

}

**Slip 2,11**

Q1) #include<stdio.h>

#include<sys/types.h>

#include<unistd.h>

int main()

{

int pid=fork();

if(pid>0)

{

printf("I'm Parent Process..\n");

printf("Id: %d\n",getpid());

printf("Hi..\n\n");

}

else if(pid==0)

{

printf("I'm Child Process..\n");

printf("Id: %d\n",getpid());

printf("Hello World..\n\n");

}

else

{

printf("Failed to create child process..");

}

return 0;

}

**Slip 4,10,12**

Q1) #include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/wait.h>

int main()

{

pid\_t pid;

pid=fork();

if(pid>0)

{

printf("\n Parent Process..");

printf("Id :%d\n",getpid());

printf("Parent Process Done...");

}

else if(pid==0)

{

printf("\n I'm Child Process..");

printf("\nId :%d\n",getpid());

printf("\nParent Id: %d\n",getpid());

sleep(5);

printf("Child Process after Parent termination...");

printf("\nId :%d\n",getpid());

}

else

{

printf("\nFailed to create the child process..");

}

}

**Slip 6,14,16**

Q1) #include<time.h>

#include<stdio.h>

int main()

{

clock\_t start\_t,end\_t;

double total\_t;

int i;

start\_t=clock();

printf("Starting of the program, start\_t =%d\n",start\_t);

printf("Going to scan a big loop, start\_t =%d\n",start\_t);

for(i=0;i<10000000;i++)

{

}

end\_t=clock();

printf("End of the big loop, end\_t =%d\n",end\_t);

total\_t=(double)(end\_t - start\_t) / CLOCKS\_PER\_SEC;

printf("Total time taken by CPU: %f\n", total\_t);

printf("Exiting of the program....\n");

return (0);

}

**Slip 7,9,15,19,20**

Q1) #include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<unistd.h>

int main()

{

pid\_t pid;

pid=fork();

if (pid<0)

{

printf(stderr,"\nFork failed\n");

return 1;

}

else if(pid == 0)

{

printf("Chile Process Id: %d\n",getpid());

execl("/bin/ls","ls","-l",NULL);

perror("\nexec failed....\n");

exit(EXIT\_FAILURE);

}

else

{

printf("Parent Process id: %d\n",getpid());

sleep(5);

printf("Parent Process: Waking up from sleep state..\n");

wait(NULL);

printf("Control back to parent..\n");

}

}

**Slip 8,18**

Q1) #include<stdio.h>

void main()

{

int n,m,i,j,k;

printf("\nHow many processes?");

scanf("%d",&n);

printf("\nEnter number of resources:");

scanf("%d",&m);

int alloc[n][m];

printf("Enter content of allocation matrix:\n");

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

{

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

{

printf("Enter element[%d,%d]:",i+1,j+1);

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

}

}

int max[n][m];

printf("Enter content of max matrix:\n");

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

{

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

{

printf("Enter element[%d,%d]:",i+1,j+1);

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

}

}

int need[n][m];

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

{

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

{

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

}

}

printf("Allocation Matrix:");

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

{

printf("\n");

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

{

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

}

}

printf("\nMax Matrix:");

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

{

printf("\n");

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

{

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

}

}

printf("\nNeed Matrix:");

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

{

printf("\n");

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

{

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

}

}

}

**Slip17**

Q1) #include<stdio.h>

int Resources(int process,int need)

{

int minR=0;

minR=process\*(need-1)+1;

return minR;

}

void main()

{

int process,need;

printf("Enter how many Processes?\n");

scanf("%d",&process);

printf("Enter needed Resources:\n");

scanf("%d",&need);

printf("Minimum Resources Required to avoid Deadlock:\n%d",Resources(process,need));

}

**Q2 for all OS slips**

**Slip 1,5,6,11,14,20**

Q2) FIFO

#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\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\t ",frame[k]);

}

printf("\n");

}

printf("Page Fault Is %d",count);

return 0;

}

**Slip 8,10,12,16,17,18**

Q2) // Optimal page replacement

#include<stdio.h>

int main()

{

int n,pg[30],fr[10];

int count[10],i,j,k,fault,f,flag,temp,current,c,dist,max,m,cnt,p,x;

fault=0;

dist=0;

k=0;

printf("Enter the total no pages:\t");

scanf("%d",&n);

printf("Enter the sequence:");

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

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

printf("\nEnter frame size:");

scanf("%d",&f);

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

{

count[i]=0;

fr[i]=-1;

}

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

{

flag=0;

temp=pg[i];

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

{

if(temp==fr[j])

{

flag=1;

break;

}

}

if((flag==0)&&(k<f))

{

fault++;

fr[k]=temp;

k++;

}

else if((flag==0)&&(k==f))

{

fault++;

for(cnt=0;cnt<f;cnt++)

{

current=fr[cnt];

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

{

if(current!=pg[c])

count[cnt]++;

else

break;

}

}

max=0;

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

{

if(count[m]>max)

{

max=count[m];

p=m;

}

}

fr[p]=temp;

}

printf("\npage %d frame\t",pg[i]);

for(x=0;x<f;x++)

{

printf("%d\t",fr[x]);

}

}

printf("\nTotal number of faults=%d",fault);

return 0;

}

**Slip 15**

Q2) //Preemptive sjf

#include <stdio.h>

int main()

{

int arrival\_time[10], burst\_time[10], temp[10];

int i, smallest, count = 0, time, limit;

double wait\_time = 0, turnaround\_time = 0, end;

float average\_waiting\_time, average\_turnaround\_time;

printf("\nEnter the Total Number of Processes:\t");

scanf("%d", &limit);

printf("\nEnter Details of %d Processes", limit);

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

{

printf("\nEnter Arrival Time:\t");

scanf("%d", &arrival\_time[i]);

printf("Enter Burst Time:\t");

scanf("%d", &burst\_time[i]);

temp[i] = burst\_time[i];

}

burst\_time[9] = 9999;

for(time = 0; count != limit; time++)

{

smallest = 9;

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

{

if(arrival\_time[i] <= time && burst\_time[i] < burst\_time[smallest] && burst\_time[i] > 0)

{

smallest = i;

}

}

burst\_time[smallest]--;

if(burst\_time[smallest] == 0)

{

count++;

end = time + 1;

wait\_time = wait\_time + end - arrival\_time[smallest] - temp[smallest];

turnaround\_time = turnaround\_time + end - arrival\_time[smallest];

}

}

average\_waiting\_time = wait\_time / limit;

average\_turnaround\_time = turnaround\_time / limit;

printf("\n\nAverage Waiting Time:%lf\n", average\_waiting\_time);

printf("Average Turnaround Time:%lf\n", average\_turnaround\_time);

return 0;

}

**Slip 7,3**

Q2) //FCFS

#include<stdio.h>

#define max 30

int main()

{

int i,j,n,bt[max],at[max],wt[max],tat[max],temp[max];

float awt=0,atat=0;

printf("Enter the number of processes");

scanf("%d",&n);

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

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

{

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

}

printf("Enter the Arrival time of the process");

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

{

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

}

temp[0]=0;

printf("Process\t\t burst time\t\tarrival time\t\t Waiting time\t\t turn around time\n");

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

{

wt[i]=0;

tat[i]=0;

temp[i+1]=temp[i]+bt[i];

wt[i]=temp[i]-at[i];

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

awt=awt+wt[i];

atat=atat+tat[i];

printf("%d\t\t%8d\t\t\t%8d\t\t%8d\t\t%8d\n",i+1,bt[i],at[i],wt[i],tat[i]);

}

awt=awt/n;

atat=atat/n;

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

printf("Average turn around time=%f",atat);

return 0;

}

**Slip 2,13**

Q2) //SJF non-preamptive

#include <stdio.h>

int main()

{

// Matrix for storing Process Id, Burst

// Time, Average Waiting Time & Average

// Turn Around Time.

int A[100][4];

int i, j, n, total = 0, index, temp;

float avg\_wt, avg\_tat;

printf("Enter number of process: ");

scanf("%d", &n);

printf("Enter Burst Time:\n");

// User Input Burst Time and alloting Process Id.

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

printf("P%d: ", i + 1);

scanf("%d", &A[i][1]);

A[i][0] = i + 1;

}

// Sorting process according to their Burst Time.

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

index = i;

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

if (A[j][1] < A[index][1])

index = j;

temp = A[i][1];

A[i][1] = A[index][1];

A[index][1] = temp;

temp = A[i][0];

A[i][0] = A[index][0];

A[index][0] = temp;

}

A[0][2] = 0;

// Calculation of Waiting Times

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

A[i][2] = 0;

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

A[i][2] += A[j][1];

total += A[i][2];

}

avg\_wt = (float)total / n;

total = 0;

printf("P BT WT TAT\n");

// Calculation of Turn Around Time and printing the

// data.

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

A[i][3] = A[i][1] + A[i][2];

total += A[i][3];

printf("P%d %d %d %d\n", A[i][0],

A[i][1], A[i][2], A[i][3]);

}

avg\_tat = (float)total / n;

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

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

}

**Slip 9**

Q2)Round robin

#include<stdio.h>

int main()

{

int i, limit, total = 0, x, counter = 0, time\_quantum;

int wait\_time = 0, turnaround\_time = 0, arrival\_time[10], burst\_time[10], temp[10];

float average\_wait\_time, average\_turnaround\_time;

printf("\nEnter Total Number of Processes:\t");

scanf("%d", &limit);

x = limit;

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

{

printf("\nEnter Details of Process[%d]\n", i + 1);

printf("Arrival Time:\t");

scanf("%d", &arrival\_time[i]);

printf("Burst Time:\t");

scanf("%d", &burst\_time[i]);

temp[i] = burst\_time[i];

}

printf("\nEnter Time Quantum:\t");

scanf("%d", &time\_quantum);

printf("\nProcess ID\t\tBurst Timet Turnaround Time\t Waiting Time\n");

for(total = 0, i = 0; x != 0;)

{

if(temp[i] <= time\_quantum && temp[i] > 0)

{

total = total + temp[i];

temp[i] = 0;

counter = 1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - time\_quantum;

total = total + time\_quantum;

}

if(temp[i] == 0 && counter == 1)

{

x--;

printf("\nProcess[%d]\t\t%d\t\t %d\t\t\t %d", i + 1, burst\_time[i], total - arrival\_time[i], total - arrival\_time[i] - burst\_time[i]);

wait\_time = wait\_time + total - arrival\_time[i] - burst\_time[i];

turnaround\_time = turnaround\_time + total - arrival\_time[i];

counter = 0;

}

if(i == limit - 1)

{

i = 0;

}

else if(arrival\_time[i + 1] <= total)

{

i++;

}

else

{

i = 0;

}

}

average\_wait\_time = wait\_time \* 1.0 / limit;

average\_turnaround\_time = turnaround\_time \* 1.0 / limit;

printf("\nAverage Waiting Time:%f", average\_wait\_time);

printf("\nnAvg Turnaround Time:%fn", average\_turnaround\_time);

return 0;

}

**Slip 4,19**

Q2)priority schedualing with non preamptive

#include <stdio.h>

void swap(int \*a,int \*b)

{

int temp=\*a;

\*a=\*b;

\*b=temp;

}

int main()

{

int n;

printf("Enter Number of Processes: ");

scanf("%d",&n);

int burst[n],priority[n],index[n];

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

{

printf("Enter Burst Time and Priority Value for Process %d: ",i+1);

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

index[i]=i+1;

}

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

{

int temp=priority[i],m=i;

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

{

if(priority[j] > temp)

{

temp=priority[j];

m=j;

}

}

swap(&priority[i], &priority[m]);

swap(&burst[i], &burst[m]);

swap(&index[i],&index[m]);

}

int t=0;

/\* printf("Order of process Execution is\n");

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

{

printf("P%d is executed from %d to %d\n",index[i],t,t+burst[i]);

t+=burst[i];

}

printf("\n");\*/

printf("Process Id\tBurst Time\tWait Time\n");

int wait\_time=0;

int total\_wait\_time = 0;

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

{

printf("P%d\t\t%d\t\t%d\n",index[i],burst[i],wait\_time);

total\_wait\_time += wait\_time;

wait\_time += burst[i];

}

float avg\_wait\_time = (float) total\_wait\_time / n;

printf("Average waiting time is %f\n", avg\_wait\_time);

int total\_Turn\_Around = 0;

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

{

total\_Turn\_Around += burst[i];

}

float avg\_Turn\_Around = total\_Turn\_Around / n;

printf("Average TurnAround Time is %f",avg\_Turn\_Around);

return 0;

}