# 1.NODE (A\* ALGORITHM)

class Node: def \_init\_(self, data, level, fval):

""" Initialize the node with the data, level of the node, and the calculated f-value """ self.data = data self.level = level self.fval = fval

def generate\_child(self):

""" Generate child nodes by moving the blank space either in the four directions {up, down, left, right} """ x, y = self.find(self.data, '\_')

# val\_list contains position values for moving the blank space in

[up, down, left, right] val\_list = [[x, y - 1], [x, y + 1], [x - 1, y], [x + 1, y]] children = [] for i in val\_list:

child = self.shuffle(self.data, x, y, i[0], i[1]) if child is not None:

child\_node = Node(child, self.level + 1, 0) children.append(child\_node)

return children

def shuffle(self, puz, x1, y1, x2, y2):

""" Move the blank space in the given direction, if the position is within limits """ if 0 <= x2 < len(self.data) and 0 <= y2 < len(self.data):

temp\_puz = self.copy(puz)

temp\_puz[x1][y1], temp\_puz[x2][y2] = temp\_puz[x2][y2], temp\_puz[x1][y1]

return temp\_puz

else:

return None

def copy(self, root):

""" Copy function to create a similar matrix of the given node """ return [row[:] for row in root]

def find(self, puz, x):

""" Find the position of the blank space """ for i in range(len(self.data)): for j in range(len(self.data)): if puz[i][j] == x:

return i, j

class Puzzle: def \_init\_(self, size):

""" Initialize the puzzle size, open and closed lists """

self.n = size self.open = [] self.closed = []

def accept(self):

""" Accepts the puzzle from the user """

puz = [] for i in range(self.n): temp = input().split(" ") puz.append(temp) return puz

def f(self, start, goal):

""" Heuristic Function to calculate f(x) = h(x) + g(x) """ return self.h(start.data, goal) + start.level

def h(self, start, goal):

""" Calculate the difference between the given puzzles """ temp = 0 for i in range(self.n): for j in range(self.n): if start[i][j] != goal[i][j] and start[i][j] != '\_':

temp += 1

return temp

def process(self):

""" Accept Start and Goal Puzzle states """ print("Enter the start state matrix:") start = self.accept() print("Enter the goal state matrix:") goal = self.accept()

start = Node(start, 0, 0) start.fval = self.f(start, goal)

# Put the start node in the open list self.open.append(start)

while True:

cur = self.open[0] print("\n | \n | \n \\\'/ \n") for row in cur.data: print(" ".join(row))

# If the difference between the current and goal node is 0, we reached the goal

if self.h(cur.data, goal) == 0:

break

for child in cur.generate\_child(): child.fval = self.f(child, goal) self.open.append(child)

self.closed.append(cur) del self.open[0]

# Sort the open list based on f-value self.open.sort(key=lambda x: x.fval) puz = Puzzle(3) puz.process()

# 2.BANKER’S ALGORITHM

// 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);

}

# 3.ROUND ROBIN SHCEDULING

//Implementation of round robin without arrival time

#include<stdio.h>

#include<conio.h>

#define max 30 int main()

{

int i,n,qt,count=0,temp,a=0,bt[max],wt[max],tat[max],rem\_bt[max]; float awt=0,atat=0; printf("Enter number of process"); scanf("%d",&n); printf("Enter burst time of process"); for(i=0;i<n;i++)

{

scanf("%d",&bt[i]); rem\_bt[i]=bt[i];

} printf("Enter quantum time"); scanf("%d",&qt); while(1) {

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

{ temp=qt; if(rem\_bt[i]==0) { count++; continue;

} if(rem\_bt[i]>qt) rem\_bt[i]=rem\_bt[i]-qt; else if(rem\_bt[i]>=0)

{ temp=rem\_bt[i]; rem\_bt[i]=0;

}

a=a+temp; tat[i]=a;

} if(n==count) break;

}

printf("process\t burst time\t waiting time\t turn arround time\n"); for(i=0;i<n;i++)

{

wt[i]=tat[i]-bt[i]; awt=awt+wt[i]; atat=atat+tat[i]; printf("%d\t%d\t\t%d\t\t%d\n",i+1,bt[i],wt[i],tat[i]);

} awt=awt/n; atat=atat/n; printf("Average waiting time=%f\n",awt); printf("Average turn arround time=%f\n",atat);

}

# 4.PRIORITY SCHEDULING

\* C program to implement priority scheduling

#include <stdio.h>

//Function to swap two variables 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);

// b is array for burst time, p for priority and index for process id int b[n],p[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",&b[i],&p[i]); index[i]=i+1;

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

{ int a=p[i],m=i;

//Finding out highest priority element and placing it at its desired position

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

{ if(p[j] > a) { a=p[j]; m=j;

}

}

//Swapping processes swap(&p[i], &p[m]); swap(&b[i], &b[m]); swap(&index[i],&index[m]);

}

// T stores the starting time of process int t=0;

//Printing scheduled process 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+b[i]); t+=b[i];

} printf("\n");

printf("Process Id Burst Time Wait Time TurnAround Time\n"); int wait\_time=0; for(int i=0;i<n;i++)

{ printf("P%d %d %d

%d\n",index[i],b[i],wait\_time,wait\_time + b[i]); wait\_time += b[i];

} return 0;

}

## 5. SJF SCEDULING WITH ARRIVAL TIME

\* C Program to Implement SJF Scheduling

#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\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);

}

# 6.FCFS WITH ARRIVAL TIME

#include<stdio.h> int main() { int p[10],at[10],bt[10],ct[10],tat[10],wt[10],i,j,temp=0,n; float awt=0,atat=0; printf("enter no of proccess you want:"); scanf("%d",&n); printf("enter %d process:",n);

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

{

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

}

printf("enter %d arrival time:",n); for(i=0;i<n;i++)

{

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

}

printf("enter %d burst time:",n); for(i=0;i<n;i++)

{

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

}

// sorting at,bt, and process according to at for(i=0;i<n;i++)

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

{ if(at[j]>at[j+1])

{

temp=p[j+1]; p[j+1]=p[j]; p[j]=temp; temp=at[j+1]; at[j+1]=at[j]; at[j]=temp; temp=bt[j+1]; bt[j+1]=bt[j]; bt[j]=temp;

}

}

}

/\* calculating 1st ct \*/ ct[0]=at[0]+bt[0]; /\* calculating 2 to n ct \*/ for(i=1;i<n;i++)

{

//when proess is ideal in between i and i+1 temp=0; if(ct[i-1]<at[i])

{

temp=at[i]-ct[i-1];

}

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

}

/\* calculating tat and wt \*/ printf("\np\t A.T\t B.T\t C.T\t TAT\t WT"); for(i=0;i<n;i++)

{ tat[i]=ct[i]-at[i]; wt[i]=tat[i]-bt[i]; atat+=tat[i]; awt+=wt[i];

} atat=atat/n; awt=awt/n; for(i=0;i<n;i++)

{

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

}

printf("\naverage turnaround time is %f",atat); printf("\naverage wating timme is %f",awt); return 0;

}

## 7.FCFS WITHOUT ARRIVAL TIME

//fcfs without arrival time

#include<stdio.h>

#include<string.h> #define max 30 int main()

{

int i,j,n,bt[max],wt[max],tat[max]; float awt=0,atat=0; printf("Enter number of process"); scanf("%d",&n); printf("Enter burst time of process"); for(i=0;i<n;i++) scanf("%d",&bt[i]); printf("process\t burst time\t waiting time\t turn arround time\n");

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

{ wt[i]=0; tat[i]=0; for(j=0;j<i;j++)

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

} tat[i]=wt[i]+bt[i]; awt=awt+wt[i]; atat=atat+tat[i];

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

} awt=awt/n; atat=atat/n; printf("Average waiting time=%f\n",awt); printf("Average turn arround time=%f\n",atat);

}

## 8. Create a child process using fork(), display parent and child process id. Child process will display the message “Hello World” and the parent process should display “Hi”

#include <stdio.h>

#include <unistd.h> #include<stdlib.h> int main() { int pid; getpid; pid=fork(); if(pid==0)

{ printf("\n Hi.., I am the child process "); printf("\n My pid is %d ",getpid());

} else { printf("\n pid of parent process is %d \n ",getpid()); }

}

## 9. Write a program to illustrate the concept of orphan process ( Using fork() and sleep())

#include <stdio.h>

#include <sys/types.h> #include <unistd.h> int main() { int pid = fork(); if (pid > 0) { printf("Parent process\n"); printf("ID : %d\n\n", getpid());

} else if (pid == 0) { printf("Child process\n"); printf("ID: %d\n", getpid()); printf("Parent -ID: %d\n\n", getppid());

sleep(20);

printf("\nChild process \n"); printf("ID: %d\n", getpid()); printf("Parent -ID: %d\n", getppid());

} else { printf("Failed to create child process");

}

return 0; }