**Dynamic Link Library:**

**B1.c**

#include<jni.h>

#include<stdio.h>

#include "B1.h"

JNIEXPORT int JNICALL Java\_B1\_add(JNIEnv \*env,jobject obj,jint a,jint b)

{

printf("\n%d+%d=%d\n",a,b,(a+b));

return;

}

JNIEXPORT int JNICALL Java\_B1\_asub(JNIEnv \*env,jobject obj,jint a,jint b)

{

printf("\n%d-%d=%d\n",a,b,(a-b));

return;

}

JNIEXPORT int JNICALL Java\_B1\_mult(JNIEnv \*env,jobject obj,jint a,jint b)

{

printf("\n%d\*%d=%d\n",a,b,(a\*b));

return;

}

JNIEXPORT int JNICALL Java\_B1\_div(JNIEnv \*env,jobject obj,jint a,jint b)

{

printf("\n%/%d=%d\n",a,b,(a/b));

return;

}

**B1.java**

import java.io.\*;

import java.util.\*;

class B1{

static{

System.loadLibrary("B1")'

}

private native int add(int a,int b);

private native int sub(int a,int b);

private native int mult(int a,int b);

private native int div(int a,int b);

public static void main(string[] args)

{

Scanner sc=new Scanner(System.in);

int a,b,ch;

System.out.println("\nEnter value of a :");

a=sc.nextInt();

System.out.println("\nEnter value of b :");

b=sc.nextInt();

do

{

System.out.println("Menu\n1.add\n2.sub\n3.mult\n4.div\nenter choice : ");

ch=sc.nextInt();

switch(ch)

{

case 1:

new B1().add(a,b);

break;

case 2:

new B1().sub(a,b);

break;

case 3:

new B1().mult(a,b);

break;

case 4:

new B1().div(a,b);

break;

default:

System.out.println("You enetered wrong choice");

}

}while(ch<5);

}

}

**Mutex and semaphore:**

#include<iostream>

using namespace std;

class synchronization

{

int mutex=1;

int empty=5;

int full=0;

int a[5];

void signal(int &x){

x++;

}

void wait(int &x){

if(x>0){

x--;

}

}

public:

void producer(){

cout<<"Empty :"<<empty<<"Full :"<<full<<"Mutex :"<<mutex<<endl;

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

cout<<"Data to be produced is :";

wait(empty);

wait(mutex);

cin>>a[full];

cout<<"Data produced is :"<<a[full]<<endl;

signal(mutex);

signal(full);

}

}

void consumer(){

cout<<"Empty :"<<empty<<"Full :"<<full<<"Mutex :"<<mutex<<endl;

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

wait(full);

wait(mutex);

cout<<"Data consumed is :"<<a[full]<<endl;

signal(mutex);

signal(empty);

}

}

};

int main(){

int ch;

synchronization s;

do{

cout<<"Menu"<<endl;

cout<<"1.producer\n2.consumer\n3.exit\nenter choice :" ;

cin>>ch;

switch(ch)

{

case 1:

s.producer();

break;

case 2:

s.consumer();

break;

case 3:

break;

default:

break;

}

}while(ch!=3);

return 0;

}

**Memory management**

**Worst** **fit**

#include<iostream>

#include<cstring>

using namespace std;

void worstfit(int blocksize[],int m,int processsize[],int n){

int allocation[n];

memset(allocation,-1,sizeof(allocation));

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

int worstidx=-1;

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

if(blocksize[j]>=processsize[i]){

if(worstidx==-1||blocksize[j]>blocksize[worstidx]){

worstidx=j;

}

}

}

if(worstidx!=-1){

allocation[i]=worstidx;

blocksize[worstidx]-=processsize[i];

}

}

cout<<"\nProcess no\tProcess size\tBlock no\n";

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

cout<<" "<<i+1<<"\t\t\t"<<processsize[i]<<"\t\t\t";

if(allocation[i]!=-1)

cout<<allocation[i]+1;

else

cout<<"Not allocated";

cout<<endl;

}

}

int main(){

int m,n;

cout<<"Enter no. of blocks : ";

cin>>m;

int blocksize[m];

cout<<"Enter size of blocks\n";

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

cout<<"Block "<<i+1<<" : ";

cin>>blocksize[i];

}

cout<<"Enter no. of processes : ";

cin>>n;

int processsize[n];

cout<<"Enter size of processes\n";

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

cout<<"Process "<<i+1<<" : ";

cin>>processsize[i];

}

worstfit(blocksize,m,processsize,n);

return 0;

}

**Best fit**

#include<iostream>

#include<cstring>

using namespace std;

void bestfit(int blocksize[],int m,int processsize[],int n){

int allocation[n];

memset(allocation,-1,sizeof(allocation));

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

int bestidx=-1;

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

if(blocksize[j]>=processsize[i]){

if(bestidx==-1||blocksize[j]<blocksize[bestidx]){

bestidx=j;

}

}

}

if(bestidx!=-1){

allocation[i]=bestidx;

blocksize[bestidx]-=processsize[i];

}

}

cout<<"\nProcess no\tProcess size\tBlock no\n";

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

cout<<" "<<i+1<<"\t\t\t"<<processsize[i]<<"\t\t\t";

if(allocation[i]!=-1)

cout<<allocation[i]+1;

else

cout<<"Not allocated";

cout<<endl;

}

}

int main(){

int m,n;

cout<<"Enter no. of blocks : ";

cin>>m;

int blocksize[m];

cout<<"Enter size of blocks\n";

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

cout<<"Block "<<i+1<<" : ";

cin>>blocksize[i];

}

cout<<"Enter no. of processes : ";

cin>>n;

int processsize[n];

cout<<"Enter size of processes\n";

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

cout<<"Process "<<i+1<<" : ";

cin>>processsize[i];

}

bestfit(blocksize,m,processsize,n);

return 0;

}

**First fit**

#include<iostream>

#include<cstring>

using namespace std;

void firstfit(int blocksize[],int m,int processsize[],int n){

int allocation[n];

memset(allocation,-1,sizeof(allocation));

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

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

if(blocksize[j]>=processsize[i]){

allocation[i]=j;

blocksize[j]-=processsize[i];

break;

}

}

}

cout<<"\nProcess no\tProcess size\tBlock no\n";

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

cout<<" "<<i+1<<"\t\t\t"<<processsize[i]<<"\t\t\t";

if(allocation[i]!=-1)

cout<<allocation[i]+1;

else

cout<<"Not allocated";

cout<<endl;

}

}

int main(){

int m,n;

cout<<"Enter no. of blocks : ";

cin>>m;

int blocksize[m];

cout<<"Enter size of blocks\n";

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

cout<<"Block "<<i+1<<" : ";

cin>>blocksize[i];

}

cout<<"Enter no. of processes : ";

cin>>n;

int processsize[n];

cout<<"Enter size of processes\n";

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

cout<<"Process "<<i+1<<" : ";

cin>>processsize[i];

}

firstfit(blocksize,m,processsize,n);

return 0;

}

**Next fit**

#include<iostream>

#include<cstring>

using namespace std;

void nextfit(int blocksize[],int m,int processsize[],int n){

int allocation[n];

fill\_n(allocation,n,-1);

int j=0;

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

int start=j;

while(true){

if(blocksize[j]>=processsize[i]){

allocation[i]=j;

blocksize[j]-=processsize[i];

j=(j+1)%m;

break;

}

j=(j+1)%m;

if(j==start){

break;

}

}

}

cout<<"\nProcess no\tProcess size\tBlock no\n";

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

cout<<" "<<i+1<<"\t\t\t"<<processsize[i]<<"\t\t\t";

if(allocation[i]!=-1)

cout<<allocation[i]+1;

else

cout<<"Not allocated";

cout<<endl;

}

}

int main(){

int m,n;

cout<<"Enter no. of blocks : ";

cin>>m;

int blocksize[m];

cout<<"Enter size of blocks\n";

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

cout<<"Block "<<i+1<<" : ";

cin>>blocksize[i];

}

cout<<"Enter no. of processes : ";

cin>>n;

int processsize[n];

cout<<"Enter size of processes\n";

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

cout<<"Process "<<i+1<<" : ";

cin>>processsize[i];

}

nextfit(blocksize,m,processsize,n);

return 0;

}

**Assembler**

**Pass 1:**

LC = 0

mnemonics = {

"STOP": ("00", "IS", 0),

"ADD": ("01", "IS", 2),

"SUB": ("02", "IS", 2),

"MUL": ("03", "IS", 2),

"MOVER": ("04", "IS", 2),

"MOVEM": ("05", "IS", 2),

"COMP": ("06", "IS", 2),

"BC": ("07", "IS", 2),

"DIV": ("08", "IS", 2),

"READ": ("09", "IS", 1),

"PRINT": ("10", "IS", 1),

"LTORG": ("05", "AD", 0),

"ORIGIN": ("03", "AD", 1),

"START": ("01", "AD", 1),

"EQU": ("04", "AD", 2),

"DS": ("01", "DL", 1),

"DC": ("02", "DL", 1),

"END": ("AD", "", 0)

}

REG = {

"AREG": 1, "BREG": 2, "CREG": 3, "DREG": 4

}

symtab = {}

pooltab = []

lit = []

symindex = 0

def print\_symtab():

print("\nSymbol Table:")

for symbol, address in symtab.items():

print(f"{symbol}\t{address[0]}")

def print\_littab():

print("\nLiteral Table:")

for literal in lit:

print(literal)

def print\_pooltab():

print("\nPool Table:")

for pool in pooltab:

print(pool)

def handle\_end():

global LC

z = 0

print("\t(AD,02)")

for x in lit:

if "\*\*" in x:

pooltab.append(z)

print("\t(AD,05)\t(DL,02)")

LC += 1

z += 1

print\_littab()

print\_pooltab()

def handle\_ltorg():

global LC

z = 0

for x in lit:

if "\*\*" in x:

pooltab.append(z)

print("\t(AD,05)\t(DL,02)")

LC += 1

z += 1

def handle\_origin(addr):

global LC

print(f"\t(AD,03)\t(C,{addr})")

LC = int(addr)

def handle\_ds(size):

global LC

print(f"\t(DL,01)\t(C,{size})")

LC += int(size)

def handle\_dc(value):

global LC

print(f"\t(DL,02)\t(C,{value})")

LC += 1

def detect\_mnemonic(words):

global LC

if words[0] == "START":

LC = int(words[1])

print(f"\t(AD,01)\t(C,{LC})")

elif words[0] == "END":

handle\_end()

elif words[0] == "LTORG":

handle\_ltorg()

elif words[0] == "ORIGIN":

handle\_origin(words[1])

elif words[0] == "DS":

handle\_ds(words[1])

elif words[0] == "DC":

handle\_dc(words[1])

else:

if words[0].isalpha():

if words[0] not in symtab:

symtab[words[0]] = (LC, "Symbol")

print(f"\t(IS,{mnemonics[words[0]][0]})")

LC += 1

def process\_assembly(file\_name):

global LC

with open(file\_name, 'r') as file:

for line in file:

line = line.strip()

if not line or line.startswith(';'):

continue

words = line.split()

print(f"Processing: {line}")

detect\_mnemonic(words)

if \_name\_ == "\_main\_":

input\_file = "example.asm"

process\_assembly(input\_file)

print\_symtab()

print\_littab()

print\_pooltab()

**Example.asm**

START 100

LOOP: LOAD A

ADD 5

STORE C

JUMP LOOP

END

**Pass 2:**

def table(file\_content, n):

for line in file\_content:

no, name, addr = line.split()

if no == n:

return addr

return "NAN"

def main():

print("\n -- ASSEMBLER PASS-2 OUTPUT --\n")

print(" LC\t <INTERMEDIATE CODE>\t\t\tLC\t <MACHINE CODE>")

ic\_lines = []

st\_lines = []

lt\_lines = []

print("Enter the Intermediate Code (type 'end' to finish):")

while True:

ic\_line = input()

if ic\_line.lower() == "end":

break

ic\_lines.append(ic\_line)

print("\nEnter the Symbol Table (type 'end' to finish):")

while True:

st\_line = input()

if st\_line.lower() == "end":

break

st\_lines.append(st\_line)

print("\nEnter the Literal Table (type 'end' to finish):")

while True:

lt\_line = input()

if lt\_line.lower() == "end":

break

lt\_lines.append(lt\_line)

print("\nProcessing...")

for ic\_line in ic\_lines:

lc, ic1, ic2, ic3 = ic\_line.split()

if ic1[1:3] == "AD" or (ic1[1:3] == "DL" and ic1[4:6] == "02"):

MC = " -No Machine Code-"

elif ic1[1:3] == "DL" and ic1[4:6] == "01":

MC = f"00\t0\t00{ic2[3:4]}"

else:

if ic1 == "(IS,00)":

MC = ic1[4:6] + "\t0\t000"

elif ic2[1] == "S":

MC = ic1[4:6] + "\t0\t" + table(st\_lines, ic2[4:5])

else:

if ic3[1] == "S":

MC = ic1[4:6] + "\t" + ic2[1] + "\t" + table(st\_lines, ic3[4:5])

else:

MC = ic1[4:6] + "\t" + ic2[1] + "\t" + table(lt\_lines, ic3[4:5])

if ic1 == "(AD,03)":

print(f" {lc}\t{ic1}\t{ic2} {ic3}\t\t\t{lc}\t{MC}")

continue

print(f" {lc}\t{ic1}\t{ic2}\t {ic3}\t\t\t{lc}\t{MC}")

if \_name\_ == "\_main\_":

main()

**Process :**

**FCFS:**

#include<iostream>

#include<algorithm>

using namespace std;

struct Process {

int id, bt, at, wt, tat, ct, start\_time;

};

bool compareArrivalTime(Process p1, Process p2) {

return p1.at < p2.at;

}

int main() {

int n;

float avg\_tat = 0, avg\_wt = 0;

cout<<"Enter no. : ";

cin >> n;

Process p[n];

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

p[i].id = i + 1;

cout<<"Enter bt for process "<<i+1<<" : ";

cin >> p[i].bt;

cout<<"Enter at for process "<<i+1<<" : ";

cin >> p[i].at;

}

sort(p, p + n, compareArrivalTime);

p[0].start\_time = p[0].at;

p[0].ct = p[0].start\_time + p[0].bt;

p[0].tat = p[0].ct - p[0].at;

p[0].wt = p[0].tat - p[0].bt;

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

p[i].start\_time = max(p[i - 1].ct, p[i].at);

p[i].ct = p[i].start\_time + p[i].bt;

p[i].tat = p[i].ct - p[i].at;

p[i].wt = p[i].tat - p[i].bt;

}

cout << "\nProcess ID | Arrival Time | Burst Time | Turnaround Time | Waiting Time\n";

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

cout << "P" << p[i].id << " | " << p[i].at << " | " << p[i].bt

<< " | " << p[i].tat << " | " << p[i].wt << endl;

}

cout << "\nGantt chart:\n";

for (int i = 0; i < n; i++) cout << "p" << p[i].id << " |";

cout << endl;

for (int i = 0; i < n; i++) cout << p[i].start\_time << "\t";

cout << p[n-1].ct << endl;

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

avg\_wt += p[i].wt;

avg\_tat += p[i].tat;

}

cout << "\nAverage turnaround time is: " << avg\_tat / n;

cout << "\nAverage waiting time is: " << avg\_wt / n;

return 0;

}

**SJF:**

#include <iostream>

#include <string>

#include <climits>

using namespace std;

void SJF\_preemptive(int n, int BT[], string p[], int AT[]) {

int remainingBT[10], Finish\_time[10], TA[10], WT[10];

int completed = 0, time = 0;

float avgTA = 0, avgWT = 0;

for (int i = 0; i < n; i++) remainingBT[i] = BT[i];

cout << "\nGantt Chart:\n";

cout << "------------------------------------------------\n|";

while (completed != n) {

int shortest = -1, minBT = INT\_MAX;

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

if (AT[i] <= time && remainingBT[i] > 0 && remainingBT[i] < minBT) {

minBT = remainingBT[i];

shortest = i;

}

}

if (shortest == -1) {

cout << " Idle |";

time++;

continue;

}

cout << " " << p[shortest] << " |";

remainingBT[shortest]--;

if (remainingBT[shortest] == 0) {

completed++;

Finish\_time[shortest] = time + 1;

TA[shortest] = Finish\_time[shortest] - AT[shortest];

WT[shortest] = TA[shortest] - BT[shortest];

avgTA += TA[shortest];

avgWT += WT[shortest];

}

time++;

}

cout << "\n------------------------------------------------\n";

cout << "0";

for (int i = 1; i <= time; i++)

cout << " " << i;

cout << endl;

cout << "\n| Process | Arrival | Burst | Turnaround | Waiting |\n";

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

cout << p[i] << "\t" << AT[i] << "\t" << BT[i] << "\t" << TA[i] << "\t" << WT[i] << "\n";

cout << "Average Turnaround Time = " << avgTA / n << " msec\n";

cout << "Average Waiting Time = " << avgWT / n << " msec\n";

}

int main() {

int n, BT[10], AT[10];

string p[10];

cout << "Enter number of processes: ";

cin >> n;

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

cout << "Process name: "; cin >> p[i];

cout << "Burst time: "; cin >> BT[i];

cout << "Arrival time: "; cin >> AT[i];

}

SJF\_preemptive(n, BT, p, AT);

return 0;

}

**Round robin:**

#include<iostream>

using namespace std;

class rr {

public:

string pr[20];

int n, tq, bt[20], at[20], tempbt[20], ttime[20], k;

string temp[20];

int comp\_time[20];

void ganttchart() {

string remainpr[20];

int remainbt[20], btime;

tempbt[0] = 0;

for (int i = 0; i < n; i++) remainpr[i] = pr[i];

for (int i = 0; i < n; i++) remainbt[i] = bt[i];

int time = 0;

int idx = 0; // index for the temp array

while (true) {

bool done = true;

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

if (remainbt[i] > 0) {

done = false;

if (remainbt[i] <= tq) {

temp[idx] = remainpr[i];

time += remainbt[i];

tempbt[idx + 1] = time;

remainbt[i] = 0;

comp\_time[i] = time;

idx++;

} else {

temp[idx] = remainpr[i];

time += tq;

tempbt[idx + 1] = time;

remainbt[i] -= tq;

idx++;

}

}

}

if (done) break; // Exit if all processes are done

}

// Print the Gantt chart

cout << "\nGantt Chart\n ";

for (int i = 0; i < idx; i++) {

cout << temp[i] << " "; // Print process names

}

cout << endl;

for (int i = 0; i <= idx; i++) {

cout << tempbt[i] << " "; // Print time slots

}

}

void turnaround() {

int sum = 0;

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

// Turnaround time = Completion time - Arrival time

int tt = comp\_time[i] - at[i];

ttime[i] = tt;

sum += tt;

cout << "Turnaround time of " << pr[i] << " is: " << tt << endl;

}

cout << "\nAverage turnaround time = " << (float)sum / n << endl;

}

void printTable() {

cout << "\nProcess ID | Arrival Time | Burst Time | Turnaround Time\n";

cout << "----------------------------------------------------------\n";

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

cout << pr[i] << " | " << at[i] << " | " << bt[i] << " | " << ttime[i] << endl;

}

}

};

int main() {

rr obj;

cout << "ROUND ROBIN\nEnter total number of processes: ";

cin >> obj.n;

for (int i = 0; i < obj.n; i++) {

cout << "Enter process[" << i + 1 << "]: ";

cin >> obj.pr[i];

cout << "Enter Arrival Time of " << obj.pr[i] << ": ";

cin >> obj.at[i];

cout << "Enter Burst Time of " << obj.pr[i] << ": ";

cin >> obj.bt[i];

}

cout << "Enter time quantum: ";

cin >> obj.tq;

obj.ganttchart(); // Generate Gantt chart

obj.turnaround(); // Calculate and print turnaround times

obj.printTable(); // Print process table

return 0;

}

**Priority:**

#include<iostream>

#define max 20

using namespace std;

int main() {

struct process {

int id, AT, TAT, BT, WT, ST, CT, priority;

} p[20];

int n, cur\_time = 0, completed = 0, min\_index = -1, is\_comp[max];

float ATA = 0;

cout << "\nEnter number of processes: ";

cin >> n;

// Input for each process

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

p[i].id = i + 1;

cout << "\nEnter burst time of p[" << i + 1 << "]: ";

cin >> p[i].BT;

cout << "\nEnter arrival time of p[" << i + 1 << "]: ";

cin >> p[i].AT;

cout << "\nEnter priority of p[" << i + 1 << "]: ";

cin >> p[i].priority;

is\_comp[i] = 0; // Mark process as not completed initially

}

// Gantt Chart

cout << "Gantt Chart" << endl;

while (completed != n) {

int min\_priority = -1;

min\_index = -1;

// Find the process with the highest priority that has arrived and is not completed

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

if (p[i].AT <= cur\_time && is\_comp[i] == 0) {

if (p[i].priority > min\_priority) {

min\_priority = p[i].priority;

min\_index = i;

}

// In case of same priority, choose the process with earliest arrival time

else if (p[i].priority == min\_priority && p[i].AT < p[min\_index].AT) {

min\_index = i;

}

}

}

if (min\_index == -1) {

cur\_time++; // If no process is ready, increment time

} else {

// Calculate start time, completion time, and turnaround time for the selected process

p[min\_index].ST = cur\_time;

p[min\_index].CT = p[min\_index].ST + p[min\_index].BT;

p[min\_index].TAT = p[min\_index].CT - p[min\_index].AT;

p[min\_index].WT = p[min\_index].TAT - p[min\_index].BT; // Calculate waiting time

is\_comp[min\_index] = 1; // Mark this process as completed

ATA += p[min\_index].TAT; // Add to total turnaround time

cur\_time = p[min\_index].CT; // Update the current time after process completes

completed++; // Increment completed processes

// Print the Gantt chart process

cout << "| P" << p[min\_index].id << "\t";

}

}

cout << "|" << endl;

cout << "0\t"; // Initial time (0) at the start of the Gantt chart

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

cout << p[i].CT << "\t"; // Print completion times

}

cout << endl;

// Print Turnaround Times

cout << "\nTURNAROUND TIME" << endl;

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

cout << "Turnaround time of p[" << i + 1 << "] is: " << p[i].TAT << endl;

}

ATA = ATA / n; // Average Turnaround Time

// Print Average Turnaround Time

cout << "\nAverage Turnaround Time: " << ATA << endl;

// Print the Process Table (ID, Arrival Time, Burst Time, Turnaround Time)

cout << "\nProcess ID | Arrival Time | Burst Time | Turnaround Time" << endl;

cout << "--------------------------------------------------------" << endl;

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

cout << "P" << p[i].id << "\t\t" << p[i].AT << "\t\t" << p[i].BT << "\t\t" << p[i].TAT << endl;

}

return 0;

}