Final

Cpu scheduling

First come first serve.

```
#include<bits/stdc++.h>
using namespace std;

int main(){

    int p[20],bt[20],wt[20],tat[20],i,n;

    float wtavg,tatavg;

    cout<<"Enter the number of processes :"<<endl;

    cin>>n;

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

        cout<<" Enter the Burst Time:"<<i<<endl;

        cin>>bt[i];
    }

    wt[0]=wtavg=0;

    tat[0]=tatavg=bt[0];
```

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

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

        tatavg=tatavg+tat[i];
    }
cout << "\nPROCESS\t\tBURST TIME\tWAITING TIME\tTURNAROUND TIME"

for(int i=0; i<n; i++)
    {
        std::cout << "\nP" << i << "\t\t" << bt[i] << "\t\t" <<
    }


  cout << "\n\n";
    cout << "Average Waiting Time --> " << wtavg/n << "\n";

   cout << "\n";
    cout << "Average Turnaround Time --> " << tatavg/n << "\n";
  cout << "\n";

    return 0;
}
```

Shortest job first

```
#include<bits/stdc++.h>
using namespace std;
```

```
int p[20], bt[20], wt[20], tat[20], i, k, n, temp;
 float wtavg, tatavg;
int main(){

cout<<"enter the number of process"<<endl;

cin>>n;

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

    p[i]=i;

    cout<<"Enter the Burst time of process "<<i<<endl;
    cin>>bt[i];
}


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

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

        if(bt[i]>bt[k]){

            temp=bt[i];
            bt[i]=bt[k];
            bt[k]=temp;

            temp=p[i];
            p[i]=p[k];
            p[k]=temp;
        }
    }
}
```

```
wt[0]=wtavg=0;
tat[0]=tatavg=bt[0];

for(i=1; i<n; i++)
 {
 wt[i] = tat[i-1];
 tat[i] = wt[i] +bt[i];
 wtavg = wtavg + wt[i];
 tatavg = tatavg + tat[i];



 }

 cout << "\nPROCESS\t\tBURST TIME\tWAITING TIME\tTURNAROUND TIME

for(int i=0; i<n; i++)
    {
        std::cout << "\nP" << i << "\t\t" << bt[i] << "\t\t" <<
    }


  cout << "\n\n";
    cout << "Average Waiting Time --> " << wtavg/n << "\n";

   cout << "\n";
    cout << "Average Turnaround Time --> " << tatavg/n << "\n";
  cout << "\n";

}
```

Priority

```
#include<bits/stdc++.h>
using namespace std;

int main(){

    int n;
    // vector<int> p(20), bt(20), pr(20), wt(20), tat(20);

    int p[20],bt[20],pr[20],wt[20],tat[20];
    float wtavg, tatavg;

    cout<<"Enter the number of processes :"<<endl;
    cin>>n;

    for(int i=0; i<n; i++){
        cout<<" Enter the Burst Time and Priority for P"<<i<<":"
        cin>>bt[i]>>pr[i];
        p[i] = i;
    }


    for(int i=0; i<n; i++){
        for(int j=i+1; j<n; j++){
            if(pr[i] > pr[j]){
                swap(pr[i], pr[j]);
                swap(bt[i], bt[j]);
                swap(p[i], p[j]);
            }
        }
    }

    wt[0] = wtavg = 0;
    tat[0] = tatavg = bt[0];

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

```
        wt[i] = tat[i-1];
        tat[i] = bt[i] + wt[i];
        wtavg = wtavg + wt[i];
        tatavg = tatavg + tat[i];
    }

    cout << "\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTU

    for(int i=0; i<n; i++){
        cout << "\nP" << p[i] << "\t\t" << pr[i] << "\t\t" << bt
    }

    cout << "\n\n";
    cout << "Average Waiting Time --> " << wtavg/n << "\n";
    cout << "\n";
    cout << "Average Turnaround Time --> " << tatavg/n << "\n";
    cout << "\n";

    return 0;
}
```

Round robin

```
#include<bits/stdc++.h>
using namespace std;

int main(){

    int n, quantum;

     int p[20],bt[20],pr[20],wt[20],tat[20],rem_bt[20];
```

```
    float wtavg, tatavg;

    cout<<"Enter the number of processes :"<<endl;
    cin>>n;

    for(int i=0; i<n; i++){
        cout<<" Enter the Burst Time for P"<<i<<":"<<endl;
        cin>>bt[i];
        rem_bt[i] = bt[i];
        p[i] = i;
    }

    cout<<"Enter the time quantum :"<<endl;
    cin>>quantum;

    int t = 0; // Current time

    // Keep traversing processes in round robin manner until all
    while (true) {
        bool done = true;

        // Traverse all processes one by one repeatedly
        for (int i = 0 ; i < n; i++) {
            // If burst time of a process is greater than 0 then
            if (rem_bt[i] > 0) {
                done = false; // There is a pending process

                if (rem_bt[i] > quantum) {
                    // Increase the value of t i.e. shows how mu
                    t += quantum;

                    // Decrease the burst_time of current proces
                    rem_bt[i] -= quantum;
                }
                // If burst time is smaller than or equal to qua
                else {
```

```
                    // Increase the value of t i.e. shows how mu
                    t = t + rem_bt[i];

                    // Waiting time is current time minus time u
                    wt[i] = t - bt[i];

                    // As the process gets fully executed make i
                    rem_bt[i] = 0;
                }
            }
        }

        // If all processes are done
        if (done == true)
          break;
    }

    // Calculate turnaround time
    for (int i = 0 ; i < n ; i++)
        tat[i] = bt[i] + wt[i];

    cout << "\nPROCESS\t\tBURST TIME\tWAITING TIME\tTURNAROUND T

    for(int i=0; i<n; i++){
        wtavg += wt[i];
        tatavg += tat[i];
        cout << "\nP" << p[i] << "\t\t" << bt[i] << "\t\t" << wt
    }

    cout << "\n\n";
    cout << "Average Waiting Time --> " << wtavg/n << "\n";
    cout << "\n";
    cout << "Average Turnaround Time --> " << tatavg/n << "\n";
    cout << "\n";
```

```
    return 0;
}
```

Memory

MFT (Multiprogramming with a Fixed number of Tasks)

```
#include<iostream>
using namespace std;

int main()
{
    int ms, bs, nob, ef,n, mp[10],tif=0;
    int i,p=0;

    cout << " Enter the total memory available (in Bytes) -- ";
    cin >> ms;  // This line was missing

    cout << " Enter the block size (in Bytes) -- ";
    cin >> bs;

    nob=ms/bs;
    ef=ms - nob*bs;
    cout << "\n Enter the number of processes -- ";
    cin >> n;
    cout << "\n";
```

```
    for(i=0; i<n; i++)
    {
        cout << " Enter memory required for process " << i+1 <<
        cin >> mp[i];
    }

    cout << "\n No. of Blocks available in memory -- " << nob;
    cout << "\n\n PROCESS\tMEMORY REQUIRED\t ALLOCATED\tINTERNAL

    for(i=0; i<n && p<nob; i++)
    {
        cout << "\n \t" << i+1 << "\t\t\t\t" << mp[i];

        if(mp[i] > bs){
            cout << "\t\t\t NO\t\t---";
        }
        else
        {
            cout << "\t\t\t YES\t\t" << bs-mp[i];
            tif = tif + bs-mp[i];
            p++;
        }
    }

    if(i<n)
        cout << "\n\n Memory is Full, Remaining Processes cannot
    cout << "\n\n Total Internal Fragmentation is " << tif;
    cout << "\n Total External Fragmentation is " << ef;

    return 0;
}
```

MVT Memory Management Technique.

```
#include<iostream>
using namespace std;

int main()
{
    int ms, n, mp[10], tif=0;
    int i,p=0;

    cout << " Enter the total memory available (in Bytes) -- ";
    cin >> ms;

    cout << "\n Enter the number of processes -- ";
    cin >> n;
    cout << "\n";

    for(i=0; i<n; i++)
    {
        cout << " Enter memory required for process " << i+1 <<
        cin >> mp[i];
    }

    cout << "\n\n PROCESS\tMEMORY REQUIRED\t ALLOCATED\tINTERNAL

    for(i=0; i<n; i++)
    {
        cout << "\n \t" << i+1 << "\t\t\t\t" << mp[i];

        if(mp[i] > ms){
            cout << "\t\t\t NO\t\t---";
        }
        else
        {
```

```
            cout << "\t\t\t YES\t\t0";
            ms = ms - mp[i];
        }
    }

    if(i<n)
        cout << "\n\n Memory is Full, Remaining Processes cannot
    cout << "\n\n Total Internal Fragmentation is " << tif;
    cout << "\n Total External Fragmentation is " << ms;

    return 0;
}
```

Implement a code to solve the Memory Management technique problem

lab report a silo

```
#include <iostream>
using namespace std;

int main()
{
    int ms, n, mp[10], tif = 0;
    int i, p = 0;

    // Hardcoded values from image
    int blocks[10] ; // Block sizes

    cout<<"enter the number of blocks"<<endl;
```

```
    cin>>n;
    //n = 4; // Number of processes

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

        cout<<"enter the number of block size"<<i<<endl;
        cin>>blocks[i];

    }

    cout<<"enter the number of process"<<endl;

    cin>>p;

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

        cout<<" memory required for process "<<i<<endl;
         cin>>mp[i];
    }


    // mp[0] = 275;
    // mp[1] = 400;
    // mp[2] = 290;
    // mp[3] = 293;

    cout << "\n\n PROCESS\t process size \t ALLOCATED\tINTERNAL

    for (i = 0; i < n; i++)
    {
        cout << "\n\t" << i + 1 << "\t" << mp[i];

        if (mp[i] > blocks[i])
        {
            cout << "\t\t NO\t\t---";
        }
```

```
        else
        {
            cout << "\t\t YES\t\t" << blocks[i] - mp[i];
            tif = tif + blocks[i] - mp[i];
        }
    }

    cout << "\n\n Total Internal Fragmentation is " << tif;

    return 0;
}
```

Contigious memory allocation

worst fit

```
#include <iostream>
#define max 25
using namespace std;

int main()
{
    int frag[max], b[max], f[max], i, j, nb, nf, temp, highest =
    int bf[max], ff[max];
    for (i = 0; i < max; i++)
    {
        b[i] = 0;
        f[i] = 0;
        frag[i] = 0;
```

```
        bf[i] = 0;
        ff[i] = 0;
    }
    cout << "\nEnter the number of blocks:";
    cin >> nb;
    cout << "Enter the number of files:";
    cin >> nf;
    cout << "\nEnter the size of the blocks:-\n";
    for (i = 1; i <= nb; i++)
    {
        cout << "Block " << i << ":";
        cin >> b[i];
    }
    cout << "Enter the size of the files:-\n";
    for (i = 1; i <= nf; i++)
    {
        cout << "File " << i << ":";
        cin >> f[i];
    }
    for (i = 1; i <= nf; i++)
    {
        for (j = 1; j <= nb; j++)
        {
            if (bf[j] != 1)
            {
                temp = b[j]-f[i];
                if (temp >= 0)
                    if (highest < temp)
                    {
                        ff[i] = j;
                        highest = temp;
                    }
            }
        }
        frag[i] = highest;
        bf[ff[i]] = 1;
```

```
        highest = 0;
    }
    cout << "\nFile_no \tFile_size \tBlock_no \tBlock_size \tFra
    for (i = 1; i <= nf; i++)
        cout << "\n"
             << i << "\t\t" << f[i] << "\t\t" << ff[i] << "\t\t
    return 0;
}
```

best fit

```
#include<iostream>
#define max 25
using namespace std;

int main()
{
    int frag[max], b[max], f[max], i, j, nb, nf, temp;
    int bf[max], ff[max];

    // Initialization of arrays to zero
    for (i = 0; i < max; i++) {
        b[i] = 0;
        f[i] = 0;
        frag[i] = 0;
        bf[i] = 0;
        ff[i] = 0;
    }

    cout << "\nEnter the number of blocks:";
    cin >> nb;
```

```
    cout << "Enter the number of files:";
    cin >> nf;

    cout << "\nEnter the size of the blocks:-\n";
    for (i = 1; i <= nb; i++) {
        cout << "Block " << i << ":";
        cin >> b[i];
    }

    cout << "Enter the size of the files:-\n";
    for (i = 1; i <= nf; i++) {
        cout << "File " << i << ":";
        cin >> f[i];
    }

    for (i = 1; i <= nf; i++) {
        int bestIdx = -1; // initialize the best index as -1
        int minFragment = 1000;  // Initialize to a large value

        for (j = 1; j <= nb; j++) {
            if (bf[j] != 1) {
                temp = b[j] - f[i];
                if (temp >= 0 && temp < minFragment) {
                    bestIdx = j;
                    minFragment = temp;
                }
            }
        }

        if (bestIdx != -1) // if a suitable block is found
        {
            ff[i] = bestIdx; // allocate the best block to the p
            frag[i] = minFragment; // calculate the internal fra
            bf[bestIdx] = 1; // mark the block as occupied
        }
    }
```

```
    cout << "\nFile_no \tFile_size \tBlock_no \tBlock_size \tFra
    for (i = 1; i <= nf; i++)
        cout << "\n" << i << "\t\t" << f[i] << "\t\t" << ff[i] <

    return 0;
}
```

First fit

```
#include<iostream>
#define max 25
using namespace std;

int main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp;
int bf[max],ff[max];
for(i=0; i<max; i++){
b[i] = 0;
f[i] = 0;
frag[i] = 0;
bf[i] = 0;
ff[i] = 0;
}
cout << "\nEnter the number of blocks:";
cin >> nb;
cout << "Enter the number of files:";
cin >> nf;
cout << "\nEnter the size of the blocks:-\n";
for(i=1; i<=nb; i++)
{
cout << "Block " << i << ":";
```

```
cin >> b[i];
}
cout << "Enter the size of the files:-\n";
for(i=1; i<=nf; i++)
{
cout << "File " << i << ":";
cin >> f[i];
}
for(i=1; i<=nf; i++)
{
for(j=1; j<=nb; j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
{
ff[i]=j; // allocate the first block that is large enough
frag[i]=temp; // calculate the internal fragmentation
bf[j]=1; // mark the block as occupied
break; // break the inner loop
}
}
}
}
cout << "\nFile_no \tFile_size \tBlock_no \tBlock_size \tFragmer
for(i=1; i<=nf; i++)
cout << "\n" << i << "\t\t" << f[i] << "\t\t" << ff[i] << "\t\t"
return 0;
}
```

Page replacement

Fifo

```
#include <iostream>
using namespace std;

int main() {
    int pageFaultCount = 0, pages[50], memory[20], memoryIndex =

    cout << "Enter number of pages:" << endl;
    cin >> numberOfPages;

    cout << "Enter the pages:" << endl;
    for (i = 0; i < numberOfPages; i++) {
        cin >> pages[i];
    }

    cout << "Enter number of frames:" << endl;
    cin >> numberOfFrames;
    for (i = 0; i < numberOfFrames; i++) {
        memory[i] = -1;
    }

    cout << "The Page Replacement Process is -->" << endl;
    for (i = 0; i < numberOfPages; i++) {
        for (j = 0; j < numberOfFrames; j++) {
            if (memory[j] == pages[i]) {
                break;
            }
        }
        if (j == numberOfFrames) {
```

```
            memory[memoryIndex] = pages[i];
            memoryIndex++;
            pageFaultCount++;
        }
        for (k = 0; k < numberOfFrames; k++) {
            cout << "\t" << memory[k];
        }
        if (j == numberOfFrames) {
            cout << "\tPage Fault No: " << pageFaultCount;
        }
        cout << endl;
        if (memoryIndex == numberOfFrames) {
            memoryIndex = 0;
        }
    }
    cout << "The number of Page Faults using FIFO is: " << pageF
    return 0;
}
```

LRU

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int m, n, position, k, l;
    int a = 0, b = 0, page_fault = 0;

    int total_frames;
    int frames[total_frames];
    int temp[total_frames];
```

```
    int total_pages;
   int pages[100] ;
cout<<"Enter number of frames"<<endl;
cin>>total_frames;

cout<<"enter number of pages"<<endl;

cin>>total_pages;

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

   cin>>pages[i];
}



   for (m = 0; m < total_frames; m++)
   {
       frames[m] = -1;
   }

   for (n = 0; n < total_pages; n++)
   {
       cout << pages[n] << ": ";
       a = 0, b = 0;
       for (m = 0; m < total_frames; m++)
       {
           if (frames[m] == pages[n])
           {
               a = 1;
               b = 1;
               break;
           }
       }
       if (a == 0)
       {
```

```
            for (m = 0; m < total_frames; m++)
            {
                if (frames[m] == -1) {
                    frames[m] = pages[n];
                    cout<<"page fault for "<<frames[m]<<endl;
                    b = 1;
                    page_fault++;
                    break;
                }
            }
        }
        if (b == 0)
        {
            for (m = 0; m < total_frames; m++)
            {
                temp[m] = 0;
            }
            for (k = n - 1, l = 1; l <= total_frames - 1; l++, k
            {
                for (m = 0; m < total_frames; m++)
                {
                    if (frames[m] == pages[k])
                    {
                        temp[m] = 1;
                    }
                }
            }
            for (m = 0; m < total_frames; m++)
            {
                if (temp[m] == 0)
                    position = m;
            }
            frames[position] = pages[n];
            cout<<"page fault for"<<frames[position]<<endl;
            page_fault++;
        }
```

```
        for (m = 0; m < total_frames; m++)
        {
            cout << frames[m] << "\t";
        }
        cout <<endl;
    }
    cout << "\nTotal Number of Page Faults:\t" << page_fault <<
    return 0;
}
```

Implement LFU page replacement algorithm.

```
#include <iostream>
using namespace std;

int main() {
    int pageFaultCount = 0, pages[50], memory[20], frequency[20]
    int numberOfPages, numberOfFrames, i, j, k;

    cout << "Enter number of pages:" << endl;
    cin >> numberOfPages;

    cout << "Enter the pages:" << endl;
    for (i = 0; i < numberOfPages; i++) {
        cin >> pages[i];
    }

    cout << "Enter number of frames:" << endl;
```

```
    cin >> numberOfFrames;

    cout << "The Page Replacement Process is -->" << endl;
    for (i = 0; i < numberOfPages; i++) {
        int page = pages[i];

        // Check if the page is in memory
        int inMemory = -1;
        for (j = 0; j < numberOfFrames; j++) {
            if (memory[j] == page) {
                inMemory = j;
                break;
            }
        }

        if (inMemory != -1) {
            // Page is in memory, update frequency
            frequency[inMemory]++;
        } else {
            // Page fault: Page is not in memory
            pageFaultCount++;

            if (i >= numberOfFrames) {
                // Memory is full, need to replace a page

                // Find the least frequently used page
                int leastFrequency = frequency[0];
                int leastUsedOrder = usedOrder[0];
                int position = 0;

                for (j = 1; j < numberOfFrames; j++) {
                    if (frequency[j] < leastFrequency || (freque
                        leastFrequency = frequency[j];
                        leastUsedOrder = usedOrder[j];
                        position = j;
                    }
```

```
                }

                // Replace with the new page
                memory[position] = page;
                frequency[position] = 1;
                usedOrder[position] = i;
            } else {
                // Memory has empty space
                memory[i] = page;
                frequency[i] = 1;
                usedOrder[i] = i;
            }
        }

        // Print the memory state after each page access
        for (k = 0; k < numberOfFrames; k++) {
            cout << "\t" << memory[k];
        }
        cout << "\tPage Fault No: " << pageFaultCount << endl;
    }

    cout << "The number of Page Faults using LFU is: " << pageFa
    return 0;
}
```