PRACTICAL - 1

Problem Statement :

Answer the following queries by executing appropriate commands on terminal:

1. How many CPU cores does your machine have?
2. How much memory does your machine have and what fraction of it is free?
3. Create a text file and write your name and roll no. in separate lines using a single command.

Output :

```
hitendra@hitendra:~$ lscpu | grep 'CPU(s)'
CPU(s):                4
On-line CPU(s) list:   0-3
NUMA node0 CPU(s):     0-3
```

```
hitendra@hitendra:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           7875        1451        4516         392        1907        5731
Swap:          7812           0        7812
```

```
hitendra@hitendra:~$ echo -e "Hitendra \n171210028" > a.txt
hitendra@hitendra:~$ head a.txt
Hitendra
171210028
```

PRACTICAL - 2

Problem Statement :

Write programs in C to demonstrate fork(), sleep(), and wait().

Source Code :

```
// demonstrating fork()

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{

      fork();
      fork();

      printf("\nHello World  %d \n",getpid());
      return 0;
}
```

Output :

```
Hello World  4641

Hello World  4643

Hello World  4642

Hello World  4644
```

Source Code :

```
// demonstrating fork()

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{

      int xy,c=5;
      c = c*5;
      xy = fork();
      if(xy)
      {
       c=c+5;
       printf("\n %d, %d \n",c,xy);
      }
      else
      {
      c = c+50;
      printf("\n %d, %d \n",c,xy);
      }
      return 0;
}
```

Output :

```
30, 4659

75, 0
```

Source Code :

```
// demonstrating sleep()

#include<stdio.h>
#include<sys/types.h>
#include<stdlib.h>
#include<unistd.h>

int main()
{

      int pid = fork();
      if(pid == -1) {

      printf("Cannot create a process\n");
      exit(1);
      }
      else if(pid == 0) {
      sleep(5);
      printf("Child process\n");

      }
      else {

      printf("Parent Process\n");
      exit(1);
      }
      return 0;
}
```

Output :

```
hitendra@hitendra:~/Documents/LAB/LAB/Operating_System/LAB_Aug26$ ./a.out
Parent Process
hitendra@hitendra:~/Documents/LAB/LAB/Operating_System/LAB_Aug26$ Child process
```

Source Code :

```
// demonstrating wait()

#include<stdio.h>
#include<sys/wait.h>
#include<stdlib.h>
#include<unistd.h>

int i = 10;

int main()
{

int pid = fork();

if(pid == 0)
{
printf("Initial value = %d\n", i);
i += 10;
printf("New Value = %d\n", i);
printf("Child process terminated\n");
}
else
{
wait(0);
printf("Value in parent process = %d\n", i);
}

return 0;
}
```

Output :

```
Initial value = 10
New Value = 20
Child process terminated
Value in parent process = 10
```

Problem Statement :

Write a C program to demonstrate the zombie and orphan process.

Source Code :

```
// Zombie Process demonstration

#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include<stdlib.h>

int main()
{

int pid = fork();

if(pid == 0)
{
printf("\nI am child process, and I am going to die.\nI am a zombie now ! \n");
exit(1);
}
else
{
sleep(10);
printf("\nI am the parent process.\nI have completed the execution.\nNow i will
reap out the child process.\nUntill now...\nIT WAS A ZOMBIE !!! \n");
}

return 0;
}
```

Output :

```
hitendra@hitendra:~/Documents/LAB/LAB/Operating_System/LAB_Aug26$ ./a.out

I am child process, and I am going to die.
I am a zombie now !

I am the parent process.
I have completed the execution.
Now i will reap out the child process.
Untill now...
IT WAS A ZOMBIE !!!
```

Source Code :

```
// Orphan Process demonstration

#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include<stdlib.h>

int main()
{

int pid = fork();

if(pid != 0)
{
printf("\nI am parent process, and I am going to die.\n\n");
}
else
{
sleep(10);
printf("\nI am the child process.\nMy parent died 10 seconds ago :( , but i was
busy executing.\nSo, I am an orphan since a few seconds.\nWaiting for Adoption
:|\n");
exit(0);
}
return 0;
}
```

Output :

```
I am parent process, and I am going to die.

hitendra@hitendra:~/Documents/LAB/LAB/Operating_System/LAB_Aug26$
I am the child process.
My parent died 10 seconds ago :( , but i was busy executing.
So, I am an orphan since a few seconds.
Waiting for Adoption :|
```

Problem Statement :

Write a C program in which main program accepts the integers to be sorted. Main program uses the fork system call to create a new process called a child process. Parent process sorts the integers using merge sort and waits for child process using wait system call to sort the integers using quick sort. Also demonstrate zombie and orphan states.

Source Code :

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>

int partition(int arr[],int l,int r)
{
 int i,j,temp,pivot;
 pivot=arr[l];
 i=l;
 j=r+1;
 do
 {
  do
   i++;
  while(arr[i]<pivot && i<=r);

  do
   j--;
  while(arr[j]>pivot);

  if(i<j)
  {
   temp=arr[i];
   arr[i]=arr[j];
   arr[j]=temp;
  }
 }
 while(i<j);

 arr[l]=arr[j];
 arr[j]=pivot;

 return j;
}

void quickSort(int arr[],int l,int r)
{
 int pivot;
 if(l<r)
 {
  pivot = partition(arr,l,r);
  quickSort(arr,l,pivot-1);
  quickSort(arr,pivot+1,r);
 }
}

void merge(int arr[],int l1,int r1,int
l2,int r2)
{
 int temp[51];
 int i,j,k;
 i=l1;
 j=l2;
 k=0;

 while(i<=r1 && j<=r2)
 {
  if(arr[i]<arr[j])
   temp[k++]=arr[i++];
  else
   temp[k++]=arr[j++];
 }

 while(i<=r1)
  temp[k++]=arr[i++];

 while(j<=r2)
  temp[k++]=arr[j++];

 for(i=l1,j=0;i<=r2;i++,j++)
  arr[i]=temp[j];
}


void mergeSort(int arr[],int l,int r)
{
 int m;
 if(l<r)
 {
  m=(l+r)/2;
  mergeSort(arr,l,m);
```

```
  mergeSort(arr,m+1,r);
  merge(arr,l,m,m+1,r);
 }
}



int main()
{
 int n,pid;
 int arr[100]; //Max size 100

 printf("\nEnter the number of elements
:");
 scanf("%d",&n);
 int i;

 printf("\nEnter the %d elements :",n);
 for(i=0;i<n;i++)
 {
  scanf("%d",&arr[i]);
 }

 pid=fork();

 if(pid==0)
 {
  printf("\nI am a child process
(PID=%d). I will perform quick sort.
\n",pid);
  quickSort(arr,0,n-1);
  printf("\nSorted Array is : ");
  int i;
  for(i=0;i<n;i++)
   printf(" %d ",arr[i]);
 }
 else
 {

  wait(0);
  printf("\nI am a Parent process
(PID=%d). I was waiting for my child
process. Now i will execute merge sort.
\n", pid);
  mergeSort(arr,0,n-1);
  printf("\nSorted Array is : ");
  for(i=0;i<n;i++)
   printf(" %d ",arr[i]);
 }

 printf("\n\n");
 return 0;
}
```

Output :

```
Enter the number of elements :6

Enter the 6 elements :4 5 1 3 2 6

I am a child process (PID=0). I will perform quick sort.

Sorted Array is :  1  2  3  4  5  6


I am a Parent process (PID=4816). I was waiting for my child process. Now i will execute merge sort.

Sorted Array is :  1  2  3  4  5  6
```

Problem Statement :

Write a C program to execute processes from bottom up which are created using fork () system call?

Source Code :

```
// Zombie Process demonstration

#include<stdio.h>
#include<sys/wait.h>
#include<sys/types.h>
#include<stdlib.h>
#include<unistd.h>

int main()
{
fork();
fork();
int pid = fork();
if(pid==0)
{
   printf("\n Child #%d\n",pid);
}
else
{
  wait(0);
  printf("\n Parent #%d\n",pid);
}
return 0;
}
```

Output :

```
Child #0
Child #0


Child #0
Child #0

Parent #4836

Parent #4840

Parent #4837

Parent #4839
```

PRACTICAL - 3

Problem Statement :

Write a program to implement the First Come First Serve CPU scheduling algorithms.

Source Code :

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

struct process
{
    int pid,burst_time,arrival_time;

    bool operator<(const process& p) const
    {
        return this->arrival_time < p.arrival_time;
    }
};

int main()
{

    int i,pno;
    cout<<"\n Enter number of processes : ";
    cin>>pno;

    set<struct process> p;
    struct process temp;
    cout<<"\n Enter the details for "<<pno<<" processes <id  arrival_time
burst_time> : \n";

    for(i=0; i<pno; i++)
    {
        cout<<"\n     Process "<<i+1<<" : ";
        cin>>temp.pid>>temp.arrival_time>>temp.burst_time;
        p.insert(temp);
    }

    set<struct process>::iterator it;

    float wt,tat;
    int st=0;
    wt=tat=0.0;
```

```
    int waiting_time, turn_around_time, completion_time;

    cout
<<"\n\n"<<setw(5)<<"PID"<<setw(5)<<"AT"<<setw(5)<<"BT"<<setw(5)<<"CT"<<setw(5)<
<"TAT"<<setw(5)<<"WT"<< endl;

    for (it = p.begin(); it != p.end(); it++)
    {
        waiting_time = max((st - (*it).arrival_time),0);
        turn_around_time = (*it).burst_time + waiting_time;
        completion_time = turn_around_time + (*it).arrival_time;
        st += (*it).burst_time;
        wt+=waiting_time;
        tat+=turn_around_time;
        cout
<<"\n"<<setw(5)<<(*it).pid<<setw(5)<<(*it).arrival_time<<setw(5)<<(*it).burst_t
ime<<setw(5)<<completion_time<<setw(5)<<turn_around_time<<setw(5)<<waiting_time
<< endl;
    }

    float avg_wt,avg_tat;

    avg_wt = wt/pno;
    avg_tat = tat/pno;

    cout<<"\n\n Average Waiting Time     : "<<avg_wt;
    cout<<"\n Average Turn Around Time : "<<avg_tat;
    cout<<"\n\n";

    return 0;
}
```

Output :

```
Enter the details for 4 processes <id  arrival_time  burst_time> :

    Process 1 : 0 0 5

    Process 2 : 1 1 3

    Process 3 : 2 2 8

    Process 4 : 3 3 6


 PID   AT   BT   CT  TAT   WT

   0    0    5    5    5    0

   1    1    3    8    7    4

   2    2    8   16   14    6

   3    3    6   22   19   13


Average Waiting Time     : 5.75
Average Turn Around Time : 11.25
```

Problem Statement :

Write a program to implement the Shortest Job First CPU scheduling algorithms.

Source Code :

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

struct process
{
    int pid,burst_time,arrival_time;
    bool done;
    bool operator<(const process& p) const
    {
        return this->burst_time < p.burst_time;
    }
};

int main()
{

    int pno,nop;
    cout<<"\n Enter number of processes : ";
    cin>>pno;
    nop=pno;
    vector<struct process> job;
    set<struct process> p;
    struct process temp;
    cout<<"\n Enter the details for "<<pno<<" processes <id  arrival_time
burst_time> : \n";

    for(int i=0; i<pno; i++)
    {
        cout<<"\n     Process "<<i+1<<" : ";
        cin>>temp.pid>>temp.arrival_time>>temp.burst_time;
        job.push_back(temp);
    }

    set<struct process>::iterator it;
    vector<struct process>::iterator i;

    float wt,tat;
    wt=tat=0.0;

    int waiting_time, turn_around_time, completion_time;
```

```
    cout
<<"\n\n"<<setw(5)<<"PID"<<setw(5)<<"AT"<<setw(5)<<"BT"<<setw(5)<<"CT"<<setw(5)<
<"TAT"<<setw(5)<<"WT"<< endl;

    int j=0,t;
    while(pno>0)
    {
        for (i = job.begin(); i != job.end(); i++)
        {

            if( (*i).arrival_time <= j && !(*i).done)
            {
                p.insert((*i));
                (*i).done = true;
            }
        }

        it=p.begin();
        temp=(*it);
        p.erase(it);
        j+=temp.burst_time;
        pno--;
        completion_time = j;
        turn_around_time = completion_time - temp.arrival_time;
        waiting_time = turn_around_time - temp.burst_time;

        wt+=waiting_time;
        tat+=turn_around_time;

        cout
<<"\n"<<setw(5)<<temp.pid<<setw(5)<<temp.arrival_time<<setw(5)<<temp.burst_time
<<setw(5)<<completion_time<<setw(5)<<turn_around_time<<setw(5)<<waiting_time<<
endl;

    }

    float avg_wt,avg_tat;
    avg_wt = wt / nop;
    avg_tat = tat / nop;

    cout<<"\n\n Average Waiting Time     : "<<avg_wt;
    cout<<"\n Average Turn Around Time : "<<avg_tat;
    cout<<"\n\n";

    return 0;
}
```

Output :

```
Enter number of processes : 4

Enter the details for 4 processes <id  arrival_time  burst_time> :

    Process 1 : 1 2 3

    Process 2 : 2 0 4

    Process 3 : 3 4 2

    Process 4 : 4 5 4


 PID   AT   BT   CT  TAT   WT

   2    0    4    4    4    0

   3    4    2    6    2    0

   1    2    3    9    7    4

   4    5    4   13    8    4


Average Waiting Time     : 2
Average Turn Around Time : 5.25
```

Problem Statement :

Write a program to implement the Shortest Remaining Job First CPU scheduling algorithms.

Source Code :

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

struct process
{
    int pid,burst_time,arrival_time,rt;
    bool done;
    bool operator<(const process& p) const
    {
        return this->rt < p.rt;
    }
};

int main()
{

    int pno,nop;
    cout<<"\n Enter number of processes : ";
    cin>>pno;
    nop=pno;
    vector<struct process> job;
    set<struct process> p;
    struct process temp;
    cout<<"\n Enter the details for "<<pno<<" processes <id  arrival_time
burst_time> : \n";

    for(int i=0; i<pno; i++)
    {
        cout<<"\n     Process "<<i+1<<" : ";
        cin>>temp.pid>>temp.arrival_time>>temp.burst_time;
        temp.rt = temp.burst_time;
        job.push_back(temp);
    }

    set<struct process>::iterator it;
    vector<struct process>::iterator i;
    float wt,tat;
    wt=tat=0.0;
    int waiting_time, turn_around_time, completion_time;
```

```
cout<<"\n\n"<<setw(5)<<"PID"<<setw(5)<<"AT"<<setw(5)<<"BT"<<setw(5)<<"CT"<<setw
(5)<<"TAT"<<setw(5)<<"WT"<< endl;

    int j=0,t;
    while(pno>0)
    {
        for (i = job.begin(); i != job.end(); i++)
        {
            if( (*i).arrival_time <= j && !(*i).done)
            {
                p.insert((*i));
                (*i).done = true;
            }
        }
        if(!p.empty())
        {
            it=p.begin();
            temp=(*it);
            p.erase(it);
            temp.rt--;

            if(temp.rt <= 0)
            {
                pno--;
                completion_time = j + 1;
                turn_around_time = completion_time - temp.arrival_time;
                waiting_time = turn_around_time - temp.burst_time;
                wt+=waiting_time;
                tat+=turn_around_time;

cout<<"\n"<<setw(5)<<temp.pid<<setw(5)<<temp.arrival_time<<setw(5)<<temp.burst_
time<<setw(5)<<completion_time<<setw(5)<<turn_around_time<<setw(5)<<waiting_tim
e<< endl;
         }
         else
            p.insert(temp);
        }
       j++;
    }
    float avg_wt,avg_tat;
    avg_wt = wt / nop;    avg_tat = tat / nop;
    cout<<"\n\n Average Waiting Time     : "<<avg_wt;
    cout<<"\n Average Turn Around Time : "<<avg_tat;
    cout<<"\n\n";
   return 0;
}
```

Output :

```
Enter number of processes : 4

Enter the details for 4 processes <id  arrival_time  burst_time> :

    Process 1 : 1 1 6

    Process 2 : 2 1 8

    Process 3 : 3 2 7

    Process 4 : 4 3 3


 PID   AT   BT   CT  TAT   WT

   4    3    3    6    3    0

   1    1    6   10    9    3

   3    2    7   17   15    8

   2    1    8   25   24   16


Average Waiting Time     : 6.75
Average Turn Around Time : 12.75
```

Problem Statement :

Write a program to implement the Round Robin CPU scheduling algorithms with time slice = 2.

Source Code :

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

struct process
{
    int pid,burst_time,arrival_time,rt;
    bool done;
};

int main()
{
    int i,pno,nop,q = 2;
    cout<<"\n Enter number of processes : ";
    cin>>pno;
    nop = pno;
    vector<struct process> p;
    struct process temp;
    cout<<"\n Enter the details for "<<pno<<" processes <id  arrival_time
burst_time> : \n";

    for(i=0; i<pno; i++)
    {
        cout<<"\n     Process "<<i+1<<" : ";
        cin>>temp.pid>>temp.arrival_time>>temp.burst_time;
        temp.rt= temp.burst_time;
        p.push_back(temp);
    }

    int it=0;
    float wt,tat;
    int st=0;
    wt=tat=0.0;
    int waiting_time, turn_around_time, completion_time,cs=0;
cout<<"\n\n"<<setw(5)<<"PID"<<setw(5)<<"AT"<<setw(5)<<"BT"<<setw(5)<<"CT"<<setw
(5)<<"TAT"<<setw(5)<<"WT"<< endl;
    int j=0;
    bool  flag = false;
    while(pno>0)
    {
        if(p[it].rt <= q && p[it].rt > 0)
        {
```

```
            j+=p[it].rt;
            p[it].rt = 0;
            flag = true;
        }
        else if(p[it].rt>0)
        {
            j += q;
            p[it].rt-=q;
        }

        if(p[it].rt==0 && flag)
        {
            pno--;

            completion_time = j;
            turn_around_time = completion_time - p[it].arrival_time;
            waiting_time = turn_around_time - p[it].burst_time;
            wt+=waiting_time;
            tat+=turn_around_time;

cout<<"\n"<<setw(5)<<p[it].pid<<setw(5)<<p[it].arrival_time<<setw(5)<<p[it].bur
st_time<<setw(5)<<completion_time<<setw(5)<<turn_around_time<<setw(5)<<waiting_
time<< endl;

        flag = false;
        }
        if (it == nop-1)
        {
            it =0;
        }
        else if( p[it+1].arrival_time <= j)
        {
            it++;
        }
        else
        {
            it = 0;
        }
    }

    float avg_wt,avg_tat;
    avg_wt = wt/nop; avg_tat = tat/nop;
    cout<<"\n\n Average Waiting Time     : "<<avg_wt;
    cout<<"\n Average Turn Around Time : "<<avg_tat;
    cout<<"\n\n";
    return 0;
}
```

Output :

```
Enter number of processes : 4

Enter the details for 4 processes <id  arrival_time  burst_time> :

    Process 1 : 1 0 9

    Process 2 : 2 1 5

    Process 3 : 3 2 3

    Process 4 : 4 3 4


 PID   AT   BT   CT  TAT   WT

   3    2    3   13   11    8

   4    3    4   15   12    8

   2    1    5   18   17   12

   1    0    9   21   21   12


Average Waiting Time     : 10
Average Turn Around Time : 15.25
```

PRACTICAL - 4

Problem Statement :

Write a program to implement the Priority Based CPU scheduling algorithms.

Priority	P.No	Arrival Time	Burst Time
4	1	4	6
7	2	6	3
6	3	3	4
6	4	2	2
1	5	1	3
3	6	2	2

Source Code :

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

struct process
{
    int pid,burst_time,arrival_time,rt,priority;
    bool done;
    bool operator<(const process& p) const
    {
        if(this->priority == p.priority)
        {
            return this->arrival_time < p.arrival_time;
        }
        return this->priority < p.priority;
    }
};

int main()
{

    int pno,nop;
    cout<<"\n Enter number of processes : ";
    cin>>pno;
    nop=pno;
    vector<struct process> job;
```

```
    set<struct process> p;
    struct process temp;
    cout<<"\n Enter the details for "<<pno<<" processes <id  priority
arrival_time  burst_time> : \n";

    for(int i=0; i<pno; i++)
    {
        cout<<"\n     Process "<<i+1<<" : ";
        cin>>temp.pid>>temp.priority>>temp.arrival_time>>temp.burst_time;
        temp.rt = temp.burst_time;
        job.push_back(temp);
    }

    set<struct process>::iterator it;
    vector<struct process>::iterator i;
    float wt,tat;
    wt=tat=0.0;
    int waiting_time, turn_around_time, completion_time;

cout<<"\n\n"<<setw(5)<<"PID"<<setw(5)<<"PR"<<setw(5)<<"AT"<<setw(5)<<"BT"<<setw
(5)<<"CT"<<setw(5)<<"TAT"<<setw(5)<<"WT"<< endl;

    int j=0,t;
    while(pno>0)
    {
        for (i = job.begin(); i != job.end(); i++)
        {
            if( (*i).arrival_time <= j && !(*i).done)
            {
                p.insert((*i));
                (*i).done = true;
            }
        }

        if(!p.empty())
        {
            it=p.begin();
            temp=(*it);
            p.erase(it);
            temp.rt--;
            if(temp.rt <= 0)
            {
                pno--;
                completion_time = j;
                turn_around_time = completion_time - temp.arrival_time;
                waiting_time = turn_around_time - temp.burst_time;

                wt+=waiting_time;
                tat+=turn_around_time;
```

```
cout<<"\n"<<setw(5)<<temp.pid<<setw(5)<<temp.priority<<setw(5)<<temp.arrival_ti
me<<setw(5)<<temp.burst_time<<setw(5)<<completion_time<<setw(5)<<turn_around_ti
me<<setw(5)<<waiting_time<< endl;
            }
          else
               p.insert(temp);
        }
        j++;
    }
    float avg_wt,avg_tat;
    avg_wt = wt / nop; avg_tat = tat / nop;
    cout<<"\n\n Average Waiting Time     : "<<avg_wt;
    cout<<"\n Average Turn Around Time : "<<avg_tat;
    cout<<"\n\n";
    return 0;
}
```

Output :

```
Enter number of processes : 6

Enter the details for 6 processes <id  priority  arrival_time  burst_time> :

    Process 1 : 1 4 4 6

    Process 2 : 2 7 6 3

    Process 3 : 3 6 3 4

    Process 4 : 4 6 2 2

    Process 5 : 5 1 1 3

    Process 6 : 6 3 2 2


 PID   PR   AT   BT   CT  TAT   WT

   5    1    1    3    4    3    0

   6    3    2    2    6    4    2

   1    4    4    6   12    8    2

   4    6    2    2   14   12   10

   3    6    3    4   18   15   11

   2    7    6    3   21   15   12


Average Waiting Time     : 6.16667
Average Turn Around Time : 9.5
```

Problem Statement :

Write a program to implement the Shortest Remaining Job First CPU scheduling algorithms considering the I/O time.

P.No	Arrival Time	CPU time	I/O Time	CPU Time
1	0	5	5	2
2	3	2	22	2
3	7	8	0	0
4	25	9	2	1

Source Code :

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

struct process
{
    int pid,burst_time_1,burst_time_2,io_time,arrival_time,arrival,rt;
    bool done,io;
    bool operator<(const process& p) const
    {
        if(this->rt == p.rt)
            return this->arrival < p.arrival;
        return this->rt < p.rt;
    }
};

int main()
{
    int pno,nop;
    cout<<"\n Enter number of processes : ";
    cin>>pno;
    nop=pno;
    vector<struct process> job;
    set<struct process> p;
    struct process temp;
    cout<<"\n Enter the details for "<<pno<<" processes <id  arrival_time
burst_time_1 io_time burst_time_2> : \n";
    for(int i=0; i<pno; i++)
    {
        cout<<"\n     Process "<<i+1<<" : ";

cin>>temp.pid>>temp.arrival_time>>temp.burst_time_1>>temp.io_time>>temp.burst_t
ime_2;
```

```
        temp.rt = temp.burst_time_1 + temp.burst_time_2;
        temp.arrival = temp.arrival_time;
        temp.done = temp.io = false;
        job.push_back(temp);
    }
    set<struct process>::iterator it;
    vector<struct process>::iterator i;
    float wt,tat;
    wt=tat=0.0;
    int waiting_time, turn_around_time, completion_time;
    cout
<<"\n\n"<<setw(5)<<"PID"<<setw(5)<<"AT"<<setw(5)<<"BT1"<<setw(5)<<"IO"<<setw(5)
<<"BT2"<<setw(5)<<"CT"<<setw(5)<<"TAT"<<setw(5)<<"WT"<< endl;
    int j=0,t;
    while(pno>0)
    {
        for (i = job.begin(); i != job.end(); i++)
        {
            if( (*i).arrival_time == j && !(*i).done)
            {
                p.insert((*i));
                (*i).done = true;
            }
        }
        if(!p.empty())
        {
            it=p.begin();
            temp=(*it);
            p.erase(it);
            temp.rt--;
            if(temp.rt <= 0)
            {
                pno--;
                completion_time = j + 1;
                turn_around_time = completion_time - temp.arrival;
                waiting_time = turn_around_time -
(temp.burst_time_1+temp.burst_time_2);
                wt+=waiting_time;
                tat+=turn_around_time;
                cout
<<"\n"<<setw(5)<<temp.pid<<setw(5)<<temp.arrival_time<<setw(5)<<temp.burst_time
_1<<setw(5)<<temp.io_time<<setw(5)<<temp.burst_time_2<<setw(5)<<completion_time
<<setw(5)<<turn_around_time<<setw(5)<<waiting_time<< endl;
            }
            else if(temp.rt <= temp.burst_time_2 && !temp.io)
            {
                for (i = job.begin(); i != job.end(); i++)
                {
                    if( (*i).pid == temp.pid)
```

```
                    {
                        (*i).done = false;
                        (*i).io = true;
                        (*i).arrival_time = j + 1 + (*i).io_time;
                        (*i).rt = temp.rt;
                    }
                }
            }
            else
                p.insert(temp);
            j++;
        }
        else
            j++;
    }
    float avg_wt,avg_tat;
    avg_wt = wt / nop;
    avg_tat = tat / nop;
    cout<<"\n\n Average Waiting Time     : "<<avg_wt;
    cout<<"\n Average Turn Around Time : "<<avg_tat;
    cout<<"\n\n";
    return 0;
}
```

Output :

```
Enter number of processes : 4

Enter the details for 4 processes <id  arrival_time  burst_time_1 io_time burst_time_2> :

    Process 1 : 1 0 5 5 2

    Process 2 : 2 3 2 22 2

    Process 3 : 3 7 8 0 0

    Process 4 : 4 25 9 2 1


 PID   AT  BT1   IO  BT2   CT  TAT   WT

   1   10    5    5    2   12   12    5

   3    7    8    0    0   17   10    2

   2   29    2   22    2   31   28   24

   4   38    9    2    1   39   14    4


Average Waiting Time     : 8.75
Average Turn Around Time : 16
```

PRACTICAL - 5

Problem Statement :

Implement Readers-Writers Problem where one reader and one writer are trying to access the shared variable containing some data and eliminate the problem of synchronization.

Source Code :

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <unistd.h>
#include<sys/wait.h>
int main()
{
    int shmid, *a, *b, i, lockid, *la, *lb;
    shmid = shmget(IPC_PRIVATE, 2*sizeof(int), 0777|IPC_CREAT);
    lockid = shmget(IPC_PRIVATE, sizeof(int), 0777|IPC_CREAT);
    if (fork() == 0)
    {
        b = (int *) shmat(shmid, 0, 0);
        lb = (int *) shmat(lockid, 0, 0);
        *lb = 0;
        for( i=0; i< 10; i++)
        {
            while(*lb != 1);
            *lb = 1;
            printf("\t\t\t Child reads: %d,%d\n",b[0],b[1]);
            *lb = 0;
        }
        shmdt(b);
    }
    else
    {
        a = (int *) shmat(shmid, 0, 0);
        la = (int *) shmat(lockid, 0, 0);
        *la = 1;
        a[0] = 0;
        a[1] = 1;
        for( i=0; i< 10; i++)
        {
            while(*la != 0);
            *la = 0;
```

```
            a[0] = a[0] + a[1];
            a[1] = a[0] + a[1];
            printf("Parent writes: %d,%d\n",a[0],a[1]);
            *la = 1;
        }
        wait(0);
        shmdt(a);
        shmctl(shmid, IPC_RMID, 0);
    }
    return 0;
}
```

Output :

```
Parent writes: 1,2
                          Child reads: 1,2
Parent writes: 3,5
                          Child reads: 3,5
Parent writes: 8,13
                          Child reads: 8,13
Parent writes: 21,34
                          Child reads: 21,34
Parent writes: 55,89
                          Child reads: 55,89
Parent writes: 144,233
                          Child reads: 144,233
Parent writes: 377,610
                          Child reads: 377,610
Parent writes: 987,1597
                          Child reads: 987,1597
Parent writes: 2584,4181
                          Child reads: 2584,4181
Parent writes: 6765,10946
                          Child reads: 6765,10946
```

Problem Statement :

We have a buffer of fixed size. A producer can produce an item and can place in the buffer. A consumer can pick items and can consume them. We need to ensure that when a producer is placing an item in the buffer, then at the same time consumer should not consume any item. In this problem, buffer is the critical section. Implement the problem using given pseudocode.

Source Code :

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
int s;
int main()
{
    int shmid,shm, *p, *c, i;
    printf("\n Enter the buffer size : ");
    scanf("%d",&s);
    shmid = shmget(IPC_PRIVATE, (s+2)*sizeof(int), 0777|IPC_CREAT);
    if (fork() == 0)
    {
        c = (int *) shmat(shmid, 0, 0);
        while(1)
        {
            while (c[s] == c[s+1]);
            printf("\n Item %d is consumed : ",c[c[s+1]]);
            c[c[s+1]] = 0;
            for(i=0; i<s; i++)
                printf("  %d",c[i]);
            c[1+s] = (c[1+s] + 1) % s;
            sleep(1);
        };
        shmdt(c);
    }
    else
    {
        p = (int *) shmat(shmid, 0, 0);
        p[s]=p[s+1]=0;
        int np=0;
        for(i=0; i<s; i++)
```

```
            p[i]=0;
        while(1)
        {
            while (((p[s] + 1) % s) == p[1+s]);
            np++;
            printf("\n Item %d is produced : ",np);
            p[p[s]] = np;
            for(i=0; i<s; i++)
                printf("  %d",p[i]);
            p[s] = (p[s] + 1) % s;
            sleep(1);
        }
        shmdt(p);
        shmctl(shmid, IPC_RMID, 0);
    }
    return 0;
}
```

Output :

```
Enter the buffer size : 3


Item 1 is produced :   1  0  0
Item 1 is consumed :   0  0  0
Item 2 is produced :   0  2  0
Item 2 is consumed :   0  0  0
Item 3 is produced :   0  0  3
Item 3 is consumed :   0  0  0
Item 4 is produced :   4  0  0
Item 4 is consumed :   0  0  0
Item 5 is produced :   0  5  0
Item 5 is consumed :   0  0  0
Item 6 is produced :   0  0  6
Item 6 is consumed :   0  0  0
Item 7 is produced :   7  0  0
Item 7 is consumed :   0  0  0
Item 8 is produced :   0  8  0
Item 8 is consumed :   0  0  0
Item 9 is produced :   0  0  9
Item 9 is consumed :   0  0  0
```

PRACTICAL - 6

Problem Statement :

Write a program to implement Banker's Algorithm and find the safe sequence for following allocation.

	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	1	1	0	0	2	1	0	1	5	2	0
P_1	1	2	3	1	1	6	5	2				
P_2	1	3	6	5	2	3	6	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

Source Code :

```
#include<bits/stdc++.h>
#define process 5
#define resources 4
using namespace std;

bool is_available(int available[],int need[][resources],int p)
{
    bool flag = true;
    for(int i=0; i<resources; i++)
        flag = flag && (available[i]>=need[p][i]);

    return flag;
}

void PrintSafeSequence(vector<int> safe,bool completed[],int available[],int
allocated[][resources],int need[][resources])
{
    for(int i=0; i<process; i++)
    {
        if(!completed[i] && is_available(available,need,i))
        {
            completed[i]=true;
            for(int j=0; j<resources; j++)
```

```
            {
                available[j]+=allocated[i][j];
            }
            safe.push_back(i);
            PrintSafeSequence(safe,completed,available,allocated,need);
            safe.pop_back();
            for(int j=0; j<resources; j++)
            {
                available[j]-=allocated[i][j];
            }
            completed[i]=false;
        }
    }

    if(safe.size()==process)
    {
        cout<<"\n --> ";
        for(int i=0; i<process; i++)
        {
            cout<<" "<<safe[i];
        }
    }
}
int main()
{
    int i,j;
    int allocated[process][resources]=
{{0,1,1,0},{1,2,3,1},{1,3,6,5},{0,6,3,2},{0,0,1,4}};
    int Max[process][resources]=
{{0,2,1,0},{1,6,5,2},{2,3,6,6},{0,6,5,2},{0,6,5,6}};
    int available[resources]= {1,5,2,0};
    int need[process][resources]= {0};
    for(i=0; i<process; i++)
    {
        for(j=0; j<resources; j++)
        {
            need[i][j] = Max[i][j]-allocated[i][j];
        }
    }
    bool completed[process];
    vector<int> safe;
    memset(completed,false,sizeof(completed));
    cout<<"\n The safe sequences are : \n";
    PrintSafeSequence(safe,completed,available,allocated,need);
    cout<<"\n\n";
    return 0;
}
```

Output :

```
The safe sequences are :

-->  0 3 1 2 4
-->  0 3 1 4 2
-->  0 3 2 1 4
-->  0 3 2 4 1
-->  0 3 4 1 2
-->  0 3 4 2 1
-->  3 0 1 2 4
-->  3 0 1 4 2
-->  3 0 2 1 4
-->  3 0 2 4 1
-->  3 0 4 1 2
-->  3 0 4 2 1
-->  3 1 0 2 4
-->  3 1 0 4 2
-->  3 1 2 0 4
-->  3 1 2 4 0
-->  3 1 4 0 2
-->  3 1 4 2 0
-->  3 2 0 1 4
-->  3 2 0 4 1
-->  3 2 1 0 4
-->  3 2 1 4 0
-->  3 2 4 0 1
-->  3 2 4 1 0
-->  3 4 0 1 2
-->  3 4 0 2 1
-->  3 4 1 0 2
-->  3 4 1 2 0
-->  3 4 2 0 1
-->  3 4 2 1 0
```

PRACTICAL - 7

Problem Statement :

Consider the following page reference string :1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6

How many page faults would occur assuming three frames for FIFO,LRU and Optimal page replacement algorithms

Source Code :

```
// First In First Out

#include<bits/stdc++.h>
using namespace std;
int main()
{
    int fno=3,pno=20;
    int pages[20]= {1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6};
    int frames[3]= {0};
    int i,j,k=-1;
    for(i=0; i<pno; i++)
    {
        for(j=0; j<fno; j++)
        {
            if(frames[j]==pages[i])
                break;
        }
        if(j==fno)
        {
            k++;
            frames[k%3]=pages[i];
            cout<<"\n MIS ";
        }
        else
            cout<<"\n HIT ";
        cout<<" >> Page: "<<pages[i]<<" =>";
        for(j=0; j<fno; j++)
        {
            cout<<"  "<<frames[j];
        }
    }
    cout<<"\n\n Page Fault : "<<k + 1<<"\n\n";
    return 0;
}
```

Output :

```
MIS   >> Page: 1 =>  1  0  0
MIS   >> Page: 2 =>  1  2  0
MIS   >> Page: 3 =>  1  2  3
MIS   >> Page: 4 =>  4  2  3
HIT   >> Page: 2 =>  4  2  3
MIS   >> Page: 1 =>  4  1  3
MIS   >> Page: 5 =>  4  1  5
MIS   >> Page: 6 =>  6  1  5
MIS   >> Page: 2 =>  6  2  5
MIS   >> Page: 1 =>  6  2  1
HIT   >> Page: 2 =>  6  2  1
MIS   >> Page: 3 =>  3  2  1
MIS   >> Page: 7 =>  3  7  1
MIS   >> Page: 6 =>  3  7  6
HIT   >> Page: 3 =>  3  7  6
MIS   >> Page: 2 =>  2  7  6
MIS   >> Page: 1 =>  2  1  6
HIT   >> Page: 2 =>  2  1  6
MIS   >> Page: 3 =>  2  1  3
MIS   >> Page: 6 =>  6  1  3

Page Fault : 16
```

Source Code :

```
// Least Recently Used

#include<bits/stdc++.h>
using namespace std;
int main()
{
    int fno=3,pno=20;
    int pages[20]= {1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6};
    int frames[3]= {0};
    list<int> f;
    f.resize(fno,0);
    int i,j,k = 0,lru;
    for(i=0; i<pno; i++)
    {
        if(find(f.begin(),f.end(),pages[i])!=f.end())
        {
            cout<<"\n HIT ";
            f.remove(pages[i]);
            f.push_back(pages[i]);
        }
        else
        {
            cout<<"\n MIS ";
            k++;
            lru = f.front();
            f.pop_front();
            f.push_back(pages[i]);
            for(j=0; j<fno; j++)
            {
                if(lru == frames[j])
                {
                    frames[j] = pages[i];
                    break;
                }
            }
        }
        cout<<" >> Page: "<<pages[i]<<" =>";
        for(j=0; j<fno; j++)
                 cout<<"  "<<frames[j];
        }
    cout<<"\n\n Page Fault : "<<k<<"\n\n";
    return 0;
}
```

Output :

```
MIS   >> Page: 1 =>  1  0  0
MIS   >> Page: 2 =>  1  2  0
MIS   >> Page: 3 =>  1  2  3
MIS   >> Page: 4 =>  4  2  3
HIT   >> Page: 2 =>  4  2  3
MIS   >> Page: 1 =>  4  2  1
MIS   >> Page: 5 =>  5  2  1
MIS   >> Page: 6 =>  5  6  1
MIS   >> Page: 2 =>  5  6  2
MIS   >> Page: 1 =>  1  6  2
HIT   >> Page: 2 =>  1  6  2
MIS   >> Page: 3 =>  1  3  2
MIS   >> Page: 7 =>  7  3  2
MIS   >> Page: 6 =>  7  3  6
HIT   >> Page: 3 =>  7  3  6
MIS   >> Page: 2 =>  2  3  6
MIS   >> Page: 1 =>  2  3  1
HIT   >> Page: 2 =>  2  3  1
HIT   >> Page: 3 =>  2  3  1
MIS   >> Page: 6 =>  2  3  6

Page Fault : 15
```

Source Code :

```
// Optimal Page Replacement

#include<bits/stdc++.h>
using namespace std;
int main()
{
    int fno=3,pno=20,i,j,k = 0,opt,f;
    int pages[20]= {1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6},frames[3]= {0};

    for(i=0; i<pno; i++)
    {
        if(find(frames,frames+3,pages[i])!=frames+3)
            cout<<"\n HIT ";
        else
        {
            cout<<"\n MIS ";
            k++;
            f=INT_MIN;
            for(j=0; j<fno; j++)
            {
                if(find(pages+i,pages+20,frames[j]) != pages+20)
                    f = max(f,find(pages+i,pages+20,frames[j])-pages);
                else
                {
                    opt = frames[j];
                    break;
                }
                opt = pages[f];
            }
            for(j=0; j<fno; j++)
            {
                if(opt == frames[j])
                {
                    frames[j] = pages[i];
                    break;
                }
            }
        }
        cout<<" >> Page: "<<pages[i]<<" =>";
        for(j=0; j<fno; j++)
            cout<<"  "<<frames[j];
    }
    cout<<"\n\n Page Fault : "<<k<<"\n\n";
    return 0;
}
```

Output :

```
MIS  >> Page: 1 =>  1  0  0
MIS  >> Page: 2 =>  1  2  0
MIS  >> Page: 3 =>  1  2  3
MIS  >> Page: 4 =>  1  2  4
HIT  >> Page: 2 =>  1  2  4
HIT  >> Page: 1 =>  1  2  4
MIS  >> Page: 5 =>  1  2  5
MIS  >> Page: 6 =>  1  2  6
HIT  >> Page: 2 =>  1  2  6
HIT  >> Page: 1 =>  1  2  6
HIT  >> Page: 2 =>  1  2  6
MIS  >> Page: 3 =>  3  2  6
MIS  >> Page: 7 =>  3  7  6
HIT  >> Page: 6 =>  3  7  6
HIT  >> Page: 3 =>  3  7  6
MIS  >> Page: 2 =>  3  2  6
MIS  >> Page: 1 =>  3  2  1
HIT  >> Page: 2 =>  3  2  1
HIT  >> Page: 3 =>  3  2  1
MIS  >> Page: 6 =>  6  2  1

Page Fault : 11
```