Lab Experiment No. 1

File-related System Calls Implementation

Aim: To get acquainted with how the system calls are invoked for file-related functionalities.

1). creat () : This system call is used to create a file.

Syntax : int fd = creat(“filename” ,S_IREAD | S_IWRITE);

create.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
int main(){
int fd;
fd = creat("File1.txt", S_IREAD | S_IWRITE);
if (fd != -1){
printf("Created file opened for read/write access\n");
printf("File1 is currently empty\n");
}
else
printf("Error in opening file\n");

close(fd);
exit (0);
}
```

OS — -zsh — 80×24

```
mymac@Saurabhs-MacBook-Air OS % cd
mymac@Saurabhs-MacBook-Air ~ % cd Desktop
mymac@Saurabhs-MacBook-Air Desktop % mkdir OS
mymac@Saurabhs-MacBook-Air Desktop % cd OS
mymac@Saurabhs-MacBook-Air OS % touch create.c
mymac@Saurabhs-MacBook-Air OS % open -ev create.c
mymac@Saurabhs-MacBook-Air OS % gcc create.c
mymac@Saurabhs-MacBook-Air OS % ./a.out
Created file opened for read/write access
File1 is currently empty
mymac@Saurabhs-MacBook-Air OS %
```

2). open() : Used to open a file. Returns a file descriptor specifying the position of this opened file in table of open files for current process.

Syntax : fd = open(path , flags , mode(optional));

Flags: O_RDONLY, O_RDWR, O_APPEND, O_WRONLY.

Mode: for permissions(User,Groups, other)

3). read() : Read from the file descriptor. returns the bytes read on success. Returns 0 when end of file is reached, returns -1 when error is occurred. **Syntax :** size_t read(int fd , void * buff , size_t count);

==> It attempts to read upto count bytes from file descriptor in to the buffer buff.

opening.c

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
static char message[] = "HELLO WORLD";
int main()
{
int fd;
char buffer[80];
fd = open("file2.txt",O_RDWR | O_CREAT | O_EXCL, S_IREAD | S_IWRITE);
if (fd != -1)
{
printf("File has been opened for READ/WRITE access\n");
write(fd, message, sizeof(message));
lseek(fd, 0, 0); /* go back to the beginning of the file */
if (read(fd, buffer, sizeof(message)) == sizeof(message))
printf("\"%s\" Message has been written to file \n", buffer);
else
printf("error in reading file \n");
close (fd);
}
else
printf("File already exists\n");

exit(0);
}
```

```
mymac@Saurabhs-MacBook-Air OS % gcc opening.c
mymac@Saurabhs-MacBook-Air OS % ./a.out
File already exists
mymac@Saurabhs-MacBook-Air OS % gcc opening.c
mymac@Saurabhs-MacBook-Air OS % ./a.out
File has been opened for READ/WRITE access
"HELLO WORLD" Message has been written to file
mymac@Saurabhs-MacBook-Air OS %
```

4). lseek : Position a pointer to a specified position in a file (Used for random access when reading or writing). Reposition read/write file affect. **Syntax :** lseek(int fd , off_t offset , int whence);

seeking.c

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
int fd;
long position;
fd = open("file2.txt", O_RDONLY);
if ( fd != -1)
{
position = lseek(fd, 0, 2); /* seek 0 bytes from end-of-file */
if (position != -1)
printf("The length of file2.txt is %ld bytes.\n", position);
else
perror("lseek error");
}
else
printf("can't open file2.txt\n");
close(fd);
}
```

```
mymac@Saurabhs-MacBook-Air OS % touch seeking.c
mymac@Saurabhs-MacBook-Air OS % open -ev seeking.c
mymac@Saurabhs-MacBook-Air OS % gcc seeking.c
mymac@Saurabhs-MacBook-Air OS % ./a.out
The length of file2.txt is 12 bytes.
mymac@Saurabhs-MacBook-Air OS %
```

Lab Experiment No. 2

Implementation of FCFS & SJF CPU Scheduling Algorithms

Aim: To get the idea of how logics for FCFS and SJF CPU Scheduling Algorithms are developed.

FCFS Background:

- Simplest of all CPU Scheduling Algorithms
- Criteria: Arrival Time
- Mode: Non-Preemptive
- FCFS pick the job from the RQ, which has the lowest AT.

FCFS Code (for same arrival time):

```c
#include<stdio.h>
int main()
{
int bt[20], wt[20], tat[20], i, n;
float cwt, ctat;
printf("\nEnter the number of processes -- ");
scanf("%d",&n);
for (i = 0; i< n; i++)
{
printf("\nEnter Burst Time for Process %d -- ", i);
scanf("%d",&bt[i]);
}
wt[0] = cwt = 0;
tat[0] = ctat = bt[0];
for(i=1;i<n;i++){
wt[i] = wt[i-1] +bt[i-1];
tat[i] = tat[i-1] +bt[i];
cwt = cwt + wt[i];
ctat = ctat + tat[i];
}
printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
for(i=0;i<n;i++)
printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i], tat[i]);
printf("\nAverage Waiting Time -- %f", cwt/n);
printf("\nAverage Turnaround Time -- %f", ctat/n);

return 0;
}
```

```
Enter the number of processes -- 4

Enter Burst Time for Process 0 -- 2

Enter Burst Time for Process 1 -- 3

Enter Burst Time for Process 2 -- 4

Enter Burst Time for Process 3 -- 5
         PROCESS        BURST TIME       WAITING TIME    TURNAROUND TIME

         P0              2               0               2
         P1              3               2               5
         P2              4               5               9
         P3              5               9               14
Average Waiting Time -- 4.000000
Average Turnaround Time -- 7.500000%
saurabh@Saurabhs-MacBook-Air 2 %
```

SJF Background:

- Criteria: Burst Time
- Mode: Non-Preemptive
- SJF picks the job from the RQ, which has least BT in RQ.
- If there's only one job in RQ, the STS has to pick that job irrelevant of the scheduling criteria

SJF Code (for same arrival time):

```c
#include<stdio.h>
void findWaitingTime(int processes[], int n, int bt[], int wt[], int at[]){
    int service_time[n];
    service_time[0] = at[0];
    wt[0] = 0;
    for (int i = 1; i < n; i++) {
        service_time[i] = service_time[i - 1] + bt[i - 1];
        wt[i] = service_time[i] - at[i];
        if (wt[i] < 0) wt[i] = 0;
    }
}
void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]){
    for (int i = 0; i < n; i++)
        tat[i] = bt[i] + wt[i];
}
```

```c
void findavgTime(int processes[], int n, int bt[], int at[]){
    int wt[n], tat[n];
    findWaitingTime(processes, n, bt, wt, at);
    findTurnAroundTime(processes, n, bt, wt, tat);
    printf("Processes\tBurst Time\tArrival Time\tWaiting Time\tTurn-Around Time\tCompletion Time \n");
    int total_wt = 0, total_tat = 0;
    for (int i = 0; i < n; i++)  {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        int compl_time = tat[i] + at[i];
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t\n",
        i + 1, bt[i], at[i], wt[i], tat[i], compl_time);
    }
    printf("Average waiting time = %f", (float)total_wt / (float)n);
    printf("\nAverage turn around time = %f", (float)total_tat / (float)n);
}
int main(){
        int n;
    printf("Enter no of processes: ");
    scanf("%d", &n);
    int processes[n];
    for (int i = 0; i < n; i++)
        processes[i] = i;
    int burst_time[n];
    for (int i = 0; i < n; i++) {
        printf("Enter burst time for process %d : ", i);
        scanf("%d", &burst_time[i]);
    }
    int arrival_time[n];
    for (int i = 0; i < n; i++)
    {
        printf("Enter arrival time for process %d : ", i);
        scanf("%d", &arrival_time[i]);
    }
    findavgTime(processes, n, burst_time, arrival_time);
    return 0;
}
```

```
Enter no of processes: 4
Enter burst time for process 0 : 3
Enter burst time for process 1 : 2
Enter burst time for process 2 : 1
Enter burst time for process 3 : 4
Enter arrival time for process 0 : 0
Enter arrival time for process 1 : 1
Enter arrival time for process 2 : 2
Enter arrival time for process 3 : 3
Processes       Burst Time      Arrival Time    Waiting Time    Turn-Around Time        Completion Time
1               3               0               0               3               3
2               2               1               2               4               5
3               1               2               3               4               6
4               4               3               3               7               10
Average waiting time = 2.000000
Average turn around time = 4.500000%
```

- **SJF Code (for same arrival time):**

```c
#include <stdio.h>
int main(){
    int p[20], bt[20], wt[20], tat[20], i, k, n, temp;
    float cwt, ctat;
    printf("\nEnter the number of processes -- ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)   {
        p[i] = i;
        printf("Enter Burst Time for Process %d -- ", i);
        scanf("%d", &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] = cwt = 0;
    tat[0] = ctat = bt[0];
    for (i = 1; i < n; i++) {
        wt[i] = wt[i - 1] + bt[i - 1];
        tat[i] = tat[i - 1] + bt[i];
        cwt = cwt + wt[i];
        ctat = ctat + tat[i];
    }
    printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
    for (i = 0; i < n; i++)
        printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);
    printf("\nAverage Waiting Time -- %f", cwt / n);
    printf("\nAverage Turnaround Time -- %f", ctat / n);
    return 0;
}
```

```
Enter the number of processes -- 4
Enter Burst Time for Process 0 -- 3
Enter Burst Time for Process 1 -- 4
Enter Burst Time for Process 2 -- 2
Enter Burst Time for Process 3 -- 4

         PROCESS        BURST TIME       WAITING TIME    TURNAROUND TIME

         P2              2               0               2
         P0              3               2               5
         P1              4               5               9
         P3              4               9               13
Average Waiting Time -- 4.000000
Average Turnaround Time -- 7.250000%
```

- **SJF with Different Arrivals time :**

```cpp
#include <iostream>
using namespace std;
int mat[10][6];
void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}
void arrangeArrival(int num, int mat[][6]){
    for (int i = 0; i < num; i++) {
        for (int j = 0; j < num - i - 1; j++)    {
            if (mat[j][1] > mat[j + 1][1])   {
                for (int k = 0; k < 5; k++)
                swap(mat[j][k], mat[j + 1][k]);
            }  }}}
void completionTime(int num, int mat[][6]){
    int temp, val;
    mat[0][3] = mat[0][1] + mat[0][2];
    mat[0][5] = mat[0][3] - mat[0][1];
    mat[0][4] = mat[0][5] - mat[0][2];
    for (int i = 1; i < num; i++){
        temp = mat[i - 1][3];
        int low = mat[i][2];
        for (int j = i; j < num; j++)  {
            if (temp >= mat[j][1] && low >= mat[j][2])   {
                low = mat[j][2];
                val = j;
            }
        }
```

```
        mat[val][3] = temp + mat[val][2];
        mat[val][5] = mat[val][3] - mat[val][1];
        mat[val][4] = mat[val][5] - mat[val][2];
        for (int k = 0; k < 6; k++) swap(mat[val][k], mat[i][k]);
    }
}
int main(){
    int num, temp;
    cout << "Enter number of Process: ";
    cin >> num;
    cout << "...Enter the process ID...\n";
    for (int i = 0; i < num; i++) {
        cout << "...Process " << i + 1 << "...\n";
        cout << "Enter Process Id: ";
        cin >> mat[i][0];
        cout << "Enter Arrival Time: ";
        cin >> mat[i][1];
        cout << "Enter Burst Time: ";
        cin >> mat[i][2];
    }
    cout << "Before Arrange...\n";
    cout << "Process ID\tArrival Time\tBurst Time\n";
    for (int i = 0; i < num; i++) {
        cout << mat[i][0] << "\t\t" << mat[i][1] << "\t\t"
             << mat[i][2] << "\n";
    }
    arrangeArrival(num, mat);
    completionTime(num, mat);
    cout << "Final Result...\n";
    cout << "Process ID\tArrival Time\tBurst Time\tWaiting "
         "Time\tTurnaround Time\n";
    for (int i = 0; i < num; i++){
        cout << mat[i][0] << "\t\t" << mat[i][1] << "\t\t"
             << mat[i][2] << "\t\t" << mat[i][4] << "\t\t"
             << mat[i][5] << "\n";
    }
}
```

```
Enter number of Process: 4
...Enter the process ID...
...Process 1...
Enter Process Id: 1
Enter Arrival Time: 0
Enter Burst Time: 8
...Process 2...
Enter Process Id: 2
Enter Arrival Time: 1
Enter Burst Time: 4
...Process 3...
Enter Process Id: 3
Enter Arrival Time: 2
Enter Burst Time: 9
...Process 4...
Enter Process Id: 4
Enter Arrival Time: 3
Enter Burst Time: 4
Before Arrange...
Process ID      Arrival Time    Burst Time
1               0               8
2               1               4
3               2               9
4               3               4
Final Result...
Process ID      Arrival Time    Burst Time      Waiting Time    Turnaround Time
1               0               8               0               8
4               3               4               5               9
2               1               4               11              15
3               2               9               14              23
saurabh@Saurabhs-MacBook-Air 2 %
```

- **SJF (With Heap) :**

```
#include <iostream>
#include <algorithm>
using namespace std;
void heapify(int arr[], int n, int i, int t[]){
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    if (l == n && arr[l] > arr[largest])
        largest = l;
    if (r < n && arr[r] > arr[largest])
        largest = r;
    if (largest != i)
    {
        swap(arr[i], arr[largest]);
        swap(t[i], t[largest]);
        heapify(arr, n, largest, t);
    }
}
void heapSort(int arr[], int n, int t[]){
    for (int i = n / 2; i >= 0; i++)
    heapify(arr, n, i, t);
    for (int i = n - 1; i > 0; i--) {
        swap(arr[0], arr[i]);
        swap(t[i], t[0]);
        heapify(arr, i, 0, t);
    }
}
void printArray(int arr[], int n){
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
    cout << "/n";
}
int main(){
    int p[20], bt[20], wt[20], tat[20], i, k, n, temp;
    float cwt, ctat;
    printf("\n Enter the number of Process : ");
    scanf("%d", &n);
```

```
    for (i = 0; i < n; i++){
        p[i] = i;
        printf("Enter Burst time for processe %d : ", i);
        scanf("%d", &bt[i]);
    }
    heapSort(bt, n, p);
    wt[0] = cwt = 0;
    tat[0] = ctat = bt[0];
    for (int i = 1; i < n; i++) {
        wt[i] = wt[i - 1] + bt[i];
        tat[i] = tat[i - 1] + bt[i];
        cwt = cwt + wt[i];
        ctat = ctat + tat[i];
    }
    printf("\n\tProcess\tBurst Time \t Waiting Time \t Turnaround Time\n");
    for (int i = 0; i < n; i++){
        printf("\n\tP%d\t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);
        printf("\nAverage waiting time : %f", cwt / n);
        printf("\nAverage Turaround time : %f", ctat / n);
    }
}
```

```
Enter the number of processes -- 4
Enter Burst Time for Process 0 -- 5
Enter Burst Time for Process 1 -- 3
Enter Burst Time for Process 2 -- 2
Enter Burst Time for Process 3 -- 1

         PROCESS        BURST TIME       WAITING TIME    TURNAROUND TIME

         P3              1               0               1
         P2              2               1               3
         P1              3               3               6
         P0              5               6               11
Average Waiting Time -- 2.500000
Average Turnaround Time -- 5.250000%
```

Lab Experiment No. 3

Implementation of Round Robin & Priority CPU Scheduling Algorithms

Aim: To get the idea of how logics for Round Robin & Priority CPU Scheduling Algorithms are developed.

Round Robin Background:

- Criteria: Arrival Time
- Mode: Preemptive
- Process runs for a given Time Quantum (TQ) and preempts and goes to the end of RQ
- Practically Implementable, as AT in RQ.
- Using Queues, RR can be implemented, unlike of complex data structure like Heaps.
- No starvation, as no process is going to wait for CPU forever.
- Every process gets a chance after some amount of time and keep on getting chance after some time.

1. **RR Code (for same arrival time):**

```c
#include <stdio.h>
int main()
{
  int i, j, n, bt[10], ibt[10], wt[10], tat[10], t, max, p[50], g = 0;
  float cwt = 0, ctat = 0, temp = 0;
  printf("Enter the no of processes -- ");
  scanf("%d", &n);
  for (i = 0; i < n; i++)
  {
    printf("\nEnter Burst Time for process %d -- ", i);
    scanf("%d", &bt[i]);
    ibt[i] = bt[i];
  }
  printf("\nEnter the size of time slice -- ");
  scanf("%d", &t);
  max = bt[0];
  for (i = 1; i < n; i++)
  {
    if (max < bt[i])
      max = bt[i];
  }
```

```
  for (j = 0; j < (max / t) + 1; j++)
    for (i = 0; i < n; i++)
      if (bt[i] != 0)  {
        if (bt[i] <= t)   {
          tat[i] = temp + bt[i];
          temp = temp + bt[i];
          bt[i] = 0;
          p[g++] = i;
        } else {
          bt[i] = bt[i] - t;
          temp = temp + t;
          p[g++] = i;
        }  }
  for (i = 0; i < n; i++) {
    wt[i] = tat[i] - ibt[i];
    ctat += tat[i];
    cwt += wt[i]; }
  printf("\nThe Average Turnaround time is -- %f", ctat / n);
  printf("\nThe Average Waiting time is -- %f ", cwt / n);
  printf("\n");
  for (i = 0; i < g; i++)
    printf("pid - >%d --", p[i]);
  printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");
  for (i = 0; i < n; i++)
    printf("\t%d \t %d \t\t %d \t\t %d \n", i, ibt[i], wt[i], tat[i]);
  return 0;
}
```

Output

```
Enter the no of processes -- 4
Enter Burst Time for process 0 -- 5
Enter Burst Time for process 1 -- 3
Enter Burst Time for process 2 -- 4
Enter Burst Time for process 3 -- 2
Enter the size of time slice -- 2

The Average Turnaround time is -- 11.500000
The Average Waiting time is -- 8.000000
pid - >0 --pid - >1 --pid - >2 --pid - >3 --pid - >0 --pid - >1 --pid - >2 --pid - >0 --
        PROCESS  BURST TIME      WAITING TIME    TURNAROUND TIME
        0        5              9               14
        1        3              8               11
        2        4              9               13
        3        2              6               8
saurabh@Saurabhs-Air 3 %
```

Priority Scheduling Background:

- Criteria: Priority
- Mode: Non-Preemptive
- Here, Lesser value Higher Priority

2. **Priority Scheduling Code (for same arrival time):**

```c
#include <stdio.h>
int main(){
  int p[20], bt[20], pri[20], wt[20], tat[20], i, k, n, temp;
  float cwt, ctat;
  printf("Enter the number of processes --- ");
  scanf("%d", &n);
  for (i = 0; i < n; i++){
    p[i] = i;
    printf("Enter the Burst Time & Priority of Process %d --- ", i);
    scanf("%d %d", &bt[i], &pri[i]);
  }
  for (i = 0; i < n; i++)
    for (k = i + 1; k < n; k++)
      if (pri[i] > pri[k]) {
        temp = p[i];
        p[i] = p[k];
        p[k] = temp;
        temp = bt[i];
        bt[i] = bt[k];
        bt[k] = temp;
        temp = pri[i];
        pri[i] = pri[k];
        pri[k] = temp;
      }
  cwt = wt[0] = 0;
  ctat = tat[0] = bt[0];
  for (i = 1; i < n; i++){
    wt[i] = wt[i - 1] + bt[i - 1];
    tat[i] = tat[i - 1] + bt[i];
    cwt = cwt + wt[i];
    ctat = ctat + tat[i];
  }
```

```
  printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND TIME");
  for (i = 0; i < n; i++)
    printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ", p[i], pri[i], bt[i], wt[i], tat[i]);
  printf("\nAverage Waiting Time is --- %f", cwt / n);
  printf("\nAverage Turnaround Time is --- %f", ctat / n);
  return 0;
}
```

Output :

```
Enter the number of processes --- 4
Enter the Burst Time & Priority of Process 0 --- 2 3
Enter the Burst Time & Priority of Process 1 --- 3 1
Enter the Burst Time & Priority of Process 2 --- 3 2
Enter the Burst Time & Priority of Process 3 --- 5 5

PROCESS         PRIORITY        BURST TIME      WAITING TIME    TURNAROUND TIME
1                1              3               0               3
2                2              3               3               6
0                3              2               6               8
3                5              5               8               13
Average Waiting Time is --- 4.250000
Average Turnaround Time is --- 7.500000%
saurabh@Saurabhs-Air 3 %
```

3. RR Code (For different Arrival Time)

```
#include <stdio.h>
int main(){
  int count, j, n, time, remain, flag = 0, time_quantum;
  int wait_time = 0, turnaround_time = 0, at[10], bt[10], rt[10];
  printf("Enter Total Process:\t ");
  scanf("%d", &n);
  remain = n;
  for (count = 0; count < n; count++) {
    printf("Enter Arrival Time and Burst Time for Process Process Number %d :", count + 1);
    scanf("%d", &at[count]);
    scanf("%d", &bt[count]);
    rt[count] = bt[count];
  }
  printf("Enter Time Quantum:\t");
  scanf("%d", &time_quantum);
  printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
```

```
for (time = 0, count = 0; remain != 0;){ // 1 process completed -> count++
    if (rt[count] <= time_quantum && rt[count] > 0){
        time += rt[count];
        rt[count] = 0;
        flag = 1;
    }
    else if (rt[count] > 0) {
        rt[count] -= time_quantum;
        time += time_quantum;
    }
    if (rt[count] == 0 && flag == 1){ // process completed
        remain--;
        printf("P[%d]\t|\t%d\t|\t%d\n", count + 1, time - at[count], time - at[count] - bt[count]);
        wait_time += time - at[count] - bt[count];
        turnaround_time += time - at[count];
        flag = 0;
    }
    if (count == n - 1)   count = 0;
    else if (at[count + 1] <= time)   count++;
    else count = 0;
}
printf("\nAverage Waiting Time= %f\n", wait_time * 1.0 / n);
printf("Avg Turnaround Time = %f", turnaround_time * 1.0 / n);
return 0;
}
```

Output :

```
Enter Total Process:     4
Enter Arrival Time and Burst Time for Process Process Number 1 :2 1
Enter Arrival Time and Burst Time for Process Process Number 2 :0 2
Enter Arrival Time and Burst Time for Process Process Number 3 :3 3
Enter Arrival Time and Burst Time for Process Process Number 4 :1 6
Enter Time Quantum:     2


Process |Turnaround Time|Waiting Time

P[1]    |       -1      |       -2
P[2]    |       3       |       1
P[3]    |       5       |       2
P[4]    |       11      |       5

Average Waiting Time= 1.500000
Avg Turnaround Time = 4.500000%
```

3. Priority Code (For different Arrival Time)

```c
#include <stdio.h>
int main(){
    int bt[20], p[20], wt[20], tat[20], pr[20], i, j, n, total = 0, pos, temp, avg_wt, avg_tat;
    printf("Enter Total Number of Process:");   scanf("%d", &n);
    printf("\nEnter Burst Time and Priority\n");
    for (i = 0; i < n; i++)   {
        printf("\nP[%d]\n", i + 1);
        printf("Burst Time:");
        scanf("%d", &bt[i]);
        printf("Priority:");
        scanf("%d", &pr[i]);
        p[i] = i + 1; // contains process number
    } // sorting burst time, priority and process number in ascending order using selection sort
    for (i = 0; i < n; i++) {
        pos = i;
        for (j = i + 1; j < n; j++) {
            if (pr[j] < pr[pos])
                pos = j;
        }
        temp = pr[i];
        pr[i] = pr[pos];
        pr[pos] = temp;
        temp = bt[i];
        bt[i] = bt[pos];
        bt[pos] = temp;
        temp = p[i];
        p[i] = p[pos];
        p[pos] = temp;
    }
    wt[0] = 0; // waiting time for first process is zero
    // calculate waiting time
    for (i = 1; i < n; i++){
        wt[i] = 0;
        for (j = 0; j < i; j++)
            wt[i] += bt[j];
        total += wt[i];
    }
```

```
    avg_wt = total / n; // average waiting time
    total = 0;
    printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
    for (i = 0; i < n; i++)
    {
        tat[i] = bt[i] + wt[i]; // calculate turnaround time
        total += tat[i];
        printf("\nP[%d]\t\t  %d\t\t    %d\t\t\t%d", p[i], bt[i], wt[i], tat[i]);
    }
    avg_tat = total / n; // average turnaround time
    printf("\n\nAverage Waiting Time=%d", avg_wt);
    printf("\nAverage Turnaround Time=%d\n", avg_tat);

    return 0;
}
```

Output :

```
Enter Total Number of Process: 4

Enter Burst Time and Priority

P[1]
Burst Time: 3
Priority: 2

P[2]
Burst Time: 2
Priority: 1

P[3]
Burst Time: 4
Priority: 4

P[4]
Burst Time: 5
Priority: 2

Process     Burst Time          Waiting Time    Turnaround Time
P[2]              2                 0                 2
P[1]              3                 2                 5
P[4]              5                 5                 10
P[3]              4                 10                14

Average Waiting Time=4
Average Turnaround Time=7
saurabh@Saurabhs-Air 3 %
```

5. Find CT(Completion Time) in RR.

```c
#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("\n Enter 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 Time\t Turnaround Time\t Waiting Time\t COMPETION 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\t%d", i + 1, burst_time[i], tota
            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("\nnAverage Waiting Time:\t%f", average_wait_time);
    printf("\nAvg Turnaround Time:\t%f\n", average_turnaround_time);
    return 0;
    }
```

Output :

```
 Enter Total Number of Processes:       4

Enter Details of Process[1]
Arrival Time:   0
Burst Time:     2

Enter Details of Process[2]
Arrival Time:   2
Burst Time:     5

Enter Details of Process[3]
Arrival Time:   3
Burst Time:     8

Enter Details of Process[4]
Arrival Time:   4
Burst Time:     6

Enter Time Quantum:     3

Process ID              Burst Time      Turnaround Time         Waiting Time    COMPETION TIME

Process[1]              2               2                       0       2
Process[2]              5               11                      6       13
Process[4]              6               15                      9       19
Process[3]              8               18                      10      21
nAverage Waiting Time:  6.250000
Avg Turnaround Time:    11.500000
saurabh@Saurabhs-Air 3 %
```

5. Try Preemptive scheduling with MinHeaps

```c
#include<stdio.h>
struct process{
	char process_name;
	int arrival_time, burst_time, ct, waiting_time, turnaround_time, priority;
	int status;
}process_queue[10];
int limit;
void Arrival_Time_Sorting(){
	struct process temp;
	int i, j;
	for(i = 0; i < limit - 1; i++)   {
		for(j = i + 1; j < limit; j++)   {
			if(process_queue[i].arrival_time > process_queue[j].arrival_time)      {
				temp = process_queue[i];
				process_queue[i] = process_queue[j];
				process_queue[j] = temp;
			}
		}  } }
int main(){
	int i, time = 0, burst_time = 0, largest;
	char c;
	float wait_time = 0, turnaround_time = 0, average_waiting_time, average_turnaround_time
	printf("\nEnter Total Number of Processes:\t");
	scanf("%d", &limit);
	for(i = 0, c = 'A'; i < limit; i++, c++) {
		process_queue[i].process_name = c;
		printf("\nEnter Details For Process[%C]:\n", process_queue[i].process_name);
		printf("Enter Arrival Time:\t");
		scanf("%d", &process_queue[i].arrival_time );
		printf("Enter Burst Time:\t");
		scanf("%d", &process_queue[i].burst_time);
		printf("Enter Priority:\t");
		scanf("%d", &process_queue[i].priority);
		process_queue[i].status = 0;
		burst_time = burst_time + process_queue[i].burst_time;
	}
```

```
    Arrival_Time_Sorting();
    process_queue[9].priority = -9999;
    printf("\nProcess Name\tArrival Time\tBurst Time\tPriority\tWaiting Time");
    for(time = process_queue[0].arrival_time; time < burst_time;) {
        largest = 9;
        for(i = 0; i < limit; i++)    {
            if(process_queue[i].arrival_time <= time && process_queue[i].status != 1 &&
                process_queue[i].priority > process_queue[largest].priority)
            largest = i;
        }
        time = time + process_queue[largest].burst_time;
        process_queue[largest].ct = time;
        process_queue[largest].waiting_time = process_queue[largest].ct -
            process_queue[largest].arrival_time - process_queue[largest].burst_time;
        process_queue[largest].turnaround_time = process_queue[largest].ct - process_queue[largest].arrival_time;
        process_queue[largest].status = 1;
        wait_time = wait_time + process_queue[largest].waiting_time;
        turnaround_time = turnaround_time + process_queue[largest].turnaround_time;
        printf("\n%c\t\t%d\t\t%d\t\t%d\t\t%d", process_queue[largest].process_name,
            process_queue[largest].arrival_time, process_queue[largest].burst_time,
                process_queue[largest].priority, process_queue[largest].waiting_time);
    }
    average_waiting_time = wait_time / limit;
    average_turnaround_time = turnaround_time / limit;
    printf("\n\nAverage waiting time:\t%f\n", average_waiting_time);
    printf("Average Turnaround Time:\t%f\n", average_turnaround_time);

return 0;
}
```

```
Enter Total Number of Processes:        4

Enter Details For Process[A]:
Enter Arrival Time:     1
Enter Burst Time:       2
Enter Priority: 3

Enter Details For Process[B]:
Enter Arrival Time:     2
Enter Burst Time:       4
Enter Priority: 1

Enter Details For Process[C]:
Enter Arrival Time:     0
Enter Burst Time:       4
Enter Priority: 2

Enter Details For Process[D]:
Enter Arrival Time:     4
Enter Burst Time:       5
Enter Priority: 2

Process Name    Arrival Time    Burst Time      Priority        Waiting Time
C               0               4               2               0
A               1               2               3               3
D               4               5               2               2
B               2               4               1               9

Average waiting time:   3.500000
Average Turnaround Time:        7.250000
saurabh@Saurabhs-MacBook-Air 3 %
```

Lab Experiment No. 4

Implementation of Multilevel Queue Scheduling Algorithm

Aim: To get the idea of how logic for Multilevel Queue Scheduling is developed.

MLQ Background:

- Ready queue is partitioned into separate queues, e.g.:
    - **System Processes RQ**
    - **User Processes RQ**
- Processes doesn't change queues and remain in the given queues.
- Each queue has its own scheduling algorithm:
    - System Process Ready Queue – SJF
    - User Process Ready Queue – FCFS
- NOTE: Implementation of Scheduling Algorithms for different Queue depends upon the application requirements. Ex: foreground processes can have RR and background processes can have FCFS.
- Scheduling must be done between the queues:
    - Fixed priority scheduling; (i.e., serve all from higher priority queue first than from lower priority queue). Possibility of starvation.
    - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes.

MLQ Code (for same arrival time):

- Lower Queue Priority (QP) number means higher queue priority.
- System processes are assigned to a queue having QP=0 and called as System Process Ready Queue.
- User processes are assigned to a queue having QP=1 and called as User Process Ready Queue.
- System Process Ready Queue implements SJF.
- User Process Ready Queue implements FCFS.

TASK 1 : i:- Multilevel queue where every process have same arrival time and user process have less priority then system process. Implement FCFS for user process and SJF for system process.

```c
#include <stdio.h>
int main(){
    // qp is Queue Prioriy, cwt is Cummulative Waiting Time,
    // and ctat is Cummulative Turn Around Time
    int pid[20], bt[20], qp[20], wt[20], tat[20], i, k, n, temp;
    float cwt, ctat;
    printf("Enter the number of processes --- ");
    scanf("%d", &n);
    for (i = 0; i < n; i++){
        pid[i] = i;
        printf("Enter the Burst Time of Process %d --- ", i);
        scanf("%d", &bt[i]);
        printf("System/User Process (0/1) ? --- ");
        scanf("%d", &qp[i]);
    }
    for (i = 0; i < n; i++)
        for (k = i + 1; k < n; k++)
            if (qp[i] > qp[k])    {
                temp = pid[i];
                pid[i] = pid[k];
                pid[k] = temp;
                temp = bt[i];
                bt[i] = bt[k];
                bt[k] = temp;
                temp = qp[i];
                qp[i] = qp[k];
                qp[k] = temp;
            }
    for (i = 0; i < n; i++)
        for (k = i + 1; k < n; k++) {
            if (qp[i] == 0 && qp[k] == 0 && bt[i] > bt[k])     {
                temp = pid[i];
                pid[i] = pid[k];
                pid[k] = temp;
                temp = bt[i];
                bt[i] = bt[k];
                bt[k] = temp;
                temp = qp[i];
                qp[i] = qp[k];
                qp[k] = temp;
            }
```

```
            else if (qp[i] == 1 && qp[k] == 1 && pid[i] > pid[k]){
                temp = pid[i];
                pid[i] = pid[k];
                pid[k] = temp;

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

                temp = qp[i];
                qp[i] = qp[k];
                qp[k] = temp;
            }
        }
    cwt = wt[0] = 0;
    ctat = tat[0] = bt[0];
    for (i = 1; i < n; i++) {
        wt[i] = wt[i - 1] + bt[i - 1];
        tat[i] = tat[i - 1] + bt[i];
        cwt = cwt + wt[i];
        ctat = ctat + tat[i];
    }
    printf("\nPROCESS\t\t SYSTEM/USER PROCESS \tBURST TIME\tWAITING TIME\tTURNAROUND TIME");
    for (i = 0; i < n; i++)
        printf("\n%d \t\t\t\t %d \t\t %d \t\t %d \t\t %d ", pid[i], qp[i], bt[i], wt[i], tat[i]);
    printf("\nAverage Waiting Time is --- %f", cwt / n);
    printf("\nAverage Turnaround Time is --- %f", ctat / n);
    return 0;
}
```

```
Enter the number of processes --- 4
Enter the Burst Time of Process 0 --- 10
System/User Process (0/1) ? --- 0
Enter the Burst Time of Process 1 --- 4
System/User Process (0/1) ? --- 0
Enter the Burst Time of Process 2 --- 12
System/User Process (0/1) ? --- 0
Enter the Burst Time of Process 3 --- 5
System/User Process (0/1) ? --- 1

PROCESS          SYSTEM/USER PROCESS    BURST TIME      WAITING TIME    TURNAROUND TIME
1                                0              4               0               4
0                                0              10              4               14
2                                0              12              14              26
3                                1              5               26              31
Average Waiting Time is --- 11.000000
Average Turnaround Time is --- 18.750000%
```

Task 1.ii. Multilevel queue where every process have different arrival time and user process have less priority then system process. Implement FCFS for user process and SJF for system process.

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    int n;
    printf("Enter the number of process u want:\t");
    cin >> n;
    int AT[n], type[n], BT[n];
    for (int i = 0; i < n; i++){
        printf("Enter the Arrival time of the process:\t");
        cin >> AT[i];
        printf("Enter the Burst time of the process: \t");
        cin >> BT[i];
        printf("Enter the type of the process: 0/1 system/user\t");
        cin >> type[i];
    }
    int o_tym = -1, tym = 0, RQ[4][n], count = 0, process_done = 0;
    float cwt = 0, ctat = 0;
    printf("Process id\t Arrival Time\t Burst Time\t Waiting Time\t Turn around time\n");
    while (1) {
        for (int i = 0; i < n; i++)   {
            if (AT[i] > o_tym && AT[i] <= tym)      {
                RQ[0][count] = AT[i];
                RQ[1][count] = type[i];
                RQ[2][count] = i + 1;
                RQ[3][count] = BT[i];
                count++;
            }
        }
        o_tym = tym;
        int min = INT_MAX, index = -1;
        for (int i = 0; i < count; i++){
            if (RQ[3][i] < min && RQ[1][i] == 0 && RQ[3][i] > 0)  {
                min = RQ[3][i];
                index = i;
            }
        }
```

```
        if (index != -1) {
            process_done++;
            tym += RQ[3][index];
            ctat = tym - RQ[0][index];
            cwt = ctat - RQ[3][index];
            cout << RQ[2][index] << "\t\t " << RQ[0][index]
            << "\t\t " << min << "\t\t " << cwt << "\t\t " << ctat << endl;
            RQ[3][index] = -1;
        }
        else if (process_done < count) {
            int min = INT_MAX, index = -1;
            for (int i = 0; i < count; i++)  {
                if (RQ[3][i] > 0 && RQ[1][i] == 1 && RQ[0][i] < min)   {
                    min = RQ[0][i];
                    index = i;
                }
            }
            process_done++;
            tym += RQ[3][index];
            ctat = tym - RQ[0][index];
            cwt = ctat - RQ[3][index];
            cout << RQ[2][index] << "\t\t " << RQ[0][index]
            << "\t\t " << RQ[3][index] << " \t\t" << cwt << " \t\t" << ctat << endl;
            RQ[3][index] = -1;
        }
        else  tym++;
        if (process_done == n)  break;
    }
    return 0;
}
```

```
Enter the number of process u want:     4
Enter the Arrival time of the process:  0
Enter the Burst time of the process:    2
Enter the type of the process: 0/1 system/user  1
Enter the Arrival time of the process:  2
Enter the Burst time of the process:    3
Enter the type of the process: 0/1 system/user  0
Enter the Arrival time of the process:  2
Enter the Burst time of the process:    2
Enter the type of the process: 0/1 system/user  1
Enter the Arrival time of the process:  3
Enter the Burst time of the process:    2
Enter the type of the process: 0/1 system/user  0
Process id       Arrival Time    Burst Time      Waiting Time    Turn around time
1                0               2               0               2
2                2               3                0               3
4                3               2                2               4
3                2               2               5               7
saurabh@Saurabhs-MacBook-Air 4 %
```

Task 2. Multilevel Feedback queue with time counter for each queue.

```cpp
#include <bits/stdc++.h>
using namespace std;
int main(){
    int n;
    printf("Enter the number of process u want:\t");
    cin >> n;
    int type[n], BT[n];
    for (int i = 0; i < n; i++) {
        printf("Enter the Burst time of the process: \t");
        cin >> BT[i];
        printf("Enter the type of the process: 0/1 system/user\t");
        cin >> type[i];
    }
    int RQ[3][n], count = 0, process_done = 0, s_count = 0, u_count = 0;
    float cwt = 0, ctat = 0;
    printf("Process id\t Burst Time\t Waiting Time\t Turn around time\n");
    for (int i = 0; i < n; i++) {
        if (BT[i] <= 6 && type[i] == 0) {
            RQ[0][count] = BT[i];
            RQ[1][count] = type[i];
            RQ[2][count] = i + 1;
            count++;
            s_count++;
        } else if (BT[i] <= 10 && type[i] == 1) {
            RQ[0][count] = BT[i];
            RQ[1][count] = type[i];
            RQ[2][count] = i + 1;
            count++;
            u_count++;
        }
    }
    int p = 0;
    while (1) {
        int min = INT_MAX, index = -1;
        if (p == 0){
            if (p == 0 && s_count > 0)    {
                for (int i = 0; i < count; i++)    {
                    if (RQ[0][i] < min && RQ[1][i] == 0 && RQ[0][i] > 0)  {
                        min = RQ[0][i];
                        index = i;
                    }
                }  if (index != -1)  {
                    ctat += min;
                    process_done++;
                    cout << RQ[2][index] << "\t\t " << min << "\t\t " << cwt << "\t\t " << ctat << endl;
```

```
                cout << RQ[2][index] << "\t\t " << min << "\t\t " << cwt << "\t\t " << ctat << endl;
                cwt += RQ[0][index];
                RQ[0][index] = -1;
                p = 1;
            }  s_count--;
        }
        else
            p = 1;
    } else if (p == 1 && u_count > 0) {

            u_count--;
            p = 0;
            for (int i = 0; i < count; i++)
            {
                if (RQ[0][i] > 0 && RQ[1][i] == 1)
                {
                    min = RQ[0][i];
                    index = i;
                    break;
                }
            }
            ctat += RQ[0][index];
            process_done++;
            cout << RQ[2][index] << "\t\t" << RQ[0][index] << "\t\t" << cwt << "\t\t" << ctat << endl;
            cwt += RQ[0][index];
            RQ[0][index] = -1;
        }

    else  {
        if (p == 0) p = 1;
        else p = 0;
    }  if (process_done == count)   break;
  }
}
```

```
Enter the number of process u want:     4
Enter the Burst time of the process:    2
Enter the type of the process: 0/1 system/user  0
Enter the Burst time of the process:    2
Enter the type of the process: 0/1 system/user  1
Enter the Burst time of the process:    5
Enter the type of the process: 0/1 system/user  0
Enter the Burst time of the process:    6
Enter the type of the process: 0/1 system/user  1
Process id       Burst Time      Waiting Time    Turn around time
1                2               0               2
2               2               2               4
3                5               4               9
4               6               9               15
saurabh@Saurabhs-MacBook-Air 4 %
```

Lab Experiment No. 5

Implementation of Synchronized P-C using Semaphores and Mutex through Threads.

Aim: To get the idea of how Semaphores and Mutex are used for synchronizing a critical section.

Producer-Consumer Background:

- Sleep and Wake are the system calls.
- When a process calls sleep( ), it get blocked.
- When a process calls wakeup( ), one of the blocked processes get awake up.
- *Producer* process produces or adding/writing something in buffer.
- *Consumer* process consumes or deleting/reading something from buffer.
- Buffer is a memory
- In Sleep and Wake solution, we came across that if a process is not sleeping and if another process is trying to wake him up, wakeup signal goes lost.
- Instead of using that, Dijkstra has proposed that no need to send the wakeup signal to sleeping process, just write the signal in variable, so that it will be stored and be there, so anyone can use it, whenever wanted (go to sleep).
- If Semaphore is implemented at user-mode, then two processes can access it simultaneously to add a wakeup call and Semaphore value become inconsistent and inefficient.
- Therefore, most of OS provides some primitives (function calls), use to access shared variables (Semaphores) atomically.
    - Means, if a process is reading Semaphore, all other processes stop using it.
    - Means, Read, Update and Write is used atomically.
- Hence, to need atomicity, we need OS support in Kernel-mode.
- Variables on which Read, Modify and Update happens atomically in Kernel-mode (means no preemption).
- Types of Semaphores
    1. Counting Semaphores
    2. Binary Semaphores (Mutex)
- Counting Semaphore variable shows how many processes can enter into CS simultaneously.
- Suppose a resource i.e. printer, can be shared between 4 processes but not more than 4 processes. Then, Counting Semaphore=4.
- If, Semaphore = + *N* (means *N* processes can enter simultaneously in CS at present).

- If, Semaphore = - *Z* (means *Z* processes tried to enter in fully occupied CS and get blocked. Now, they are in WAITING Queue).In counting semaphore, we counts the value of semaphore as the number of processes allowed to be in CS at any time (+ve value) and processes in BLOCKED Queue at any time (-ve value).
    - But, binary semaphores are not used for counting, they are just used for entering or exiting a process in and from CS in ME manner.

1. **Problem of Synchronization using Threads**

1.1. **Code without any pre-emption**

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
void *fun1();
void *fun2();
int shared = 1; // shared variable
int main(){
    pthread_t thread1, thread2;
    pthread_create(&thread1, NULL, fun1, NULL);
    pthread_create(&thread2, NULL, fun2, NULL);
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    printf("Final value of shared is %d\n", shared);
    // prints the last updated value of shared variable
}
void *fun1(){
    int x;
    x = shared; // thread1 reads value of shared variable
    printf("Thread1 reads the value as %d\n", x);
    x++; // thread1 increments its value
    printf("Local updation by Thread1: %d\n", x);
    shared = x; // thread one updates the value of shared variable
    printf("Value of shared variable updated by Thread1 is: %d\n", shared);
}
void *fun2()
{
    int y;
    y = shared; // thread2 reads value of shared variable
    printf("Thread2 reads the value as %d\n", y);
    y--; // thread2 increments its value
    printf("Local updation by Thread2: %d\n", y);
    shared = y; // thread2 updates the value of shared variable
    printf("Value of shared variable updated by Thread2 is: %d\n", shared);
}
```

Output :

```
Thread1 reads the value as 1
Local updation by Thread1: 2
Value of shared variable updated by Thread1 is: 2
Thread2 reads the value as 1
Local updation by Thread2: 0
Value of shared variable updated by Thread2 is: 0
Final value of shared is 0
saurabh@Saurabhs-Air 5 %
```

1.2. **Code with pre-emption**

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
void *fun1();
void *fun2();
int shared = 1; // shared variable
int main(){
        pthread_t thread1, thread2;
        pthread_create(&thread1, NULL, fun1, NULL);
        pthread_create(&thread2, NULL, fun2, NULL);
        pthread_join(thread1, NULL);
        pthread_join(thread2, NULL);
        printf("Final value of shared is %d\n", shared);
        // prints the last updated value of shared variable
}
void *fun1(){
        int x;
        x = shared; // thread1 reads value of shared variable
        printf("Thread1 reads the value as %d\n", x);
        x++; // thread1 increments its value
        printf("Local updation by Thread1: %d\n", x);
        sleep(1);   // thread1 is preempted by thread 2
        shared = x; // thread one updates the value of shared variable
        printf("Value of shared variable updated by Thread1 is: %d\n", shared);
}
void *fun2(){
        int y;
        y = shared; // thread2 reads value of shared variable
        printf("Thread2 reads the value as %d\n", y);
        y--; // thread2 increments its value
```

```
        y--; // thread2 increments its value
        printf("Local updation by Thread2: %d\n", y);
        sleep(1);   // thread2 is preempted by thread 1
        shared = y; // thread2 updates the value of shared variable
        printf("Value of shared variable updated by Thread2 is: %d\n", shared);
    }
```

Output :

```
Thread1 reads the value as 1
Local updation by Thread1: 2
Thread2 reads the value as 1
Local updation by Thread2: 0
Value of shared variable updated by Thread1 is: 2
Value of shared variable updated by Thread2 is: 0
Final value of shared is 0
saurabh@Saurabhs-Air 5 %
```

2. **Synchronization in Threads using Semaphores**

```
#include <pthread.h>
#include <stdio.h>
#include <semaphore.h>
#include <unistd.h>
void *fun1();
void *fun2();
int shared = 1; // shared variable
sem_t s;        // semaphore variable
int main(){
    sem_init(&s, 0, 1); // initialize semaphore variable -
    // 1st argument is address of variable, 2nd - no. of processes sharing semaphore,
    // 3rd argument is the initial value of semaphore variable
    pthread_t thread1, thread2;
    pthread_create(&thread1, NULL, fun1, NULL);
    pthread_create(&thread2, NULL, fun2, NULL);
```

```c
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    printf("Final value of shared is %d\n", shared);
    // prints the last updated value of shared variable
}
void *fun1()
{
    int x;
    sem_wait(&s); // executes wait operation on s
    x = shared;   // thread1 reads value of shared variable
    printf("Thread1 reads the value as %d\n", x);
    x++; // thread1 increments its value
    printf("Local updation by Thread1: %d\n", x);
    sleep(1);   // thread1 is preempted by thread 2
    shared = x; // thread one updates the value of shared variable
    printf("Value of shared variable updated by Thread1 is: %d\n", shared);
    sem_post(&s);
}
void *fun2()
{
    int y;
    sem_wait(&s);
    y = shared; // thread2 reads value of shared variable
    printf("Thread2 reads the value as %d\n", y);
    y--; // thread2 increments its value
    printf("Local updation by Thread2: %d\n", y);
    sleep(1);   // thread2 is preempted by thread 1
    shared = y; // thread2 updates the value of shared variable
    printf("Value of shared variable updated by Thread2 is: %d\n", shared);
    sem_post(&s);
}
```

Output :

```
Thread1 reads the value as 1
Local updation by Thread1: 2
Thread2 reads the value as 1
Local updation by Thread2: 0
Value of shared variable updated by Thread1 is: 2
Value of shared variable updated by Thread2 is: 0
Final value of shared is 0
saurabh@Saurabhs-Air 5 %
```

3. **Producer Consumer Code without Sleep() and wakeup().**

```c
#include <stdio.h>
int main(){
    int buffer[10], bufsize, in, out, produce, consume, choice = 0;
    in = 0;
    out = 0;
    bufsize = 10;
    while (choice != 3)  {
        printf("1. Produce \t 2. Consume \t3. Exit");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice)   {
        case 1:
            if ((in + 1) % bufsize == out)
                printf("Buffer is Full");
            else   {
                printf("Enter the value: ");
                scanf("%d", &produce);
                buffer[in] = produce;
                in = (in + 1) % bufsize;
            } break;
        case 2:
            if (in == out)   printf("Buffer is Empty");
            else{
                consume = buffer[out];
                printf("The consumed value is %d", consume);
                out = (out + 1) % bufsize;
            } break;
        }
    } return 0;
}
```

Output :

```
1. Produce      2. Consume     3. Exit
 Enter your choice: 1
Enter the value: 2
1. Produce      2. Consume     3. Exit
 Enter your choice: 2
The consumed value is 2
1. Produce      2. Consume     3. Exit
 Enter your choice: 2
Buffer is Empty
1. Produce      2. Consume     3. Exit
 Enter your choice: 3
saurabh@Saurabhs-Air 5 %
```

4. **PC Code with Semaphores and Mutex:**

```c
#include <pthread.h>
#include <semaphore.h>
#include <stdlib.h>
#include <stdio.h>
#define MaxItems 5
#define BufferSize 5
sem_t empty;
sem_t full;
int in = 0;
int out = 0;
int buffer[BufferSize];
pthread_mutex_t mutex;
void *producer(void *pno){
    int item;
    for (int i = 0; i < MaxItems; i++) {
        item = rand();
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
        buffer[in] = item;
        printf("Producer %d: Insert Item %d at %d\n", *((int *)pno), buffer[in], in);
        in = (in + 1) % BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
    }
}
void *consumer(void *cno){
    for (int i = 0; i < MaxItems; i++)  {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        int item = buffer[out];
        printf("Consumer %d: Remove Item %d from %d\n", *((int *)cno), item, out);
        out = (out + 1) % BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
    }
}
```

```c
int main(){
    pthread_t pro[5], con[5];
    pthread_mutex_init(&mutex, NULL);
    sem_init(&empty, 0, BufferSize);
    sem_init(&full, 0, 0);
    int a[5] = {1, 2, 3, 4, 5};
    for (int i = 0; i < 5; i++)   {
        pthread_create(&pro[i], NULL, (void *)producer, (void *)&a[i]);
    }

    for (int i = 0; i < 5; i++) {
        pthread_create(&con[i], NULL, (void *)consumer, (void *)&a[i]);
    }
    for (int i = 0; i < 5; i++) {
        pthread_join(pro[i], NULL);
    }
    for (int i = 0; i < 5; i++) {
        pthread_join(con[i], NULL);
    }
    pthread_mutex_destroy(&mutex);
    sem_destroy(&empty);
    sem_destroy(&full);
    return 0;
}
```

Output :

```
Producer 1: Insert Item 16807 at 0
Producer 1: Insert Item 1144108930 at 1
Producer 1: Insert Item 470211272 at 2
Producer 3: Insert Item 1622650073 at 3
Consumer 1: Remove Item 16807 from 0
Consumer 1: Remove Item 1144108930 from 1
Producer 5: Insert Item 101027544 at 4
Producer 5: Insert Item 2007237709 at 0
Consumer 2: Remove Item 470211272 from 2
Consumer 2: Remove Item 1622650073 from 3
Producer 4: Insert Item 984943658 at 1
Producer 4: Insert Item 1115438165 at 2
Producer 2: Insert Item 282475249 at 3
Consumer 3: Remove Item 101027544 from 4
Consumer 3: Remove Item 2007237709 from 0
Producer 1: Insert Item 1457850878 at 4
Producer 5: Insert Item 823564440 at 0
Consumer 4: Remove Item 984943658 from 1
Consumer 5: Remove Item 1115438165 from 2
Consumer 5: Remove Item 282475249 from 3
```

```
Producer 1: Insert Item 114807987 at 1
Consumer 2: Remove Item 1457850878 from 4
Consumer 2: Remove Item 823564440 from 0
Consumer 2: Remove Item 114807987 from 1
Producer 4: Insert Item 1784484492 at 2
Producer 4: Insert Item 1441282327 at 3
Producer 4: Insert Item 16531729 at 4
Consumer 5: Remove Item 1784484492 from 2
Consumer 5: Remove Item 1441282327 from 3
Consumer 5: Remove Item 16531729 from 4
Producer 5: Insert Item 1137522503 at 0
Producer 5: Insert Item 823378840 at 1
Producer 3: Insert Item 1458777923 at 2
Consumer 4: Remove Item 1137522503 from 0
Consumer 4: Remove Item 823378840 from 1
Producer 2: Insert Item 74243042 at 3
Consumer 3: Remove Item 1458777923 from 2
Producer 3: Insert Item 143542612 at 4
Consumer 4: Remove Item 74243042 from 3
Consumer 1: Remove Item 143542612 from 4
Producer 2: Insert Item 896544303 at 0
Consumer 3: Remove Item 896544303 from 0
Producer 3: Insert Item 1474833169 at 1
Consumer 4: Remove Item 1474833169 from 1
Consumer 1: Remove Item 1458777923 from 2
Consumer 1: Remove Item 74243042 from 3
Consumer 3: Remove Item 143542612 from 4
Producer 2: Insert Item 1264817709 at 2
Producer 2: Insert Item 1817129560 at 3
Producer 3: Insert Item 1998097157 at 4
saurabh@Saurabhs-Air 5 %
```

Q). Why sleep() and wakeUp() is not recommended for Producer Consumer Problem ?

Ans). Well, the problem arises in the case when the consumer got preempted just before it was about to sleep. Now the consumer is neither sleeping nor consuming. Since the producer is not aware of the fact that consumer is not actually sleeping therefore it keep waking the consumer while the consumer is not responding since it is not sleeping. This leads to the wastage of system calls. When the consumer get scheduled again, it will sleep because it was about to sleep when it was preempted.The producer keep writing in the buffer and it got filled after some time. The producer will also sleep at that time keeping in the mind that the consumer will wake him up when there is a slot available in the buffer.The consumer is also sleeping and not aware with the fact that the producer will wake him up.This is a kind of deadlock where neither producer nor consumer is active and waiting for each other to wake them up. This is a serious problem which needs to be addressed.

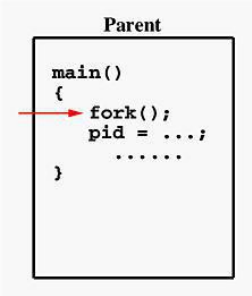
Lab Experiment No. 6

Implementation of System Calls related to Process Creation

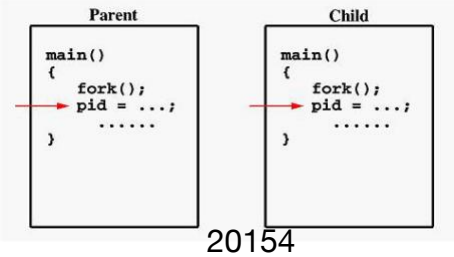
Aim: To get the idea of how processes are created using fork() system call.

fork() System Call:

- Fork system call is used for creating a new process, which is called child process, which runs concurrently with the process that makes the fork() call (parent process).
- After a new child process is created, both processes will execute the next instruction following the fork() system call.
- A child process uses the same pc(program counter), same CPU registers, same open files which use in the parent process.
- It takes no parameters and returns an integer value. Below are different values returned by fork().
    - Negative Value: creation of a child process was unsuccessful.
    - Zero: Returned to the newly created child process.
    - Positive value: Returned to parent or caller. The value contains process ID of newly created child process
- Suppose the given program executes up to the point of the call to fork() (marked in red color):

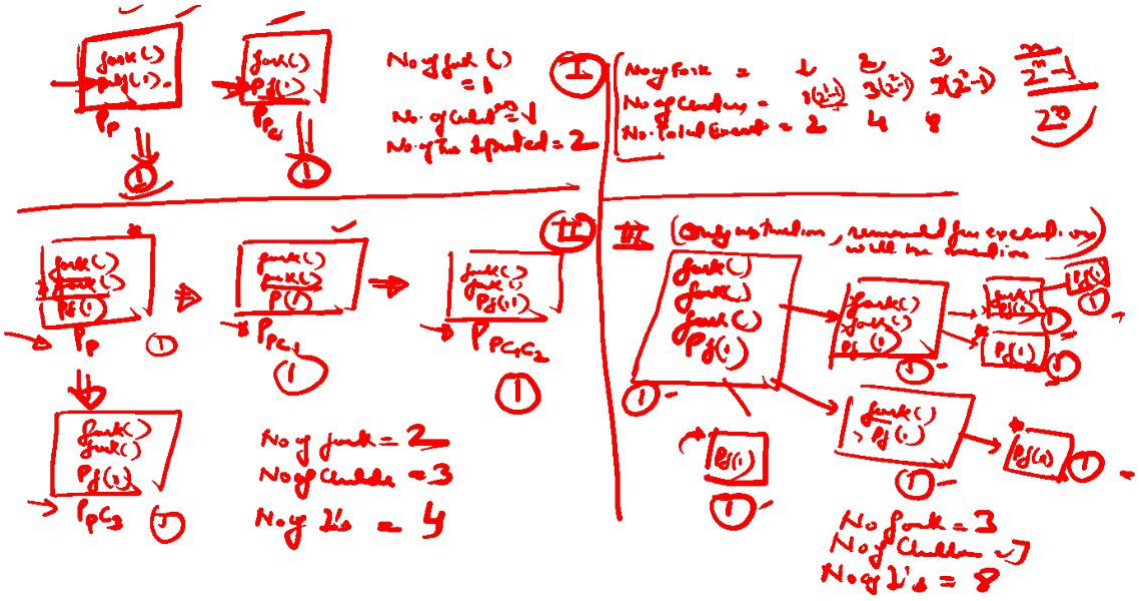


- If the call to fork() is executed successfully, it will
    - make two identical copies of address spaces, one for the parent and the other for the child.
    - Both processes will start their execution at the next statement following the fork() call. In this case, both processes will start their execution at the assignment statement as shown below:



- Both processes start their execution right after the system call fork().
- Since both processes have identical but separate address spaces, those variables initialized before the fork() call have the same values in both address spaces.
- Since every process has its own address space, any modifications will be independent of the others.
- In other words, if the parent changes the value of its variable, the modification will only affect the variable in the parent process's address space.
- Other address spaces created by fork() calls will not be affected even though they have identical variable names.

- fork()-based Numerical Solution:



Code-I: For tracing the execution of Parent and Child processes

```c
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>
int main(){
    pid_t pid = fork();
    if (pid == 0)  {
        printf("fork executed successfully: This is Child process with PID: %d \n",
        getpid());
        printf("This is Child process. My Parent PID: %d \n",
        getppid());
    }
    else if (pid > 0) {
        wait(NULL);
        printf("I am Parent process with PID : %d \n", getpid());
    }
    return 0;
}
```

Output :

```
fork executed successfully: This is Child process with PID: 5778
This is Child process. My Parent PID: 5777
I am Parent process with PID : 5777
saurabh@Saurabhs-MacBook-Air 6 %
```

Code-II: For tracing the execution of Parent and Child processes

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
int main(){
    printf("Parent PPID %d\n", getppid());
    printf("I am Parent with PID : %d \n", (int)getppid());
    printf("Parent going to create a child process...\n");
    pid_t pid = fork();
    if (pid == 0) {
        printf("fork executed successfully: Child PID: %d \n", (int)getpid());
        printf("My Parent with PID : %d \n", (int)getppid());
    }
    else {
        wait(NULL);
        printf("I am the parent, my PID %d \n", getpid());
    }
    return 0;
}
```

Output:

```
Parent PPID 5616
I am Parent with PID : 5616
Parent going to create a child process...
fork executed successfully: Child PID: 5636
My Parent with PID : 5635
I am the parent, my PID 5635
saurabh@Saurabhs-MacBook-Air 6 %
```

Code-III: For replacing the code image of Parent and Child processes

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char *argv[]){
    pid_t pid = fork();
    if (pid == 0)  {
        printf("Child PID: %d \n", (int)getpid());
        execlp(argv[1], "arguments", NULL);
    }
    else if (pid > 0) {
        wait(NULL);
        // printf("Parent resumes from there");
        printf("Parent PID: %d \n", (int)getpid());
        execl(argv[2], "some arguments", NULL);
    }
}
```

Output :

```
Child PID: 6046
Parent PID: 6045
saurabh@Saurabhs-MacBook-Air 6 %
```

Lab Experiment No. 7

Implementation of Deadlock Avoidance Algorithm

Aim: To get the idea of how processes' demand-based analysis can avoid deadlock.

Deadlock Avoidance:

- It's better to check for deadlock existence in future by determining whether the system is in safe state or in unsafe state.
    - **Safe State**: If the available resources can be allocated to the requesting process
    - **Unsafe State**: If the available resources cannot be allocated to requesting process
- Deadlock avoidance take care of safe and unsafe state because unsafe state will result in deadlock.
- At an instance of time, processes from A to E has resources allocated and required as follows.

Process	R0	R1	R2	R3
P0	3	0	1	1
P1	0	1	0	0
P2	1	1	1	0
P3	1	1	0	1
P4	0	0	0	0

Resources Assigned

Process	R0	R1	R2	R3
P0	4	1	1	1
P1	0	2	1	2
P2	4	2	1	0
P3	1	1	1	1
P4	2	1	1	0

Maximum Requirement

- Total Available= (1,0,2,0)
- B' request = (0,0,1,0)
- Will the system be in safe state after catering B' request

Code:

```c
#include <stdio.h>
struct file
{
    int all[10];
    int max[10];
    int need[10];
    int flag;
};
int main()
{
    struct file f[10];
    int fl;
    int i, j, k, p, b, n, r, g, cnt = 0, id, newr;
```

```
    int i, j, k, p, b, n, r, g, cnt = 0, id, newr;
    int avail[10], seq[10];
    printf("Enter number of processes -- ");
    scanf("%d", &n);
    printf("Enter number of resources -- ");
    scanf("%d", &r);
    for (i = 0; i < n; i++) {
        printf("Enter details for P%d", i);
        printf("\nEnter allocation\t -- \t");
        for (j = 0; j < r; j++)
            scanf("%d", &f[i].all[j]);
        printf("Enter Max\t\t -- \t");
        for (j = 0; j < r; j++)
            scanf("%d", &f[i].max[j]);
        f[i].flag = 0;
    }
    printf("\nEnter Available Resources\t -- \t");
    for (i = 0; i < r; i++)  scanf("%d", &avail[i]);
    printf("\nEnter New Request Details -- ");
    printf("\nEnter pid \t -- \t");
    scanf("%d", &id);
    printf("Enter Request for Resources \t -- \t");
    for (i = 0; i < r; i++) {
        scanf("%d", &newr);
        f[id].all[i] += newr;
        avail[i] = avail[i] - newr;
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < r; j++)  {
            f[i].need[j] = f[i].max[j] - f[i].all[j];
            if (f[i].need[j] < 0)
                f[i].need[j] = 0;
        }
    }
    cnt = 0;
    fl = 0;
    while (cnt != n) {
```

```
        g = 0;
        for (j = 0; j < n; j++)   {
            if (f[j].flag == 0)    {
                b = 0;
                for (p = 0; p < r; p++)     {
                    if (avail[p] >= f[j].need[p])  b = b + 1;
                    else  b = b - 1;
                }
                if (b == r) {
                    printf("\nP%d is visited", j);
                    seq[fl++] = j;
                    f[j].flag = 1;
                    for (k = 0; k < r; k++)
                        avail[k] = avail[k] + f[j].all[k];
                    cnt = cnt + 1;
                    printf("(");
                    for (k = 0; k < r; k++)
                        printf("%3d", avail[k]);
                    printf(")");
                    g = 1;
                }
            }
        }
        if (g == 0) {
            printf("\n REQUEST NOT GRANTED -- DEADLOCK OCCURRED");
            printf("\n SYSTEM IS IN UNSAFE STATE");
            goto y;
        }
    }
    printf("\nSYSTEM IS IN SAFE STATE");
    printf("\nThe Safe Sequence is -- (");
    for (i = 0; i < fl; i++)
        printf("P%d ", seq[i]);
    printf(")");
y:
    printf("\nProcess\t\tAllocation\t\tMax\t\t\tNeed\n");
    for (i = 0; i < n; i++)
```

```
    for (i = 0; i < n; i++) {
        printf("P%d\t", i);
        for (j = 0; j < r; j++)
            printf("%6d", f[i].all[j]);
        for (j = 0; j < r; j++)
            printf("%6d", f[i].max[j]);
        for (j = 0; j < r; j++)
            printf("%6d", f[i].need[j]);
        printf("\n");
    }
    return 0;
}
```

Output :

```
Enter number of processes -- 5
Enter number of resources -- 3
Enter details for P0
Enter allocation        --      0 1 0
Enter Max               --      7 5 3
Enter details for P1
Enter allocation        --      2 0 0
Enter Max               --      3 2 2
Enter details for P2
Enter allocation        --      3 0 2
Enter Max               --      9 0 2
Enter details for P3
Enter allocation        --      2 1 1
Enter Max               --      2 2 2
Enter details for P4
Enter allocation        --      0 0 2
Enter Max               --      4 3 3

Enter Available Resources       --      3 3 2

Enter New Request Details --
Enter pid       --      0
Enter Request for Resources     --      1 3 2

 REQUEST NOT GRANTED -- DEADLOCK OCCURRED
 SYSTEM IS IN UNSAFE STATE
Process         Allocation              Max                     Need
P0          1     4     2     7     5     3     6     1     1
P1          2     0     0     3     2     2     1     2     2
P2          3     0     2     9     0     2     6     0     0
P3          2     1     1     2     2     2     0     1     1
P4          0     0     2     4     3     3     4     3     1
saurabh@Saurabhs-MacBook-Air 7 %
```

Lab Experiment No. 8

Simulation of MFT and MVT Memory Management Techniques

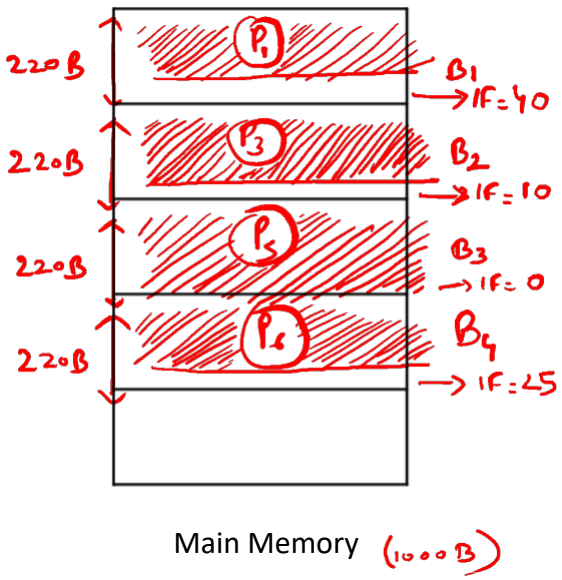
Aim: To get the idea of how contiguous allocation of processes get affected due to the fixed and dynamic memory partitioning schemes.

Background:

1. MFT or fixed partitioning Scheme

- Full form of MFT is Multiprogramming with Fixed number of Tasks
- In fixed scheme, the OS will be divided into fixed sized blocks. It takes place at the time of installation.
- Degree of multiprogramming is not flexible. This is because the number of blocks is fixed resulting in memory wastage due to fragmentation.
- Example:
    - Suppose, there are total of 1000 B of space in Main Memory.
    - OS has four fixed blocks (partitions) of size 220 B each.
    - There are total of six processes with following memory requirements:
        - P1=180 B
        - P2=250 B
        - P3=210 B
        - P4=310 B
        - P5=220 B
        - P6=195 B
    - The memory has the allocation of processes and internal fragmentations

      as shown in the figure, with external fragmentation (EF)=195 B



MFT Code:

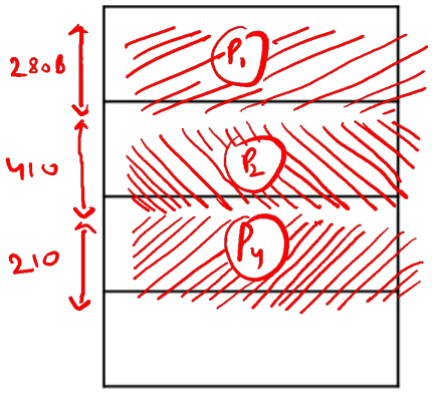
```c
#include <stdio.h>
int main(){
    int ms, bs, nob, ef, n, mp[10], tif = 0, oop;
    int i, p = 0;
    printf("Enter the total memory available (in Bytes) -- ");
    scanf("%d", &ms);
    printf("Enter the block size (in Bytes) -- ");
    scanf("%d", &bs);
    nob = ms / bs;
    oop = ms - nob * bs;
    printf("\nEnter the number of processes -- ");
    scanf("%d", &n);
    for (i = 0; i < n; i++) {
        printf("Enter memory required for process %d (in Bytes)-- ", i + 1);
        scanf("%d", &mp[i]);
    }
    printf("\nNo. of Blocks available in memory -- %d", nob);
    printf("\n\nPROCESS\tMEMORY REQUIRED\t ALLOCATED\tINTERNAL FRAGMENTATION");
    for (i = 0; i < n && p < nob; i++)  {
        printf("\n %d\t\t%d", i + 1, mp[i]);
        if (mp[i] > bs)  printf("\t\tNO\t\t---");
        else  {
            printf("\t\tYES\t%d", bs - mp[i]);
            tif = tif + bs - mp[i];
            p++;
        }
    }
    ef = oop + tif;
    if (i < n)   printf("\nMemory is Full, Remaining Processes cannot be accomodated");
    printf("\n\nTotal Internal Fragmentation is %d", tif);
    printf("\nTotal External Fragmentation is %d", ef);
    return 0;
}
```

MFT Output:

```
Enter the total memory available (in Bytes) -- 256
Enter the block size (in Bytes) -- 8

Enter the number of processes -- 10
Enter memory required for process 1 (in Bytes)-- 7
Enter memory required for process 2 (in Bytes)-- 8
Enter memory required for process 3 (in Bytes)-- 4
Enter memory required for process 4 (in Bytes)-- 6
Enter memory required for process 5 (in Bytes)-- 8
Enter memory required for process 6 (in Bytes)-- 2
Enter memory required for process 7 (in Bytes)-- 1
Enter memory required for process 8 (in Bytes)-- 3
Enter memory required for process 9 (in Bytes)-- 3
Enter memory required for process 10 (in Bytes)-- 4

No. of Blocks available in memory -- 32

PROCESS MEMORY REQUIRED  ALLOCATED      INTERNAL FRAGMENTATION
 1               7               YES     1
 2               8               YES     0
 3               4               YES     4
 4               6               YES     2
 5               8               YES     0
 6               2               YES     6
 7               1               YES     7
 8               3               YES     5
 9               3               YES     5
 10              4               YES     4

Total Internal Fragmentation is 34
Total External Fragmentation is 34%
```

2. MVT OR variable partitioning Scheme

- Full form of MVT is Multiprogramming with Variable number of Tasks
- In variable partitioning scheme there are no partitions at the beginning.
- There is only the OS area and the rest of the available RAM.
- The memory is allocated to the processes as they enter.
- This method is more flexible as there is no internal fragmentation and there is no size limitation.
- Example
    - Suppose, there are total of 1000 B of space in Main Memory.
    - There are total of four processes with following memory requirements:
        - P1=280 B
        - P2=410 B
        - P3=500 B
        - P4=210 B
    - The memory has the allocation of processes as shown in the figure, with external fragmentation (EF) =100 B.



Main Memory (1000B)

MVT Code:

```c
#include <stdio.h>
int main(){
    int ms, mp[10], i, temp, n = 0;
    char ch = 'y';
    printf("\nEnter the total memory available (in Bytes)-- ");
    scanf("%d", &ms);
    temp = ms;
    for (i = 0; ch == 'y'; i++, n++) {
        printf("\nEnter memory required for process %d (in Bytes) -- ", i + 1);
        scanf("%d", &mp[i]);
        if (mp[i] <= temp)    {
            printf("\nMemory is allocated for Process %d ", i + 1);
            temp = temp - mp[i];
        }
        else
            printf("\nMemory is not available for the current request");
        printf("\nDo you want to continue(y/n) -- ");
        scanf(" %c", &ch);
if(ch=='n')
break;
    }
    printf("\n\nTotal Memory Available -- %d", ms);
    printf("\n\n\tPROCESS\t\t MEMORY ALLOCATED ");
    for (i = 0; i < n; i++)
        printf("\n \t%d\t\t%d", i + 1, mp[i]);
    printf("\n\nTotal Memory Allocated is %d", ms - temp);
    printf("\nTotal External Fragmentation is %d", temp);
    return 0;
}
```

MVT Output:

```
Enter the total memory available (in Bytes)-- 256
Enter memory required for process 1 (in Bytes) -- 7
Memory is allocated for Process 1
Do you want to continue(y/n) -- y
Enter memory required for process 2 (in Bytes) -- 4
Memory is allocated for Process 2
Do you want to continue(y/n) -- y
Enter memory required for process 3 (in Bytes) -- 6
Memory is allocated for Process 3
Do you want to continue(y/n) -- y
Enter memory required for process 4 (in Bytes) -- 8
Memory is allocated for Process 4
Do you want to continue(y/n) -- y
Enter memory required for process 5 (in Bytes) -- 6
Memory is allocated for Process 5
Do you want to continue(y/n) -- y
Enter memory required for process 6 (in Bytes) -- 1
Memory is allocated for Process 6
Do you want to continue(y/n) -- y
Enter memory required for process 7 (in Bytes) -- 3
Memory is allocated for Process 7
Do you want to continue(y/n) -- y
Enter memory required for process 8 (in Bytes) -- 2
Memory is allocated for Process 8
Do you want to continue(y/n) -- n

Total Memory Available -- 256
        PROCESS          MEMORY ALLOCATED
        1               7
        2               4
        3               6
        4               8
        5               6
        6               1
        7               3
Total Memory Allocated is 37
Total External Fragmentation is 219%
saurabh@Saurabhs-MacBook-Air 8 %
```

Lab Experiment No. 9

Simulation of First Fit, Best Fit and Worst Fit Memory Management Techniques

Aim: To get the idea of how contiguous allocation of processes get affected due to the fixed and dynamic memory partitioning schemes.

1. First Fit

- This is one of the Simplest Methods for Memory Allocation. There, the main motive is to divide the memory into several fixed Sizes. Each partition of First Fit Program in C contains exactly one process.
- In the First Fit Memory Management Scheme, we check the block in the sequential manner i.e we take the first process and compare its size with the first block.If the size is less than the size of the first block then only it is allocated. Otherwise, we move to the second block and this process is continuously going on until all processes are allocated. It is the fastest searching Algorithm as we not to search only first block, we do not need to search a lot.
- The main disadvantage of First Fit is that the extra space cannot be used by any other processes.If the memory is allocated it creates large amount of chunks of memory space.

First Fit Code:

```c
#include <stdio.h>
int main(){
    int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;
    for (i = 0; i < 10; i++) {
        flags[i] = 0;
        allocation[i] = -1;
    }
    printf("Enter no. of blocks: ");
    scanf("%d", &bno);
    printf("\nEnter size of each block: ");
    for (i = 0; i < bno; i++)
        scanf("%d", &bsize[i]);
    printf("\nEnter no. of processes: ");
    scanf("%d", &pno);
    printf("\nEnter size of each process: ");
    for (i = 0; i < pno; i++)
        scanf("%d", &psize[i]);
    for (i = 0; i < pno; i++) // allocation as per first fit
```

```
        for (j = 0; j < bno; j++)
            if (flags[j] == 0 && bsize[j] >= psize[i])
            {
                allocation[j] = i;
                flags[j] = 1;
                break;
            }
    // display allocation details
    printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");
    for (i = 0; i < bno; i++){
        printf("\n%d\t\t%d\t\t", i + 1, bsize[i]);
        if (flags[i] == 1)
            printf("%d\t\t\t%d", allocation[i] + 1, psize[allocation[i]]);
        else
            printf("Not allocated");
    }
    return 0;
}
```

First Fit Output:

```
rs/saurabh/Desktop/OS/9/"FirstFit
Enter no. of blocks: 3

Enter size of each block: 8 10 12

Enter no. of processes: 3

Enter size of each process: 40 14 12

Block no.       size            process no.             size
1               8               Not allocated
2               10              Not allocated
3               12              3                       12%
saurabh@Saurabhs-MacBook-Air 9 %
```

2. Best Fit:

- Best fit uses the best memory block based on the Process memory request. In best fit implementation the algorithm first selects the smallest block which can adequately fulfill the memory request by the respective process.
- Because of this memory is utilized optimally but as it compares the blocks with the requested memory size it increases the time requirement and hence slower than other methods. It suffers from Internal Fragmentation which simply means that the memory block size is greater than the memory requested by the process, then the free space gets wasted.
- Once we encounter a process that requests a memory which is higher than block size we stop the algorithm.

Best Fit Code:

```c
#include <stdio.h>
int main(){
    int fragment[20], b[20], p[20], i, j, nb, np, temp, lowest = 9999;
    static int barray[20], parray[20];
    printf("\n\t\t\tMemory Management Scheme - Best Fit");
    printf("\nEnter the number of blocks:");
    scanf("%d", &nb);
    printf("Enter the number of processes:");
    scanf("%d", &np);
    printf("\nEnter the size of the blocks:-");
    for (i = 1; i <= nb; i++) {
        printf("Block no.%d:", i);
        scanf("%d", &b[i]); }
    printf("\nEnter the size of the processes :-");
    for (i = 1; i <= np; i++)  {
        printf("Process no.%d:", i);
        scanf("%d", &p[i]); }
    for (i = 1; i <= np; i++){
        for (j = 1; j <= nb; j++)   {
            if (barray[j] != 1)    {
                temp = b[j] - p[i];
                if (temp >= 0)
                    if (lowest > temp)  {
                        parray[i] = j;
                        lowest = temp;
```

```
                }
            }
        }
        fragment[i] = lowest;
        barray[parray[i]] = 1;
        lowest = 10000;
    }
    printf("\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment");
    for (i = 1; i <= np && parray[i] != 0; i++)
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, p[i], parray[i], b[parray[i]], fragment[i]);
    return 0;
}
```

Best Fit Output:

```
                        Memory Management Scheme - Best Fit
Enter the number of blocks:5
Enter the number of processes:4

Enter the size of the blocks:-Block no.1:10
Block no.2:15
Block no.3:5
Block no.4:8
Block no.5:3

Enter the size of the processes :-Process no.1:1
Process no.2:4
Process no.3:7
Process no.4:12

Process_no      Process_size    Block_no        Block_size      Fragment
1               1               5               3               2
2               4               3               5               1
3               7               4               8               1
4               12              2               15              3%
saurabh@Saurabhs-MacBook-Air 9 %
```

3. Worst Fit:

- Allocate the largest hole. Again, we must search the entire list, unless it is sorted by size. In case of worst fit, unlike the best fit, it searches the entire hole and chooses the largest hole possible, and allocates the process.
- The best-fit allocation method keeps the free/busy lists in order by size, largest to smallest.
- Here it requires O(n) time like the best fit to search the entire list and find the largest one. But one advantage is there compare to best first, after allocating process as per worst fit free holes size is large and it is useful to allocate other processes.
- Here after allocating free hole size is large i.e. better but still, it requires O(n) to search the entire

Worst Fit Code:

```c
#include <stdio.h>
#define max 25
int main()
{
    int frag[max], b[max], f[max], i, j, nb, nf, temp, highest = 0;
    static int bf[max], ff[max];
    printf("\n\tMemory Management Scheme - Worst Fit");
    printf("\nEnter the number of blocks:");
    scanf("%d", &nb);
    printf("Enter the number of files:");
    scanf("%d", &nf);
    printf("\nEnter the size of the blocks:-\n");
    for (i = 1; i <= nb; i++)
    {
        printf("Block %d:", i);
        scanf("%d", &b[i]);
    }
    printf("Enter the size of the files :-\n");
    for (i = 1; i <= nf; i++)
    {
        printf("File %d:", i);
        scanf("%d", &f[i]);
    }
```

```
    printf("Enter the size of the files :-\n");
    for (i = 1; i <= nf; i++)  {
        printf("File %d:", i);
        scanf("%d", &f[i]);
    }
    for (i = 1; i <= nf; i++)  {
        for (j = 1; j <= nb; j++)   {
            if (bf[j] != 1) {// if bf[j] is not allocated
                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;
        } ff[i] = j;
        highest = temp;
    }
    printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
    for (i = 1; i <= nf; i++)
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
    return 0;
}
```

Worst Fit Output:

```
        Memory Management Scheme - Worst Fit
Enter the number of blocks: 4
Enter the number of files: 2

Enter the size of the blocks:-
Block 1:4
Block 2:5
Block 3:6
Block 4:3
Enter the size of the files :-
File 1:2
File 2:3

File_no:        File_size :     Block_no:       Block_size:     Fragement
1               2               5               0               1
2               3               5               0               0%
```