

Swami Vivekanand College of Engineering Indore
Department of Information Technology



Session: June - Dec

LAB MANUAL
Operating Sysytem
IT – 504

Submitted To
Ms. Aarti khare
Assistant Professor

Submitted By
Abhishek Mishra
0822IT211004

Department's Vision and Mission

Vision: To achieve global standard in quality of education, research & development in Information Technology by adapting to the rapid technological advancement to empowering the IT-industry with the wings of knowledge and power of innovation through knowledge creation, acquisition and dissemination for the benefit of Society and Humanity.

Mission:

M1: To provide students with Innovative and research skills, which is need of the hour for technical students.

M2: To impart knowledge to the students of Information Technology with relevant core and practical knowledge inculcating real time experience in the promising field of computing.

M3: To prepare the graduates to meet information technology challenges with a blend of social, human, ethical and value based education.

M4: To engage in emerging research areas and establishing leadership.

M5: To contribute to the community services at large as well as catering to socio- economic goals in local and national levels.

PROGRAM OUTCOMES (POs)

PO1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2. Problem analysis: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change .

PROGRAM EDUCATION OBJECTIVES (PEOs)

PEO-I: Meet Market Demands: To prepare students to become a successful engineer to meet the demand driven needs of industries/technical profession.

PEO-II: Core Competence: Graduates will demonstrate core competence in mathematical, scientific and basic engineering fundamentals necessary to formulate, analyze and solve engineering problems and also to pursue advanced study or research.

PEO-III: Design and Analysis : Graduates will demonstrate good breadth of knowledge in core areas of Information Technology and related engineering so as to comprehend engineering trade-offs, analyze, design, and synthesize data and technical concepts to create novel designs in solving the real life problems.

PEO-IV: Professional Responsibility : Graduates will demonstrate professional responsibility by offering a wide spectrum of consultancy and testing services by addressing social, cultural, economic, sustainability, and environmental considerations in the solution of real world engineering problems.

PEO-V: Life-long learning: Graduates will engage themselves in life-long learning through independent study and by participating in professional activities or continuing education .

Course Outcomes (CO)

After the completion of this course, the students will be able to:

CO 1. Gain knowledge of history of operating systems

CO 2 Understand design issues associated with operating systems

CO 3. Gain knowledge of various process management concepts including scheduling ,
synchronization, deadlocks

CO 4. Understand concepts of memory management including virtual memory

CO 5. Understand issues related to file system interface and implementation , diskmanagement

CO 6. Be familiar with protection and security mechanisms

CO 7. Be familiar with various types of operating systems including Unix

INDEX

S. NO	Name of Experiment	Date of Experiment	Date of Submission	Signature	Remark
1.	Program to implement FCFS CPU scheduling algorithm				
2.	Program to implement SJF CPU scheduling algorithm				
3.	Program to implement priority CPU scheduling				
4.	Program to implement Round Robin CPU scheduling algorithm				
5.	Program to implement Round Robin CPU scheduling algorithm.				
6.	Program to implement classical inter process communication problem (Reader Writers).				
7.	Program to implement classical inter process communication problem (Dining Philosophers).				
8.	Program to implement FIFO page replacement algorithm				
9.	Program to implement FIFO page replacement algorithm.				

Experiment: -1

Question :1) Program to implement FCFS CPU scheduling algorithm

First Come First Serve is a scheduling algorithm used by the CPU to schedule jobs. It is a Non-Preemptive Algorithm. Priority is given according to which they are requested by the processor. Let's understand the concept in the following order:

Terms In First Come First Serve Scheduling

Example Code

FCFS Scheduling Explanation

First Come First Serve Scheduling The process that requests the services of CPU first, get the CPU first. This is the philosophy used by the first come first serve algorithm.

TERMS In First Come First Serve

Completion time: Time when the execution is completed.

Turn Around Time: The sum of the burst time and waiting time gives the turn-around time

Waiting time: The time the processes need to wait for it to get the CPU and start execution is called waiting time.

Important Points

It is non-preemptive

Average waiting time is not optimal

Cannot utilize resources in parallel

Example Code

```
#include <stdio.h>

int main()
{
    int n, bt[20], wt[20], tat[20], avwt = 0, avtat = 0, i, j;

    printf("Enter total number of processes (maximum 20):");

    scanf("%d", &n);

    printf("\nEnter Process Burst Time\n");

    for (i = 0; i < n; i++)
    {
        printf("P[%d]:", i + 1);

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

    wt[0] = 0;

    for (i = 1; i < n; i++)
    {
        wt[i] = 0;

        for (j = 0; j < i; j++)
            wt[i] += bt[j];
```

```

}

printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time");

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

    avwt += wt[i];

    avtat += tat[i];

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

}

avwt /= n;

avtat /= n;

printf("\nAverage Waiting Time:%d", avwt);

printf("\nAverage Turnaround Time:%d\n", avtat);

return 0;

}

```

OUTPUT:

```

E:\New C++\Untitled1.exe
Enter total number of processes (maximum 20):2
Enter Process Burst Time
P[1]:1
P[2]:2

Process Burst Time      Waiting Time      Turnaround Time
P[1]          1          0          1
P[2]          2          1          3
Average Waiting Time:0
Average Turnaround Time:2

```

EXPLANATION

In the above code, the demonstration of the first come first serve scheduling algorithm is shown. The user is asked to enter the number of processes. On entering the number of processes, we have to enter the burst times for each of the processes.

The waiting time is calculated first. First, the waiting time of the first process is zero.

```
for(i=1;i<n;i++)  
{  
    wt[i]=0;  
    for(j=0;j<i;j++)  
        wt[i]+=bt[j];  
}
```

Calculation of the waiting time is done by adding the burst time of the previous process. Consider the previous process had a burst time of 10, then the waiting time of second will be 10. Similarly, for the third process, the waiting time will be the sum of burst times of first and second processes.

```
for(i=0;i<n;i++)  
{  
    tat[i]=bt[i]+wt[i];  
    avwt+=wt[i];  
    avtat+=tat[i];  
    printf("nP[%d]tt%dttdtt%d",i+1,bt[i],wt[i],tat[i]);  
}
```

The next part we calculate the turn around time. The turn around time for each process is calculated by adding the burst time and the waiting time.

Last, the average turn around time and the average waiting time is calculated.

$avwt = \sum w_i / i;$

$avtat = \sum t_i / i;$

i gives the total number of processes. We divide the sum of all the waiting times and turn around times to get the average. This is how the first come first serve algorithm works.

With this, we come to an end of this First Come First Serve Scheduling in C Programming.

I hope you found this informative and helpful, stay tuned for more tutorials on similar topics. You may also check out our training program to get in-depth knowledge on jQuery along with its various applications, you can enroll here for live online training with 24/7 support and lifetime access. Implement the above code with different strings and modifications. Now, we have a good understanding of all key concepts related to the pointer.

Experiment: -2

Question :2) Write a program to implement SJF scheduling.

Solution:-

```
#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <conio.h>

int main()

{

    char **p; // Using dynamic memory allocation for process names

    int *pt, *wt; // Arrays for process times and waiting times

    int tot = 0;

    float avg = 0;

    int n, temp1;

    printf("Enter the number of processes: ");

    scanf("%d", &n);

    // Allocate memory for process names dynamically

    p = (char **)malloc(n * sizeof(char *));

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

```
{  
  
    p[i] = (char *)malloc(5 * sizeof(char));  
  
    printf("Enter process %d name: ", i + 1);  
  
    scanf("%s", p[i]);  
  
}
```

```
pt = (int *)malloc(n * sizeof(int));  
  
wt = (int *)malloc(n * sizeof(int));
```

```
for (int i = 0; i < n; i++)  
  
{  
  
    printf("Enter process time for %s: ", p[i]);  
  
    scanf("%d", &pt[i]);  
  
}
```

```
for (int i = 0; i < n - 1; i++)  
  
{  
  
    for (int j = i + 1; j < n; j++)  
  
    {  
  
        if (pt[i] > pt[j])  
  
        {  
  
            temp1 = pt[i];  
  
            pt[i] = pt[j];
```

```
    pt[j] = temp1;

    char temp[5];

    strcpy(temp, p[i]);

    strcpy(p[i], p[j]);

    strcpy(p[j], temp);

}

}

}
```

```
wt[0] = 0;

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

{

    wt[i] = wt[i - 1] + pt[i - 1];

    tot = tot + wt[i];

}
```

```
avg = (float)tot / n;

printf("Process Name\tProcess Time\tWaiting Time\n");

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

    printf("%s\t\t%d\t\t%d\n", p[i], pt[i], wt[i]);

printf("Total Waiting Time = %d\nAverage Waiting Time = %f\n", tot, avg);

getch(); // This function is specific to some IDEs
```

```
// Free dynamically allocated memory
```

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

```
    free(p[i]);
```

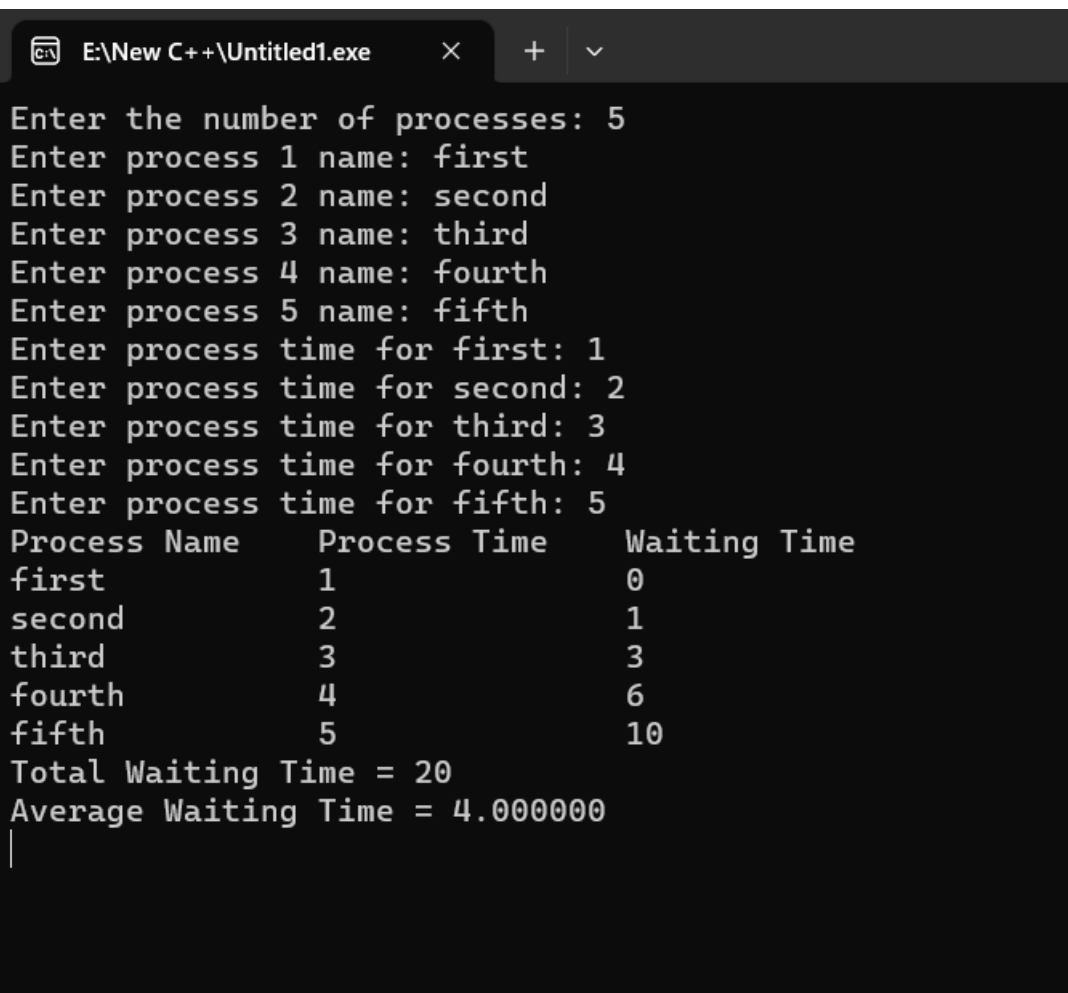
```
free(p);
```

```
free(pt);
```

```
free(wt);
```

```
return 0;
```

```
}Output:-
```



The screenshot shows a terminal window with the title "E:\New C++\Untitled1.exe". The program prompts the user to enter the number of processes (5), then the names of five processes (first, second, third, fourth, fifth), and then the process times for each (1, 2, 3, 4, 5). It then displays a table of process names, times, and waiting times, followed by the total and average waiting times.

```
E:\New C++\Untitled1.exe  ×  +  v

Enter the number of processes: 5
Enter process 1 name: first
Enter process 2 name: second
Enter process 3 name: third
Enter process 4 name: fourth
Enter process 5 name: fifth
Enter process time for first: 1
Enter process time for second: 2
Enter process time for third: 3
Enter process time for fourth: 4
Enter process time for fifth: 5
Process Name      Process Time      Waiting Time
first             1                 0
second            2                 1
third             3                 3
fourth            4                 6
fifth             5                 10
Total Waiting Time = 20
Average Waiting Time = 4.000000
|
```


Experiment: - 3

Question :3) Write a program to implement priority scheduling

Solution:-

```
#include <iostream>
#include <conio.h>

using namespace std;

int main()
{
    int x, n, p[10], pp[10], pt[10], w[10], t[10], awt = 0, atat = 0, i;

    cout << "Enter the number of processes: ";
    cin >> n;
    cout << "\nEnter process time priorities: \n";

    for (i = 0; i < n; i++)
    {
        cout << "\nProcess no " << i + 1 << ": ";
        cin >> pt[i] >> pp[i];
        p[i] = i + 1;
    }

    for (i = 0; i < n - 1; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            if (pp[i] < pp[j])
            {
                x = pp[i];
                pp[i] = pp[j];
                pp[j] = x;

                x = pt[i];
                pt[i] = pt[j];
                pt[j] = x;

                x = p[i];
                p[i] = p[j];
            }
        }
    }
}
```

```

        p[j] = x;
    }
}

w[0] = 0;
awt = 0;
t[0] = pt[0];
atat = t[0];

for (i = 1; i < n; i++)
{
    w[i] = t[i - 1];
    awt += w[i];

    t[i] = w[i] + pt[i];
    atat += t[i];
}

cout << "\n\n Job \t Burst Time \t Wait Time \t Turn Around Time \t Priority \n";

for (i = 0; i < n; i++)
{
    cout << "\n " << p[i] << "\t\t " << pt[i] << "\t\t " << w[i] << "\t\t " << t[i]
        << "\t\t " << pp[i] << "\n";
}

awt /= n;
atat /= n;

cout << "\n Average Wait Time : " << awt << "\n";
cout << "\n Average Turn Around Time : " << atat << "\n";

getch();

return 0;
}

```

Output:-

```
E:\New C++\Untitled1.exe  ×  +  ∨
Enter the number of processes: 3
Enter process time priorities:
Process no 1: 2
3
Process no 2: 1
2
Process no 3: 5
2

Job      Burst Time      Wait Time      Turn Around Time      Priority
1          2          0          2          3
2          1          2          3          2
3          5          3          8          2

Average Wait Time : 1
Average Turn Around Time : 4
|
```