

# Dr. B.R Ambedkar National Institute of Technology, Jalandhar



## **Operating Systems Lab (ITPC - 224)**

### **Submitted to**

Dr. Jitendra Kumar Samriya

Department of IT

### **Submitted By**

Aditya Anand

20124009

IT (G1)

## CONTENTS

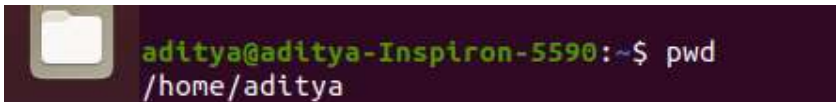
Sr. No.	Name of Practical	Date of Implementation	Page	Remarks
1.	Basic commands in Linux	06/01/2022	4	
2.	Round Robin Process  First Come First Serve - FCFS	17/01/2022	6	
3.	Shortest Job First – SJF  First Come First Serve – FCFS  Priority Based Scheduling	02/02/2022	8	
4.	Critical Section Problem Using Semaphores	09/02/2022	18	
5.	Producer Consumer Problem	16/02/2022	22	
6.	Banker's Deadlock Avoidance Algorithm	23/02/2022	24	
7.	Process Synchronization:  Reader – Writer Sleeping Barber	02/03/2022	30	
8.		28/03/2022	37	

	Dining Philosopher Problem for Process Synchronization			
9.	Page Replacement Algorithms – <ul style="list-style-type: none"> <li>a. FIFO</li> <li>b. LRU</li> <li>c. LFU</li> </ul> Simulation of Paging Technique	26/04/2022	45	

# OS LAB 1

## BASIC LINUX COMMANDS

- 1) pwd : prints the name of the current working directory

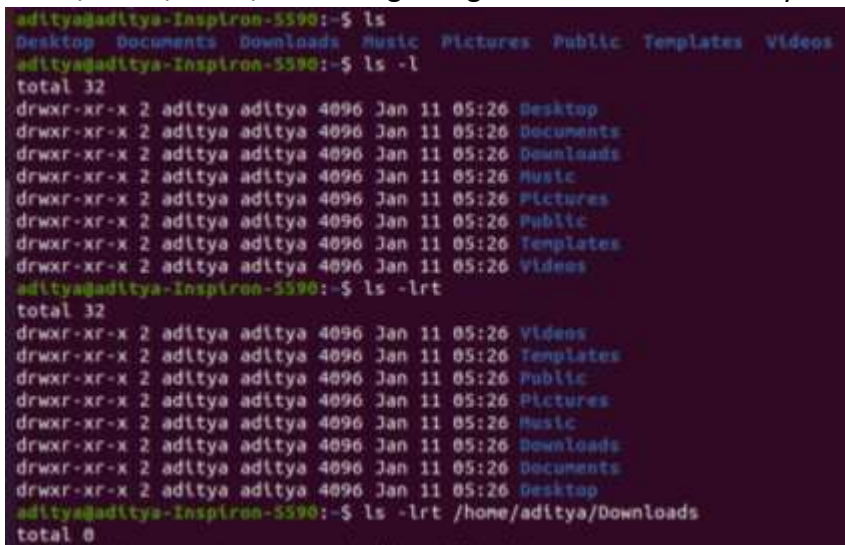


```
aditya@aditya-Inspiron-5590:~$ pwd
/home/aditya
```

- 2) ls : list contents of the current working directory

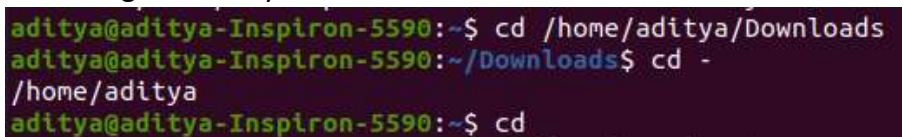
ls -lrt : long listing

ls -lrt /home/name/something : long list a different directory



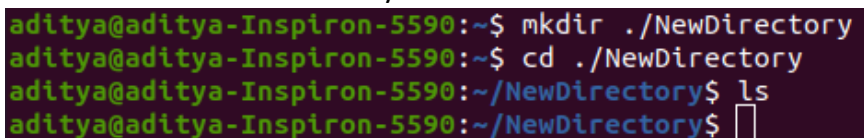
```
aditya@aditya-Inspiron-5590:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
aditya@aditya-Inspiron-5590:~$ ls -l
total 32
drwxr-xr-x 2 aditya aditya 4096 Jan 11 05:26 Desktop
drwxr-xr-x 2 aditya aditya 4096 Jan 11 05:26 Documents
drwxr-xr-x 2 aditya aditya 4096 Jan 11 05:26 Downloads
drwxr-xr-x 2 aditya aditya 4096 Jan 11 05:26 Music
drwxr-xr-x 2 aditya aditya 4096 Jan 11 05:26 Pictures
drwxr-xr-x 2 aditya aditya 4096 Jan 11 05:26 Public
drwxr-xr-x 2 aditya aditya 4096 Jan 11 05:26 Templates
drwxr-xr-x 2 aditya aditya 4096 Jan 11 05:26 Videos
aditya@aditya-Inspiron-5590:~$ ls -lrt
total 32
drwxr-xr-x 2 aditya aditya 4096 Jan 11 05:26 Videos
drwxr-xr-x 2 aditya aditya 4096 Jan 11 05:26 Templates
drwxr-xr-x 2 aditya aditya 4096 Jan 11 05:26 Public
drwxr-xr-x 2 aditya aditya 4096 Jan 11 05:26 Pictures
drwxr-xr-x 2 aditya aditya 4096 Jan 11 05:26 Music
drwxr-xr-x 2 aditya aditya 4096 Jan 11 05:26 Downloads
drwxr-xr-x 2 aditya aditya 4096 Jan 11 05:26 Documents
drwxr-xr-x 2 aditya aditya 4096 Jan 11 05:26 Desktop
aditya@aditya-Inspiron-5590:~$ ls -lrt /home/aditya/Downloads
total 0
```

- 3) cd : change directory



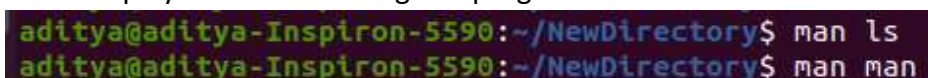
```
aditya@aditya-Inspiron-5590:~$ cd /home/aditya/Downloads
aditya@aditya-Inspiron-5590:~/Downloads$ cd -
/home/aditya
aditya@aditya-Inspiron-5590:~$ cd
```

- 4) mkdir : create a new directory



```
aditya@aditya-Inspiron-5590:~$ mkdir ./NewDirectory
aditya@aditya-Inspiron-5590:~$ cd ./NewDirectory
aditya@aditya-Inspiron-5590:~/NewDirectory$ ls
aditya@aditya-Inspiron-5590:~/NewDirectory$
```

- 5) man : display the manual of a given program



```
aditya@aditya-Inspiron-5590:~/NewDirectory$ man ls
aditya@aditya-Inspiron-5590:~/NewDirectory$ man man
```

```

aditya@aditya-laptop: ~/NewDirectory
ls(1)
NAME
ls - list directory contents

SYNOPSIS
ls [OPTION]... [FILE]...

DESCRIPTION
List information about the FILES (the current directory by default).  Sort entries alphabetically if none of -eftuvsX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.

-a, --all
    do not ignore entries starting with .

-A, --almost-all
    do not list implied . and ..

--author
    with -l, print the author of each file

-B, --escape
    print C-style escapes for nongraphical characters

--block-size=SIZE
    with -l, scale sizes by SIZE when printing them; e.g., '--block-size=M'; see SIZE format below

-B, --ignore-backups
    do not list implied entries ending with ~

-c
    with -lt: sort by, and show, ctime (time of last modification of file status information); with -l: show ctime and sort by name; otherwise: sort by ctime, newest first

-C
    list entries by columns

--color[=MODE]
    colorize the output; MODE can be 'always' (default if omitted), 'auto', or 'never'; more info below

-d, --directory
    list directories themselves, not their contents

-D, --dired
    generate output designed for Emacs' dired mode

-F
    do not sort, enable -d, disable -ls --color

-F, --classify
    append indicator (one of */=>) to entries

--file-type
    likewise, except do not append '*'

--format=FORMAT
    across -a, commas on, horizontal -a, long -l, single-column -l, verbose -l, vertical -C

*** Manual page ls(1) line 1728 248 (press h for help or q to quit)

```

A screenshot of a Linux terminal window titled "Terminal". The address bar shows "ssh@eddye-laptop-5390 ~NewDirectory". The terminal displays the manual page for the 'utils' section. It includes sections for NAME, SYNOPSIS, DESCRIPTION, and EXAMPLES. The DESCRIPTION section explains how 'man' works and lists various file formats it can handle. The EXAMPLES section shows how to view specific manual pages like 'ls'.

```
NAME
    man - an interface to the system reference manuals

SYNOPSIS
    man [man options] [section:]page ...
    man -k [search options] keyword ...
    man -t [man options] [section:]term ...
    man -f [shell's filename] page ...
    man -l [man options] file ...
    man -w [-W [man options]] page ...

DESCRIPTION
    man is the system's manual pager. Each page argument given to man is normally the name of a program, utility or function. The manual page associated with each of these arguments is then found and displayed. A section, if provided, will direct man to look only in that section of the manual. The default action is to search in all of the available sections following a pre-defined order (see DEFAULTS), and to show only the first page found, even if page exists in several sections.

    The table below shows the section numbers of the manual followed by the types of pages they contain.

        1 Executable programs or shell commands
        2 System calls (functions provided by the kernel)
        3 Library calls (functions within program libraries)
        4 Special files (usually found in /dev)
        5 File formats and conventions, e.g. /etc/passwd
        6 Games
        7 Miscellaneous (including macro packages and conventions), e.g. man(7), groff(7)
        8 System administration commands (usually only for root)
        9 Kernel routines [No standard]

    A manual page consists of several sections. Conventional section names include NAME, SYNOPSIS, CONFIGURATION, DESCRIPTION, OPTIONS, EXIT STATUS, RETURN VALUE, ERRORS, ENVIRONMENT, FILES, VERSIONS, CONFORMING TO, NOTES, BUGS, EXAMPLE, AUTHORS, and SEE ALSO.

    The following conventions apply to the SYNOPSIS section and can be used as a guide in other sections.

        Bold test           type exactly as shown.
        Italic test         replace with appropriate argument;
                            one or all arguments within [ ] are optional.
        -x|-y               options delimited by | cannot be used together.
        argument ...       argument is repeatable.
        {expression} ...    entire expression within [ ] is repeatable.

    Exact rendering may vary depending on the output device. For instance, man will usually not be able to render italics when running in a terminal, and will typically use underlined or coloured text instead.

    The command or function illustration is a pattern that should match all possible invocations. In some cases it is advisable to illustrate several exclusive invocations as is shown in the SYNOPSIS section of this manual page.

EXAMPLES
    man ls
    Display the manual page for the ls (program) ls.

    man page man(1) line 1 ignore & for help on & to exit.
```

6) **echo** : to create a file  
SYNTAX: **echo** 'Text to be added' > filename

```
aditya@aditya-Inspiron-5590:~/NewDirectory$ echo 'This is a test file' > data.txt
aditya@aditya-Inspiron-5590:~/NewDirectory$ ls
data.txt
```

7) mv : move or rename a file

```
aditya@aditya-Inspiron-5590:~/NewDirectory$ mv data.txt newData.txt
aditya@aditya-Inspiron-5590:~/NewDirectory$ ls
newData.txt
```

8) cp : copy a file

```
aditya@aditya-Inspiron-5590:~/NewDirectory$ cp newData.txt data.txt
aditya@aditya-Inspiron-5590:~/NewDirectory$ ls
data.txt  newData.txt
```

9) chmod : change the permissions of a file

```
aditya@aditya-Inspiron-5590:~/NewDirectory$ chmod a+r data.txt
```

10) rm : delete a file forever

```
aditya@aditya-Inspiron-5590:~/NewDirectory$ rm newData.txt
aditya@aditya-Inspiron-5590:~/NewDirectory$ ls
data.txt
```

11) more : displays the contents of a file page by page

less : a replacement for more, displays data more nicely and makes search easy

```
aditya@aditya-Inspiron-5590:~/NewDirectory$ more data.txt
This is a test file
aditya@aditya-Inspiron-5590:~/NewDirectory$ less data.txt
```



12) find : search for files across directories

```
aditya@aditya-Inspiron-5590:~$ find ./ -name 'da*.txt'
./NewDirectory/data.txt
```



13) df : check how much space is left

```
aditya@aditya-Inspiron-5590:~$ df
Filesystem      1K-blocks    Used Available Use% Mounted on
udev            3935372         0   3935372  0% /dev
tmpfs           793000      1940    791060  1% /run
/dev/sda5       56958080 5118240   48916784 10% /
tmpfs           3964988         0   3964988  0% /dev/shm
tmpfs           5120         4      5116  1% /run/lock
tmpfs           3964988         0   3964988  0% /sys/fs/cgroup
/dev/loop0      224256     224256         0 100% /snap/gnome-3-34-1804/72
/dev/loop1      56832      56832         0 100% /snap/core18/2128
/dev/loop2      66688      66688         0 100% /snap/gtk-common-themes/1515
/dev/loop3      52224     52224         0 100% /snap/snap-store/547
/dev/loop4      33152     33152         0 100% /snap/snapd/12704
tmpfs           792996       24    792972  1% /run/user/1000

aditya@aditya-Inspiron-5590:~$ df .
Filesystem      1K-blocks    Used Available Use% Mounted on
/dev/sda5       56958080 5118240   48916784 10% /
```

14) top : list processes in order of CPU storage

```
top - 01:43:09 up 21 min, 1 user, load average: 0.02, 0.04, 0.10
tasks: 283 total, 1 running, 282 sleeping, 0 stopped, 0 zombie
MiB Mem : 8.5 total, 0.5 free, 8.0 us, 0.5 bu, 0.0 ca, 0.0 hi, 0.0 sy, 0.0 so
MiB Swap : 7940.0 total, 8255.0 free, 685.0 used, 606.7 buff/cache
MiB Swap: 0 total, 0 free, 0 used, 606.7 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR   S    CPU   MEM COMMAND
 1000 root        0   0 644216 32832 1000+  S    0.0   0.4  rsyncd
 234 root        0   0 24736 32842 7272+  S    0.0   0.3  rsync
1000 aditya      0   0 36412 4216 330+  S    0.0   0.1  top
 1 root        0   0 157416 8130 8300+  S    0.0   0.1  systemd
 2 root        0   0 0 0 0  S    0.0   0.0  kthreadd
 3 root        0 -20 0 0 0  S    0.0   0.0  rcu_gp
 4 root        0 -20 0 0 0  S    0.0   0.0  rcu_bh
 5 root        0 -20 0 0 0  S    0.0   0.0  kworker/0:0H-kblockd
 6 root        0 -20 0 0 0  S    0.0   0.0  rcu_sched
 7 root        0 -20 0 0 0  S    0.0   0.0  rcu_wk
 8 root        0 -20 0 0 0  S    0.0   0.0  kworker/0:0H-events_highpri
 9 root        0 -20 0 0 0  S    0.0   0.0  kworker/0:0H-rcu_tasks_rude
10 root        0 -20 0 0 0  S    0.0   0.0  kworker/0:0H-rcu_tasks_trace
11 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
12 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
13 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
14 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
15 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
16 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
17 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
18 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
19 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
20 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
21 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
22 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
23 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
24 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
25 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
26 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
27 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
28 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
29 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
30 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
31 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
32 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
33 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
34 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
35 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
36 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
37 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
38 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
39 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
40 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
41 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
42 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
43 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
44 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
45 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
46 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
47 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
48 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
49 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
50 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
51 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
52 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
53 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
54 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
55 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
56 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
57 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
58 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
59 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
60 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
61 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
62 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
63 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
64 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
65 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
66 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
67 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
68 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
69 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
70 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
71 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
72 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
73 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
74 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
75 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
76 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
77 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
78 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
79 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
80 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
81 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
82 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
83 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
84 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
85 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
86 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
87 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
88 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
89 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
90 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
91 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
92 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
93 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
94 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
95 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
96 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
97 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
98 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
99 root        0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
100 root       0 0 0 0 0  S    0.0   0.0  kworker/0:0H-kfsync
```

## OS LAB 2

S NO.	TITLE	DATE OF IMPLEMENTATION	REMARKS
1	Program to implement Round Robin Process of CPU Scheduling	17-01-2022	
2	Program to implement First Come First Serve Process of CPU Scheduling	24-01-2022	



# ROUND ROBIN CPU SCHEDULING

CRITERIA: Time Quantum

MODE: Pre-emptive

GIVEN: Time Quantum and list of processes with their arrival and burst time.

CODE:

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

class process{
public:
    int id;
    int arrivalTime;
    int burstTime;
    int remainingBurstTime;
    int completionTime;
    int TAT;
    int WT;
    int RT;
    bool isRunning;
};

void round_robin_scheduling(vector<process> &v, int quantum){
    queue<process> q;
    stack<process> s;
    int cur_time = 0;
    int id=0;
    while(true){
        // Add all the processes that have arrived to the ready queue.
        while(id<v.size() && v[id].arrivalTime<=cur_time){
            q.push(v[id]);
            id++;
        }
        // Add the last process from running queue at the end of ready queue if it is not completed.
        if(!s.empty()){
            process p = s.top();
            if(p.remainingBurstTime>0){
                q.push(p);
            }
        }

        // If ready queue is empty => no process to be completed => stop.
        if(q.empty()){
            break;
        }

        // Pick the front process from the ready queue for processing by the CPU.
        process rning_proc = q.front();
        q.pop();
        // Store the Response time for a process the first time it reaches the CPU.
        if(rning_proc.isRunning==false){
```

```

        v[rning_proc.id].RT = cur_time-v[rning_proc.id].arrivalTime;
        rning_proc.isRunning=true;
    }
    // If the remaining burst time > quantum, the process is not complete (CONTEXT SWITCHING)
    if(rning_proc.remainingBurstTime>=quantum){
        rning_proc.remainingBurstTime-=quantum;
        cur_time+=quantum;
    }
    // Process is complete
    else{
        cur_time+=rning_proc.remainingBurstTime;
        rning_proc.remainingBurstTime=0;
    }
    s.push(rning_proc);
    // If process is complete, store the completion time for the process
    if(rning_proc.remainingBurstTime==0){
        v[rning_proc.id].completionTime = cur_time;
    }

}
return;
}

```

```

int main(){

    cout<<"ROUND ROBIN CPU SCHEDULING ALGORITHM C++ IMPLEMENTATION\n";
    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int quantum = 0;
    int n=0;
    cout<<"Enter the number of processes: ";
    cin>>n;
    cout<<"Enter value of time quantum: ";
    cin>>quantum;

    cout<<"Enter the arrival times and burst times of "<<n<<" processes: \n";

    vector<process> v(n);
    for(int i=0; i<n; i++){
        cin>>v[i].arrivalTime>>v[i].burstTime;
        v[i].id = i;
        v[i].isRunning = false;
        v[i].remainingBurstTime = v[i].burstTime;
    }

    round_robin_scheduling(v, quantum);
    int t_TAT=0;
    int t_CT=0;
    for(int i=0; i<n; i++){
        v[i].TAT = v[i].completionTime-v[i].arrivalTime;
        v[i].WT = v[i].TAT-v[i].burstTime;
    }
}

```

```

        t_TAT+=v[i].TAT;
        t_CT+=v[i].completionTime;
    }

    for(auto p:v){
        cout<<"Process: "<<p.id<<"\tArrival Time:"<<p.arrivalTime<<"\tBurst
Time:"<<p.burstTime<<"\tCompletion Time:"<<p.completionTime;
        cout<<"\tTurn Around Time:"<<p.TAT<<"\tWaiting Time:"<<p.WT<<"\tResponse Time:"<<p.RT<<"\n";
    }

    cout<<"\nAverage Turn Around Time: "<<(float)((1.0*t_TAT)/(1.0*n))<<"\n";
    cout<<"\nAverage Completion Time: "<<(float)((1.0*t_CT)/(1.0*n))<<"\n";

    return 0;
}

```

# FIRST COME FIRST SERVE CPU SCHEDULING

CRITERIA: Arrival Time

MODE: Non pre-emptive

GIVEN: List of processes with their arrival and burst time.

CODE:

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

class process{
public:
    int id;
    int arrivalTime;
    int burstTime;
    int completionTime;
    int TAT;
    int WT;
    int RT;
};

void FCFS(vector<process> &v){
    int cur_time = 0;
    int id = 0;
    for(int i=0; i<v.size(); i++){
        if(cur_time<v[i].arrivalTime){
            cout<<"CPU idle from "<<cur_time<<" to "<<v[i].arrivalTime<<endl;
            cur_time = v[i].arrivalTime;
        }
        v[i].completionTime = cur_time+v[i].burstTime;
        v[i].RT = cur_time-v[i].arrivalTime;

        cout<<"Process P"<<v[i].id+1<<": start time = "<<cur_time<<" completion time = 
"<<v[i].completionTime<<endl;
        cur_time+=v[i].burstTime;
    }
}

int main(){

    cout<<"FIRST COME FIRST SERVE CPU SCHEDULING ALGORITHM C++ IMPLEMENTATION\n";
    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int n=0;
    cout<<"Enter the number of processes: ";
    cin>>n;

    cout<<"Enter the arrival times and burst times of "<<n<<" processes: \n";

    vector<process> v(n);
```

```

for(int i=0; i<n; i++){
    cin>>v[i].arrivalTime>>v[i].burstTime;
    v[i].id = i;
}

cout<<"-----\n";
cout<<"\n";
FCFS(v);
cout<<"\n";
cout<<"-----\n";
cout<<"\n\n";

int t_TAT=0;
int t_CT=0;
for(int i=0; i<n; i++){
    v[i].TAT = v[i].completionTime-v[i].arrivalTime;
    v[i].WT = v[i].TAT-v[i].burstTime;
    t_TAT+=v[i].TAT;
    t_CT+=v[i].completionTime;
}

for(auto p:v){
    cout<<"Process: "<<p.id<<"\tArrival Time:"<<p.arrivalTime<<"\tBurst
Time:"<<p.burstTime<<"\tCompletion Time:"<<p.completionTime;
    cout<<"\tTurn Around Time:"<<p.TAT<<"\tWaiting Time:"<<p.WT<<"\tResponse Time:"<<p.RT<<"\n";
}

cout<<"\nAverage Turn Around Time: "<<(float)((1.0*t_TAT)/(1.0*n))<<"\n";
cout<<"\nAverage Completion Time: "<<(float)((1.0*t_CT)/(1.0*n))<<"\n";

return 0;
}

```

## OS LAB 3

S NO.	TITLE	DATE OF IMPLEMENTATION	REMARKS
1	Program to implement Shortest Job First Process of CPU Scheduling	02-02-2022	
2	Program to implement First Come First Serve Process of CPU Scheduling	02-02-2022	
3	Program to implement Priority based Scheduling Process of CPU Scheduling	02-02-2022	

# SHORTEST JOB FIRST CPU SCHEDULING

CRITERIA: Burst Time

NOTE: In case of same burst time, process with lower arrival time is executed first.

MODE: Non pre-emptive

GIVEN: List of processes with their arrival and burst time.

CODE:

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

class process{
public:
    int priority;
    int id;
    int arrivalTime;
    int burstTime;
    bool ready;
    int completionTime;
    int TAT;
    int WT;
    int RT;
};

struct comp{
    bool operator()(process const &p1, process const &p2){
        return p1.burstTime > p2.burstTime;
    }
};

void SJF(vector<process> &v){
    priority_queue<process, vector<process>, comp> p;
    int cur_time = INT_MAX;
    int n=v.size();
    for(int i=0; i<n; i++){
        cur_time = min(cur_time, v[i].arrivalTime);
    }

    int count = 0;
    while(true){
        for(int i=0; i<n; i++){
            if(v[i].arrivalTime<=cur_time && !v[i].ready){
                v[i].ready = true;
                p.push(v[i]);
                count++;
            }
        }
        if(count<n && p.empty()){
            cout<<"CPU empty from "<<cur_time<<" to "<<cur_time+1<<"\n";
            cur_time++;
            continue;
        }
    }
}
```



```

    }
    if(p.empty()){
        break;
    }
    process cur_process = p.top();
    p.pop();
    v[cur_process.id].RT = cur_time-cur_process.arrivalTime;
    cur_time+=cur_process.burstTime;
    v[cur_process.id].completionTime=cur_time;
}
}
int main(){
    cout<<"SHORTEST JOB FIRST CPU SCHEDULING ALGORITHM C++ IMPLEMENTATION\n";
    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int n=0;
    cout<<"Enter the number of processes: ";
    cin>>n;

    cout<<"Enter the arrival times and burst times of "<<n<<" processes: \n";

    vector<process> v(n);
    for(int i=0; i<n; i++){
        cin>>v[i].arrivalTime>>v[i].burstTime;
        v[i].id = i;
        v[i].ready = false;
    }
    cout<<"-----\n";
    cout<<"\n";
    SJF(v);
    cout<<"\n";
    cout<<"-----\n";
    cout<<"\n\n";

    int t_TAT=0;
    int t_CT=0;
    for(int i=0; i<n; i++){
        v[i].TAT = v[i].completionTime-v[i].arrivalTime;
        v[i].WT = v[i].TAT-v[i].burstTime;
        t_TAT+=v[i].TAT;
        t_CT+=v[i].completionTime;
    }

    for(auto p:v){
        cout<<"Process: "<<p.id<<"\tArrival Time:"<<p.arrivalTime<<"\tBurst
Time:"<<p.burstTime<<"\tCompletion Time:"<<p.completionTime;
        cout<<"\tTurn Around Time:"<<p.TAT<<"\tWaiting Time:"<<p.WT<<"\tResponse Time:"<<p.RT<<"\n";
    }

    cout<<"\nAverage Turn Around Time: "<<(float)((1.0*t_TAT)/(1.0*n))<<"\n";
    cout<<"\nAverage Completion Time: "<<(float)((1.0*t_CT)/(1.0*n))<<"\n";

    return 0;
}

```

# FIRST COME FIRST SERVE CPU SCHEDULING

CRITERIA: Arrival Time

MODE: Non pre-emptive

GIVEN: List of processes with their arrival and burst time.

CODE:

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

class process{
public:
    int id;
    int arrivalTime;
    int burstTime;
    int completionTime;
    int TAT;
    int WT;
    int RT;
};

void FCFS(vector<process> &v){
    int cur_time = 0;
    int id = 0;
    for(int i=0; i<v.size(); i++){
        if(cur_time<v[i].arrivalTime){
            cout<<"CPU idle from "<<cur_time<<" to "<<v[i].arrivalTime<<endl;
            cur_time = v[i].arrivalTime;
        }
        v[i].completionTime = cur_time+v[i].burstTime;
        v[i].RT = cur_time-v[i].arrivalTime;

        cout<<"Process P"<<v[i].id+1<<": start time = "<<cur_time<<" completion time = 
"<<v[i].completionTime<<endl;
        cur_time+=v[i].burstTime;
    }
}

int main(){

    cout<<"FIRST COME FIRST SERVE CPU SCHEDULING ALGORITHM C++ IMPLEMENTATION\n";
    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int n=0;
    cout<<"Enter the number of processes: ";
    cin>>n;

    cout<<"Enter the arrival times and burst times of "<<n<<" processes: \n";

    vector<process> v(n);
```

```

for(int i=0; i<n; i++){
    cin>>v[i].arrivalTime>>v[i].burstTime;
    v[i].id = i;
}

cout<<"-----\n";
cout<<"\n";
FCFS(v);
cout<<"\n";
cout<<"-----\n";
cout<<"\n\n";

int t_TAT=0;
int t_CT=0;
for(int i=0; i<n; i++){
    v[i].TAT = v[i].completionTime-v[i].arrivalTime;
    v[i].WT = v[i].TAT-v[i].burstTime;
    t_TAT+=v[i].TAT;
    t_CT+=v[i].completionTime;
}

for(auto p:v){
    cout<<"Process: "<<p.id<<"\tArrival Time:"<<p.arrivalTime<<"\tBurst
Time:"<<p.burstTime<<"\tCompletion Time:"<<p.completionTime;
    cout<<"\tTurn Around Time:"<<p.TAT<<"\tWaiting Time:"<<p.WT<<"\tResponse Time:"<<p.RT<<"\n";
}

cout<<"\nAverage Turn Around Time: "<<(float)((1.0*t_TAT)/(1.0*n))<<"\n";
cout<<"\nAverage Completion Time: "<<(float)((1.0*t_CT)/(1.0*n))<<"\n";

return 0;
}

```

# PRIORITY BASED CPU SCHEDULING

CRITERIA: Priority (higher the value, greater the priority)

NOTE: In case of same priority, process with lower arrival time is executed first.

MODE: Non pre-emptive

GIVEN: List of processes with their arrival and burst time.

CODE:

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

class process{
public:
    int priority;
    int id;
    int arrivalTime;
    int burstTime;
    bool ready;
    int completionTime;
    int TAT;
    int WT;
    int RT;
};

struct comp{
    bool operator()(process const &p1, process const &p2){
        return p1.priority < p2.priority;
    }
};

void PriorityBasedScheduling(vector<process> &v){
    priority_queue<process, vector<process>, comp> p;
    int cur_time = INT_MAX;
    int n=v.size();
    for(int i=0; i<n; i++){
        cur_time = min(cur_time, v[i].arrivalTime);
    }

    int count = 0;
    while(true){
        for(int i=0; i<n; i++){
            if(v[i].arrivalTime<=cur_time && !v[i].ready){
                v[i].ready = true;
                p.push(v[i]);
                count++;
            }
        }
        if(count<n && p.empty()){
            cout<<"CPU empty from "<<cur_time<<" to "<<cur_time+1<<"\n";
            cur_time++;
            continue;
        }
    }
}
```

```

    }
    if(p.empty()){
        break;
    }

    process cur_process = p.top();
    p.pop();

    v[cur_process.id].RT = cur_time-cur_process.arrivalTime;
    cur_time+=cur_process.burstTime;
    v[cur_process.id].completionTime=cur_time;
}
}

int main(){

    cout << "PRIORITY BASED CPU SCHEDULING ALGORITHM C++ IMPLEMENTATION\n";
    cout << "Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int n = 0;
    cout << "Enter the number of processes: ";
    cin >> n;

    cout << "Enter the arrival times and burst times and priority values of " << n << " processes: \n";

    vector<process> v(n);
    for (int i = 0; i < n; i++){
        cin >> v[i].arrivalTime >> v[i].burstTime >> v[i].priority;
        v[i].ready = false;
        v[i].id = i;
    }

    cout << "-----\n";
    cout << "\n";
    PriorityBasedScheduling(v);
    cout << "\n";
    cout << "-----\n";
    cout << "\n\n";

    int t_TAT = 0;
    int t_CT = 0;
    for (int i = 0; i < n; i++){
        v[i].TAT = v[i].completionTime - v[i].arrivalTime;
        v[i].WT = v[i].TAT - v[i].burstTime;
        t_TAT += v[i].TAT;
        t_CT += v[i].completionTime;
    }

    for (auto p : v){
        cout << "Process: " << p.id << "\tArrival Time:" << p.arrivalTime << "\tBurst Time:" <<
p.burstTime << "\tCompletion Time:" << p.completionTime;
        cout << "\tTurn Around Time:" << p.TAT << "\tWaiting Time:" << p.WT << "\tResponse Time:" << p.RT
<< "\n";
    }
}

```

```
cout << "\nAverage Turn Around Time: " << (float)((1.0 * t_TAT) / (1.0 * n)) << "\n";
cout << "\nAverage Completion Time: " << (float)((1.0 * t_CT) / (1.0 * n)) << "\n";

return 0;
}
```

## OS LAB 4

S NO.	TITLE	DATE OF IMPLEMENTATION	REMARKS
1	Program to handle the Critical Section Problem using semaphore	09-02-2022	



# CRITICAL SECTION PROBLEM SOLUTION USING SEMAPHORE

The Critical Section is the part of the program where shared resources are accessed by multiple processes.

When two or more process try to access the critical section at the same time, errors such as race condition may arise. This is known as the **critical section problem**.

**Semaphore** is an integer variable which is used in mutually exclusive manner by various concurrent cooperative processes in order to achieve process synchronization.

CODE:

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

queue<int> ready;

class semaphore{
    int value;
public:
    queue<int> blocked_list, CS;

    semaphore(){
        this->value = 9; // Last digit of my roll number.
    }

    void down(int p){          // Entry code before entering Critical Section
        this->value--;
        if(this->value<0){
            // Put the process in the blocked list
            blocked_list.push(p);
            cout<<"Process "<<p<<" has been put in the blocked list\n";
        }
        else{
            // Put the process in the Critical Section
            CS.push(p);
            cout<<"Process "<<p<<" has entered the Critical Section\n";
        }
    }

    void up(){
        this->value++;
        if(!CS.empty()){
            int p=CS.front();
            cout<<"Process "<<p<<" has exited the Critical Section\n";
            CS.pop();
        }
        if(this->value<=0){
            // Wake up a sleeping process from the blocked list
            int p=blocked_list.front();
            cout<<"Process "<<p<<" has been removed from the blocked list\n";
            blocked_list.pop();
            // Put the blocked process back in the ready queue
        }
    }
}
```

```

        ready.push(p);
    }
}

void inCS(){
    if(CS.empty()){
        cout<<"The Critical Section is empty\n";
    }
    else{
        queue<int> temp = CS;
        cout<<"Processes present in Critical Section are: ";
        while(!temp.empty()){
            cout<<temp.front()<<" ";
            temp.pop();
        }
        cout<<"\n";
    }
}

void inBlockedList(){
    if(blocked_list.empty()){
        cout<<"The Blocked List is empty\n";
    }
    else{
        queue<int> temp = blocked_list;
        cout<<"Processes present in Blocked List are: ";
        while(!temp.empty()){
            cout<<temp.front()<<" ";
            temp.pop();
        }
        cout<<"\n";
    }
}

};

int main(){

    cout<<"PROCESS SYNCHRONISATION USING SEMAPHORE C++ IMPLEMENTATION\n";
    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int n=0;
    cout<<"Enter number of processes: ";
    cin>>n;
    for(int i=1; i<=n; i++){
        ready.push(i);
    }

    semaphore S;

    int itr=0;

    while(!ready.empty()){
        itr++;
        while(!ready.empty()){

```

```
    int process = ready.front();
    ready.pop();
    S.down(process);          // Each process tries to enter the critical section
    if(itr!=1){
        break;
    }
}

S.inCS();
cout<<"\n";
S.inBlockedList();
cout<<"\n\n";

while(!S.CS.empty()){
    S.up();
}

return 0;
}
```

## OS LAB 5

S NO.	TITLE	DATE OF IMPLEMENTATION	REMARKS
1	Program to handle the Critical Section Problem using semaphore	16-02-2022	

# PRODUCER-CONSUMER PROBLEM SOLUTION USING SEMAPHORE AND MUTEX

The producer-consumer problem is an example of a [multi-process synchronization](#) problem. The problem describes two processes, the producer and the consumer that shares a common fixed-size buffer use it as a queue.

The producer's job is to generate data, put it into the buffer, and start again.

At the same time, the consumer is consuming the data (i.e., removing it from the buffer), one piece at a time.

**Problem:** Given the common fixed-size buffer, the task is to make sure that the producer can't add data into the buffer when it is full and the consumer can't remove data from an empty buffer.

**Solution:** The producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same manner, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer.

CODE:

```
// Solution of the Producer Consumer Problem using semaphore
```

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

```
// Initially we have an empty buffer with a size 5
int m=1;           // Mutex
int full=0;        // Counting Semaphore
int empty=5;       // Counting Semaphore
int IN=0;
int OUT=0;
```

```
void down(int &s){
    s--;
}
```

```
void up(int &s){
    s++;
}
```

```
void producer(){
    down(m);
    up(full);
    down(empty);

    IN++;
    cout<<"\nProducer produced the item "<<IN;
    up(m);
}
```

```
void consumer(){
    down(m);
    down(full);
    up(empty);
```

```

    OUT++;
    cout<<"\nConsumer consumed the item "<<OUT;
    up(m);
}

int main(){
    cout<<"SOLUTION OF PRODUCER CONSUMER PROBLEM USING SEMAPHORE AND MUTEX C++
IMPLEMENTATION\n";
    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int n=0;
    while(1){
        cout<<"Select the preference: (1, 2, or 3) \n";
        cout<<"1) Producer\n2)Consumer\n3)Exit\n";

        cin>>n;

        if(n==1){
            if(m==1 && empty!=0){
                producer();
                cout<<endl;
            }
            else{
                cout<<"Buffer is full\n";
            }
        }
        else if(n==2){
            if(m==1 && full!=0){
                consumer();
                cout<<endl;
            }
            else{
                cout<<"Buffer is empty\n";
            }
        }
        else{
            break;
        }
    }
    return 0;
}

```

## OS LAB 6

S NO.	TITLE	DATE OF IMPLEMENTATION	REMARKS
1	Simulation of Bankers Deadlock Avoidance and Prevention algorithms	23-02-2022	



# SIMULATION OF BANKERS DEADLOCK AVOIDANCE AND PREVENTION ALGORITHM

The **banker's algorithm** is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

It checks if allocation of any resource will lead to deadlock or not, OR is it safe to allocate a resource to a process and if not then resource is not allocated to that process. Determining a safe sequence (even if there is only 1) will assure that system will not go into deadlock.

Banker's algorithm is generally used to find if a safe sequence exist or not.

CODE:

```
// C++ implementation of Banker's deadlock avoidance and prevention algorithm
#include<bits/stdc++.h>
using namespace std;

// Class representing state of the system: how various resources are allocated and requested by various processes
struct state{
    int resources, processes;
    vector<vector<int> > allocated;
    vector<vector<int> > max_req;
    vector<vector<int> > remaining_req;
    vector<int> available;
    vector<bool> executed;

    state(int res, int pro){
        this->resources = res;
        this->processes = pro;

        allocated.resize(pro, vector<int>(res, 0));
        max_req.resize(pro, vector<int>(res, 0));
        remaining_req.resize(pro, vector<int>(res, 0));
        available.resize(res, 0);
        executed.resize(pro, false);
        cout<<"executed";
    }
};

queue<int> safe_seq;

bool bankers(state s){
    int n=s.processes;
    while(n--){
        cout<<"Current availability: ";
        for(int i=0; i<s.resources; i++){
            cout<<s.available[i]<<" ";
        }
        cout<<"\n";
    }
}
```

```

    bool found=false;
    for(int i=0; i<s.processes; i++){
        bool t=true;
        if(!s.executed[i]){
            for(int j=0; j<s.resources; j++){
                if(s.available[j]<s.remaining_req[i][j]){
                    t=false;
                    break;
                }
            }
            if(t){
                found=true;
                safe_seq.push(i+1);
                s.executed[i]=true;
                cout<<"P"<<i+1<<" has been allocated the resources\n\n";
                for(int j=0; j<s.resources; j++){
                    s.available[j]+=s.allocated[i][j];
                }
                break;
            }
        }
    }
    if(!found){
        cout<<"Resources could not be allocated to any process.\n\n";
        return true;
    }
}
return false;
}

int main(){
    cout<<"BANKER'S DEADLOCK AVOIDANCE AND PREVENTION ALGORITHM C++ IMPLEMENTATION\n";
    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int res=0, pro=0;
    cout<<"Enter number of resources: ";
    cin>>res;
    cout<<"Enter number of processes: ";
    cin>>pro;

    state s(res, pro);

    cout<<"For "<<pro<<" processes enter: \n";
    for(int i=0; i<pro; i++){
        cout<<"Resources allocated to process "<<i+1<<": ";
        for(int j=0; j<res; j++){
            cin>>s.allocated[i][j];
        }
        cout<<"Max resources required by process "<<i+1<<": ";
        for(int j=0; j<res; j++){
            cin>>s.max_req[i][j];
            s.remaining_req[i][j]=s.max_req[i][j]-s.allocated[i][j];
        }
    }
}

```

```

}

cout<<"Enter the currently available resources: ";
for(int i=0; i<res; i++){
    cin>>s.available[i];
}

cout<<"\n\n-----
\n\n";
bool deadlock = bankers(s);
cout<<"\n\n-----
\n\n";

if(deadlock){
    cout<<"Deadlock Situation Detected!\n";
}
else{
    cout<<"No Deadlock Detected!\n";
    cout<<"Safe sequence of resource allocation: ";
    while(!safe_seq.empty()){
        cout<<"P"<<safe_seq.front()<<" ";
        safe_seq.pop();
    }
    cout<<"\n";
}
return 0;
}

```

# OS LAB 7

S NO.	TITLE	DATE OF IMPLEMENTATION	REMARKS
1	Implementation of solution of Reader-Writer Problem	02-03-2022	
2	Implementation of solution of Sleeping Barber Problem	02-03-2022	

# IMPLEMENTATION OF SOLUTION OF READER-WRITER PROBLEM

Consider a situation where we have a file shared between many people.

If one of the people tries editing the file, no other person should be reading or writing at the same time, otherwise changes will not be visible to him/her. However if some person is reading the file, then others may read it at the same time.

Precisely in OS we call this situation as the **readers-writers problem**

Problem parameters:

- One set of data is shared among a number of processes
- Once a writer is ready, it performs its write. Only one writer may write at a time
- If a process is writing, no other process can read it
- If at least one reader is reading, no other process can write
- Readers may not write and only read

**Solution:** We create two binary semaphores, mutex- to synchronise the increment and decrement of the reader\_count variable and db- to synchronise and restrict access to readers and writers based on the problem statement conditions. We also have a struct database to represent an actual database of the real world.

CODE:

```
// C++ implementation of solution of Reader-Writer Problem using Binary Semaphore (Process Synchronisation)
#include<bits/stdc++.h>
using namespace std;

struct Semaphore{
    bool s;

    Semaphore(){
        this->s=1;
    }

    void down(){
        if(this->s==0){
            cout<<"RESTRICTED!\n";
            return;
        }
        this->s=this->s-1;
    }

    void up(){
        this->s=this->s+1;
    }
};

// structure to represent the database
struct database{
    int value;
};

int rc=0; // Reader Count
```

```

void Reader(Semaphore &mutex, Semaphore &db, database &d){
    mutex.down();
    rc++;
    if(rc==1){
        db.down();
    }
    mutex.up();

    // -----
    cout<<"Reader "<<rc<<" is reading the database\n";
    cout<<"Value stored in the database = "<<d.value<<"\n\n";
    // -----

    mutex.down();
    rc--;
    if(rc==0){
        db.up();
    }
    mutex.up();
}

void Writer(Semaphore &mutex, Semaphore &db, database &d, int val){
    db.down();

    // -----
    cout<<"Writer is writing in the database\n";
    d.value=val;
    cout<<"Value updated!\n\n";
    // -----

    db.up();
}

int main(){
    cout<<"SOLUTION TO READER-WRITER PROBLEM USING SEMAPHORE C++ IMPLEMENTATION\n";
    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    Semaphore mutex;    // Binary semaphore to synchronise incrementing of reader count
    Semaphore db;        // Binary semaphore to synchronise the access to a database

    database d;
    d.value = 9;        // Suppose the database stores the value 9 initially

    while(1){
        int n=0;
        cout<<"Enter \n1 for reader\n2 for writer\n3 to exit\n";
        cin>>n;
        if(n==1){
            Reader(mutex, db, d);
        }
        else if(n==2){
            cout<<"Enter the value you want to write: ";
            int val=0;

```

```

        cin>>val;
        Writer(mutex, db, d, val);
    }
    else break;
}

return 0;
}

```

## IMPLEMENTATION OF SOLUTION OF SLEEPING BARBER PROBLEM

**Problem :** The analogy is based upon a hypothetical barber shop with one barber. There is a barber shop which has one barber, one barber chair, and n chairs for waiting for customers if there are any to sit on the chair.

- If there is no customer, then the barber sleeps in his own chair.
- When a customer arrives, he has to wake up the barber.
- If there are many customers and the barber is cutting a customer's hair, then the remaining customers either wait if there are empty chairs in the waiting room or they leave if no chairs are empty.

**Solution:** Here we have used 2 binary semaphores, barber- to synchronise sleeping/awake state of the barber and cut- to synchronise the process of cutting of hair. We also have used a counting semaphore freeChairs- to synchronise the number of customers waiting in the waiting room.

In the following code, we have made an assumption that the customers arrive in groups and the next group arrives only when the previous group has been dealt with.

CODE:

```

// C++ implementation of solution of Sleeping Barber Problem using Semaphore (Process Synchronisation)
#include<bits/stdc++.h>
using namespace :: std;

struct binarySemaphore{
    bool s;
};

struct countingSemaphore{
    int s;
};

void customer(binarySemaphore &barber, binarySemaphore &cut, int Customers){
    int id=1;
    while(id<=Customers){
        // Wake the barber up if he is sleeping
        if(barber.s==0){
            barber.s=1;
        }

        // Cut the hair
        cut.s=1;
        cout<<"Customer "<<id<<" is getting a haircut\n\n";
    }
}

```



```

        cut.s=0;

        id++;
    }

    // The barber goes to sleep after tending to all the customers
    barber.s=0;
}

int main(){
    cout<<"SOLUTION TO SLEEPING BARBER PROBLEM USING SEMAPHORE C++ IMPLEMENTATION\n";
    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    binarySemaphore barber;    // denotes if the barber is sleeping or awake
    binarySemaphore cut;        // semaphore to synchronise hair cutting
    countingSemaphore freeChairs;

    cout<<"Enter the number of free chairs: ";
    cin>>freeChairs.s;

    barber.s=0; // initially the barber is sleeping
    cut.s=0;

    // Suppose the customers visit the shop in groups. The group visits the barber only after all the
    customers of first group have left the shop.
    while(1){
        int freeSpace=freeChairs.s;
        int customers=0;
        cout<<"Enter the number of customers entering the shop (enter 0 to exit): ";
        cin>>customers;

        if(customers==0){
            break;
        }

        // The first customer can always go to the sleeping barber
        int id=2;
        while(id<=customers){
            // Chair occupied by customer
            freeChairs.s--;
            id++;
            if(freeChairs.s==0){
                break;
            }
        }

        while(id<=customers){
            cout<<"Customer "<<id<<" returned back without getting a haircut\n";
            id++;
        }

        customer(barber, cut, (freeSpace-freeChairs.s+1));
        freeChairs.s=freeSpace;
    }
}

```

```
}
```

```
return 0;
```

```
}
```

## OS LAB 8

S NO.	TITLE	DATE OF IMPLEMENTATION	REMARKS
1	Implementation of solution of Dining Philosophers Problem	28-03-2022	

# IMPLEMENTATION OF SOLUTION OF DINING PHILOSOPHERS PROBLEM

## The Dining Philosopher Problem:

The Dining Philosopher Problem states that K philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pick up the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both.

CODE:

```
// C++ implementation of solution of Dining Philosopher Problem using Semaphore (Process Synchronisation)
```

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

```
queue<int> ready;
```

```
struct binarySemaphore{
    bool s;

    bool down(){
        if(!s){
            return false;
        }

        s=0;

        return true;
    }

    bool up(){
        if(s){
            return false;
        }

        s=1;
        return true;
    }
};
```

```
struct countingSemaphore{
    int s;

    bool down(){
        if(s==0){
            return false;
        }

        s--;

        return true;
    }
};
```

```

}

bool up(){
    if(s==5){
        return false;
    }

    s++;
    return true;
}

};

void philosopherEat(int phil, vector<binarySemaphore> &fork, queue<int> &temp2){
    cout<<"Philosopher "<<phil<<" is hungry\n";
    if(fork[(phil-1)%5].down()){
        if(fork[(phil)%5].down()){
            cout<<"Philosopher "<<phil<<" has started eating\n";
            temp2.push(phil);
        }
        else{
            fork[(phil-1)%5].up();
            cout<<"Forks are not available for Philosopher "<<phil<<" to use\n";
            ready.push(phil);
            return;
        }
    }
    else{
        cout<<"Forks are not available for Philosopher "<<phil<<" to use\n";
        ready.push(phil);
    }
}

void philosopherFinish(vector<binarySemaphore> &fork, queue<int> &temp2){
    while(!temp2.empty()){
        int phil = temp2.front();
        temp2.pop();

        cout<<"Philosopher "<<phil<<" has finished eating\n";
        fork[(phil-1)%5].up();
        fork[(phil)%5].up();
    }
}

void DiningPhilosopher(){
    vector<binarySemaphore> fork(5);

    // Initially all the forks are available
    for(int i=0; i<5; i++){
        fork[i].s = true;
    }

    while(true){
        if(ready.empty()){
            break;

```

```

    }

    queue<int> temp1, temp2;

    while(!ready.empty()){
        temp1.push(ready.front());
        // temp2.push(ready.front());
        ready.pop();
    }

    while(!temp1.empty()){
        int phil = temp1.front();
        temp1.pop();

        philosopherEat(phil, fork, temp2);
    }

    while(!temp2.empty()){
        philosopherFinish(fork, temp2);
    }
}

}

int main(){
    cout<<"SOLUTION TO DINING PHILOSOPHERS PROBLEM USING SEMAPHORE C++ IMPLEMENTATION\n";
    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    cout<<"There are 5 philosophers(numbered 1-5) sitting on a round table and 5 forks.\n";
    cout<<"Enter the order in which Philosophers get hungry: ";

    for(int i=0; i<5; i++){
        int n=0;
        cin>>n;
        ready.push(n);
    }

    DiningPhilosopher();

    return 0;
}

```

# OS LAB 9

NAME: Aditya Anand

ROLL NO.: 20124009

BRANCH: IT

S NO.	TITLE	DATE OF IMPLEMENTATION	REMARKS
1	Implementation of First In First Out Page Replacement Algorithm	18-04-2022	
2	Implementation of First In First Out Page Replacement Algorithm	18-04-2022	
3	Implementation of First In First Out Page Replacement Algorithm	18-04-2022	
4	Simulation Of Paging Techniques In Memory Management	18-04-2022	

# IMPLEMENTATION OF FIRST IN FIRST OUT PAGE REPLACEMENT ALGORITHM

CODE:

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

int search(vector<int> arr, int key){
    for(int i=0; i<arr.size(); i++){
        if(arr[i]==key){
            return i;
        }
    }
    return -1;
}

void FIFO(vector<int> ref, int frames){
    vector<int> fr(frames, -1);
    int k = 0;
    int hit = 0, miss = 0;

    for(int i=0; i<ref.size(); i++){
        int id = search(fr, ref[i]);
        if(id!=-1){
            hit++;
        }
        else{
            fr[k]=ref[i];
            k=(k+1)%frames;
            miss++;
        }
    }

    cout<<"Hit Percentage: "<<(100.0*hit)/(1.0*(hit+miss))<<"%\n";
    cout<<"Miss Percentage: "<<(100.0*miss)/(1.0*(hit+miss))<<"%\n";
}

int main(){
    cout << "FIRST IN FIRST OUT PAGE REPLACEMENT ALGORITHM C++ IMPLEMENTATION\n";
    cout << "Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int frames = 0;
    cout<<"Enter the number of frames: ";
    cin>>frames;

    if(frames<1){
        cout<<"No frames available!";
        return 0;
    }

    cout<<"Enter the size of the reference string: ";
```



```

int n = 0;
cin>>n;

cout<<"Enter the order in which pages are accessed by the CPU\n";
vector<int> ref(n, 0);
for(int i=0; i<n; i++){
    cin>>ref[i];
}

FIFO(ref, frames);

return 0;
}

```

## IMPLEMENTATION OF LEAST RECENTLY USED PAGE REPLACEMENT ALGORITHM

CODE:

```

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

struct frame{
    int frNo, lastOcc;
};

// returns the index at which the key is found and the index of the least recently used
// page
pair<int, int> Search(vector<frame> &arr, int key, vector<int> ref, int cur){
    int id = -1, lru = INT_MAX;

    for(int i=0; i<arr.size(); i++){
        if(arr[i].frNo==key){
            id = i;
        }
        for(int j=cur-1; j>=0; j--){
            if(ref[j]==arr[i].frNo){
                arr[i].lastOcc = j;
                lru = min(lru, arr[i].lastOcc);
                break;
            }
        }
    }
}

int pRep = -1;
for(int i=0; i<arr.size(); i++){
    if(arr[i].lastOcc == lru){
        pRep = i;
        break;
    }
}

```

```

    }

    return make_pair(id, pRep);
}

void LRU(vector<int> ref, int frames){
    vector<frame> fr(frames);
    for(int i=0; i<frames; i++){
        fr[i].frNo = -1;
        fr[i].lastOcc = -1;
    }
    int hit = 0, miss = 0;
    int k=0, it=0;

    while(k<frames && it<ref.size()){
        bool found = false;
        for(int j=k; j<frames; j++){
            if(fr[j].frNo == ref[it]){
                found=true;
                hit++;
                break;
            }
        }

        if(!found){
            fr[k].frNo = ref[it];
            fr[k].lastOcc = it;
            miss++;
            k++;
        }
        it++;
    }

    for(int i=it; i<ref.size(); i++){
        pair<int, int> p = Search(fr, ref[i], ref, i);
        if(p.first!=-1){
            hit++;
        }
        else{
            fr[p.second].frNo=ref[i];
            fr[p.second].lastOcc=i;
            miss++;
        }
    }

    cout<<"Hit Percentage: "<<(100.0*hit)/(1.0*(hit+miss))<<"%\n";
    cout<<"Miss Percentage: "<<(100.0*miss)/(1.0*(hit+miss))<<"%\n";
}

int main(){
    cout << "LEAST RECENTLY USED REPLACEMENT ALGORITHM C++ IMPLEMENTATION\n";
    cout << "Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int frames = 0;

```

```

    cout<<"Enter the number of frames: ";
    cin>>frames;

    if(frames<1){
        cout<<"No frames available!";
        return 0;
    }

    cout<<"Enter the size of the reference string: ";
    int n = 0;
    cin>>n;

    cout<<"Enter the order in which pages are accessed by the CPU\n";
    vector<int> ref(n);
    for(int i=0; i<n; i++){
        cin>>ref[i];
    }

    LRU(ref, frames);

    return 0;
}

```

## IMPLEMENTATION OF LEAST FREQUENTLY USED PAGE REPLACEMENT ALGORITHM

CODE:

```

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

struct frame{
    int frNo, freq;
};

// returns the index at which the key is found and the index of the least frequently used
page
pair<int, int> Search(vector<frame> &arr, int key, vector<int> ref, int cur){
    int id = -1, lfu = INT_MIN;

    for(int i=0; i<arr.size(); i++){
        if(arr[i].frNo==key){
            id = i;
        }
        lfu = max(lfu, arr[i].freq);
    }

    int pRep = -1;
    for(int i=0; i<arr.size(); i++){

```

```

        if(arr[i].freq == 1fu){
            pRep = i;
            break;
        }
    }

    return make_pair(id, pRep);
}

void LRU(vector<int> ref, int frames){
    vector<frame> fr(frames);
    for(int i=0; i<frames; i++){
        fr[i].frNo = -1;
        fr[i].freq = 0;
    }
    int hit = 0, miss = 0;
    int k=0, it=0;

    while(k<frames && it<ref.size()){
        bool found = false;
        for(int j=k; j>=0; j--){
            if(fr[j].frNo == ref[it]){
                found=true;
                fr[j].freq++;
                hit++;
                break;
            }
        }

        if(!found){
            fr[k].frNo = ref[it];
            fr[k].freq=1;
            miss++;
            k++;
        }
        it++;
    }

    for(int i=it; i<ref.size(); i++){
        pair<int , int> p = Search(fr, ref[i], ref, i);
        if(p.first!=-1){
            fr[p.first].freq++;
            hit++;
        }
        else{
            fr[p.second].frNo=ref[i];
            fr[p.second].freq=1;
            miss++;
        }
    }

    cout<<"Hit Percentage: "<<(100.0*hit)/(1.0*(hit+miss))<<"%\n";
    cout<<"Miss Percentage: "<<(100.0*miss)/(1.0*(hit+miss))<<"%\n";
}

```

```

int main(){
    cout << "LEAST FREQUENTLY USED PAGE REPLACEMENT ALGORITHM C++ IMPLEMENTATION\n";
    cout << "Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int frames = 0;
    cout<<"Enter the number of frames: ";
    cin>>frames;

    if(frames<1){
        cout<<"No frames available!";
        return 0;
    }

    cout<<"Enter the size of the reference string: ";
    int n = 0;
    cin>>n;

    cout<<"Enter the order in which pages are accessed by the CPU\n";
    vector<int> ref(n);
    for(int i=0; i<n; i++){
        cin>>ref[i];
    }

    LRU(ref, frames);

    return 0;
}

```

## SIMULATION OF PAGING TECHNIQUES IN MEMORY MANAGEMENT

CODE:

```

#include <iostream>
using namespace std;
#define MAX 50

int main(){
    cout << "SIMULATION OF PAGING TECHNIQUES C++ IMPLEMENTATION\n";
    cout << "Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int page[MAX], i, no_of_pgs, no_of_frms, pg_sz, off, pno;
    int choice = 0;

    cout << "Enter no of pages in memory: " << endl;
    cin >> no_of_pgs;
    cout << "Enter page size: " << endl;
    cin >> pg_sz;

    cout << "Enter no of frames: " << endl;
    cin >> no_of_frms;

```

```

for (i = 0; i < no_of_frms; i++)
    page[i] = -1;
cout << "\nEnter the page table\n";
cout << "(Enter frame no as -1 if that page is not present in any frame)\n\n"
    << endl;
cout << "\npageno\tframenon-----\t-----\n";

for (i = 0; i < no_of_pgs; i++){
    cout << "\n\n"
        << i << "\t\t";
    cin >> page[i];
}

do{
    cout << "\n\nEnter the logical address(i.e,page no & offset): ";
    cin >> pno >> off;

    if (page[pno] == -1)
        cout << "\n\nThe required page is not available in any of frms";
    else
        cout << "Physical address (i.e, frame no and offset) : " << page[pno] << off <<
endl;
    cout << "\nDo you want to continue(1/0)?:";
    cin >> choice;

}while (choice == 1);

return 0;
}

```

# OUTPUTS:

## LAB 2:

1)

```
PS C:\Users\beadi\Desktop\OS\Assignment 2> cd "c:\Users\beadi\Desktop\OS\Assignment 2\" ; if ($?) { g++ RoundRobin.cpp -o RoundRobin } ; if ($?) { .\RoundRobin.exe }
ROUND ROBIN CPU SCHEDULING ALGORITHM C++ IMPLEMENTATION
Name: Aditya Anand      Roll No.:20124009      Branch: IT

Enter the number of processes: 4
Enter value of time quantum: 2
Enter the arrival times and burst times of 4 processes:
0 5
1 4
2 2
4 1
Process: 0      Arrival Time:0      Burst Time:5      Completion Time:12      Turn Around Time:12      Waiting Time:7      Response Time:0
Process: 1      Arrival Time:1      Burst Time:4      Completion Time:11      Turn Around Time:10      Waiting Time:6      Response Time:1
Process: 2      Arrival Time:2      Burst Time:2      Completion Time:6       Turn Around Time:4       Waiting Time:2      Response Time:2
Process: 3      Arrival Time:4      Burst Time:1      Completion Time:9       Turn Around Time:5       Waiting Time:4      Response Time:4

Average Turn Around Time: 7.75
Average Completion Time: 9.5
```

(2)

```
PS C:\Users\beadi\Desktop\OS LAB\Assignment 3> cd "c:\Users\beadi\Desktop\OS LAB\Assignment 3\" ; if ($?) { g++ FCFS.cpp -o FCFS } ; if ($?) { .\FCFS.exe }
FIRST COME FIRST SERVE CPU SCHEDULING ALGORITHM C++ IMPLEMENTATION
Name: Aditya Anand      Roll No.:20124009      Branch: IT

Enter the number of processes: 4
Enter the arrival times and burst times of 4 processes:
0 2
1 2
5 3
6 4
-----

Process P1: start time = 0 completion time = 2
Process P2: start time = 2 completion time = 4
CPU idle from 4 to 5
Process P3: start time = 5 completion time = 8
Process P4: start time = 8 completion time = 12
-----

Process: 0      Arrival Time:0      Burst Time:2      Completion Time:2      Turn Around Time:2      Waiting Time:0      Response Time:0
Process: 1      Arrival Time:1      Burst Time:2      Completion Time:4      Turn Around Time:3      Waiting Time:1      Response Time:1
Process: 2      Arrival Time:5      Burst Time:3      Completion Time:8      Turn Around Time:3      Waiting Time:0      Response Time:0
Process: 3      Arrival Time:6      Burst Time:4      Completion Time:12     Turn Around Time:6      Waiting Time:2      Response Time:2

Average Turn Around Time: 3.5
Average Completion Time: 6.5
```

## LAB 3:

(1)

### SHORTEST JOB FIRST CPU SCHEDULING ALGORITHM C++ IMPLEMENTATION

Name: Aditya Anand Roll No.:20124009 Branch: IT

```
Enter the number of processes: 4
Enter the arrival times and burst times of 4 processes:
0 4
1 2
2 4
4 1
```

```
-----
-----

Process: 0      Arrival Time:0  Burst Time:4  Completion Time:4  Turn Around Time:4  Waiting Time:0  Response Time:0
Process: 1      Arrival Time:1  Burst Time:2  Completion Time:7  Turn Around Time:6  Waiting Time:4  Response Time:4
Process: 2      Arrival Time:2  Burst Time:4  Completion Time:11 Turn Around Time:9  Waiting Time:5  Response Time:5
Process: 3      Arrival Time:4  Burst Time:1  Completion Time:5  Turn Around Time:1  Waiting Time:0  Response Time:0
```

Average Turn Around Time: 5

Average Completion Time: 6.75

(2)

```
PS C:\Users\beadi\Desktop\OS LAB\Assignment 3> cd "c:\Users\beadi\Desktop\OS LAB\Assignment 3\" ; if ($?) { g++ FCFS.cpp -o FCFS } ;
```

### FIRST COME FIRST SERVE CPU SCHEDULING ALGORITHM C++ IMPLEMENTATION

Name: Aditya Anand Roll No.:20124009 Branch: IT

```
Enter the number of processes: 4
Enter the arrival times and burst times of 4 processes:
0 2
1 2
5 3
6 4
```

```
-----
Process P1: start time = 0 completion time = 2
Process P2: start time = 2 completion time = 4
CPU idle from 4 to 5
Process P3: start time = 5 completion time = 8
Process P4: start time = 8 completion time = 12
```

```
-----

Process: 0      Arrival Time:0  Burst Time:2  Completion Time:2  Turn Around Time:2  Waiting Time:0  Response Time:0
Process: 1      Arrival Time:1  Burst Time:2  Completion Time:4  Turn Around Time:3  Waiting Time:1  Response Time:1
Process: 2      Arrival Time:5  Burst Time:3  Completion Time:8  Turn Around Time:3  Waiting Time:0  Response Time:0
Process: 3      Arrival Time:6  Burst Time:4  Completion Time:12 Turn Around Time:6  Waiting Time:2  Response Time:2
```

Average Turn Around Time: 3.5

Average Completion Time: 6.5



(3)

PRIORITY BASED CPU SCHEDULING ALGORITHM C++ IMPLEMENTATION

Name: Aditya Anand      Roll No.:20124009      Branch: IT

Enter the number of processes: 4

Enter the arrival times and burst times and priority values of 4 processes:

0 4 10

1 2 20

2 4 30

4 1 40

-----

-----

Process: 0	Arrival Time:0	Burst Time:4	Completion Time:4	Turn Around Time:4	Waiting Time:0	Response Time:0
Process: 1	Arrival Time:1	Burst Time:2	Completion Time:11	Turn Around Time:10	Waiting Time:8	Response Time:8
Process: 2	Arrival Time:2	Burst Time:4	Completion Time:9	Turn Around Time:7	Waiting Time:3	Response Time:3
Process: 3	Arrival Time:4	Burst Time:1	Completion Time:5	Turn Around Time:1	Waiting Time:0	Response Time:0

Average Turn Around Time: 5.5

Average Completion Time: 7.25

## LAB 4:

```
PS C:\Users\beadi\Desktop\OS LAB\Assignment 4> cd "c:\Users\beadi\Desktop\OS LAB\Assignment 4\" ;  
PROCESS SYNCHRONISATION USING SEMAPHORE C++ IMPLEMENTATION  
Name: Aditya Anand      Roll No.:20124009      Branch: IT
```

```
Enter number of processes: 10  
Process 1 has entered the Critical Section  
Process 2 has entered the Critical Section  
Process 3 has entered the Critical Section  
Process 4 has entered the Critical Section  
Process 5 has entered the Critical Section  
Process 6 has entered the Critical Section  
Process 7 has entered the Critical Section  
Process 8 has entered the Critical Section  
Process 9 has entered the Critical Section  
Process 10 has been put in the blocked list  
Processes present in Critical Section are: 1 2 3 4 5 6 7 8 9
```

```
Processes present in Blocked List are: 10
```

```
Process 1 has exited the Critical Section  
Process 10 has been removed from the blocked list  
Process 2 has exited the Critical Section  
Process 3 has exited the Critical Section  
Process 4 has exited the Critical Section  
Process 5 has exited the Critical Section  
Process 6 has exited the Critical Section  
Process 7 has exited the Critical Section  
Process 8 has exited the Critical Section  
Process 9 has exited the Critical Section  
Process 10 has entered the Critical Section  
Processes present in Critical Section are: 10
```

```
The Blocked List is empty
```

```
Process 10 has exited the Critical Section
```

## LAB 5:

### SOLUTION OF PRODUCER CONSUMER PROBLEM USING SEMAPHORE AND MUTEX C++ IMPLEMENTATION

Name: Aditya Anand      Roll No.:20124009      Branch: IT

Select the preference: (1, 2, or 3)

1) Producer

2)Consumer

3)Exit

1

Producer produced the item 1

Select the preference: (1, 2, or 3)

1) Producer

2)Consumer

3)Exit

1

Producer produced the item 2

Select the preference: (1, 2, or 3)

1) Producer

2)Consumer

3)Exit

2

Consumer consumed the item 1

Select the preference: (1, 2, or 3)

1) Producer

2)Consumer

3)Exit

2

Consumer consumed the item 2

Select the preference: (1, 2, or 3)

1) Producer

2)Consumer

3)Exit

2

Buffer is empty

## LAB 6:

### In case of Deadlock:

BANKER'S DEADLOCK AVOIDANCE AND PREVENTION ALGORITHM C++ IMPLEMENTATION

Name: Aditya Anand      Roll No.:20124009      Branch: IT

```
Enter number of resources: 3
Enter number of processes: 3
executedFor 3 processes enter:
Resources allocated to process 1: 1 0 2
Max resources required by process 1: 3 1 2
Resources allocated to process 2: 4 3 2
Max resources required by process 2: 5 5 5
Resources allocated to process 3: 1 0 0
Max resources required by process 3: 1 1 0
Enter the currently available resources: 1 1 1
```

-----

Current availability: 1 1 1  
P3 has been allocated the resources

Current availability: 2 1 1  
P1 has been allocated the resources

Current availability: 3 1 3  
Resources could not be allocated to any process.

-----

Deadlock Situation Detected!

### No Deadlock:

BANKER'S DEADLOCK AVOIDANCE AND PREVENTION ALGORITHM C++ IMPLEMENTATION

Name: Aditya Anand      Roll No.:20124009      Branch: IT

```
Enter number of resources: 3
Enter number of processes: 3
executedFor 3 processes enter:
Resources allocated to process 1: 1 1 1
Max resources required by process 1: 2 3 2
Resources allocated to process 2: 1 0 0
Max resources required by process 2: 2 2 2
Resources allocated to process 3: 3 1 4
Max resources required by process 3: 5 1 5
Enter the currently available resources: 3 3 3
```

-----

Current availability: 3 3 3  
P1 has been allocated the resources

Current availability: 4 4 4  
P2 has been allocated the resources

Current availability: 5 4 4  
P3 has been allocated the resources

-----

No Deadlock Detected!  
Safe sequence of resource allocation: P1 P2 P3

## LAB 7:

(1)

```
PS C:\Users\beadi\Desktop\OS LAB\Assignment 7> cd "c:\Users\beadi\Desktop\OS LAB\Assignment 7\" ;  
em }
```

SOLUTION TO READER-WRITER PROBLEM USING SEMAPHORE C++ IMPLEMENTATION

Name: Aditya Anand Roll No.:20124009 Branch: IT

```
Enter  
1 for reader  
2 for writer  
3 to exit  
1  
Reader 1 is reading the database  
Value stored in the database = 9  
  
Enter  
1 for reader  
2 for writer  
3 to exit  
2  
Enter the value you want to write: 5  
Writer is writing in the database  
Value updated!
```

```
Enter  
1 for reader  
2 for writer  
3 to exit  
1  
Reader 1 is reading the database  
Value stored in the database = 5
```

```
Enter  
1 for reader  
2 for writer  
3 to exit  
3
```

(2)

```
PS C:\Users\beadi\Desktop\OS LAB\Assignment 7> cd "c:\Users\beadi\Desktop\OS LAB\Assignment 7\" ;  
rProblem }
```

SOLUTION TO SLEEPING BARBER PROBLEM USING SEMAPHORE C++ IMPLEMENTATION

Name: Aditya Anand Roll No.:20124009 Branch: IT

```
Enter the number of free chairs: 3  
Enter the number of customers entering the shop (enter 0 to exit): 6  
Customer 5 returned back without getting a haircut  
Customer 6 returned back without getting a haircut  
Customer 1 wakes the barber  
Customer 1 is getting a haircut  
  
Customer 2 is getting a haircut  
  
Customer 3 is getting a haircut  
  
Customer 4 is getting a haircut  
  
The barber goes back to sleep  
  
Enter the number of customers entering the shop (enter 0 to exit): 2  
Customer 1 wakes the barber  
Customer 1 is getting a haircut  
  
Customer 2 is getting a haircut  
  
The barber goes back to sleep  
  
Enter the number of customers entering the shop (enter 0 to exit): 4  
Customer 1 wakes the barber  
Customer 1 is getting a haircut  
  
Customer 2 is getting a haircut  
  
Customer 3 is getting a haircut  
  
Customer 4 is getting a haircut  
  
The barber goes back to sleep  
  
Enter the number of customers entering the shop (enter 0 to exit): 0
```

## LAB 8:

```
PS C:\Users\beadi\Desktop\OS LAB> cd "c:\Users\beadi\Desktop\OS LAB\Assignment 8\" ;  
SOLUTION TO DINING PHILOSOPHERS PROBLEM USING SEMAPHORE C++ IMPLEMENTATION  
Name: Aditya Anand      Roll No.:20124009      Branch: IT
```

```
There are 5 philosophers(numbered 1-5) sitting on a round table and 5 forks.  
Enter the order in which Philosophers get hungry: 4 2 3 1 5  
Philosopher 4 is hungry  
Philosopher 4 has started eating  
Philosopher 2 is hungry  
Philosopher 2 has started eating  
Philosopher 3 is hungry  
Forks are not available for Philosopher 3 to use  
Philosopher 1 is hungry  
Forks are not available for Philosopher 1 to use  
Philosopher 5 is hungry  
Forks are not available for Philosopher 5 to use  
Philosopher 4 has finished eating  
Philosopher 2 has finished eating  
Philosopher 3 is hungry  
Philosopher 3 has started eating  
Philosopher 1 is hungry  
Philosopher 1 has started eating  
Philosopher 5 is hungry  
Forks are not available for Philosopher 5 to use  
Philosopher 3 has finished eating  
Philosopher 1 has finished eating  
Philosopher 5 is hungry  
Philosopher 5 has started eating  
Philosopher 5 has finished eating
```

## LAB 9:

(1)

```
PS C:\Users\beadi\Desktop\OS LAB> cd "c:\Users\beadi\Desktop\OS LAB\Assignment 9\"  
FIRST IN FIRST OUT PAGE REPLACEMENT ALGORITHM C++ IMPLEMENTATION  
Name: Aditya Anand      Roll No.:20124009      Branch: IT
```

```
Enter the number of frames: 3  
Enter the size of the reference string: 7  
Enter the order in which pages are accessed by the CPU  
1 4 2 3 5 2 1  
Hit Percentage: 14.2857%  
Miss Percentage: 85.7143%
```

(2)

```

PS C:\Users\beadi\Desktop\OS LAB\Assignment 9> cd "c:\Users\beadi\Desktop\OS LAB\Assignment 9\"
}
LEAST RECENTLY USED REPLACEMENT ALGORITHM C++ IMPLEMENTATION
Name: Aditya Anand      Roll No.:20124009      Branch: IT

Enter the number of frames: 3
Enter the size of the reference string: 7
Enter the order in which pages are accessed by the CPU
1 4 2 3 5 2 1
Hit Percentage: 14.2857%
Miss Percentage: 85.7143%

```

(3)

```

PS C:\Users\beadi\Desktop\OS LAB\Assignment 9> cd "c:\Users\beadi\Desktop\OS LAB\Assignment 9\"
}
LEAST FREQUENTLY USED PAGE REPLACEMENT ALGORITHM C++ IMPLEMENTATION
Name: Aditya Anand      Roll No.:20124009      Branch: IT

Enter the number of frames: 3
Enter the size of the reference string: 7
Enter the order in which pages are accessed by the CPU
1 4 2 3 5 2 1
Hit Percentage: 14.2857%
Miss Percentage: 85.7143%

```

(4)

```

SIMULATION OF PAGING TECHNIQUES C++ IMPLEMENTATION
Name: Aditya Anand      Roll No.:20124009      Branch: IT

Enter no of pages in memory:
5
Enter page size:
10
Enter no of frames:
5

Enter the page table
(Enter frame no as -1 if that page is not present in any frame)

pageno  frameno
-----

0          2

1          1

2          0

3          5

4          4

Enter the logical address(i.e,page no & offset): 3 10
Physical address (i.e, frame no and offset) : 510

Do you want to continue(1/0)? : 0

```