RAMANUJAN COLLEGE, DELHI UNIVERSITY

PRACTICALS

OPERATING SYSTEM

Submitted By: Ravikant Maurya

Submitted To: Mrs.Sheetal Singh

Roll no:20221433

Semester – III

Course= BSc (Hons.)Computer Science

Exam Roll No.:22020570026

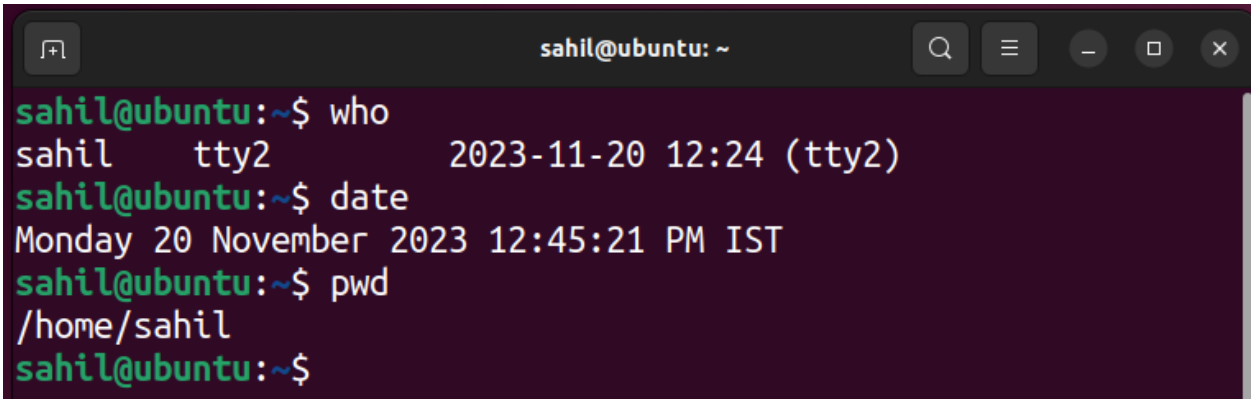
1. Execute various LINUX commands for:

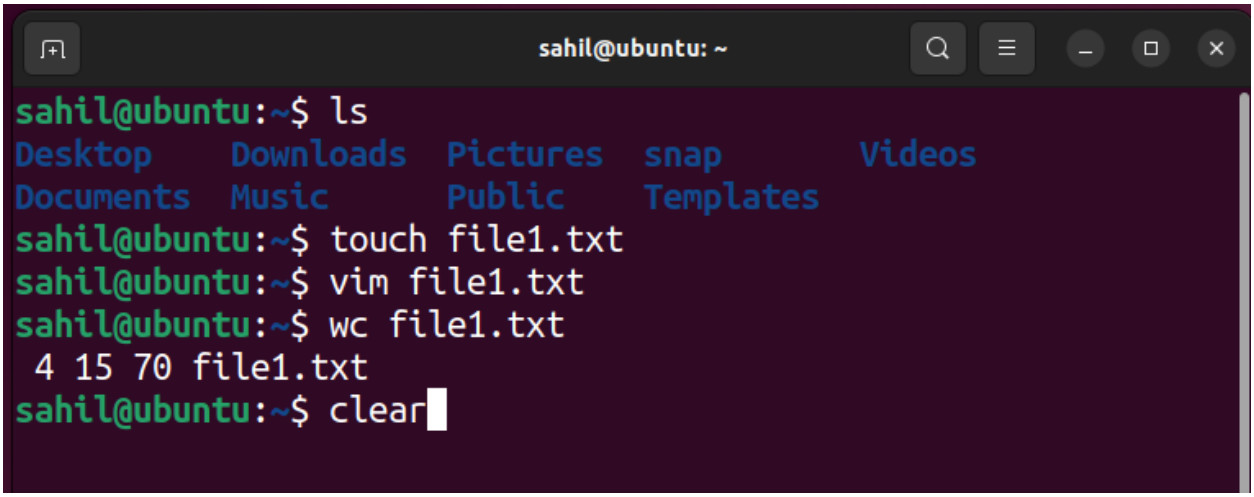
i. Information Maintenance: wc, clear, cal, who, date, pwd

ii. File Management: cat, cp, rm, mv, cmp, comm, diff, find, grep, awk

iii. Directory Management: cd, mkdir, rmdir, ls

OUTPUT:





```
sahil@ubuntu: ~
sahil@ubuntu:~$ cal
   November 2023
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30

sahil@ubuntu:~$ cat file1.txt file2.txt
Hi!
My name is Sahil
It's nice to meet you
The wheather is nice today
Hello!
India lost final against Australia
Australia won its 6th world cup
sahil@ubuntu:~$ cp file1.txt file2.txt
sahil@ubuntu:~$ rm file2.txt
sahil@ubuntu:~$ mv file1.txt mynewdir
sahil@ubuntu:~$ cmp file1.txt file2.txt
cmp: file1.txt: No such file or directory
sahil@ubuntu:~$ cmp file1.txt file2.txt
file1.txt file2.txt differ: byte 1, line 1
sahil@ubuntu:~$
```

```
sahil@ubuntu: ~
sahil@ubuntu:~$ comm file2.txt file1.txt
        My name is Sahil
comm: file 2 is not in sorted order
        I am 18 years old
this is november 20 , 2023
comm: input is not in sorted order
sahil@ubuntu:~$ diff file2.txt file1.txt
1c1,2
< this is november 20 , 2023
---
> My name is Sahil
> I am 18 years old
sahil@ubuntu:~$ find . -name "*Sahil*"
sahil@ubuntu:~$ find . -name "*Sahil*"
./Sahil
```

```
sahil@ubuntu: ~/Sahil
sahil@ubuntu:~$ grep Sahil file2.txt
sahil@ubuntu:~$ grep Sahil file1.txt
My name is Sahil
sahil@ubuntu:~$ awk '/Sahil/' file1.txt
My name is Sahil
sahil@ubuntu:~$ cd Sahil
sahil@ubuntu:~/Sahil$ mkdir mynewdir
sahil@ubuntu:~/Sahil$ ls
mynewdir
sahil@ubuntu:~/Sahil$ rmdir mynewdir/
sahil@ubuntu:~/Sahil$ ls
sahil@ubuntu:~/Sahil$
```

2. Execute various LINUX commands for:

i. Process Control: fork, getpid, ps, kill, sleep

ii. Communication: Input-output redirection, Pipe

iii. Protection Management: chmod, chown, chgrp

Open

fork_example.cpp
~/Sahil

Save

```cpp
#include <iostream>
#include <unistd.h>

int main() {
    pid_t child_pid = fork();

    if (child_pid == 0) {
        // Child process
        std::cout << "Child process" << std::endl;
    } else if (child_pid > 0) {
        // Parent process
        std::cout << "Parent process" << std::endl;
    } else {
        // Fork failed
        perror("fork");
        return 1;
    }

    return 0;
}
```

Open

getpid.cpp
~/Sahil

```cpp
#include <iostream>
#include <unistd.h>

int main() {
    pid_t pid = getpid();
    std::cout << "Process ID: " << pid << std::endl;
    return 0;
}

```

```
sahil@ubuntu:~/Sahil$ g++ -o fork_example fork_example.cpp
sahil@ubuntu:~/Sahil$ ./fork_example
Parent process
Child process
sahil@ubuntu:~/Sahil$ g++ -o getpid getpid.cpp
sahil@ubuntu:~/Sahil$ ./getpid
Process ID: 6285
sahil@ubuntu:~/Sahil$ ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.1  0.2 166632  5212 ?        Ss   12:23   0:03 /sbin/init splash
root           2  0.0  0.0      0     0 ?        S    12:23   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   12:23   0:00 [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<   12:23   0:00 [rcu_par_gp]
root           5  0.0  0.0      0     0 ?        I<   12:23   0:00 [slub_flushwq]
root           6  0.0  0.0      0     0 ?        I<   12:23   0:00 [netns]
root           8  0.0  0.0      0     0 ?        I<   12:23   0:00 [kworker/0:0H-events_highpri]
root          10  0.0  0.0      0     0 ?        I<   12:23   0:00 [mm_percpu_wq]
root          11  0.0  0.0      0     0 ?        I    12:23   0:00 [rcu_tasks_kthread]
root          12  0.0  0.0      0     0 ?        I    12:23   0:00 [rcu_tasks_rude_kthread]
root          13  0.0  0.0      0     0 ?        I    12:23   0:00 [rcu_tasks_trace_kthread]
root          14  0.0  0.0      0     0 ?        S    12:23   0:00 [ksoftirqd/0]
root          15  0.1  0.0      0     0 ?        I    12:23   0:04 [rcu_preempt]
root          16  0.0  0.0      0     0 ?        S    12:23   0:00 [migration/0]
root          17  0.0  0.0      0     0 ?        S    12:23   0:00 [idle_inject/0]
```

```
sahil       5004 21.5 16.5 12005300 331344 ?      Sl   13:10   0:35 /snap/firefox/2987/usr/lib/firefox
root        5131  0.0  0.0      0     0 ?        I    13:10   0:00 [kworker/u8:3-ext4-rsv-conversion]
sahil       5159  0.0  1.2 210628 25088 ?        Sl   13:10   0:00 /snap/firefox/2987/usr/lib/firefox
sahil       5179  1.2  5.0 2462416 100904 ?      Sl   13:10   0:03 /snap/firefox/2987/usr/lib/firefox
sahil       5340  0.3  3.9 2435932 78316 ?       Sl   13:10   0:00 /snap/firefox/2987/usr/lib/firefox
sahil       5651 14.4 17.4 11423116 350124 ?     Sl   13:10   0:35 /snap/firefox/2987/usr/lib/firefox
sahil       5682  0.0  2.7 2399280 54340 ?       Sl   13:10   0:00 /snap/firefox/2987/usr/lib/firefox
sahil       5704  0.0  2.7 2399880 55424 ?       Sl   13:10   0:00 /snap/firefox/2987/usr/lib/firefox
root        5726  0.0  0.0      0     0 ?        I    13:10   0:00 [kworker/u8:4-ext4-rsv-conversion]
sahil       5738  0.0  2.7 2399876 55424 ?       Sl   13:10   0:00 /snap/firefox/2987/usr/lib/firefox
sahil       5772  0.0  1.9 328804 39424 ?        Sl   13:10   0:00 /snap/firefox/2987/usr/lib/firefox
sahil       5776  0.1  2.1 331648 42368 ?        Sl   13:10   0:00 /snap/firefox/2987/usr/lib/firefox
root        5800  0.6  0.0      0     0 ?        R    13:10   0:01 [kworker/u8:5-events_unbound]
root        6048  0.0  0.0      0     0 ?        I    13:11   0:00 [kworker/0:1-events]
sahil       6192  0.1  0.2  11000  4736 pts/1    Ss   13:14   0:00 bash
sahil       6201  0.0  0.1  12668  3328 pts/1    R+   13:14   0:00 ps aux
sahil@ubuntu:~/Sahil$ kill -15 5003
bash: kill: (5003) - Operation not permitted
sahil@ubuntu:~/Sahil$ kill -15 3381
sahil@ubuntu:~/Sahil$ sleep 5
sahil@ubuntu:~/Sahil$
```

```
sahil@ubuntu: ~/Sahil
sahil@ubuntu:~/Sahil$ ls > output.txt
sahil@ubuntu:~/Sahil$ cat < input.txt
who are you?
where do you live?
sahil@ubuntu:~/Sahil$ ls | grep "example"
fork_example
fork_example.cpp
sahil@ubuntu:~/Sahil$ ls | grep "output"
output.txt
sahil@ubuntu:~/Sahil$ ls | grep "output.txt"
output.txt
sahil@ubuntu:~/Sahil$
```

```
sahil@ubuntu: ~/Sahil
sahil@ubuntu:~/Sahil$ chmod +x commands.sh
sahil@ubuntu:~/Sahil$ chown sahil:users any.txt
chown: changing ownership of 'any.txt': Operation not permitt
ed
sahil@ubuntu:~/Sahil$ sudo chown sahil:users any.txt
[sudo] password for sahil:
sahil@ubuntu:~/Sahil$ chgrp users any.txt
sahil@ubuntu:~/Sahil$
```

3. Write a program (using fork () and/or exec () commands) where parent and child execute:

i. same program, same code.

ii. same program, different code.

iii. before terminating, the parent waits for the child to finish its task.

same_different.cpp
~/Sahil

```cpp
#include <iostream>
#include <unistd.h>

int main() {
    pid_t child_pid = fork();

    if (child_pid == 0) {
        // Child process
        std::cout << "Child process executing different code." << std::endl;
        // Replace this with the actual code you want the child to execute.
    } else if (child_pid > 0) {
        // Parent process
        std::cout << "Parent process executing same program with the same code." << std::endl;
    } else {
        // Fork failed
        perror("fork");
        return 1;
    }

    return 0;
}
```

parent_waits_for_child.cpp
~/Sahil

```cpp
#include <iostream>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t child_pid = fork();

    if (child_pid == 0) {
        // Child process
        std::cout << "Child process executing its task." << std::endl;
        // Replace this with the actual code you want the child to execute.
    } else if (child_pid > 0) {
        // Parent process
        std::cout << "Parent process waiting for the child to finish." << std::endl;
        wait(NULL); // Parent waits for the child to finish
        std::cout << "Child process has finished, and parent continues." << std::endl;
    } else {
        // Fork failed
        perror("fork");
        return 1;
    }

    return 0;
}
```

same_same.cpp
~/Sahil

```cpp
#include <iostream>
#include <unistd.h>

int main() {
    pid_t child_pid = fork();

    if (child_pid == 0) {
        // Child process
        std::cout << "Child process executing same program with the same code." << std::endl;
    } else if (child_pid > 0) {
        // Parent process
        std::cout << "Parent process executing same program with the same code." << std::endl;
    } else {
        // Fork failed
        perror("fork");
        return 1;
    }

    return 0;
}
```

OUTPUT:

sahil@ubuntu: ~/Sahil

```
sahil@ubuntu:~/Sahil$ g++ -o same_same same_same.cpp
sahil@ubuntu:~/Sahil$ ./same_same
Parent process executing same program with the same code.
Child process executing same program with the same code.
sahil@ubuntu:~/Sahil$ g++ -o same_different same_different.cpp
sahil@ubuntu:~/Sahil$ ./same_different
Parent process executing same program with the same code.
Child process executing different code.
sahil@ubuntu:~/Sahil$ g++ -o parent_waits_for_child parent_waits
_for_child.cpp
sahil@ubuntu:~/Sahil$ ./parent_waits_for_child
Parent process waiting for the child to finish.
Child process executing its task.
Child process has finished, and parent continues.
sahil@ubuntu:~/Sahil$
```

4. Write a program to report behaviour of Linux kernel including kernel version, CPU type and CPU information.

Practical_4.cpp
~/Documents/OS_Practicals

```cpp
#include <iostream>
#include <fstream>
#include <sstream>

int main() {
  std::cout << "### Kernel Information ###\n\n";

  std::ifstream versionFile("/proc/version_signature");
  if (versionFile.is_open()) {
    std::string line;
    while (std::getline(versionFile, line)) {
      std::istringstream iss(line);
      std::string kernelVersion;
      iss >> kernelVersion;
      std::cout << "Kernel Version: " << kernelVersion << std::endl;
      break;
    }
    versionFile.close();
  } else {
    std::cerr << "Error opening /proc/version_signature" << std::endl;
  }

  std::cout << "\n### CPU Information ###\n\n";

  std::ifstream cpuInfoFile("/proc/cpuinfo");
  if (cpuInfoFile.is_open()) {
    std::string line;
    while (std::getline(cpuInfoFile, line)) {
      if (line.find("processor") != std::string::npos ||
          line.find("model") != std::string::npos) {
        std::cout << line << std::endl;
      }
    }
    cpuInfoFile.close();
  } else {
    std::cerr << "Error opening /proc/cpuinfo" << std::endl;
  }

  return 0;
}
```

C++ Tab Width: 8 Ln 1, Col 20 INS

OUTPUT:

```
ravikant@ubuntu: ~
   34 |                                                cpuInfoFile.clode();
      |                                                            ^~~~~
      |                                                            close
make: *** [<builtin>: practical_4] Error 1
ravikant@ubuntu:~$ vim practical_4.cpp
ravikant@ubuntu:~$ make practical_4
g++     practical_4.cpp   -o practical_4
ravikant@ubuntu:~$ vim practical_4.cpp
ravikant@ubuntu:~$ make parctical_4
make: *** No rule to make target 'parctical_4'.  Stop.
ravikant@ubuntu:~$ ./practical_4
### Kernel Information  ####

Kernel Versions: Ubuntu

### CPU  Information ###

processor       : 0
model           : 78
model name      : Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz
processor       : 1
model           : 78
model name      : Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz
ravikant@ubuntu:~$
```

5. Write a program to report behavior of Linux kernel including information on configured memory, amount of free and used memory. (Memory information)

Practical_5.cpp
~/Documents/OS_Practicals

```cpp
#include <iostream>
#include <fstream>
#include <sstream>

using namespace std;

int main() {
  cout << "### Memory Information ###\n\n";

  // Get total memory
  long totalMem = 0;
  ifstream fTotalMem("/proc/meminfo");
  if (fTotalMem.is_open()) {
    string line;
    while (getline(fTotalMem, line)) {
      if (line.find("MemTotal:") != string::npos) {
        istringstream iss(line);
        string label, memoryValue;
        iss >> label >> memoryValue;
        totalMem = stol(memoryValue);
        break;
      }
    }
    fTotalMem.close();
  } else {
    cerr << "Error opening /proc/meminfo" << endl;
  }

  // Get free memory
  long freeMem = 0;
  ifstream fFreeMem("/proc/meminfo");
  if (fFreeMem.is_open()) {
    string line;
    while (getline(fFreeMem, line)) {
      if (line.find("MemFree:") != string::npos) {
        istringstream iss(line);
        string label, memoryValue;
        iss >> label >> memoryValue;
        freeMem = stol(memoryValue);
        break;
      }
    }
    fFreeMem.close();
  } else {
    cerr << "Error opening /proc/meminfo" << endl;
  }

```

C++ Tab Width: 8 Ln 1, Col 1 INS

```cpp
  // Calculate used memory
  long usedMem = totalMem - freeMem;

  cout << "Total Memory: " << totalMem / 1024 << " MB" << endl;
  cout << "Free Memory: " << freeMem / 1024 << " MB" << endl;
  cout << "Used Memory: " << usedMem / 1024 << " MB" << endl;

  return 0;
}
```

OUTPUT:

```
ravikant@ubuntu: ~/Documents/os_practical
ravikant@ubuntu:~/Documents/os_practical$ g++ -o Practical_5 Practical_5.cpp
ravikant@ubuntu:~/Documents/os_practical$ ./Practical_5
### Memory Information ###

Total Memory: 1959 MB
Free Memory: 165 MB
Used Memory: 1794 MB
ravikant@ubuntu:~/Documents/os_practical$
```

6. Write a program to copy files using system calls.

   OUTPUT:

Practical_6.cpp
~/Documents/OS_Practicals

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <fcntl.h>
#include <sys/stat.h>
#include <unistd.h>

#define BUFFER_SIZE 1024

using namespace std;

void copy(const string& src, const string& dst) {
  int sfd, dfd, count;
  char buffer[BUFFER_SIZE];
  mode_t fileMode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;

  sfd = open(src.c_str(), O_RDONLY);
  if (sfd < 0) {
    cerr << "ERROR: " << src << " doesn't exist" << endl;
    exit(1);
  }

  dfd = open(dst.c_str(), O_CREAT | O_WRONLY | O_TRUNC, fileMode);
  if (dfd < 0) {
    close(sfd);
    cerr << "ERROR: Failed to create or open destination file " << dst << endl;
    exit(1);
  }

  while ((count = read(sfd, buffer, BUFFER_SIZE)) > 0) {
    if (write(dfd, buffer, count) < 0) {
      close(sfd);
      close(dfd);
      cerr << "ERROR: Failed to write to destination file " << dst << endl;
      exit(1);
    }
  }

  close(sfd);
  close(dfd);
}

int main(int argc, char* argv[]) {
  if (argc != 3) {
    cerr << "ERROR: Usage: " << argv[0] << " <source_file> <destination_file>" << endl;
    return 1;
  }

  cout << "Copying " << argv[1] << " to " << argv[2] << endl;
  copy(argv[1], argv[2]);

  return 0;
}
```

C++ Tab Width: 8 Ln 1, Col 1 INS

OUTPUT:

```
ities    Terminal
ravikant@ubuntu: ~/Documents/os_practical
ravikant@ubuntu:~/Documents/os_practical$ g++ -o Practical_6 Practical_6.cpp
ravikant@ubuntu:~/Documents/os_practical$ ./Practical_6 source.txt destination.t
xt
Copying source.txt to destination.txt
```

7. Write a program to im plement FCFS scheduling algorithm.

Open Practical_7.cpp
~/Documents/OS_Practicals Save

```cpp
// C++ program for implementation of FCFS
// scheduling
#include<iostream>
using namespace std;

// Function to find the waiting time for all
// processes
void findWaitingTime(int processes[], int n,
                                                int bt[], int wt[])
{
        // waiting time for first process is 0
        wt[0] = 0;

        // calculating waiting time
        for (int i = 1; i < n ; i++ )
                wt[i] = bt[i-1] + wt[i-1] ;
}

// Function to calculate turn around time
void findTurnAroundTime( int processes[], int n,
                                int bt[], int wt[], int tat[])
{
        // calculating turnaround time by adding
        // bt[i] + wt[i]
        for (int i = 0; i < n ; i++)
                tat[i] = bt[i] + wt[i];
}

//Function to calculate average time
void findavgTime( int processes[], int n, int bt[])
{
        int wt[n], tat[n], total_wt = 0, total_tat = 0;

        //Function to find waiting time of all processes
        findWaitingTime(processes, n, bt, wt);

        //Function to find turn around time for all processes
        findTurnAroundTime(processes, n, bt, wt, tat);

        //Display processes along with all details
        cout << "Processes "<< " Burst time "
                << " Waiting time " << " Turn around time\n";

        // Calculate total waiting time and total turn
        // around time
        for (int i=0; i<n; i++)
        {
                total_wt = total_wt + wt[i];
                total_tat = total_tat + tat[i];
                cout << " " << i+1 << "\t\t" << bt[i] <<"\t "
                        << wt[i] <<"\t\t " << tat[i] <<endl;
        }

        cout << "Average waiting time = "
```

```cpp
        cout << "Average waiting time = "
                << (float)total_wt / (float)n;
        cout << "\nAverage turn around time = "
                << (float)total_tat / (float)n;
}

// Driver code
int main()
{
        //process id's
        int processes[] = { 1, 2, 3};
        int n = sizeof processes / sizeof processes[0];

        //Burst time of all processes
        int burst_time[] = {10, 5, 8};

        findavgTime(processes, n, burst_time);
        return 0;
}
```

C++ Tab Width: 8 Ln 18, Col 1 INS

OUTPUT:

ities Terminal

ravikant@ubuntu: ~/Documents/os_practical

```
ravikant@ubuntu:~/Documents/os_practical$ g++ -o Practical_7 Practical_7.cpp
ravikant@ubuntu:~/Documents/os_practical$ ./Practical_7
Processes  Burst time  Waiting time  Turn around time
 1              10        0               10
 2              5         10              15
 3              8         15              23
Average waiting time = 8.33333
Average turn around time = 16ravikant@ubuntu:~/Documents/os_practical$
```

8. Write a program to implement SJF scheduling algorithm.

Practical_8.cpp
~/Documents/OS_Practicals

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

int main() {

        // Matrix for storing Process Id, Burst
        // Time, Average Waiting Time & Average
        // Turn Around Time.
        int A[100][4];
        int i, j, n, total = 0, index, temp;
        float avg_wt, avg_tat;

        cout << "Enter number of process: ";
        cin >> n;

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

        // User Input Burst Time and alloting Process Id.
        for (i = 0; i < n; i++) {
                cout << "P" << i + 1 << ": ";
                cin >> A[i][1];
                A[i][0] = i + 1;
        }

        // Sorting process according to their Burst Time.
        for (i = 0; i < n; i++) {
                index = i;
                for (j = i + 1; j < n; j++)
                        if (A[j][1] < A[index][1])
                                index = j;
                temp = A[i][1];
                A[i][1] = A[index][1];
                A[index][1] = temp;

                temp = A[i][0];
                A[i][0] = A[index][0];
                A[index][0] = temp;
        }

        A[0][2] = 0;
        // Calculation of Waiting Times
        for (i = 1; i < n; i++) {
                A[i][2] = 0;
                for (j = 0; j < i; j++)
                        A[i][2] += A[j][1];
                total += A[i][2];
        }

        avg_wt = (float)total / n;
        total = 0;
        cout << "P       BT      WT      TAT" << endl;

        // Calculation of Turn Around Time and printing the
        // data.
```

C++ Tab Width: 8 Ln 1, Col 1 INS

```cpp
        for (i = 0; i < n; i++) {
                A[i][3] = A[i][1] + A[i][2];
                total += A[i][3];
                cout << "P" << A[i][0] << "      " << A[i][1] << "       " << A[i][2] << "       " << A[i][3] << endl;
        }

        avg_tat = (float)total / n;
        cout << "Average Waiting Time= " << avg_wt << endl;
        cout << "Average Turnaround Time= " << avg_tat << endl;
}
```

C++ Tab Width: 8 Ln 1, Col 1 INS

OUTPUT:

```
ities    Terminal
ravikant@ubuntu: ~/Documents/os_practical
ravikant@ubuntu:~/Documents/os_practical$ g++ -o Practical_8 Practical_8.cpp
ravikant@ubuntu:~/Documents/os_practical$ ./Practical_8
Enter number of process: 8
Enter Burst Time:
P1: 4
P2: 3
P3: 2
P4: 5
P5: 6
P6: 8
P7: 4
P8: 2
P        BT      WT      TAT
P3       2       0       2
P8       2       2       4
P2       3       4       7
P7       4       7       11
P1       4       11      15
P4       5       15      20
P5       6       20      26
P6       8       26      34
Average Waiting Time= 10.625
Average Turnaround Time= 14.875
ravikant@ubuntu:~/Documents/os_practical$
```

9. Write a program to implement non-preemptive priority-based scheduling algorithm.

Practical_9.cpp
~/Documents/OS_Practicals

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Process {
  int processId;
  int arrivalTime;
  int burstTime;
  int priority;
  int waitingTime = 0;
  int turnAroundTime = 0;
};

bool compareProcesses(const Process& p1, const Process& p2) {
  if (p1.priority < p2.priority) {
    return true;
  } else if (p1.priority == p2.priority) {
    return p1.arrivalTime < p2.arrivalTime;
  } else {
    return false;
  }
}

void calculateWaitingTime(vector<Process>& processes) {
  for (int i = 1; i < processes.size(); i++) {
    int previousCompletionTime = processes[i - 1].arrivalTime + processes[i - 1].burstTime;
    processes[i].waitingTime = previousCompletionTime - processes[i].arrivalTime;
  }
}

void calculateTurnAroundTime(vector<Process>& processes) {
  for (Process& process : processes) {
    process.turnAroundTime = process.waitingTime + process.burstTime;
  }
}

void printProcessDetails(vector<Process>& processes) {
  cout << "\nProcess ID\tArrival Time\tBurst Time\tPriority\tWaiting Time\tTurn Around Time\n";
  for (Process& process : processes) {
    cout << process.processId << "\t\t" << process.arrivalTime << "\t\t" << process.burstTime << "\t\t" << process.priority << "\t\t" << process.waitingTime << "\t\t" << process.turnAroundTime << endl;
  }
}

int main() {
  int numberOfProcesses;
  cout << "Enter the number of processes: ";
  cin >> numberOfProcesses;

  vector<Process> processes(numberOfProcesses);

```

C++ Tab Width: 8 Ln 1, Col 1 INS

```cpp
  for (int i = 0; i < numberOfProcesses; i++) {
    cout << "Enter process " << i + 1 << " details:" << endl;
    cout << "Process ID: ";
    cin >> processes[i].processId;
    cout << "Arrival Time: ";
    cin >> processes[i].arrivalTime;
    cout << "Burst Time: ";
    cin >> processes[i].burstTime;
    cout << "Priority: ";
    cin >> processes[i].priority;
  }

  sort(processes.begin(), processes.end(), compareProcesses);

  calculateWaitingTime(processes);
  calculateTurnAroundTime(processes);

  cout << "\nProcess Details:\n";
  printProcessDetails(processes);

  return 0;
}

```

C++ Tab Width: 8 Ln 1, Col 1 INS

OUTPUT:

```
ities  Terminal                                                    Nov 23  11:59
                                                   ravikant@ubuntu: ~/Documents/os_practical
ravikant@ubuntu:~/Documents/os_practical$ g++ -o Practical_9 Practical_9.cpp
ravikant@ubuntu:~/Documents/os_practical$ ./Practical_9
Enter the number of processes: 5
Enter process 1 details:
Process ID: 2345
Arrival Time: 3
Burst Time: 6
Priority: 5
Enter process 2 details:
Process ID: 2
Arrival Time: 45
Burst Time: 3
Priority: 3
Enter process 3 details:
Process ID: 2345
Arrival Time: 5
Burst Time: 67
Priority: 2
Enter process 4 details:
Process ID: 345
Arrival Time: 1
Burst Time: 66
Priority: 4
Enter process 5 details:
Process ID: 321
Arrival Time: 32
Burst Time: 45
Priority: 1

Process Details:

Process ID      Arrival Time    Burst Time      Priority        Waiting Time    Turn Around Time
321             32              45              1               0               45
2345            5               67              2               72              139
2               45              3               3               27              30
345             1               66              4               47              113
2345            3               6               5               64              70
ravikant@ubuntu:~/Documents/os_practical$
```

10. Write a program to implement SRTF scheduling algorithm.

Practical_10.cpp
~/Documents/OS_Practicals

```cpp
// C++ program to implement Shortest Remaining Time First
// Shortest Remaining Time First (SRTF)

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

struct Process {
        int pid; // Process ID
        int bt; // Burst Time
        int art; // Arrival Time
};

// Function to find the waiting time for all
// processes
void findWaitingTime(Process proc[], int n,
                                                        int wt[])
{
        int rt[n];

        // Copy the burst time into rt[]
        for (int i = 0; i < n; i++)
                rt[i] = proc[i].bt;

        int complete = 0, t = 0, minm = INT_MAX;
        int shortest = 0, finish_time;
        bool check = false;

        // Process until all processes gets
        // completed
        while (complete != n) {

                // Find process with minimum
                // remaining time among the
                // processes that arrives till the
                // current time`
                for (int j = 0; j < n; j++) {
                        if ((proc[j].art <= t) &&
                        (rt[j] < minm) && rt[j] > 0) {
                                minm = rt[j];
                                shortest = j;
                                check = true;
                        }
                }

                if (check == false) {
                        t++;
                        continue;
                }

                // Reduce remaining time by one
                rt[shortest]--;

                // Update minimum
                minm = rt[shortest];
```

C++ Tab Width: 8 Ln 2, Col 1 INS

Practical_10.cpp
~/Documents/OS_Practicals

```cpp
                if (minm == 0)
                        minm = INT_MAX;

                // If a process gets completely
                // executed
                if (rt[shortest] == 0) {

                        // Increment complete
                        complete++;
                        check = false;

                        // Find finish time of current
                        // process
                        finish_time = t + 1;

                        // Calculate waiting time
                        wt[shortest] = finish_time -
                                                proc[shortest].bt -
                                                proc[shortest].art;

                        if (wt[shortest] < 0)
                                wt[shortest] = 0;
                }
                // Increment time
                t++;
        }
}

// Function to calculate turn around time
void findTurnAroundTime(Process proc[], int n,
                                                int wt[], int tat[])
{
        // calculating turnaround time by adding
        // bt[i] + wt[i]
        for (int i = 0; i < n; i++)
                tat[i] = proc[i].bt + wt[i];
}

// Function to calculate average time
void findavgTime(Process proc[], int n)
{
        int wt[n], tat[n], total_wt = 0,
                                        total_tat = 0;

        // Function to find waiting time of all
        // processes
        findWaitingTime(proc, n, wt);

        // Function to find turn around time for
        // all processes
        findTurnAroundTime(proc, n, wt, tat);

        // Display processes along with all
        // details
```

C++ Tab Width: 8 Ln 2, Col 1 INS

```cpp
        cout << " P\t\t"
                << "BT\t\t"
                << "WT\t\t"
                << "TAT\t\t\n";

        // Calculate total waiting time and
        // total turnaround time
        for (int i = 0; i < n; i++) {
                total_wt = total_wt + wt[i];
                total_tat = total_tat + tat[i];
                cout << " " << proc[i].pid << "\t\t"
                        << proc[i].bt << "\t\t " << wt[i]
                        << "\t\t " << tat[i] << endl;
        }

        cout << "\nAverage waiting time = "
                << (float)total_wt / (float)n;
        cout << "\nAverage turn around time = "
                << (float)total_tat / (float)n;
}

// Driver code
int main()
{
        Process proc[] = { { 1, 6, 2 }, { 2, 2, 5 },
                                        { 3, 8, 1 }, { 4, 3, 0}, {5, 4, 4} };
        int n = sizeof(proc) / sizeof(proc[0]);

        findavgTime(proc, n);
        return 0;
}
```

C++ Tab Width: 8 Ln 2, Col 1 INS

OUTPUT:

```
ravikant@ubuntu: ~/Documents/os_practical

ravikant@ubuntu:~/Documents/os_practical$ g++ -o Practical_10 Practical_10.cpp
ravikant@ubuntu:~/Documents/os_practical$ ./Practical_10
 P             BT             WT             TAT
 1             6              7              13
 2             2              0              2
 3             8              14             22
 4             3              0              3
 5             4              2              6

Average waiting time = 4.6
Average turn around time = 9.2ravikant@ubuntu:~/Documents/os_practical$
```

11. Write a program to calculate sum of n numbers using Pthreads. A list of n numbers is divided into two smaller lists of equal size, two separate threads are used to sum the sub lists.

Practical_11.cpp
~/Documents/OS_Practicals

```cpp
#include <iostream>
#include <pthread.h>

using namespace std;

pthread_mutex_t mutex;
int sum1 = 0, sum2 = 0;

void* sumThread1(void* arg) {
  int* arr1 = (int*)arg;
  int n = sizeof(arr1) / sizeof(int);
  for (int i = 0; i < n; i++) {
    sum1 += arr1[i];
  }
  pthread_mutex_lock(&mutex);
  cout << "Sum of first half: " << sum1 << endl;
  pthread_mutex_unlock(&mutex);
  pthread_exit(NULL);
}

void* sumThread2(void* arg) {
  int* arr2 = (int*)arg;
  int n = sizeof(arr2) / sizeof(int);
  for (int i = 0; i < n; i++) {
    sum2 += arr2[i];
  }
  pthread_mutex_lock(&mutex);
  cout << "Sum of second half: " << sum2 << endl;
  pthread_mutex_unlock(&mutex);
  pthread_exit(NULL);
}

int main() {
  int n;
  cout << "Enter the number of elements: ";
  cin >> n;

  int arr[n];
  cout << "Enter the elements: ";
  for (int i = 0; i < n; i++) {
    cin >> arr[i];
  }

  pthread_t thread1, thread2;

  int half = n / 2;
  int* arr1 = arr;
  int* arr2 = arr + half;

  pthread_mutex_init(&mutex, NULL);
  pthread_create(&thread1, NULL, sumThread1, (void*)arr1);
  pthread_create(&thread2, NULL, sumThread2, (void*)arr2);

  pthread_join(thread1, NULL);
```

C++ Tab Width: 8 Ln 1, Col 1 INS

```
pthread_join(thread2, NULL);

int totalSum = sum1 + sum2;
cout << "Total sum: " << totalSum << endl;

pthread_mutex_destroy(&mutex);

return 0;
}
```

C++ Tab Width: 8 Ln 1, Col 1 INS

OUTPUT:

```
ravikant@ubuntu: ~/Documents/os_practical
ravikant@ubuntu:~/Documents/os_practical$ g++ -o Practical_11 Practical_11.cpp
ravikant@ubuntu:~/Documents/os_practical$ ./Practical_11
Enter the number of elements: 6
Enter the elements: 34 56 78 90 345 678
Sum of second half: 435
Sum of first half: 90
Total sum: 525
ravikant@ubuntu:~/Documents/os_practical$
```

12. Write a program to implement first-fit, best-fit and worst-fit allocation strategies

Open Practical_12.cpp ~/Documents/OS_Practicals Save

```cpp
#include <iostream>
#include <vector>
#include <climits>

using namespace std;

struct Block {
  int size;
  bool isFree;
};

struct Process {
  int processId;
  int size;
};

void firstFit(vector<Block>& blocks, vector<Process>& processes) {
  for (Process& process : processes) {
    bool allocated = false;
    for (int i = 0; i < blocks.size(); i++) {
      if (blocks[i].isFree && blocks[i].size >= process.size) {
        blocks[i].isFree = false;
        process.processId = i;
        allocated = true;
        break;
      }
    }

    if (!allocated) {
      cout << "Process " << process.processId << " cannot be allocated using first-fit" << endl;
    }
  }
}

void bestFit(vector<Block>& blocks, vector<Process>& processes) {
  for (Process& process : processes) {
    int bestFitIndex = -1;
    int bestFitSize = INT_MAX;

    for (int i = 0; i < blocks.size(); i++) {
      if (blocks[i].isFree && blocks[i].size >= process.size) {
        if (blocks[i].size < bestFitSize) {
          bestFitIndex = i;
          bestFitSize = blocks[i].size;
        }
      }
    }

    if (bestFitIndex != -1) {
      blocks[bestFitIndex].isFree = false;
      process.processId = bestFitIndex;
    } else {
      cout << "Process " << process.processId << " cannot be allocated using best-fit" << endl;
    }
```

C++ Tab Width: 8 Ln 1, Col 1 INS

Practical_12.cpp
~/Documents/OS_Practicals

```cpp
  }
}

void worstFit(vector<Block>& blocks, vector<Process>& processes) {
  for (Process& process : processes) {
    int worstFitIndex = -1;
    int worstFitSize = 0;

    for (int i = 0; i < blocks.size(); i++) {
      if (blocks[i].isFree && blocks[i].size >= process.size) {
        if (blocks[i].size > worstFitSize) {
          worstFitIndex = i;
          worstFitSize = blocks[i].size;
        }
      }
    }

    if (worstFitIndex != -1) {
      blocks[worstFitIndex].isFree = false;
      process.processId = worstFitIndex;
    } else {
      cout << "Process " << process.processId << " cannot be allocated using worst-fit" << endl;
    }
  }
}

int main() {
  int numberOfBlocks;
  cout << "Enter the number of memory blocks: ";
  cin >> numberOfBlocks;

  vector<Block> blocks(numberOfBlocks);
  for (int i = 0; i < numberOfBlocks; i++) {
    cout << "Enter block " << i + 1 << " size: ";
    cin >> blocks[i].size;
    blocks[i].isFree = true;
  }

  int numberOfProcesses;
  cout << "\nEnter the number of processes: ";
  cin >> numberOfProcesses;

  vector<Process> processes(numberOfProcesses);
  for (int i = 0; i < numberOfProcesses; i++) {
    cout << "Enter process " << i + 1 << " size: ";
    cin >> processes[i].size;
  }

  cout << "\nFirst-Fit Allocation:" << endl;
  firstFit(blocks, processes);

  cout << "\nBest-Fit Allocation:" << endl;
  bestFit(blocks, processes);

```

C++ Tab Width: 8 Ln 1, Col 1 INS

```
  cout << "\nWorst-Fit Allocation:" << endl;
  worstFit(blocks, processes);
}
```

C++ Tab Width: 8 Ln 1, Col 1 INS

OUTPUT:

ravikant@ubuntu: ~/Documents/os_practical

```
ravikant@ubuntu:~/Documents/os_practical$ g++ -o Practical_12 Practical_12.cpp
ravikant@ubuntu:~/Documents/os_practical$ ./Practical_12
Enter the number of memory blocks: 4
Enter block 1 size: 45
Enter block 2 size: 67
Enter block 3 size: 78
Enter block 4 size: 98

Enter the number of processes: 5
Enter process 1 size: 23
Enter process 2 size: 45
Enter process 3 size: 67
Enter process 4 size: 86
Enter process 5 size: 43

First-Fit Allocation:
Process 0 cannot be allocated using first-fit

Best-Fit Allocation:
Process 0 cannot be allocated using best-fit
Process 1 cannot be allocated using best-fit
Process 2 cannot be allocated using best-fit
Process 3 cannot be allocated using best-fit
Process 0 cannot be allocated using best-fit

Worst-Fit Allocation:
Process 0 cannot be allocated using worst-fit
Process 1 cannot be allocated using worst-fit
Process 2 cannot be allocated using worst-fit
Process 3 cannot be allocated using worst-fit
Process 0 cannot be allocated using worst-fit
ravikant@ubuntu:~/Documents/os_practical$
```