PRACTICAL FILE

RAMANUJAN COLLEGE

UNIVERSITY OF DELHI

DSC: OPERATING SYSTEMS

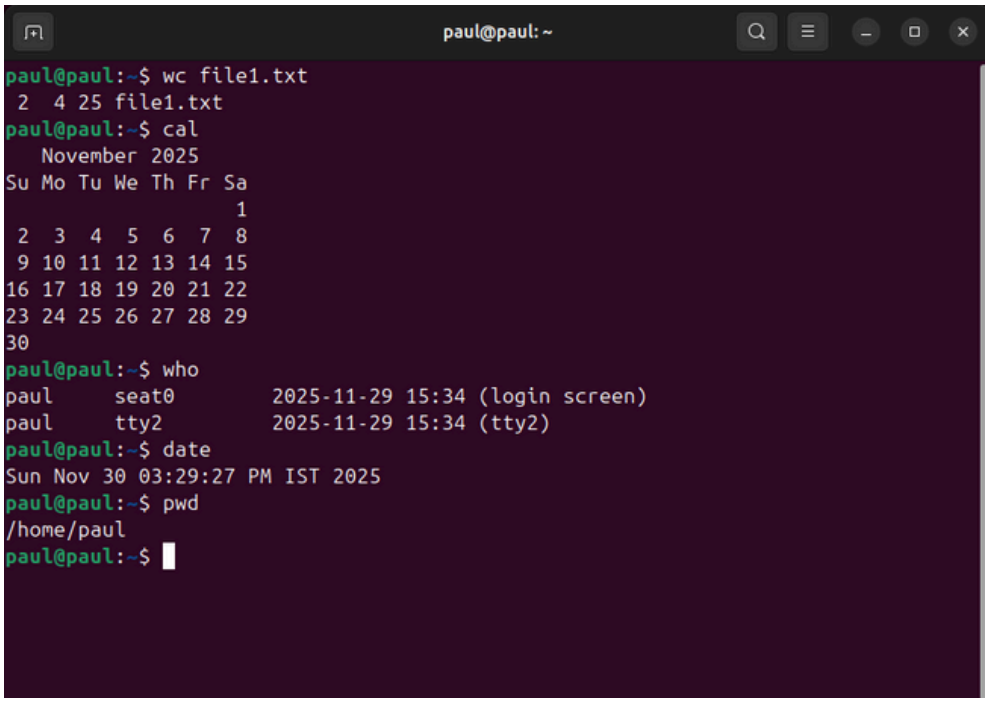
2025-26

SEMESTER 3

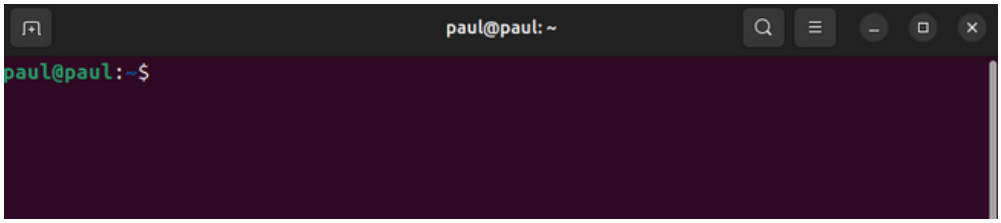
	SUBMITTED TO
SUBMITTED BY NAME: GURPREET SINGH PAUL ROLL NO. 24570022(24020570023) COURSE: B.Sc(H)Computer Science SEMESTER:3	Mrs.Sheetal Singh

Q-1 :- Execute various LINUX commands for:

(i) information Maintenance: wc, clear, cal, who, date, pwd.



Clear



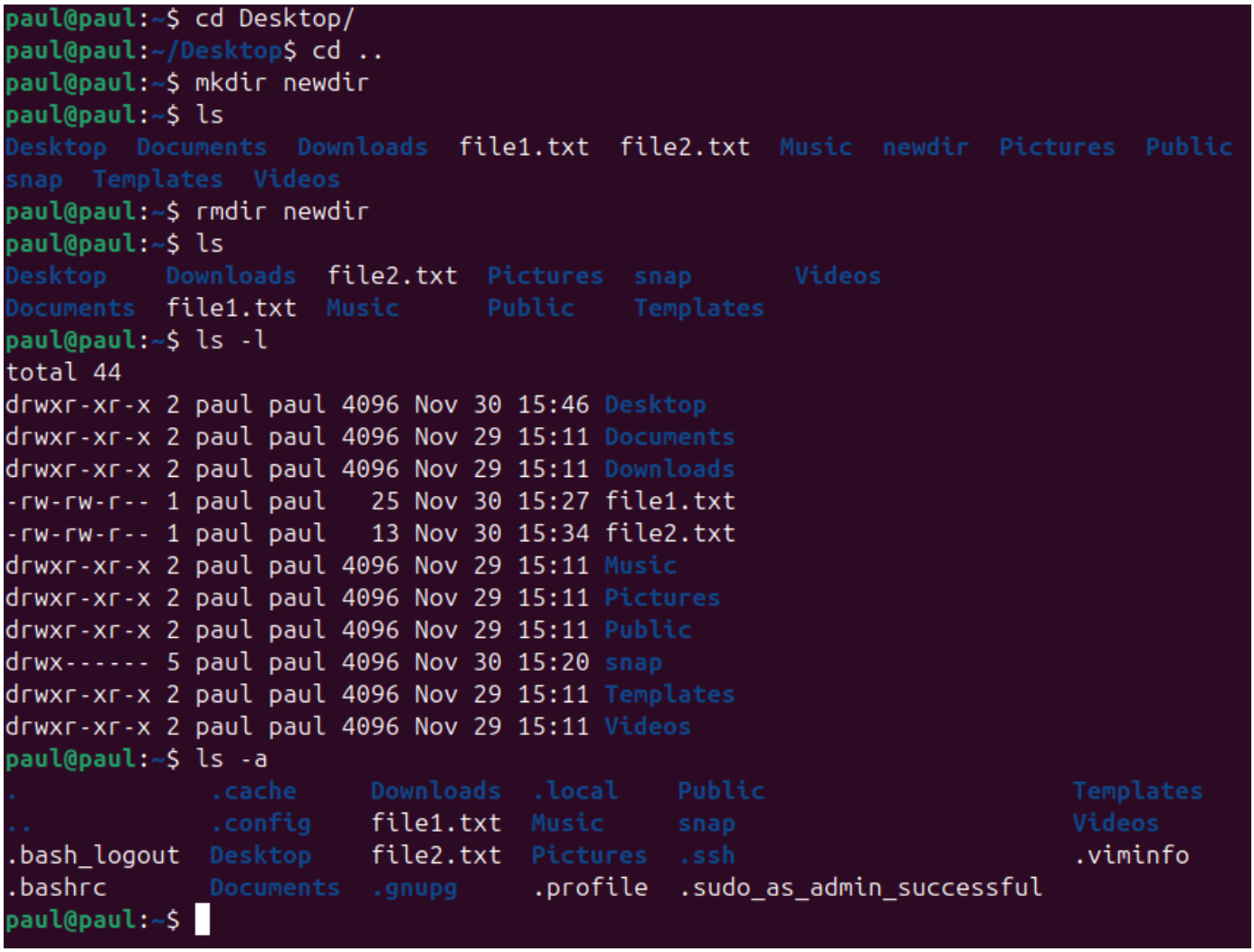
(ii) File Management: cat, cp, rm, mv, cmp, comm, diff, find, grep, awk.

```
paul@paul:~$ cat file1.txt
hello world
os practical
paul@paul:~$ cat > file2.txt
its 2nd file
paul@paul:~$ cat file2.txt
its 2nd file
paul@paul:~$ cp file1.txt copy.txt
paul@paul:~$ cat copy.txt
hello world
os practical
paul@paul:~$ rm copy.txt
paul@paul:~$ mv file1.txt Desktop/
paul@paul:~$ cd Desktop/
paul@paul:~/Desktop$ ls
file1.txt
paul@paul:~/Desktop$ mv file1.txt /home/paul/
paul@paul:~/Desktop$ cd
paul@paul:~$ cmp file1.txt file2.txt
file1.txt file2.txt differ: byte 1, line 1
paul@paul:~$ comm file1.txt file2.txt
hello world
        its 2nd file
os practical
paul@paul:~$ diff file1.txt file2.txt
1,2c1
< hello world
< os practical
---
> its 2nd file
```

```
paul@paul:~$ find . -name "file1.txt"
./file1.txt
paul@paul:~$ grep "world" file1.txt
hello world
paul@paul:~$ awk '{print $1}' file1.txt
hello
os
paul@paul:~$
```

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

Cd (change directory)



Q– 2 - Execute various LINUX commands for

i. Communication: Input-output redirection, Pipe

```
paul@paul:~$ sort < file1.txt
hello world
os practical
paul@paul:~$ ls > out.txt
paul@paul:~$ date >> log.txt
paul@paul:~$ cat out.txt
a.out
Desktop
Documents
Downloads
file1.txt
file2.txt
log.txt
Music
out.txt
Pictures
program
program.c
Public
snap
Templates
Videos
paul@paul:~$ cat log.txt
Sun Nov 30 04:15:48 PM IST 2025
Sun Nov 30 04:16:55 PM IST 2025
paul@paul:~$
```

```
paul@paul:~$ ls | grep "c"
Documents
Music
Pictures
program.c
Public
paul@paul:~$
```

ii. Protection Management; chmod,chown, chgrp

```
paul@paul:~$ chmod 755 out.txt
paul@paul:~$ sudo chown paul out.txt
[sudo] password for paul:
paul@paul:~$ sudo groupadd developers
paul@paul:~$ sudo chgrp developers out.txt
paul@paul:~$ sudo chown gg out.txt
paul@paul:~$
```

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

```cpp
#include <sys/types.h>
#include <unistd.h>   // fork, getpid, getppid
#include <cstdlib>    // system()

using namespace std;

int main() {
    cout << "I am the single parent. About to fork...\n";

    pid_t pid = fork();   // create child

    if (pid < 0) {
        perror("fork failed");
        return 1;
    }

    if (pid == 0) {
        // ----- CHILD -----
        cout << "Child: Hello from the C++ CHILD process!  (PID: "
             << getpid() << ", Parent PID: " << getppid() << ")\n";
        // optional: show processes
        // system("ps -o pid,ppid,cmd | head");
    } else {
        // ----- PARENT -----
        cout << "Parent: Hello from the C++ PARENT process! (Child's PID: "
             << pid << ", My PID: " << getpid() << ")\n";
        // optional: give child time to print
        // sleep(1);
    }

    return 0;
```

```
paul@paul:~$ g++ fork.cpp -o fork
paul@paul:~$ ./fork
I am the single parent. About to fork...
Parent: Hello from the C++ PARENT process! (Child's PID: 6474, My PID: 6473)
Child: Hello from the C++ CHILD process!  (PID: 6474, Parent PID: 6473)
paul@paul:~$ sleep 1000 &
[1] 6480
paul@paul:~$ sleep 1
paul@paul:~$ ps
    PID TTY          TIME CMD
   3239 pts/0    00:00:00 bash
   6480 pts/0    00:00:00 sleep
   6482 pts/0    00:00:00 ps
paul@paul:~$ kill 6480
paul@paul:~$ ps
    PID TTY          TIME CMD
   3239 pts/0    00:00:00 bash
   6495 pts/0    00:00:00 ps
[1]+  Terminated              sleep 1000
paul@paul:~$
```

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

i. same program, same code.

```
paul@paul:~$ vi sameproco.cpp
paul@paul:~$ cat sameproco.cpp
#include <iostream>
#include <unistd.h>
using namespace std;

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

    if(pid < 0) {
        cout << " Fork failed " << endl;
    }
    else {
        cout << "Process execution,PID: " << getpid() << endl;
    }

    return 0;
}

paul@paul:~$ g++ sameproco.cpp -o sameproco
paul@paul:~$ ./sameproco
Process execution,PID: 6638
Process execution,PID: 6639
paul@paul:~$
```

iii. same program, different code.

```
paul@paul:~$ cat sameprodiffco.cpp
#include<iostream>
#include<unistd.h>
using namespace std;

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

    if(pid < 0) {
        cout << " Fork failed" << endl;
    }
    else if(pid == 0){
        cout << "Parent process executing" << endl;
        cout << "Child PID:" << getpid() << endl;
    }
    else {
        cout << "Parent Process is executing" << endl;
        cout << "Parent PID:" << getpid() << endl;
    }

    return 0;
}

paul@paul:~$ g++ sameprodiffco.cpp -o sameprodiffco
paul@paul:~$ ./sameprodiffco
Parent Process is executing
Parent PID:6759
Parent process executing
Child PID:6760
paul@paul:~$
```

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

```cpp
#include <iostream>
#include <unistd.h>     // fork, sleep, getpid
#include <sys/types.h>  // pid_t
#include <sys/wait.h>   // wait

using namespace std;

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

    if(pid < 0) {
        cout << " Fork failed " << endl;
    }
    else if(pid == 0) {
        cout << " Child running, PID: " << getpid() << endl;
        sleep(2);  // simulate some work
        cout << " Child finished\n";
    }
    else {
        wait(NULL);  // parent waits for child to finish
        cout << " Parent waited for child. Parent PID: " << getpid() << endl;
    }

    return 0;
}
```

```
paul@paul:~$ g++ parentwait.cpp -o parentwait
paul@paul:~$ ./parentwait
 Child running, PID: 6852
 Child finished
 Parent waited for child. Parent PID: 6851
```

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

```
paul@paul:~$ nano report.cpp
paul@paul:~$ cat  report.cpp
#include <iostream>
#include <cstdlib>   // for system()
using namespace std;

int main() {

    cout << "----- Linux System Information -----\n" << endl;

    // Kernel version
    cout << "Kernel Version:\n";
    system("uname -r");
    cout << endl;

    // CPU type / architecture
    cout << "CPU Type (Architecture):\n";
    system("uname -m");
    cout << endl;

    // Full CPU information
    cout << "Detailed CPU Information:\n";
    system("cat /proc/cpuinfo");
    cout << endl;

    return 0;
}
paul@paul:~$ g++ report.cpp -o report
paul@paul:~$ ./report
----- Linux System Information -----

Kernel Version:
6.14.0-36-generic

CPU Type (Architecture):
x86_64

Detailed CPU Information:
processor       : 0
vendor_id       : AuthenticAMD
cpu family      : 26
model           : 68
model name      : AMD Ryzen 5 9600X 6-Core Processor
stepping        : 0
microcode       : 0xffffffff
cpu MHz         : 3892.604
cache size      : 1024 KB
physical id     : 0
siblings        : 6
core id         : 0
cpu cores       : 6
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht s
yscall nx mmxext fxsr_opt rdtscp lm constant_tsc rep_good nopl nonstop_tsc cpuid extd_apicid tsc_known_freq pni pclmulqd
q ssse3 fma cx16 sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand hypervisor lahf_lm cmp_legacy cr8_legacy abm sse4a
bugs            : fxsave_leak sysret_ss_attrs null_seg spectre_v1 spectre_v2 srso
bogomips        : 7785.20
TLB size        : 192 4K pages
clflush size    : 64
cache_alignment : 64
address sizes   : 48 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id       : AuthenticAMD
cpu family      : 26
model           : 68
model name      : AMD Ryzen 5 9600X 6-Core Processor
stepping        : 0
microcode       : 0xffffffff
cpu MHz         : 3892.604
cache size      : 1024 KB
physical id     : 0
siblings        : 6
core id         : 1
cpu cores       : 6
apicid          : 1
initial apicid  : 1
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht s
yscall nx mmxext fxsr_opt rdtscp lm constant_tsc rep_good nopl nonstop_tsc cpuid extd_apicid tsc_known_freq pni pclmulqd
q ssse3 fma cx16 sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand hypervisor lahf_lm cmp_legacy cr8_legacy abm sse4a
 misalignsse 3dnowprefetch vmmcall fsgsbase bmi1 avx2 bmi2 invpcid rdseed adx clflushopt sha_ni arat
```

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

```
paul@paul:~$ nano memory.cpp
paul@paul:~$ cat memory.cpp
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main() {
    ifstream meminfo("/proc/meminfo");
    if (!meminfo) {
        cout << "Error: Cannot open /proc/meminfo" << endl;
        return 1;
    }

    string key, unit;
    long memTotal = 0;
    long memFree = 0;
    long buffers = 0;
    long cached = 0;
    long value = 0;

    // Read data line by line
    while (meminfo >> key >> value >> unit) {
        if (key == "MemTotal:")   memTotal = value;
        else if (key == "MemFree:")  memFree = value;
        else if (key == "Buffers:")  buffers = value;
        else if (key == "Cached:")   cached = value;
    }

    long usedMemory = memTotal - (memFree + buffers + cached);

    cout << "\n--- Linux Memory Information ---\n";
    cout << "Configured Memory (Total): " << memTotal  << " kB\n";
    cout << "Free Memory:               " << memFree   << " kB\n";
    cout << "Buffers:                   " << buffers   << " kB\n";
    cout << "Cached:                    " << cached    << " kB\n";
    cout << "Used Memory (Calculated): " << usedMemory << " kB\n";

    return 0;
}


paul@paul:~$ g++ memory.cpp -o memory
paul@paul:~$ ./memory

--- Linux Memory Information ---
Configured Memory (Total): 6067340 kB
Free Memory:               368080 kB
Buffers:                   74116 kB
Cached:                    2457288 kB
Used Memory (Calculated): 3167856 kB
paul@paul:~$
```

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

```
paul@paul:~$ vi copy_syscalls.cpp
paul@paul:~$ cat copy_syscalls.cpp
#include <iostream>
 #include <fcntl.h> // open
 #include <unistd.h> // read, write, close
 #include <sys/stat.h>

 using namespace std;

 int main(int argc, char* argv[]) {
 if (argc != 3) {
 cerr << "Usage: " << argv[0] << " <source> <destination>\n";
 return 1;
 }

 int src = open(argv[1], O_RDONLY);
 if (src < 0) {
 perror("open source");
 return 1;
 }

 int dest = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, 0644);
 if (dest < 0) {
 perror("open dest");
 close(src);
 return 1;
 }
```

```
 char buffer[4096];
 ssize_t bytes;
 while ((bytes = read(src, buffer, sizeof(buffer))) > 0) {
 if (write(dest, buffer, bytes) != bytes) {
 perror("write");
 close(src);
 close(dest);
 return 1;
 }
 }

 if (bytes < 0) perror("read");

 close(src);
 close(dest);

 cout << "File copied successfully.\n";
 return 0;
 }
paul@paul:~$ g++ copy_syscalls.cpp -o copy_syscalls
paul@paul:~$ ./copy_syscalls
Usage: ./copy_syscalls <source> <destination>
paul@paul:~$ echo "hello os lab" > src.txt
paul@paul:~$ ./copy_syscalls src.txt dst.txt
File copied successfully.
paul@paul:~$ cat dst.txt
hello os lab
paul@paul:~$
```

7. Write a program to implement first-come, first-served FCFS scheduling algorithm.

```
paul@paul:~$ vi fcfs.cpp
paul@paul:~$ cat fcfs.cpp
#include <iostream>
using namespace std;

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

    int pid[n], arrival[n], burst[n], wt[n], tat[n];
    int current = 0;
    double totalW = 0, totalT = 0;

    // Input
    for (int i = 0; i < n; ++i) {
        pid[i] = i + 1;
        cout << "\nEnter Arrival Time for P" << pid[i] << ": ";
        cin >> arrival[i];
        cout << "Enter Burst Time for P" << pid[i] << ": ";
        cin >> burst[i];
    }

    // FCFS Calculation
    for (int i = 0; i < n; ++i) {
        if (current < arrival[i])
            current = arrival[i];   // CPU idle until process arrives

        wt[i] = current - arrival[i];
        current += burst[i];
        tat[i] = wt[i] + burst[i];
        totalW += wt[i];
        totalT += tat[i];
    }

    // Output
    cout << "\nFCFS Scheduling Result\n";
    cout << "PID\tAT\tBT\tWT\tTAT\n";

    for (int i = 0; i < n; ++i) {
        cout << pid[i] << '\t'
             << arrival[i] << '\t'
             << burst[i]   << '\t'
             << wt[i]      << '\t'
             << tat[i]     << '\n';
    }

    cout << "\nAverage Waiting Time    = " << totalW / n;
    cout << "\nAverage Turnaround Time = " << totalT / n << endl;

    return 0;
}

paul@paul:~$ g++ fcfs.cpp -o fcfs
paul@paul:~$ ./fcfs
Enter number of processes: 4

Enter Arrival Time for P1: 0
Enter Burst Time for P1: 5

Enter Arrival Time for P2: 1
Enter Burst Time for P2: 3

Enter Arrival Time for P3: 2
Enter Burst Time for P3: 1

Enter Arrival Time for P4: 3
Enter Burst Time for P4: 2

FCFS Scheduling Result
PID     AT      BT      WT      TAT
1       0       5       0       5
2       1       3       4       7
3       2       1       6       7
4       3       2       6       8

Average Waiting Time    = 4
Average Turnaround Time = 6.75
paul@paul:~$
```

8. Write a program to implement Shortest Job First SJF Scheduling Algorithm.

```
paul@paul:~$ vi sjf.cpp
paul@paul:~$ cat sjf.cpp
#include <iostream>
using namespace std;

struct Process {
    int pid;
    int at;     // Arrival Time
    int bt;     // Burst Time
    int wt;     // Waiting Time
    int tat;    // Turnaround Time
    bool finished;
};

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

    Process p[n];

    // Input
    for (int i = 0; i < n; ++i) {
        p[i].pid = i + 1;
        cout << "\nEnter Arrival Time for P" << p[i].pid << ": ";
        cin >> p[i].at;
        cout << "Enter Burst Time for P" << p[i].pid << ": ";
        cin >> p[i].bt;

        p[i].finished = false;
    }
    int current = 0, completed = 0;
    double totalW = 0, totalT = 0;

    // SJF Non-preemptive Scheduling
    while (completed < n) {
        int idx = -1;
        int minBT = 1e9;

        // Select shortest job among arrived and not finished
        for (int i = 0; i < n; ++i) {
            if (!p[i].finished && p[i].at <= current && p[i].bt < minBT) {
                minBT = p[i].bt;
                idx = i;
            }
        }

        if (idx == -1) {   // CPU is idle
            current++;
            continue;
        }

        p[idx].wt = current - p[idx].at;
        current += p[idx].bt;
        p[idx].tat = p[idx].wt + p[idx].bt;
        p[idx].finished = true;
        completed++;

        totalW += p[idx].wt;
        totalT += p[idx].tat;
    }
```

```
    // Output
    cout << "\nSJF (Non-preemptive) Scheduling Result\n";
    cout << "PID\tAT\tBT\tWT\tTAT\n";

    for (int i = 0; i < n; ++i) {
        cout << p[i].pid << '\t'
             << p[i].at  << '\t'
             << p[i].bt  << '\t'
             << p[i].wt  << '\t'
             << p[i].tat << '\n';
    }

    cout << "\nAverage Waiting Time   = " << totalW / n;
    cout << "\nAverage Turnaround Time = " << totalT / n << endl;

    return 0;
}

paul@paul:~$ g++ sjf.cpp -o sjf
paul@paul:~$ ./sjf
Enter number of processes: 4

Enter Arrival Time for P1: 0
Enter Burst Time for P1: 5

Enter Arrival Time for P2: 1
Enter Burst Time for P2: 3

Enter Arrival Time for P3: 2
Enter Burst Time for P3: 1

Enter Arrival Time for P4: 3
Enter Burst Time for P4: 2

SJF (Non-preemptive) Scheduling Result
PID     AT      BT      WT      TAT
1       0       5       0       5
2       1       3       7       10
3       2       1       3       4
4       3       2       3       5

Average Waiting Time   = 3.25
Average Turnaround Time = 6
paul@paul:~$
```

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

```
paul@paul:~$ vi priority.cpp
paul@paul:~$ cat priority.cpp
#include <iostream>
using namespace std;

struct Process {
    int pid;
    int at;     // Arrival Time
    int bt;     // Burst Time
    int pr;     // Priority (smaller value = higher priority)
    int wt;     // Waiting Time
    int tat;    // Turnaround Time
    bool finished;
};

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

    Process p[n];

    //  Input
    for (int i = 0; i < n; ++i) {
        p[i].pid = i + 1;

        cout << "\nEnter Arrival Time for P" << p[i].pid << ": ";
        cin >> p[i].at;

        cout << "Enter Burst Time for P" << p[i].pid << ": ";
        cin >> p[i].bt;
        cout << "Enter Priority for P" << p[i].pid
             << " (smaller number = higher priority): ";
        cin >> p[i].pr;

        p[i].finished = false;
    }

    int current = 0, completed = 0;
    double totalW = 0, totalT = 0;

    //  Non-Preemptive Priority Scheduling Logic
    while (completed < n) {
        int idx = -1;
        int bestPr = 1e9;

        // Select highest priority process among arrived & not finished
        for (int i = 0; i < n; ++i) {
            if (!p[i].finished && p[i].at <= current &&
                p[i].pr < bestPr) {
                bestPr = p[i].pr;
                idx = i;
            }
        }

        if (idx == -1) {   // CPU is idle
            current++;
            continue;
        }
```

```
        p[idx].wt = current - p[idx].at;
        current += p[idx].bt;
        p[idx].tat = p[idx].wt + p[idx].bt;
        p[idx].finished = true;
        completed++;

        totalW += p[idx].wt;
        totalT += p[idx].tat;
    }

    //  Output
    cout << "\nNon-Preemptive Priority Scheduling Result\n";
    cout << "PID\tAT\tBT\tPR\tWT\tTAT\n";

    for (int i = 0; i < n; ++i) {
        cout << p[i].pid << '\t'
             << p[i].at  << '\t'
             << p[i].bt  << '\t'
             << p[i].pr  << '\t'
             << p[i].wt  << '\t'
             << p[i].tat << '\n';
    }

    cout << "\nAverage Waiting Time   = " << totalW / n;
    cout << "\nAverage Turnaround Time = " << totalT / n << endl;

    return 0;
}
paul@paul:~$ g++ priority.cpp -o priority
paul@paul:~$ ./priority
Enter number of processes: 4

Enter Arrival Time for P1: 0
Enter Burst Time for P1: 10
Enter Priority for P1 (smaller number = higher priority): 2

Enter Arrival Time for P2: 1
Enter Burst Time for P2: 5
Enter Priority for P2 (smaller number = higher priority): 1

Enter Arrival Time for P3: 2
Enter Burst Time for P3: 8
Enter Priority for P3 (smaller number = higher priority): 3

Enter Arrival Time for P4: 3
Enter Burst Time for P4: 6
Enter Priority for P4 (smaller number = higher priority): 2

Non-Preemptive Priority Scheduling Result
PID     AT      BT      PR      WT      TAT
1       0       10      2       0       10
2       1       5       1       9       14
3       2       8       3       19      27
4       3       6       2       12      18

Average Waiting Time   = 10
Average Turnaround Time = 17.25
```

10. Write a program to implement SRTF scheduling algorithm.

```
paul@paul:~$ vi srtf.cpp
paul@paul:~$ cat srtf.cpp
#include <iostream>
#include <climits>
using namespace std;

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

    int pid[n], at[n], bt[n], rt[n];
    int ct[n], wt[n], tat[n];

    //  Input
    for (int i = 0; i < n; ++i) {
        pid[i] = i + 1;

        cout << "\nEnter Arrival Time for P" << pid[i] << ": ";
        cin >> at[i];

        cout << "Enter Burst Time for P" << pid[i] << ": ";
        cin >> bt[i];

        rt[i] = bt[i];   // remaining time initially = burst time
    }

    int completed = 0;
    int time = 0;

   //  Preemptive SRTF Scheduling
   while (completed < n) {
       int idx = -1;
       int minRT = INT_MAX;

       // Select process with smallest remaining time among arrived
       for (int i = 0; i < n; ++i) {
           if (at[i] <= time && rt[i] > 0 && rt[i] < minRT) {
               minRT = rt[i];
               idx = i;
           }
       }

       if (idx == -1) {   // CPU idle
           time++;
           continue;
       }

       // Run selected process for 1 time unit
       rt[idx]--;
       time++;

       // If process finishes
       if (rt[idx] == 0) {
           completed++;
           ct[idx] = time;   // completion time
       }
   }

   double totalW = 0, totalT = 0;
```

```
    //  Calculate WT and TAT
    for (int i = 0; i < n; ++i) {
        tat[i] = ct[i] - at[i];     // Turnaround Time
        wt[i]  = tat[i] - bt[i];   // Waiting Time

        totalW += wt[i];
        totalT += tat[i];
    }

    //  Output
    cout << "\nSRTF (Preemptive) Scheduling Result\n";
    cout << "PID\tAT\tBT\tCT\tWT\tTAT\n";

    for (int i = 0; i < n; ++i) {
        cout << pid[i] << '\t'
             << at[i]  << '\t'
             << bt[i]  << '\t'
             << ct[i]  << '\t'
             << wt[i]  << '\t'
             << tat[i] << '\n';
    }

    cout << "\nAverage Waiting Time   = " << totalW / n;
    cout << "\nAverage Turnaround Time = " << totalT / n << endl;

    return 0;
}

paul@paul:~$ g++ srtf.cpp -o srtf
paul@paul:~$ ./srtf
Enter number of processes: 4

Enter Arrival Time for P1: 0
Enter Burst Time for P1: 8

Enter Arrival Time for P2: 1
Enter Burst Time for P2: 4

Enter Arrival Time for P3: 2
Enter Burst Time for P3: 9

Enter Arrival Time for P4: 3
Enter Burst Time for P4: 5

SRTF (Preemptive) Scheduling Result
PID     AT      BT      CT      WT      TAT
1       0       8       17      9       17
2       1       4       5       0       4
3       2       9       26      15      24
4       3       5       10      2       7

Average Waiting Time   = 6.5
Average Turnaround Time = 13
paul@paul:~$
```

11. Write a program to calculate sum of n numbers using pthreads.

```
paul@paul:~$ vi thread.cpp
paul@paul:~$ cat thread.cpp
#include <iostream>
#include <pthread.h>
using namespace std;

long long sum = 0;          // Shared global sum
pthread_mutex_t mutex;     // Mutex for synchronization

struct ThreadData {
    int start;
    int end;
};

// Thread function
void* partial_sum(void* arg) {
    ThreadData* data = (ThreadData*)arg;
    long long local_sum = 0;

    for (int i = data->start; i <= data->end; i++) {
        local_sum += i;
    }

    pthread_mutex_lock(&mutex);   // Lock before updating shared sum
    sum += local_sum;
    pthread_mutex_unlock(&mutex); // Unlock after update

    pthread_exit(NULL);
}
int main() {
    int n, num_threads;

    cout << "Enter n: ";
    cin >> n;

    cout << "Enter number of threads: ";
    cin >> num_threads;

    pthread_t threads[num_threads];
    ThreadData tdata[num_threads];

    pthread_mutex_init(&mutex, NULL);

    int range = n / num_threads;
    int start = 1;

    for (int i = 0; i < num_threads; i++) {
        tdata[i].start = start;

        if (i == num_threads - 1)
            tdata[i].end = n;
        else
            tdata[i].end = start + range - 1;

        start = tdata[i].end + 1;

        pthread_create(&threads[i], NULL, partial_sum, &tdata[i]);
    }
```

```
    for (int i = 0; i < num_threads; i++) {
        pthread_join(threads[i], NULL);
    }

    pthread_mutex_destroy(&mutex);

    cout << "Sum = " << sum << endl;

    return 0;
}

paul@paul:~$ g++ thread.cpp -o thread -pthread
paul@paul:~$ ./thread
Enter n: 4
Enter number of threads: 4
Sum = 10
paul@paul:~$ ./thread
Enter n: 8
Enter number of threads: 4
Sum = 36
paul@paul:~$
```

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

```
paul@paul:~$ vi allocation.cpp
paul@paul:~$ cat allocation.cpp
#include <iostream>
#include <climits>
using namespace std;

void printAllocation(const char* title,
                     int alloc[],
                     int process[],
                     int NP) {
    cout << '\n' << title << ":\n";
    cout << "Process\tSize\tBlock\n";
    for (int i = 0; i < NP; ++i) {
        cout << "P" << i + 1 << "\t" << process[i] << "\t";
        if (alloc[i] == -1)
            cout << "Not Allocated";
        else
            cout << "B" << alloc[i] + 1;
        cout << '\n';
    }
}

int main() {
    int NB, NP;

    cout << "Enter number of memory blocks: ";
    cin >> NB;

    cout << "Enter number of processes: ";
    cin >> NP;

    int blocks[NB], process[NP];

    // Input memory blocks
    cout << "\nEnter sizes of memory blocks:\n";
    for (int i = 0; i < NB; ++i) {
        cout << "Block " << i + 1 << ": ";
        cin >> blocks[i];
    }

    // Input process sizes
    cout << "\nEnter sizes of processes:\n";
    for (int i = 0; i < NP; ++i) {
        cout << "Process P" << i + 1 << ": ";
        cin >> process[i];
    }

    /* ---------- First Fit ---------- */
    int b1[NB], allocFF[NP];
    for (int i = 0; i < NB; ++i) b1[i] = blocks[i];
    for (int i = 0; i < NP; ++i) allocFF[i] = -1;

    for (int i = 0; i < NP; ++i) {
        for (int j = 0; j < NB; ++j) {
            if (b1[j] >= process[i]) {
                allocFF[i] = j;
                b1[j] -= process[i];
                break;
            }
        }
    }
```

```
    printAllocation("First Fit Allocation", allocFF, process, NP);

    /* ---------- Best Fit ---------- */
    int b2[NB], allocBF[NP];
    for (int i = 0; i < NB; ++i) b2[i] = blocks[i];
    for (int i = 0; i < NP; ++i) allocBF[i] = -1;

    for (int i = 0; i < NP; ++i) {
        int bestIdx = -1;
        int bestRem = INT_MAX;

        for (int j = 0; j < NB; ++j) {
            if (b2[j] >= process[i] && b2[j] - process[i] < bestRem) {
                bestRem = b2[j] - process[i];
                bestIdx = j;
            }
        }

        if (bestIdx != -1) {
            allocBF[i] = bestIdx;
            b2[bestIdx] -= process[i];
        }
    }
    printAllocation("Best Fit Allocation", allocBF, process, NP);

    /* ---------- Worst Fit ---------- */
    int b3[NB], allocWF[NP];
    for (int i = 0; i < NB; ++i) b3[i] = blocks[i];
    for (int i = 0; i < NP; ++i) allocWF[i] = -1;
    for (int i = 0; i < NP; ++i) {
        int worstIdx = -1;
        int worstRem = -1;

        for (int j = 0; j < NB; ++j) {
            if (b3[j] >= process[i] && b3[j] - process[i] > worstRem) {
                worstRem = b3[j] - process[i];
                worstIdx = j;
            }
        }

        if (worstIdx != -1) {
            allocWF[i] = worstIdx;
            b3[worstIdx] -= process[i];
        }
    }
    printAllocation("Worst Fit Allocation", allocWF, process, NP);

    return 0;
}

paul@paul:~$ g++ allocation.cpp -o allocation
paul@paul:~$ ./allocation
Enter number of memory blocks: 4
Enter number of processes: 4

Enter sizes of memory blocks:
Block 1: 5
Block 2: 2
Block 3: 4
Block 4: 6
```

```
Enter sizes of processes:
Process P1: 2
Process P2: 6
Process P3: 5
Process P4: 2

First Fit Allocation:
Process Size    Block
P1      2       B1
P2      6       B4
P3      5       Not Allocated
P4      2       B1

Best Fit Allocation:
Process Size    Block
P1      2       B2
P2      6       B4
P3      5       B1
P4      2       B3

Worst Fit Allocation:
Process Size    Block
P1      2       B4
P2      6       Not Allocated
P3      5       B1
P4      2       B3
paul@paul:~$
```