PRACTICAL FILE

RAMANUJAN COLLEGE

UNIVERSITY OF DELHI

DSC: OPERATING SYSTEMS

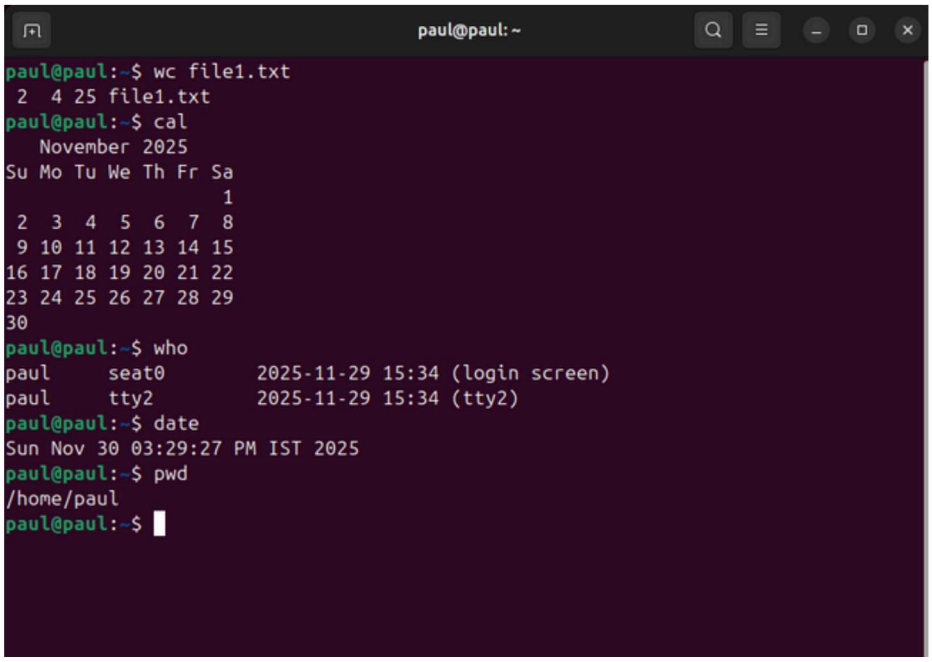
2025-26

SEMESTER 3

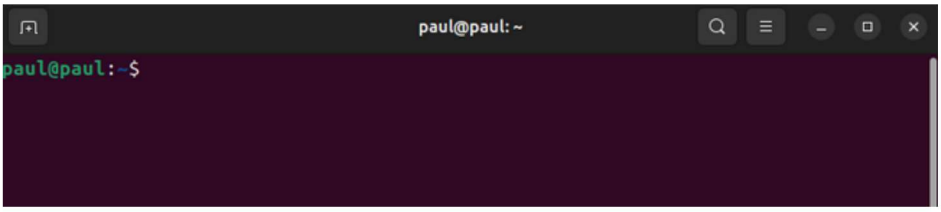
	SUBMITTED TO
SUBMITTED BY NAME: GURPREET SINGH PAUL ROLL NO. 24570022(24020570023) COURSE: B.Sc(H)Computer Science SEMESTER:3	Mrs.Sheetal Singh

Q-1 :- Execute various LINUX commands for:

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



Clear



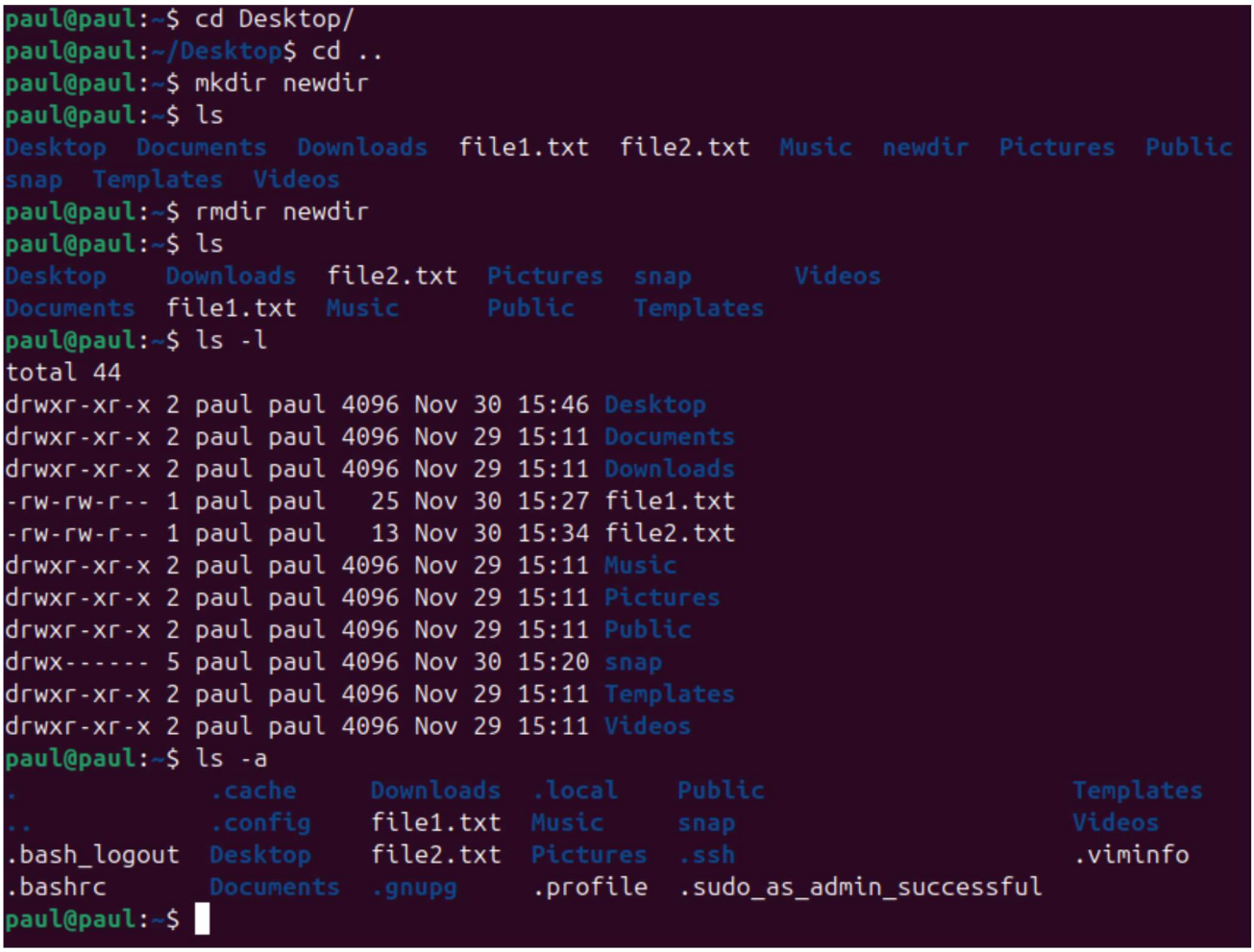
(ii) File Management: cat, cp, rm, mv, cmp, comm, diff, find, grep, awk.

```
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:~$ cat report1.cpp
#include <iostream>
#include <sys/utsname.h>
using namespace std;

int main() {
    struct utsname u;
    uname(&u);

    cout << "System Name  : " << u.sysname << endl;
    cout << "Node Name    : " << u.nodename << endl;
    cout << "Kernel Ver   : " << u.version << endl;
    cout << "Kernel Rel   : " << u.release << endl;
    cout << "Machine Type: " << u.machine << endl;

    system("cat /proc/cpuinfo | head -10");
    return 0;
}
```

```
paul@paul:~$ g++ report1.cpp -o report1
paul@paul:~$ ./report1
System Name  : Linux
Node Name    : paul
Kernel Ver   : #36~24.04.1-Ubuntu SMP PREEMPT_DYNAMIC Wed Oct 15 15:45:17 UTC 2
Kernel Rel   : 6.14.0-36-generic
Machine Type: x86_64
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
paul@paul:~$
```

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

```
paul@paul:~$ vi memory1.cpp
paul@paul:~$ cat memory1.cpp
#include <iostream>
using namespace std;

int main() {
    cout << "Memory Information:\n";
    system("cat /proc/meminfo | head -5");
    return 0;
}

paul@paul:~$ g++ memory1.cpp -o memory1
paul@paul:~$ ./memory1
Memory Information:
MemTotal:        6067340 kB
MemFree:          866680 kB
MemAvailable:    3585384 kB
Buffers:           65132 kB
Cached:          2265248 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 in CPP code.

```
paul@paul:~$ vi fcfs1.cpp
paul@paul:~$ cat fcfs1.cpp
#include <iostream>
using namespace std;

int main() {
    const int n = 4;
    int pid[n]     = {1, 2, 3, 4};
    int arrival[n] = {0, 1, 2, 3};
    int burst[n]   = {5, 3, 1, 2};

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

    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];
    }

    cout << "FCFS Scheduling\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++ fcfs1.cpp -o fcfs1
paul@paul:~$ ./fcfs1
FCFS Scheduling
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 s
sameproco          sameprodiffco      sjf               snap/             s.txt
sameproco.cpp      sameprodiffco.cpp  sjf.cpp           src.txt
paul@paul:~$ cat sjf.cpp
#include <iostream>
using namespace std;

struct Process {
    int pid;
    int at;
    int bt;
    int wt;
    int tat;
    bool finished;
};

int main() {
    const int n = 4;
    Process p[n] = {
        {1, 0, 6, 0, 0, false},
        {2, 1, 2, 0, 0, false},
        {3, 2, 8, 0, 0, false},
        {4, 3, 3, 0, 0, false}
    };

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

    while (completed < n) {
       int idx = -1;
       int minBT = 1e9;

       // choose shortest burst among arrived & 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) {       // no process has arrived yet
           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;
   }

   cout << "SJF (Non-preemptive) Scheduling\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
SJF (Non-preemptive) Scheduling
PID     AT      BT      WT      TAT
1       0       6       0       6
2       1       2       5       7
3       2       8       9       17
4       3       3       5       8

Average Waiting Time   = 4.75
Average Turnaround Time = 9.5
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;
    int bt;
    int pr;    // smaller value = higher priority
    int wt;
    int tat;
    bool finished;
};

int main() {
    const int n = 4;
    Process p[n] = {
        {1, 0, 10, 2, 0, 0, false},
        {2, 1,  5, 1, 0, 0, false},
        {3, 2,  8, 3, 0, 0, false},
        {4, 3,  6, 2, 0, 0, false}
    };

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

    while (completed < n) {
        int idx = -1;
        int bestPr = 1e9;
```

```
        // pick highest priority (smallest pr) among arrived
        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) {
            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;
    }

    cout << "Non-preemptive Priority Scheduling\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
Non-preemptive Priority Scheduling
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
paul@paul:~$
```

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() {
    const int n = 4;

    int pid[n] = {1, 2, 3, 4};
    int at[n]  = {0, 1, 2, 3};        // arrival times
    int bt[n]  = {8, 4, 9, 5};        // burst times

    int rt[n];                        // remaining times
    int ct[n];                        // completion times
    int wt[n], tat[n];

    for (int i = 0; i < n; ++i)
        rt[i] = bt[i];

    int completed = 0;
    int time = 0;
    int idx = -1;
    int minRT;

    while (completed < n) {
        idx = -1;
        minRT = INT_MAX;

        // pick 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) {        // no process has arrived yet
            time++;
            continue;
        }

        // run this process for 1 unit
        rt[idx]--;
        time++;

        // if finished now
        if (rt[idx] == 0) {
            completed++;
            ct[idx] = time;     // completion time
        }
    }

    double totalW = 0, totalT = 0;

    for (int i = 0; i < n; ++i) {
        tat[i] = ct[i] - at[i];          // turnaround = completion - arrival
        wt[i]  = tat[i] - bt[i];         // waiting = turnaround - burst
        totalW += wt[i];
        totalT += tat[i];
    }
```

```
    cout << "SRTF (Preemptive) Scheduling\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
SRTF (Preemptive) Scheduling
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 pthreads.cpp
paul@paul:~$ cat pthreads.cpp
#include <iostream>
#include <pthread.h>
#include <vector>
using namespace std;

#define NUM_THREADS 4   // how many threads we will use

struct ThreadData {
    int start_index;            // starting index in the array
    int end_index;              // ending index (exclusive)
    const vector<int>* arr;     // pointer to the full array
    long long partial_sum;      // sum calculated by this thread
};

// Function executed by each thread
void* sumPart(void* arg) {
    ThreadData* data = (ThreadData*)arg;
    long long sum = 0;

    for (int i = data->start_index; i < data->end_index; ++i)
        sum += (*(data->arr))[i];

    data->partial_sum = sum;
    return nullptr;
}

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

    vector<int> nums(n);
    cout << "Enter elements:\n";
    for (int i = 0; i < n; ++i)
        cin >> nums[i];

    pthread_t threads[NUM_THREADS];
    ThreadData tdata[NUM_THREADS];

    // Divide the work among threads as evenly as possible
    int base_chunk = n / NUM_THREADS;   // minimum elements per thread
    int rem = n % NUM_THREADS;          // some threads get one extra element

    int start = 0;
    for (int i = 0; i < NUM_THREADS; ++i) {
        int extra = (i < rem) ? 1 : 0;          // distribute remainder
        int end = start + base_chunk + extra;   // end index (exclusive)

        tdata[i].start_index = start;
        tdata[i].end_index = end;
        tdata[i].arr = &nums;
        tdata[i].partial_sum = 0;

        pthread_create(&threads[i], nullptr, sumPart, &tdata[i]);
        start = end;    // next thread starts from here
    }

    // Collect results from all threads
    long long total_sum = 0;
```

```
    for (int i = 0; i < NUM_THREADS; ++i) {
        pthread_join(threads[i], nullptr);
        total_sum += tdata[i].partial_sum;
    }

    cout << "Total sum of the elements: " << total_sum << endl;
    return 0;
}

paul@paul:~$ g++ pthreads.cpp -o pthreads -pthread
paul@paul:~$ ./pthreads
Enter number of elements: 4
Enter elements:
1
2
3
4
Total sum of the elements: 10
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;

const int NB = 5; // number of blocks
const int NP = 4; // number of processes

void printAllocation(const char* title,
                     int alloc[NP],
                     const int process[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 blocks[NB]   = {100, 500, 200, 300, 600};
    int process[NP]  = {212, 417, 112, 426};

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

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

    // ---------- Worst Fit ----------
    int b3[NB];
    for (int i = 0; i < NB; ++i) b3[i] = blocks[i];
    int allocWF[NP];
    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);

    return 0;
}
```

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

First Fit Allocation:
Process Size    Block
P1      212     B2
P2      417     B5
P3      112     B2
P4      426     Not Allocated

Best Fit Allocation:
Process Size    Block
P1      212     B4
P2      417     B2
P3      112     B3
P4      426     B5

Worst Fit Allocation:
Process Size    Block
P1      212     B5
P2      417     B2
P3      112     B5
P4      426     Not Allocated
paul@paul:~$
```