Name : Atul_Arya
Roll No. : CSC/22/11

Practical File OS:

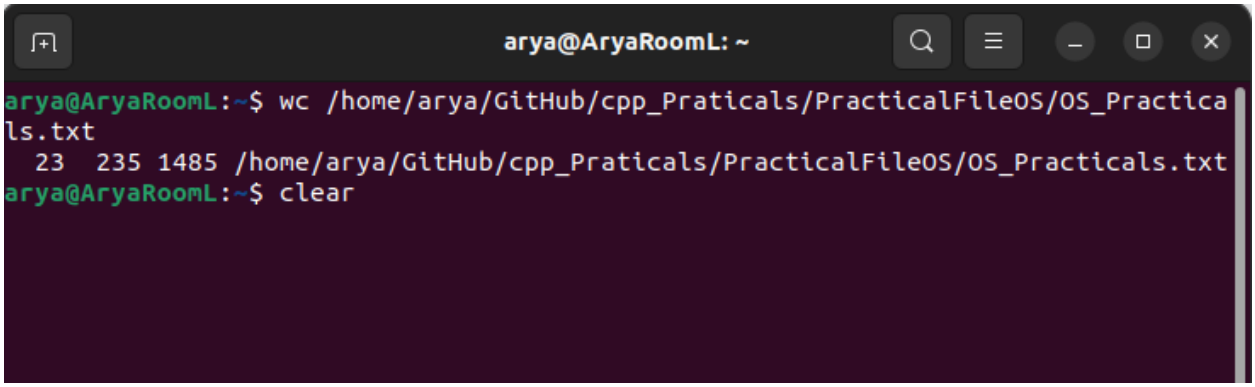
Question-1

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.

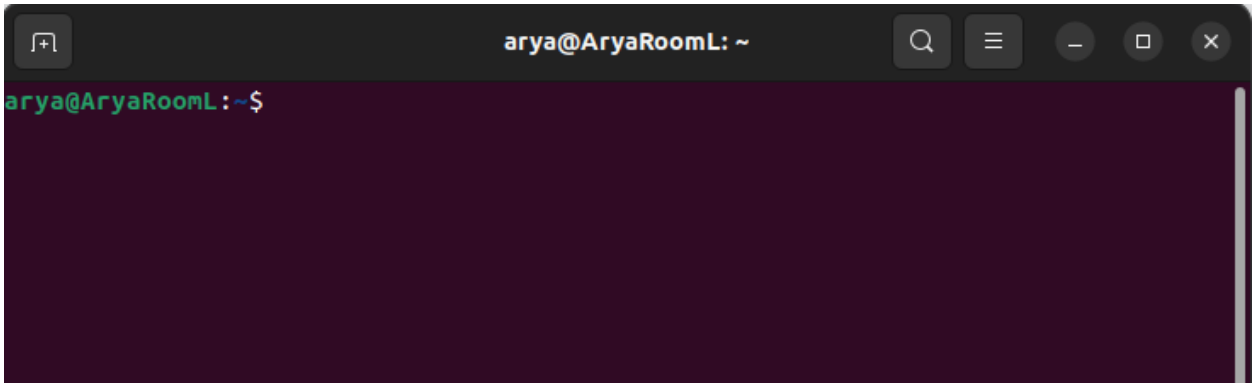
Solution-1

i.) **Information Maintenance**

command “wc”

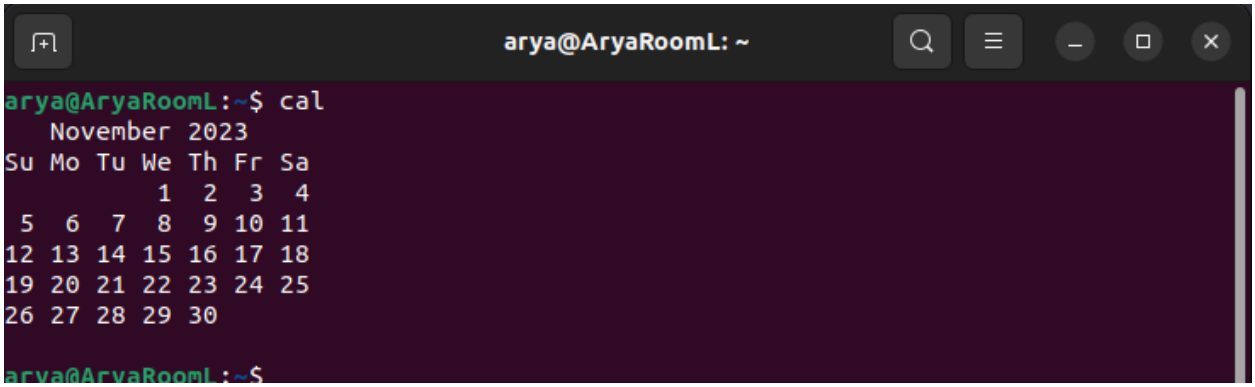


Command “clear”

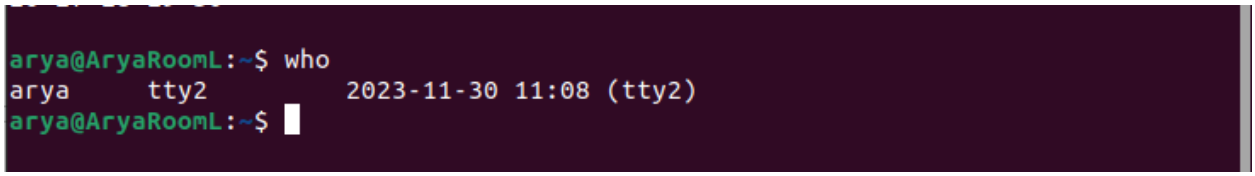


Name : Atul_Arya
Roll No. : CSC/22/11

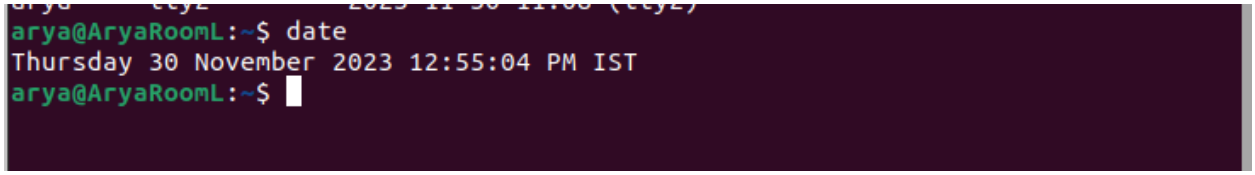
Command “cal”



Command “who”



Command “date”

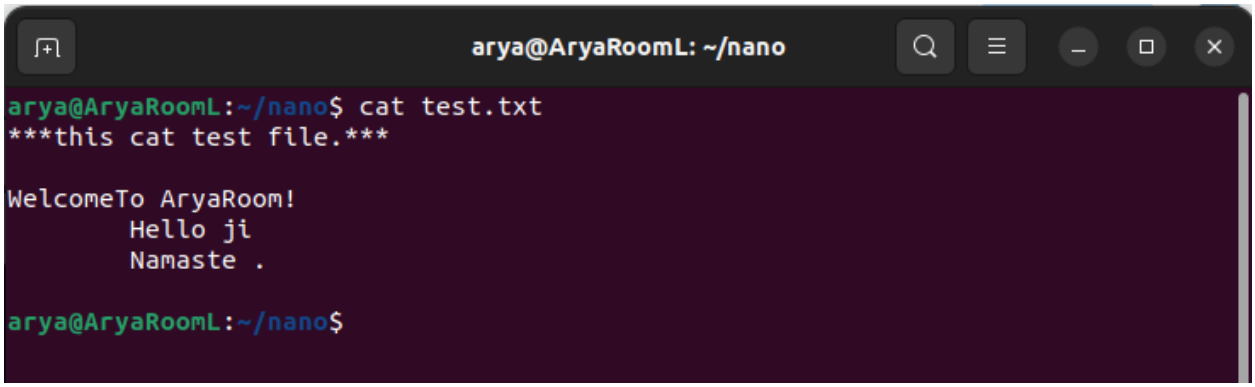


Command “pwd”

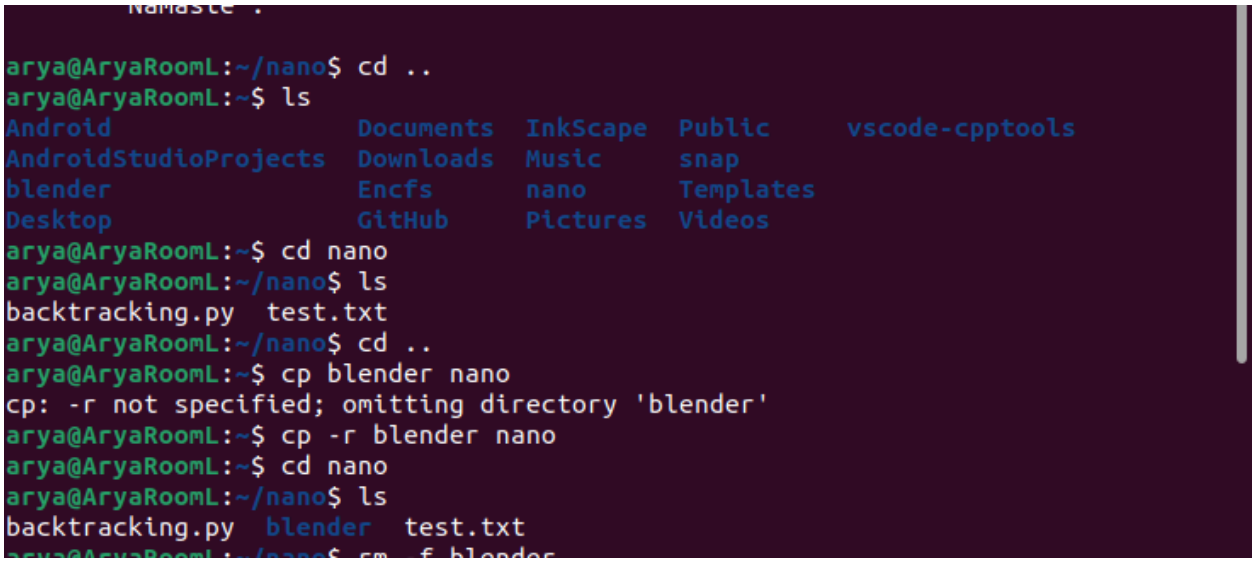
ii.) **File Management**

Name : Atul_Arya
Roll No. : CSC/22/11

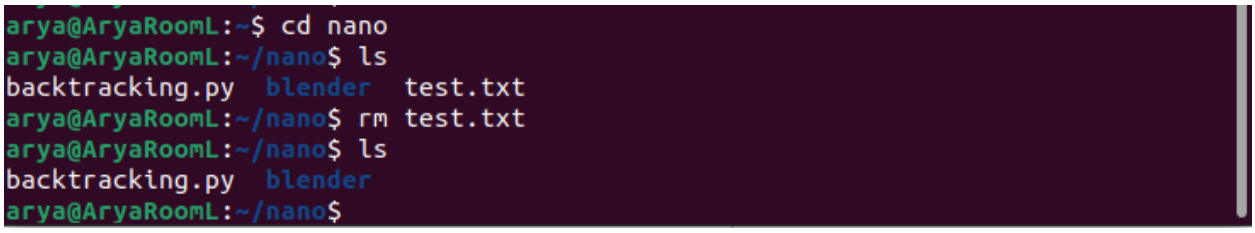
Command “cat”



Command “cp”

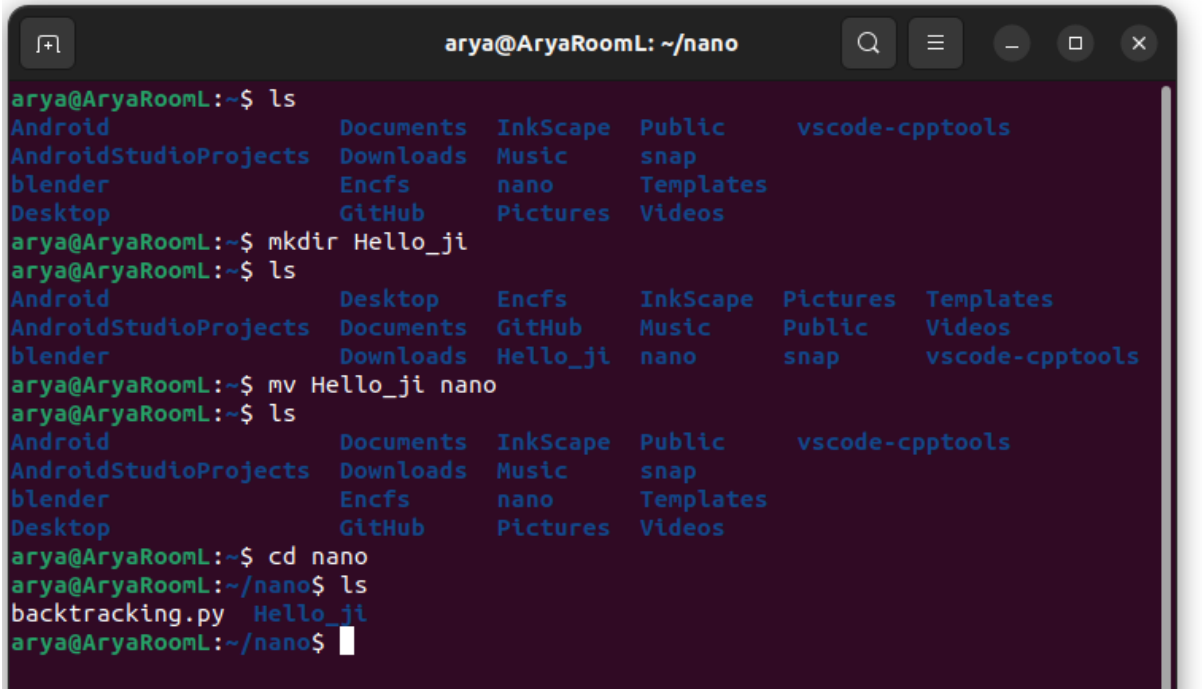


Command “rm”



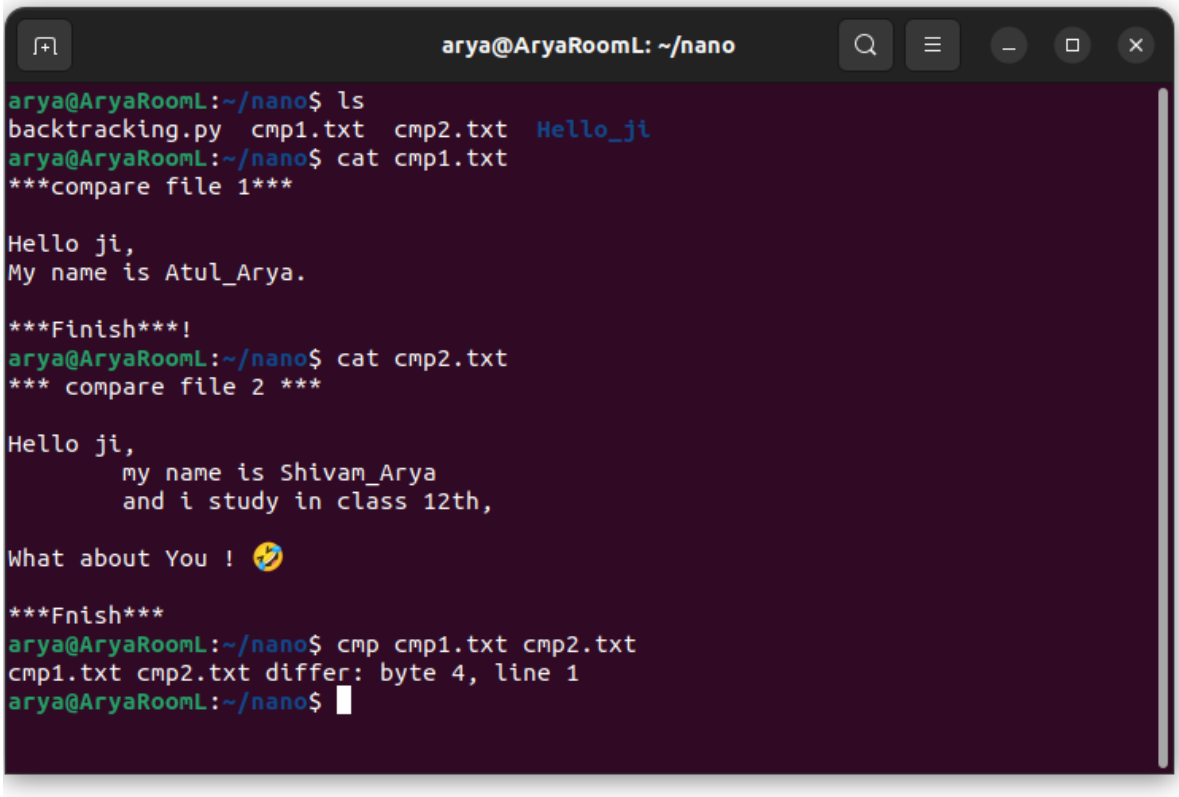
Name : Atul_Arya
Roll No. : CSC/22/11

Command “mv”



Command “cmp”

Name : Atul_Arya
Roll No. : CSC/22/11



Name : Atul_Arya
Roll No. : CSC/22/11

Command “diff”

```
arya@AryaRoomL:~/nano$ ls
backtracking.py  cmp1.txt  cmp2.txt  Hello_ji
arya@AryaRoomL:~/nano$ diff cmp1.txt cmp2.txt
1c1
< ***compare file 1***
---
> *** compare file 2 ***
4c4,5
< My name is Atul_Arya.
---
>       my name is Shivam_Arya
>       and i study in class 12th,
6c7,9
< ***Finish***!
---
> What about You ! 🤣
>
> ***Fnish***
arya@AryaRoomL:~/nano$
```

Command “find”

```
arya@AryaRoomL:~/nano$ find "text.txt"
find: ‘text.txt’: No such file or directory
arya@AryaRoomL:~/nano$ ls
backtracking.py  cmp1.txt  cmp2.txt  Hello_ji
arya@AryaRoomL:~/nano$ find "cmp1.txt"
cmp1.txt
arya@AryaRoomL:~/nano$
```

Command “grep”

```
arya@AryaRoomL:~/nano$ ls
backtracking.py  cmp1.txt  cmp2.txt  Hello_ji
arya@AryaRoomL:~/nano$ grep "Shivam" cmp1.txt
arya@AryaRoomL:~/nano$ grep "Atul" cmp1.txt
My name is Atul_Arya.
arya@AryaRoomL:~/nano$ cat cmp1.txt
***compare file 1***

Hello ji,
My name is Atul_Arya.

***Finish***!
arya@AryaRoomL:~/nano$
```

Name : Atul_Arya
Roll No. : CSC/22/11

Command “awk”

```
arya@AryaRoomL:~/nano$ cat cmp1.txt
***compare file 1***

Hello ji,
My name is Atul_Arya.

***Finish***!
arya@AryaRoomL:~/nano$ awk '{print $1 "      <------>       " $2}' cmp1.txt
***compare      <------>       file
      <------>
Hello      <------>       ji,
My      <------>       name
      <------>
***Finish***!      <------>
arya@AryaRoomL:~/nano$
```

iii.) **Directory Management**

Command “cd”

```
arya@AryaRoomL:~/nano$ cd ..
arya@AryaRoomL:~$ cd nano
arya@AryaRoomL:~/nano$
```

Command “mkdir”

```
arya@AryaRoomL:~/nano$ ls
backtracking.py  cmp1.txt  cmp2.txt  Hello_ji
arya@AryaRoomL:~/nano$ mkdir "this_empty_directory"
arya@AryaRoomL:~/nano$ ls
backtracking.py  cmp1.txt  cmp2.txt  Hello_ji  this_empty_directory
arya@AryaRoomL:~/nano$
```

Command “rmdir”

```
arya@AryaRoomL:~/nano$ ls
backtracking.py  cmp1.txt  cmp2.txt  Hello_ji  this_empty_directory
arya@AryaRoomL:~/nano$ rmdir "this_empty_directory"
arya@AryaRoomL:~/nano$ ls
backtracking.py  cmp1.txt  cmp2.txt  Hello_ji
arya@AryaRoomL:~/nano$
```

Command “ls”

Name : Atul_Arya
Roll No. : CSC/22/11

```
arya@AryaRoomL:~$ ls
Android                 Documents  InkScape  Public     vscode-cpptools
AndroidStudioProjects   Downloads  Music     snap
blender                 Encfs      nano      Templates
Desktop                 GitHub     Pictures  Videos
arya@AryaRoomL:~$
```

Execute LINUX commands for :

i.) **Process Control**:

fork(), getpid()

```
#include <stdio.h>
#include <unistd.h>

int main() {
    fork();
    printf("process ID of newly created process usig fork: %d\n", getpid());
    return 0;
}
```

Ouput:

```
}
arya@AryaRoomL:~/nano$ ./Q2_a
process ID of newly created process usig fork: 28062
process ID of newly created process usig fork: 28063
arya@AryaRoomL:~/nano$ ^C
arya@AryaRoomL:~/nano$
```

Command “ps”

```
arya@AryaRoomL:~/nano$ ps
    PID TTY          TIME CMD
  22668 pts/2    00:00:00 bash
  28392 pts/2    00:00:00 ps
arya@AryaRoomL:~/nano$
```

Name : Atul_Arya
Roll No. : CSC/22/11

ii.) **Communication:**

Input-output redirection

```
arya@AryaRoomL:~/nano$ ls
backtracking.py  cmp1.txt  cmp2.txt  Hello_ji  Q2_a  Q2_a.cpp
arya@AryaRoomL:~/nano$ ls > file.txt
arya@AryaRoomL:~/nano$ cat < file.txt
backtracking.py
cmp1.txt
cmp2.txt
file.txt
Hello_ji
Q2_a
Q2_a.cpp
arya@AryaRoomL:~/nano$
```

iii.) **Protection Management:**

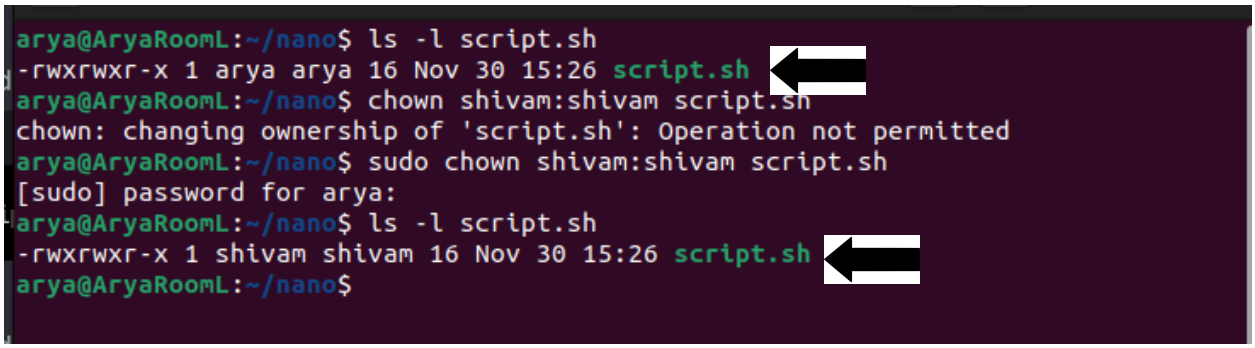
Command “chmod”

```
arya@AryaRoomL: ~/nano
arya@AryaRoomL:~/nano$ ls
backtracking.py  cmp1.txt  cmp2.txt  file.txt  Hello_ji  Q2_a  Q2_a.cpp
arya@AryaRoomL:~/nano$ nano script.sh
arya@AryaRoomL:~/nano$ ./script.sh
bash: ./script.sh: Permission denied
arya@AryaRoomL:~/nano$ ls
backtracking.py  cmp2.txt  Hello_ji  Q2_a.cpp
cmp1.txt         file.txt  Q2_a      script.sh
arya@AryaRoomL:~/nano$ chmod +x script.sh
arya@AryaRoomL:~/nano$ ./script.sh
[sudo] password for arya:
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Hit:2 http://in.archive.ubuntu.com/ubuntu jammy InRelease
Get:3 http://in.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Hit:4 https://ppa.launchpadcontent.net/gencfsm/ppa/ubuntu jammy InRelease
Hit:5 https://ppa.launchpadcontent.net/inkscape.dev/stable/ubuntu jammy InReleas
e
Hit:6 http://in.archive.ubuntu.com/ubuntu jammy-backports InRelease
Get:7 http://in.archive.ubuntu.com/ubuntu jammy-updates/main i386 Packages [537
kB]
Get:8 http://in.archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [1,2
11 kB]
Get:9 http://in.archive.ubuntu.com/ubuntu jammy-updates/restricted i386 Packages
 [32.8 kB]
```

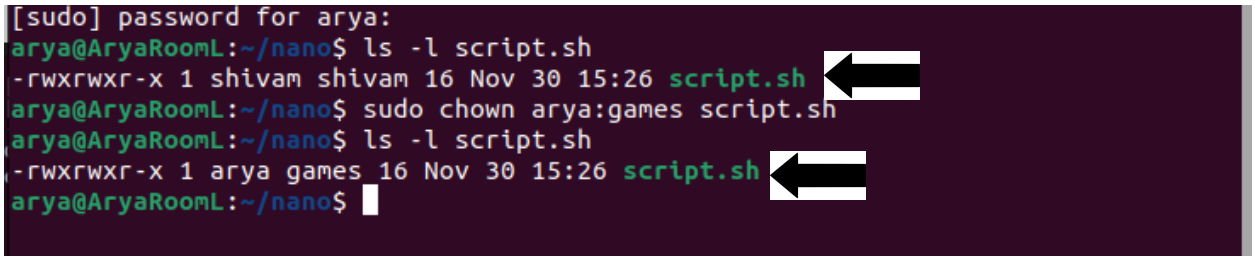
Name : Atul_Arya
Roll No. : CSC/22/11

Name : Atul_Arya
Roll No. : CSC/22/11

Command “chown”



Command “chgrp”



Question-3.

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

i.) **same program, same code.**

```
#include <stdio.h>
#include <unistd.h>

int main() {
    fork();
    printf("process ID of newly created process usig fork: %d\n", getpid());
    return 0;
}
```

Output:

Name : Atul_Arya
Roll No. : CSC/22/11

```
arya@AryaRoomL:~/nano$ ./Q2_a
process ID of newly created process usig fork: 30188
process ID of newly created process usig fork: 30189
arya@AryaRoomL:~/nano$
```

ii.) **same program, different code.**

```
#include<iostream>
#include<unistd.h>
using namespace std;

int main() {
int pid = fork();
        if (pid<0) {
                cout<<"UNSUCCESSFUL"<<endl;
                return -1;
        }
        else if(pid==0) {
                cout<<"I am a child process" <<" "<< pid <<" " <<getpid()<<endl;
        }else{
                sleep(-5);
                cout<<"I am parent process " <<" " << pid <<" " << getpid() <<endl;
        }
        return 0;
}
```

Output:

```
arya@AryaRoomL:~/nano$ nano Q2_b.cpp
arya@AryaRoomL:~/nano$ g++ Q2_b.cpp -o "Q2_b"
arya@AryaRoomL:~/nano$ ./Q2_b
I am a child process 0 30277
I am parent process  30277 30276
arya@AryaRoomL:~/nano$
```

Name : Atul_Arya
Roll No. : CSC/22/11

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

```
#include<iostream>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
using namespace std;
int main() {
        int code = fork();
        int status,x;
        if(code<0) {
                cout<<"UNSUCCESSFUL"<<endl;
        } else {
                x = wait(&status);
                cout<<"pid= " <<" "<< getpid() <<" "<< "return code = " << code<<"
"<<"x="<<" " << x <<endl;
        }
        return 0;
}
```

Output:

```
arya@AryaRoomL:~/nano$ ./Q2_c
pid=  30371 return code = 0 x= -1
pid=  30370 return code = 30371 x= 30371
arya@AryaRoomL:~/nano$
```

Question-4.

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

```
#include<iostream>
using namespace std;
int main(){
        cout<<"\n Kernel version:\n";
        system("uname -s");
        cout<<"\nCPU space: \n";
        system("cat /proc/cpuinfo |awk 'NR==3,NR==4{print}' \n");
        return 0;
}
```

Name : Atul_Arya
Roll No. : CSC/22/11

Output:

```
arya@AryaRoomL:~/nano$ ./Q4

 Kernel version:
Linux

CPU space:
cpu family      : 21
model           : 19
arya@AryaRoomL:~/nano$
```

Question-5.

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

```
#include<iostream>
using namespace std;
int main(){
        cout<<"\nConfigured memory is :\n";
        system("cat /proc/meminfo |awk 'NR==1{print $2}'\n");
        cout<<"\nAmount of free memory is :\n";
        system("cat /proc/meminfo |awk 'NR==2{print $2}'\n");
        cout<<"\nAmount of used memory is :\n";system("cat /proc/meminfo |awk '{if
(NR==1) a=$2; if (NR==2) b=$2 } END {print a-b}'\n");
        return 0;
}
```

Output:

```
arya@AryaRoomL:~/nano$ ./Q5

Configured memory is :
15540136

Amount of free memory is :
9539580

Amount of used memory is :
6000556
arya@AryaRoomL:~/nano$
```

Name : Atul_Arya
Roll No. : CSC/22/11

Question-6.

Write a program to copy files using system calls.

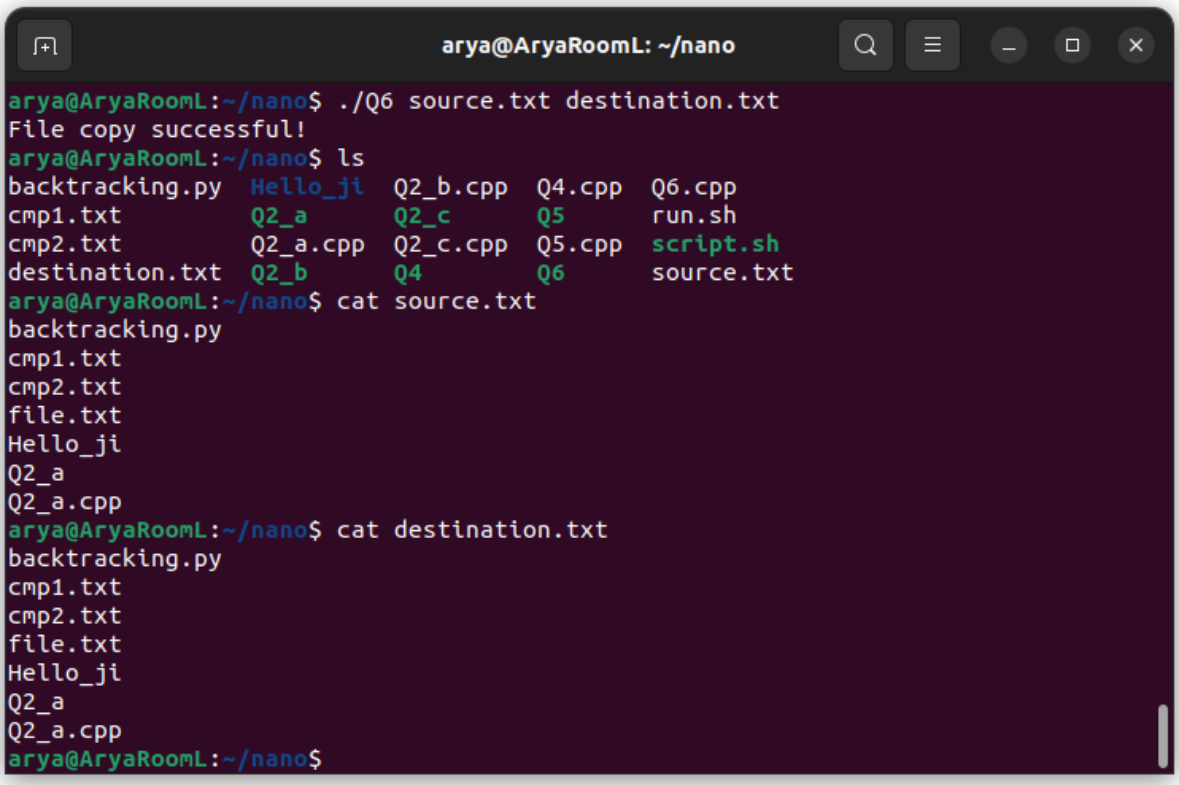
```
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

#define BUFFER_SIZE 4096

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <source_file> <destination_file>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    // Open the source file for reading
    int source_fd = open(argv[1], O_RDONLY);
    if (source_fd == -1) {
        perror("Error opening source file");
        exit(EXIT_FAILURE);
    }

    // Open or create the destination file for writing
    int dest_fd = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR |
S_IWUSR | S_IRGRP | S_IROTH);
    if (dest_fd == -1) {
        perror("Error opening destination file");
        close(source_fd);
        exit(EXIT_FAILURE);
    }

    // Copy contents from source to destination
    char buffer[BUFFER_SIZE];
    ssize_t bytesRead, bytesWritten;

    while ((bytesRead = read(source_fd, buffer, BUFFER_SIZE)) > 0) {
        bytesWritten = write(dest_fd, buffer, bytesRead);
        if (bytesWritten != bytesRead) {
            perror("Error writing to destination file");
            close(source_fd);
            close(dest_fd);
```

Name : Atul_Arya
Roll No. : CSC/22/11

```
            exit(EXIT_FAILURE);
        }
    }

    if (bytesRead == -1) {
        perror("Error reading from source file");
        close(source_fd);
        close(dest_fd);
        exit(EXIT_FAILURE);
    }

    // Close the file descriptors
    if (close(source_fd) == -1 || close(dest_fd) == -1) {
        perror("Error closing file descriptors");
        exit(EXIT_FAILURE);
    }

    printf("File copy successful!\n");

    return 0;
}
```

Output:

Name : Atul_Arya
Roll No. : CSC/22/11



Question-7.

Write a program to implement FCFS scheduling algorithm.

```
#include <stdio.h>

// Structure to represent a process
struct Process {
    int processID;
    int arrivalTime;
    int burstTime;
    int completionTime;
    int turnaroundTime;
    int waitingTime;
};
```

Name : Atul_Arya
Roll No. : CSC/22/11

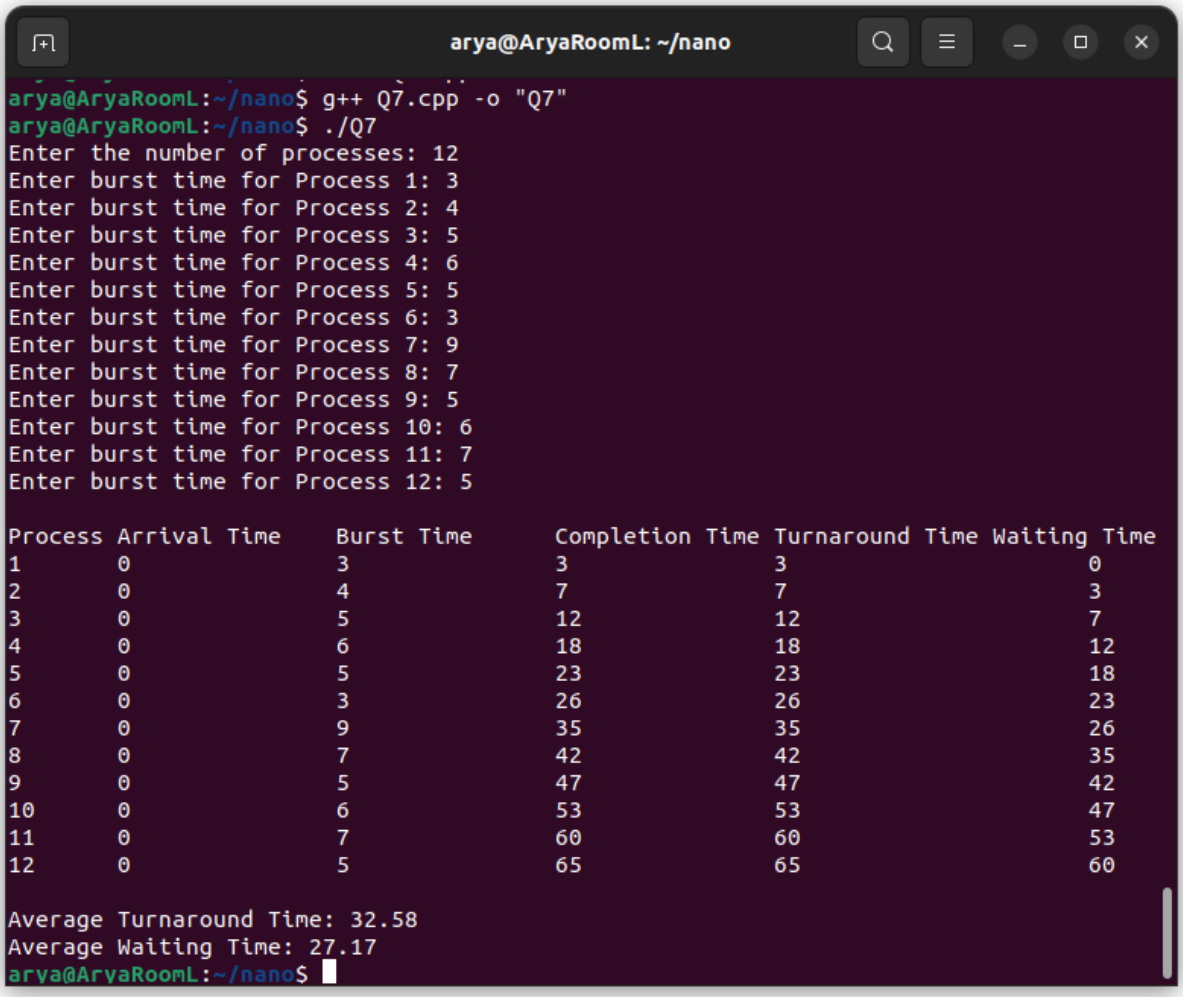
```
// Function to calculate completion, turnaround, and waiting times
void calculateTimes(struct Process processes[], int n) {
    int currentTime = 0;

    for (int i = 0; i < n; i++) {
        // Set completion time
        processes[i].completionTime = currentTime + processes[i].burstTime;

        // Set turnaround time
        processes[i].turnaroundTime = processes[i].completionTime -
processes[i].arrivalTime;

        // Set waiting time
        processes[i].waitingTime = processes[i].turnaroundTime -
processes[i].burstTime;

        // Update current time
        currentTime = processes[i].completionTime;
    }
}

// Function to display the process details and average times
void displayResults(struct Process processes[], int n) {
    float totalTurnaroundTime = 0, totalWaitingTime = 0;

    printf("\nProcess\tArrival Time\tBurst Time\tCompletion Time\tTurnaround
Time\tWaiting Time\n");

    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t\t%d\n",
               processes[i].processID, processes[i].arrivalTime,
               processes[i].burstTime, processes[i].completionTime,
               processes[i].turnaroundTime, processes[i].waitingTime);

        // Calculate total turnaround and waiting times for averages
        totalTurnaroundTime += processes[i].turnaroundTime;
        totalWaitingTime += processes[i].waitingTime;
    }

    // Display average turnaround and waiting times
    printf("\nAverage Turnaround Time: %.2f\n", totalTurnaroundTime / n);
    printf("Average Waiting Time: %.2f\n", totalWaitingTime / n);
}
```

Name : Atul_Arya
Roll No. : CSC/22/11

```
int main() {
    int n;

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    struct Process processes[n];

    // Input process details
    for (int i = 0; i < n; i++) {
        processes[i].processID = i + 1;
        processes[i].arrivalTime = 0; // Assume arrival time is 0 for simplicity
        printf("Enter burst time for Process %d: ", processes[i].processID);
        scanf("%d", &processes[i].burstTime);
    }

    // Calculate completion, turnaround, and waiting times
    calculateTimes(processes, n);

    // Display process details and average times
    displayResults(processes, n);

    return 0;
}
```

Output:

Name : Atul_Arya
Roll No. : CSC/22/11



Name : Atul_Arya
Roll No. : CSC/22/11

Question-8.

Write a program to implement SJF scheduling algorithm.

```
#include <stdio.h>

// Structure to represent a process
struct Process {
    int processID;
    int arrivalTime;
    int burstTime;
    int completionTime;
    int turnaroundTime;
    int waitingTime;
};

// Function to sort processes based on burst time
void sortProcesses(struct Process processes[], int n) {
    struct Process temp;

    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (processes[j].burstTime > processes[j + 1].burstTime) {
                // Swap processes
                temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }
}

// Function to calculate completion, turnaround, and waiting times
void calculateTimes(struct Process processes[], int n) {
    int currentTime = 0;

    for (int i = 0; i < n; i++) {
        // Set completion time
        processes[i].completionTime = currentTime + processes[i].burstTime;

        // Set turnaround time
        processes[i].turnaroundTime = processes[i].completionTime -
processes[i].arrivalTime;
```

Name : Atul_Arya
Roll No. : CSC/22/11

```
        // Set waiting time
        processes[i].waitingTime = processes[i].turnaroundTime -
processes[i].burstTime;

        // Update current time
        currentTime = processes[i].completionTime;
    }
}

// Function to display the process details and average times
void displayResults(struct Process processes[], int n) {
    float totalTurnaroundTime = 0, totalWaitingTime = 0;

    printf("\nProcess\tArrival Time\tBurst Time\tCompletion Time\tTurnaround
Time\tWaiting Time\n");

    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t\t%d\n",
               processes[i].processID, processes[i].arrivalTime,
               processes[i].burstTime, processes[i].completionTime,
               processes[i].turnaroundTime, processes[i].waitingTime);

        // Calculate total turnaround and waiting times for averages
        totalTurnaroundTime += processes[i].turnaroundTime;
        totalWaitingTime += processes[i].waitingTime;
    }

    // Display average turnaround and waiting times
    printf("\nAverage Turnaround Time: %.2f\n", totalTurnaroundTime / n);
    printf("Average Waiting Time: %.2f\n", totalWaitingTime / n);
}

int main() {
    int n;

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    struct Process processes[n];

    // Input process details
    for (int i = 0; i < n; i++) {
```

Name : Atul_Arya
Roll No. : CSC/22/11

```
        processes[i].processID = i + 1;
        processes[i].arrivalTime = 0; // Assume arrival time is 0 for simplicity
        printf("Enter burst time for Process %d: ", processes[i].processID);
        scanf("%d", &processes[i].burstTime);
    }

    // Sort processes based on burst time (SJF)
    sortProcesses(processes, n);

    // Calculate completion, turnaround, and waiting times
    calculateTimes(processes, n);

    // Display process details and average times
    displayResults(processes, n);

    return 0;
}
```

Output:

```
arya@AryaRoomL:~/nano$ nano Q8.cpp
arya@AryaRoomL:~/nano$ g++ Q7.cpp -o "Q7"
arya@AryaRoomL:~/nano$ g++ Q8.cpp -o "Q8"
arya@AryaRoomL:~/nano$ ./Q8
Enter the number of processes: 6
Enter burst time for Process 1: 4
Enter burst time for Process 2: 6
Enter burst time for Process 3: 32
Enter burst time for Process 4: 4
Enter burst time for Process 5: 9
Enter burst time for Process 6: 5

Process Arrival Time    Burst Time      Completion Time Turnaround Time Waiting Time
1       0               4               4               4                       0
4       0               4               8               8                       4
6       0               5               13              13                      8
2       0               6               19              19                      13
5       0               9               28              28                      19
3       0               32              60              60                      28

Average Turnaround Time: 22.00
Average Waiting Time: 12.00
arya@AryaRoomL:~/nano$
```

Name : Atul_Arya
Roll No. : CSC/22/11

Question-9.

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

```
#include <stdio.h>

// Structure to represent a process
struct Process {
    int processID;
    int priority;
    int burstTime;
    int completionTime;
    int turnaroundTime;
    int waitingTime;
};

// Function to sort processes based on priority (and process ID for tie-breaking)
void sortProcesses(struct Process processes[], int n) {
    struct Process temp;

    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (processes[j].priority < processes[j + 1].priority ||
                (processes[j].priority == processes[j + 1].priority &&
                 processes[j].processID > processes[j + 1].processID)) {
                // Swap processes
                temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }
}

// Function to calculate completion, turnaround, and waiting times
void calculateTimes(struct Process processes[], int n) {
    int currentTime = 0;

    for (int i = 0; i < n; i++) {
        // Set completion time
        processes[i].completionTime = currentTime + processes[i].burstTime;

        // Set turnaround time
```

Name : Atul_Arya
Roll No. : CSC/22/11

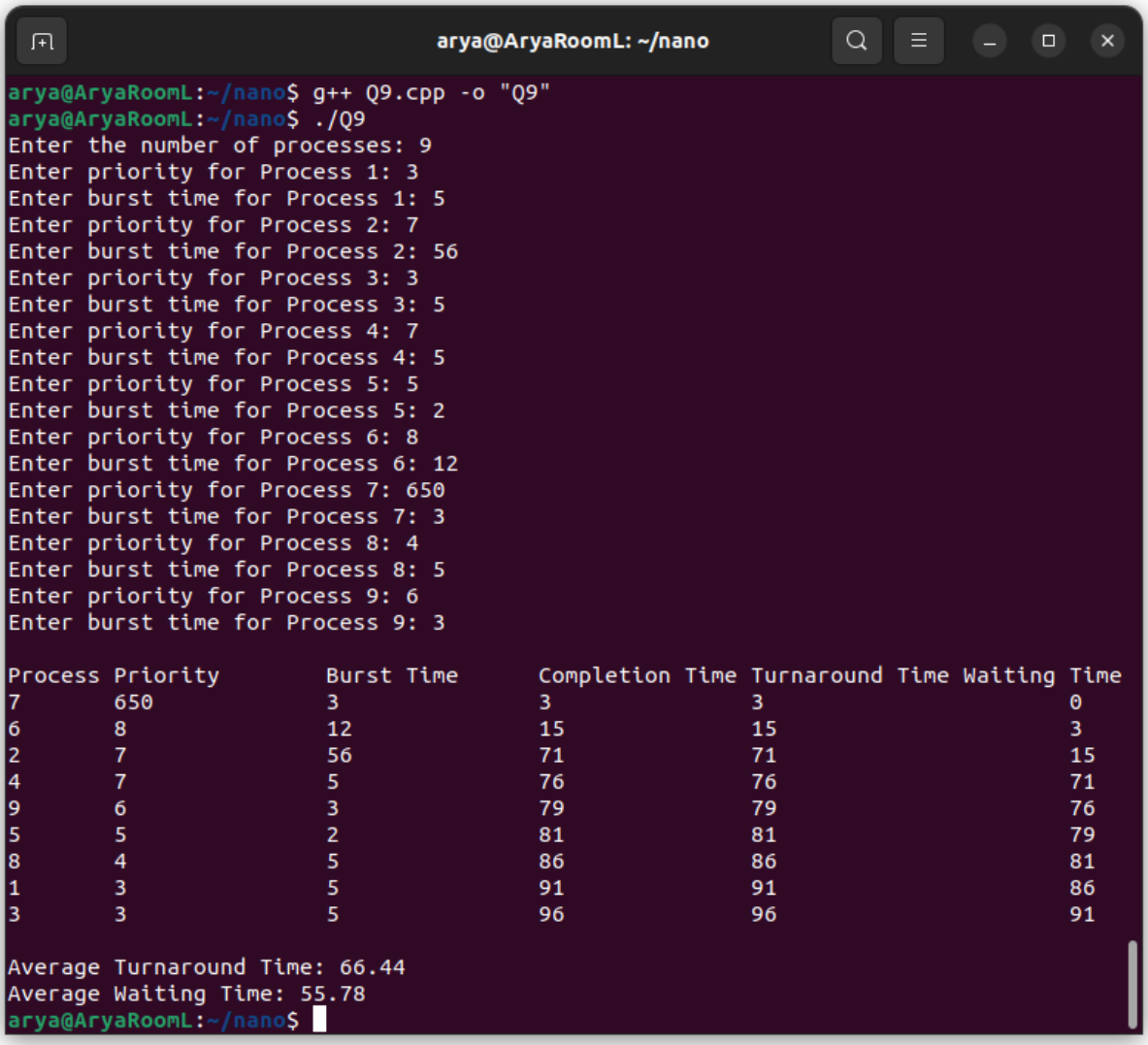
```
        processes[i].turnaroundTime = processes[i].completionTime;

        // Set waiting time
        processes[i].waitingTime = processes[i].turnaroundTime -
processes[i].burstTime;

        // Update current time
        currentTime = processes[i].completionTime;
    }
}

// Function to display the process details and average times
void displayResults(struct Process processes[], int n) {
    float totalTurnaroundTime = 0, totalWaitingTime = 0;

    printf("\nProcess\tPriority\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting
Time\n");

    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t\t%d\n",
               processes[i].processID, processes[i].priority,
               processes[i].burstTime, processes[i].completionTime,
               processes[i].turnaroundTime, processes[i].waitingTime);

        // Calculate total turnaround and waiting times for averages
        totalTurnaroundTime += processes[i].turnaroundTime;
        totalWaitingTime += processes[i].waitingTime;
    }

    // Display average turnaround and waiting times
    printf("\nAverage Turnaround Time: %.2f\n", totalTurnaroundTime / n);
    printf("Average Waiting Time: %.2f\n", totalWaitingTime / n);
}

int main() {
    int n;

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    struct Process processes[n];

    // Input process details
```

Name : Atul_Arya
Roll No. : CSC/22/11

```
    for (int i = 0; i < n; i++) {
        processes[i].processID = i + 1;
        printf("Enter priority for Process %d: ", processes[i].processID);
        scanf("%d", &processes[i].priority);
        printf("Enter burst time for Process %d: ", processes[i].processID);
        scanf("%d", &processes[i].burstTime);
    }

    // Sort processes based on priority (and process ID for tie-breaking)
    sortProcesses(processes, n);

    // Calculate completion, turnaround, and waiting times
    calculateTimes(processes, n);

    // Display process details and average times
    displayResults(processes, n);

    return 0;
}
```

Output:

Name : Atul_Arya
Roll No. : CSC/22/11



Name : Atul_Arya
Roll No. : CSC/22/11

Question-10.

Write a program to implement SRTF scheduling algorithm.

```
#include <stdio.h>
#include <limits.h>

// Structure to represent a process
struct Process {
    int processID;
    int arrivalTime;
    int burstTime;
    int remainingTime;
    int completionTime;
    int turnaroundTime;
    int waitingTime;
};

// Function to find the process with the shortest remaining time
int findShortestRemainingTime(struct Process processes[], int n, int currentTime) {
    int shortest = INT_MAX;
    int shortestIndex = -1;

    for (int i = 0; i < n; i++) {
        if (processes[i].arrivalTime <= currentTime && processes[i].remainingTime <
shortest && processes[i].remainingTime > 0) {
            shortest = processes[i].remainingTime;
            shortestIndex = i;
        }
    }

    return shortestIndex;
}

// Function to calculate completion, turnaround, and waiting times
void calculateTimes(struct Process processes[], int n) {
    int currentTime = 0;
    int remainingProcesses = n;

    while (remainingProcesses > 0) {
        int shortestIndex = findShortestRemainingTime(processes, n, currentTime);

        if (shortestIndex == -1) {
```

Name : Atul_Arya
Roll No. : CSC/22/11

```
            currentTime++;
        } else {
            // Update remaining time for the selected process
            processes[shortestIndex].remainingTime--;

            // If the process is completed
            if (processes[shortestIndex].remainingTime == 0) {
                remainingProcesses--;

                // Set completion time
                processes[shortestIndex].completionTime = currentTime + 1;

                // Set turnaround time
                processes[shortestIndex].turnaroundTime =
processes[shortestIndex].completionTime - processes[shortestIndex].arrivalTime;

                // Set waiting time
                processes[shortestIndex].waitingTime =
processes[shortestIndex].turnaroundTime - processes[shortestIndex].burstTime;
            }

            currentTime++;
        }
    }
}

// Function to display the process details and average times
void displayResults(struct Process processes[], int n) {
    float totalTurnaroundTime = 0, totalWaitingTime = 0;

    printf("\nProcess\tArrival Time\tBurst Time\tCompletion Time\tTurnaround
Time\tWaiting Time\n");

    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t\t%d\n",
               processes[i].processID, processes[i].arrivalTime,
               processes[i].burstTime, processes[i].completionTime,
               processes[i].turnaroundTime, processes[i].waitingTime);

        // Calculate total turnaround and waiting times for averages
        totalTurnaroundTime += processes[i].turnaroundTime;
        totalWaitingTime += processes[i].waitingTime;
    }
```

Name : Atul_Arya
Roll No. : CSC/22/11

```
    // Display average turnaround and waiting times
    printf("\nAverage Turnaround Time: %.2f\n", totalTurnaroundTime / n);
    printf("Average Waiting Time: %.2f\n", totalWaitingTime / n);
}

int main() {
    int n;

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    struct Process processes[n];

    // Input process details
    for (int i = 0; i < n; i++) {
        processes[i].processID = i + 1;
        printf("Enter arrival time for Process %d: ", processes[i].processID);
        scanf("%d", &processes[i].arrivalTime);
        printf("Enter burst time for Process %d: ", processes[i].processID);
        scanf("%d", &processes[i].burstTime);
        processes[i].remainingTime = processes[i].burstTime;
    }

    // Calculate completion, turnaround, and waiting times
    calculateTimes(processes, n);

    displayResults(processes, n);

    return 0;
}
```

Output:

Name : Atul_Arya
Roll No. : CSC/22/11

```
arya@AryaRoomL:~/nano$ nano Q10.cpp
arya@AryaRoomL:~/nano$ g++ Q10.cpp -o "Q10"
arya@AryaRoomL:~/nano$ ./Q10
Enter the number of processes: 4
Enter arrival time for Process 1: 0
Enter burst time for Process 1: 3
Enter arrival time for Process 2: 0
Enter burst time for Process 2: 4
Enter arrival time for Process 3: 2
Enter burst time for Process 3: 23
Enter arrival time for Process 4: 4
Enter burst time for Process 4: 43

Process Arrival Time    Burst Time      Completion Time Turnaround Time Waiting Time
1       0               3               3               3                       0
2       0               4               7               7                       3
3       2               23              30              28                      5
4       4               43              73              69                      26

Average Turnaround Time: 26.75
Average Waiting Time: 8.50
arya@AryaRoomL:~/nano$
```

Name : Atul_Arya
Roll No. : CSC/22/11

Question-11.

Write a program to calculate the 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.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define MAX_NUMBERS 1000

// Structure to pass arguments to the thread function
struct ThreadArgs {
    int* numbers;
    int start;
    int end;
};

// Function to calculate the sum of a sublist
void* calculateSum(void* args) {
    struct ThreadArgs* threadArgs = (struct ThreadArgs*)args;
    int* numbers = threadArgs->numbers;
    int start = threadArgs->start;
    int end = threadArgs->end;

    int sum = 0;
    for (int i = start; i < end; i++) {
        sum += numbers[i];
    }

    // Allocate memory to store the result
    int* result = (int*)malloc(sizeof(int));
    *result = sum;

    pthread_exit(result);
}

int main() {
    int n;
```

Name : Atul_Arya
Roll No. : CSC/22/11

```
    printf("Enter the number of elements (n): ");
    scanf("%d", &n);

    if (n <= 0 || n > MAX_NUMBERS) {
        printf("Invalid number of elements. Please enter a value between 1 and %d.\n",
MAX_NUMBERS);
        return 1;
    }

    int numbers[MAX_NUMBERS];

    printf("Enter %d numbers:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &numbers[i]);
    }

    // Create two threads
    pthread_t thread1, thread2;

    // Divide the array into two halves
    int mid = n / 2;

    // Arguments for the first thread
    struct ThreadArgs args1 = {numbers, 0, mid};

    // Arguments for the second thread
    struct ThreadArgs args2 = {numbers, mid, n};

    // Variables to store thread results
    int* result1;
    int* result2;

    // Create the first thread
    if (pthread_create(&thread1, NULL, calculateSum, (void*)&args1) != 0) {
        fprintf(stderr, "Error creating thread 1.\n");
        return 1;
    }

    // Create the second thread
    if (pthread_create(&thread2, NULL, calculateSum, (void*)&args2) != 0) {
        fprintf(stderr, "Error creating thread 2.\n");
        return 1;
    }
```

Name : Atul_Arya
Roll No. : CSC/22/11

```
    // Wait for the first thread to finish
    if (pthread_join(thread1, (void**)&result1) != 0) {
        fprintf(stderr, "Error joining thread 1.\n");
        return 1;
    }

    // Wait for the second thread to finish
    if (pthread_join(thread2, (void**)&result2) != 0) {
        fprintf(stderr, "Error joining thread 2.\n");
        return 1;
    }

    // Calculate the final sum
    int finalSum = *result1 + *result2;

    // Display the result
    printf("Sum of the numbers: %d\n", finalSum);

    // Free allocated memory
    free(result1);
    free(result2);

    return 0;
}
```

Output:

```
arya@AryaRoomL:~/nano$ g++ Q11.cpp -o "Q11"
arya@AryaRoomL:~/nano$ ./Q11
Enter the number of elements (n): 6
Enter 6 numbers:
5
66
34
5
3
54
Sum of the numbers: 167
arya@AryaRoomL:~/nano$
```

Name : Atul_Arya
Roll No. : CSC/22/11

Question-12.

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

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

// Structure to represent a memory block
struct MemoryBlock {
    int processID;
    int size;
    int allocated;
};

// Function to display the memory status
void displayMemory(struct MemoryBlock memory[], int numBlocks) {
    printf("\nMemory Status:\n");
    printf("Block\tProcess ID\tSize\tAllocated\n");

    for (int i = 0; i < numBlocks; i++) {
        printf("%d\t", i + 1);
        if (memory[i].allocated) {
            printf("%d\t\t%d\tYes\n", memory[i].processID, memory[i].size);
        } else {
            printf("-\t\t%d\tNo\n", memory[i].size);
        }
    }
}

// Function to allocate memory using First-Fit strategy
void firstFit(struct MemoryBlock memory[], int numBlocks, int processID, int size) {
    for (int i = 0; i < numBlocks; i++) {
        if (!memory[i].allocated && memory[i].size >= size) {
            memory[i].allocated = 1;
            memory[i].processID = processID;
            break;
        }
    }
}

// Function to allocate memory using Best-Fit strategy
void bestFit(struct MemoryBlock memory[], int numBlocks, int processID, int size) {
```

Name : Atul_Arya
Roll No. : CSC/22/11

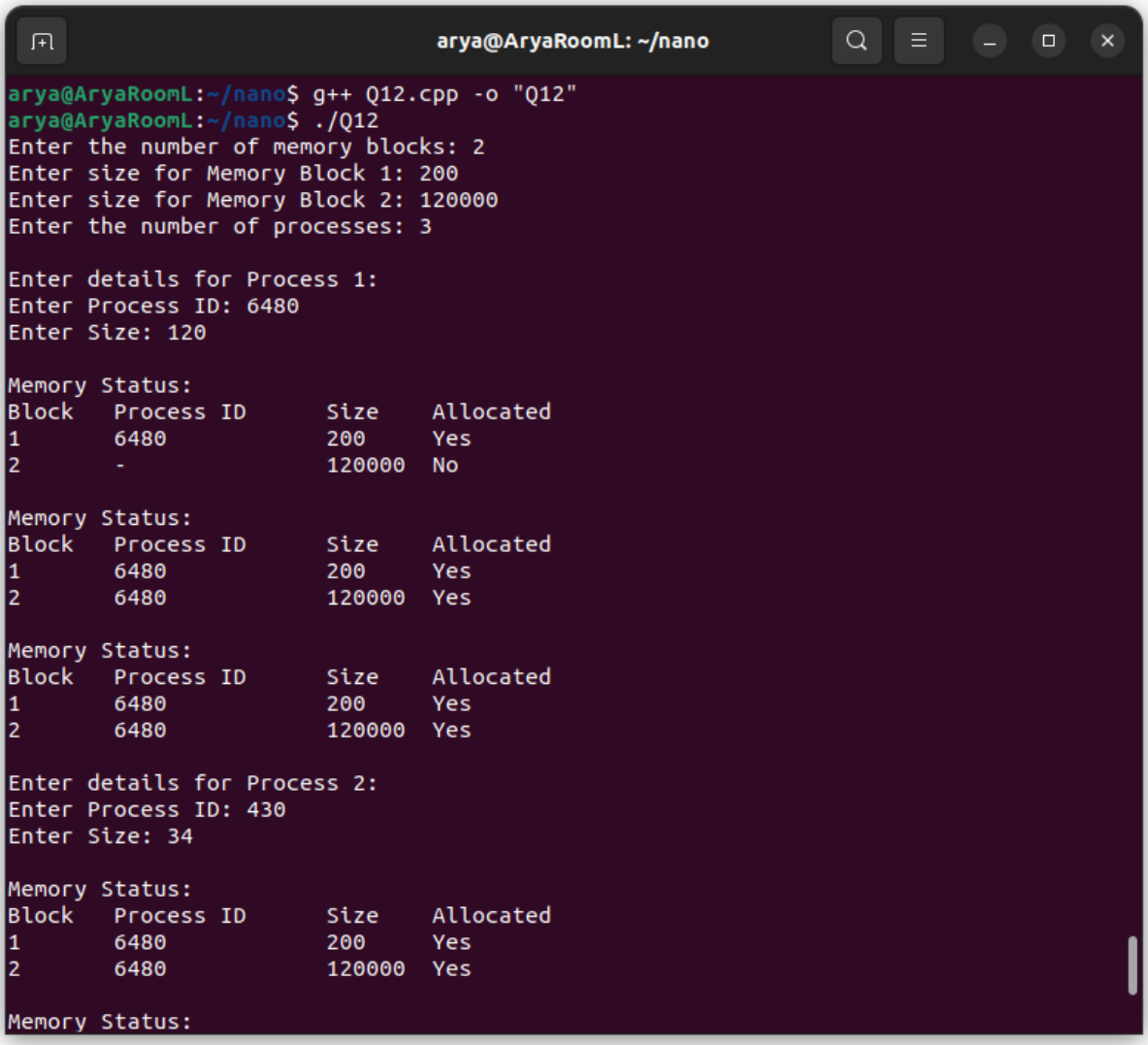
```
    int bestFitIndex = -1;
    int bestFitSize = INT_MAX;

    for (int i = 0; i < numBlocks; i++) {
        if (!memory[i].allocated && memory[i].size >= size) {
            if (memory[i].size < bestFitSize) {
                bestFitSize = memory[i].size;
                bestFitIndex = i;
            }
        }
    }

    if (bestFitIndex != -1) {
        memory[bestFitIndex].allocated = 1;
        memory[bestFitIndex].processID = processID;
    }
}

// Function to allocate memory using Worst-Fit strategy
void worstFit(struct MemoryBlock memory[], int numBlocks, int processID, int size) {
    int worstFitIndex = -1;
    int worstFitSize = -1;

    for (int i = 0; i < numBlocks; i++) {
        if (!memory[i].allocated && memory[i].size >= size) {
            if (memory[i].size > worstFitSize) {
                worstFitSize = memory[i].size;
                worstFitIndex = i;
            }
        }
    }

    if (worstFitIndex != -1) {
        memory[worstFitIndex].allocated = 1;
        memory[worstFitIndex].processID = processID;
    }
}

int main() {
    int numBlocks;

    printf("Enter the number of memory blocks: ");
    scanf("%d", &numBlocks);
```

Name : Atul_Arya
Roll No. : CSC/22/11

```
    struct MemoryBlock memory[numBlocks];

    // Initialize memory blocks
    for (int i = 0; i < numBlocks; i++) {
        memory[i].processID = -1;
        memory[i].allocated = 0;

        printf("Enter size for Memory Block %d: ", i + 1);
        scanf("%d", &memory[i].size);
    }

    int numProcesses;

    printf("Enter the number of processes: ");
    scanf("%d", &numProcesses);

    for (int i = 0; i < numProcesses; i++) {
        int processID, size;
        printf("\nEnter details for Process %d:\n", i + 1);
        printf("Enter Process ID: ");
        scanf("%d", &processID);
        printf("Enter Size: ");
        scanf("%d", &size);

        // First-Fit
        firstFit(memory, numBlocks, processID, size);
        displayMemory(memory, numBlocks);

        // Best-Fit
        bestFit(memory, numBlocks, processID, size);
        displayMemory(memory, numBlocks);

        // Worst-Fit
        worstFit(memory, numBlocks, processID, size);
        displayMemory(memory, numBlocks);
    }

    return 0;
}
```

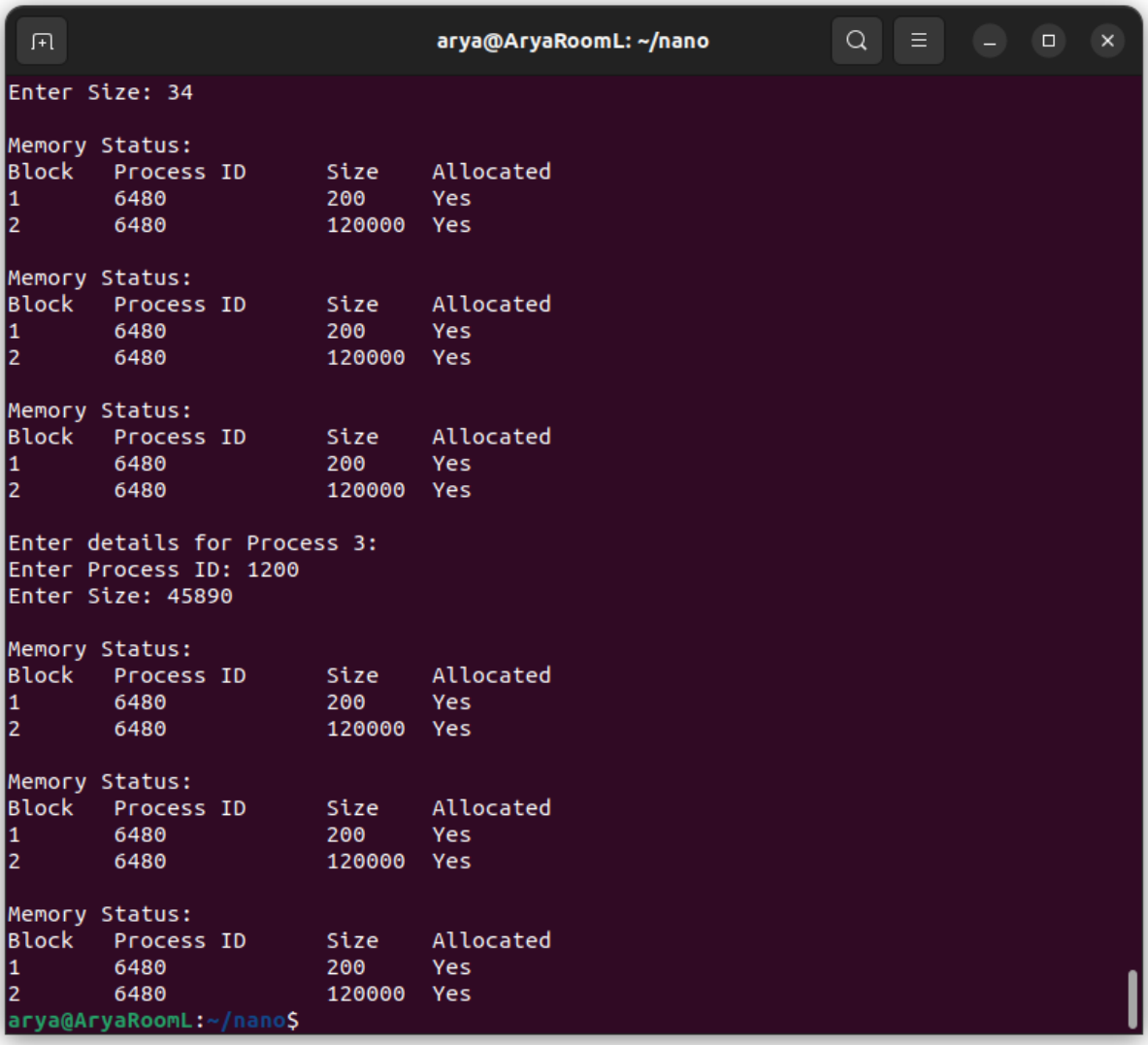
Output:

Name : Atul_Arya
Roll No. : CSC/22/11



.
.
.

Name : Atul_Arya
Roll No. : CSC/22/11



**********Finish**********