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

****

**Operating Systems Lab**

**(ITPC - 224)**

***Submitted to Submitted By***

Dr. Jitendra Kumar Samriya Aditya Anand

Department of IT 20124009

IT (G1)

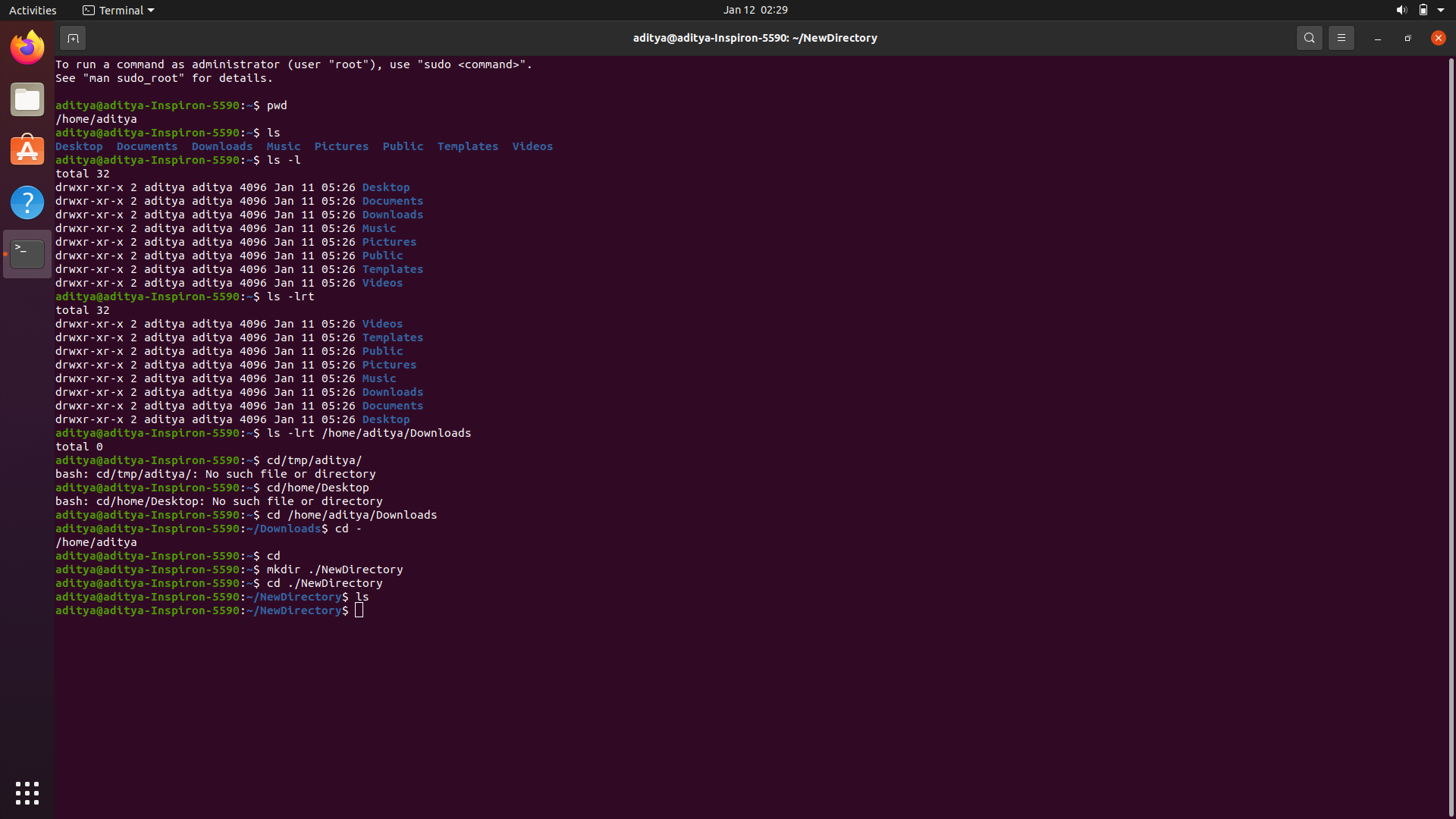
CONTENTS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No.** | **Name of Practical** | **Date of Implementation** | **Page** | **Remarks** |
| 1. | Basic commands in Linux | 06/01/2022 | 4 |  |
| 2. | Round Robin Process  First Come First Serve - FCFS | 17/01/2022 | 6 |  |
| 3. | Shortest Job First – SJF  First Come First Serve – FCFS  Priority Based Scheduling | 02/02/2022 | 8 |  |
| 4. | Critical Section Problem  Using Semaphores | 09/02/2022 | 18 |  |
| 5. | Producer Consumer Problem | 16/02/2022 | 22 |  |
| 6. | Banker’s Deadlock Avoidance Algorithm | 23/02/2022 | 24 |  |
| 7. | Process Synchronization:  Reader – Writer  Sleeping Barber | 02/03/2022 | 30 |  |
| 8. | Dining Philosopher Problem for Process Synchronization | 28/03/2022 | 37 |  |
| 9. | Page Replacement Algorithms –   1. FIFO 2. LRU 3. LFU   Simulation of Paging Technique | 26/04/2022 | 45 |  |

OS LAB 1

**BASIC LINUX COMMANDS**

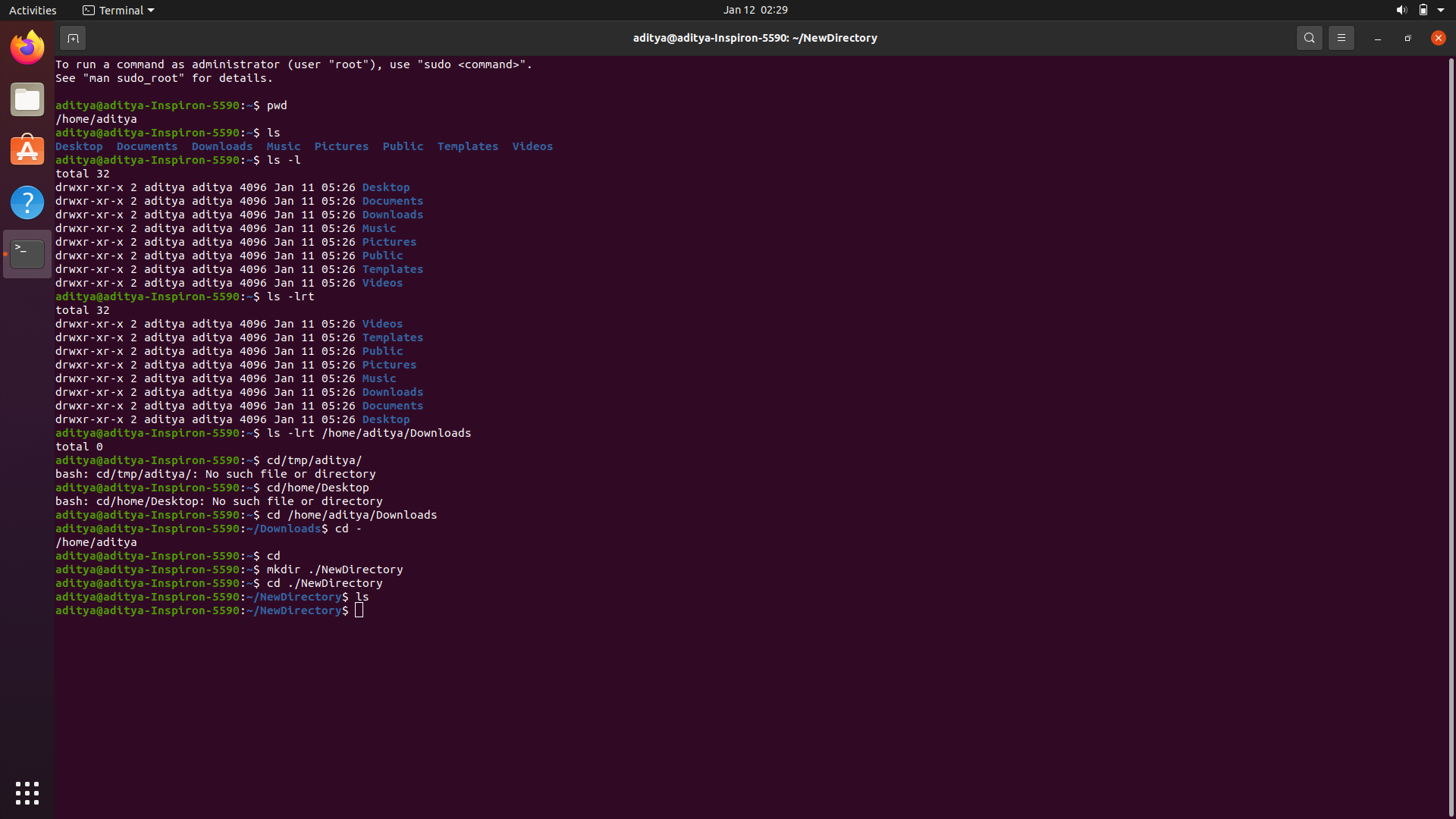
1. pwd : prints the name of the current working directory



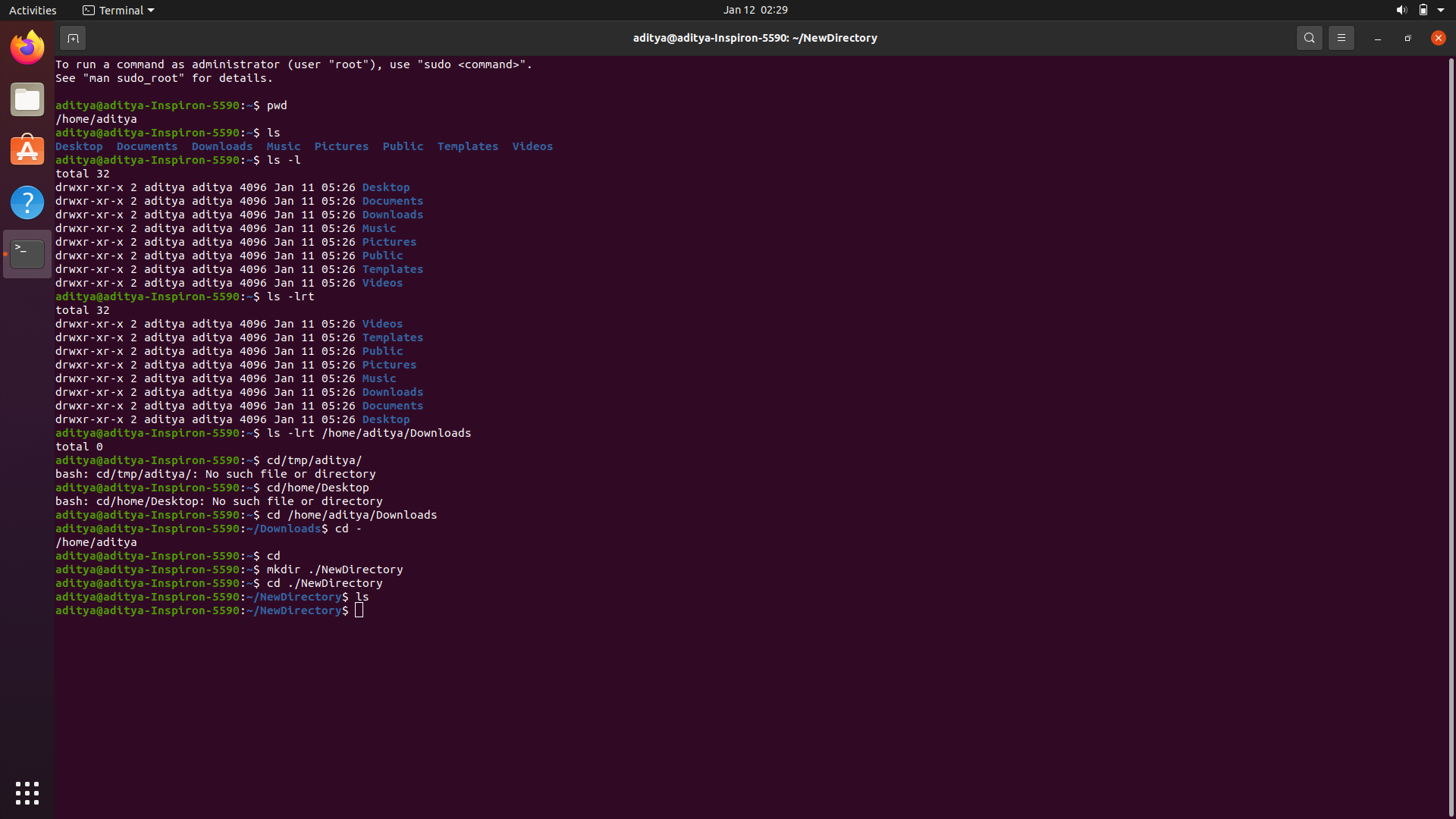
1. ls : list contents of the current working directory

ls -lrt : long listing

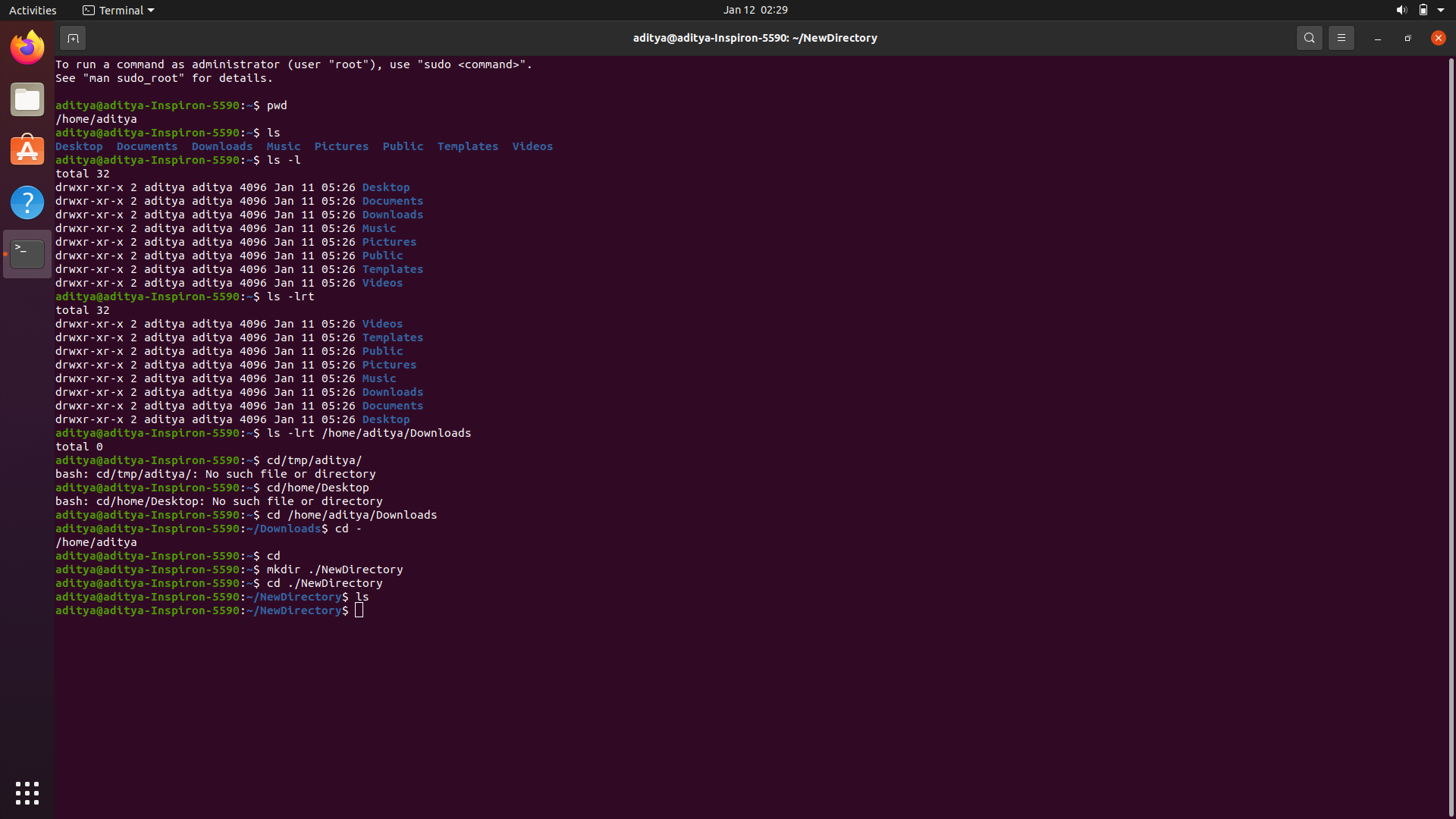
ls -lrt /home/name/something : long list a different directory

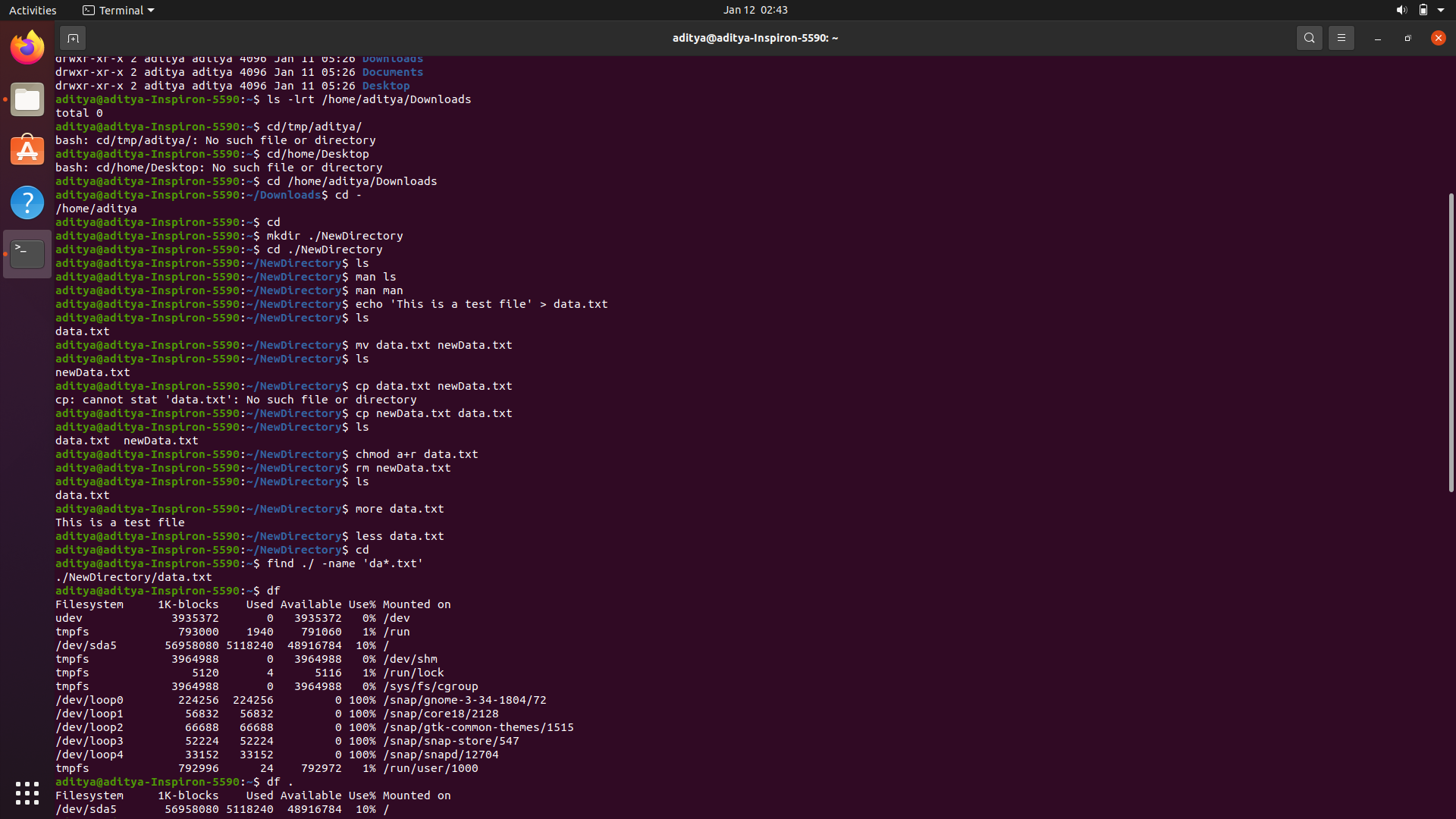


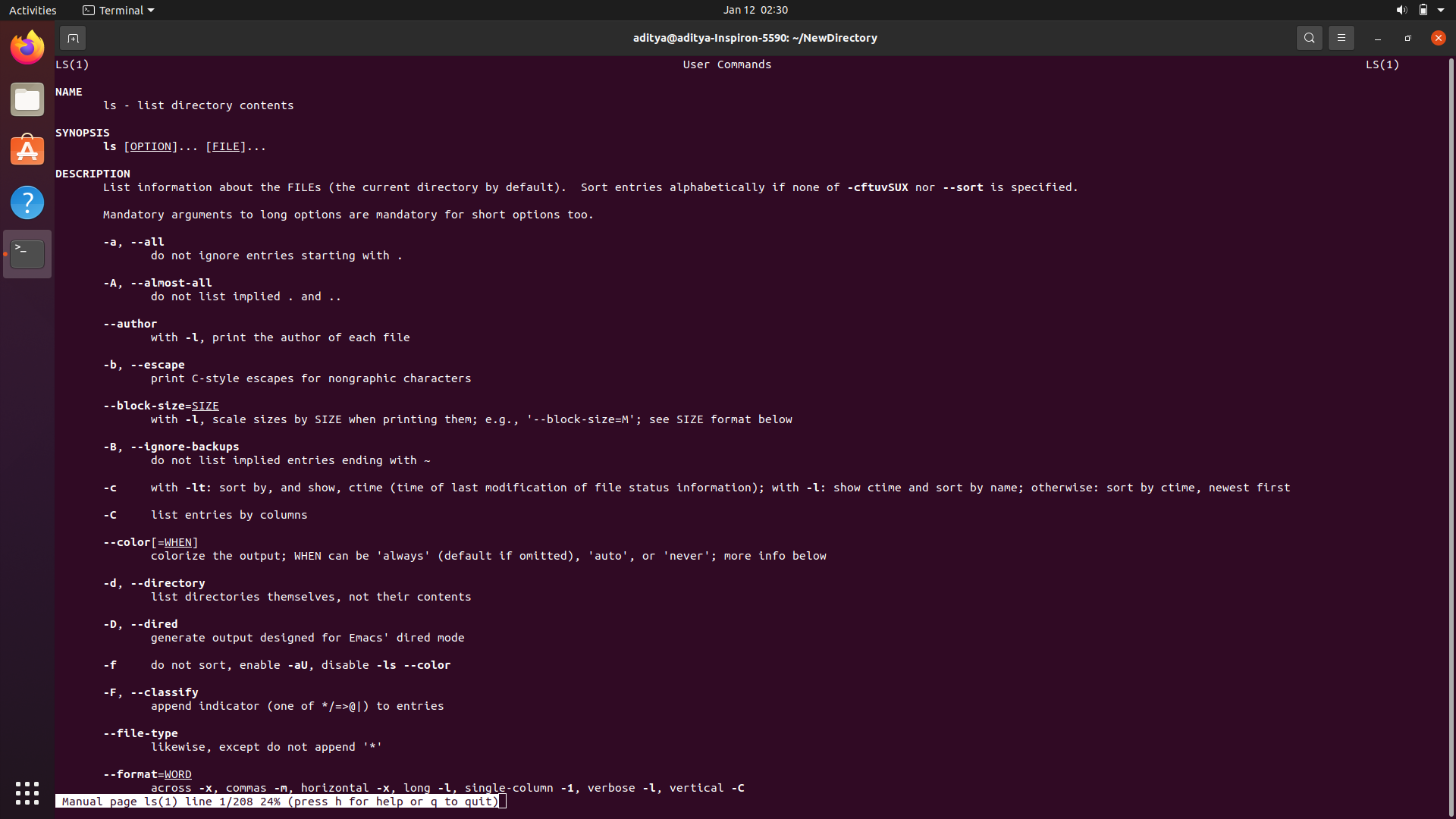
1. cd : change directory

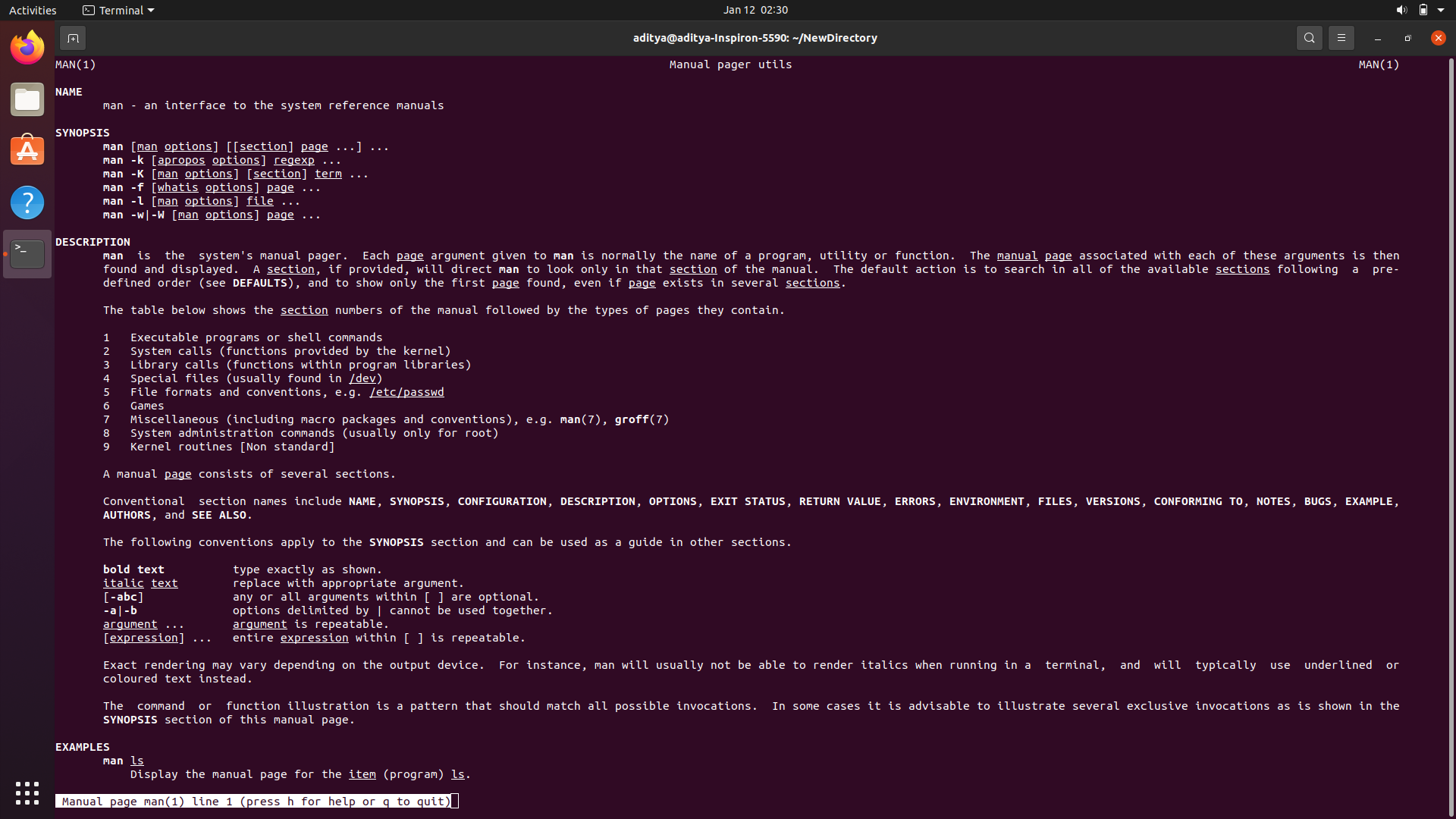


1. mkdir : create a new directory



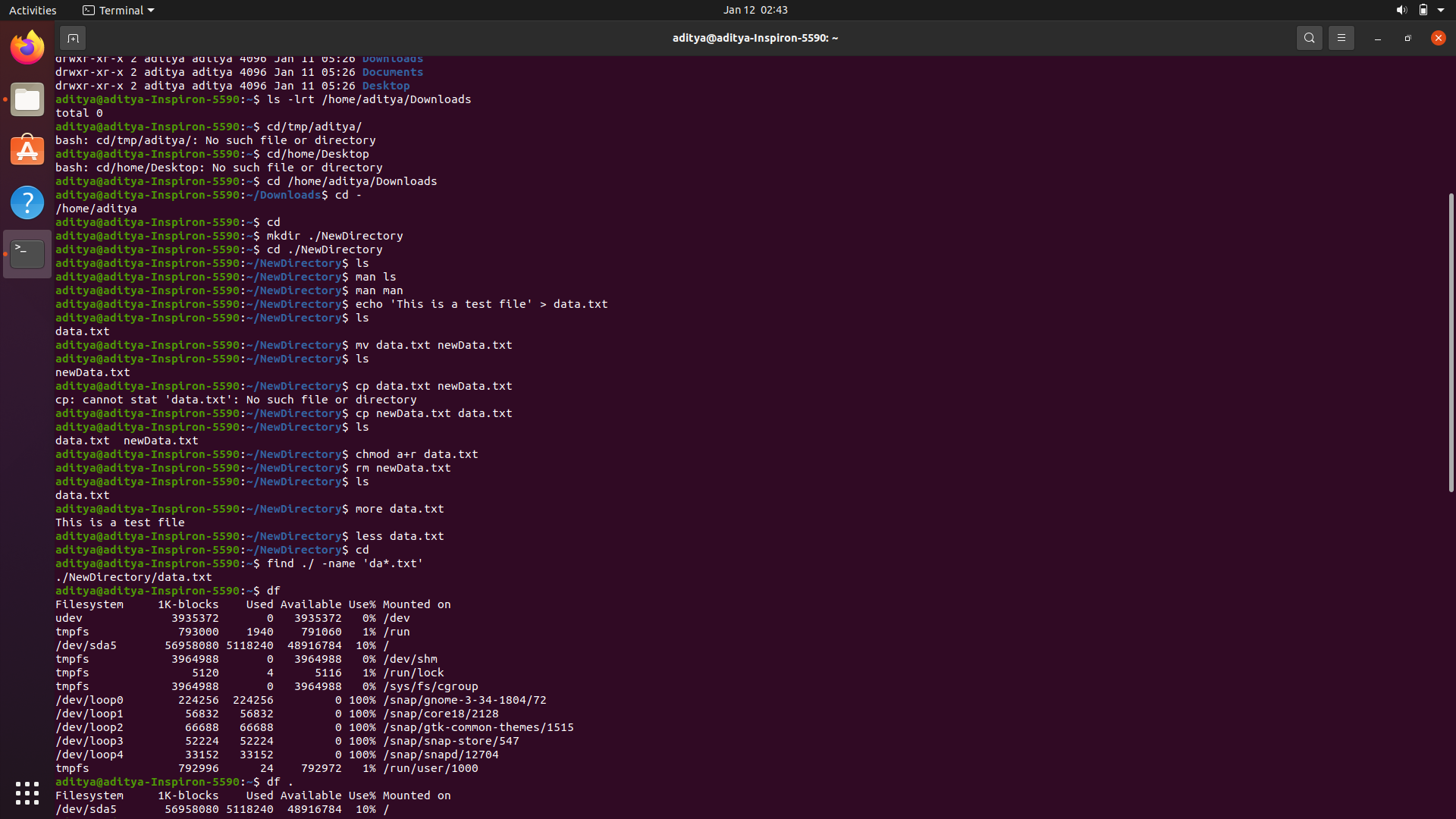
1. man : display the manual of a given program



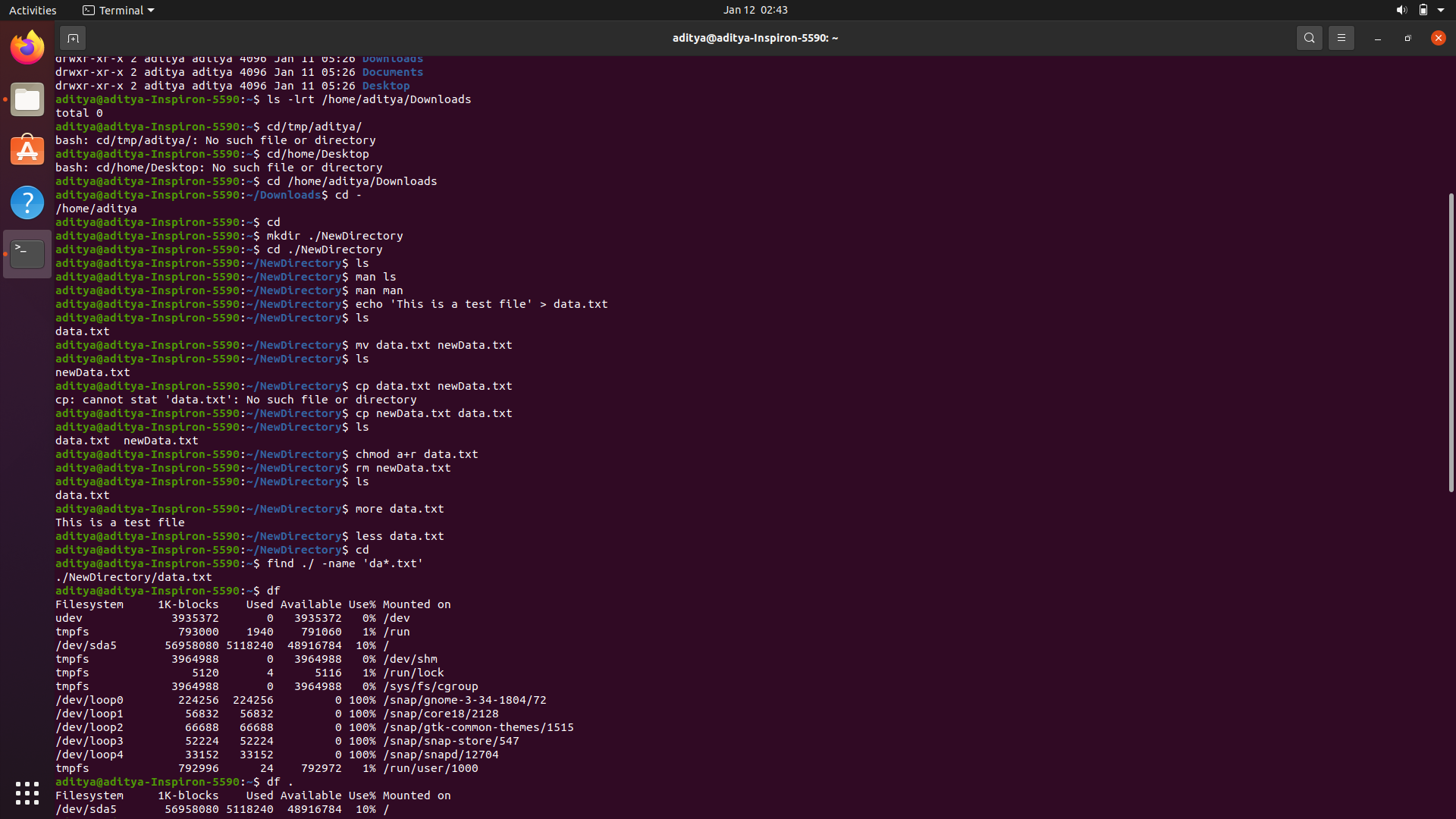


1. echo : to create a file

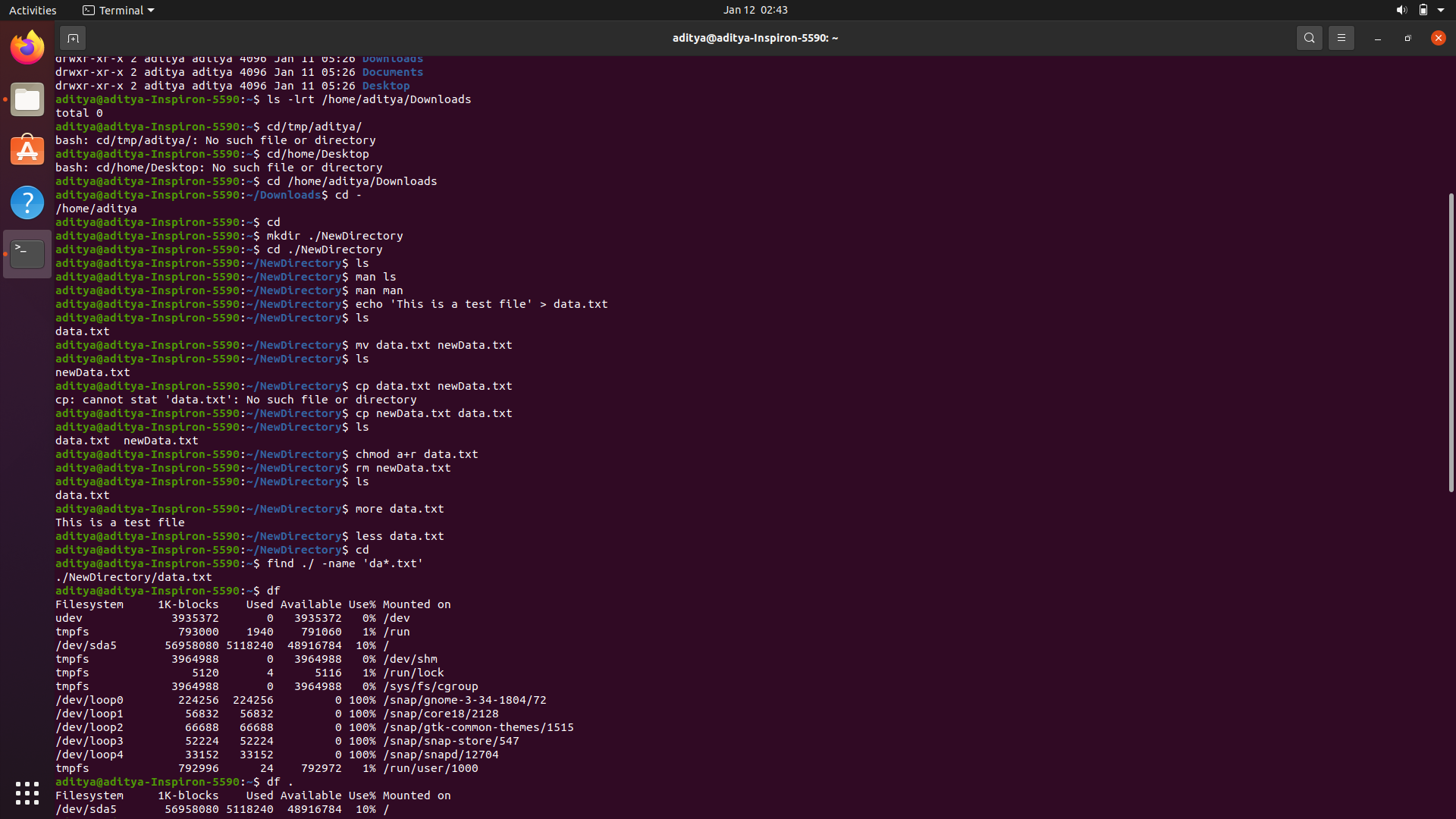
SYNTAX: echo ‘Text to be added’ > filename



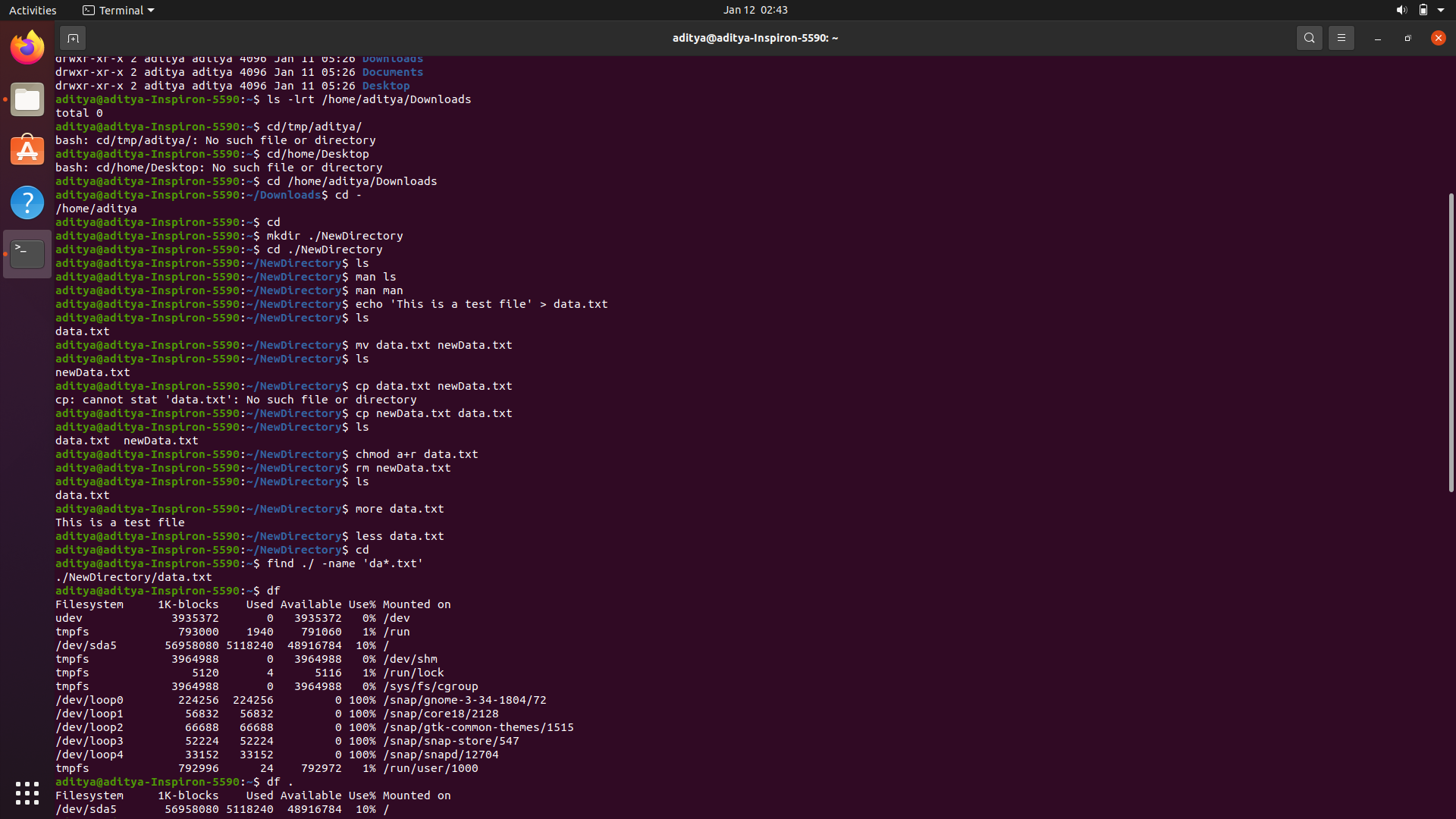
1. mv : move or rename a file



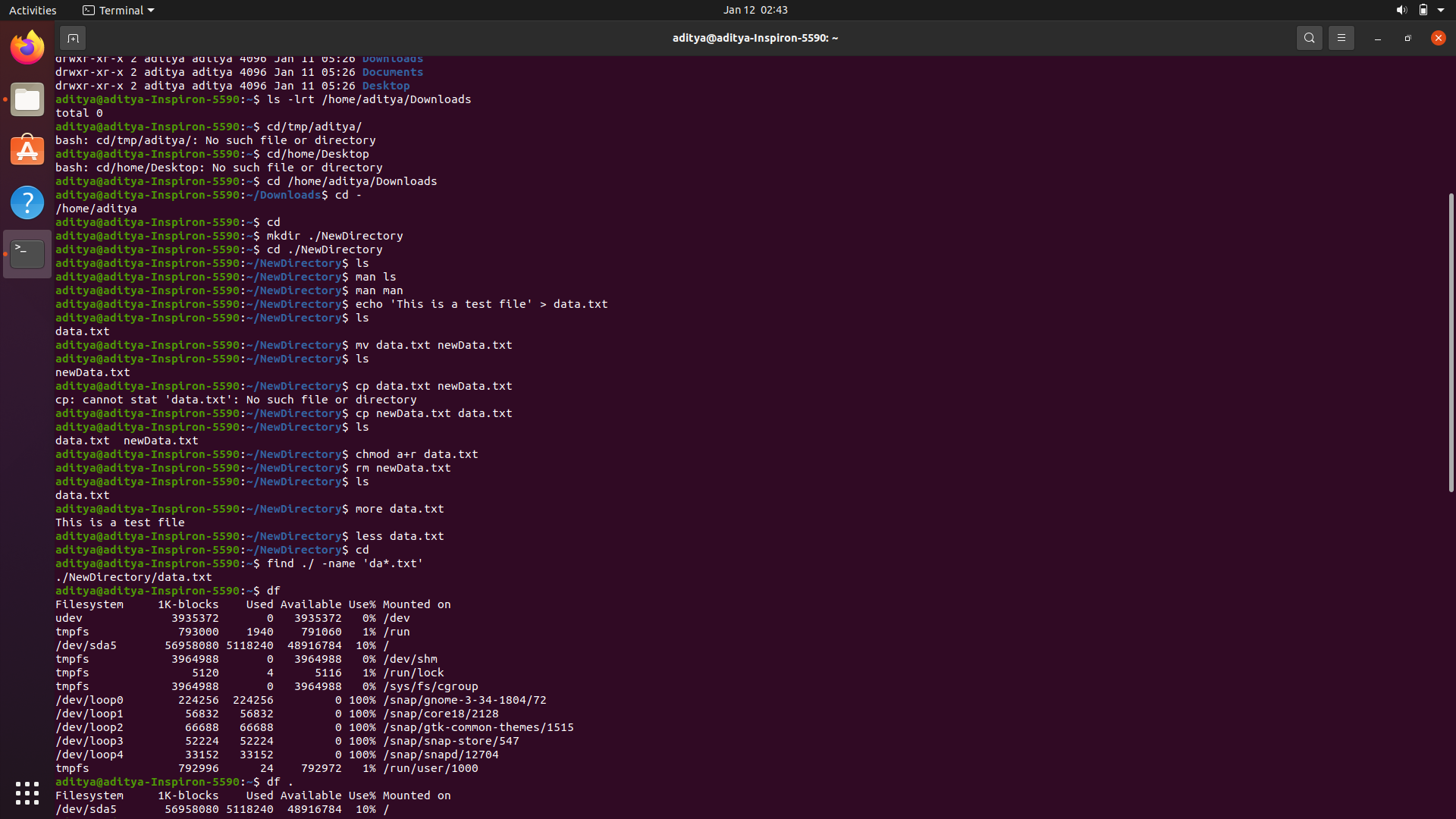
1. cp : copy a file



1. chmod : change the permissions of a file

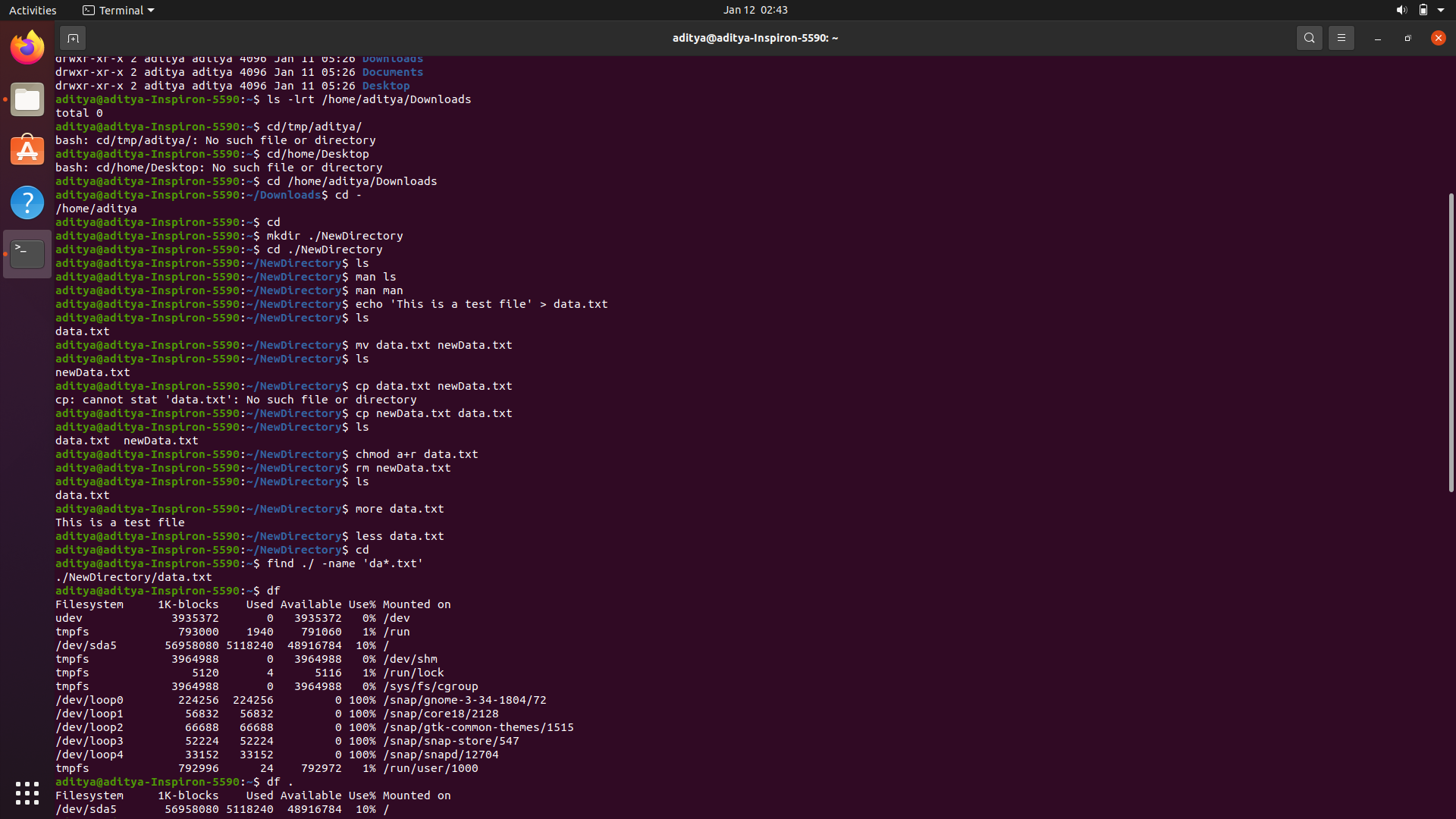


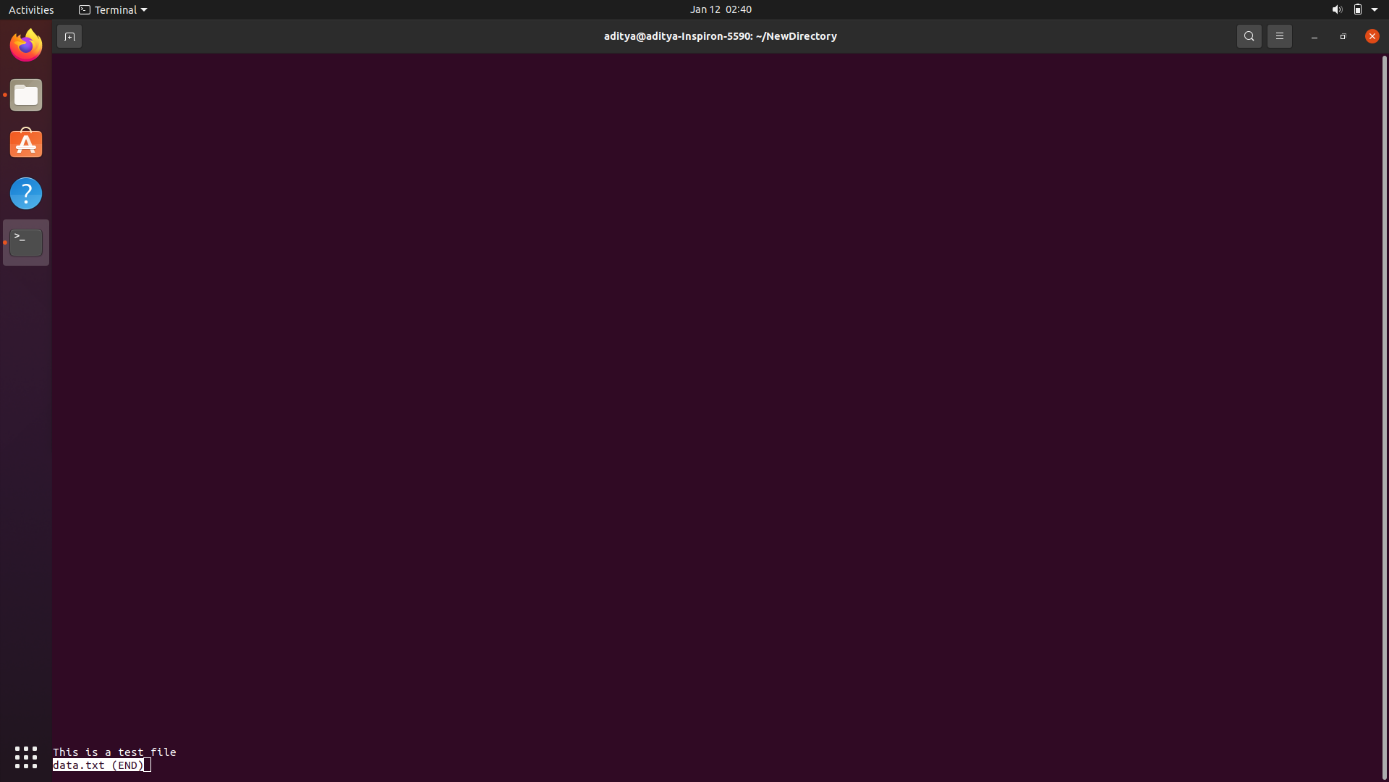
1. rm : delete a file forever



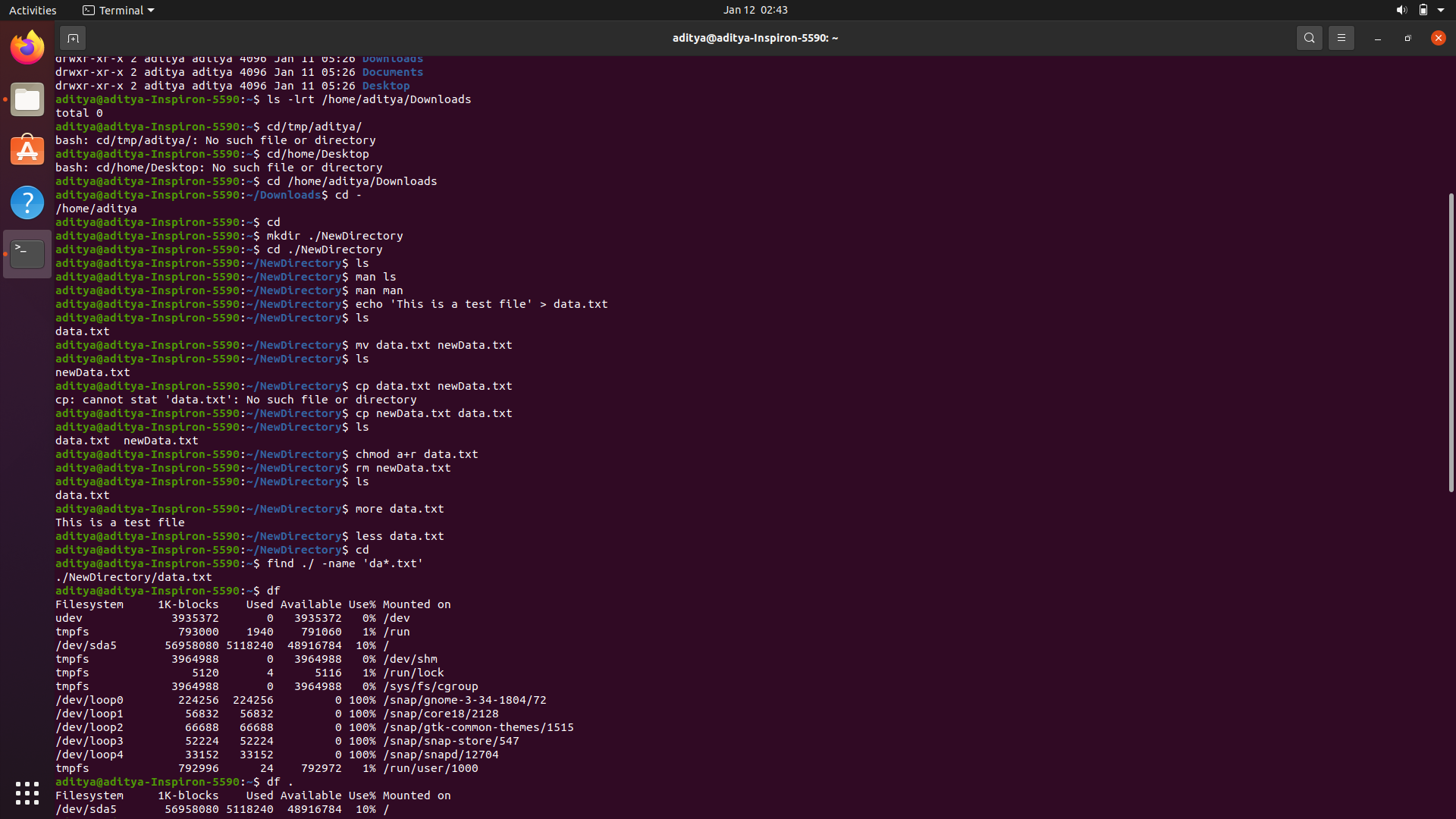
1. more : displays the contents of a file page by page

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

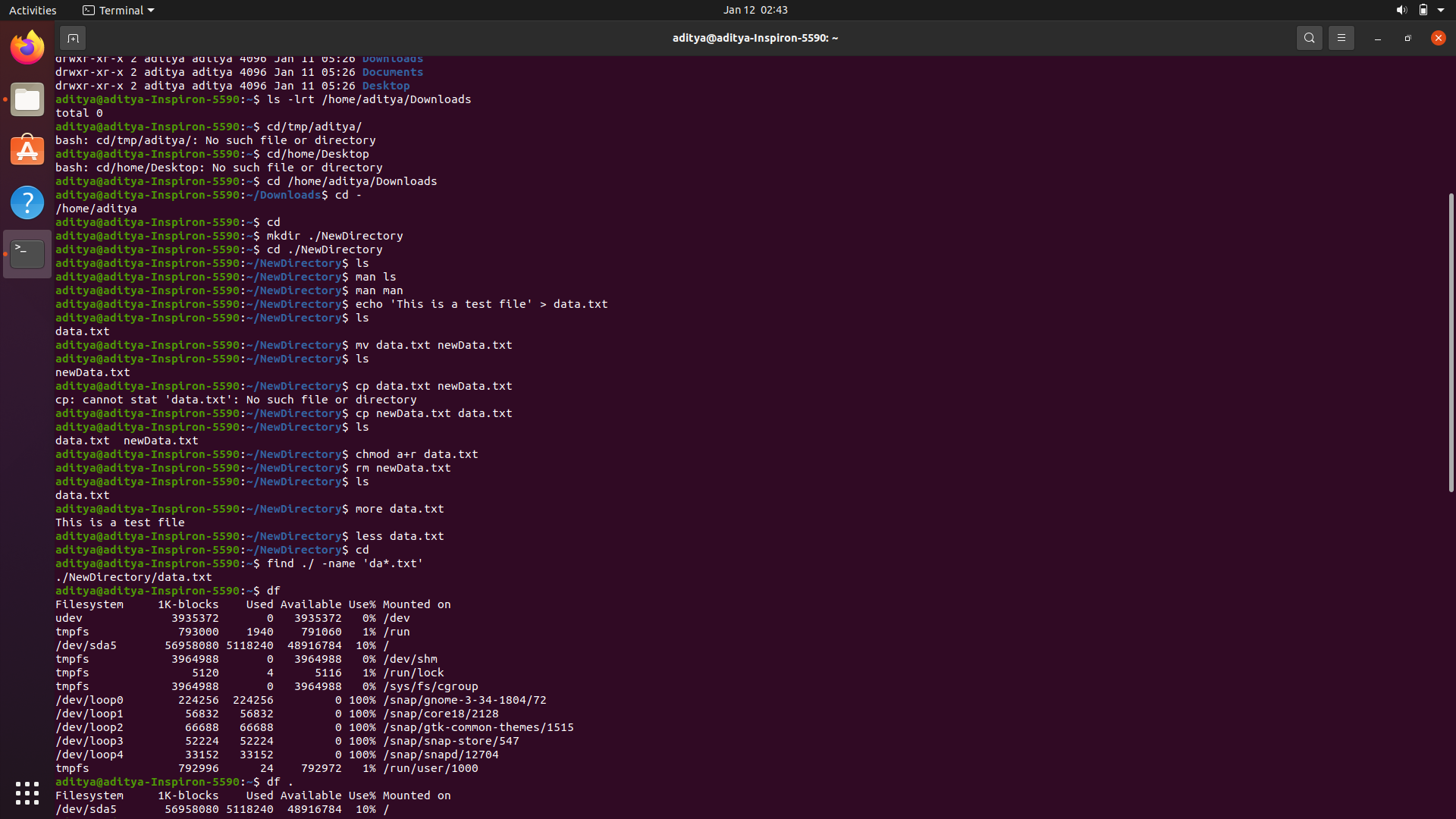




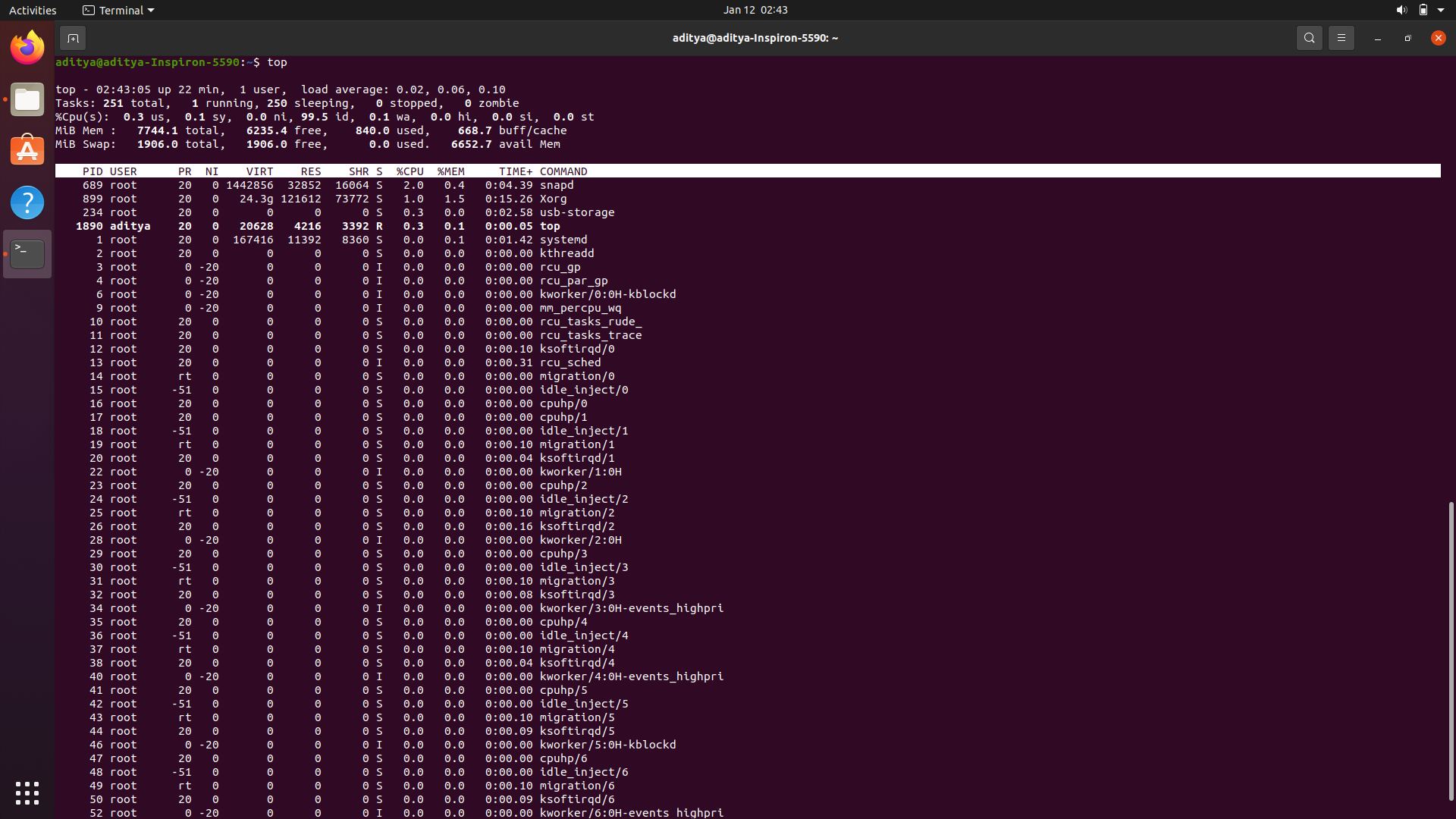
1. find : search for files across directories



1. df : check how much space is left



1. top : list processes in order of CPU storage



OS LAB 2

|  |  |  |  |
| --- | --- | --- | --- |
| S No. | Title | Date of Implementation | Remarks |
| 1 | Program to implement Round Robin Process of CPU Scheduling | 17-01-2022 |  |
| 2 | Program to implement First Come First Serve Process of CPU Scheduling | 24-01-2022 |  |

ROUND ROBIN CPU SCHEDULING

CRITERIA: Time Quantum

MODE: Pre-emptive

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

CODE:

#include<bits/stdc++.h>

using namespace :: std;

class process{

    public:

        int id;

        int arrivalTime;

        int burstTime;

        int remainingBurstTime;

        int completionTime;

        int TAT;

        int WT;

        int RT;

        bool isRunning;

};

void round\_robin\_scheduling(vector<process> &v, int quantum){

    queue<process> q;

    stack<process> s;

    int cur\_time = 0;

    int id=0;

    while(true){

        // Add all the processes that have arrived to the ready queue.

        while(id<v.size() && v[id].arrivalTime<=cur\_time){

            q.push(v[id]);

            id++;

        }

        // Add the last process from running queue at the end of ready queue if it is not completed.

        if(!s.empty()){

            process p = s.top();

            if(p.remainingBurstTime>0){

                q.push(p);

            }

        }

        // If ready queue is empty => no process to be completed => stop.

        if(q.empty()){

            break;

        }

        // Pick the front process from the ready queue for processing by the CPU.

        process rning\_proc = q.front();

        q.pop();

        // Store the Response time for a process the first time it reaches the CPU.

        if(rning\_proc.isRunning==false){

            v[rning\_proc.id].RT = cur\_time-v[rning\_proc.id].arrivalTime;

            rning\_proc.isRunning=true;

        }

        // If the remaining burst time > quantum, the process is not complete (CONTEXT SWITCHING)

        if(rning\_proc.remainingBurstTime>=quantum){

            rning\_proc.remainingBurstTime-=quantum;

            cur\_time+=quantum;

        }

        // Process is complete

        else{

            cur\_time+=rning\_proc.remainingBurstTime;

            rning\_proc.remainingBurstTime=0;

        }

        s.push(rning\_proc);

        // If process is complete, store the completeion time for the process

        if(rning\_proc.remainingBurstTime==0){

            v[rning\_proc.id].completionTime = cur\_time;

        }

    }

    return;

}

int main(){

    cout<<"ROUND ROBIN CPU SCHEDULING ALGORITHM C++ IMPLEMENTATION\n";

    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int quantum = 0;

    int n=0;

    cout<<"Enter the number of processes: ";

    cin>>n;

    cout<<"Enter value of time quantum: ";

    cin>>quantum;

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

    vector<process> v(n);

    for(int i=0; i<n; i++){

        cin>>v[i].arrivalTime>>v[i].burstTime;

        v[i].id = i;

        v[i].isRunning = false;

        v[i].remainingBurstTime = v[i].burstTime;

    }

    round\_robin\_scheduling(v, quantum);

    int t\_TAT=0;

    int t\_CT=0;

    for(int i=0; i<n; i++){

        v[i].TAT = v[i].completionTime-v[i].arrivalTime;

        v[i].WT = v[i].TAT-v[i].burstTime;

        t\_TAT+=v[i].TAT;

        t\_CT+=v[i].completionTime;

    }

    for(auto p:v){

        cout<<"Process: "<<p.id<<"\tArrival Time:"<<p.arrivalTime<<"\tBurst Time:"<<p.burstTime<<"\tCompletion Time:"<<p.completionTime;

        cout<<"\tTurn Around Time:"<<p.TAT<<"\tWaiting Time:"<<p.WT<<"\tResponse Time:"<<p.RT<<"\n";

    }

    cout<<"\nAverage Turn Around Time: "<<(float)((1.0\*t\_TAT)/(1.0\*n))<<"\n";

    cout<<"\nAverage Completion Time: "<<(float)((1.0\*t\_CT)/(1.0\*n))<<"\n";

    return 0;

}

FIRST COME FIRST SERVE CPU SCHEDULING

CRITERIA: Arrival Time

MODE: Non pre-emptive

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

CODE:

#include<bits/stdc++.h>

using namespace :: std;

class process{

    public:

        int id;

        int arrivalTime;

        int burstTime;

        int completionTime;

        int TAT;

        int WT;

        int RT;

};

void FCFS(vector<process> &v){

    int cur\_time = 0;

    int id = 0;

    for(int i=0; i<v.size(); i++){

        if(cur\_time<v[i].arrivalTime){

            cout<<"CPU idle from "<<cur\_time<<" to "<<v[i].arrivalTime<<endl;

            cur\_time = v[i].arrivalTime;

        }

        v[i].completionTime = cur\_time+v[i].burstTime;

        v[i].RT = cur\_time-v[i].arrivalTime;

        cout<<"Process P"<<v[i].id+1<<": start time = "<<cur\_time<<" completion time = "<<v[i].completionTime<<endl;

        cur\_time+=v[i].burstTime;

    }

}

int main(){

    cout<<"FIRST COME FIRST SERVE CPU SCHEDULING ALGORITHM C++ IMPLEMENTATION\n";

    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int n=0;

    cout<<"Enter the number of processes: ";

    cin>>n;

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

    vector<process> v(n);

    for(int i=0; i<n; i++){

        cin>>v[i].arrivalTime>>v[i].burstTime;

        v[i].id = i;

    }

    cout<<"-------------------------------------------------------------------------------\n";

    cout<<"\n";

    FCFS(v);

    cout<<"\n";

    cout<<"-------------------------------------------------------------------------------\n";

    cout<<"\n\n";

    int t\_TAT=0;

    int t\_CT=0;

    for(int i=0; i<n; i++){

        v[i].TAT = v[i].completionTime-v[i].arrivalTime;

        v[i].WT = v[i].TAT-v[i].burstTime;

        t\_TAT+=v[i].TAT;

        t\_CT+=v[i].completionTime;

    }

    for(auto p:v){

        cout<<"Process: "<<p.id<<"\tArrival Time:"<<p.arrivalTime<<"\tBurst Time:"<<p.burstTime<<"\tCompletion Time:"<<p.completionTime;

        cout<<"\tTurn Around Time:"<<p.TAT<<"\tWaiting Time:"<<p.WT<<"\tResponse Time:"<<p.RT<<"\n";

    }

    cout<<"\nAverage Turn Around Time: "<<(float)((1.0\*t\_TAT)/(1.0\*n))<<"\n";

    cout<<"\nAverage Completion Time: "<<(float)((1.0\*t\_CT)/(1.0\*n))<<"\n";

    return 0;

}

OS LAB 3

|  |  |  |  |
| --- | --- | --- | --- |
| S No. | Title | Date of Implementation | Remarks |
| 1 | Program to implement Shortest Job First Process of CPU Scheduling | 02-02-2022 |  |
| 2 | Program to implement First Come First Serve Process of CPU Scheduling | 02-02-2022 |  |
| 3 | Program to implement Priority based Scheduling Process of CPU Scheduling | 02-02-2022 |  |

SHORTEST JOB FIRST CPU SCHEDULING

CRITERIA: Burst Time

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

MODE: Non pre-emptive

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

CODE:

#include <bits/stdc++.h>

using namespace ::std;

class process{

    public:

        int priority;

        int id;

        int arrivalTime;

        int burstTime;

        bool ready;

        int completionTime;

        int TAT;

        int WT;

        int RT;

};

struct comp{

    bool operator()(process const &p1, process const &p2){

        return p1.burstTime > p2.burstTime;

    }

};

void SJF(vector<process> &v){

    priority\_queue<process, vector<process>, comp> p;

    int cur\_time = INT\_MAX;

    int n=v.size();

    for(int i=0; i<n; i++){

        cur\_time = min(cur\_time, v[i].arrivalTime);

    }

    int count = 0;

    while(true){

        for(int i=0; i<n; i++){

            if(v[i].arrivalTime<=cur\_time && !v[i].ready){

                v[i].ready = true;

                p.push(v[i]);

                count++;

            }

        }

        if(count<n && p.empty()){

            cout<<"CPU empty from "<<cur\_time<<" to "<<cur\_time+1<<"\n";

            cur\_time++;

            continue;

        }

        if(p.empty()){

            break;

        }

        process cur\_process = p.top();

        p.pop();

        v[cur\_process.id].RT = cur\_time-cur\_process.arrivalTime;

        cur\_time+=cur\_process.burstTime;

        v[cur\_process.id].completionTime=cur\_time;

    }

}

int main(){

    cout<<"SHORTEST JOB FIRST CPU SCHEDULING ALGORITHM C++ IMPLEMENTATION\n";

    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int n=0;

    cout<<"Enter the number of processes: ";

    cin>>n;

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

    vector<process> v(n);

    for(int i=0; i<n; i++){

        cin>>v[i].arrivalTime>>v[i].burstTime;

        v[i].id = i;

        v[i].ready = false;

    }

    cout<<"-------------------------------------------------------------------------------\n";

    cout<<"\n";

    SJF(v);

    cout<<"\n";

    cout<<"-------------------------------------------------------------------------------\n";

    cout<<"\n\n";

    int t\_TAT=0;

    int t\_CT=0;

    for(int i=0; i<n; i++){

        v[i].TAT = v[i].completionTime-v[i].arrivalTime;

        v[i].WT = v[i].TAT-v[i].burstTime;

        t\_TAT+=v[i].TAT;

        t\_CT+=v[i].completionTime;

    }

    for(auto p:v){

        cout<<"Process: "<<p.id<<"\tArrival Time:"<<p.arrivalTime<<"\tBurst Time:"<<p.burstTime<<"\tCompletion Time:"<<p.completionTime;

        cout<<"\tTurn Around Time:"<<p.TAT<<"\tWaiting Time:"<<p.WT<<"\tResponse Time:"<<p.RT<<"\n";

    }

    cout<<"\nAverage Turn Around Time: "<<(float)((1.0\*t\_TAT)/(1.0\*n))<<"\n";

    cout<<"\nAverage Completion Time: "<<(float)((1.0\*t\_CT)/(1.0\*n))<<"\n";

    return 0;

}

FIRST COME FIRST SERVE CPU SCHEDULING

CRITERIA: Arrival Time

MODE: Non pre-emptive

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

CODE:

#include<bits/stdc++.h>

using namespace :: std;

class process{

    public:

        int id;

        int arrivalTime;

        int burstTime;

        int completionTime;

        int TAT;

        int WT;

        int RT;

};

void FCFS(vector<process> &v){

    int cur\_time = 0;

    int id = 0;

    for(int i=0; i<v.size(); i++){

        if(cur\_time<v[i].arrivalTime){

            cout<<"CPU idle from "<<cur\_time<<" to "<<v[i].arrivalTime<<endl;

            cur\_time = v[i].arrivalTime;

        }

        v[i].completionTime = cur\_time+v[i].burstTime;

        v[i].RT = cur\_time-v[i].arrivalTime;

        cout<<"Process P"<<v[i].id+1<<": start time = "<<cur\_time<<" completion time = "<<v[i].completionTime<<endl;

        cur\_time+=v[i].burstTime;

    }

}

int main(){

    cout<<"FIRST COME FIRST SERVE CPU SCHEDULING ALGORITHM C++ IMPLEMENTATION\n";

    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int n=0;

    cout<<"Enter the number of processes: ";

    cin>>n;

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

    vector<process> v(n);

    for(int i=0; i<n; i++){

        cin>>v[i].arrivalTime>>v[i].burstTime;

        v[i].id = i;

    }

    cout<<"-------------------------------------------------------------------------------\n";

    cout<<"\n";

    FCFS(v);

    cout<<"\n";

    cout<<"-------------------------------------------------------------------------------\n";

    cout<<"\n\n";

    int t\_TAT=0;

    int t\_CT=0;

    for(int i=0; i<n; i++){

        v[i].TAT = v[i].completionTime-v[i].arrivalTime;

        v[i].WT = v[i].TAT-v[i].burstTime;

        t\_TAT+=v[i].TAT;

        t\_CT+=v[i].completionTime;

    }

    for(auto p:v){

        cout<<"Process: "<<p.id<<"\tArrival Time:"<<p.arrivalTime<<"\tBurst Time:"<<p.burstTime<<"\tCompletion Time:"<<p.completionTime;

        cout<<"\tTurn Around Time:"<<p.TAT<<"\tWaiting Time:"<<p.WT<<"\tResponse Time:"<<p.RT<<"\n";

    }

    cout<<"\nAverage Turn Around Time: "<<(float)((1.0\*t\_TAT)/(1.0\*n))<<"\n";

    cout<<"\nAverage Completion Time: "<<(float)((1.0\*t\_CT)/(1.0\*n))<<"\n";

    return 0;

}

PRIORITY BASED CPU SCHEDULING

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

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

MODE: Non pre-emptive

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

CODE:

#include <bits/stdc++.h>

using namespace ::std;

class process{

    public:

        int priority;

        int id;

        int arrivalTime;

        int burstTime;

        bool ready;

        int completionTime;

        int TAT;

        int WT;

        int RT;

};

struct comp{

    bool operator()(process const &p1, process const &p2){

        return p1.priority < p2.priority;

    }

};

void PriorityBasedScheduling(vector<process> &v){

    priority\_queue<process, vector<process>, comp> p;

    int cur\_time = INT\_MAX;

    int n=v.size();

    for(int i=0; i<n; i++){

        cur\_time = min(cur\_time, v[i].arrivalTime);

    }

    int count = 0;

    while(true){

        for(int i=0; i<n; i++){

            if(v[i].arrivalTime<=cur\_time && !v[i].ready){

                v[i].ready = true;

                p.push(v[i]);

                count++;

            }

        }

        if(count<n && p.empty()){

            cout<<"CPU empty from "<<cur\_time<<" to "<<cur\_time+1<<"\n";

            cur\_time++;

            continue;

        }

        if(p.empty()){

            break;

        }

        process cur\_process = p.top();

        p.pop();

        v[cur\_process.id].RT = cur\_time-cur\_process.arrivalTime;

        cur\_time+=cur\_process.burstTime;

        v[cur\_process.id].completionTime=cur\_time;

    }

}

int main(){

    cout << "PRIORITY BASED CPU SCHEDULING ALGORITHM C++ IMPLEMENTATION\n";

    cout << "Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int n = 0;

    cout << "Enter the number of processes: ";

    cin >> n;

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

    vector<process> v(n);

    for (int i = 0; i < n; i++){

        cin >> v[i].arrivalTime >> v[i].burstTime >> v[i].priority;

        v[i].ready = false;

        v[i].id = i;

    }

    cout << "-------------------------------------------------------------------------------\n";

    cout << "\n";

    PriorityBasedScheduling(v);

    cout << "\n";

    cout << "-------------------------------------------------------------------------------\n";

    cout << "\n\n";

    int t\_TAT = 0;

    int t\_CT = 0;

    for (int i = 0; i < n; i++){

        v[i].TAT = v[i].completionTime - v[i].arrivalTime;

        v[i].WT = v[i].TAT - v[i].burstTime;

        t\_TAT += v[i].TAT;

        t\_CT += v[i].completionTime;

    }

    for (auto p : v){

        cout << "Process: " << p.id << "\tArrival Time:" << p.arrivalTime << "\tBurst Time:" << p.burstTime << "\tCompletion Time:" << p.completionTime;

        cout << "\tTurn Around Time:" << p.TAT << "\tWaiting Time:" << p.WT << "\tResponse Time:" << p.RT << "\n";

    }

    cout << "\nAverage Turn Around Time: " << (float)((1.0 \* t\_TAT) / (1.0 \* n)) << "\n";

    cout << "\nAverage Completion Time: " << (float)((1.0 \* t\_CT) / (1.0 \* n)) << "\n";

    return 0;

}

OS LAB 4

|  |  |  |  |
| --- | --- | --- | --- |
| S No. | Title | Date of Implementation | Remarks |
| 1 | Program to handle the Critical Section Problem using semaphore | 09-02-2022 |  |

CRITICAL SECTION PROBLEM SOLUTION USING SEMAPHORE

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

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

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

CODE:

#include<bits/stdc++.h>

using namespace :: std;

queue<int> ready;

class semaphore{

    int value;

    public:

        queue<int> blocked\_list, CS;

        semaphore(){

            this->value = 9; // Last digit of my roll number.

        }

        void down(int p){         // Entry code before entering Critical Section

            this->value--;

            if(this->value<0){

                // Put the process in the blocked list

                blocked\_list.push(p);

                cout<<"Process "<<p<<" has been put in the blocked list\n";

            }

            else{

                // Put the process in the Critical Section

                CS.push(p);

                cout<<"Process "<<p<<" has entered the Critical Section\n";

            }

        }

        void up(){

            this->value++;

            if(!CS.empty()){

                int p=CS.front();

                cout<<"Process "<<p<<" has exited the Critical Section\n";

                CS.pop();

            }

            if(this->value<=0){

                // Wake up a sleeping process from the blocked list

                int p=blocked\_list.front();

                cout<<"Process "<<p<<" has been removed from the blocked list\n";

                blocked\_list.pop();

                // Put the blocked process back in the ready queue

                ready.push(p);

            }

        }

        void inCS(){

            if(CS.empty()){

                cout<<"The Critical Section is empty\n";

            }

            else{

                queue<int> temp = CS;

                cout<<"Processes present in Critical Section are: ";

                while(!temp.empty()){

                    cout<<temp.front()<<" ";

                    temp.pop();

                }

                cout<<"\n";

            }

        }

        void inBlockedList(){

            if(blocked\_list.empty()){

                cout<<"The Blocked List is empty\n";

            }

            else{

                queue<int> temp = blocked\_list;

                cout<<"Processes present in Blocked List are: ";

                while(!temp.empty()){

                    cout<<temp.front()<<" ";

                    temp.pop();

                }

                cout<<"\n";

            }

        }

};

int main(){

    cout<<"PROCESS SYNCHRONISATION USING SEMAPHORE C++ IMPLEMENTATION\n";

    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int n=0;

    cout<<"Enter number of processes: ";

    cin>>n;

    for(int i=1; i<=n; i++){

        ready.push(i);

    }

    semaphore S;

    int itr=0;

    while(!ready.empty()){

        itr++;

        while(!ready.empty()){

            int process = ready.front();

            ready.pop();

            S.down(process);       // Each process tries to enter the critical section

            if(itr!=1){

                break;

            }

        }

        S.inCS();

        cout<<"\n";

        S.inBlockedList();

        cout<<"\n\n";

        while(!S.CS.empty()){

            S.up();

        }

    }

    return 0;

}

OS LAB 5

|  |  |  |  |
| --- | --- | --- | --- |
| S No. | Title | Date of Implementation | Remarks |
| 1 | Program to handle the Critical Section Problem using semaphore | 16-02-2022 |  |

PRODUCER-CONSUMER PROBLEM SOLUTION USING SEMAPHORE AND MUTEX

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

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

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

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

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

CODE:

// Solution of the Producer Consumer Problem using semaphore

#include<bits/stdc++.h>

using namespace :: std;

// Initially we have an empty buffer with a size 5

int m=1;                    // Mutex

int full=0;                 // Counting Semaphore

int empty=5;                // Counting Semaphore

int IN=0;

int OUT=0;

void down(int &s){

    s--;

}

void up(int &s){

    s++;

}

void producer(){

    down(m);

    up(full);

    down(empty);

    IN++;

    cout<<"\nProducer produced the item "<<IN;

    up(m);

}

void consumer(){

    down(m);

    down(full);

    up(empty);

    OUT++;

    cout<<"\nConsumer consumed the item "<<OUT;

    up(m);

}

int main(){

    cout<<"SOLUTION OF PRODUCER CONSUMER PROBLEM USING SEMAPHORE AND MUTEX C++ IMPLEMENTATION\n";

    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int n=0;

    while(1){

        cout<<"Select the preference: (1, 2, or 3) \n";

        cout<<"1) Producer\n2)Consumer\n3)Exit\n";

        cin>>n;

        if(n==1){

            if(m==1 && empty!=0){

                producer();

                cout<<endl;

            }

            else{

                cout<<"Buffer is full\n";

            }

        }

        else if(n==2){

            if(m==1 && full!=0){

                consumer();

                cout<<endl;

            }

            else{

                cout<<"Buffer is empty\n";

            }

        }

        else{

            break;

        }

    }

    return 0;

}

OS LAB 6

|  |  |  |  |
| --- | --- | --- | --- |
| S No. | Title | Date of Implementation | Remarks |
| 1 | Simulation of Bankers Deadlock Avoidance and Prevention algorithms | 23-02-2022 |  |

SIMULATION OF BANKERS DEADLOCK AVOIDANCE AND PREVENTION ALGORITHM

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

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

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

CODE:

// C++ implementation of Banker's deadlock avoidance and prevention algorithm

#include<bits/stdc++.h>

using namespace :: std;

// Class representing state of the system: how various resources are allocated and requested by various processes

struct state{

    int resources, processes;

    vector<vector<int> > allocated;

    vector<vector<int> > max\_req;

    vector<vector<int> > remaining\_req;

    vector<int> available;

    vector<bool> executed;

    state(int res, int pro){

        this->resources = res;

        this->processes = pro;

        allocated.resize(pro, vector<int>(res, 0));

        max\_req.resize(pro, vector<int>(res, 0));

        remaining\_req.resize(pro, vector<int>(res, 0));

        available.resize(res, 0);

        executed.resize(pro, false);

        cout<<"executed";

    }

};

queue<int> safe\_seq;

bool bankers(state s){

    int n=s.processes;

    while(n--){

        cout<<"Current availability: ";

        for(int i=0; i<s.resources; i++){

            cout<<s.available[i]<<" ";

        }

        cout<<"\n";

        bool found=false;

        for(int i=0; i<s.processes; i++){

            bool t=true;

            if(!s.executed[i]){

                for(int j=0; j<s.resources; j++){

                    if(s.available[j]<s.remaining\_req[i][j]){

                        t=false;

                        break;

                    }

                }

                if(t){

                    found=true;

                    safe\_seq.push(i+1);

                    s.executed[i]=true;

                    cout<<"P"<<i+1<<" has been allocated the resources\n\n";

                    for(int j=0; j<s.resources; j++){

                        s.available[j]+=s.allocated[i][j];

                    }

                    break;

                }

            }

        }

        if(!found){

            cout<<"Resources could not be allocated to any process.\n\n";

            return true;

        }

    }

    return false;

}

int main(){

    cout<<"BANKER'S DEADLOCK AVOIDANCE AND PREVENTION ALGORITHM C++ IMPLEMENTATION\n";

    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int res=0, pro=0;

    cout<<"Enter number of resources: ";

    cin>>res;

    cout<<"Enter number of processes: ";

    cin>>pro;

    state s(res, pro);

    cout<<"For "<<pro<<" processes enter: \n";

    for(int i=0; i<pro; i++){

        cout<<"Resources allocated to process "<<i+1<<": ";

        for(int j=0; j<res; j++){

            cin>>s.allocated[i][j];

        }

        cout<<"Max resources required by process "<<i+1<<": ";

        for(int j=0; j<res; j++){

            cin>>s.max\_req[i][j];

            s.remaining\_req[i][j]=s.max\_req[i][j]-s.allocated[i][j];

        }

    }

    cout<<"Enter the currently available resources: ";

    for(int i=0; i<res; i++){

        cin>>s.available[i];

    }

    cout<<"\n\n---------------------------------------------------------------------------\n\n";

    bool deadlock = bankers(s);

    cout<<"\n\n---------------------------------------------------------------------------\n\n";

    if(deadlock){

        cout<<"Deadlock Situation Detected!\n";

    }

    else{

        cout<<"No Deadlock Detected!\n";

        cout<<"Safe sequence of resource allocation: ";

        while(!safe\_seq.empty()){

            cout<<"P"<<safe\_seq.front()<<" ";

            safe\_seq.pop();

        }

        cout<<"\n";

    }

    return 0;

}

OS LAB 7

|  |  |  |  |
| --- | --- | --- | --- |
| S No. | Title | Date of Implementation | Remarks |
| 1 | Implementation of solution of Reader-Writer Problem | 02-03-2022 |  |
| 2 | Implementation of solution of Sleeping Barber Problem | 02-03-2022 |  |

IMPLEMENTATION OF SOLUTION OF READER-WRITER PROBLEM

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

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

Problem parameters:

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

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

CODE:

// C++ implementation of solution of Reader-Writer Problem using Binary Semaphore (Process Synchronisation)

#include<bits/stdc++.h>

using namespace :: std;

struct Semaphore{

    bool s;

    Semaphore(){

        this->s=1;

    }

    void down(){

        if(this->s==0){

            cout<<"RESTRICTED!\n";

            return;

        }

        this->s=this->s-1;

    }

    void up(){

        this->s=this->s+1;

    }

};

// structure to represent the database

struct database{

    int value;

};

int rc=0;  // Reader Count

void Reader(Semaphore &mutex, Semaphore &db, database &d){

    mutex.down();

    rc++;

    if(rc==1){

        db.down();

    }

    mutex.up();

    // -------------------------------------------------------------------------

    cout<<"Reader "<<rc<<" is reading the database\n";

    cout<<"Value stored in the database = "<<d.value<<"\n\n";

    // -------------------------------------------------------------------------

    mutex.down();

    rc--;

    if(rc==0){

        db.up();

    }

    mutex.up();

}

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

    db.down();

    // --------------------------------------------------------------------------

    cout<<"Writer is writing in the database\n";

    d.value=val;

    cout<<"Value updated!\n\n";

    // --------------------------------------------------------------------------

    db.up();

}

int main(){

    cout<<"SOLUTION TO READER-WRITER PROBLEM USING SEMAPHORE C++ IMPLEMENTATION\n";

    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    Semaphore mutex;   // Binary semaphore to synchronise incrementing of reader count

    Semaphore db;      // Binary semaphore to synchronise the access to a database

    database d;

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

    while(1){

        int n=0;

        cout<<"Enter \n1 for reader\n2 for writer\n3 to exit\n";

        cin>>n;

        if(n==1){

            Reader(mutex, db, d);

        }

        else if(n==2){

            cout<<"Enter the value you want to write: ";

            int val=0;

            cin>>val;

            Writer(mutex, db, d, val);

        }

        else break;

    }

    return 0;

}

IMPLEMENTATION OF SOLUTION OF SLEEPING BARBER PROBLEM

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

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

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

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

CODE:

// C++ implementation of solution of Sleeping Barber Problem using Semaphore (Process Synchronisation)

#include<bits/stdc++.h>

using namespace :: std;

struct binarySemaphore{

    bool s;

};

struct countingSemaphore{

    int s;

};

void customer(binarySemaphore &barber, binarySemaphore &cut, int Customers){

    int id=1;

    while(id<=Customers){

        // Wake the barber up if he is sleeping

        if(barber.s==0){

            barber.s=1;

        }

        // Cut the hair

        cut.s=1;

        cout<<"Customer "<<id<<" is getting a haircut\n\n";

        cut.s=0;

        id++;

    }

    // The barber goes to sleep after tending to all the customers

    barber.s=0;

}

int main(){

    cout<<"SOLUTION TO SLEEPING BARBER PROBLEM USING SEMAPHORE C++ IMPLEMENTATION\n";

    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    binarySemaphore barber;    // denotes if the barber is sleeping or awake

    binarySemaphore cut;       // semaphore to synchronise hair cutting

    countingSemaphore freeChairs;

    cout<<"Enter the number of free chairs: ";

    cin>>freeChairs.s;

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

    cut.s=0;

    // Suppose the customers visit the shop in groups. The group visits the barber only after all the customers of first group have left the shop.

    while(1){

        int freeSpace=freeChairs.s;

        int customers=0;

        cout<<"Enter the number of customers entering the shop (enter 0 to exit): ";

        cin>>customers;

        if(customers==0){

            break;

        }

        // The first customer can always go to the sleeping barber

        int id=2;

        while(id<=customers){

            // Chair occupied by customer

            freeChairs.s--;

            id++;

            if(freeChairs.s==0){

                break;

            }

        }

        while(id<=customers){

            cout<<"Customer "<<id<<" returned back without getting a haircut\n";

            id++;

        }

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

        freeChairs.s=freeSpace;

    }

    return 0;

}

OS LAB 8

|  |  |  |  |
| --- | --- | --- | --- |
| S No. | Title | Date of Implementation | Remarks |
| 1 | Implementation of solution of Dining Philosophers Problem | 28-03-2022 |  |

IMPLEMENTATION OF SOLUTION OF DINING PHILOSOPHERS PROBLEM

**The Dining Philosopher Problem:**

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

CODE:

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

#include<bits/stdc++.h>

using namespace :: std;

queue<int> ready;

struct binarySemaphore{

    bool s;

    bool down(){

       if(!s){

           return false;

       }

       s=0;

       return true;

    }

    bool up(){

        if(s){

            return false;

        }

        s=1;

        return true;

    }

};

struct countingSemaphore{

    int s;

    bool down(){

       if(s==0){

           return false;

       }

       s--;

       return true;

    }

    bool up(){

        if(s==5){

            return false;

        }

        s++;

        return true;

    }

};

void philosopherEat(int phil, vector<binarySemaphore> &fork, queue<int> &temp2){

    cout<<"Philosopher "<<phil<<" is hungry\n";

    if(fork[(phil-1)%5].down()){

        if(fork[(phil)%5].down()){

            cout<<"Philosopher "<<phil<<" has started eating\n";

            temp2.push(phil);

        }

        else{

            fork[(phil-1)%5].up();

            cout<<"Forks are not available for Philosopher "<<phil<<" to use\n";

            ready.push(phil);

            return;

        }

    }

    else{

        cout<<"Forks are not available for Philosopher "<<phil<<" to use\n";

        ready.push(phil);

    }

}

void philosopherFinish(vector<binarySemaphore> &fork, queue<int> &temp2){

    while(!temp2.empty()){

        int phil = temp2.front();

        temp2.pop();

        cout<<"Philosopher "<<phil<<" has finished eating\n";

        fork[(phil-1)%5].up();

        fork[(phil)%5].up();

    }

}

void DiningPhilosopher(){

    vector<binarySemaphore> fork(5);

    // Initially all the forks are available

    for(int i=0; i<5; i++){

        fork[i].s = true;

    }

    while(true){

        if(ready.empty()){

            break;

        }

        queue<int> temp1, temp2;

        while(!ready.empty()){

            temp1.push(ready.front());

            // temp2.push(ready.front());

            ready.pop();

        }

        while(!temp1.empty()){

            int phil = temp1.front();

            temp1.pop();

            philosopherEat(phil, fork, temp2);

        }

        while(!temp2.empty()){

            philosopherFinish(fork, temp2);

        }

    }

}

int main(){

    cout<<"SOLUTION TO DINING PHILOSOPHERS PROBLEM USING SEMAPHORE C++ IMPLEMENTATION\n";

    cout<<"Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    cout<<"There are 5 philosophers(numbered 1-5) sitting on a round table and 5 forks.\n";

    cout<<"Enter the order in which Philosphers get hungry: ";

    for(int i=0; i<5; i++){

        int n=0;

        cin>>n;

        ready.push(n);

    }

    DiningPhilosopher();

    return 0;

}

OS LAB 9

NAME: Aditya Anand

ROLL NO.: 20124009

BRANCH: IT

|  |  |  |  |
| --- | --- | --- | --- |
| S No. | Title | Date of Implementation | Remarks |
| 1 | Implementation of First In First Out Page Replacement Algorithm | 18-04-2022 |  |
| 2 | Implementation of First In First Out Page Replacement Algorithm | 18-04-2022 |  |
| 3 | Implementation of First In First Out Page Replacement Algorithm | 18-04-2022 |  |
| 4 | Simulation Of Paging Techniques In Memory Management | 18-04-2022 |  |

IMPLEMENTATION OF FIRST IN FIRST OUT PAGE REPLACEMENT ALGORITHM

CODE:

#include<bits/stdc++.h>

using namespace :: std;

int search(vector<int> arr, int key){

    for(int i=0; i<arr.size(); i++){

        if(arr[i]==key){

            return i;

        }

    }

    return -1;

}

void FIFO(vector<int> ref, int frames){

    vector<int> fr(frames, -1);

    int k = 0;

    int hit = 0, miss = 0;

    for(int i=0; i<ref.size(); i++){

        int id = search(fr, ref[i]);

        if(id!=-1){

            hit++;

        }

        else{

            fr[k]=ref[i];

            k=(k+1)%frames;

            miss++;

        }

    }

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

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

}

int main(){

    cout << "FIRST IN FIRST OUT PAGE REPLACEMENT ALGORITHM C++ IMPLEMENTATION\n";

    cout << "Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int frames = 0;

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

    cin>>frames;

    if(frames<1){

        cout<<"No frames available!";

        return 0;

    }

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

    int n = 0;

    cin>>n;

    cout<<"Enter the order in which pages are accessed by the CPU\n";

    vector<int> ref(n, 0);

    for(int i=0; i<n; i++){

        cin>>ref[i];

    }

    FIFO(ref, frames);

    return 0;

}

IMPLEMENTATION OF LEAST RECENTLY USED PAGE REPLACEMENT ALGORITHM

CODE:

#include<bits/stdc++.h>

using namespace :: std;

struct frame{

    int frNo, lastOcc;

};

// returns the index at which the key is found and the index of the least recently used page

pair<int, int> Search(vector<frame> &arr, int key, vector<int> ref, int cur){

    int id = -1, lru = INT\_MAX;

    for(int i=0; i<arr.size(); i++){

        if(arr[i].frNo==key){

            id = i;

        }

        for(int j=cur-1; j>=0; j--){

            if(ref[j]==arr[i].frNo){

                arr[i].lastOcc = j;

                lru = min(lru, arr[i].lastOcc);

                break;

            }

        }

    }

    int pRep = -1;

    for(int i=0; i<arr.size(); i++){

        if(arr[i].lastOcc == lru){

            pRep = i;

            break;

        }

    }

    return make\_pair(id, pRep);

}

void LRU(vector<int> ref, int frames){

    vector<frame> fr(frames);

    for(int i=0; i<frames; i++){

        fr[i].frNo = -1;

        fr[i].lastOcc = -1;

    }

    int hit = 0, miss = 0;

    int k=0, it=0;

    while(k<frames && it<ref.size()){

        bool found = false;

        for(int j=k; j>=0; j--){

            if(fr[j].frNo == ref[it]){

                found=true;

                hit++;

                break;

            }

        }

        if(!found){

            fr[k].frNo = ref[it];

            fr[k].lastOcc = it;

            miss++;

            k++;

        }

        it++;

    }

    for(int i=it; i<ref.size(); i++){

        pair<int , int> p = Search(fr, ref[i], ref, i);

        if(p.first!=-1){

            hit++;

        }

        else{

            fr[p.second].frNo=ref[i];

            fr[p.second].lastOcc=i;

            miss++;

        }

    }

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

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

}

int main(){

    cout << "LEAST RECENTLY USED REPLACEMENT ALGORITHM C++ IMPLEMENTATION\n";

    cout << "Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int frames = 0;

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

    cin>>frames;

    if(frames<1){

        cout<<"No frames available!";

        return 0;

    }

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

    int n = 0;

    cin>>n;

    cout<<"Enter the order in which pages are accessed by the CPU\n";

    vector<int> ref(n);

    for(int i=0; i<n; i++){

        cin>>ref[i];

    }

    LRU(ref, frames);

    return 0;

}

IMPLEMENTATION OF LEAST FREQUENTLY USED PAGE REPLACEMENT ALGORITHM

CODE:

#include<bits/stdc++.h>

using namespace :: std;

struct frame{

    int frNo, freq;

};

// returns the index at which the key is found and the index of the least frequently used page

pair<int, int> Search(vector<frame> &arr, int key, vector<int> ref, int cur){

    int id = -1, lfu = INT\_MIN;

    for(int i=0; i<arr.size(); i++){

        if(arr[i].frNo==key){

            id = i;

        }

        lfu = max(lfu, arr[i].freq);

    }

    int pRep = -1;

    for(int i=0; i<arr.size(); i++){

        if(arr[i].freq == lfu){

            pRep = i;

            break;

        }

    }

    return make\_pair(id, pRep);

}

void LRU(vector<int> ref, int frames){

    vector<frame> fr(frames);

    for(int i=0; i<frames; i++){

        fr[i].frNo = -1;

        fr[i].freq = 0;

    }

    int hit = 0, miss = 0;

    int k=0, it=0;

    while(k<frames && it<ref.size()){

        bool found = false;

        for(int j=k; j>=0; j--){

            if(fr[j].frNo == ref[it]){

                found=true;

                fr[j].freq++;

                hit++;

                break;

            }

        }

        if(!found){

            fr[k].frNo = ref[it];

            fr[k].freq=1;

            miss++;

            k++;

        }

        it++;

    }

    for(int i=it; i<ref.size(); i++){

        pair<int , int> p = Search(fr, ref[i], ref, i);

        if(p.first!=-1){

            fr[p.first].freq++;

            hit++;

        }

        else{

            fr[p.second].frNo=ref[i];

            fr[p.second].freq=1;

            miss++;

        }

    }

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

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

}

int main(){

    cout << "LEAST FREQUENTLY USED PAGE REPLACEMENT ALGORITHM C++ IMPLEMENTATION\n";

    cout << "Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int frames = 0;

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

    cin>>frames;

    if(frames<1){

        cout<<"No frames available!";

        return 0;

    }

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

    int n = 0;

    cin>>n;

    cout<<"Enter the order in which pages are accessed by the CPU\n";

    vector<int> ref(n);

    for(int i=0; i<n; i++){

        cin>>ref[i];

    }

    LRU(ref, frames);

    return 0;

}

SIMULATION OF PAGING TECHNIQUES IN MEMORY MANAGEMENT

CODE:

#include <iostream>

using namespace std;

#define MAX 50

int main(){

    cout << "SIMULATION OF PAGING TECHNIQUES C++ IMPLEMENTATION\n";

    cout << "Name: Aditya Anand\tRoll No.:20124009\t Branch: IT\n\n\n";

    int page[MAX], i, no\_of\_pgs, no\_of\_frms, pg\_sz, off, pno;

    int choice = 0;

    cout << "Enter no of pages in memory: " << endl;

    cin >> no\_of\_pgs;

    cout << "Enter page size: " << endl;

    cin >> pg\_sz;

    cout << "Enter no of frames: " << endl;

    cin >> no\_of\_frms;

    for (i = 0; i < no\_of\_frms; i++)

      page[i] = -1;

    cout << "\nEnter the page table\n";

    cout << "(Enter frame no as -1 if that page is not present in any frame)\n\n"

        << endl;

    cout << "\npageno\tframeno\n-------\t-------\n";

    for (i = 0; i < no\_of\_pgs; i++){

      cout << "\n\n"

          << i << "\t\t";

      cin >> page[i];

    }

    do{

      cout << "\n\nEnter the logical address(i.e,page no & offset): ";

      cin >> pno >> off;

      if (page[pno] == -1)

        cout << "\n\nThe required page is not available in any of frms";

      else

        cout << "Physical address (i.e, frame no and offset) : " << page[pno] << off << endl;

      cout << "\nDo you want to continue(1/0)?:";

      cin >> choice;

    }while (choice == 1);

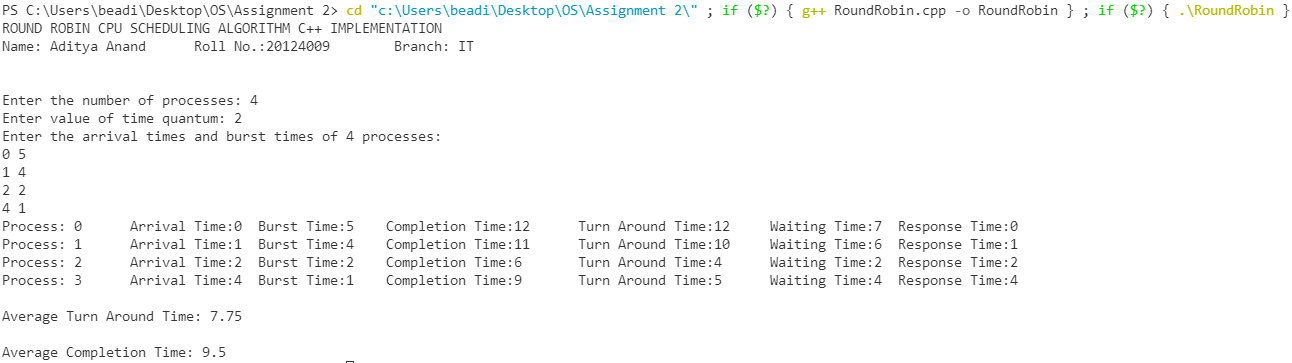
    return 0;

}

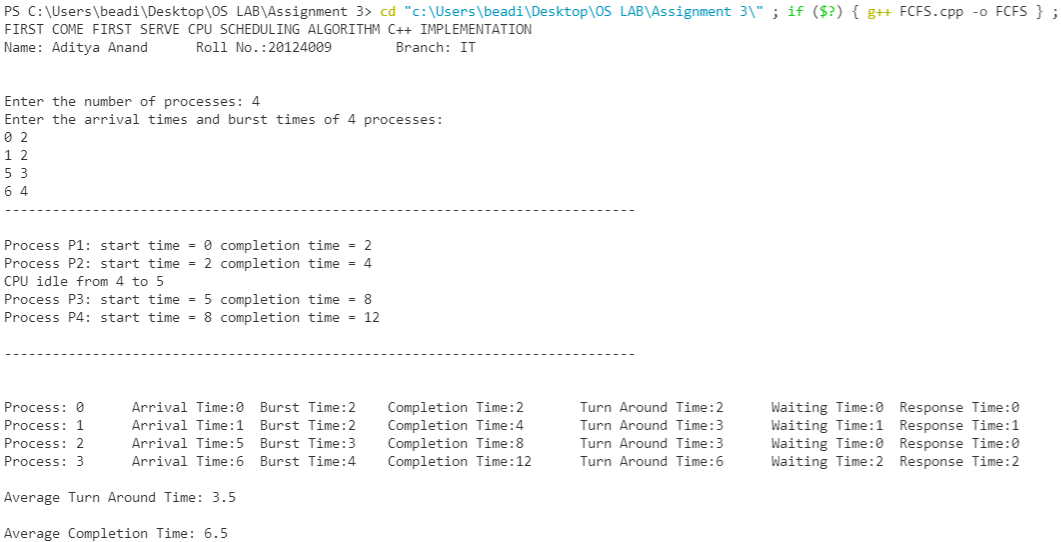
OUTPUTS:

LAB 2:

1)

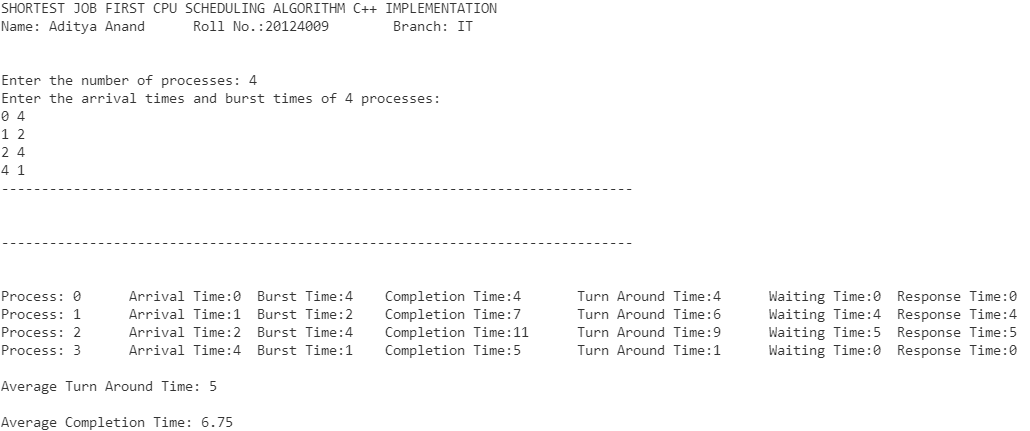


(2)

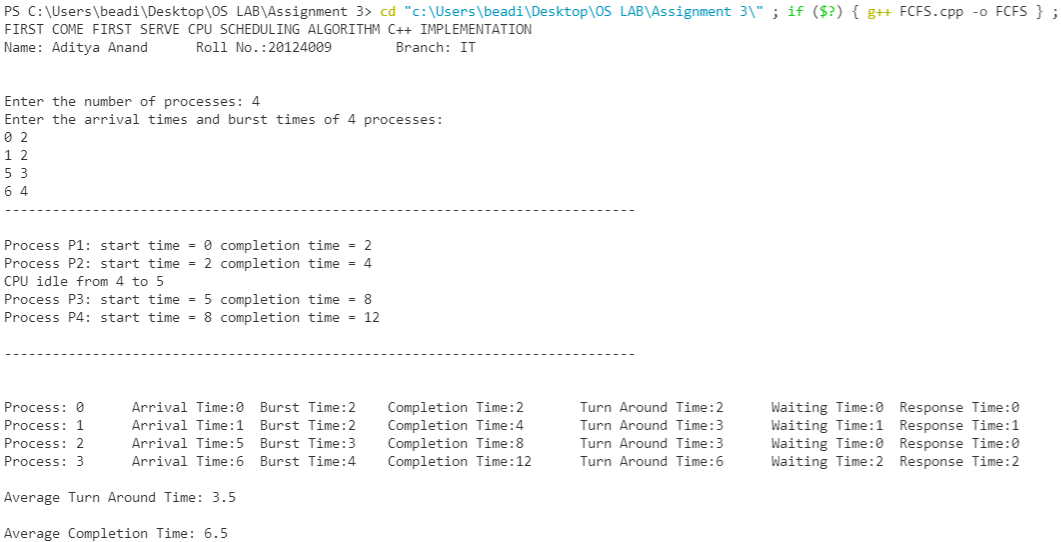


LAB 3:

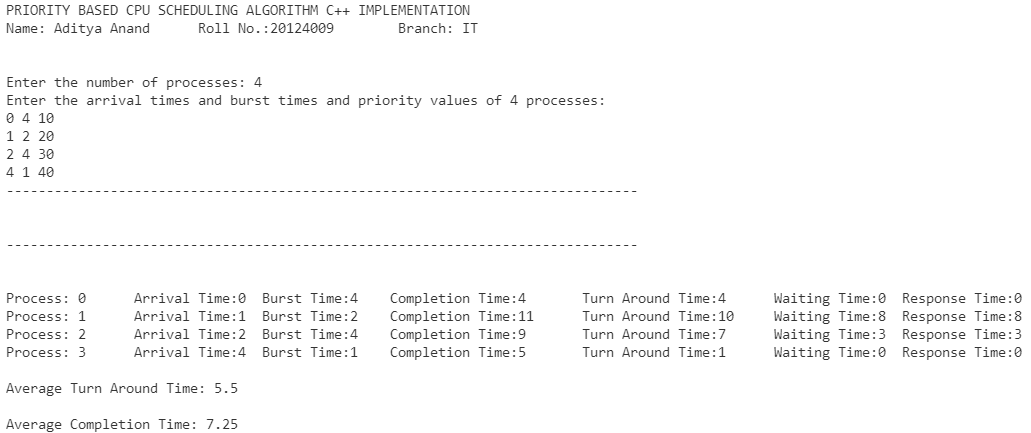
(1)



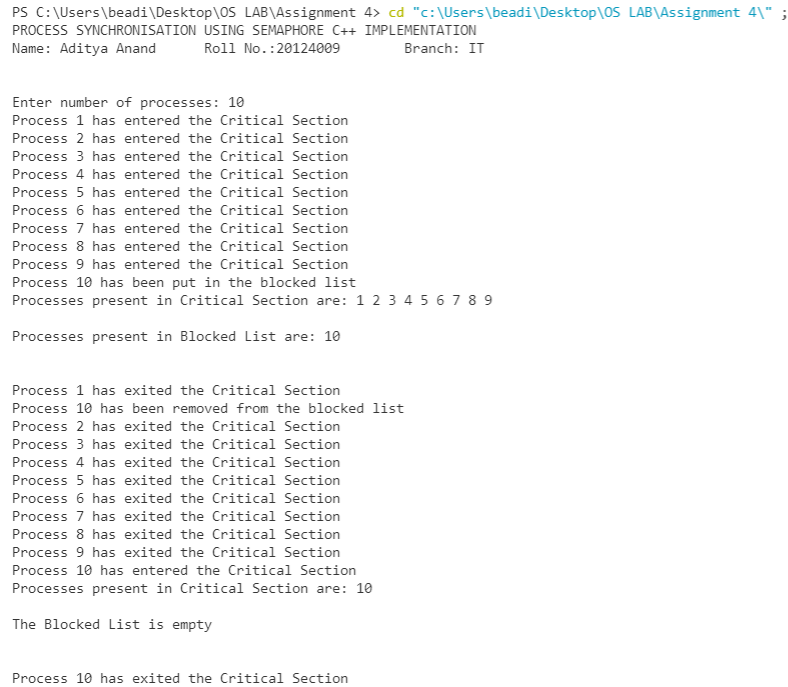
(2)



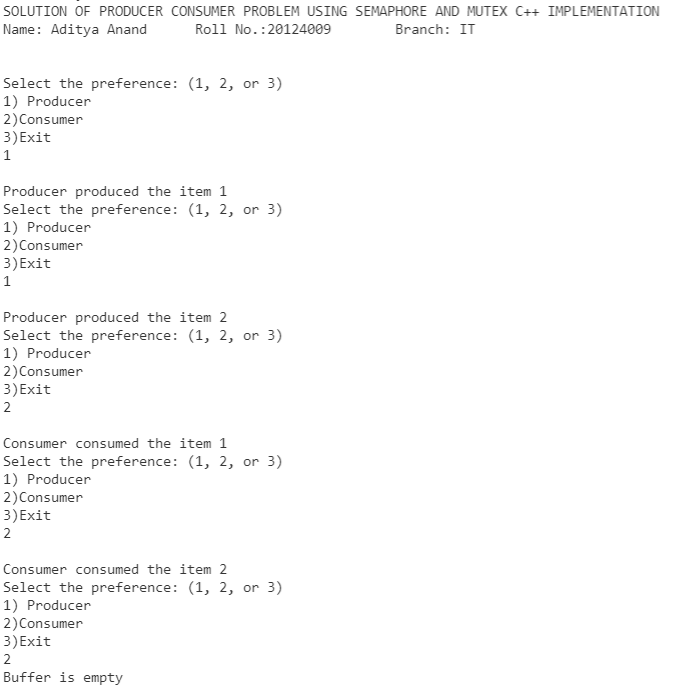
(3)



LAB 4:

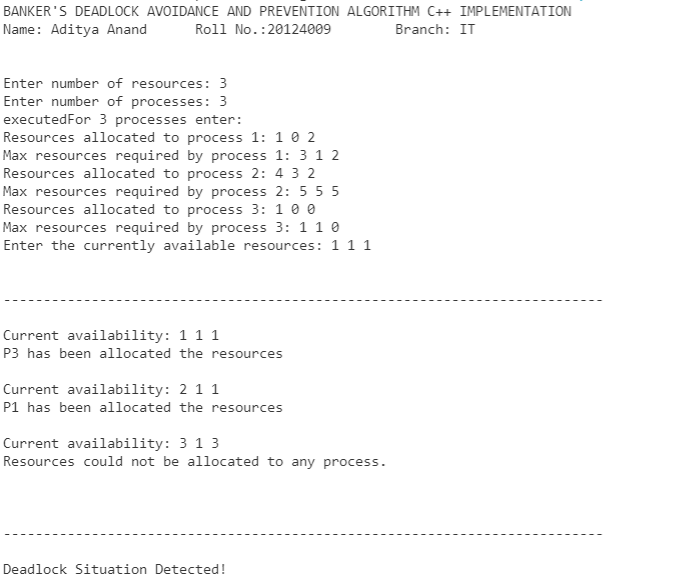


LAB 5:

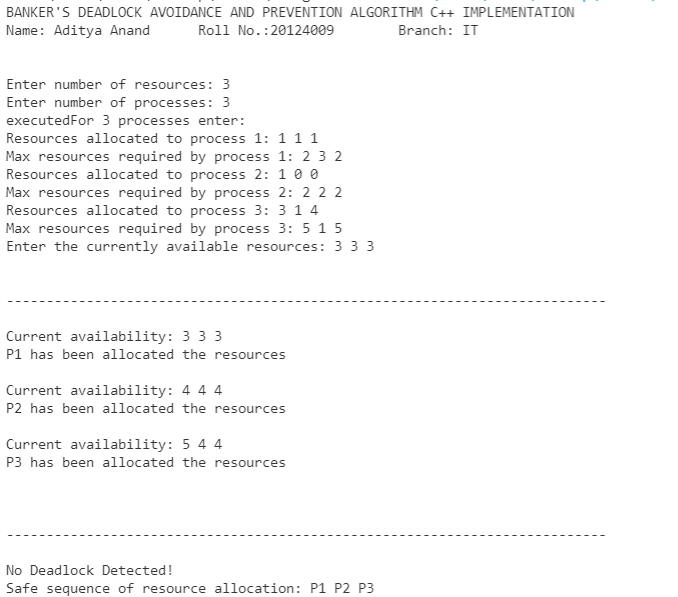


LAB 6:

**In case of Deadlock:**

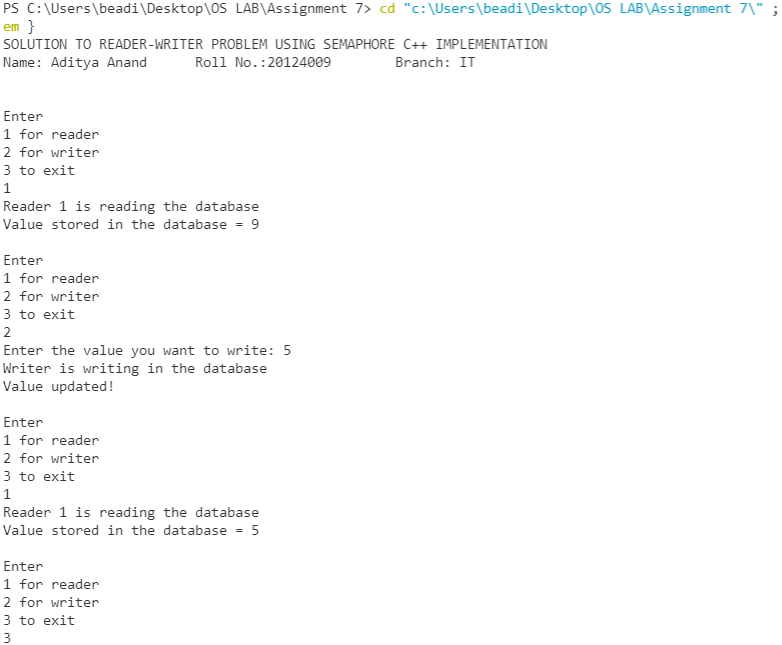


**No Deadlock:**

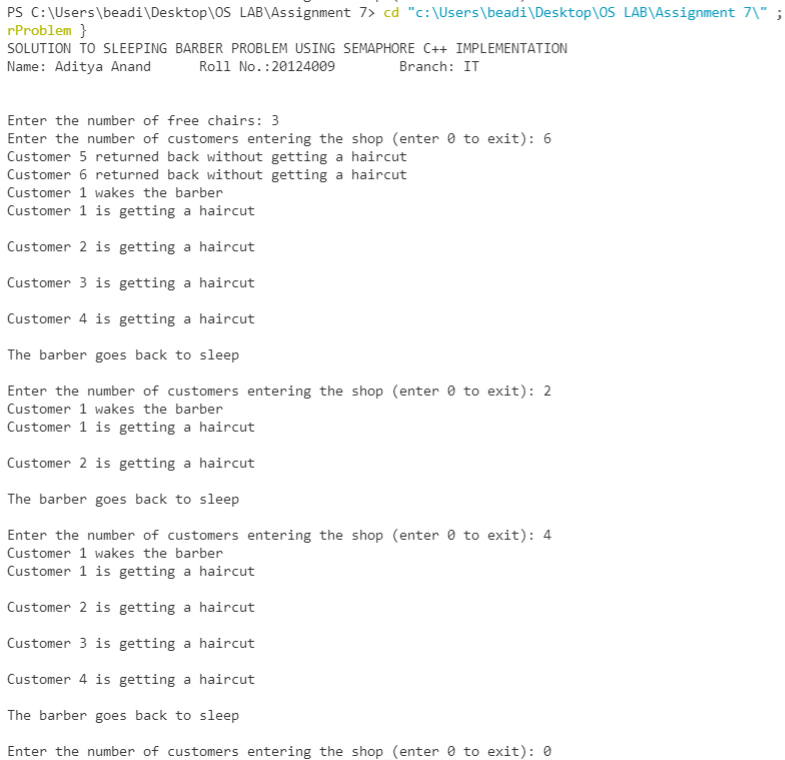
****

LAB 7:

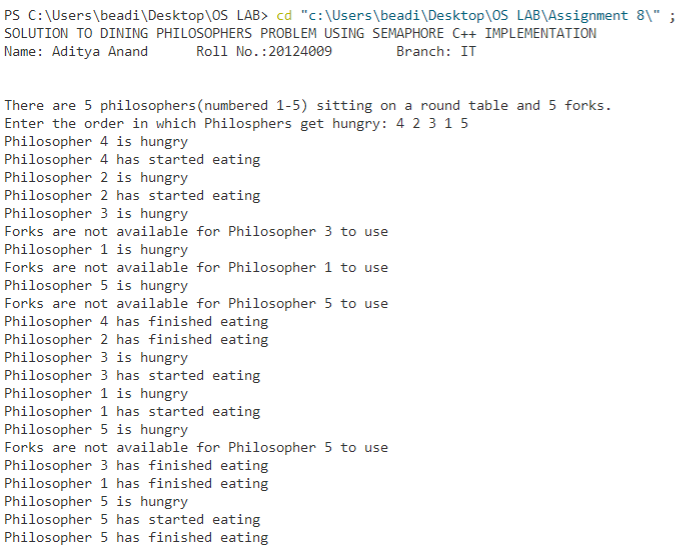
(1)



(2)

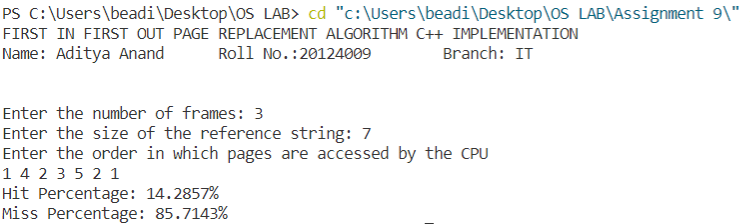


LAB 8:

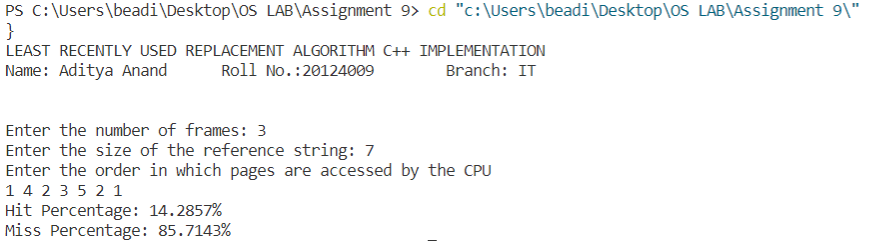


LAB 9:

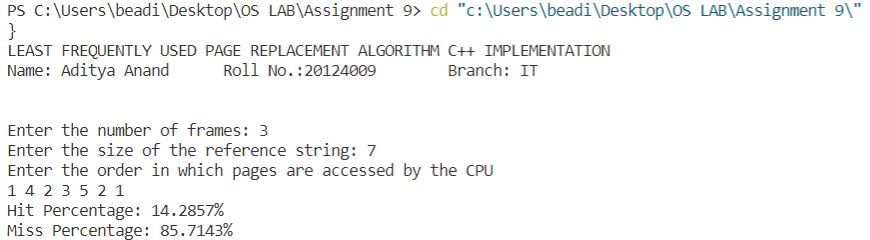
(1)



(2)



(3)



(4)

