**Batch: A2 Roll No.: 16010121045**

**Experiment No. 01**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE: Exploring basic Commands of UNIX: Shell, Processes, Files** |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:** To Explore basic commands for handling File system under Unix/Linux using shell scripts.(Creating groups, chown , chmod , directory name, tty , diff, umask).

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

**CO 1.** To introduce basic concepts and functions of operating systems.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **Silberschatz A., Galvin P., Gagne G. “Operating Systems Principles”, Willey Eight edition.**
2. **Achyut S. Godbole , Atul Kahate “Operating Systems”, McGraw Hill Third Edition.**
3. **Sumitabha Das “ UNIX Concepts & Applications”, McGraw Hill Second**

**Edition.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

An operating system (OS) is a resource manager. It takes the form of a set of software routines that allow users and application programs to access system resources (e.g. the CPU, memory, disks, modems, printers network cards etc.) in safe efficient and abstract way.

* The operating system kernel is in direct control of the underlying hardware. The kernel provides low-level device, memory and processor management functions (e.g. dealing with interrupts from hardware devices, sharing the processor among multiple programs, allocating memory for programs etc.)
* Basic hardware-independent kernel services are exposed to higher-level programs through a library of system calls (e.g. services to create a file, begin execution of a program, or open a logical network connection to another computer).
* Application programs (e.g. word processors, spreadsheets) and system utility programs (simple but useful application programs that come with the operating system, e.g. programs which find text inside a group of files) make use of system calls. Applications and system utilities are launched using a shell (a textual command line interface) or a graphical user interface that provides direct user interaction.

Operating systems can be distinguished from one another by the system calls, system utilities and user interface they provide, as well as by the resource scheduling policies implemented by the kernel.

UNIX has been a popular OS for more than two decades because of its multi-user, multi-tasking environment, stability, portability and powerful networking capabilities.

Linux is a free open source UNIX OS for PCs.

Linux has all of the components of a typical OS :

* **Kernel**

The Linux kernel includes device driver support for a large number of PC hardware devices (graphics cards, network cards, hard disks etc.), advanced processor and memory management features, and support for many different types of file systems. In terms of the services that it provides to application programs and system utilities, the kernel implements most BSD and SYSV system calls, as well as the system calls described in the POSIX.1 specification.

The kernel (in raw binary form that is loaded directly into memory at system startup time) is typically found in the file /boot/vmlinuz, while the source files can usually be found in /usr/src/linux.

* **Shells and GUIs**

Linux supports two forms of command input: through textual command line shells similar to those found on most UNIX systems (e.g. sh - the Bourne shell, bash - the Bourne again shell and csh - the C shell) and through graphical interfaces (GUIs) such as the KDE and GNOME window managers.

* **System Utilities**

Virtually every system utility that you would expect to find on standard implementations of UNIX has been ported to Linux. This includes commands such as ls, cp, grep, awk, sed, bc, wc, more, and so on. These system utilities are designed to be powerful tools that do a single task extremely well (e.g. grep finds text inside files while wc counts the number of words, lines and bytes inside a file). Users can often solve problems by interconnecting these tools instead of writing a large monolithic application program.

* **Application programs**

Linux distributions typically come with several useful application programs as standard. Examples include the emacseditor, xv (an image viewer), gcc (a C compiler), g++ (a C++ compiler), xfig (a drawing package), latex (a powerful typesetting language) and soffice (StarOffice, which is an MS-Office style clone that can read and write Word, Excel and PowerPoint files).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Description of Commands and options:

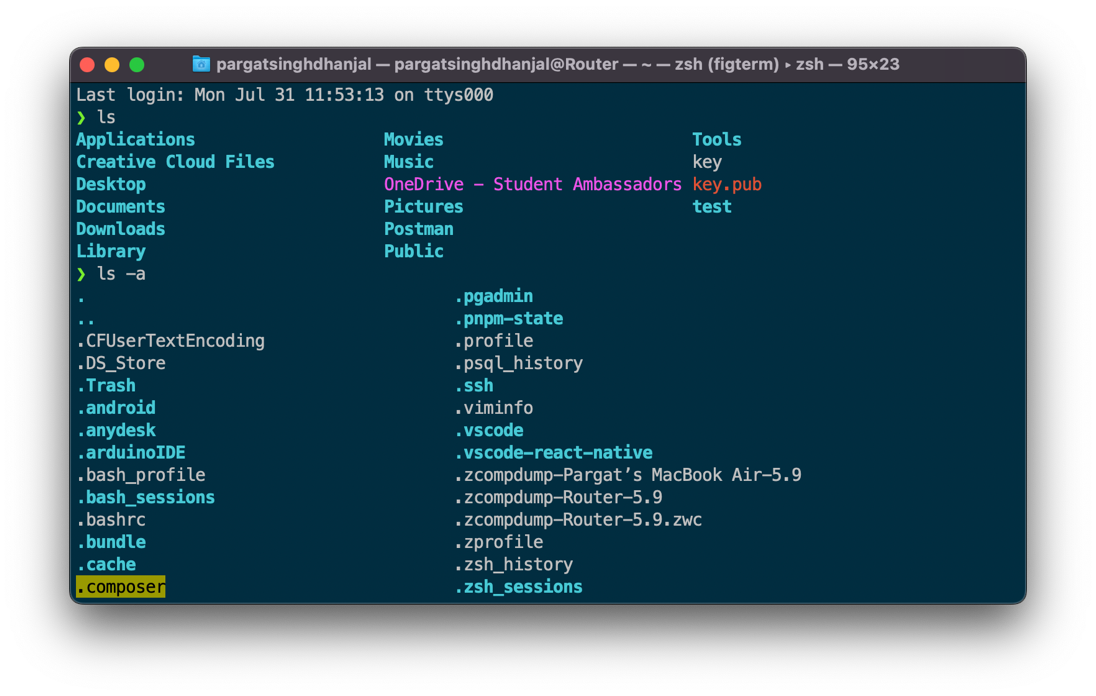
DOS commands: Attrib, dir, at, chkdsk, shutdown, tree, create a batch file, output and input redirection

Windows utilities: msconfig, defragmenter, performance monitor, task manager, registry editor, event viewer, process explorer

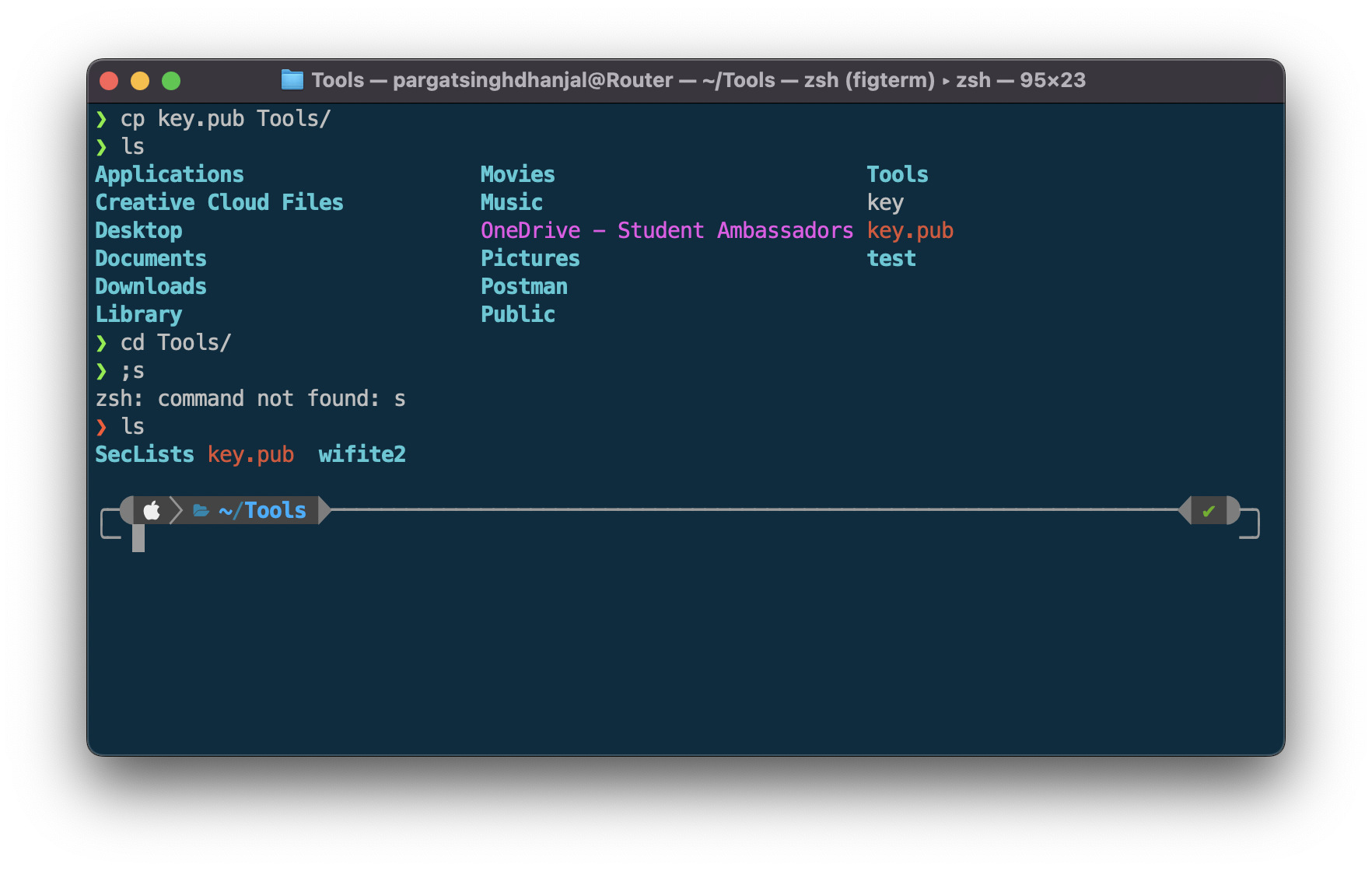
Unix Commands:

1. Unix file operations: ls, cp, rm , mv, chmod, chown ,chgrp
2. Text file operations in Unix : cat , more , less , head, tail , grep
3. Unix directory management commands : cd, pwd , ln, mkdir, rmdir
4. Unix system status commands: hostname, w, uname
5. Process management: ps, top, kill
6. Unix users commands: whoami , id, groups, passwd , who, last

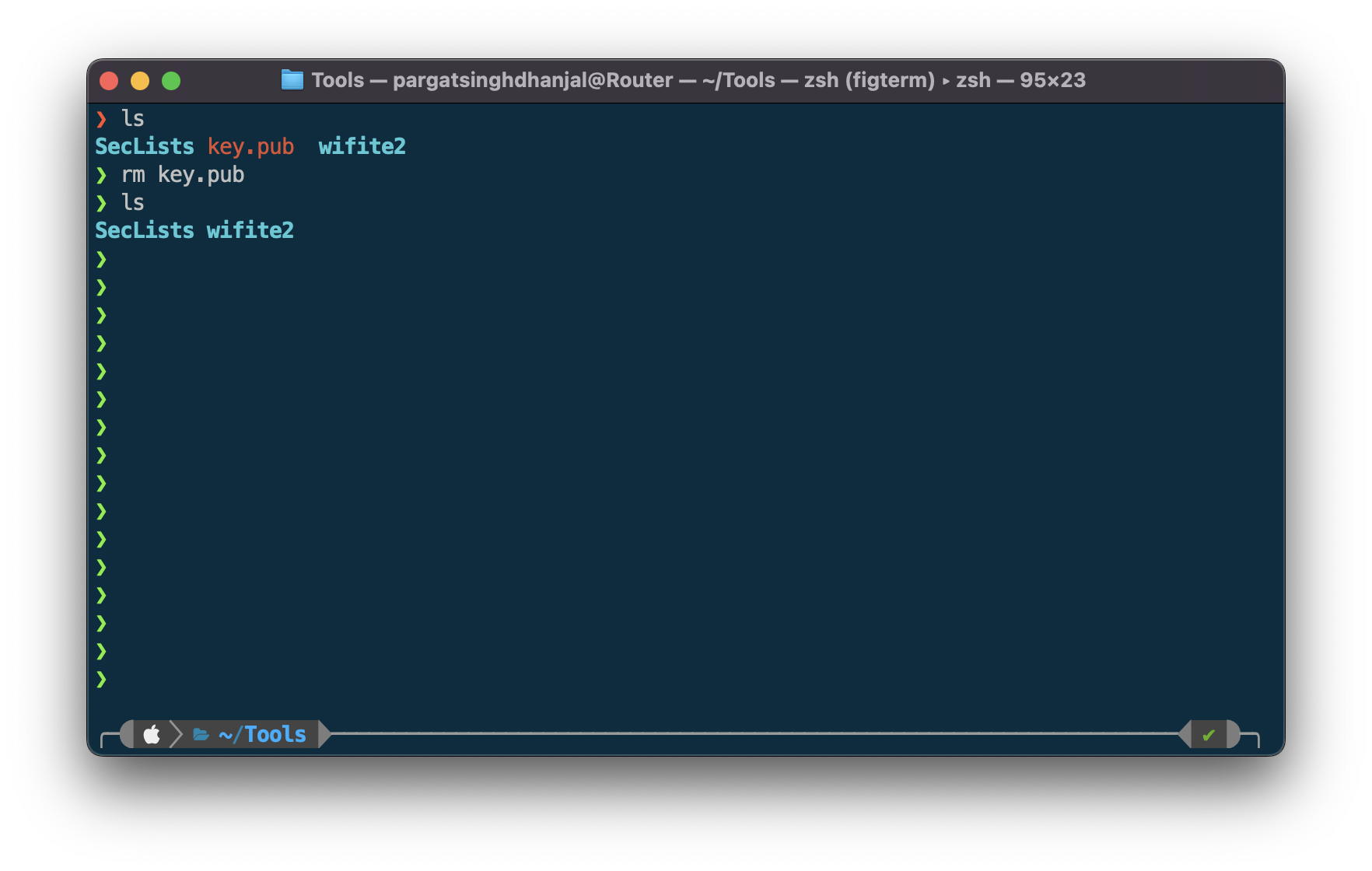
**Implementation details:**

ls: List files and directories in the current directory.

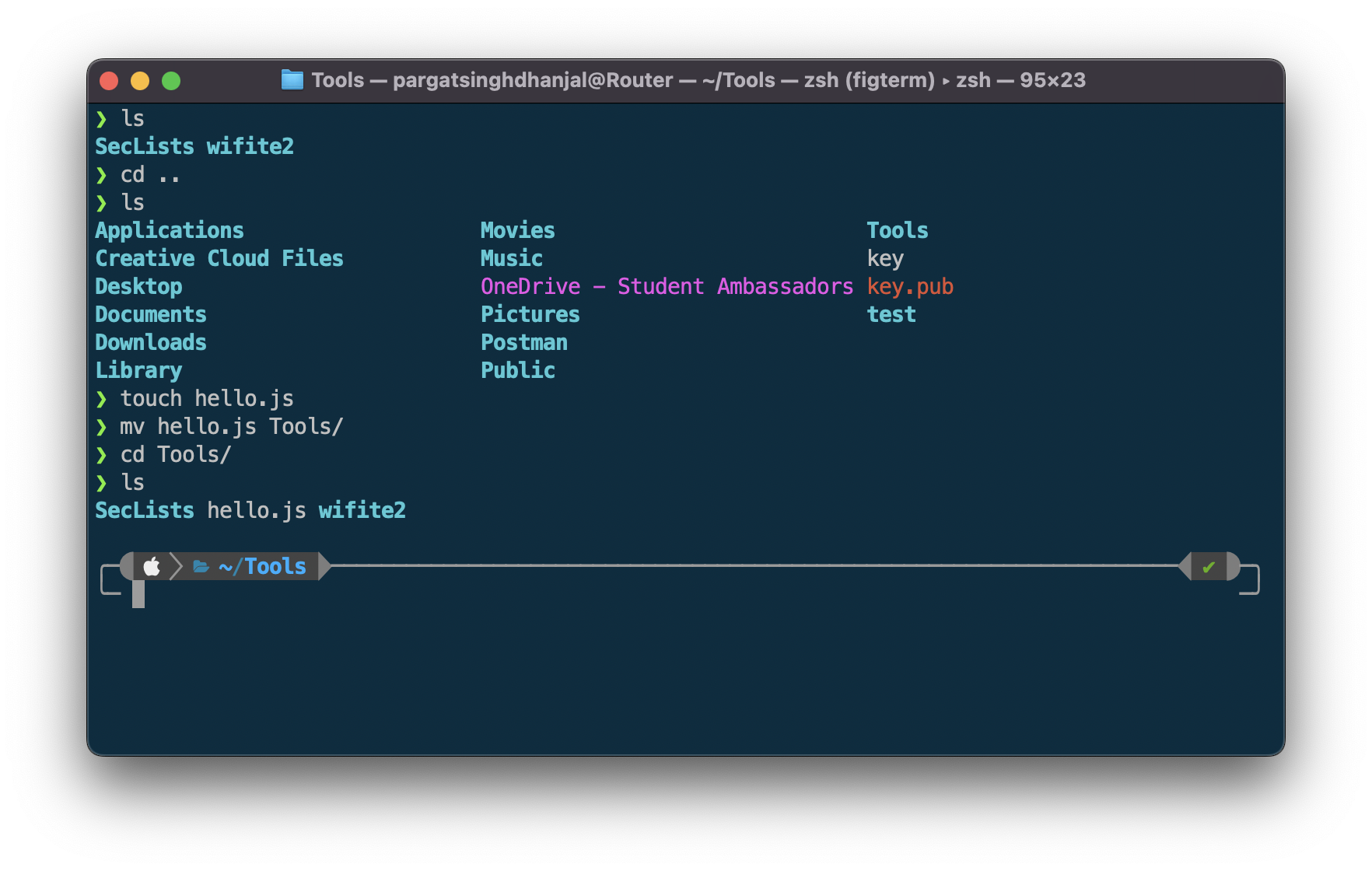
cp: Copy files or directories from one location to another.



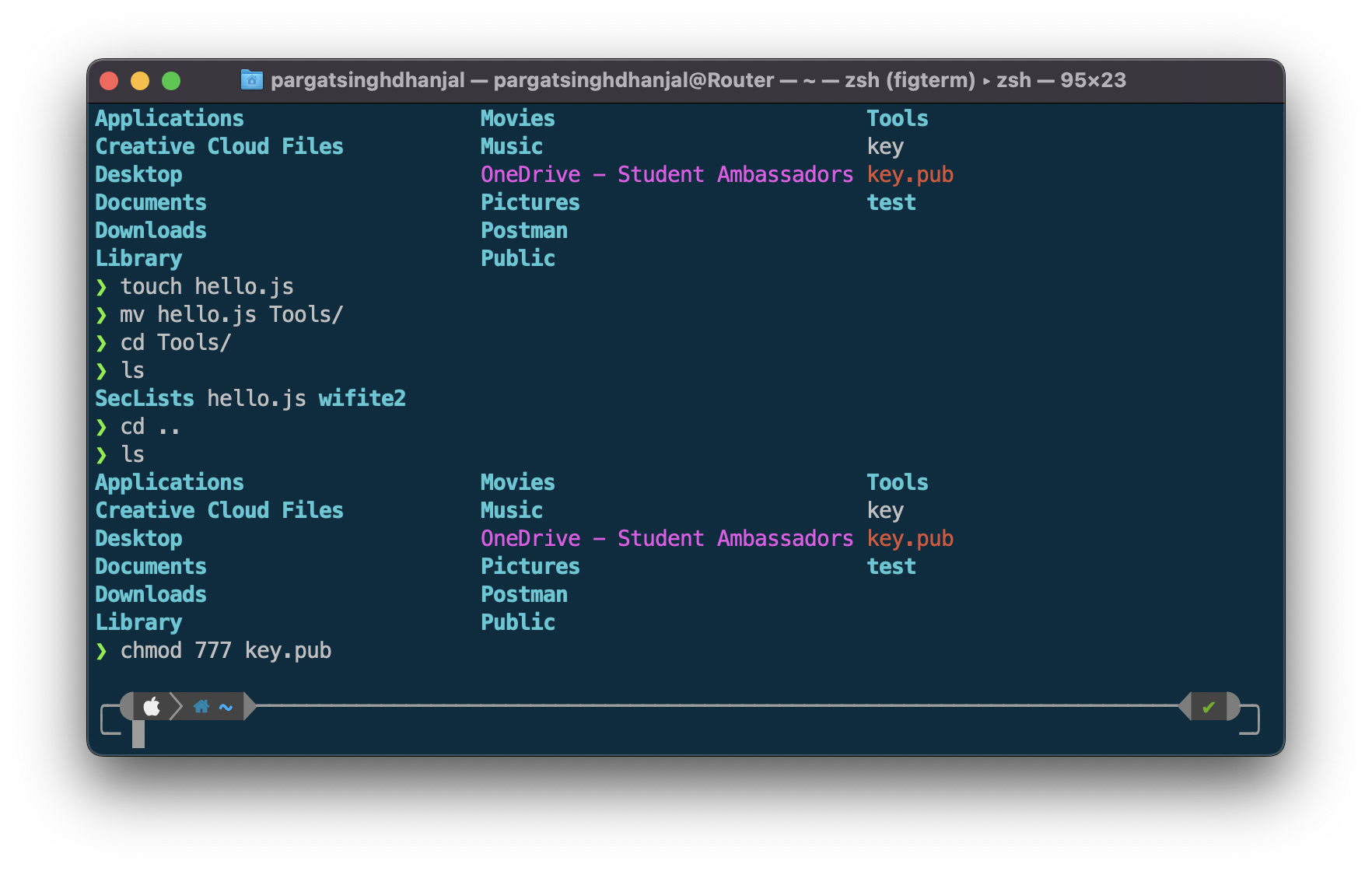
rm: Remove (delete) files or directories.



mv: Move or rename files or directories.



chmod: Change file permissions (e.g., read, write, execute) for users.



cat: Concatenate and display the contents of a file.

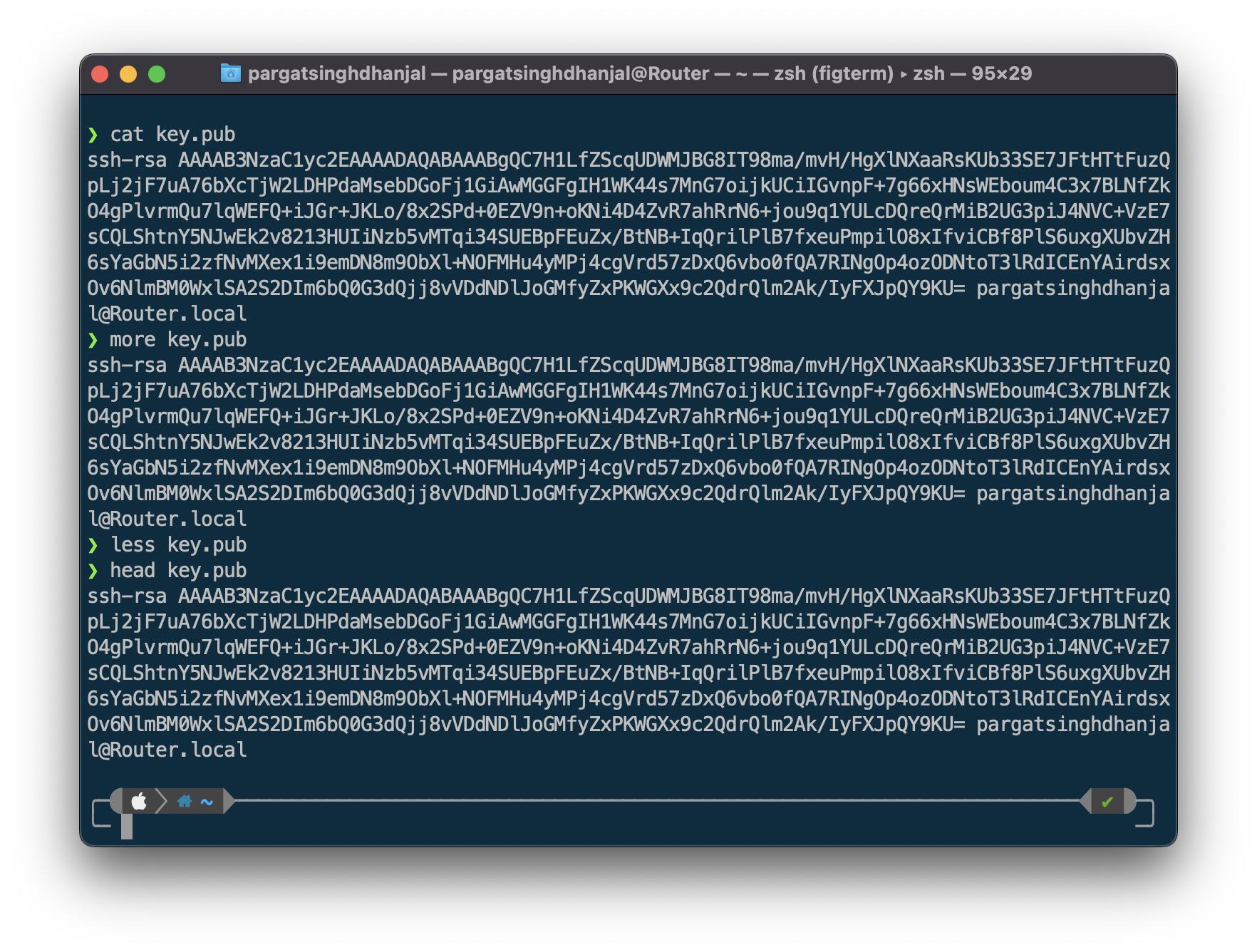
more: Display a file one screen at a time.

less: Display a file with backward navigation support.

head: Display the beginning lines of a file.

tail: Display the ending lines of a file.

grep: Search for a pattern in a file or stream and print matching lines.



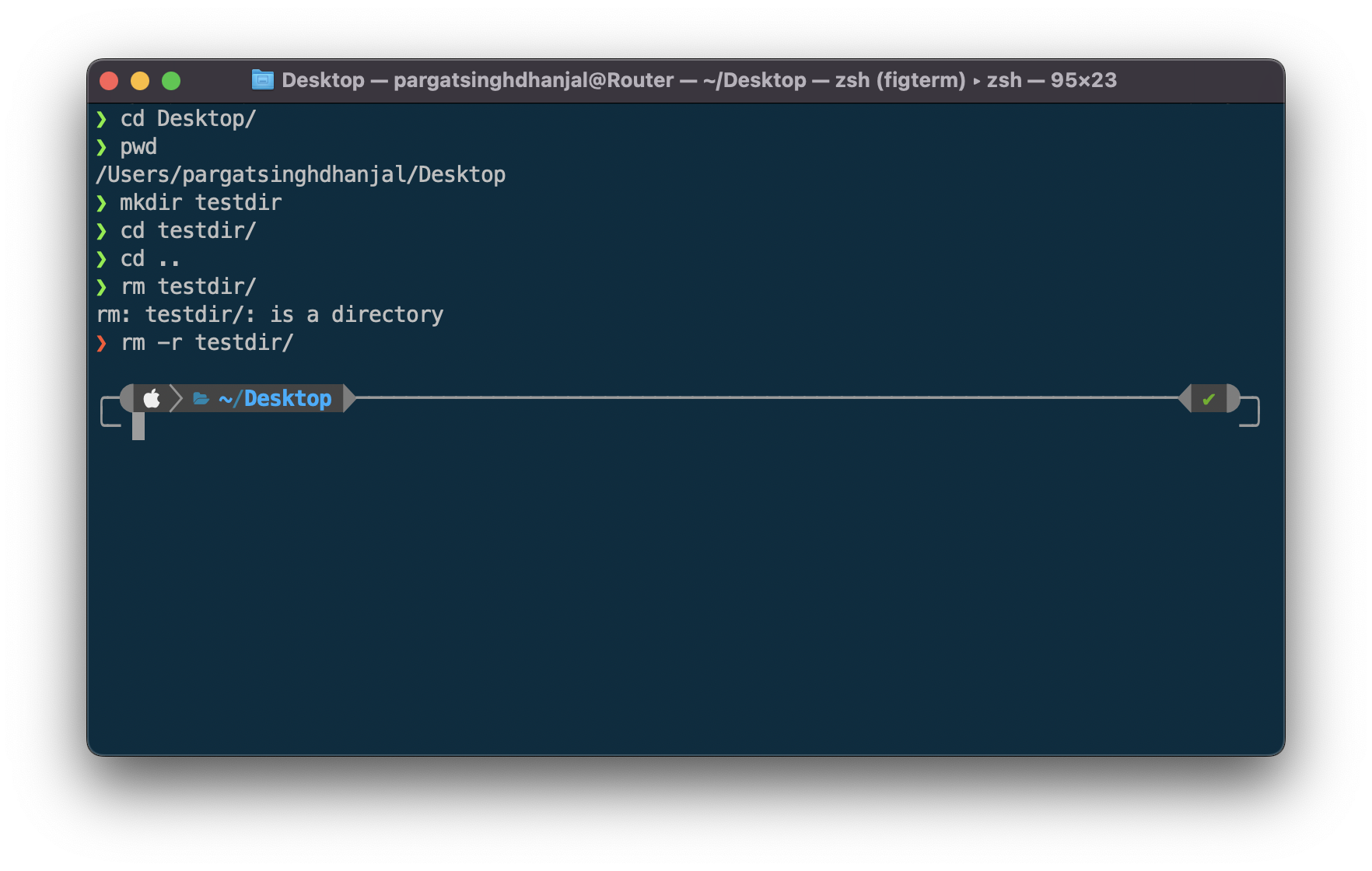
cd: Change the current working directory.

pwd: Print the current working directory.

ln: Create hard or symbolic links to files or directories.

mkdir: Create a new directory.

rmdir: Remove an empty directory.

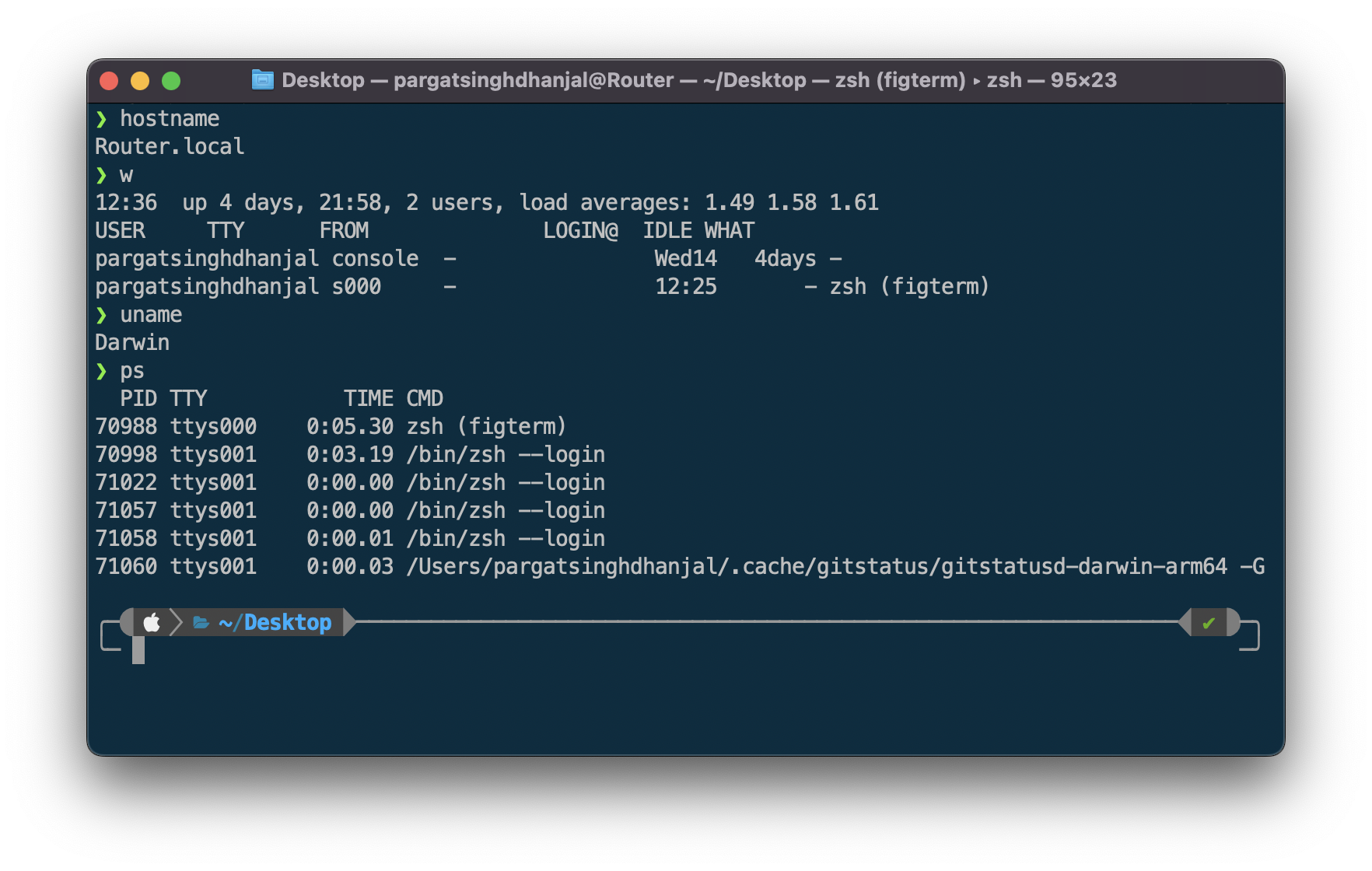


hostname: Display or set the system's host name.

w: Show who is logged on and what they are doing.

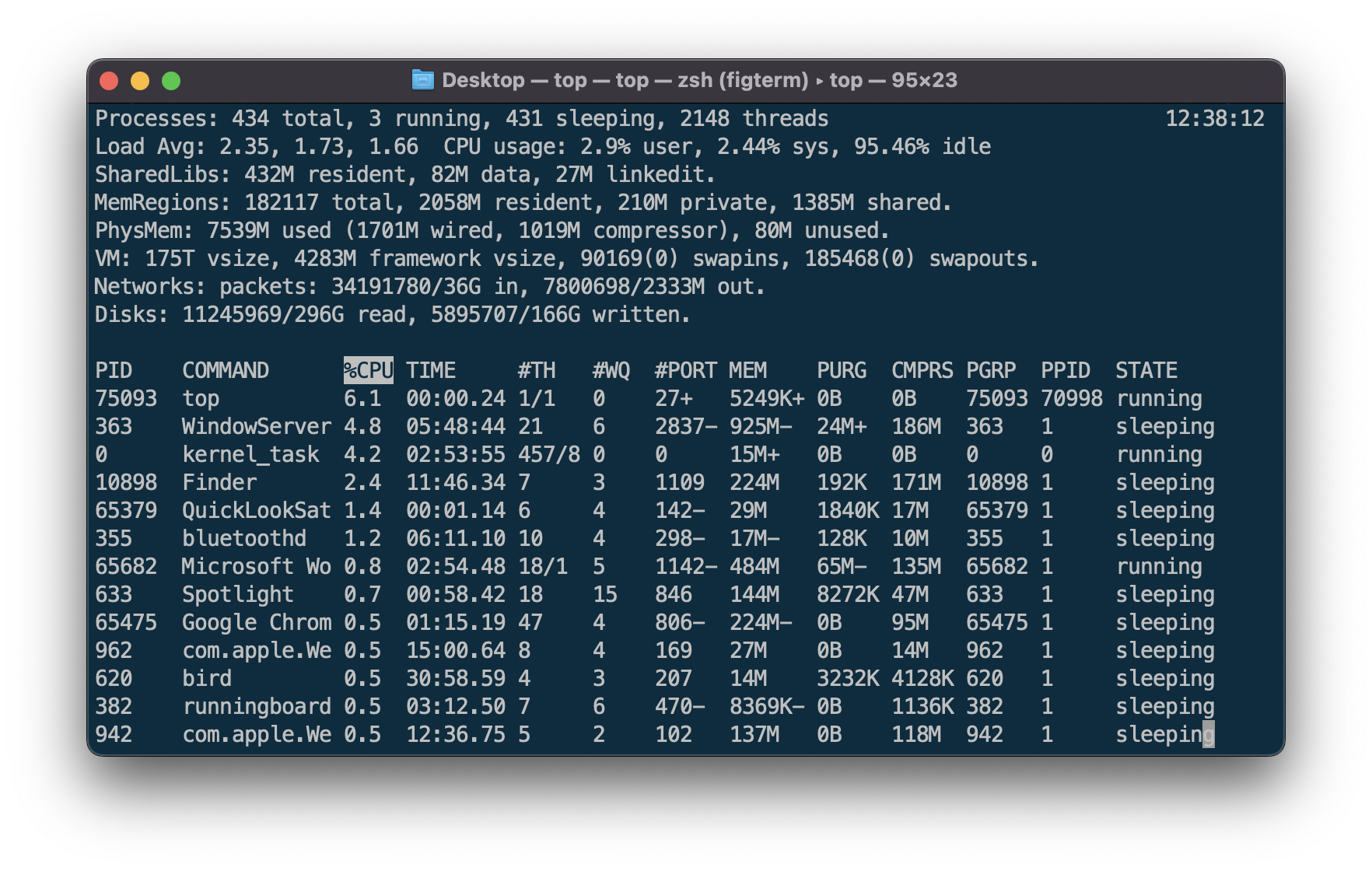
uname: Print system information (e.g., kernel name, node name).

ps: Display information about running processes.



top: Monitor and display dynamic system processes and resource usage.

kill: Send a signal to terminate a process or send specific signals to control a process.

whoami: Print the effective user ID.

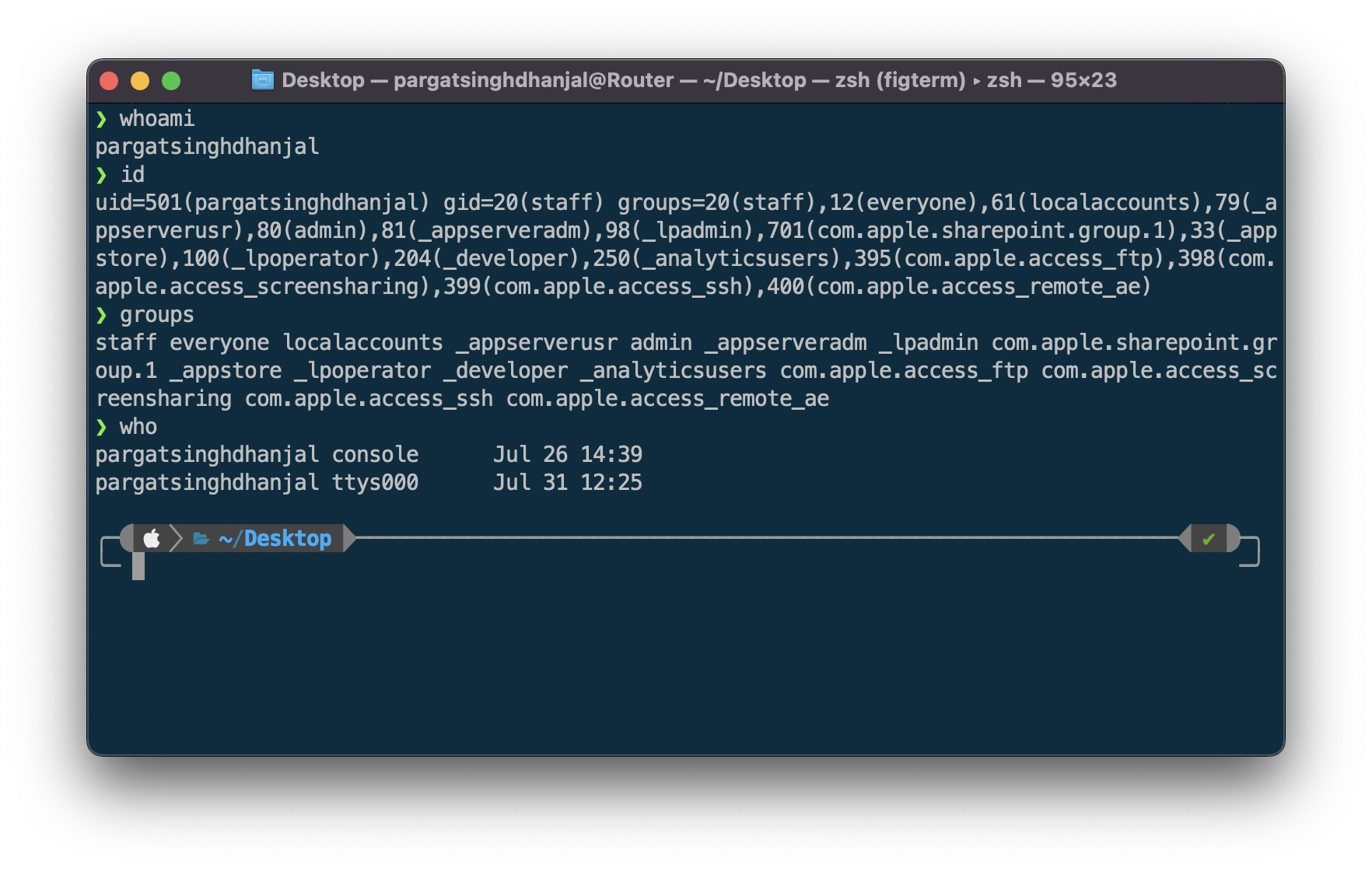
id: Display user and group ID information.

groups: Display the groups a user belongs to.

passwd: Change a user's password.

who: Show who is logged on.

last: Display recent login activity of users.



**Conclusion:**

Explored basic Commands of UNIX: Shell, Processes, Files.

**Post Lab Descriptive Questions**

1. Explain how do you read and interpret syntax of any OS command.

Reading and interpreting the syntax of any OS command involves understanding the structure and format of the command and its parameters. In most operating systems, commands follow a specific pattern: the command itself is typically followed by options or flags that modify its behavior, and these options are often preceded by a hyphen or double hyphen. Additionally, the command may require arguments that provide specific inputs or targets for the operation. To interpret a command, one must refer to the OS documentation or use the built-in help command (e.g., "man" in Unix-like systems or "help" in Windows) to understand the purpose and proper usage of each command, along with its available options and argument syntax.

1. Explain different functions of the operating systems.

Operating systems serve various functions to manage computer resources efficiently and provide a user-friendly environment. Some key functions include process management, memory management, file system management, device management, and user interface. Process management involves scheduling and controlling processes to ensure efficient utilization of the CPU. Memory management handles the allocation and deallocation of memory to running processes. File system management handles file organization and storage. Device management ensures proper communication with hardware devices like printers and disk drives. User interface allows users to interact with the system through a graphical or command-line interface, making it easier to access and manage resources.

1. What are the default permissions assigned by Unix for Directory.

In Unix-like systems, the default permissions assigned for directories are usually "rwxr-xr-x" or 755 in octal notation. This means the owner of the directory has read (r), write (w), and execute (x) permissions, while the group and others have only read and execute permissions. The read permission allows viewing the contents of the directory, write permission allows creating or deleting files within the directory, and execute permission enables access to the contents of the directory, such as listing its files.

1. Give the difference between DOS and WINDOWS.

DOS (Disk Operating System) and Windows are both operating systems, but they differ significantly in terms of architecture, features, and capabilities. DOS was a single-tasking, command-line-based operating system developed by Microsoft, primarily used during the early days of personal computing. It lacked multitasking support and a graphical user interface (GUI). Windows, on the other hand, is a multitasking, multi-user operating system with a GUI. It provides a more sophisticated and user-friendly environment, allowing users to run multiple programs simultaneously and access various hardware devices more efficiently. Windows also supports networking, making it easier to connect computers and share resources.

1. Explain Booting Process.

The booting process is the sequence of events that occur when a computer is powered on, leading to the loading of the operating system. When the computer is turned on, the Basic Input/Output System (BIOS) or Unified Extensible Firmware Interface (UEFI) performs a Power-On Self-Test (POST) to check the hardware's integrity. Afterward, the BIOS/UEFI searches for the boot loader in the system's boot devices (usually the hard drive). The boot loader (e.g., GRUB for Linux or NTLDR for older versions of Windows) then loads the kernel or core components of the operating system into memory. The kernel takes control and initializes necessary drivers and services. Finally, the user interface or desktop environment is presented, and the computer is ready for user interaction.

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Batch: A2 Roll No.: 16010121045**

**Experiment No. 02**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE:** Shell Programming and system calls |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:** To study the shell script and write the program using shell.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

**CO 1.** To introduce basic concepts and functions of operating systems.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **Silberschatz A., Galvin P., Gagne G. “Operating Systems Principles”, Willey Eight edition.**
2. **William Stallings “Operating Systems” Person, Seventh Edition**

**Edition.**

1. **Sumitabha Das “ UNIX Concepts & Applications”, McGraw Hill Second**

**Edition.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

The shell provides you with an interface to the UNIX system. It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.

**Shell Scripts**

The basic concept of a shell script is a list of commands, which are listed in the order of execution. A good shell script will have comments, preceded by a pound sign, #, describing the steps.

**Steps to create a Shell Script:**

create a file using any text editor say vi, gedit, nano etc

1.$ vi filename

2.Insert the script/ commands in file and save the file to execute the file we need to give execute permission to the file

3.$ chmod 775 filename

4.Now execute the above file using any of following methods:

$ sh filename

OR

$ ./filename

NOTE: Before adding anything to your script, you need to alert the system that a shell script is being started. This is done using the shebang construct. For example −

#!/bin/sh.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Description of the application to be implemented**:

1. Write a shell Script that accepts two file names as command line arguments and compare two file contents and check whether contents are same or not. If they are same, then delete second file.
2. Write a shell script that accepts integer and find the factorial of number.
3. Write a shell script for adding users.
4. Write a shell script for counting no of logged in users.
5. Write a shell script for counting no of processes running on system

**Implementation details:** (printout of code / screen shot)

Code:

echo "Enter a number"

read num

fact=1

*while* [ $num -gt 1 ]

*do*

fact=$((fact \* num)) *#fact = fact \* num*

num=$((num - 1)) *#num = num - 1*

*done*

echo $fact

Output:

A black background with white text

Description automatically generated

Code:

echo "Enter first file name"

read f1

echo "Enter second file name"

read f2

*if* cmp -s $f1 $f2

*then*

echo "Files are same"

rm -rf $f2

*else*

echo "Files are different"

*fi*

Output:

A screenshot of a computer

Description automatically generated

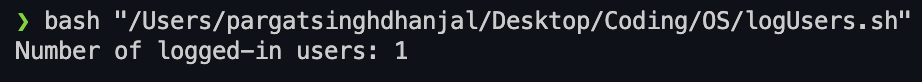
Code:

logged\_in\_users=$(who)

count=$(echo "$logged\_in\_users" | wc -l | xargs)

echo "Number of logged-in users: $count"

Output:



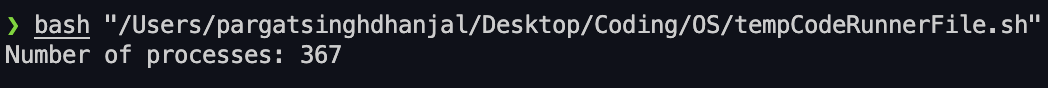
Code:

p=$(ps -A)

count=$(echo "$p" | wc -l | xargs)

echo "Number of processes: $count"

Output:



**Conclusion :**

Learnt the to write and execute bash shell scripts.

**Post Lab Descriptive Questions**

**1. What are the different types of commonly used shells on a typical linux system?**

On a typical Linux system, there are several commonly used shells. A shell is a command-line interface that allows users to interact with the operating system. Some of the popular shells include:

* **Bash (Bourne-Again Shell)**: This is the default shell for most Linux distributions. It's known for its powerful scripting capabilities and wide usage.
* **Zsh (Z Shell)**: Zsh is an extended version of Bash with additional features and customization options. It offers advanced tab completion and theming capabilities.
* **Fish (Friendly Interactive Shell)**: Fish is designed to be user-friendly and interactive. It has syntax highlighting, autocompletion, and a modern command-line experience.
* **Ksh (Korn Shell)**: Ksh is an older shell with advanced scripting features. There are different variants like ksh88 and ksh93, offering varying levels of functionality.
* **Csh (C Shell)**: Csh has a C-like syntax and features, but it's not as widely used today due to its limitations.

Each of these shells has its own set of features, advantages, and user communities.

**2. How do you find out what’s your shell?**

To find out which shell you are currently using, you can use the **echo** command along with the **$SHELL** environment variable. Open a terminal and enter the following command:

echo $SHELL

This will display the path to the shell you're currently using, such as **/bin/bash**, **/usr/bin/zsh**, etc.

**3. List the advantages and disadvantages of shell scripting.**

**Advantages:**

* **Automation**: Shell scripting allows you to automate repetitive tasks, making system administration and maintenance more efficient.
* **Rapid Development**: Shell scripts are generally quick to write and test, making them useful for creating small utilities or scripts on-the-fly.
* **Accessibility**: Shell scripting provides a powerful command-line interface that can be accessed remotely, which is beneficial for remote administration and scripting tasks.
* **Integration**: Shell scripts can easily interact with system utilities and other command-line tools, allowing seamless integration with existing software.
* **Customization**: Shells like Bash offer a range of features like variables, functions, and conditional statements, enabling you to create complex and customized scripts.

**Disadvantages:**

* **Limited Performance**: For resource-intensive tasks, interpreted shell scripts may have performance issues compared to compiled languages.
* **Complexity**: As scripts grow in size and complexity, maintaining and debugging them can become challenging.
* **Portability**: Shell scripts can be dependent on specific shell features, making them less portable between different shell environments.
* **Security**: Poorly written shell scripts can have security vulnerabilities if not properly sanitized, potentially leading to system compromises.
* **Lack of GUI**: Shell scripts operate in a command-line environment, so they may not be suitable for tasks requiring a graphical user interface.

Despite these disadvantages, shell scripting remains a powerful tool for various system administration and automation tasks, especially when used judiciously and with proper consideration for security and performance.

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Batch: Roll No.:**

**Experiment No. 03**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE:** Implementation of Basic Process management algorithms – Non Pre-emptive ( FCFS , SJF, priority) |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:** To implement basic Non –Pre-emptive Process management algorithms ( FCFS , SJF , Priority)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

**CO 2.** To understand the concept of process, thread and resource management.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **Silberschatz A., Galvin P., Gagne G. “Operating Systems Principles”, Willey Eight edition.**
2. **Achyut S. Godbole , Atul Kahate “Operating Systems” McGraw Hill Third**

**Edition.**

1. **William Stallings, “Operating System Internal & Design Principles”, Pearson.**
2. **Andrew S. Tanenbaum, “Modern Operating System”, Prentice Hall.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

Most systems have a large number of processes with short CPU bursts interspersed between I/O requests and a small number of processes with long CPU bursts. To provide good time-sharing performance, we may preempt a running process to let another one run. The ready list, also known as a run queue, in the operating system keeps a list of all processes that are ready to run and not blocked on some I/O or other system request, such as a semaphore. Then entries in this list are pointers to the process control block, which stores all information and state about a process.

When an I/O request for a process is complete, the process moves from the *waiting* state to the *ready* state and gets placed on the run queue.

The process scheduler is the component of the operating system that is responsible for deciding whether the currently running process should continue running and, if not, which process should run next. There are four events that may occur where the scheduler needs to step in and make this decision:

1. The current process goes from the *running* to the *waiting* state because it issues an I/O request or some operating system request that cannot be satisfied immediately.
2. The current process terminates.
3. A timer interrupt causes the scheduler to run and decide that a process has run for its allotted interval of time and it is time to move it from the *running* to the *ready* state.
4. An I/O operation is complete for a process that requested it and the process now moves from the *waiting* to the*ready* state. The scheduler may then decide to preempt the currently-running process and move this *ready* process into the *running* state.

The decisions that the scheduler makes concerning the sequence and length of time that processes may run is called the scheduling algorithm (or scheduling policy). These decisions are not easy ones, as the scheduler has only a limited amount of information about the processes that are ready to run. A good scheduling algorithm should:

1. Be fair – give each process a fair share of the CPU, allow each process to run in a reasonable amount of time.
2. Be efficient – keep the CPU busy all the time.
3. Maximize throughput – service the largest possible number of jobs in a given amount of time; minimize the amount of time users must wait for their results.
4. Minimize response time – interactive users should see good performance
5. Minimize overhead – don’t waste too many resources. Keep scheduling time and context switch time at a minimum.
6. Maximize resource use – favor processes that will use underutilized resources. There are two motives for this. Most devices are slow compared to CPU operations. We’ll achieve better system throughput by keeping devices busy as often as possible. The second reason is that a process may be holding a key resource and other, possibly more important, processes cannot use it until it is released. Giving the process more CPU time may free up the resource quicker.
7. Avoid indefinite postponement – every process should get a chance to run eventually.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Description of the application to be implemented**:

**First-Come, First-Served Scheduling:**

# Shortest job first :

# Priority scheduling

**Implementation details:** (printout of code)

**Conclusion:**

**Post Lab Objective Questions**

* 1. What is the ready state of a process?  
     a) when process is scheduled to run after some execution  
     b) when process is unable to run until some task has been completed  
     c) when process is using the CPU  
     d) none of the mentioned

**Ans:**

* 1. A process stack does not contain  
     a) function parameters  
     b) local variables  
     c) return addresses  
     d) PID of child process

**Ans:**

* 1. A process can be terminated due to  
     a) normal exit  
     b) fatal error  
     c) killed by another process  
     d) all of the mentioned

**Ans:**

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Batch: Roll No.:**

**Experiment No. 04**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE:** Implementation of Basic Process management algorithms - Preemptive (SRTN, RR, priority ) |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:** To implement basic Process management algorithms ( Round Robin,SRTN, Priority)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

**CO 2.** To understand the concept of process, thread and resource management.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **Silberschatz A., Galvin P., Gagne G. “Operating Systems Principles”, Willey Eight edition.**
2. **Achyut S. Godbole , Atul Kahate “Operating Systems” McGraw Hill Third**

**Edition.**

1. **William Stallings, “Operating System Internal & Design Principles”, Pearson.**
2. **Andrew S. Tanenbaum, “Modern Operating System”, Prentice Hall.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

Most systems have a large number of processes with short CPU bursts interspersed between I/O requests and a small number of processes with long CPU bursts. To provide good time-sharing performance, we may preempt a running process to let another one run. The ready list, also known as a run queue, in the operating system keeps a list of all processes that are ready to run and not blocked on some I/O or other system request, such as a semaphore. Then entries in this list are pointers to the process control block, which stores all information and state about a process.

When an I/O request for a process is complete, the process moves from the *waiting* state to the *ready* state and gets placed on the run queue.

The process scheduler is the component of the operating system that is responsible for deciding whether the currently running process should continue running and, if not, which process should run next. There are four events that may occur where the scheduler needs to step in and make this decision:

1. The current process goes from the *running* to the *waiting* state because it issues an I/O request or some operating system request that cannot be satisfied immediately.
2. The current process terminates.
3. A timer interrupt causes the scheduler to run and decide that a process has run for its allotted interval of time and it is time to move it from the *running* to the *ready* state.
4. An I/O operation is complete for a process that requested it and the process now moves from the *waiting* to the*ready* state. The scheduler may then decide to preempt the currently-running process and move this *ready* process into the *running* state.

The decisions that the scheduler makes concerning the sequence and length of time that processes may run is called the scheduling algorithm (or scheduling policy). These decisions are not easy ones, as the scheduler has only a limited amount of information about the processes that are ready to run. A good scheduling algorithm should:

1. Be fair – give each process a fair share of the CPU, allow each process to run in a reasonable amount of time.
2. Be efficient – keep the CPU busy all the time.
3. Maximize throughput – service the largest possible number of jobs in a given amount of time; minimize the amount of time users must wait for their results.
4. Minimize response time – interactive users should see good performance
5. Minimize overhead – don’t waste too many resources. Keep scheduling time and context switch time at a minimum.
6. Maximize resource use – favor processes that will use underutilized resources. There are two motives for this. Most devices are slow compared to CPU operations. We’ll achieve better system throughput by keeping devices busy as often as possible. The second reason is that a process may be holding a key resource and other, possibly more important, processes cannot use it until it is released. Giving the process more CPU time may free up the resource quicker.
7. Avoid indefinite postponement – every process should get a chance to run eventually.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Description of the application to be implemented**:

**Round Robin Algorithm**

# Shortest Remaining Time First Algorithm :

# Priority scheduling:

**Implementation details:** (printout of code)

**Conclusion:**

**Post Lab Descriptive Questions**

1. Consider three processes, all arriving at time zero, with total execution time of 10, 20 and 30 units, respectively. Each process spends the first 20% of execution time doing I/O, the next 70% of time doing computation, and the last 10% of time doing I/O again. The operating system uses a shortest remaining compute time first scheduling algorithm and schedules a new process either when the running process gets blocked on I/O or when the running process finishes its compute burst. Assume that all I/O operations can be overlapped as much as possible. For what percentage of time does the CPU remain idle?
2. What is Starvation?

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Batch: Roll No.:**

**Experiment No. 05**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE:** Implementation of Process synchronization algorithms using semaphore - producer consumer problem , reader-writers problem |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:** Implementation of Process synchronization algorithms using semaphore - producer consumer problem, reader-writers problem

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

**CO 3.** To understand the concepts of process synchronization and deadlock.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **Silberschatz A., Galvin P., Gagne G. “Operating Systems Principles”, Willey Eight edition.**
2. **Achyut S. Godbole , Atul Kahate “Operating Systems” McGraw Hill Third**

**Edition.**

1. **William Stallings, “Operating System Internal & Design Principles”, Pearson.**
2. **Andrew S. Tanenbaum, “Modern Operating System”, Prentice Hall.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

Knowledge of Concurrency, Mutual Exclusion, Synchronization, Deadlock, Starvation.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Description of the chosen process synchronization algorithm:

**Implementation details:** (printout of code)

**Conclusion:**

**Post Lab Objective Questions**

1. A semaphore is a shared integer variable
   1. That can’t drop below zero
   2. That can’t be more than
   3. That can’t drop below one

**Ans:**

1. Mutual exclusion can be provided by the
   1. Mute locks
   2. Binary semaphores
   3. Both a and b
   4. None of these

**Ans:**

1. A monitor is a module that encapsulates
   1. Shared data structures
   2. Procedures that operate on shared data structure
   3. Synchronization between concurrent procedure invocation
   4. All of the above

**Ans:**

1. To enable a process to wait within the monitor
   1. A condition variable must be declared as condition
   2. Condition Variables must be used as Boolean objects
   3. Semaphore must be used
   4. All of the above

**Ans:**

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Batch:**

**Roll No.:**

**Experiment No. 06**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE:** Implementation of dining philosopher problem using threads. |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:** Implementation of Process synchronization algorithms using threads – Dining Philosopher problem

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

**CO 2.** To understand the concept of process, thread and resource management.

**CO 3.** To understand the concepts of process synchronization and deadlock.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **Silberschatz A., Galvin P., Gagne G. “Operating Systems Principles”, Willey Eight edition.**
2. **Achyut S. Godbole , Atul Kahate “Operating Systems” McGraw Hill Third**

**Edition.**

1. **William Stallings, “Operating System Internal & Design Principles”, Pearson.**
2. **Andrew S. Tanenbaum, “Modern Operating System”, Prentice Hall.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

Knowledge of Concurrency, Mutual Exclusion, Synchronization, Deadlock, Starvation,threads.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Description of the chosen process synchronization algorithm:

**Implementation details:** (printout of code)

**Conclusion:**

**Post Lab Descriptive Questions**

1.Differentiate between a monitor, semaphore and a binary semaphore?

2. Define clearly the dining-philosophers problem?

3. Identify the scenarios in the dining-philosophers problem that leads to the deadlock situations?

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Batch:**

**Roll No.:**

**Experiment No. 07**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE: Simulate Bankers Algorithm for Deadlock Avoidance** |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:** Implementation of Banker’s Algorithm for Deadlock Avoidance

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

**CO 3.** To understand the concepts of process synchronization and deadlock.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **Silberschatz A., Galvin P., Gagne G. “Operating Systems Principles”, Willey Eight edition.**
2. **Achyut S. Godbole , Atul Kahate “Operating Systems” McGraw Hill Third**

**Edition.**

1. **William Stallings, “Operating System Internal & Design Principles”, Pearson.**
2. **Andrew S. Tanenbaum, “Modern Operating System”, Prentice Hall.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

Knowledge of deadlocks and all deadlock avoidance methods.

**Description of the application to be implemented**:

The Banker's algorithm is a [resource allocation](http://en.wikipedia.org/wiki/Resource_allocation) and [deadlock](http://en.wikipedia.org/wiki/Deadlock) avoidance [algorithm](http://en.wikipedia.org/wiki/Algorithm) developed by [Edsger Dijkstra.](http://en.wikipedia.org/wiki/Edsger_Dijkstra)

# DATA STRUCTURES

(where *n* is the number of processes in the system and *m* is the number of resource types)

**Implementation details:** (printout of code)

**Conclusion:**

**Post Lab Objective Questions**

1. The wait-for graph is a deadlock detection algorithm that is applicable when:
   1. All resources have a single instance
   2. All resources have multiple instances
   3. Both a and b
   4. None of the above

**Ans:**

1. Resources are allocated to the process on non-sharable basis is \_
   1. Hold and Wait
   2. Mutual Exclusion
   3. No pre-emption
   4. Circular Wait

**Ans:**

1. Which of the following approaches require knowledge of the system state?
   1. Deadlock Detection
   2. Deadlock Prevention
   3. Deadlock Avoidance
   4. All of the above

**Ans:**

1. Consider a system having ‘m’ resources of the same type. These resources are shared by 3 processes A, B, C which have peak time demands of 3, 4, 6 respectively. The minimum value of ‘m’ that ensures that deadlock will never occur is

|  |  |
| --- | --- |
| a) | 11 |
| b) | 12 |
| c) | 13 |
| d)  A | 14 |
| **Ans:** |  |

**Post Lab Descriptive Questions**

1. Consider a system with total of 150 units of memory allocated to three processes as shown:

|  |  |  |
| --- | --- | --- |
| **Process** | **Max** | **Hold** |
| P1 | 70 | 45 |
| P2 | 60 | 40 |
| P3 | 60 | 15 |

Apply Banker’s algorithm to determine whether it would be safe to grant each of the following request. If yes, indicate sequence of termination that could be possible.

1. The P4 process arrives with max need of 60 and initial need of 25 units.
2. The P4 process arrives with max need of 60 and initial need of 35 units.

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Batch:**

**Roll No.:**

**Experiment No. 08**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE: Disk Scheduling Algorithms** |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:** Implementation of Disk Scheduling Algorithm like FCFS, SSTF, SCAN, CSCAN, LOOK.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

**CO 4.** To understand various Memory, I/O and File management techniques.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **Silberschatz A., Galvin P., Gagne G. “Operating Systems Principles”, Willey Eight edition.**
2. **Achyut S. Godbole , Atul Kahate “Operating Systems” McGraw Hill Third**

**Edition.**

1. **William Stallings, “Operating System Internal & Design Principles”, Pearson.**
2. **Andrew S. Tanenbaum, “Modern Operating System”, Prentice Hall.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

* Knowledge of disk scheduling algorithm.
* Calculation of seek time and transfer time etc.

**Description of the application to be implemented**:

First Come-First Serve (FCFS):

Shortest Seek Time First (SSTF):

Elevator (SCAN):

CSAN:

LOOK:

**Implementation details:** (printout of code)

**Conclusion**:

**Post Lab Descriptive Questions**

1. A disk drive has 200 cylinders numbered from 0 to 199. The disk head is initially at cylinder 53. The queue of pending requests in FIFO order is :

98, 183, 37, 122, 14, 124, 65, 67.

Starting from the current head position, what is the total distance travelled (in cylinders) by disk arm to satisfy the requests using CSCAN and Look. Illustrate with figures in each case.

**Post Lab Objective Questions**

1. In a hard disk, what rotates about a central spindle
   1. Disk
   2. Platter
   3. Sector
   4. None of the above

**Ans:**

1. The time required to move the disk arm to the required track is known as
   1. Latency time
   2. Access time
   3. Seek time
   4. None of the above

**Ans:**

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Batch:**

**Roll No.:**

**Experiment No. 09**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE:** Implementation of Memory Management Using Address Translation |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:** To understand process of Address Translation in Memory

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

**CO 5.** Understand Storage management with allocation, segmentation & virtual memory concepts

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **Silberschatz A., Galvin P., Gagne G. “Operating Systems Principles”, Willey Eight edition.**
2. **Achyut S. Godbole , Atul Kahate “Operating Systems” McGraw Hill Third**

**Edition.**

1. **William Stallings, “Operating System Internal & Design Principles”, Pearson.**
2. **Andrew S. Tanenbaum, “Modern Operating System”, Prentice Hall.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

Knowledge about the types of memory

**Stepwise-Procedure:**

**Implementation details (Printout of code)**

**Output:**

**Conclusion:**

**Post Lab Descriptive Questions**

**Post Lab Objective Questions**

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Batch:**

**Roll No.:**

**Experiment No. 10**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE:** Implementation of Memory Allocation Algorithms |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:** To learn about various Memory Allocation Algorithms

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

**CO 5.** Understand Storage management with allocation, segmentation & virtual memory concepts

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **Silberschatz A., Galvin P., Gagne G. “Operating Systems Principles”, Willey Eight edition.**
2. **Achyut S. Godbole , Atul Kahate “Operating Systems” McGraw Hill Third**

**Edition.**

1. **William Stallings, “Operating System Internal & Design Principles”, Pearson.**
2. **Andrew S. Tanenbaum, “Modern Operating System”, Prentice Hall.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

Knowledge of types of memory

**Stepwise-Procedure:**

**Algorithms Used-**

**Implementation details:** (printout of code)

**Output-**

**Conclusion:**

**Post Lab Subjective Questions**

**Post Lab objective Questions**

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**