| Practical 1 | |
| --- | --- |
| Aim: | Installation of Operating System. |
| Questionnaires: | 1. **List out the various ways to install operating system and explain them**   **Ans :**   * DVD/CD Installation * USB Installation * Network Installation * Pre-Installed Image * Online/Cloud Installation * Dual Boot Installation * Upgrade Installation  1. **From given ways which one is better to use for specific applications.**   **Ans :**   * DVD/CD Installation: This method is useful when you have an OS installation disc and a computer with an optical drive * USB Installation: This method is versatile and widely used because most computers have USB ports. * Network Installation: Network installations are beneficial in enterprise environments or scenarios where you need to deploy OSs to multiple machines simultaneously. * Pre-Installed Image: Pre-installed images are convenient when purchasing a new computer or when an OS comes bundled with specific hardware. * Online/Cloud Installation: Online installations are useful when you want to quickly provision virtual machines or use cloud-based services. * Dual Boot Installation: Dual booting is ideal if you need to run multiple operating systems on the same computer. * Upgrade Installation: Upgrade installations are suitable when you want to update your existing OS to a newer version.  1. **Which operating system is used for various tasks and applications ? Mention when to use which operating system with its advantages and disadvantages**   **Ans :**  Here are some popular operating systems and their common use cases:  1. Windows:  - Use Windows when: You require broad software compatibility, gaming, business productivity, or specific applications that are primarily available on Windows.  - Advantages: Windows has a vast software library, extensive hardware support, and a user-friendly interface. It's widely used in the business world and offers compatibility with a wide range of applications and devices.  - Disadvantages: Windows can be susceptible to malware and viruses, and its performance may degrade over time. It's also a proprietary OS, which means you may have limited control over certain aspects of the system.  2. macOS:  - Use macOS when: You are working with creative applications, require a seamless integration with other Apple devices, or prefer a UNIX-based environment with a polished user interface.  - Advantages: macOS offers a sleek and intuitive user experience, excellent integration with Apple hardware and devices, and is favored by professionals in creative industries. It provides a robust UNIX foundation and strong security features.  - Disadvantages: macOS is limited to Apple hardware, which can be more expensive compared to other options. Software compatibility may be limited in certain areas, and gaming options are relatively smaller compared to Windows.  3. Linux:  - Use Linux when: You require a highly customizable and flexible OS, prefer open-source software, or need a stable and secure platform for development, servers, or specialized applications.  - Advantages: Linux is highly customizable, secure, and stable. It offers a wide variety of distributions (such as Ubuntu, Fedora, and CentOS) tailored for different use cases. It excels in server environments and is the preferred choice for developers and enthusiasts. Additionally, Linux distributions are often free to use.  - Disadvantages: Linux can have a steeper learning curve for beginners, and some hardware may have limited driver support. It may also require manual configuration and troubleshooting. Software availability can vary, and certain specialized applications may be designed for Windows or macOS.  4. Chrome OS:  - Use Chrome OS when: You primarily use web-based applications, require a simple and affordable option for basic computing tasks, or prefer a lightweight and secure OS.  - Advantages: Chrome OS offers fast boot times, seamless integration with Google services, automatic updates, and strong security. It's designed around web applications and is often found in Chromebooks, which are affordable and portable.  - Disadvantages: Chrome OS heavily relies on internet connectivity and web applications, which may limit functionality when offline. It may have limited support for specialized software or tasks that require high-performance hardware.   1. **Compare virtual machine vmware and hyper v. Based on comparison list out specific usage for both**   **Ans :**  VMware and Hyper-V are both popular virtualization platforms used for creating and managing virtual machines (VMs) on host systems. Here's a comparison of the two:  Manufacturer:  VMware: Developed by VMware, a subsidiary of Dell Technologies.  Hyper-V: Developed by Microsoft as a component of Windows Server and Windows client operating systems.  Host OS Compatibility:  VMware: Supports a wide range of host operating systems, including Windows, Linux, and macOS.  Hyper-V: Primarily designed for Windows-based hosts, such as Windows Server and Windows 10 Pro/Enterprise.  Licensing:  VMware: Requires separate licenses for VMware vSphere and additional features like vCenter Server for advanced management.  Hyper-V: Included as a free feature in Windows Server editions and Windows 10 Pro/Enterprise.   1. **List out the application for each way of installation of os.**   **Ans :**  Here are some common applications for each method of operating system installation:  1. DVD/CD Installation:  - Installing Windows 10 from a retail DVD or CD.  - Installing Linux distributions like Ubuntu or Fedora from installation discs.  - Installing macOS from a macOS installation DVD.  2. USB Installation:  - Creating a bootable USB drive to install Windows 10, Linux distributions, or macOS.  - Installing lightweight Linux distributions like Puppy Linux or Lubuntu on older computers without DVD drives.  - Creating a Windows To Go USB drive for portable Windows installations.  3. Network Installation:  - Enterprise-level deployments of Windows Server or Linux server distributions using network-based installation tools like Windows Deployment Services (WDS) or Preboot Execution Environment (PXE).  - Installing Linux distributions like CentOS or Debian via network boot and installation servers.  - Automated OS installations in data centers or large-scale computing environments.  4. Pre-Installed Image:  - Purchasing a new laptop or desktop computer with Windows, macOS, or Linux pre-installed by the manufacturer.  - Acquiring pre-installed images of virtual machines from cloud service providers like Amazon Web Services (AWS) or Microsoft Azure.  5. Online/Cloud Installation:  - Creating virtual machines with specific operating systems in cloud platforms like AWS, Azure, or Google Cloud Platform (GCP).  - Provisioning cloud-based instances with pre-configured OS images, such as Ubuntu Server or Windows Server.  - Deploying containerized applications with predefined OS images using platforms like Docker or Kubernetes.  6. Dual Boot Installation:  - Installing multiple operating systems on a computer to choose between them during startup.  - Setting up a dual boot configuration with Windows and Linux distributions like Ubuntu or Fedora.  - Dual booting macOS and Windows on an Apple computer using Boot Camp.  7. Upgrade Installation:  - Upgrading an existing Windows installation to a newer version, such as upgrading from Windows 7 to Windows 10.  - Updating macOS to the latest version available via the Mac App Store.  - Applying updates and patches to Linux distributions using package managers like apt, yum, or dnf.   1. **How we can convert windows terminal to linux command prompt**   **Ans**  To convert the Windows Terminal into a Linux terminal-like experience, we can follow these steps:  1. Install Windows Subsystem for Linux (WSL):  - Open PowerShell or Command Prompt as an administrator.  - Run the command `wsl --install` to enable WSL and install the necessary components.  2. Install a Linux distribution:  - Open the Microsoft Store.  - Search for a Linux distribution of your choice, such as Ubuntu, Debian, or Fedora.  - Select the desired Linux distribution and click "Install" to download and install it.  3. Launch the Linux distribution:  - Open the Windows Terminal application.  - Click on the "+" button in the tab bar to open a new terminal tab.  - In the drop-down menu, select the installed Linux distribution to start a new terminal session with that distribution.  4. Customize the Windows Terminal:  - Open the Windows Terminal application.  - Click on the downward arrow icon in the title bar and select "Settings" to open the settings.json file.  - Customize the settings to modify the appearance, behavior, and key bindings to match your preferences. For example, you can change the color scheme, font, or shortcut keys.  5. Install Linux terminal applications:  - Once you have your Linux distribution running in the Windows Terminal, you can install and use Linux terminal applications just like you would on a native Linux system.   1. **Why to use virtualization**   **Ans:**  Virtualization offers several benefits and use cases that make it a valuable technology in various scenarios. Here are some reasons why virtualization is widely used:  1. Server Consolidation: Virtualization allows you to run multiple virtual machines (VMs) on a single physical server. This consolidation reduces hardware requirements, saves space, and lowers power consumption. It maximizes the utilization of server resources and reduces overall infrastructure costs.  2. Resource Optimization: Virtualization enables efficient allocation and management of computing resources. By dynamically adjusting resource allocations to VMs based on demand, you can optimize CPU, memory, storage, and network utilization. This flexibility allows for better scalability, performance, and overall resource efficiency.  3. Isolation and Security: Virtualization provides strong isolation between VMs, allowing each VM to operate independently and securely. This isolation helps prevent the spread of malware and improves overall system security. If one VM is compromised, it does not affect the others.  4. Testing and Development: Virtualization is widely used in software development and testing environments. Developers can create VMs to simulate different operating systems, network configurations, or software environments. It enables efficient testing, debugging, and compatibility checking without impacting the production environment.  5. Disaster Recovery and High Availability: Virtualization facilitates robust disaster recovery and high availability solutions. VMs can be easily replicated or migrated to another physical server or data center in case of hardware failures or emergencies. This capability ensures business continuity and minimizes downtime.  6. Scalability and Flexibility: Virtualization allows for quick and easy scaling of resources to meet changing needs. You can add or remove VMs as required without significant hardware changes. It provides the flexibility to adapt to workload fluctuations and accommodate business growth.  7. Legacy System Support: Virtualization enables the hosting of legacy applications and operating systems that may require outdated hardware or software dependencies. By virtualizing these systems, you can extend their lifespan and maintain compatibility while transitioning to newer hardware and software infrastructure.  8. Cloud Computing: Virtualization is a fundamental technology underlying cloud computing. Cloud service providers use virtualization to create and manage virtualized resources, providing customers with scalable, on-demand computing environments. Users can deploy and manage their VMs easily in the cloud. |
| Installation: (Steps , SS ) | |
|  | **Step 1: Download and install VirtualBox**  [**https://www.virtualbox.org/wiki/Downloads**](https://www.virtualbox.org/wiki/Downloads)  **Step 2: Download the Linux ISO**  **https://ubuntu.com/download/desktop**  **Step 3: Install Linux using VirtualBox**   * Start VirtualBox, and click on the New symbol. Give the virtual OS a relevant name.      * Allocate RAM to the virtual OS. My system has 8GB of RAM and I decided to allocate 2GB of it. You can use more RAM if your system has enough extra.      * Create a virtual disk. This serves as the hard disk of the virtual Linux system. It is where the virtual system will store its files.   Start the created virtual machine . Following screen will appear. Select the Language and Click On **Install Ubuntu**     * Select ‘Erase disk and install Ubuntu’. Don’t worry. It won’t delete anything on your Windows operating system. You are using the virtual disk space of 15-20GB that we created in previous steps. It won’t impact the real operating system.      * Set User Credentials and password      * Restart the Machine and Ubuntu is installed successfully. |
| Conclusion: | From this practical , we learnt that how to install Ubuntu Operating System also I learnt hardware specifications required for various Operating System. |

### 

### 

| Practical 2 | |
| --- | --- |
| Aim | Introduction to OS and shell.  1. Access the command Line  2. Manage files and directories from command line  3. Create, edit and and view text files |
| Exercise Question | Exercise - 0  Enter these commands at the UNIX prompt, and try to interpret the output.  1. Passwd  2. Date  3. Hostname  4. Arch  5. uname –a  6. whoami  7. who  8. id  9. echo $SHELL  10. echo {con,pre}{sent,fer}{s,ed}  11. man ls (you may need to press q to quit)  12. man who (you may need to press q to quit)  13. clear  14. cal 2000  15. cal 9 1752 (do you notice anything unusual?)  16. bc -l (type quit or press Ctrl-d to quit)  17. echo 5+4 | bc –l  18. yes please (you may need to press Ctrl-c to quit)  19. time sleep 5  20. history |
|  | Exercise 1    Try the following command sequence  1. Display username of current user.  2. Display current working directory.  3. Make a sub directory named CE in a directory named CSPIT.  4. Create an empty file “ce1.txt” from command prompt.  5. Add the content from command prompt in “ce1.txt”.  6. Display the content of “ce1.txt” file.  7. Change working directory to CE.  8. Make 5 empty files named file1.txt to file5.txt in same directory.  9. List all the files in the directory CE.  10. Add the Name, ID no, and address with pin code to “file1.txt”.  11. Copy contents of file1.txt to file2.txt.  12. Rename file3.txt to “f3.txt”.  13. Display the number of lines, number of words, number of characters of “file1.txt”.  14. Compare the files “file1.txt” to “file2.txt”  15. Update the content of “file2.txt”. Add your skill to existing file.  16. Compare the files “file1.txt” to “file2.txt”  17. Report what is common in the above given files.  18. Add the content in “file4.txt” as given:  India  United States of America  United Kingdom  Australia  19. Add the content in “file5.txt” as given:  India  Canada  United Kingdom  Australia  Germany  20. Find the difference between “file4.txt” and “file5.txt”. How to make these files identical?  21. Create “file6.txt” by adding ten name of students.  22. Create “file7.txt” by adding ten name of students.( few names should be common to  “file6.txt”)  23. Sort the content of “file6.txt” and “file7.txt”  24. Find the common and unique content in “file6.txt” and “file7.txt”  25. Merge the content of above two files in “file8.txt”  26. Remove the duplicate names from “file8.txt”  27. Translate the content of “file1.txt”:  a. Lower case to upper case  b. Remove digits from file  28. Apply head and tail command to see the content of “file8.txt” with different arguments.  29. Differentiate between less and more command and check why less is faster than more  command.  30. Create a file “file9.txt” having content:  Linux is great os. Linux is open source. Linux is free os.  You can learn operating system with linux.  Unix or linux which one you choose.  liNux is easy to learn. Linux is a multiuser os. Learn linux.  Linux is a powerful.  31. Find the lines which contains “linux”.  32. Count the number of lines that matches the “linux”  33. Show the line number of file with the line matched  34. Find the lines which start with “linux”  35. Find the lines which ends with “os”  36. Display the file name that contains “linux”.  37. Download lab manual from department course website.  38. Check the file type of lab manual and other files created.  39. Display IP address of your system (ip addr)  40. Extract only IP address from this |
| Output |  |
| Conclusion | From this practical we learned how to create user along with its functionalities like password and create files with functionalities like copy one file to another and so on. |

### 

| **Practical-3** | |
| --- | --- |
|  | Exercise  PART A |
| Aim | Run id command to view the current user and group information. |
| Command | Id |
| Output |  |
| Aim | display the current working directory. |
| Command | Pwd |
| Output |  |
| Aim | print the value of HOME and PATH variable to determine the home directory and user’s  executable’s path respectively. |
| Command | Echo “$PATH” |
| Output |  |
| Aim | Run su and su - command. Observe the output for the same.what is the main difference between them? |
| Command | Su  Sudo su |
| Output |  |
| Aim | Run sudo su at the shell prompt to become the root user. |
| Command | Sudo su |
| Output |  |
| Aim | Attempt to view the last five lines of /var/log/auth.log without using sudo |
| Command | Su tail /var/log/auth.log |
| Output |  |
| Aim | Attempt to view the last five lines of /var/log/auth.log using sudo |
| Command | Sudo tail /var/log/auth.log |
| Output |  |
| Aim | Attempt to make a copy of /etc/rpc as /etc/rpcOLD without using sudo |
| Command | Cp /etc/rpc as /etc/rpcOLD |
| Output |  |
| Aim | Attempt to make a copy of /etc/rpc as /etc/rpcOLD with sudo. |
| Command | Sudo cp /etc/rpc as /etc/rpcOLD |
| Output |  |
| Aim | Attempt to delete /etc/rpcOLD without using sudo |
| Command | Rm /etc/rpcOLD |
| Output |  |
| Aim | Attempt to delete /etc/rpcdOLD with sudo |
| Command | Sudo rm /etc/rpcOLD |
| Output |  |
| Aim | check the UID for root user, administrator and local users. |
| Command | id |
| Output |  |
| Aim | Adduser user01. |
| Command | Sudo adduser user01 –force-badname |
| Output |  |
| Aim | Create the group group01 with the GID of 10000. |
| Command | Groupadd -g 10000 group group01 |
| Output |  |
| Aim | Create the group group02 |
| Command | Groupadd -group02 |
| Output |  |
| Aim | Examine /etc/group to verify the supplemental group memberships. |
| Command | Cat /etc/group |
| Output |  |
| Aim | Use the usermod -aG command to add a user to a supplementary group. Add user01 to  the group created. |
| Command | Usermod -a -G group01 user01 |
| Output |  |
| Aim | Observe /etc/group and /etc/passwd |
| Command | /etc/group |
| Output |  |

|  | PART B  Aim : Control access to files |
| --- | --- |
| Aim | Check the permission of files created. |
| Command | Ls -l |
| Output |  |
| Aim | Check the permission of directories created. |
| Command | Ls -ld |
| Output |  |
| Aim | Set read and write permissions for others with numeric mode to file1.txt |
| Command | Chmod 666 file1.txt |
| Output |  |
| Aim | Remove write permission for user, group and others to folder CE. |
| Command | Chmod 555 ce |
| Output |  |
| Aim | Create a directory 5CE under CE. Observe the response. |
| Command | Cd ce  Sudo mkdir CE5  Cd ce5  Ls -l |
| Output |  |
| Aim | Set read, write and execute permissions for user, group and others to 5CE. |
| Command | Sudo chmod 777 CE5 |
| Output |  |
| Aim | Set read and execute permission for group and no permission for other to file2.txt. |
| Command | Chmod 750 file2.txt |
| Aim | Change the ownership of file to user01 |
| Command | Sudo chown user01 file2.txt  Ls -l |
| Aim | Change the group ownership of file to group01 |
| Command | Sudo chown :user01 file2.txt  Ls -l |
| Aim | Change the ownership of both group and user at the same time. |
| Command | Sud chown user01:user01 file4.txt  Ls -l |
| Aim | Set the special permissions on directory.  a. The setuid permission on an executable file means that commands run as the  user owning the file, not as the user that ran the command. One example is the  passwd command:run ls -l /usr/bin/passwd  b. The special permission setgid on a directory means that files created in the  directory inherit their group ownership from the directory, rather than inheriting  it from the creating user. run ls -ld /run/log/journal  c. the sticky bit for a directory sets a special restriction on deletion of files. Only  the owner of the file (and root) can delete files within the directory. run ls -ld /tmp |
| Command | 1. run ls -l /usr/bin/passwd 2. ls -ld /run/log/journal 3. ls -ld /tmp |
| Aim | Set the setusid, setgid and sticky bit for different files and perform the operations accordingly. |
| Aim | Display the current value of shell’s mask. |
| Command | umask |
| Output |  |
| Aim | Set the umask to 542. |
| Command | Umask 542 |
| Output |  |
| Aim | Try to open the file and directory created. |
| Command | Vi file1.txt |
| Output |  |
| Aim | Try to open the file as other user |
| Command | Su user01  Vi file1.txt |
| Output |  |

### 

| **Practical-4** | |
| --- | --- |
| AIM | List down all processes with their states sorted by their CPU Usage. Identify current  running process. |
| Command | Ps |
| Output |  |
| AIM | List down all processes associated with current user. |
| Command | Ps -u |
| Output |  |
| AIM | List down all processes associated with their terminal and their states. Identify current  running process. |
| Command | Ps aux |
| Output |  |
| AIM | Compare the output of “ps lx” and “ps l” commands. |
| Command | Ps lx  Ps l |
| Output |  |
| AIM | List down all the names and numbers of all available signals. |
| Command | Kill -l |
| Output |  |
| AIM | Run the “sleep 10000” in background. (i.e. sleep 10000 &) |
| Command | Sleep 10000 & |
| Output |  |
| AIM | Check the PID of sleep process and kill it using PID. |
| Command | Kill 4411 |
| Output |  |
| AIM | Apply w command and observer the output |
| Command | w |
| Output |  |
| AIM | Open the firefox browser. Check the processes associated with firefox. |
| Command | Pgrep firefox |
| Output |  |
| AIM | Kill all processes associated with firefox by its name. |
| Command | Kill 3012 |
| Output |  |
| AIM | Give the difference between kill and pkill. |
| Command | Kill  Pkill |
| Output | pkill is similar to kill, but it allows you to send signals to processes based on their name or other attributes. |
| AIM | Run “lscpu” command and observer the output. |
| Command | lscpu |
| Output |  |

|  | PART B  Aim : Control Services and daemons |
| --- | --- |
| AIM | List all services on your system.(systemctl list-units --type=service) |
| Command | systemctl list-units --type=service |
| Output |  |
| AIM | Check whether the ssh service is active or not. (sudosystemctl status service\_name) |
| Command | sudosystemctl status ssh |
| Output |  |
| AIM | If the package is not available, i nstall ssh package (sudo apt-get install ssh) |
| Command | sudo apt-get install ssh |
| Output |  |
| AIM | If the service is available and active, check the process state usng ps –p PID |
| Command | ps –p 6077 |
| Output |  |
| AIM | Add the firewall rule to allow remote service using ssh(sudo ufw allow ssh) |
| Command | Ip a |
| Output |  |
| AIM | Check your IP address |
| Command | Ip a |
| Output |  |
| AIM | Access another user terminal using ssh |
| Command | sudo ssh <username>@<any pc’s ip address>, it will connect to shell of another linux pc with the specified username and ip address. |
| AIM | Stop the service and check the status |
| Output | exit  sudo systemctl status ssh, the shell will be exited by “exit” command and we will return to our shell then the status will be stopped. |
| AIM | Disable the service and check the status |
| Output | udo stop ssh, it will stop the service permanently |
| AIM | Enable it again and check the status |
| Output | sudo start ssh  sudo systemctl status ssh, the status of the service will not be changed to “active”. |
| AIM | Restart the service and check the status |
| Output | sudo systemctl status ssh, the status will be “active start” because it was started already. |
| AIM | Observe the analyze the output of below mentioned command  1. systemctl is-active ssh  2. systemctl is-enabled ssh  3. systemctl is-failed ssh |
| Output | a. systemctl is-active ssh, it will check that whether the service is active or not  b. systemctl is-enabled ssh, similarly it will check of enable  c. systemctl is-failed ssh, similarly it will check whether the service is enabled or not? |

|  | PART C  Aim: Improve Command Line Productivity  I/o Redirection |
| --- | --- |
| AIM | Create a file named “newfile.txt” and insert a text into created file as follow:  The operating system is a system program that serves as an interface between the  computing system and the end-user. |
| Command |  |
| Output |  |
| AIM |  |
| Command | Redirect the output of “newfile.txt” file to file “new.txt” using command. |
| Output |  |
| AIM | Type command cat, then enter key and enter some text. Observe the output. |
| Command |  |
| Output |  |
| AIM | Type command i) cat <newfile.txt ii) cat newfile.txt. Interpret the output in both  cases. |
| Command |  |
| Output |  |
| AIM | Type command cat – and enter any text. |
| Command |  |
| Output |  |
| AIM | Use both redirection operator < and > at once to redirect the output of one file to  another. |
| Command | cat < input\_file.txt > output\_file.txt |
| AIM | Summarize the use of cat command with redirection operator based on your done  exercise. |
| ANS | There are mainly three symbols that we can use to optimise the file reading, writing operations. The first operator is “>” this operator takes the thing on the left of it as input and redirects it to the thong which is on the right. The operator “<” takes the thing on let of it, as output and redirects it to the thing which is on the left. |
| AIM | Try following command andinterpret the output:  a) ls >filelist  b) cat newfile.txt new.txt >> report  c) cat newfile.txt > newfile.txt  d) date; who  e) date; who>logfile  f) (date; who) > logfile |
| Output | A)    B)    C)    D)    E)    F) |

|  | Piping |
| --- | --- |
| AIM | ls | wc -l |
| Output |  |
| AIM | ls | less |
| Output |  |
| AIM | store the value of count in file named “countfile” using pipeline. |
| Output |  |
| AIM | Try command who | sort and observe the output. |
| Output |  |
| AIM | Strore the sorted output in file named “sortedlist” |
| Output |  |
| AIM | Try cal 1996 | head -10 |
| Output |  |
| AIM | Who | sort – logfile >newfile |
| Output |  |

### 

## **Practical 5**

## **Part A**

| 1 | The File Hierarchy Standard (FHS) is a specification that defines the file system hierarchy of a Linux OS. Illustrate about the use of all directories under the “\” as given in the figure. |
| --- | --- |
| Ans. | /bin: Contains essential binary executables.  /boot: Contains boot-related files.  /dev: Contains device files.  /etc: Contains system-wide configuration files.  /lib: Contains essential shared libraries.  /proc: Contains information about processes and system status.  /root: Home directory of the system administrator (root).  /sbin: Contains system binaries.  /tmp: Temporary files.  /usr: Contains secondary hierarchy, including user and system programs.  /var: Contains variable data, like logs and spool files. |
|  |  |
| 2) | List out files in your directory. |
| Ans) | Command: ls  List Files in Current Directory |
| 3) | Create a hard link to one of the file exist in your directory. (ln [original file] [link name]) |
| Ans) | Command:ln file1.txt link\_file1  Create a hard link with |
| 4) | Apply ls –l and check whether the link is created or not. Also check the size of linked file  created. |
| Ans) | Command:ls -l  check if the link is created, and verify the size of the linked file. |
| 5) | Update the existing file. |
|  | Command:gedit file1.txt |
| 6) | Apply ls –l and check the size of linked file created. |
| Ans) | Command:ls -l  Check the file size |
|  |  |
| 7) | Check the content of both the files and write your observation. |
| Ans) | Command:cat file1.txt  Cat link\_file1  Display the content of files. |
|  | Delete the existing file on which you have created the link. |
| Ans) | Command:rm file1.txt  Delete the file |
| 9) | Apply ls –l and observer the output. |
| Ans) | Command:ls -l  See the file size |
| 10) | Perform exercise 2 to 9 for creation of soft link and write your observation. (ln –s [original file] [link name]) |
| Ans) | Command:ls -l  gedit f1.txt  ln -s f1.txt link\_f1 ls -l  create a soft link and perform all task |
| 11) | Write difference between hard link and soft link. |
| Ans) | **Hard Link:** A hard link is a reference to the same inode (data structure) as the original file, essentially creating multiple directory entries for the same data.  **Soft Link (Symbolic Link):** A soft link, also known as a symbolic link or symlink, is a separate file that contains the path to the target file or directory. It acts as a pointer. |
|  | File Systems, Storage, and Block Devices |
| 12) | Apply ls –l /dev/sda1 |
| Ans) | Command:ls -l /dev/sda1  List details of a specified block device. |
| 13) | To get an overview of local and remote file system devices and the amount of free space available, run the df command |
|  |  |

### 

**Practical – 7**

**Part - A**

**AIM: The demonstration of fork () system call.**

1. Ex1

Code:

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

int main(void)

{

pid\_t pid=fork();

if(pid==0)

{

printf("Child => PPID: %d PID: %d\n",getppid(),getpid());

exit(EXIT\_SUCCESS);

}

else if(pid>0)

{

printf("Parent => PID: %d\n",getpid());

printf("Waiting for child process to finish.\n");

wait(NULL);

printf("Child process finished.\n");

}

else

{

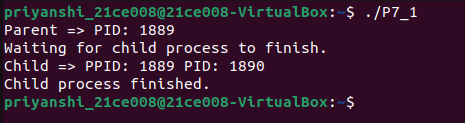
printf("Unable to create child process.\n");

}

return EXIT\_SUCCESS;

}

Output:



1. Ex2

Code:

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

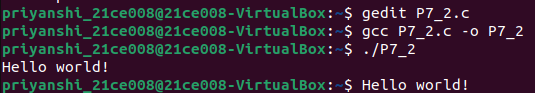
fork();

printf("Hello world!\n");

return 0;

}

Output:



1. Ex3

Code:

#include <stdio.h>

#include <sys/types.h>

int main()

{

fork();

fork();

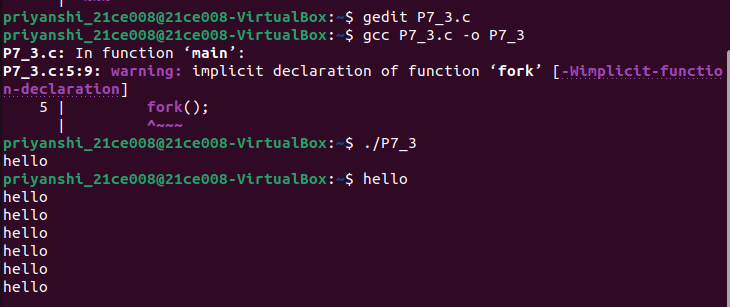
fork();

printf("hello\n");

return 0;

}

Output:



1. Ex4

Code:

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

void forkexample()

{

// child process because return value zero

if (fork() == 0)

printf("Child!\n");

// parent process because return value non-zero.

else

printf("Parent!\n");

}

int main()

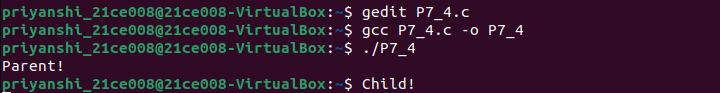
{

forkexample();

return 0;

}

Output:



**Part - B**

**AIM: Demonstration of execve () and wait () system calls along with zombie and orphan states.**

1. Zombie state (process)

Code:

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

#include <stdio.h>

int main()

{

// Fork returns process id

// in parent process

pid\_t child\_pid = fork();

// Parent process

if (child\_pid > 0)

{

sleep(10);

printf("10 seconds waited..");

}

else

exit(0);

return 0;

}

Output:



1. Orphan state (process)

Code:

#include<stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

// Create a child process

int pid = fork();

if (pid > 0)

printf("in parent process \n");

// Note that pid is 0 in child process

// and negative if fork() fails

else if (pid == 0)

{

sleep(5);

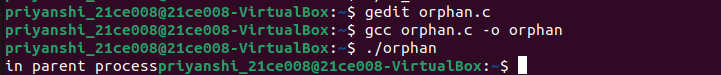
printf("in child process \n");

}

return 0;

}

Output:



1. Merge-Quick sort (use of execve() and wait())

Code:

/\*Implement the C program in which main program accepts the integers to be sorted Main program uses the fork system call to create a new process called a child process.Parent process sorts the integers using merge sort and waits for child process using wait system call to sort the integers using quick sort.Also demonstrate zombie and orphan states. \*/

# include<stdio.h>

# include <stdlib.h>

# include<sys/types.h>

# include<unistd.h>

int split ( int[], int , int );

void quickSort(int\* ,int, int);

void mergeSort(int arr[],int low,int mid,int high)

{

int i,j,k,l,b[20];

l=low;

i=low;

j=mid+1;

while((l<=mid)&&(j<=high)){

if(arr[l]<=arr[j])

{

b[i]=arr[l];

l++;

}

else

{

b[i]=arr[j];

j++;

}

i++;

}

if(l>mid)

{

for(k=j;k<=high;k++)

{

b[i]=arr[k];

i++;

}

}

else

{

for(k=l;k<=mid;k++)

{

b[i]=arr[k];

i++;

}

}

for(k=low;k<=high;k++)

{

arr[k]=b[k];

}

}

void partition(int arr[],int low,int high)

{

int mid;

if(low<high)

{

double temp;

mid=(low+high)/2;

partition(arr,low,mid);

partition(arr,mid+1,high);

mergeSort(arr,low,mid,high);

}

}

void display(int a[],int size)

{

int i;

for(i=0;i<size;i++){

printf("%d\t\t",a[i]);

}

printf("\n");

}

int main()

{

int pid, child\_pid;

int size,i,status;

/\* Input the Integers to be sorted \*/

printf("Enter the number of Integers to Sort::::\t");

scanf("%d",&size);

int a[size];

int pArr[size];

int cArr[size];

for(i=0;i<size;i++){

printf("Enter number %d:",(i+1));

scanf("%d",&a[i]);

pArr[i]=a[i];

cArr[i]=a[i];

}

/\* Display the Enterd Integers \*/

printf("Your Entered Integers for Sorting\n");

display(a,size);

/\* Process ID of the Parent \*/

pid=getpid();

printf("Current Process ID is : %d\n",pid);

/\* Child Process Creation \*/

printf("[ Forking Child Process ... ] \n");

child\_pid=fork(); /\* This will Create Child Process and

Returns Child's PID \*/

if( child\_pid < 0){

/\* Process Creation Failed ... \*/

printf("\nChild Process Creation Failed!!!!\n");

exit(-1);

}

else if( child\_pid==0) {

/\* Child Process \*/

printf("\nThe Child Process\n");

printf("\nchild process is %d",getpid());

printf("\nparent of child process is %d",getppid());

printf("Child is sorting the list of Integers by QUICK SORT::\n");

quickSort(cArr,0,size-1);

printf("The sorted List by Child::\n");

display(cArr,size);

printf("Child Process Completed ...\n");

sleep(10);

printf("\nparent of child process is %d",getppid());

}

else {

/\* Parent Process \*/

printf("parent process %d started\n",getpid());

printf("Parent of parent is %d\n",getppid());

sleep(30);

printf("The Parent Process\n");

printf("Parent %d is sorting the list of Integers by MERGE SORT\n",pid);

partition(pArr,0,size-1);

printf("The sorted List by Parent::\n");

display(pArr,size);

wait(&status);

printf("Parent Process Completed ...\n");

}

return 0;

}

int split ( int a[ ], int lower, int upper )

{

int i, p, q, t ;

p = lower + 1 ;

q = upper ;

i = a[lower] ;

while ( q >= p )

{

while ( a[p] < i )

p++ ;

while ( a[q] > i )

q-- ;

if ( q > p )

{

t = a[p] ;

a[p] = a[q] ;

a[q] = t ;

}

}

t = a[lower] ;

a[lower] = a[q] ;

a[q] = t ;

return q ;

}

void quickSort(int a[],int lower, int upper){

int i ;

if ( upper > lower )

{

i = split ( a, lower, upper ) ;

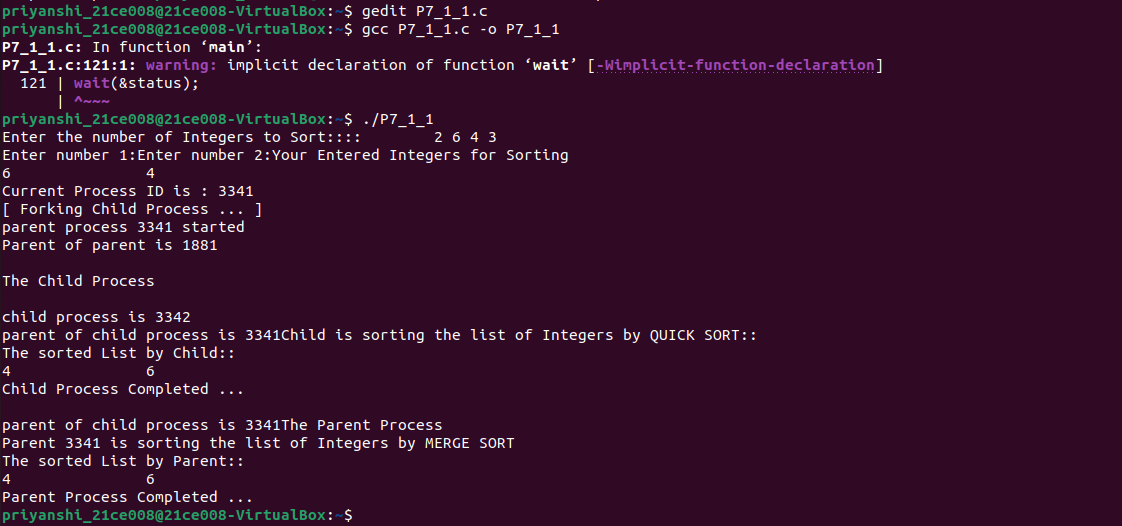
quickSort ( a, lower, i - 1 ) ;

quickSort ( a, i + 1, upper ) ;

}

}

Output:



1. Sort-Search (Parent-child process)

Code:

main

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

int main(int argc, char \*argv[])

{

int val[10],ele;

pid\_t pid;

char\* cval[10];

char \*newenviron[] = { NULL };

int i,j,n,temp;

printf("\nEnter the size for an array: ");

scanf("%d",&n);

printf("\nEnter %d elements : ", n);

for(i=0;i<n;i++)

scanf("%d",&val[i]);

printf("\nEntered elements are: ");

for(i=0;i<n;i++)

printf("\t%d",val[i]);

for(i=1;i<n;i++)

{

for(j=0;j<n-1;j++)

{

if(val[j]>val[j+1])

{

temp=val[j];

val[j]=val[j+1];

val[j+1]=temp;

}

}

}

printf("\nSorted elements are: ");

for(i=0;i<n;i++)

printf("\t%d",val[i]);

printf("\nEnter element to search: ");

scanf("%d",&ele);

val[i] = ele;

for (i=0; i < n+1; i++)

{

char a[sizeof(int)];

snprintf(a, sizeof(int), "%d", val[i]);

cval[i] = malloc(sizeof(a));

strcpy(cval[i], a);

}

cval[i]=NULL;

pid=fork();

if(pid==0)

{

char \*child\_executable = "./Child"; // Update the path if needed

execve(child\_executable, cval, newenviron);

perror("Error in execve call...");

}

}

Child

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main(int argc, char \*argv[],char \*en[])

{

int i,j,c,ele;

int arr[argc];

for (j = 0; j < argc-1; j++)

{

int n=atoi(argv[j]);

arr[j]=n;

}

ele=atoi(argv[j]);

i=0;

j=argc-1;

c=(i+j)/2;

while(arr[c]!=ele && i<=j)

{

if(ele > arr[c])

i = c+1;

else

j = c-1;

c = (i+j)/2;

}

if(i<=j)

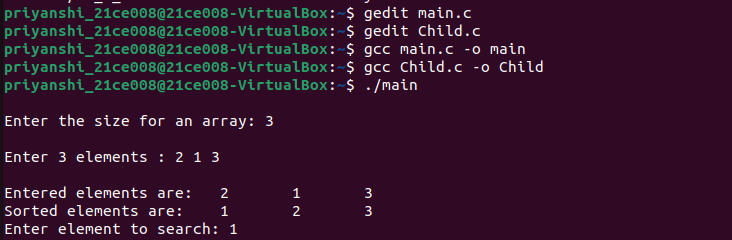
printf("\nElement Found in the given Array...!!!\n");

else

printf("\nElement Not Found in the given Array...!!!\n");

}

Output:



1. Execvparent and execvchild

Code:

execvParent

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

int main(int argc, char \*argv[]){

printf("\n-> PID of parent.c = %d ",getpid());

int parent;

parent = fork();

if(parent == -1)

{

printf("\n-> Some errors in calling ");

}

if(parent == 0)

{

printf("\n-> The child process is running.");

printf("\n-> Now execv will call child.c from child process.");

char \*args[]={"239",NULL};

execv("./execvchild",args);

}

else{

printf("\n-> Now parent is running \n");

}

return 0;

}

execvChild

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

int main(int argc,char \*argv[]){

printf("\n-> Now we are in child.c");

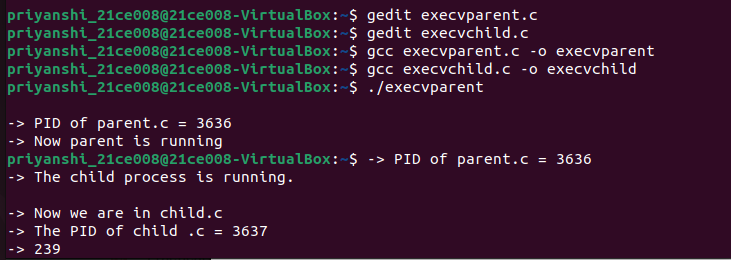
printf("\n-> The PID of child .c = %d",getpid());

printf("\n-> %s",\*argv);

return 0;

}

Output:



### 

# **Practical-8**

**AIM:-**

**Implementation of Process Scheduling Algorithm:**

**a. FCFS**

**b. Round Robing**

**c. SJF**

**d. Priority Scheduling**

| **Sr. No.** | **Code** |
| --- | --- |
| **8.1** | **FCFS**  **Code:-**  #include <iostream>  #include <vector>  using namespace std;  void FCFS(vector<int> arrivalTime, vector<int> burstTime)  {      int n = arrivalTime.size();      int completionTime[n], waitingTime[n], turnaroundTime[n];      // Calculate completion, waiting, and turnaround times      completionTime[0] = burstTime[0];      waitingTime[0] = 0;      turnaroundTime[0] = completionTime[0] - arrivalTime[0];      for (int i = 1; i < n; i++)      {          completionTime[i] = completionTime[i - 1] + burstTime[i];          waitingTime[i] = completionTime[i - 1] - arrivalTime[i];          turnaroundTime[i] = completionTime[i] - arrivalTime[i];      }      // Display results      cout << "Process\tArrival Time\tBurst Time\tCompletion Time\tWaiting Time\tTurnaround Time\n";      for (int i = 0; i < n; i++)      {          cout << i + 1 << "\t" << arrivalTime[i] << "\t\t" << burstTime[i] << "\t\t" << completionTime[i] << "\t\t"               << waitingTime[i] << "\t\t" << turnaroundTime[i] << "\n";      }  }  int main()  {      vector<int> arrivalTime = {0, 1, 2, 3};      vector<int> burstTime = {5, 3, 8, 6};      cout << "FCFS Scheduling:\n";      FCFS(arrivalTime, burstTime);      return 0;  }  **Output:-** |
| **8.2** | **Round Robin**  **Code:-**  #include <iostream>  #include <vector>  #include <queue>  using namespace std;  struct Process  {      int processID;      int burstTime;  };  void roundRobin(vector<int> arrivalTime, vector<int> burstTime, int quantum)  {      int n = arrivalTime.size();      queue<Process> readyQueue;      vector<int> remainingBurstTime(burstTime.begin(), burstTime.end());      vector<int> completionTime(n, 0); // Initialize completion time to 0 for all processes      int currentTime = 0;      int completed = 0;      int timeQuantum = quantum;      int waitingTime[n], turnaroundTime[n];      // Enqueue the first process      readyQueue.push({0, burstTime[0]});      // Simulate the Round Robin scheduling      while (completed < n)      {          Process currentProcess = readyQueue.front();          readyQueue.pop();          if (remainingBurstTime[currentProcess.processID] <= timeQuantum)          {              currentTime += remainingBurstTime[currentProcess.processID];              timeQuantum = remainingBurstTime[currentProcess.processID];              remainingBurstTime[currentProcess.processID] = 0;              completed++;              // Set completion time for this process              completionTime[currentProcess.processID] = currentTime;          }          else          {              currentTime += timeQuantum;              remainingBurstTime[currentProcess.processID] -= timeQuantum;          }          // Enqueue the processes that arrive in the meantime          for (int i = 0; i < n; i++)          {              if (arrivalTime[i] <= currentTime && remainingBurstTime[i] > 0)              {                  readyQueue.push({i, remainingBurstTime[i]});              }          }          // Enqueue the current process again if it's not completed          if (remainingBurstTime[currentProcess.processID] > 0)          {              readyQueue.push(currentProcess);          }          // Calculate waiting and turnaround times          waitingTime[currentProcess.processID] = currentTime - burstTime[currentProcess.processID];          turnaroundTime[currentProcess.processID] = waitingTime[currentProcess.processID] + burstTime[currentProcess.processID];      }      // Display results      cout << "Process\tArrival Time\tBurst Time\tCompletion Time\tWaiting Time\tTurnaround Time\n";      for (int i = 0; i < n; i++)      {          cout << i + 1 << "\t" << arrivalTime[i] << "\t\t" << burstTime[i] << "\t\t" << completionTime[i] << "\t\t"               << waitingTime[i] << "\t\t" << turnaroundTime[i] << "\n";      }  }  int main()  {      vector<int> arrivalTime = {0, 1, 2, 3};      vector<int> burstTime = {5, 3, 8, 6};      int quantum = 2;      cout << "Round Robin Scheduling:\n";      roundRobin(arrivalTime, burstTime, quantum);      return 0;  }  **Output:-** |
| **8.3** | **SJF**  **Coode:-**  #include <iostream>  #include <vector>  #include <algorithm>  using namespace std;  struct Process  {      int processID;      int arrivalTime;      int burstTime;  };  bool compareBurstTime(const Process &a, const Process &b)  {      return a.burstTime < b.burstTime;  }  void SJF(vector<int> arrivalTime, vector<int> burstTime)  {      int n = arrivalTime.size();      vector<Process> processes(n);      for (int i = 0; i < n; i++)      {          processes[i].processID = i;          processes[i].arrivalTime = arrivalTime[i];          processes[i].burstTime = burstTime[i];      }      sort(processes.begin(), processes.end(), compareBurstTime);      int completionTime[n], waitingTime[n], turnaroundTime[n];      int currentTime = processes[0].arrivalTime;      for (int i = 0; i < n; i++)      {          currentTime = max(currentTime, processes[i].arrivalTime);          completionTime[processes[i].processID] = currentTime + processes[i].burstTime;          waitingTime[processes[i].processID] = currentTime - processes[i].arrivalTime;          turnaroundTime[processes[i].processID] = waitingTime[processes[i].processID] + processes[i].burstTime;          currentTime = completionTime[processes[i].processID];      }      // Display results      cout << "Process\tArrival Time\tBurst Time\tCompletion Time\tWaiting Time\tTurnaround Time\n";      for (int i = 0; i < n; i++)      {          cout << processes[i].processID + 1 << "\t" << processes[i].arrivalTime << "\t\t" << processes[i].burstTime               << "\t\t" << completionTime[processes[i].processID] << "\t\t" << waitingTime[processes[i].processID]               << "\t\t" << turnaroundTime[processes[i].processID] << "\n";      }  }  int main()  {      vector<int> arrivalTime = {0, 1, 2, 3};      vector<int> burstTime = {5, 3, 8, 6};      cout << "SJF Scheduling:\n";      SJF(arrivalTime, burstTime);      return 0;  }  **Output:-** |
| **8.4** | **Prioriy Scheduling**  **Code:-**  #include <iostream>  #include <vector>  #include <algorithm>  using namespace std;  struct Process  {      int processID;      int arrivalTime;      int burstTime;      int priority;  };  bool comparePriority(const Process &a, const Process &b)  {      return a.priority < b.priority;  }  void priorityScheduling(vector<int> arrivalTime, vector<int> burstTime, vector<int> priority)  {      int n = arrivalTime.size();      vector<Process> processes(n);      for (int i = 0; i < n; i++)      {          processes[i].processID = i;          processes[i].arrivalTime = arrivalTime[i];          processes[i].burstTime = burstTime[i];          processes[i].priority = priority[i];      }      sort(processes.begin(), processes.end(), comparePriority);      int completionTime[n], waitingTime[n], turnaroundTime[n];      int currentTime = processes[0].arrivalTime;      for (int i = 0; i < n; i++)      {          currentTime = max(currentTime, processes[i].arrivalTime);          completionTime[processes[i].processID] = currentTime + processes[i].burstTime;          waitingTime[processes[i].processID] = currentTime - processes[i].arrivalTime;          turnaroundTime[processes[i].processID] = waitingTime[processes[i].processID] + processes[i].burstTime;          currentTime = completionTime[processes[i].processID];      }      // Display results      cout << "Process\tArrival Time\tBurst Time\tPriority\tCompletion Time\tWaiting Time\tTurnaround Time\n";      for (int i = 0; i < n; i++)      {          cout << processes[i].processID + 1 << "\t" << processes[i].arrivalTime << "\t\t" << processes[i].burstTime               << "\t\t" << processes[i].priority << "\t\t" << completionTime[processes[i].processID] << "\t\t"               << waitingTime[processes[i].processID] << "\t\t" << turnaroundTime[processes[i].processID] << "\n";      }  }  int main()  {      vector<int> arrivalTime = {0, 1, 2, 3};      vector<int> burstTime = {5, 3, 8, 6};      vector<int> priority = {2, 1, 3, 4};      cout << "Priority Scheduling:\n";      priorityScheduling(arrivalTime, burstTime, priority);      return 0;  }  **Output:-** |

### 

| Practical 9 (A) | |
| --- | --- |
| **Aim:** | Compare the Execution of single Process with threads execution. |
| **THREAD**  Threading is a light weight process which shares all the section of the process except for the  stack. A process can have multiple threads. | |
| **Fork Vs Thread**  While threads can execute in parallel with same context. Also, memory and other resources are  shared between the threads causing less overhead. A thread process is considered a sibling while  a forked process is considered a child. Also, threads are known as light-weight processes as they  don't have any overhead as compared to processes (as it doesn't issue any separate command for  creating completely new virtual address space). A single process can have multiple threads. For  all threads of any process, communication between them is direct. While process needs some  interprocess communication mechanism to talk to other processes. Thought, threads seem to be  more useful for any reason, do note that changes in any thread may lead to changes in other  threads of the same process. While, changes in child processes is independent as parent process  has its own execution copy. | |
| **Code:** | #include <stdio.h>  #include <stdlib.h>  #include <pthread.h>  #define NO\_OF\_THREADS 5  #define ELEMENTS 100  static int arr[ELEMENTS];  static int sum;  int retval[5] = {  1,  2,  3,  4,  5  };  int j = 0;  void \* thread\_sum(void \* arg) {  int i;  int \* current\_thread\_data = (int \* ) arg;  printf("-> Current thread no is : %d\n", current\_thread\_data[j]);  int end = (current\_thread\_data[j]) \* (ELEMENTS / NO\_OF\_THREADS);  int start = end - (ELEMENTS / NO\_OF\_THREADS);  printf("-> Here we will calculate the sum of %d to %d\n", arr[start], arr[end - 1]);  int current\_thread\_sum = 0;  for (i = start; i < end; i++) {  current\_thread\_sum += arr[i];  }  sum += current\_thread\_sum;  printf("-> current\_thread\_sum : %d\n", current\_thread\_sum);  pthread\_exit( & retval[j]);  j++;  return NULL;  }  int main() {  int i, thread\_no = 1;  for (i = 0; i < ELEMENTS; i++)  arr[i] = i + 1;  pthread\_t id[NO\_OF\_THREADS];  int data\_arr[NO\_OF\_THREADS];  printf("-> Creating %d number of threads...\n", NO\_OF\_THREADS);  for (thread\_no = 1; thread\_no <= NO\_OF\_THREADS; thread\_no++) {  data\_arr[thread\_no - 1] = thread\_no;  pthread\_create( & id[thread\_no - 1], NULL, thread\_sum, & data\_arr[thread\_no - 1]);  }  for (i = 1; i <= NO\_OF\_THREADS; i++)  pthread\_join(id[i - 1], NULL);  printf("-> Total sum: %d\n", sum);  return 0;  } |
| Output |  |
| **Questions:** | Differentiate between fork and thread. |
| Answer: | Forking and threading are both mechanisms for achieving concurrent execution in computer programs, but they differ in fundamental ways. Forking involves creating a new independent process that duplicates the entire address space of the parent process, resulting in separate memory and execution contexts. Threads, on the other hand, are lighter-weight units of execution that share the same memory space as the parent process but have their own stack and program counter, allowing them to execute concurrently. Threads are generally more efficient for achieving parallelism within a single process, as they have lower overhead compared to forking, which involves duplicating resources. However, forking is more suitable for achieving true process-level isolation when strong separation between tasks is required. |

| Practical 9 (B) | |
| --- | --- |
| **Aim:** | Perform Thread synchronization using counting semaphores and  mutual exclusion using mutex |
| Code | **B) Perform Thread synchronization using counting semaphores**  #include<pthread.h>  #include<stdio.h>  #include<semaphore.h>  #include<unistd.h>  void \*fun1();  void \*fun2();  int shared=1; //shared variable  sem\_t s; //semaphore variable  int main()  {  sem\_init(&s,0,1); //initialize semaphore variable - 1st argument is address of variable, 2nd is  number of processes sharing semaphore, 3rd argument is the initial value of semaphore  variable  pthread\_t thread1, thread2;  pthread\_create(&thread1, NULL, fun1, NULL);  pthread\_create(&thread2, NULL, fun2, NULL);  pthread\_join(thread1, NULL);  pthread\_join(thread2,NULL);  printf("Final value of shared is %d\n",shared); //prints the last updated value of shared  variable  }  void \*fun1()  {  int x;  sem\_wait(&s); //executes wait operation on s  x=shared;//thread1 reads value of shared variable  printf("Thread1 reads the value as %d\n",x);  x++; //thread1 increments its value  printf("Local updation by Thread1: %d\n",x);  sleep(1); //thread1 is preempted by thread 2  shared=x; //thread one updates the value of shared variable  printf("Value of shared variable updated by Thread1 is: %d\n",shared);  sem\_post(&s);  }  void \*fun2()  {  int y;  sem\_wait(&s);  y=shared;//thread2 reads value of shared variable  printf("Thread2 reads the value as %d\n",y);  y--; //thread2 increments its value  printf("Local updation by Thread2: %d\n",y);  sleep(1); //thread2 is preempted by thread 1  shared=y; //thread2 updates the value of shared variable  printf("Value of shared variable updated by Thread2 is: %d\n",shared);  sem\_post(&s);  } |
| Output |  |
|  | **B) Perform Thread synchronization using mutex**  An example code to study synchronization problems  #include <pthread.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <unistd.h>  pthread\_t tid[2];  int counter;  void\* trythis(void\* arg)  {  unsigned long i = 0;  counter +=7 1;  printf("\n Job %d has started\n", counter);  for (i = 0; i < (0xFFFFFFFF); i++) ;  printf("\n Job %d has finished\n", counter);  return NULL;  }  int main(void)  {  int i = 0;  int error;  while (i < 2) {  error = pthread\_create(&(tid[i]), NULL, &trythis, NULL);  if (error != 0)  printf("\nThread can't be created : [%s]", strerror(error));  i++;  }  pthread\_join(tid[0], NULL);  pthread\_join(tid[1], NULL);  return 0;  } |
| Output |  |

### 

|  | Practical 10 |
| --- | --- |
| **Aim:** | Implement inter process communication (IPC) using PIPEs and FIFOs |
| **Code:** | **Using Pipes:**  #include<stdio.h>  #include<unistd.h>  int main() {     int pipefds[2];     int returnstatus;     char writemessages[2][20]={"Hi", "Hello"};     char readmessage[20];     returnstatus = pipe(pipefds);       if (returnstatus == -1) {        printf("Unable to create pipe\n");        return 1;     }       printf("Writing to pipe - Message 1 is %s\n", writemessages[0]);     write(pipefds[1], writemessages[0], sizeof(writemessages[0]));     read(pipefds[0], readmessage, sizeof(readmessage));     printf("Reading from pipe – Message 1 is %s\n", readmessage);     printf("Writing to pipe - Message 2 is %s\n", writemessages[0]);     write(pipefds[1], writemessages[1], sizeof(writemessages[0]));     read(pipefds[0], readmessage, sizeof(readmessage));     printf("Reading from pipe – Message 2 is %s\n", readmessage);     return 0;  } |
| **Output:** |  |
|  | **Using FIFO**  #include <iostream>  #include <cstdlib>  #include <cstring>  #include <unistd.h>  #include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>  int main() {  *const* char\* fifo\_path = "myfifo";    *// Create the FIFO if it doesn't exist*      if (mkfifo(fifo\_path, 0666) == -1) {          perror("mkfifo");          exit(EXIT\_FAILURE);      }        pid\_t pid = fork();        if (pid == -1) {          perror("fork");          exit(EXIT\_FAILURE);      }        if (pid == 0) { *// Child process*          int fifo\_read = open(fifo\_path, O\_RDONLY);          if (fifo\_read == -1) {              perror("open");              exit(EXIT\_FAILURE);          }          char buffer[100];          ssize\_t bytes\_read = read(fifo\_read, buffer, sizeof(buffer));          if (bytes\_read == -1) {              perror("read");              exit(EXIT\_FAILURE);          }          std::cout << "Child received: " << std::string(buffer, bytes\_read) << std::endl;          close(fifo\_read);      } else { *// Parent process*          int fifo\_write = open(fifo\_path, O\_WRONLY);          if (fifo\_write == -1) {              perror("open");              exit(EXIT\_FAILURE);          }          std::string message = "Hello from parent";          ssize\_t bytes\_written = write(fifo\_write, message.c\_str(), message.size());          if (bytes\_written == -1) {              perror("write");              exit(EXIT\_FAILURE);          }          close(fifo\_write);      }    *// Remove the FIFO file*      if (unlink(fifo\_path) == -1) {          perror("unlink");          exit(EXIT\_FAILURE);      }        return 0;  } |
| **Output:** |  |