

**Operating Systems**

**40201341**

**R/615/1700**

**Section (2)**

**Operating System Analysis**

**Submitted to**

Eng. Malik Al Louzi

**Submitted on**

January 29th, 2024

**Submitted by**

Marwan Tarek Shafiq Al Farah

**Student ID**

21110011

**Fall 2023 - 2024**

**Table of Content**

[***Task 1*** 3](#_Toc157119834)

[***Task 2*** 6](#_Toc157119835)

[***Task 3*** 12](#_Toc157119836)

[***Task 4*** 19](#_Toc157119837)

[***Task 5*** 22](#_Toc157119838)

[***Task 6*** 26](#_Toc157119839)

[***Task 7*** 28](#_Toc157119840)

[***Task 8*** 33](#_Toc157119841)

[***References*** 36](#_Toc157119842)

# ***Task 1***

***(Silberschatz, Galvin and Gagne, 2017; GeeksforGeeks, 2022; B. Rand, 2023; Kanade, 2023)***

***Overview of Virtual Memory***

Virtual memory is an operating system memory management feature that uses hardware and software to compensate for physical memory shortages by temporarily transferring data from random access memory (RAM) to disk storage, giving user applications the illusion of much larger memory size. This process is smooth and allows the system to execute huge programs or run many apps simultaneously, even if the physical RAM is insufficient to handle the load. While physical memory (RAM) remains the high-speed access point for running programs, virtual memory overcomes its limitations by using secondary storage like a seamless extension.

At the heart of virtual memory lies a dynamic address translation mechanism. Programs interact with virtual addresses, a larger address space separate from the physical addresses of RAM. The operating system, using a data structure called a page table, maps these virtual addresses to their corresponding physical locations in RAM. When a program tries to access data in the virtual address space, the CPU utilizes the Memory Management Unit (MMU) to consult the page table and locate the actual physical address in RAM.

***Role in Memory Management***

The primary role of virtual memory in an operating system is to extend the available memory on a computer. It does this by providing an “illusion” of a very large memory space that can be much larger than the actual physical memory. The OS achieves this by using a portion of the hard drive, designated as the “swap space” or “page file,” to store parts of programs and data files that are not currently in use or that do not fit in the available RAM.

Virtual memory allows the operating system to handle memory management more efficiently. It does this by breaking down physical memory into blocks known as pages. By dividing programs into pages and loading only active pages into RAM, multiple programs can coexist, enhancing user experience and system utilization, which is called demand paging. Correspondingly, the virtual memory is also divided into pages of the same size.

Since the instructions that are being processed must be in the RAM in order to be able to execute it, when a program requires a page that is not in the physical memory a page fault occurs which requires a page replacement algorithm to help the operating system transfer data between the RAM and the disk by helping it decide which pages to swap, optimizing memory usage and allowing processes to exceed physical RAM constraints. Inactive pages can be swapped out to secondary storage to free up RAM for other processes, further enhancing memory usage. This process is managed through a page table, which keeps track of where each page of memory is.

There are many page replacement algorithms, such as FIFO, Optimal, LRU, and LRU Approximation Page Replacement Algorithms. In case of not having any free frames in the memory, these algorithms help the OS decide which pages to swap out of the memory (victims) in order to free up space for another page that is needed for a running process.

***Enhancing System Performance***

Virtual memory enhances overall system performance in several ways:

1. **Increased Application Size and Multitasking:** It allows systems to run larger applications than would be possible with just the available RAM, which is important in modern computing environments where multitasking is common, and multiple applications may need to run simultaneously.
2. **Efficient Memory Utilization:** By storing only the necessary parts of programs and data in RAM by the use of swapping and demand paging, virtual memory ensures that the physical memory is used more efficiently, minimizing idle memory periods. This can lead to better performance, as the most frequently accessed data is kept in the faster-to-access RAM.
3. **Stability and Security:** Virtual memory ensures system stability and security. Each program is assigned its own virtual address space, which separates it from other applications. This means that one application cannot interact with another’s memory space, lowering the likelihood of system crashes and increasing security against malicious software.
4. **Cost-Effectiveness:** It is a cost-effective solution for memory management, as physical RAM is significantly more expensive than disk storage. By using disk space as virtual memory, systems can achieve more memory resources without the high cost of adding more RAM.
5. **Increased System Responsiveness:** With multiple programs sharing RAM efficiently, response times improve as applications don’t wait for disk access for every resource.
6. **Facilitating Multitasking:** Multiple apps can run at once in a multitasking environment thanks to virtual memory. It controls how much memory each program needs, making sure they all have access to the resources they need without interfering with one another. The user experience and overall system efficiency are improved by this smooth handling of several programs.
7. **Reduction in Physical Memory Requirements:** Virtual memory lessens the requirement for large quantities of RAM by efficiently utilizing disk space to supplement physical memory. This can be especially helpful for systems with low physical memory capacity.
8. **Adaptability and Scalability:** Virtual memory systems are very flexible and can evolve to meet the system’s requirements, as it may be configured to automatically expand to accommodate larger applications and datasets, guaranteeing reliable performance in a variety of scenarios.

**Impact of Page Faults on Performance**

* **Page Fault:** When a program tries to access a page that is not in the RAM, a page fault occurs. This situation requires the operating system to fetch the needed pages from the disk and load it into RAM, which can be a time-consuming process.
* **Hit vs. Miss:** A ‘hit’ in virtual memory means that the page required by a program is already in the RAM, leading to quick access and minimal performance impact. However, a ‘miss’ indicates a page fault, necessitating page retrieval from slower disk storage.
* **Swapping and Performance:** Swapping data between RAM and disk storage is an important mechanism to handle page faults. Because disk storage has slower access times than RAM, swapping can significantly reduce system performance even though it’s used to extend the memory size.
* **Result on Performance:** Frequent page faults and extensive swapping can lead to performance degradation, which could lead to a scenario called “thrashing,” where the system spends more time swapping data in and out of memory than executing tasks. This lead to an increase in the Effective Access Time. To overcome this, effective page replacement algorithms and large RAMs are essential for maintaining peak performance and minimizing the negative impacts of page faults.

# ***Task 2***

***(GeeksforGeeks, 2023a)***

***Calculating Required Page Tables for a Process***

Meaning that the process needs to use pages in order to be able to divide it into pages using only an inner page table, which causes a problem as the page table needs to be consecutive, meaning that we have to use outer page table in order to be able to use paging correctly.

Meaning that the inner page table needs to use pages in order to be able to divide it into pages using an outer page table, which causes a problem as the page table needs to be consecutive, meaning that we have to use 2nd outer page table in order to be able to use paging correctly.

Meaning that the outer page table needs to use pages in order to be able to divide it into pages using a 2nd outer page table, which causes a problem as the page table needs to be consecutive, meaning that we have to use 3rd outer page table in order to be able to use paging correctly.

Meaning that in order to use paging for this 44-bit address space process we will need to use 3 levels of page tables. Our virtual address space will be 16 TB, taking up , along with 16 GB for the page table (last level), taking up , 16 MB for the page table (second last level), taking up , and 16 KB for the page table (third last level).

***Logical Address Division and Bit Count in CPU***

The logical address would be divided as follows where each part of the address is used to index into different levels of the page table structure:

1. **Page Size (16 KB) / Page Offset:** The size of a page determines the number of bits used to index within a page or a frame. Since the page size is bytes, the last 14 bits of the logical address are used as the page offset, meaning that the remaining 30 bits will be divided among the 3 levels of page tables.
2. **Page Number (or Frame Number):** This part of the address identifies which page (in virtual memory) or frame (in physical memory) is being referenced. The mapping of pages to frames is what the page table handles. Since the page offset is 14 bits out of the 44 bit address space, that means that the page number should be 30 bits.
3. **Page Table Entry Size (16 bytes):** This size affects how many entries each page table can hold. Since each entry is bytes and each page is bytes, a single page can hold entries.
4. **Division of the Page Number Bits:** The remaining bits are used to index into the different levels of page tables. Given that we have multiple levels of page tables, the bits will be divided among these levels.

* **Page Tables (Third Last Level):** Since the page tables (third last level), has entries (), we need 10 bits to index into it.
* **Page Tables (Second Last Level):** The page tables (second last level), with entries, needs 20 bits to index into it. However, 10 of these bits have already been used by the page tables (third last level), leaving 10 bits for the page tables (second last level).
* **Page Tables (Last Level):** The page tables (last level), with entries, needs 30 bits to index into it. However, 20 of these bits have already been used by the page tables (second last level) and page tables (third last level), leaving 10 bits for the page tables (last level)

Therefore, the 44-bit logical address can be divided as follows:

* 10 bits for the page tables (third last level)
* 10 bits for the page tables (second last level)
* 10 bits for the page tables (last level)
* 14 bits for the Page Offset

This division allows the CPU to navigate through the multi-level page table structure to translate a logical address to a physical address using the MMU and the relocation register. Each part of the logical address is used to index into a specific level of the page table, eventually leading to the page frame number, which, combined with the page offset, gives the physical address.

|  |  |  |  |
| --- | --- | --- | --- |
| Third last level | Second last level | Last level | Offset |
| p1 | p2 | p3 | d |
| 10 bits | 10 bits | 10 bits | 14 bits |

***Reflection on the Logical Address Division in the Paging System***

1. **Page Tables (Third Last Level) (10 bits):**

* **Usage:** These 10 bits are used to index into the selected page tables (third last level). This table contains entries, each pointing to a page table (second last level).
* **Role in Address Translation:** These bits further refine the search to locate the exact page table (second last level) that will contain the entry for the page table (last level).

1. **Page Table (Second Last Level) (10 bits):**

* **Usage:** These 10 bits index into the page table (second last level), which has entries, each pointing to a different page table (last level).
* **Role in Address Translation:** By indexing with these bits, the system locates the specific page table (last level) that contains the page frame number.

1. **Page Table (Last Level) (10 bits):**

* **Usage:** The following 10 bits are used to index into the page table (last level). Each entry in this table directly points to a page frame that contains a part of the original process in the physical memory.
* **Role in Address Translation:** These bits determine the specific page frame in physical memory where the required data resides.

1. **Page Offset (14 bits):**

* **Usage:** The last 14 bits in the logical address are used as the offset within the selected page frame.
* **Role in Address Translation:** This offset specifies the exact location within the page frame where the desired data begins.

***Critical Evaluation of Memory Management Functionality in Operating Systems with Paging***

1. **Scalability:** The multilevel paging system demonstrates strong scalability. By dividing the address space into three levels, the operating system can efficiently manage a large virtual address space. This structure is crucial for modern computers that need to handle vast amounts of data and numerous processes.
2. **Memory Overhead:** However, there is a significant memory overhead associated with maintaining multiple page tables. In this example, the total memory required for page tables is substantial (over 16 GB), which could be a concern in systems with limited physical memory.
3. **Access Time:** Accessing a memory location involves multiple steps - traversing through three levels of page tables before reaching the actual data, and then translating this logical address into a physical address using the MMU. This multi-step process can increase the time it takes to access memory, potentially impacting system performance, especially if the tables are not kept in faster memory (like cache).
4. **Complexity:** The complexity of managing multilevel page tables increases with the depth of levels. While this is manageable by the operating system, it adds to the complexity of the memory management.
5. **Flexibility and Efficient Memory Utilization:** The use of paging allows for non-contiguous memory allocation, which leads to efficient utilization of memory. It avoids issues like external fragmentation and enables more flexible memory allocation strategies.
6. **Support for Large Address Spaces:** The use of a 44-bit address space, as shown in the task, indicates support for very large process address spaces. This is essential in modern computing environments where applications may require extensive memory resources.

***Examples of Paging in Operating Systems*** (Moose, 2016; Ukessays, 2018; Hacker News, 2020; Chen, 2021; The Linux Kernel documentation, 2021; Dancuk, 2022; Raut, 2022; Travis, 2022; Ashwathnarayana, 2023; Tutorlalspoint, 2023; Kasiya, 2023; Learn, 2023)

* **Microsoft Windows**

Windows implements paging with a focus on providing a balance between performance and resource utilization suitable for a wide range of devices, from personal computers to enterprise servers.

In Windows, paging is utilized extensively as a part of its virtual memory management system. Windows uses a cluster demand paging system “page file” located on the system disk which acts as an extension of physical RAM, storing pages (fixed-size blocks) of inactive memory when RAM becomes scarce, meaning pages are only swapped to the page file when needed, minimizing unnecessary disk access. This means that not all parts of a program need to be loaded into physical memory at the same time, allowing Windows to manage larger programs and multitask more effectively. The “commit charge” represents the total amount of memory allocated to all processes, including both RAM and page file usage. Windows uses FIFO as its page replacement algorithm.

Windows tends to use a pre-paging strategy, where it attempts to predict and load pages that will be needed soon, aiming to reduce waiting times for users. Windows uses techniques like write combining and prefetching to optimize memory access patterns, which improves overall system performance, particularly in graphical applications and data-intensive processes. Windows’ VMM is sophisticated, handling not just paging but also other memory management tasks. It ensures that the system operates smoothly under various loads, balancing the demands of different applications and services. Windows 10 and later versions include a memory compression feature, which stores more information in the physical memory by compressing less frequently used pages, thus reducing the amount of data swapped to the disk.

Windows primarily uses a fixed-size swap file (pagefile.sys), which is automatically configured based on system memory. However, administrators can manually adjust the size or let Windows manage it dynamically based on system needs. This simplifies user configuration but reduces control over performance optimization. Besides the standard 4KB pages, Windows supports large pages of size 2MB, particularly beneficial for larger applications requiring continuous memory blocks. This feature is advantageous for performance optimization, as it can reduce the overhead of page table entries and improve cache utilization.

* **Linux**

Linux, an open-source operating system, also incorporates paging in its memory management system. Like Windows, Linux uses demand paging, allowing efficient use of physical memory, better handling of large applications, reducing overhead and improving overall system performance. However, Linux uses the LRU page replacement algorithm.

Linux utilizes “swap space,” which can be a dedicated partition or a regular file on any storage device. Unlike Windows’ singular page file, Linux can have multiple swap spaces of varying sizes and priorities. The “virtual memory” concept encompasses both RAM and swap space, offering a more comprehensive view of memory allocation.

Linux uses a multi-level page table structure. Originally, a two-level paging mechanism was standard, but with the advent of x86-64 architecture, this evolved into a three-level and then a four-level page table. This hierarchical structure efficiently manages the large virtual address space provided by modern processors.

Linux typically operates with a fixed page size of 4KB, although support for larger page sizes (2MB and 1GB) exists in the form of Transparent Huge Pages (THP) and HugeTLB pages. These larger pages are used selectively, mainly for applications that can benefit from them, such as databases or large scientific computing tasks, as it reduces the overhead of page table lookups for large memory applications and improves performance.

Linux’s memory management system includes a ‘swappiness’ parameter, allowing administrators to fine-tune the balance between swap usage and RAM retention. This control can optimize performance based on the specific workload and system capabilities.

Users have greater control over swap spaces in Linux, allowing them to adjust size, location, and even disable specific spaces based on performance needs and preferences.

**Comparison**

Both Windows and Linux operating systems employ demand paging as a key component of their memory management strategies but differ in their approach to page management and system optimization. They also differ in their page replacement algorithms as Windows uses FIFO and Linux uses LRU.

Both systems provide support for larger page sizes beyond the standard 4KB. Windows offers larger page sizes such as 2MB, while Linux uses Transparent Huge Pages (THP) and HugeTLB for similar larger page sizes such as 2MB and 1GB sizes. These larger pages are beneficial for large applications, as they reduce the overhead of page table entries and improve cache utilization.

Windows uses a fixed-size swap file (pagefile.sys), which is either auto-configured or manually adjustable by administrators. This setup simplifies user configuration but offers less flexibility in performance optimization. Linux, in contrast, provides a more flexible approach with its “swap space,” which can be a dedicated partition or a file on any storage device. Users have greater control over swap spaces, including their size, location, and priority, allowing for more tailored performance optimization.

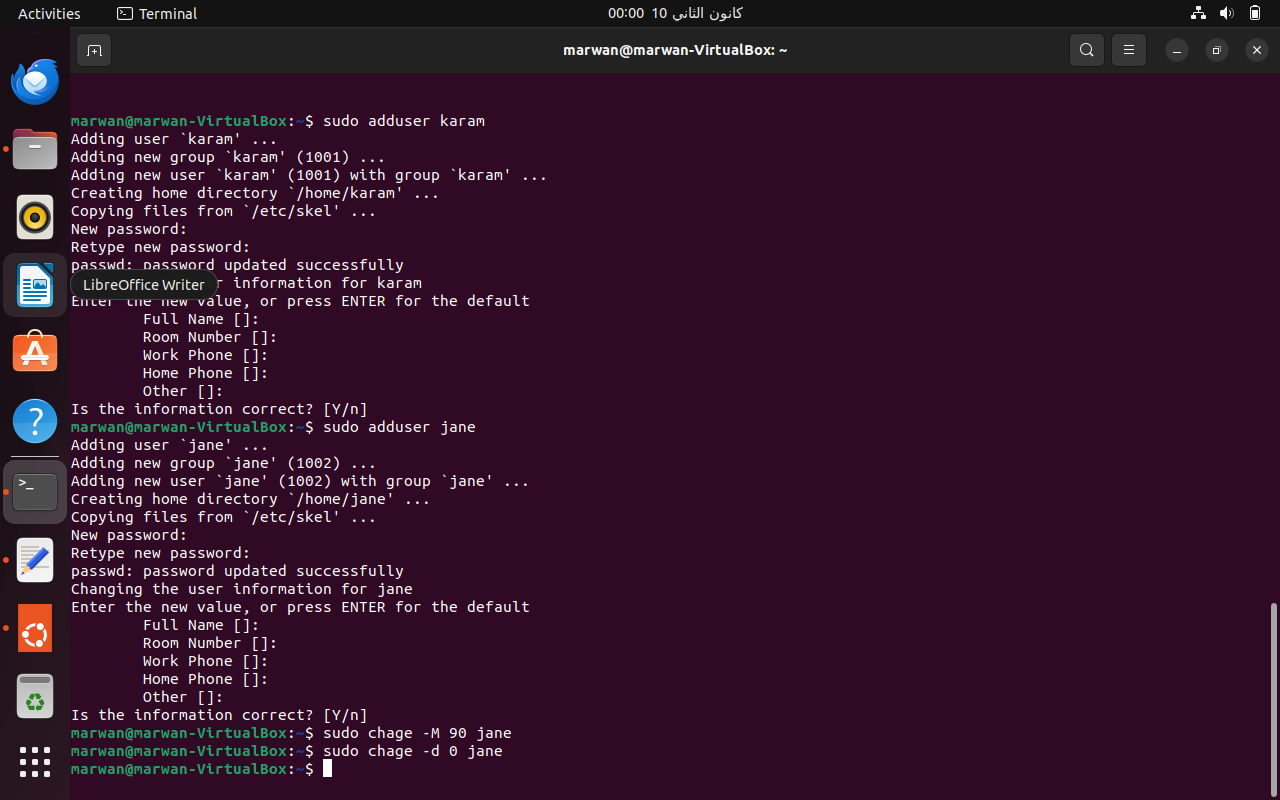
Windows employs a pre-paging strategy, predicting and loading pages that will likely be needed soon. This reduces waiting times and is particularly effective in graphical applications and data-intensive processes. Linux does not emphasize pre-paging as prominently, focusing more on the efficient management of its multi-level page table structure.

Windows’ Virtual Memory Manager (VMM) is a sophisticated system handling not just paging but also other memory management tasks. It includes features like memory compression in later versions, optimizing the balance between different applications and services under various loads. Linux’s memory management is characterized by its multi-level page table structure and the ‘swappiness’ parameter. The multi-level page table efficiently manages large virtual address spaces, while ‘swappiness’ allows fine-tuning of the balance between swap usage and RAM retention.

In Windows, while administrators can adjust the size of the swap file, the overall system leans towards automatic configuration and management, simplifying user involvement but potentially limiting optimization capabilities. Linux offers more extensive control to users and administrators over its memory management aspects, including detailed adjustments of swap spaces and tuning of system parameters to optimize performance based on specific workloads and system capabilities.

In summary, while both Windows and Linux employ demand paging and support large page sizes, they differ in their approach to swap space management, pre-paging strategies, the complexity of their memory management systems, and the level of user control and customization offered. Windows provides a more user-friendly but less flexible system, whereas Linux offers extensive customization at the cost of increased complexity.

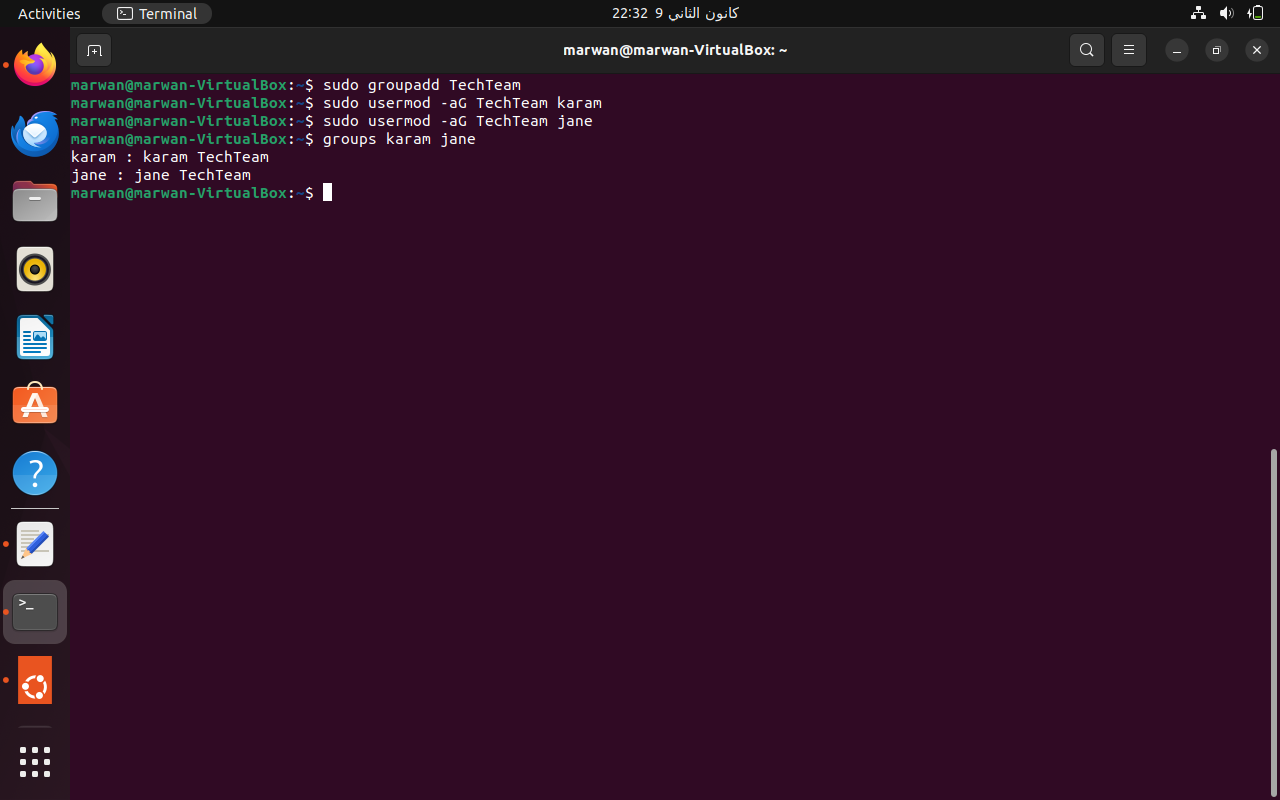
# ***Task 3***



Part 1 Figure

The commands in Part 1 are focused on creating two user accounts, ‘karam’ and ‘jane’. The additional security configurations for ‘jane’, involving password expiration and forced password change on the first login, align with the requirement of heightened security for the second user. This step ensures that both users are set up with their respective security settings, laying the foundation for their involvement in the project.

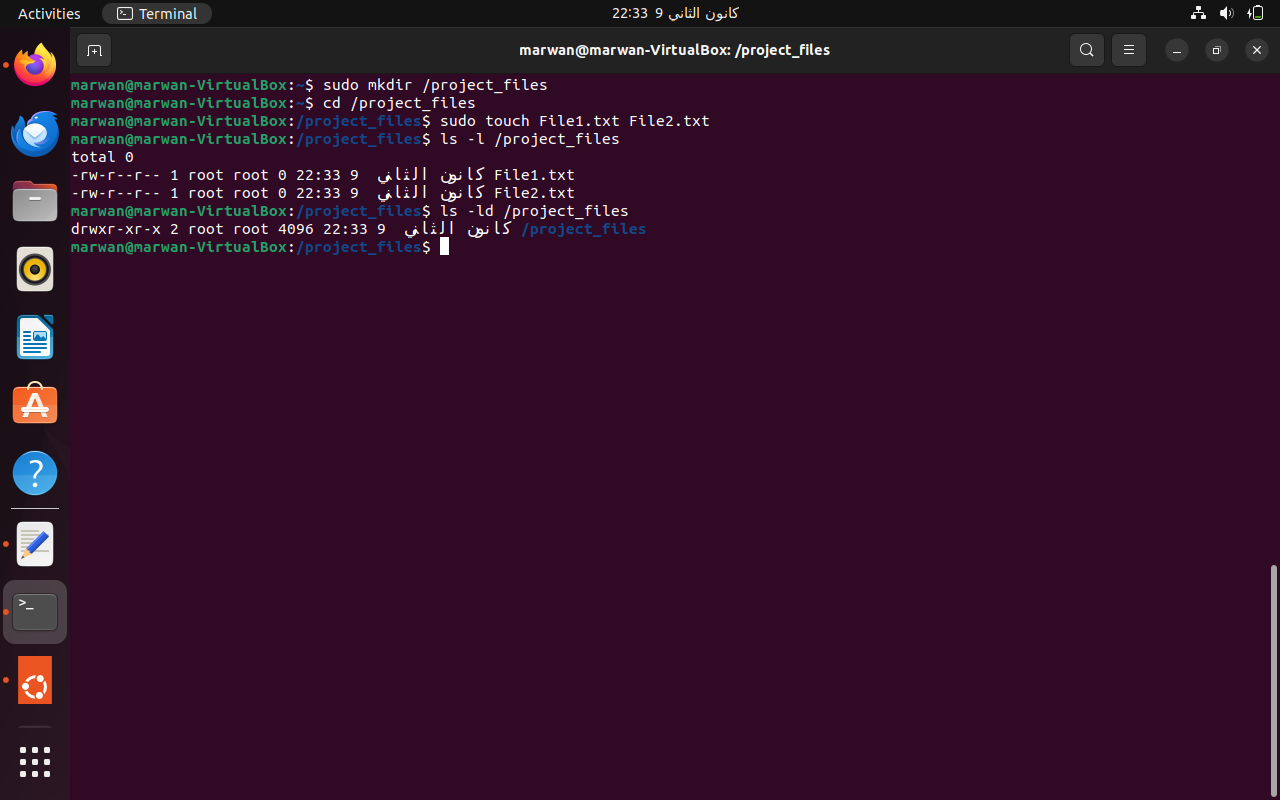
* **sudo adduser karam**: This command creates a new user account named ‘karam’. The sudo prefix gives administrative privileges, ensuring the command is executed with root access.
* **sudo adduser jane**: Similar to the above, this creates a new user named ‘jane’.
* **sudo chage -M 90 jane**: Sets the password expiration for ‘jane’ to 90 days. This is part of heightened security configurations, ensuring regular password updates.
* **sudo chage -d 0 jane**: Forces ‘jane’ to change her password upon first login. This is a security measure to ensure that the initial password set by the administrator is updated by the user.



Part 2 Figure

In Part 2, the commands establish a new group, ‘TechTeam’, and add both ‘karam’ and ‘jane’ to this group. This directly addresses the requirement of forming a specialized team and facilitates collaboration between the two users by associating them with a common group.

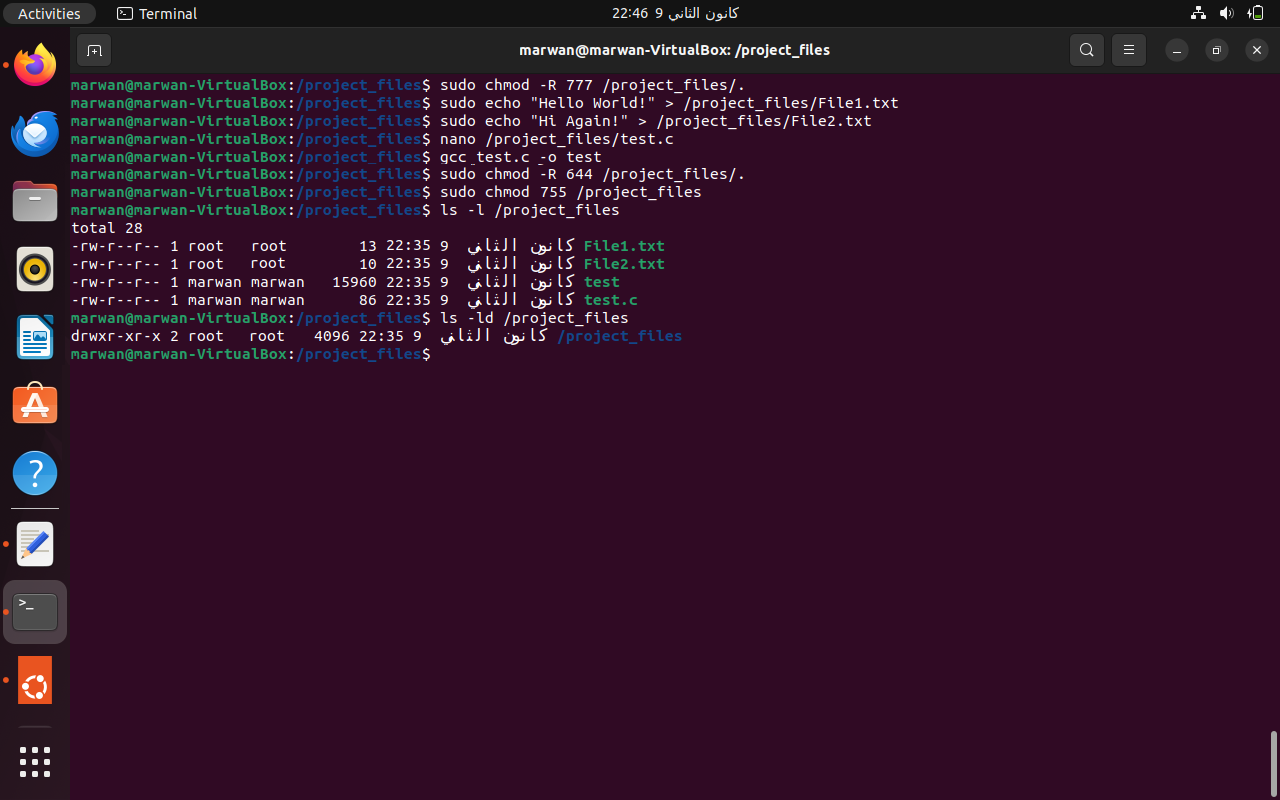
* **sudo groupadd TechTeam**: Creates a new group named ‘TechTeam’.
* **sudo usermod -aG TechTeam karam** and **sudo usermod -aG TechTeam jane**: These commands add the users ‘karam’ and ‘jane’ to the ‘TechTeam’ group, facilitating collaboration.
* **groups karam jane**: Verifies that ‘karam’ and ‘jane’ are members of the ‘TechTeam’ group.



Part 3 Figure

The creation of the ‘/project\_files’ directory and the addition of ‘File1.txt’ and ‘File2.txt’ within it meet the specifications for setting up a dedicated workspace for the project. This step ensures that there is a central location for project-related files, which is essential for organized collaboration.

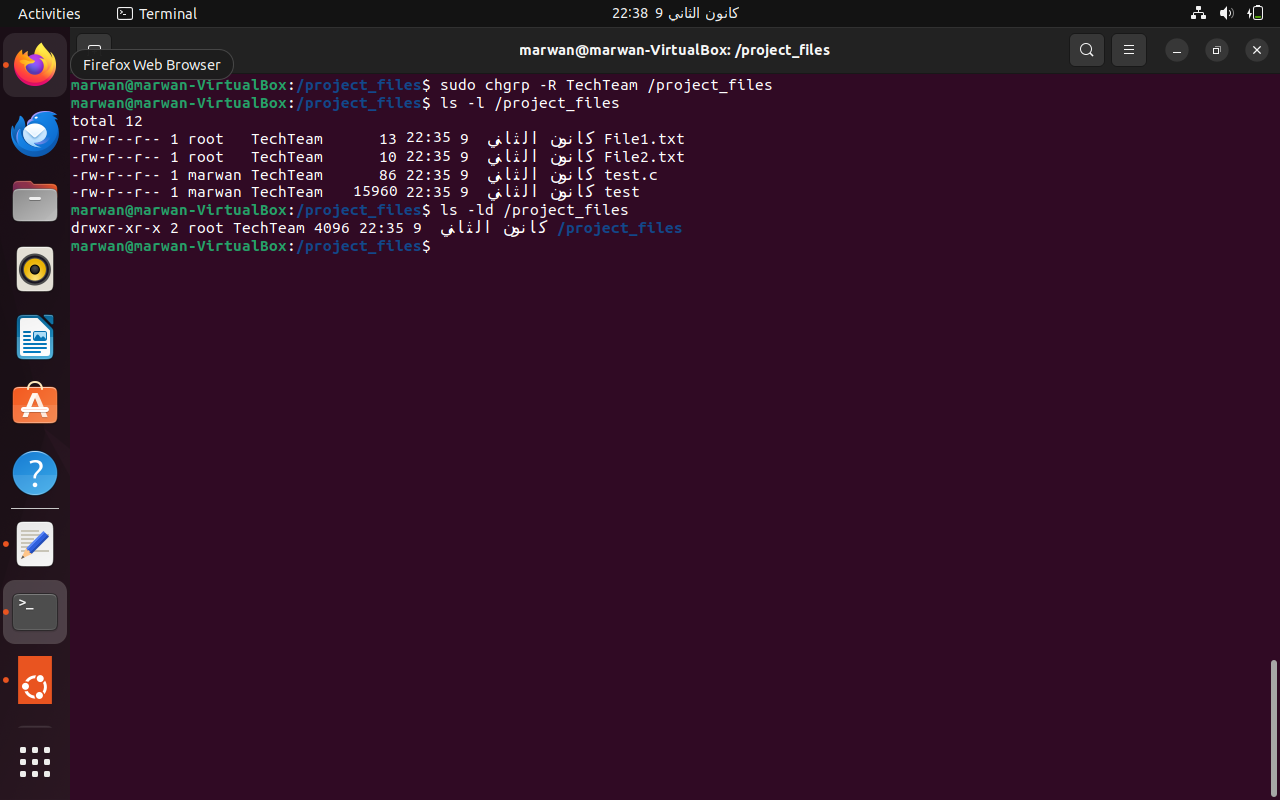
* **sudo mkdir /project\_files**: Creates a directory named ‘project\_files’ at the root.
* **cd /project\_files**: Changes the current directory to ‘/project\_files’.
* **sudo touch File1.txt File2.txt**: Creates two files, ‘File1.txt’ and ‘File2.txt’, in the ‘/project\_files’ directory.
* **ls -l /project\_files** and **ls -ld /project\_files**: These commands list the contents of ‘/project\_files’ and display the directory details, respectively, confirming the creation of the files and directory.



Setting up for Part 7 figure

These steps, though not part of the specified tasks, are critical for setting the ground for checking all the permissions during Part 7. They serve as a setup for the subsequent tasks. We first modify all the permissions, then return them to their original state.

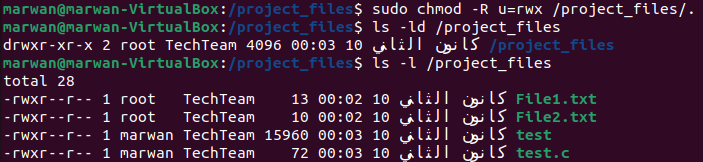
* **sudo chmod -R 777 /project\_files/.**: Temporarily sets full permissions for all users on the ‘/project\_files’ directory and its contents for testing purposes.
* **echo**, **nano** and **gcc test.c -o test**: Fills the .txt files with data and creats an executable file to be used in Part 7.
* **sudo chmod -R 644 /project\_files/.** and **sudo chmod 755 /project\_files**: After testing, the permissions are reset to a more restrictive configuration.
* **ls -l /project\_files** and **ls -ld /project\_files**: Confirm the changes in permissions and file contents.

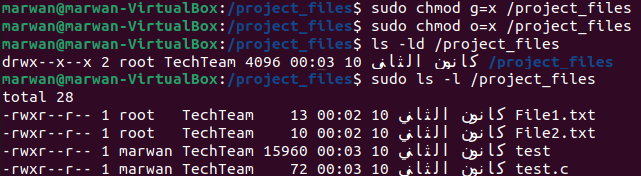


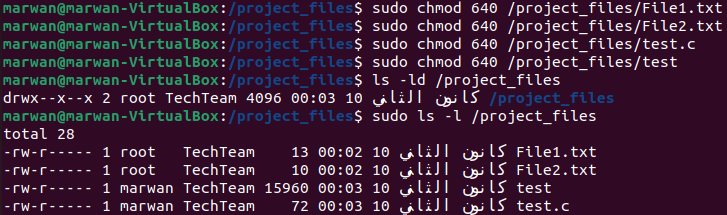
Part 4 Figure

Changing the group ownership of the ‘/project\_files’ directory and its contents to ‘TechTeam’ directly relates to the specification of ensuring that the ‘TechTeam’ group has the necessary access for collaboration. It aligns the directory’s access controls with the team’s structure.

* **sudo chgrp -R TechTeam /project\_files**: Changes the group ownership of ‘/project\_files’ and its contents to the ‘TechTeam’ group.
* **ls -l /project\_files** and **ls -ld /project\_files**: Validates the change in group ownership.



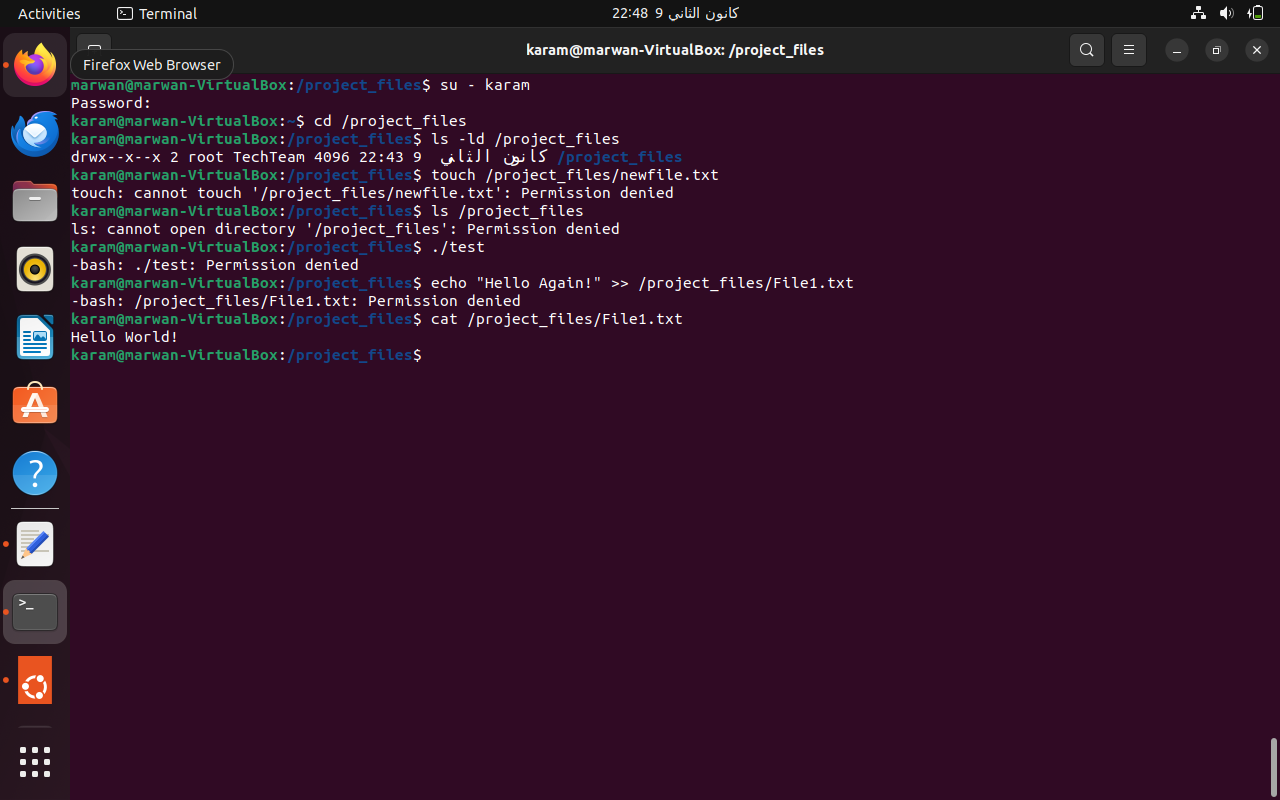




Part 5 and 6 Figures

The commands here are designed to establish a specific file permission strategy within the ‘project\_files’ directory. By setting distinct permissions for users, groups, and others, they align with the specifications of granting comprehensive permissions to users and more restrictive access to groups and others. This step is crucial for balancing collaborative access with security. The rational for the chosen permissions is specified at the end of this task.

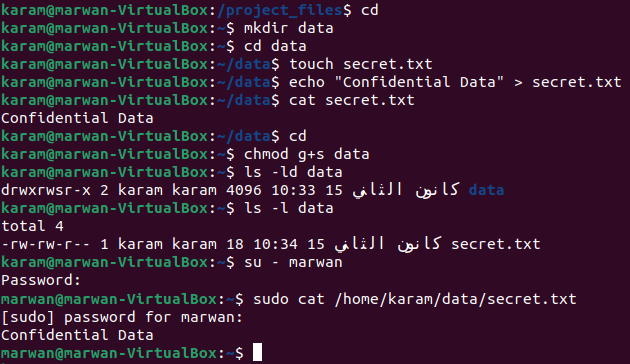
* **sudo chmod -R u=rwx /project\_files/.**: Sets full permissions for the user, over all files and directories in the ‘/project\_files’ directory.
* **sudo chmod g=x /project\_files** and **sudo chmod o=x /project\_files**: Sets only execute permissions for the groups and the other for the ‘/project\_files’ directory.
* **sudo chmod 640 /project\_files/AllFiles**: Sets read and write permissions for the user, read-only for the group, and no permissions for others on files in ‘/project\_files’.
* **sudo ls -l /project\_files** and **ls -ld /project\_files**: Verifies the implemented permissions.



Part 7 Figure

This task involves ‘karam’ verifying their access to the ‘project\_files’ directory, in line with the set permissions. It’s a practical demonstration of how the file permissions affect a team member’s access, ensuring that the permission architecture supports both collaboration and security.

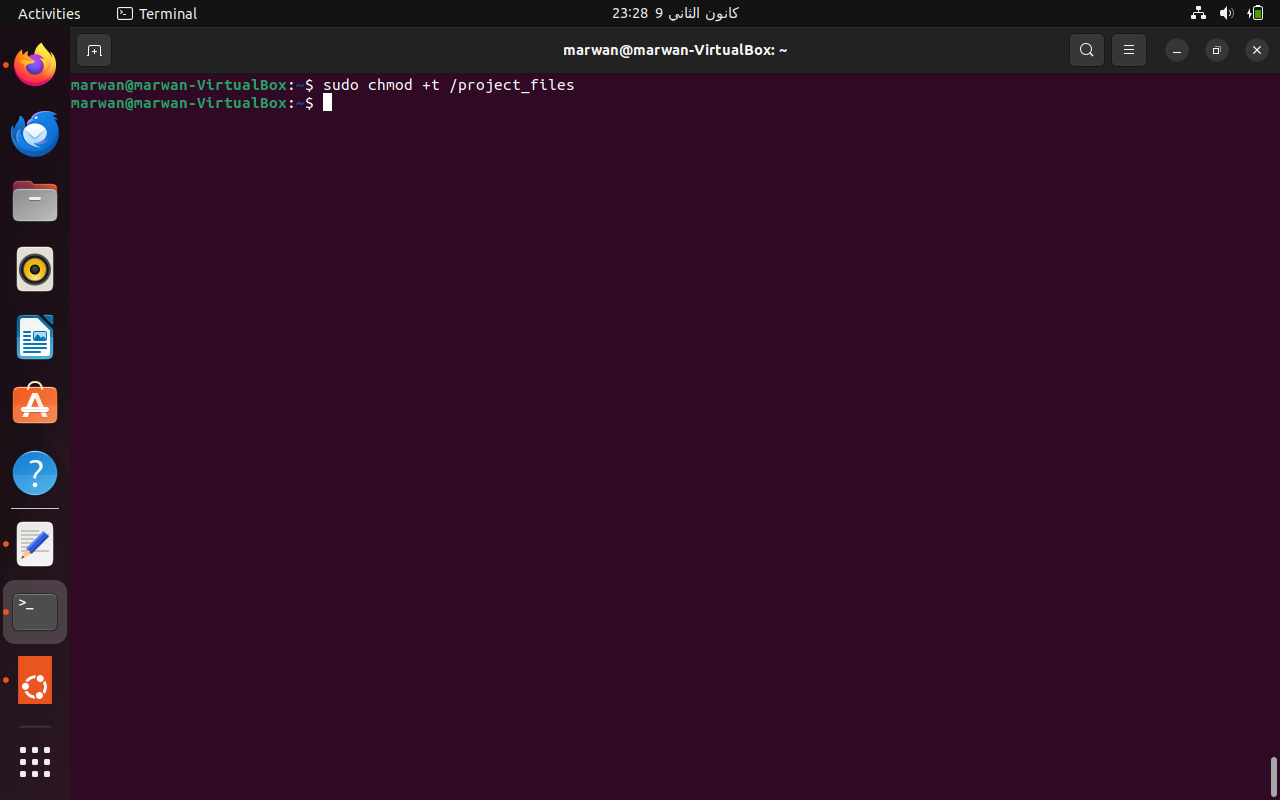
* **su- karam** and **cd /project\_files**: The series of commands demonstrates ‘karam’ logging in, navigating to the ‘/project\_files’.
* **ls -ld /project\_files**: Check execute permission for directory. It shows that karam has the execute permission on the directory as expected from a member in TechTeam.
* **touch /project\_files/newfil.txt**: Check write permission for directory. It shows that karam doesn’t have the write permission on the directory as expected from a member in TechTeam.
* **ls /project\_files**: Check read permission for directory. It shows that karam doesn’t have the read permission on the directory as expected from a member in TechTeam.
* **./test**: Check execute permission for file. It shows that karam doesn’t have the execute permission on the file as expected from a member in TechTeam.
* **echo “Hello Again!” >> /project\_files/File1.txt**: Check write permission for file. It shows that karam doesn’t have the write permission on the file as expected from a member in TechTeam.
* **cat /project\_files/File.txt**: Check read permission for file. It shows that karam has the read permission on the file as expected from a member in TechTeam.



Part 8 Figure

The manipulation of group membership to access a restricted file addresses a unique scenario where access to sensitive information is required without altering file permissions or overtly changing group settings. This demonstrates an advanced understanding of group and user permissions in Linux.

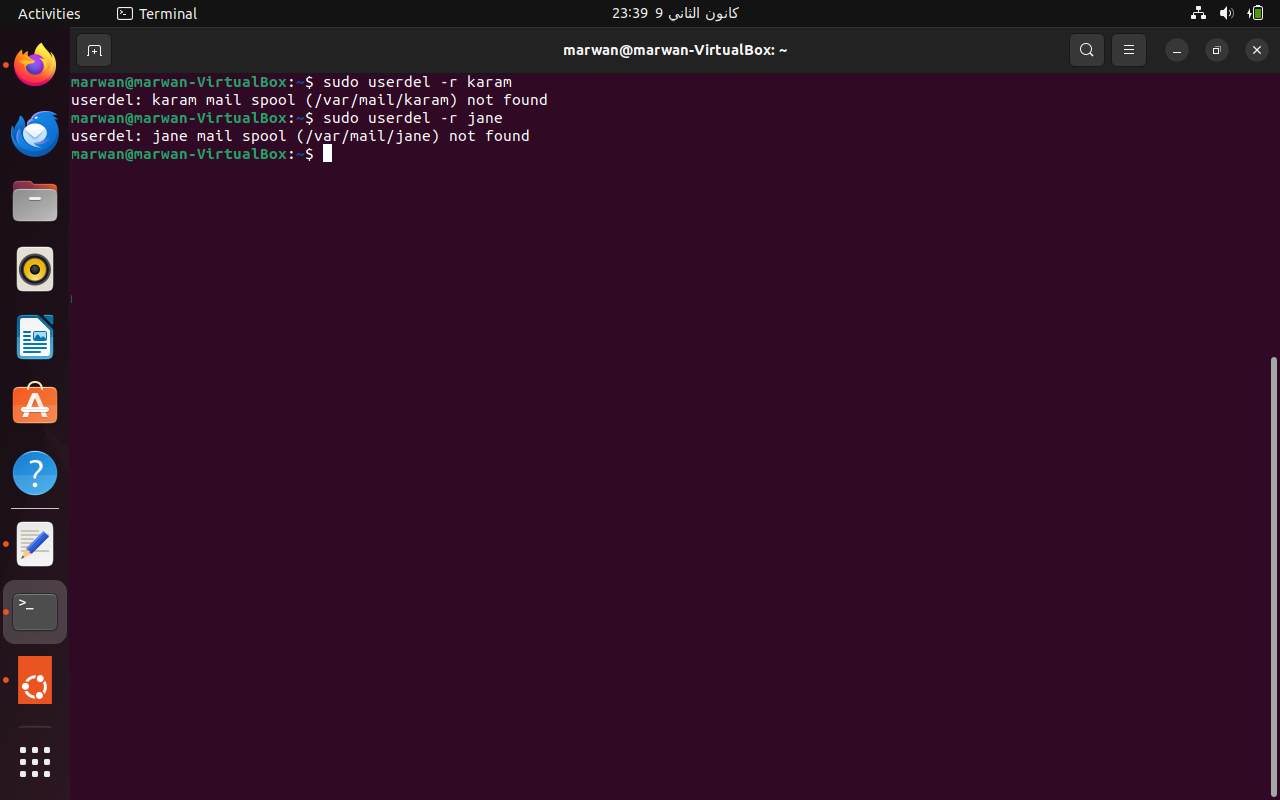
* **cd**, **mkdir data**, and **cd data**: navigating to and creating the data folder.
* **touch secrets.txt**, **echo**, and **cat secret.txt**: karam creates a new file named ‘secrets.txt’ and puts some text in it, and then reads it.
* **chmod g+s data**: karam sets the set-group-ID (SGID) bit on the directory data, which makes new files and subdirectories created within this directory inherit the group ownership of the directory, rather than the primary group of the user who created the file
* **su - marwan**: Switches the current user to ‘marwan’, a user who is not a member of ‘TechTeam’.
* **sudo cat /home/karam/data/secret.txt**: Reads the contents of ‘secrets.txt’.



Part 9 Figure

Setting the sticky bit on the ‘project\_files’ directory ensures that users can only delete files they own as part of security measures. This step is in direct response to the need for preventing users from deleting each other’s files, thus maintaining an organized and respectful shared workspace.

* **sudo chmod +t /project\_files**: Sets the sticky bit on ‘/project\_files’, preventing users from deleting files owned by others within the directory.



Part 10 Figure

Finally, the removal of ‘karam’ and ‘jane’ and their directories signifies the completion of the project and the decommissioning of user accounts and resources. This aligns with the final phase of the project specifications, ensuring that no unnecessary user accounts or files are left on the system.

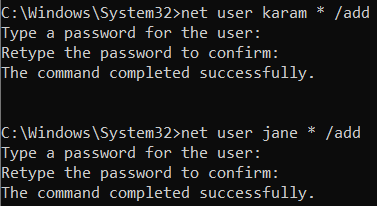
* **sudo userdel -r karam** and **sudo userdel -r jane**: These commands remove the users ‘karam’ and ‘jane’ and their associated directories, completing the final phase of the project.

***Rationale for Implemented File Permissions in ‘project\_files’***

1. **User Permissions:** The full read and write permissions on files and directories, and then read-only permissions for users ensure that they have very high control over files and directories they own. This level of control is essential for the primary contributors to the project, allowing them to create, modify, and delete their files as needed. This approach aligns with the operational requirements for active project development, where team members need to frequently update and manage their work.
2. **Restrictive Group Permissions:** Read-only access for group members on files is a strategic decision. It allows all members of the ‘TechTeam’ to view and review each other’s work, fostering a collaborative environment. Execute permission on the directory ensures that team members can traverse the directory structure, essential for navigating and organizing the workspace. This setup protects the integrity of individual contributions while promoting team collaboration. Group members can learn from and build upon each other’s work without the risk of unintended alterations.
3. **Restricted Access for Others:** By denying any permissions on files, and only execute permission on the directory to users outside of the project team (‘others’), we maintain a high level of security. This precaution ensures that sensitive project files are not exposed to unauthorized access or tampering. This strict boundary is crucial in a corporate environment where multiple teams may operate on the same server but require isolation from each other’s work.

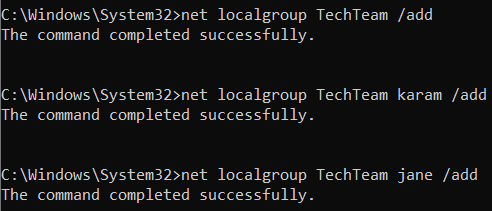
The chosen permissions represent a careful balance between collaboration and security. By granting full control to the owners (users) and restricting group members to read-only access, we encourage collaboration without compromising individual control over files. This balance is crucial in a team environment, where respect for individual work coexists with the necessity for team interaction and oversight. The exclusion of ‘others’ from accessing these files ensures that the team’s work remains confidential and secure from external interventions. This is particularly important in a corporate setting, where data security and integrity are paramount.

# ***Task 4***



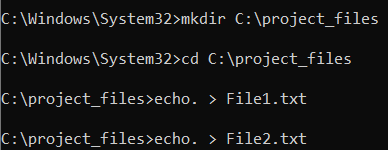
Part 1 Figure

This step involves creating two new user accounts on a Windows system. The command net user is used for managing user accounts. Here, two users named “karam” and “jane” are being added to the system. The asterisk (\*) after the username prompts for a password to be set for each user account.



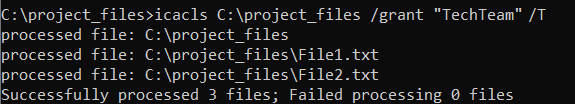
Part 2 Figure

In this part, a new user group named ‘TechTeam’ is created using the net localgroup command. This command is specifically used for managing local groups on a Windows system. After creating the ‘TechTeam’ group, the users ‘karam’ and ‘jane’ are added to this group. This is done by using the net localgroup command again, specifying the group name (‘TechTeam’), the user to be added, and the ‘/add’ flag.



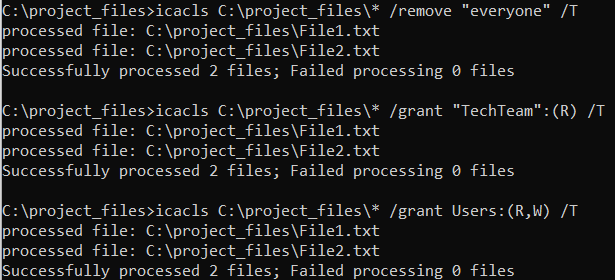
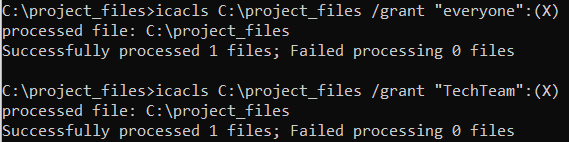
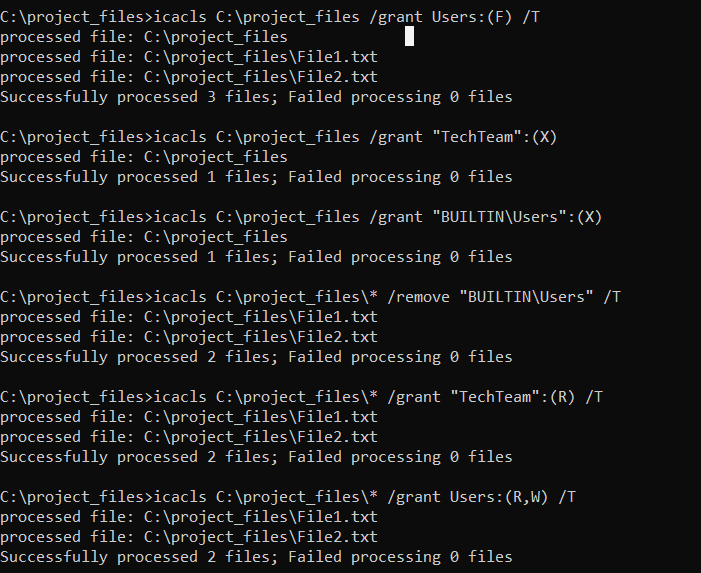
Part 3 Figure

Here, a new directory named ‘project\_files’ is created on the C: drive using the mkdir command. The cd command is then used to change the current directory to ‘C:\project\_files’. Finally, two new files, ‘File1.txt’ and ‘File2.txt’, are created inside the ‘project\_files’ directory using the echo. command. This command creates empty files with the specified names.



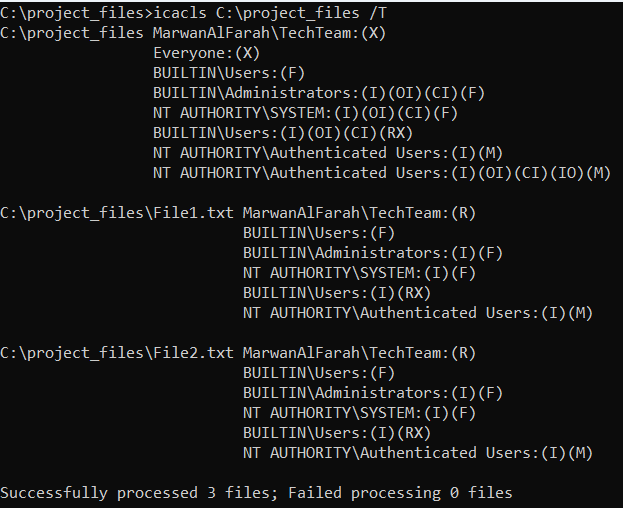
Part 4 Figure

This step involves changing the ownership of the ‘project\_files’ directory to the ‘TechTeam’ group. The icacls command is used for this purpose, which manages file and directory permissions in Windows. The /setowner flag changes the owner to ‘TechTeam’, and the ‘/T’ flag applies this change to all subfolders and files within the directory.



Part 5 Figure

The final part sets specific permissions for the ‘project\_files’ directory and its contents. The icacls command is used again with various flags. Firstly, we grant full control to all users for the ‘project\_files’ directory and its subfolders/files. Then, we restrict the ‘TechTeam’ group and ‘everyone’ (there is no direct equivalent to ‘others’ in Windows, but ‘everyone’ is the closest equivalent to ‘others’ in Windows CMD) to execute-only permissions for the director. Then, we specify the permissions for the files within ‘project\_files’. The ‘Users’ is granted read and write permissions, while the ‘TechTeam’ is granted read-only permissions. Finally, the ‘everyone’ are removed from having any specific permissions on these files.



# ***Task 5***

***(Tanenbaum and Steen, 2007; Bryan Soliman Blog, 2010; Silberschatz, Galvin and Gagne, 2017; Javatpoint, 2022b; Toppr, 2022; GeeksforGeeks, 2023c; Priya, 2023)***

***Definition and Overview***

A distributed operating system (DOS) is a kind of computing architecture and software infrastructure that joins several physical computers connected by a network into a connected functional system, giving users and applications an illusion that there is a single computing unit. Traditional centralized operating systems are made to run entirely on a single machine; this system architecture is quite different. This type of operating system is made for a network of linked computers, each of which serves as a node in the network.

Instead of residing on a single machine, the operating system functionality is distributed across various nodes, each containing its own processor, memory, and resources. This design provides high scalability and robustness, allowing programs to run smoothly across several servers, taking advantage of their combined processing power and storage capacity.

**Characteristics of Distributed Operating Systems**

* **Transparency:** One of the defining features is transparency. It makes the distributed nature of the system invisible to the user, where the complexity of the underlying network is masked, offering users and applications a single, unified view of the system. This involves the concealment of the complexities associated with data location, access, relocation, migration, replication, and failure of components within the system.
* **Openness:** DOSs are designed to support well-defined, universally agreed-upon interfaces. They support standard communication protocols and data formats, enabling interoperability among diverse systems.
* **Scalability:** DOSs are scalable both in terms of the size and geographical spread such that these systems can easily scale up or down, allowing for more nodes to be added to the network without significant disruptions. They handle an increasing number of nodes and manage resources across widespread locations without a performance decline.
* **Concurrency:** It allows multiple processes to run simultaneously across different machines, enhancing performance.
* **Fault Tolerance:** They are designed to provide continued service in the face of hardware or software failures, ensuring higher reliability.
* **Resource Sharing:** Distributed operating systems enable the sharing of both hardware and software resources, leading to optimized utilization.

***Key Differences from Centralized Operating Systems***

* **Resource Sharing:** In DOS, resources such as memory, processing power, and storage are spread across multiple machines. This contrasts with centralized operating systems where resources are confined to one machine.
* **Scalability:** DOS excels in scalability. New resources can be added easily without disrupting the existing system, a feature less fluid in centralized systems. However, with the COS, the system is not very scalable as it utilizes a single device.
* **Fault Tolerance:** Due to its multiple component nature, DOS can offer higher fault tolerance. The failure of one node often doesn’t halt the entire system, unlike in a centralized setup where the failure of the central unit can be lead for the whole system to stop working.
* **Performance:** Distributed systems can handle more computation load by distributing tasks across various nodes, which means that tasks can be dynamically distributed across various nodes, significantly accelerating computation-intensive workloads and offering better performance compared to a centralized system where all tasks rely on the resources of a single machine.
* **Processing:** Processing is conducted concurrently across multiple machines in DOS, enhancing computational power and efficiency, unlike the centralized system where processing is limited to the capabilities of a single machine. Whereas DOS makes use of several processors spread across various computers, COS depends on a single CPU.

**Tightly Coupled vs. Loosely Coupled Systems**

* **Tightly Coupled:** These systems, also known as multiprocessor systems, operate under a single operating system in which they feature processors that share a common memory space and operate in close synchronization, enhancing processing speed and inter-process communication. They offer high performance and a uniform memory access architecture but can be limited by the shared memory’s bandwidth and the complexity of managing memory consistency.
* **Loosely Coupled:** Also referred to as multicomputer systems, these consist of multiple CPUs connected through a communication network. Each processor has its own memory and copy of the operating system, and they communicate usually using message passing. Operating across a network of heterogeneous and independent machines, these systems emphasize fault tolerance flexibility and scalability, accommodating a variety of hardware and operating systems.
* **Centralized Operating Systems:** Centralized Operating Systems are traditionally ‘tightly coupled’ systems, where all processing is managed by a single processor or a tightly integrated group of processors. These processors share a common memory space and are designed to operate in unison, providing fast and efficient computational capabilities. The main goal of a Centralized OS is to manage system resources effectively, executing programs as if on a single processor. This coupling allows for easy communication between processes but also ties the system to a single point of failure.
* **Distributed Operating Systems:** (Tanenbaum and Steen, 2007) refers to it as a ‘tightly coupled’ system that spans across multiple machines or processors. In a DOS, multiprocessors work collaboratively, and in a homogeneous multicomputer environment where each node is similar in function and design. The primary objective of a DOS is to present the user with a single, coherent system image that hides the complexities of the distributed nature of the resources. Despite the physical distribution, the OS manages resources in a manner that makes them appear as a single entity to the user, thereby achieving transparency in distribution. However, (Toppr, 2022; GeeksforGeeks, 2023c) refer to it as a ‘loosely coupled’ system.
* **Network Operating Systems:** In contrast, Network Operating Systems (NOS) are considered ‘loosely coupled’ systems. They operate across a network of heterogeneous machines, potentially different in hardware and operating system, connected via Local Area Networks (LAN) or Wide Area Networks (WAN). The NOS manages each node independently, offering services to local and remote clients without the need for uniformity in the network. The decentralization in NOS allows for greater flexibility and scalability, as new nodes can be added without the need for homogeneity. However, this comes at the cost of lesser transparency, where the distribution of resources is more evident to the user.
* **Middleware:** It is a software, acting as an intermediary layer, where it enables communication and data management for distributed applications. It allows different components of an application to interact across network systems and various operating systems, ensuring that the application functions as a unified whole.

***Real-World Application Example*** (Dalton *et al.*, 2018; Aggarwal, 2023)

Imagine this, millions of users across the globe, simultaneously bombarding Google with countless search queries. Each query demands complex calculations, data retrieval from massive databases, and near-instantaneous responses. How does Google deliver such unparalleled efficiency and scalability? The answer lies in its distributed operating system, Andromeda.

Instead than living on a single, powerful server, Andromeda manages a massive network of linked computers, each with an impressive amount of processing power and data storage. This distributed architecture empowers Google Search in several remarkable ways.

When a user launches a search, Andromeda dynamically distributes the workload across multiple nodes in the network. Each node simultaneously retrieves relevant data, performs calculations, and ranks potential results. This parallel processing significantly reduces overall processing time, delivering lightning-fast responses even under massive demand.

No machine is indestructible, but with Andromeda, a single server malfunction doesn’t bring the entire search engine down. Other nodes seamlessly pick up the slack, ensuring uninterrupted service for users. This fault tolerance guarantees high availability and prevents frustrating downtime, even during hardware failures.

As the number of users grows, Andromeda effortlessly adapts. Adding more nodes to the network expands processing power and data storage capacity, allowing Google Search to keep pace with ever-increasing demand. This scalability ensures that even as the global population searches the web, the experience remains consistently fast and efficient.

Andromeda efficiently manages the vast resources available across the network. Files, databases, and processing power are seamlessly shared, eliminating unnecessary duplication and maximizing resource utilization. This efficient resource management translates to cost benefits for Google and ultimately, a more affordable search experience for users.

In conclusion, Google Search serves as a shining example of how distributed operating systems unlock the true potential of cloud computing. From parallel processing to fault tolerance and resource optimization, Andromeda empowers Google to deliver unrivaled speed, reliability, and scalability, ensuring that billions of users across the globe can access information in an instant.

# ***Task 6***

***(Silberschatz, Galvin and Gagne, 2017; Akhand, 2018; PP. Pankaj, 2019; Wilson, 2019; Libre Texts, 2021; OMSCS Notes, 2021; Javatpoint, 2022a; Chakraborty, 2023)***

***Introduction to Concurrency***

Concurrency in operating systems refers to the system’s ability to manage many tasks or processes at the same time, which improves computer system efficiency and responsiveness. This principle is critical in modern computing, where fast processing and multitasking are essential. Concurrency allows an operating system to accomplish several tasks by rapidly switching between them, creating an illusion that they are all occurring at the same time. Concurrency is performed via a variety of approaches, including multiprocessing, multithreading, and asynchronous processing. In essence, concurrency is about dealing with several things at once (for example, managing numerous programs), whereas parallelism is performing multiple things at the same time.

Concurrency is not only a technological need, but also a strategic strategy to maximize computer resources. It plays an important role in optimizing the user experience by ensuring that applications respond fast, especially when under a lot of load. For example, with a GUI-based operating system, a user may desire to start a new program while watching a movie. Concurrency ensures that these tasks can be handled simultaneously without significant lag, thereby enhancing the overall system performance and responsiveness.

***Impact on Computer Performance***

The impact of concurrency on computer performance can be seen in two different lights: maximization and minimization of performance.

* **Maximization of Performance:** At first glance, the most evident benefit of concurrency is its ability to maximize performance, particularly in systems with multi-core processors, where different cores can handle different processes/threads simultaneously, drastically improving computational speed and efficiency. This parallel processing leads to a considerable increase in the speed and efficiency of computational tasks. For example, in a graphic rendering task, concurrency allows different aspects of the rendering process to be handled by different cores, thereby speeding up the process. Moreover, even in single-core processors, the operating system’s ability to manage multiple tasks concurrently means that the CPU can be efficiently utilized. It switches tasks when one is idle (e.g., waiting for I/O operations) or when an interrupt happened, or when the process exceeded its timer, thus reducing wasted CPU cycles and improving overall throughput.
* **Minimization of Performance:** On the flip side, concurrency can sometimes lead to a minimization of performance. This occurs mainly due to the overhead associated with managing multiple concurrent tasks, such as context switching and synchronization, which can consume significant system resources. When tasks are highly interdependent or when there are insufficient resources to handle multiple tasks efficiently, the performance might degrade. Moreover, concurrency introduces complexities such as race conditions, deadlocks, starvation and blocking. These issues can not only degrade performance but also lead to more critical system failures if not managed correctly.

***Technique to Achieve Concurrency: Multithreading***

One popular technique to achieve concurrency is multithreading. It involves dividing a single process into several smaller threads, with each thread handling a specific part of the overall task. This division allows for concurrent execution of these threads on multi-core processors, capturing the full capabilities of modern CPU architectures. By doing so, it enhances computational efficiency and performance.

Threads often compete for access to shared resources, such as memory and I/O channels. This competition can lead to conflicts and inconsistencies if not managed correctly. To address this, synchronization mechanisms are employed. These mechanisms function similarly to traffic lights in a busy intersection, regulating the access of threads to shared resources. They ensure that only one thread can access a particular resource at a given time, thus maintaining order and consistency.

However, these synchronization methods may potentially end up acting as a bottleneck. If they are overly restrictive, they can disrupt the efficient execution of threads, defeating the purpose of concurrency. Thus, it is crucial to find the ideal balance between synchronization and multithreading. In order to optimize performance and guarantee that the system runs as efficiently as possible without jeopardizing the use of the shared resources, this balance is essential.

**Example:** Consider an example of a complex graphical program, such 3D animation software, to show the value of multithreading. In an application like this, distinct threads may be in charge of handling system resource management, processing user input, and rendering various areas of the scene. The program may take full use of the processing power available by splitting these processes across many threads, which speeds up rendering times and improves user responsiveness. A web server is an additional example, as it must manage many requests from clients.The server can handle multiple requests concurrently by assigning a different thread to each client request using multithreading. This improves server efficiency and user experience by decreasing response times for individual clients and expanding the server’s ability to handle additional requests.

# ***Task 7***

***(Tanenbaum and Steen, 2007; Silberschatz, Galvin and Gagne, 2017; Rosencrance and Matturro, 2021; Javatpoint, 2022c; Awati, 2023; GeeksforGeeks, 2023b)***

***Remote Procedure Call (RPC)***

RPC is a technique that allows apps to communicate to one other over a network as if they were local calls in distributed operating systems. By enabling a program to call a remote server procedure in the same manner as a local process, RPC abstracts away the complexities of network communication. This is accomplished by offering a collection of communication methods and protocols that manage the complexities of data transfer throughout the network.

The process involves the client requesting that the server execute a procedure with particular parameters, then the server returns the operation’s results to the client. Building an accessible and efficient system of communication across processes running on various machines requires the use of this technology.

RPC encapsulates the network communication process, allowing a client program to execute a procedure on a remote server without the need to understand underlying network details. RPC uses a client-stub and server-stub model. The client-stub serves as a proxy for the actual procedure on the server. The client-stub packages the parameters of a remote procedure and sends a message to the server-stub when a client executes it. After the message has been unpacked and the process has been executed on the server by the server-stub, the client-stub receives the result and forwards it to the client.

Implementing RPC involves several challenges, including handling data representation differences between heterogeneous systems (marshalling), ensuring reliable communication over unreliable networks, and providing valid error handling mechanisms. Moreover, the system needs to manage network latency and optimize performance for different network conditions.

When distributed computing is crucial, like in cloud services, distributed file systems, and network services, RPC is frequently utilized. Its main benefit is that it’s easy to use and abstracts away some of the complexity of network communication, letting developers focus on the core logic of the program.

RPC primarily operates in a synchronous manner, where the client initiates a request and waits for a response from the server. This synchronous nature of RPC is evident in its basic operation and implementation. The client sends a request and blocks until the response is received, which is a typical synchronous interaction pattern. The server, on the other hand, waits for incoming requests and processes them upon arrival.

However, variations of RPC, like asynchronous RPCs, have been developed to address some limitations of the synchronous model, as it allows the client to continue its execution without waiting for the server’s response, which introduces a more flexible communication model, catering to scenarios where immediate response is not critical.

In call by value, the actual parameter value is sent to the procedure. This method is straightforward in RPC, as only the value is transmitted, and no state is shared between the client and server. RPC traditionally does not support call by reference as it implies sharing state between distributed components, which is complex and risky due to the disconnected nature of a network.

**Marshalling in RPC**

In RPC, marshalling involves more than just packaging procedure arguments into a message. The process needs to account for the fact that client and server machines might have different data representations, such as byte ordering. This necessitates a transformation of values into a sequence of bytes, which is then sent over the network.

There are several key aspects to RPC marshalling:

* **Data Representation Agreement:** Both client and server must agree on a common encoding for basic data values (like integers and floats) and complex data structures (like arrays and unions).
* **Parameter Passing Mechanics:** As mentioned above, RPC typically assumes a copy-in/copy-out semantics where parameters are copied for remote execution, and results are copied back. This process includes converting the data into a network-compatible format and then back into a machine-specific format at the destination.
* **Transparency Challenges:** Full access transparency is difficult to achieve with RPC. However, introducing remote reference mechanisms can enhance access transparency, allowing for a unified approach to remote data access.

**Stubs in Remote Procedure Call (RPC)**

In RPC, a stub represents a client-side proxy, that handles the server-side procedure as if it were a local procedure. When a client invokes a procedure, it is actually calling the stub. The primary responsibilities of the RPC stub include:

* **Parameter Marshaling:** The stub is responsible for marshaling the parameters of the procedure call. This process involves converting the procedure parameters into a form that can be transmitted over the network. This is crucial because different systems may have different data representations.
* **Communication Management:** After marshaling the parameters, the stub sends the procedure call request over the network to the server-side stub.
* **Result Handling:** Upon receiving the response from the server, the stub unmarshals the return values and presents them to the client application as if they were returned by a local procedure.

***Remote Method Invocation (RMI)***

RMI is a Java-centric methodology that extends the concept of RPC. While similar to RPC, RMI offers a more object-oriented approach and is specifically tailored for Java applications in a distributed environment, as it allows a Java object residing in one Java Virtual Machine (JVM) to invoke methods on an object in another JVM, which is facilitated by Java’s platform-independent nature, enabling RMI to seamlessly integrate with Java’s object-oriented design. RMI uses a registry service for locating remote objects and supports the transmission of any object that is serializable.

This is especially helpful for distributed systems since it combines easily with Java’s object-oriented programming approach. RMI manages the underlying network communications, giving the impression that a method is being called locally instead of across a network, which makes distributed Java application development easier.

Implementing RMI involves dealing with issues like object serialization, maintaining object state across different JVMs, and handling exceptions that may occur during remote method invocations. RMI also provides features to manage the lifecycle of remote objects and garbage collection in a distributed environment.

RMI’s strength lies in its ability to merge network programming with Java’s object-oriented features, making it ideal for developing distributed applications in Java. It is extensively used in enterprise applications, distributed computing projects, and client-server architectures where Java is the primary programming language.

RMI, similar to RPC, primarily supports synchronous communication. The client invokes a method on the server-side object (skeleton), which then waits for the operation to complete and a response to be returned. This synchronous interaction is central to the standard RMI model, where the client’s request is processed in a sequential order, and the client awaits the server’s response.

Asynchronous communication in RMI can be implemented, although it is not inherently part of the standard RMI framework. Asynchronous RMI can be realized through additional programming constructs or frameworks that allow methods to be invoked in a non-blocking manner. These constructs enable the client to proceed with other tasks without waiting for the server’s response, thereby increasing the efficiency and responsiveness of distributed applications.

In RMI, when an object reference is passed, the server can bind to the referenced object and invoke its methods. This approach enhances access transparency as it allows remote references to be used more naturally and efficiently compared to RPC. Remote references can also be passed as parameters in RMI, facilitating a more integrated and seamless distributed object model. RMI also allows for the passing of complete objects by value. In this scenario, the object, including its state and methods (or references to its methods), is marshaled and sent to the server. The server then unmarshals this object, effectively creating a copy of the original object. While this method supports more versatile interactions, it introduces complexity and potential issues related to object state management and consistency.

**Marshalling in RMI**

In contrast, RMI simplifies the marshalling process, especially when dealing with object references. The server can bind to a referenced object and invoke methods directly, simplifying the data transformation process. Additionally, RMI supports the passing of entire objects, either by reference or by value. This involves several steps:

* **Object-by-Value:** When an object is passed by value, it requires marshalling of its state and possibly its methods. The server then unmarshals this object, effectively creating a copy of the original object at the destination.
* **Object Reference:** Passing an object by reference is more straightforward. The server binds to the referenced object, invoking methods as needed, and unbinds when the object is no longer required.

**Stubs and Skeletons in Remote Method Invocation (RMI)**

* **Client Stub (Proxy):** The client stub in RMI acts as a local representative or proxy for the remote object. When a method of a remote object is invoked, the call is directed to the client stub.
* **Server Skeleton:** It receives method invocations from the client stub, decodes the marshaled parameters, identifies the corresponding method on the remote object, invokes the method on the remote object, and encodes the return value and sends it back to the client stub.
* **Interface Implementation:** The client stub implements the same set of interfaces as the remote object, making it appear to the client code as if it’s interacting directly with the remote object.
* **Method Invocation and Parameter Passing:** Similar to RPC, the RMI stubs and skeletons take care of marshaling and unmarshaling of method parameters and return values. Additionally, they handle the complexities of object serialization, which is crucial for preserving the state and behavior of objects during remote method invocations.
* **Handling Remote Object References:** In RMI, skeletons also deal with remote object references, which involves resolving these references to actual objects on the server side.

***Facilitating Distributed Operating Systems through RPC and RMI***

RPC and RMI are both essential for the operation of distributed operating systems since they allow for smooth inter-process communication across several machines in a network.

* **Simplification of Complex Processes:** They abstract the complexity of network programming by providing a higher-level communication interface, which significantly reduces development time and increases system efficiency while allowing developers to focus on the core logic of their applications without delving into the complexities of network programming, allowing developers to write distributed applications as if they were local applications.
* **Enhanced Scalability and Flexibility:** By enabling processes to communicate across different machines, RPC and RMI contribute to the scalability of distributed systems, as they allow for the distribution of workload across multiple servers, which not only improves the performance of individual applications but also enhances the overall efficiency of the system.
* **Interoperability and Integration:** RPC and RMI enhance the interoperability of applications in a distributed environment, as is vital for the integration of diverse systems within a distributed environment. Multiple systems can successfully interact thanks to RPC’s flexible protocol and RMI’s object-oriented approach. This leads to an effective distributed operating system of which resources and data is shared and used more effectively.
* **Resource Sharing:** Distributed applications can leverage RPC and RMI to access and utilize resources available on other machines across the network, promoting resource sharing and efficient utilization.
* **Location Independence:** Applications don’t need to be aware of the physical location of resources or services. They can invoke procedures or methods wherever they exist in the distributed system, leading to location-independent application development.
* **Efficiency:** RPC and RMI can minimize network traffic and improve overall system performance by allowing code to be performed remotely, rather than passing vast volumes of data over the network for processing.
* **Encapsulation and Modularity:** RMI, with its object-oriented approach, promotes encapsulation and modularity, as it allows the development of distributed applications where remote objects can be easily reused.

# ***Task 8***

***(Watts and Raza, 2019; Google Cloud, 2020; Wesley Chai, 2021; Big Commerce, 2022; IBM, 2022; Chai, 2023; Wesley Chai, Kate Brush and Stephen J. Bigelow, 2023)***

***Service Models in Cloud Computing***

1. **Infrastructure as a Service (IaaS):**

IaaS offers users the ability to rent computing resources from the cloud provider (such as virtual servers, storage, and networking hardware). Users have complete control over the operating systems, storage, and applications that are deployed. However, they have limited control over some networking components (such as host firewalls). IaaS requires a high level of technical skill from the user; however, this approach offers users more flexibility and personalization.

**Functionalities and Operations of Modern and Distributed OS in IaaS:**

* **Virtualization:** IaaS uses virtualization technologies to create virtual machines (VMs) that act as independent servers within a shared physical infrastructure, enabling multiple users to share physical hardware securely.
* **Resource Management:** IaaS provide tools for users to manage virtual resources efficiently, whereas the cloud provider manages all parts of the physical infrastructure.
* **High Availability:** IaaS usually uses distributed systems and redundant infrastructure to ensure high availability, which means applications running on IaaS can withstand hardware failures or outages without significant downtime.
* **Customization and Control:** IaaS allows users have the ability to customize their operating systems and software stacks according to their specific needs, which is essential for businesses that require business-specific computing environments.

1. **Platform as a Service (PaaS):**

PaaS provides an already set up platform for creating, deploying, and executing applications. This allows for a more efficient development process without the difficulties of infrastructure administration. This architecture is especially useful for developers that want to construct, operate, manage, and deploy their applications without worrying about the underlying infrastructure (storage, servers, or network administration). This platform comprises of operating systems, programming languages, middleware, database management systems, business analytics services, and development tools.

**Functionalities and Operations of Modern and Distributed OS in PaaS:**

* **Containerization:** PaaS often relies on containerization technologies like Docker to package applications with their dependencies into isolated units. This simplifies deployment, scalability, and portability compared to traditional VMs.
* **Orchestration:** PaaS platforms often include orchestration tools like Kubernetes to automate the deployment, scaling, and management of containerized applications across multiple servers.
* **Serverless Computing:** Some PaaS platforms offer serverless computing environments in which customers can deploy code without managing servers, allowing for a pay-per-use execution approach that lowers operating expenses.
* **Abstraction:** In PaaS, the operating system is abstracted, minimizing the need for users to manage OS-level configurations. This abstraction layer simplifies the development process.
* **Automated System Tasks:** PaaS automates crucial system tasks like scaling, load balancing, and resource allocation. This automation streamlines the application deployment and operational process for developers.
* **Support for Multiple Languages and Frameworks:** The flexibility that PaaS platforms offer in terms of supporting a broad range of programming languages and frameworks is essential for developers who wish to create and implement a range of software solutions.

1. **Software as a Service (SaaS):**

Over the internet, SaaS provides complete software applications as a service. The distributed operating system is fully controlled by the cloud provider as users access the program via a web browser or a mobile app without having to handle any platform or infrastructure components. Most SaaS apps are multi-tenant, which allows several users to share them with their own setups and data. SaaS apps are developed on a distributed operating system that is not visible to the user. The program itself, not the operating system underneath, is the main emphasis. Because users may access complex programs using straightforward web interfaces or APIs, this offers a seamless user experience. SaaS apps include anything from office suites and email to complex corporate programs like ERP and CRM. This model eliminates the need for installing and running applications on individual computers, streamlining maintenance and support.

**Functionalities and Operations of Modern and Distributed OS in SaaS:**

* **Multi-tenancy:** SaaS platforms must effectively isolate data and resources for different tenants while ensuring performance and security. This requires specialized architectures and operating system configurations.
* **Security:** SaaS providers are responsible for securing the underlying infrastructure and application software. This involves robust security measures like encryption, access controls, and vulnerability management.
* **Identity Management:** SaaS platforms typically integrate with identity and access management (IAM) systems to authenticate and authorize users for different applications and data.
* **Automatic Updates:** In order to guarantee that customers always have access to the newest features and security patches, SaaS applications are usually offered using a web-based model that allows automated updates and patches without user interaction.
* **Managed Distributed Operating System:** In SaaS, the cloud provider fully manages the distributed operating system, keeping it hidden from the end user. This management includes regular updates and maintenance tasks.
* **Simple User Interfaces for Complex Applications:** SaaS enables users to access complex applications through straightforward interfaces, enhancing user experience and accessibility.
* **Global Accessibility and Security:** The distributed OS in SaaS supports features like global accessibility, data redundancy, and robust security protocols, ensuring that applications are both widely accessible and secure.

Selecting the best service model is dependent on a number of criteria, including the amount of control wanted, budget, technical proficiency, and particular demands. Understanding the features and operations of modern and distributed operating systems within their respective frameworks may assist enterprises in making informed decisions and leverage the full potential cloud computing.

# ***References***

Aggarwal, S. (2023) *Andromeda : Google Cloud Platform’s network virtualization stack*, *Medium*. Available at: https://medium.com/@mani.saksham12/googles-andromeda-23147b9af86b (Accessed: 24 January 2024).

Akhand, M. (2018) *Operating System: Threads and Concurrency*, *Medium*. Available at: https://medium.com/@akhandmishra/operating-system-threads-and-concurrency-aec2036b90f8 (Accessed: 20 January 2024).

Ashwathnarayana, S. (2023) *Understanding Huge Pages*, *Netdata Blog*. Available at: https://blog.netdata.cloud/understanding-huge-pages/ (Accessed: 18 January 2024).

Awati, R. (2023) *What is Remote Method Invocation (RMI)?*, *TheServerSide*. Available at: https://www.theserverside.com/definition/Remote-Method-Invocation-RMI (Accessed: 22 January 2024).

B. Rand (2023) *What Is Virtual Memory and How Does It Work? (With Example)*, *Indeed*. Available at: https://www.indeed.com/career-advice/career-development/virtual-memory (Accessed: 16 January 2024).

Big Commerce (2022) *SaaS vs. PaaS vs. IaaS: Examples & How to Tell Them Apart*, *eG Innovations*. Available at: https://www.bigcommerce.com/articles/ecommerce/saas-vs-paas-vs-iaas/%0Ahttps://www.bigcommerce.com/articles/ecommerce/saas-vs-paas-vs-iaas/#the-3-types-of-cloud-computing-service-models-explained (Accessed: 20 January 2024).

Bryan Soliman Blog (2010) *Distributed OS vs. Centralized OS*, *Bryan Soliman Blog*. Available at: https://bsolimantechhub.ca/2011/07/20/distributed-os-vs-centralized-os/ (Accessed: 22 January 2024).

Chai, W. (2023) *What is IaaS? Infrastructure as a Service Definition*, *TechTarget*. Available at: https://www.techtarget.com/searchcloudcomputing/definition/Infrastructure-as-a-Service-IaaS (Accessed: 20 January 2024).

Chakraborty, A. (2023) *Concurrency in Operating System*, *TutorialsPoint*. Available at: https://www.tutorialspoint.com/concurrency-in-operating-system (Accessed: 20 January 2024).

Chen, R. (2021) *What are the page sizes used by Windows on various processors?*, *Devblogs Microsoft*. Available at: https://devblogs.microsoft.com/oldnewthing/20210510-00/?p=105200 (Accessed: 18 January 2024).

Dalton, M. *et al.* (2018) *Andromeda: Performance, Isolation, and Velocity at Scale in Cloud Network Virtualization*. Available at: https://www.usenix.org/conference/nsdi18/presentation/dalton (Accessed: 24 January 2024).

Dancuk, M. (2022) *Swappiness: What it Is, How it Works & How to Adjust*, *phoenixNAP*. Available at: https://phoenixnap.com/kb/swappiness (Accessed: 18 January 2024).

GeeksforGeeks (2022) *Virtual Memory in Operating System - GeeksforGeeks*, *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/virtual-memory-in-operating-system/?ref=lbp (Accessed: 16 January 2024).

GeeksforGeeks (2023a) *Multilevel Paging in Operating System*, *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/multilevel-paging-in-operating-system/ (Accessed: 23 January 2024).

GeeksforGeeks (2023b) *Remote Procedure Call (RPC) in Operating System*, *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/remote-procedure-call-rpc-in-operating-system/ (Accessed: 22 January 2024).

GeeksforGeeks (2023c) *What is a Distributed Operating System?*, *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/what-is-a-distributed-operating-system/ (Accessed: 22 January 2024).

Google Cloud (2020) *PaaS vs IaaS vs SaaS: What’s the difference?*, *Google Cloud*. Available at: https://cloud.google.com/learn/paas-vs-iaas-vs-saas (Accessed: 20 January 2024).

Hacker News (2020) *x86 has 4k, 2MB, and 1GB page sizes.*, *Hacker News*. Available at: https://news.ycombinator.com/item?id=24886428 (Accessed: 18 January 2024).

IBM (2022) *IaaS vs. PaaS vs. SaaS*, *IBM*. Available at: https://www.ibm.com/topics/iaas-paas-saas (Accessed: 20 January 2024).

Javatpoint (2022a) *Concurrency in Operating System*, *Javatpoint*. Available at: https://www.javatpoint.com/concurrency-in-operating-system (Accessed: 20 January 2024).

Javatpoint (2022b) *Distributed Operating System*, *Javatpoint*. Available at: https://www.javatpoint.com/distributed-operating-system (Accessed: 22 January 2024).

Javatpoint (2022c) *Remote Method Invocation (RMI)*, *Javatpoint*. Available at: https://www.javatpoint.com/RMI (Accessed: 22 January 2024).

Kanade, V. (2023) *Virtual Memory: Overcoming the Limitations of Physical Memory*, *Spiceworks*. Available at: https://www.spiceworks.com/tech/devops/articles/what-is-virtual-memory/ (Accessed: 16 January 2024).

Kasiya, C. (2023) *What Is Memory Compression in Windows and How Do You Enable or Disable It?*, *Make Use Of*. Available at: https://www.makeuseof.com/windows-memory-compression-guide/ (Accessed: 18 January 2024).

Learn, M. (2023) *What is VMM?*, *Microsoft Learn*. Available at: https://learn.microsoft.com/en-us/system-center/vmm/overview?view=sc-vmm-2022 (Accessed: 18 January 2024).

Libre Texts (2021) ‘5.1: Introduction to Concurrency’, *Libre Texts*. Available at: https://eng.libretexts.org/Courses/Delta\_College/Operating\_System%3A\_The\_Basics/05%3A\_Process\_Synchronization/5.1%3A\_Introduction\_to\_Concurrency (Accessed: 20 January 2024).

Moose, S. (2016) *Divergent Memory Management in Windows & Linux: A Comparative Analysis Free Essay Example*, *Study Moose*. Available at: https://studymoose.com/differences-in-memory-management-between-windows-and-linux-essay (Accessed: 18 January 2024).

OMSCS Notes (2021) *Threads And Concurrency*, *OMSCS Notes*. Available at: https://www.omscs-notes.com/operating-systems/threads-and-concurrency/ (Accessed: 20 January 2024).

PP. Pankaj (2019) *Concurrency in Operating System*, *Geeks For Geeks*. Available at: https://www.geeksforgeeks.org/concurrency-in-operating-system/ (Accessed: 20 January 2024).

Priya, B. (2023) *What is a distributed Operating System?*, *TutorialsPoint*. Available at: https://www.tutorialspoint.com/what-is-a-distributed-operating-system (Accessed: 22 January 2024).

Raut, M. (2022) *Memory management in windows vs Linux*, *Medium*. Available at: https://medium.com/@mitali.raut19/memory-management-in-windows-vs-linux-a3f7e8514d8b (Accessed: 18 January 2024).

Rosencrance, L. and Matturro, B. (2021) *What Is Remote Procedure Call (RPC)?*, *Techtarget*. Available at: https://www.techtarget.com/searchapparchitecture/definition/Remote-Procedure-Call-RPC (Accessed: 22 January 2024).

Silberschatz, A., Galvin, P.B. and Gagne, G. (2017) ‘Operating System Concepts Essentials’, *WILEY* [Preprint].

Tanenbaum, A.S. and Steen, M. Van (2007) *Distributed Systems: Principles and Paradigms*, *Computer Communications*. Available at: https://doi.org/10.1016/0140-3664(95)90078-0.

The Linux Kernel documentation (2021) *Page Tables - The Linux Kernel documentation*, *The Linux Kernel documentation*. Available at: https://docs.kernel.org/mm/page\_tables.html (Accessed: 18 January 2024).

Toppr (2022) *Distributed Operating System: Definition, Types and Characteristics*, *Toppr*. Available at: https://www.toppr.com/guides/computer-science/computer-fundamentals/operating-system/distributed-operating-system/ (Accessed: 22 January 2024).

Travis (2022) *Windows Vs Linux: A Comparison Of Memory Management*, *Systran Box*. Available at: https://www.systranbox.com/windows-vs-linux-a-comparison-of-memory-management/ (Accessed: 18 January 2024).

Tutorlalspoint (2023) *Prepaging In Operating Systems*, *TutorialsPoint*. Available at: https://www.tutorialspoint.com/prepaging-in-operating-systems (Accessed: 18 January 2024).

Ukessays (2018) *Compare the memory management of windows with linux*, *Ukessays*. Available at: https://www.ukessays.com/essays/engineering/compare-the-memory-management.php?vref=1 (Accessed: 18 January 2024).

Watts, S. and Raza, M. (2019) *SaaS vs PaaS vs IaaS: What’s The Difference & How To Choose*, *BMC Blogs*. Available at: https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/ (Accessed: 20 January 2024).

Wesley Chai, K.C. (2021) *What is SaaS (Software as a Service)? Everything You Need to Know*, *SearchCloudComputing*. Available at: https://searchcloudcomputing.techtarget.com/definition/Software-as-a-Service (Accessed: 20 January 2024).

Wesley Chai, Kate Brush and Stephen J. Bigelow (2023) *What is PaaS? Platform as a service definition and guide*, *TechTarget*. Available at: https://www.techtarget.com/searchcloudcomputing/definition/Platform-as-a-Service-PaaS (Accessed: 20 January 2024).

Wilson, C. (2019) *Multithreading and concurrency fundamentals*, *Educative.Io*. Available at: https://www.educative.io/blog/multithreading-and-concurrency-fundamentals (Accessed: 20 January 2024).