Task 1:

Explain the concept of virtual memory and its role in the memory management process of operating systems, and how does virtual memory enhance the overall system performance.

Virtual Memory is a key concept in computer systems, designed to extend the apparent memory capacity beyond the physical limitations of RAM. It creates an illusion of a larger memory space using a combination of hardware and software.

Virtual memory allows secondary memory (like hard drives or SSDs) to be treated as part of the main memory. It separates the addresses used by programs from actual physical storage locations. Addresses generated by programs are automatically translated to machine addresses. It involves a hierarchy that includes the system’s memory and a disk. This setup enables processes to function with only portions of their address space in memory at any time. The virtual memory manager, a software component, plays a critical role in this, managing the allocation of memory and disk space. The size of virtual memory isn't limited by the number of main storage locations but by the computer system's addressing scheme and the amount of secondary memory available. Virtual memory is implemented through a combination of hardware and software. It maps virtual addresses used by a program to physical addresses in the computer's memory.

Virtual memory is an essential component in the memory management of operating systems, offering a range of benefits that significantly enhance system performance. It enables systems to use more memory than the available physical RAM by utilizing a portion of the hard drive as an extension. This creates an illusion of a larger memory space, allowing for the execution of larger applications and processes. Virtual memory improves the efficiency of physical memory usage by selectively loading only the necessary parts of a process into RAM, while the rest remain on the disk.

Each process is provided with its own isolated virtual address space, which ensures that processes do not interfere with each other's memory. This isolation enhances system stability and security. The abstraction of the memory allocation process is another advantage of virtual memory. Programs use virtual addresses, and the operating system transparently handles the mapping to physical memory, thereby simplifying programming tasks and reducing the complexity for developers.

Dynamic allocation of memory to processes as needed is achieved through techniques like demand paging or segmentation. This flexibility is more efficient compared to static allocation methods and allows for the handling of processes that are larger than the main memory. Virtual memory also enables higher levels of multiprogramming, improving CPU utilization and overall system throughput.

Furthermore, virtual memory efficiently manages page faults and the swapping of pages in and out of memory. In situations where RAM is full, it uses algorithms such as LRU (Least Recently Used) or FIFO (First-In-First-Out) to determine which pages to swap out. This management is crucial in preventing and controlling thrashing, a condition of excessive paging that can significantly hinder performance. The system monitors and adjusts the level of multiprogramming to avoid such scenarios.

Finally, the performance of virtual memory systems is carefully measured and optimized based on factors like the page fault rate and effective access time. This involves balancing the time taken for memory access and the service time for page faults, ensuring optimal performance of the system.

**Task 2:**

1. Number of Page Tables Required

To determine the number of page tables required for a process with a 44-bit address space, a 16 KB page size, and a 16-byte entry size for the page table, heres the calculation step by step:

Step 1: Calculate the number of pages:

With a 16 KB (2^14 bytes) page size, the process can have 2^44 / 2^14 = 2^30 pages. and we divided it into 2^14 because its bigger number

Step 2: Determine the size of the inner page table:

Each entry in the inner page table corresponds to a single page.

Since each entry is 16 bytes (2^4), so that's means that the size of the inner page table is 2^30 \* 2^4 = 2^34 bytes = 17,179,869,184 which is equivalent to 16 GB.

Step 3: Calculate the number of pages in the first outer page table:

The size of the first outer page table should ideally be less than or equal to the page size (2^14 bytes) to ensure efficient paging.

Therefore, the number of pages in the first outer page table is 2^34 / 2^14 = 2^20 pages.

Step 4: Determine the size of the first outer page table:

With 2^20 pages, each entry in the first outer page table corresponds to a page in the inner page table.

Each entry is 16 bytes (2^4), so the size of the first outer page table is 2^20 \* 2^4 = 2^24 bytes, which is equivalent to 16 MB.

Step 5: Calculate the number of pages in the second outer page table:

Similar to the previous step, the number of pages in the second outer page table is determined to be 2^24 / 2^14 = 2^10 pages. (and now we are done the number is less that 2^14)

Step 6: Determine the size of the second outer page table:

With 2^10 pages, each entry in the second outer page table corresponds to a page in the first outer page table.

Each entry is 16 bytes (2^4), so the size of the second outer page table is 2^10 \* 2^4 = 2^14 bytes, which is equivalent to 16 KB.

Therefore, the process will have an inner page table, a first-level outer page table, and a second-level outer page table. In total, there will be 3 page tables.

1. CPU Logical Address Division

The logical address in a paged memory system is divided into two parts: the page number and the offset within the page.

Page Number: With 2^30 pages, 30 bits are required to represent the page number. because the total number of pages in the logical address space determines how many bits are required for the page number and as this for the all them

Offset within the Page: With a 16 KB (2^14 bytes) page size, 14 bits are needed for the offset.

From the 44-bit logical address, it will be divided for four as we said three for the pages and also one for the offset that's leaves us to have :

10 bits for each (inner page table/first outer page table/second outer page table) and the rest and its 14 bits for the offset.

1. Logical Address Navigation in a Multi-level Page Table Hierarchy

In a paged memory management system, the logical address is structured to enable efficient translation to the corresponding physical address. This translation involves traversing a hierarchy of page tables. Here's a step-by-step breakdown of how this process unfolds:

Start with the Inner Page Table: (10 bits)

The journey begins with the innermost page table, also known as the Level 1 page table. This table is responsible for managing a significant portion of the address space.

Use the Bits of Index for Page Selection:

The first task is to identify the entry in the inner page table that corresponds to the desired page. To do this, the logical address provides a set of bits used as an index into the inner page table. These bits determine which entry in the inner page table is relevant for the address being accessed.

Retrieve the Physical Address of the First Outer Page Table:

Once the correct entry in the inner page table is located, it contains valuable information. Specifically, it holds the physical address of the first outer page table. This address points to the starting point of the next level in the page table hierarchy.

Proceed to the First Outer Page Table: (10 bits)

Armed with the physical address of the first outer page table, we move to the next level of the hierarchy. This is the Level 2 page table, which is responsible for a coarser level of address translation.

Use the Next Set of Bits for Entry Selection:

In the first outer page table, the logical address provides another set of bits, used as an index to select the relevant entry. These bits point to the entry that corresponds to the page being accessed within the inner page table.

Access the Physical Address of the Second Outer Page Table:

Within this selected entry of the first outer page table, there's critical information – the physical address of the second outer page table. This address is pivotal for reaching the final level of address translation.

Navigate to the Second Outer Page Table: (10 bits)

Armed with the physical address of the second outer page table, we progress to the outermost level of the page table hierarchy. This is the Level 3 page table, which fine-tunes the translation process.

Utilize the Last Set of Bits for Entry Selection:

In the second outer page table, the logical address yet again provides a set of bits for entry selection. These bits help pinpoint the exact entry that corresponds to the desired page within the first outer page table.

Obtain the Base Physical Address of the Desired Page:

The selected entry in the second outer page table contains a crucial piece of information – the base physical address of the desired page. This address forms the foundation upon which the final physical address will be built.

Incorporate the Offset for Precise Byte Location: (14 bits)

With the base physical address obtained, the final step involves applying the bits of the offset from the logical address. These bits indicate the precise byte location within the selected page. By adding the offset to the base physical address, the system successfully identifies the exact location of the data in the main memory.

In essence, the logical address acts as a guide through the intricate layers of the page table hierarchy. It uses specific bits at each level to navigate and retrieve the necessary information, culminating in the determination of the exact physical memory location where the desired data is stored. This hierarchical approach to memory management optimizes the translation process, ensuring efficient and accurate data access.

Linux:

Linux uses paging extensively as part of its virtual memory management. It employs a two-level page table structure, which includes the top-level page directory and lower-level page tables. This allows Linux to efficiently manage large address spaces and provides flexibility in memory allocation.

The page table structure can vary depending on the architecture. For example, on x86 systems, it typically uses a three-level page table structure (PML4, PDPT, Page Directory, and Page Table), while on x86-64 systems, it employs a four-level page table structure (PML4, PDP, PD, and PT).

Linux provides various page replacement algorithms, including the popular Least Recently Used (LRU) algorithm. These algorithms determine which pages to evict from physical memory when needed.

Linux also implements swapping, which involves moving entire processes or parts of processes to disk when memory is under pressure. This allows Linux to free up physical memory for more critical tasks.

Windows:

Windows, like Linux, uses paging extensively for virtual memory management. It employs a similar two-level page table structure, consisting of a Page Directory and Page Table Entries.

In Windows, the page table structure is also architecture-dependent. On x86 systems, it uses a two-level page table structure (Page Directory and Page Table), while on x86-64 systems, it employs a four-level page table structure (PML4, PDP, PD, and PT), similar to Linux.

Windows employs its page replacement algorithms, which can vary across different versions of the operating system. Common algorithms include FIFO (First-In-First-Out) and a variation of the LRU algorithm.

Windows includes memory management features like SuperFetch and ReadyBoost, which aim to optimize paging by predicting and preloading frequently used data into RAM or utilizing external storage devices as cache.

Differences:

One of the most apparent differences between Linux and Windows is the user interface. Windows provides a graphical user interface (GUI) by default, making it more user-friendly for many users. Linux, on the other hand, often requires command-line interactions, though it also offers various desktop environments with GUIs.

Linux is an open-source operating system, which means its source code is publicly available and can be modified by the community. Windows, on the other hand, is a proprietary operating system developed by Microsoft, and its source code is not freely accessible.

Windows has a vast ecosystem of software applications and games designed specifically for it, making it a preferred choice for many users, especially in the desktop and gaming markets. Linux, while growing in software availability, may have limitations in software compatibility.

Linux offers extensive customization and control over the operating system, making it a preferred choice for developers and server environments. Windows provides customization options but to a lesser extent.

Linux is typically distributed under open-source licenses like the GNU General Public License (GPL), allowing free use and modification. Windows requires licenses for most versions, which may incur costs for users and organizations.

**Task 3:**

Karam: loke&ljleF$&011

Jane: hlsdkj&jfdd%nKn4

A computer screen shot of a code

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

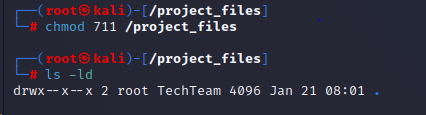


A screen shot of a computer

Description automatically generated

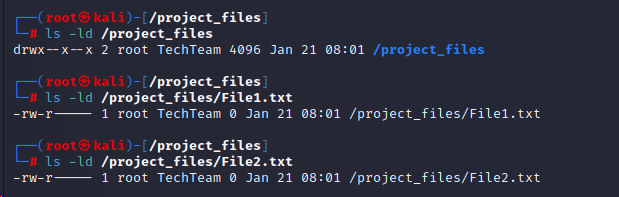
A screen shot of a computer program

Description automatically generated



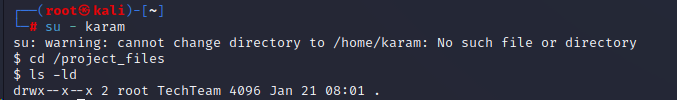
A screenshot of a computer program

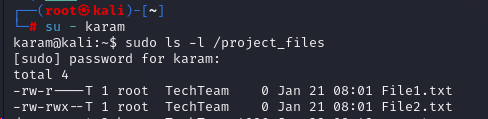
Description automatically generated



A computer screen shot of a computer screen

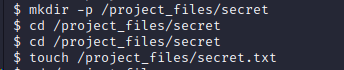
Description automatically generated





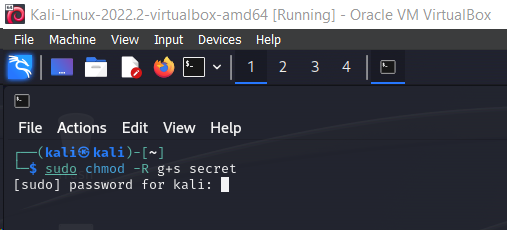
A screen shot of a computer

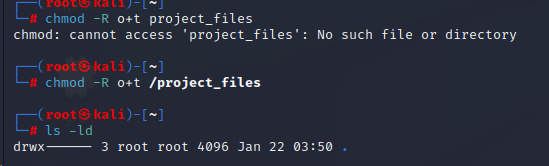
Description automatically generated

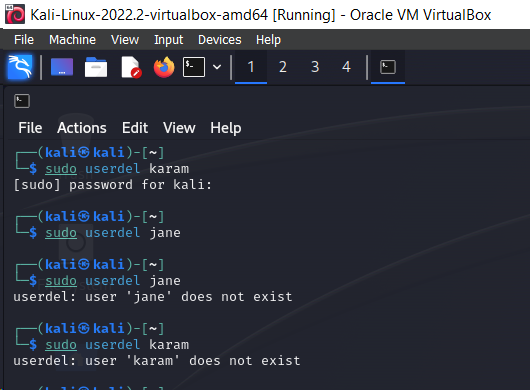


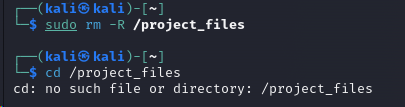
A screenshot of a computer

Description automatically generated









Task 4:

A black screen with white text

Description automatically generated

A black background with white text

Description automatically generated

Net LocalUser - "jane" -PasswordNeverExpires $false

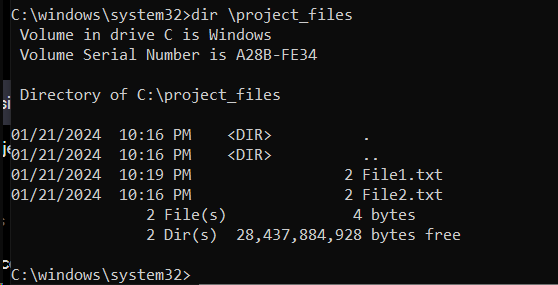
Net LocalUser "jane" -AccountExpires (Get-Date).AddDays(90)

A computer screen with white text

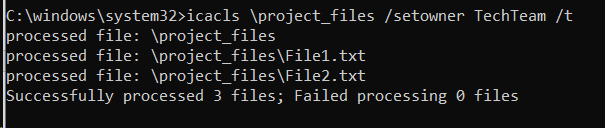
Description automatically generated

A screen shot of a computer program

Description automatically generated



1. Change it give the TechTeam the permission to the project\_files

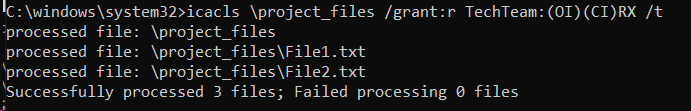


1. G

icacls \project\_files /grant:r users/name of the group/everyone or inheritance: (OI)(CI)F /t

A screen shot of a computer program

Description automatically generated



A black screen with white text

Description automatically generated

1. F

A screen shot of a computer program

Description automatically generated

A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

Task 5:

Distributed Operating Systems (DOS) and traditional centralized operating systems differ significantly in their architecture, management of resources, and scalability.

In traditional centralized operating systems, the focus is on managing the resources of a single computer or server. These systems handle tasks like task scheduling, memory management, and input/output operations for one specific computer. Users interact directly with these systems, whether through a personal computer or a central server in a client-server model. One of the limitations of these operating systems is their scalability, which is confined to the capacity of the single computer or server they manage. Enhancing their capabilities often requires additional infrastructure and management tools.

On the other hand, Distributed Operating Systems manage a network of independent computers, presenting them to the user as a single coherent system. This type of system excels in resource sharing and task execution across multiple networked computers, effectively coordinating resources like CPU, memory, and storage distributed across these systems. DOS is inherently scalable, enabling easy addition of new nodes to the network, and is designed to handle the complexity of distributed resources and tasks efficiently. One of the key features of DOS is its ability to maintain transparency in resource usage, making the distributed nature of tasks and resources seamless to the end user.

The core difference between these two types of operating systems lies in their scope and approach to resource management. While traditional operating systems are limited to managing resources within a single system, DOS extends its management capabilities across a network of interconnected computers. This fundamental distinction highlights the greater scalability and flexibility of DOS compared to the more localized and limited scope of traditional centralized operating systems.

Cloud computing platforms, such as Amazon Web Services, Google Cloud Platform, and Microsoft Azure, serve as prominent real-world examples of the effective use of Distributed Operating Systems (DOS). These platforms utilize DOS to manage and distribute resources like storage and processing power across a vast network of physical and virtual servers. This architecture enables cloud services to offer scalable, flexible, and reliable solutions, ensuring high availability and fault tolerance. DOS allows for efficient load balancing and resource utilization, facilitating global reach and consistent performance across geographically dispersed data centers. The distributed nature of these systems is crucial in supporting diverse cloud services, including Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), demonstrating the practical and versatile applications of DOS in managing complex, large-scale cloud computing environments.

Task 6:

Concurrency in operating systems is a fundamental concept that allows the execution of multiple instruction sequences simultaneously, leading to several processes or threads running in parallel. This ability is crucial in modern computing environments, enabling systems to efficiently handle multiple tasks and serve multiple users at once. Concurrency involves processes communicating through shared memory or message passing, and it introduces complexities such as deadlocks, where processes get stuck waiting for each other, and resource starvation, where a process may not get the necessary resources due to their allocation to other processes.

The implementation of concurrency aims to maximize system throughput by coordinating the execution of processes, efficiently allocating memory, and scheduling tasks. Motivations for allowing concurrent execution include physical resource sharing in multiuser environments, logical resource sharing like accessing the same file by different processes, speeding up computations through parallel execution, and achieving modularity by dividing system functions into separate processes. However, concurrency also presents challenges in managing global resources safely, optimizing resource allocation, and locating programming errors, which are often difficult to reproduce. Additionally, the operating system must handle the complexities of ensuring efficient execution without unnecessary blocking or performance overheads.

Despite these challenges, the advantages of concurrency are significant. It enables the simultaneous running of multiple applications, improves resource utilization (as resources unused by one application can be employed by others), enhances average response time, and overall leads to better system performance. However, these benefits come with the need for protective measures to ensure applications do not interfere destructively with each other and additional mechanisms for coordination.

Concurrency in operating systems, when managed effectively, generally maximizes the performance of a computer. It allows for the parallel execution of processes or threads, which can significantly enhance the system's throughput, especially in multi-user or multitasking environments. By efficiently utilizing the system's resources, such as CPU, memory, and I/O, concurrency enables multiple tasks to be processed simultaneously, improving overall system responsiveness and reducing idle time.

However, the performance benefits of concurrency are not without challenges. If not managed properly, concurrency can lead to issues like deadlocks, where two or more processes prevent each other from continuing, and resource starvation, where some processes are deprived of necessary resources. These issues can degrade system performance. Furthermore, the complexity of managing concurrent processes can introduce overheads, particularly in systems with many concurrent tasks. This can sometimes lead to a decrease in performance due to the additional system resources required for managing these processes.

Task 7:

Remote Method Invocation, or RMI, is a Java-based technology integral for building distributed applications. It enables an object in one Java Virtual Machine (JVM) to access and invoke methods on an object in another JVM. This capability is essential for Java programs to communicate remotely, and RMI is provided in the Java package `java.rmi`.

In RMI applications, the architecture typically involves two main programs: the server program and the client program. The server program hosts a remote object, and the client program requests and invokes methods on this remote object. Key components include the Transport Layer, which connects the client and server, the Stub and Skeleton, which act as client-side and server-side proxies for the remote object, and the Remote Reference Layer (RRL), which manages client references to the remote object.

The process starts with the client making a call to the remote object through the stub. This request is passed to the server's RRL and then to the skeleton, which finally invokes the method on the actual remote object. The results are returned to the client through the same path.

Marshalling, the process of packaging parameters for transmission, and unmarshalling, unpacking and invoking the method, are crucial steps in RMI. The RMI registry is also an important aspect, acting as a namespace for all server objects, where the server registers objects using unique bind names that clients use to fetch object references.

The goals of RMI are to minimize the complexity of distributed applications, ensure type safety, implement distributed garbage collection, and minimize differences between working with local and remote objects.

Remote Procedure Call (RPC) is a technique used in distributed, client-server based applications, extending local procedure calling to enable procedures to execute in different address spaces, possibly on different systems.

RPC involves the client invoking a client stub procedure, which then marshals the parameters into a message sent to the server. The server stub unmarshals the parameters, calls the desired server routine, and, post-execution, returns the results back to the client in a similar marshalled message format.

Key considerations in designing RPC systems include security, scalability, fault tolerance, standardization, and performance tuning. RPC faces issues like runtime management, stub functionality, binding (client-server linking), and call semantics (handling request retries, duplicate filtering, etc.).

RPC's advantages lie in its ability to abstract the complexities of network communication, optimize performance by omitting unnecessary protocol layers, facilitate distributed applications beyond local environments, minimize code rewriting, and support various models like process-oriented and thread-oriented approaches.

Task 8:

Infrastructure as a Service (IaaS)

IaaS provides virtualized computing resources over the internet, using a pay-as-you-go model, allowing businesses to rent IT infrastructure without the capital expense of owning hardware. It's flexible and scalable, ideal for variable workloads, new application testing, and high-performance computing. Features include rapid resource provisioning, geographical data center distribution, high reliability, and robust backup options. IaaS parallels a traditional OS in resource management (memory, processing, storage) but extends these functionalities across a distributed architecture. It supports dynamic resource scaling and complex network configurations, offering scalability and resilience beyond a single OS's capabilities.

Platform as a Service (PaaS)

PaaS offers a cloud platform for developing, testing, deploying, and managing software applications. It provides the infrastructure and additional tools like middleware, development tools, and databases. PaaS simplifies application development with pre-coded components and multi-platform support, managing the entire application lifecycle. It offers a comprehensive runtime environment, supporting various programming languages and frameworks, and integrated development environments. PaaS parallels an OS in managing applications, focusing on web applications and automating resource allocation and management in a distributed environment.

Software as a Service (SaaS)

SaaS delivers software applications over the internet, accessible through web browsers, encompassing the application, infrastructure, and platform. It provides scalable and accessible data management, user access, and authentication, eliminating the need for installation or maintenance. SaaS offers applications like email, calendaring, and office tools in a cost-effective model, managing security and compliance at network and application levels, akin to an OS but within a cloud-based context.

The integration of cloud services with modern and distributed operating systems (OS) is characterized by several key aspects. Firstly, distributed computing in cloud services extends the capabilities of an OS into a distributed environment. This involves managing resources across multiple servers and locations, thereby enhancing scalability and resilience beyond what traditional, centralized OS can provide. Secondly, cloud computing places a significant emphasis on automation and orchestration. This mirrors the advanced functions of an OS but is executed across distributed environments. It includes automated resource allocation, scaling, and management, showcasing a level of efficiency and complexity that surpasses conventional OS operations. Lastly, security and compliance are fundamental elements in cloud service models. These models manage security at both the network and application levels, adhering to stringent compliance standards. This approach parallels an OS's security management but extends it to a larger scale, ensuring robust security measures are in place across the dispersed and varied components of cloud computing.

**Resources**

*Virtual memory: What is virtual memory and how it is managed by os ?* (2020) *Learn Computer Science*. Available at: https://www.learncomputerscienceonline.com/virtual-memory/ (Accessed: 29 January 2024).

*Virtual memory: What is virtual memory and how it is managed by os ?* (2020) *Learn Computer Science*. Available at: https://www.learncomputerscienceonline.com/virtual-memory/ (Accessed: 29 January 2024).

Michigan Technological University (2023) *Five ways to improve your site’s ranking (SEO)*, *Michigan Technological University*. Available at: https://www.mtu.edu/umc/services/websites/seo/ (Accessed: 28 January 2024).

*Difference between linux and windows* (2023) *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/difference-between-linux-and-windows/ (Accessed: 29 January 2024).

(No date) *Remote procedure calls (RPC)*. Available at: https://users.cs.cf.ac.uk/Dave.Marshall/C/node33.html (Accessed: 29 January 2024).

:                                                  Lawrence Williams and Williams, L. (2023) *Remote procedure call (RPC) protocol in Distributed System*, *Guru99*. Available at: https://www.guru99.com/remote-procedure-call-rpc.html (Accessed: 29 January 2024).

*Distributed operating system - javatpoint* (no date) *www.javatpoint.com*. Available at: https://www.javatpoint.com/distributed-operating-system (Accessed: 29 January 2024).

*Java RMI - Introduction* (no date) *Tutorialspoint*. Available at: https://www.tutorialspoint.com/java\_rmi/java\_rmi\_introduction.htm (Accessed: 29 January 2024).

*Memory management in operating system* (2023) *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/memory-management-in-operating-system/ (Accessed: 29 January 2024).

Prepbytes (2023) *Distributed operating system : Types, features & applications*, *PrepBytes Blog*. Available at: https://www.prepbytes.com/blog/operating-system/distributed-operating-system/ (Accessed: 29 January 2024).

*Remote procedure call (RPC) in operating system* (2023) *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/remote-procedure-call-rpc-in-operating-system/ (Accessed: 29 January 2024).

*What is paas (platform-as-a-service)?* (no date a) *IBM*. Available at: https://www.ibm.com/topics/paas (Accessed: 29 January 2024).

*What is paas (platform-as-a-service)?* (no date b) *IBM*. Available at: https://www.ibm.com/topics/paas (Accessed: 29 January 2024).

*What is paas (platform-as-a-service)?* (no date c) *IBM*. Available at: https://www.ibm.com/topics/paas (Accessed: 29 January 2024).

*What is Paas? platform as a service: Microsoft Azure* (no date) *Platform as a Service | Microsoft Azure*. Available at: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-paas/ (Accessed: 29 January 2024).

*What is SAAS (software as a service)?* (no date) *Salesforce.com*. Available at: https://www.salesforce.com/saas/ (Accessed: 29 January 2024).

*What is Saas? software as a service: Microsoft Azure* (no date) *Software as a Service | Microsoft Azure*. Available at: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-saas (Accessed: 29 January 2024).