

To run the NFS server and client scripts by simply clicking on them, you can create a desktop launcher for each script.

Here are the steps to create a desktop launcher for the receiver script:

1. Create a new text file on your desktop and name it `receiver.desktop`.
2. Open the file in a text editor and enter the following lines:

```
[Desktop Entry]
```

```
Name=Receiver Script
```

```
Comment=Run NFS receiver script
```

```
Exec=/path/to/receiver_script.py
```

```
Icon=/path/to/icon.png
```

```
Terminal=true
```

```
Type=Application
```

```
Categories=Utility;
```

Replace `/path/to/receiver_script.py` with the actual path to your receiver script, and replace `/path/to/icon.png` with the path to an icon file for the launcher (if you want to use an icon). The `Terminal=true` option tells the launcher to open a terminal window when the script is run.

3. Save the file and exit the text editor.
4. Right-click on the `receiver.desktop` file and select "Properties."
5. Under the "Permissions" tab, make sure that the "Allow executing file as program" option is checked.

6. Close the properties window.
7. You should now be able to double-click on the `receiver.desktop` file to run the receiver script.

Repeat the above steps to create a desktop launcher for the sender script.

Note that when you run the scripts using the desktop launchers, the terminal window will remain open until the script finishes executing. If you want to close the terminal window automatically when the script finishes, you can modify the scripts to include the `exit` command at the end. For example, at the end of the receiver script, add the following line:

```
os.system("exit")
```



Sure, here's the Python program that executes the commands you listed:



python

Copy code

```
import os

# Adding user to sudo
os.system("sudo usermod -aG sudo <username>")

# Updating apt
os.system("sudo apt update")

# Installing net-tools
os.system("sudo apt install net-tools")

# Installing NFS kernel server
os.system("sudo apt install nfs-kernel-server")

# Installing NFS common
os.system("sudo apt install nfs-common")

# Agreeing to install
os.system("echo Y | sudo apt install <package_name>")

# Creating swap_share directory
os.system("sudo mkdir /swap_share")

# Setting permissions
os.system("sudo chmod 777 /swap_share")

# Creating swap_share directory again
os.system("sudo mkdir /swap_share")

# Setting permissions again
os.system("sudo chmod 777 /swap_share")

# Getting IP address and writing to file
os.system("ifconfig | grep inet > ip_.txt")
```

Make sure to replace `<username>` and `<package_name>` with the appropriate values for your system. You can save this code as a Python file (e.g. `setup.py`) and run it on your Linux machine using the following command:

arduino

Copy code

```
python setup.py
```

Note: This program assumes that you have sufficient permissions to execute these commands with sudo.

After analysis write documentation about analysis, reason, how and why is script working.

Repository from video:

https://www.youtube.com/watch?v=ypkItN4d4Y0&ab_channel=DanielPersson

<https://github.com/kalaspudding/simple-cdd-example/tree/master/profiles>

Both Comps on linux images

1. Adding user to sudo
2. Sudo apt update
3. Sudo apt install net-tools
4. Sudo apt install nfs-kernel-server
5. Sudo apt install nfs-common
6. Echo "Y"
7. Sudo Mkdir /swap_share
8. Sudo chmod 777 /swap_share
9. Sudo mkdir /swap_share
10. Sudo chmod 777 /swap_share
11. Ifconfig | grep inet > ip_.txt

*** Take IP address of other computer as input.

Sender Script.sh

#Install NFS server

```
sudo echo "Y" | apt-get install nfs-kernel-server nfs-common
```

```
#Shared Folder
```

```
sudo mkdir /swap_share
```

```
sudo chmod 777 /swap_share
```

```
#Edit Exports file
```

```
sudo echo "/swap_share <ip-address-of-the-provider-machine>(rw,sync,no_subtree_check)" >>  
/etc/exports
```

```
#Restart NFS Server
```

```
sudo systemctl restart nfs-kernel-server
```

```
#Create Swap File
```

```
sudo fallocate -l 4G /swapfile_network
```

```
sudo chmod 600 /swapfile_network
```

```
mkswap /swapfile_network
```

```
#Copy Swap File to Shared Folder
```

```
cp /swapfile_network /swap_share/
```

```
sudo chmod 777 /swap_share/swapfile_network
```

Reciever Script.sh

```
Install NFS server
```

```
sudo echo "Y" | apt-get install nfs-kernel-server nfs-common
```

```
#Restart NFS Server
```

```
sudo systemctl restart nfs-kernel-server
```

```
#Mount Folder
```

```
sudo mkdir /swap_mount
```

```
sudo chmod 777 /swap_mount
```

```
#Mount
```

```
sudo mount <ip-address-of-the-provider-machine>:/swap_share /swap_mount
```

```
#Turn on Mounted Swap File
```

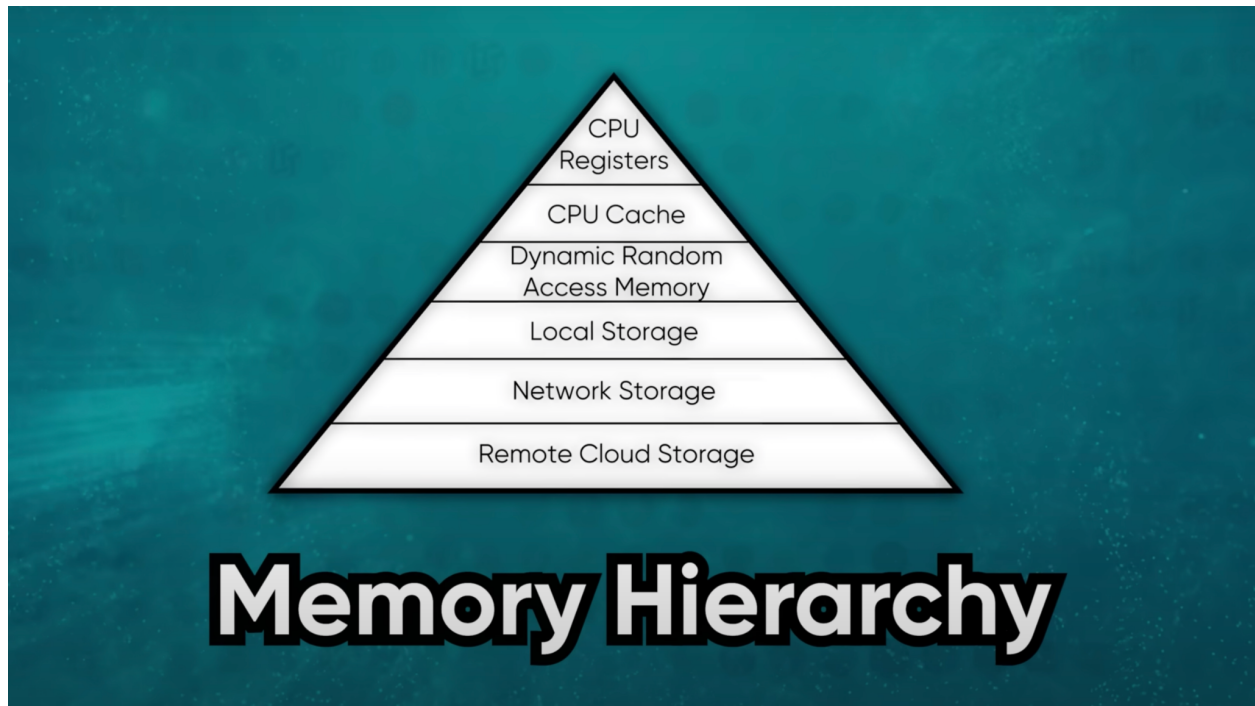
```
sudo swapon /swap_mount/swapfile_network
```


Memory hierarchy is a critical component of modern computer systems. One important layer of the memory hierarchy is swap space, which is a reserved area of a hard disk or other storage device that is used to temporarily store data that cannot be held in RAM. In this project, we present a new approach to memory hierarchy that involves connecting swap space over the network to increase processing power and physical resources. We propose a new product that enables the sharing of swap space over the network, allowing multiple computers to share resources and greatly increase processing power and physical resources. By leveraging the power of multiple computers, our product can help reduce processing times and improve the overall performance of computing systems. Our approach is particularly suited for use in distributed computing environments, where resources may be spread across different locations. We present a detailed design and implementation of our product, along with experimental results that demonstrate its effectiveness in increasing processing power and physical resources. Our product offers a new approach to memory hierarchy that can help to overcome many of the limitations of traditional memory hierarchy, and enable the development of more powerful and efficient computing systems.

https://www.youtube.com/watch?v=minxwFqinpw&ab_channel=LinusTechTips

Memory Hierarchy

CPU Register -> CPU cache -> Dynamic RAM -> Local Storage -> Network Storage -> Cloud



CPU has registers

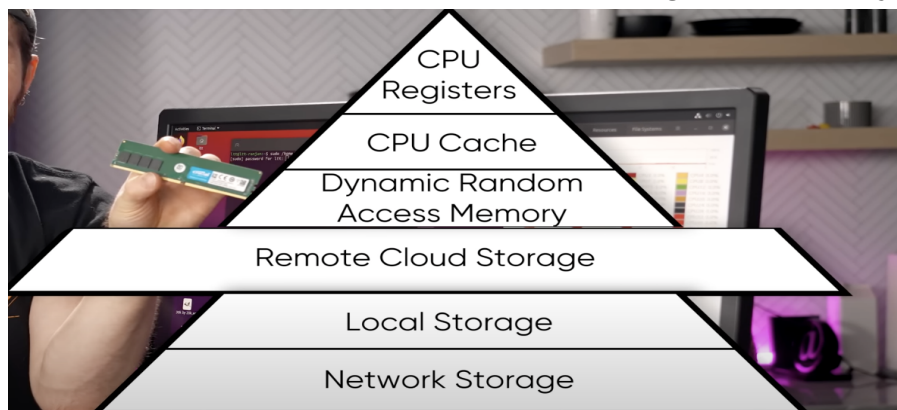
register stores which processor needs to work on.

Processor has Caches

DRAM connected to the CPU, giving quick access to data.

Then, Network (Internet) followed by cloud.

Linux OS lets us Change the Hierarchy.



Proof of Concept:

Swap Space: Swap space is a portion of a computer's hard disk that is used as an extension of the physical memory (RAM) in a computer. When a computer's physical memory (RAM) becomes full, the operating system will temporarily move data from RAM to the swap space on the hard disk. This allows the system to continue to operate smoothly even if it is running low on physical memory.

Swap space is an important component of a Linux-based system, as it helps the system to manage its memory resources more effectively, and to continue to run even when physical memory is running low. The amount of swap space that a system needs depends on the size of its physical memory and the types of applications that are running on the system.

In general, it is a good practice to have at least as much swap space as the amount of physical memory in the system, or to have at least 2-4GB of swap space on modern systems, regardless of the amount of physical memory. However, the actual amount of swap space that you need will depend on the specific requirements of your system and the applications that are running on it.

Add a swap memory on node 1 to node 2, by writing a script that:

- 1) Mount Drives**
- 2) Mount swapfile as a loop device**
- 3) Make Swap**
- 4) Setup Swap Space**
- 5) Turn swap space on.**

Monitor the swap space

Or

- 1) Create a swap file on the network-based storage device.**
- 2) On each Linux system that will use the shared swap space, mount the swap file as a swap space using the swapon command.**
- 3) Configure each Linux system to automatically mount the shared swap space on boot, by adding an entry for the swap file to the /etc/fstab file.**

This can be done as a proof of concept on docker VM's.

If success on running heavy program using the swap-space:

Focus on:

- 1) Implement it on real hardware (Raspberry Pi).**
- 2) Making the flow dynamic (as in, download script and run it on any computer).**
- 3) A cli based or automated gui for using the software**
- 4) Address Security issues, Bug fixes and more testing.**

How can we do distributed computing with two virtual machines?

To perform distributed computing with two virtual machines, you can follow these steps:

Set up the virtual machines: Create two virtual machines, each with its own operating system and network connectivity.

Choose a distributed computing framework: Choose a distributed computing framework that works with your virtual machines' operating systems. Some popular options include Apache Hadoop, Apache Spark, and MPI (Message Passing Interface).

Install the framework on each virtual machine: Install the chosen framework on each virtual machine.

Connect the virtual machines: Configure the virtual machines to communicate with each other, typically over the network.

Define the task: Define the task that you want to perform in a way that can be distributed across the virtual machines.

Run the task: Start the task, and the distributed computing framework will take care of dividing the task into smaller sub-tasks, distributing the sub-tasks to the virtual machines, and coordinating their processing.

Gather and analyze the results: Collect the results from each virtual machine and analyze them to obtain the final result of the task.

It's important to note that the specific steps to set up a distributed computing environment with virtual machines can vary depending on the chosen framework and the operating systems used.

How do we do this over the network without virtual machines?

To perform distributed computing over a network without using virtual machines, you can follow these steps:

Choose a distributed computing framework: Choose a distributed computing framework that works for your network. Some popular options include Apache Hadoop, Apache Spark, and MPI (Message Passing Interface).

Install the framework on each node: Install the chosen framework on each node that will participate in the distributed computing network.

Connect the nodes: Configure the nodes to communicate with each other over the network.

Define the task: Define the task that you want to perform in a way that can be distributed across the nodes.

Run the task: Start the task, and the distributed computing framework will take care of dividing the task into smaller sub-tasks, distributing the sub-tasks to the nodes, and coordinating their processing.

Gather and analyze the results: Collect the results from each node and analyze them to obtain the final result of the task.

It's important to note that the specific steps to set up a distributed computing environment over a network can vary depending on the chosen framework and the operating systems used on the nodes.

How to connect the nodes to communicate with each other over the network?

To connect the nodes to communicate with each other over a network, you need to configure the network communication between the nodes. Here are some common steps to do this:

Assign unique IP addresses to each node: Each node should have a unique IP address that can be used to identify it on the network.

Configure the network firewall: Make sure the firewall on each node is configured to allow incoming and outgoing network traffic as needed for the distributed computing framework to function.

Set up a network communication protocol: Choose a network communication protocol that works with the chosen distributed computing framework and configure it on each node. Some common protocols used for distributed computing are TCP/IP, Infiniband, and Myrinet.

Configure the distributed computing framework: Configure the chosen distributed computing framework to use the network communication protocol and to communicate with the other nodes. This typically involves specifying the IP addresses of the other nodes and any necessary network settings in the framework's configuration file.

Test the network connection: Test the network connection between the nodes to make sure they can communicate with each other as expected.

It's important to note that the specific steps to set up the network communication between the nodes can vary depending on the chosen distributed computing framework, the operating systems used on the nodes, and the network infrastructure.

What is apache hadoop used for?

Apache Hadoop is an open-source software framework used for storing and processing large amounts of data in a distributed computing environment. Hadoop is designed to handle big data, which refers to data sets that are too large or complex to be processed by traditional data processing tools.

Hadoop consists of two main components:

Hadoop Distributed File System (HDFS): A scalable and fault-tolerant file system that stores data in a distributed manner across a cluster of nodes.

MapReduce: A programming model and an implementation of it that allows large-scale data processing across a cluster of nodes.

Hadoop is used for a variety of big data processing tasks, including:

Data warehousing: Hadoop is used for storing and managing large amounts of structured and semi-structured data for data warehousing applications.

Data analysis: Hadoop is used for analyzing large data sets to gain insights and extract useful information.

Data mining: Hadoop is used for data mining tasks, such as finding patterns in large data sets and making predictions.

Machine learning: Hadoop is used for training machine learning models on large data sets.

Log processing: Hadoop is used for processing log files from servers and other systems to gain insights into system performance and usage.

In summary, Apache Hadoop is a powerful tool for processing big data and is widely used in industries such as finance, healthcare, and e-commerce for data analysis, data warehousing, and machine learning applications.

what is apache spark used for?

Apache Spark is an open-source, distributed computing framework used for big data processing. Spark is designed to handle big data and provides a fast, in-memory data processing engine for big data analysis. Spark was created to overcome the limitations of the Hadoop MapReduce framework for processing big data.

Spark can be used for a wide range of big data processing tasks, including:

Batch processing: Spark is used for processing large data sets in batch mode and generating results.

Real-time streaming: Spark is used for processing real-time data streams and performing actions such as filtering, transforming, and aggregating data.

Machine learning: Spark is used for building and training machine learning models on large data sets.

Graph processing: Spark is used for processing graph data and performing graph algorithms such as PageRank.

SQL processing: Spark provides a SQL interface for querying big data using SQL-like syntax.

Interactive analytics: Spark provides an interactive shell for data exploration and analysis.

In summary, Apache Spark is a fast and efficient big data processing framework that can be used for a wide range of big data processing tasks, including batch processing, real-time streaming, machine learning, graph processing, SQL processing, and interactive analytics.

what is Infiniband?

Infiniband is a type of high-performance, low-latency computer network technology used for interconnecting computer servers, storage systems, and other data center equipment. Infiniband provides a high-speed, low-latency communication path between nodes, allowing them to work together more efficiently to process large amounts of data.

Infiniband uses a switched fabric architecture and supports data transfer speeds of up to 100 Gbps. It provides support for both data transfer and remote direct memory access (RDMA), which enables high-speed communication between nodes without involving the operating system or CPU.

Infiniband is commonly used in high-performance computing (HPC) environments, such as supercomputing centers and scientific research labs, where high-speed data transfer and low-latency communication are critical. It is also used in large-scale data centers for big data processing and storage, as well as in cloud computing environments.

Infiniband provides several benefits over other types of network technologies, including high-speed data transfer, low-latency communication, and support for RDMA. These benefits make Infiniband an important technology for big data processing, high-performance computing, and cloud computing.

what is Myrinet?

Myrinet is a type of high-performance computer network technology used for interconnecting computer servers, storage systems, and other data center equipment. Myrinet was specifically designed for use in high-performance computing (HPC) environments, such as supercomputing centers and scientific research labs, where high-speed data transfer and low-latency communication are critical.

Myrinet uses a switched fabric architecture and provides data transfer speeds of up to 10 Gbps. It provides support for remote direct memory access (RDMA), which enables high-speed communication between nodes without involving the operating system or CPU.

Myrinet was one of the first high-performance computer network technologies and was widely used in HPC environments for many years. However, it has since been largely replaced by newer technologies, such as Infiniband and 10 Gigabit Ethernet, which offer higher data transfer speeds and more advanced features.

Despite its limited use today, Myrinet remains an important part of the history of high-performance computing and has had a significant impact on the development of high-speed computer networks.

<https://www.cgdirector.com/combine-processing-power-two-computers/>

how to create framework for data processing over two virtual computers

Here are the high-level steps to create a framework for data processing over two virtual computers:

Choose a programming language and tools: Decide on the programming language, tools and libraries that you want to use for data processing.

Define the architecture: Determine the architecture of the data processing framework, such as the communication protocol between virtual computers, the data storage format, etc.

Implement the framework: Based on the architecture, start implementing the data processing framework. This can include writing code for data transfer, data processing, and storing the results.

Test the framework: Test the framework thoroughly to ensure that it works as expected, especially when it comes to processing large amounts of data and handling errors.

Deploy the framework: Finally, deploy the framework on two virtual computers. Make sure that the configuration is correct and that the virtual computers have the necessary resources (e.g., memory, storage, network bandwidth) to support the data processing framework.

Note: The above steps are high-level and the actual implementation would depend on various factors such as the complexity of the data processing tasks, the size of the data, and the hardware and software resources available.

Andy Pacheco

what are the possible protocols for architecture

There are several protocols that can be used in the architecture of a data processing framework over two virtual computers. Some of the most commonly used protocols are:

Remote Procedure Call (RPC): This is a protocol that allows one virtual computer to request a service from another virtual computer, and receive a response.

Message Queuing: This protocol uses a queue-based messaging system to send and receive messages between virtual computers.

HTTP/HTTPS: This protocol is used to transfer data over the internet, and is commonly used in web-based applications.

TCP/IP: The Transmission Control Protocol/Internet Protocol is a set of communication protocols used to transmit data over a network.

FTP: The File Transfer Protocol is a standard network protocol used for transferring files between virtual computers.

SSH: Secure Shell is a secure protocol for logging into a remote machine and executing commands on it.

The choice of protocol would depend on various factors such as the nature of the data being processed, the size of the data, the security requirements, and the performance requirements of the data processing framework.

Andy Pacheco

how can we do it this over tcp/ip

To create a data processing framework over TCP/IP, you can follow these steps:

Define the communication protocol: Determine the structure and format of the data being transmitted between virtual computers, such as the message format, data encoding, and error handling.

Implement the client and server: Write the code for the client and server components of the data processing framework. The client component would be responsible for sending requests to the server, and the server component would be responsible for processing the requests and sending back the results.

Use the TCP/IP socket API: To implement the communication between the client and server, use the socket API provided by the operating system. The socket API provides a low-level interface for sending and receiving data over a network using TCP/IP.

Implement data processing: Implement the data processing logic in the server component. This can include reading and parsing the data, performing calculations, and storing the results.

Test the framework: Test the data processing framework thoroughly to ensure that it works as expected, especially when it comes to processing large amounts of data and handling errors.

Deploy the framework: Finally, deploy the framework on two virtual computers. Make sure that the network configuration is correct and that the virtual computers have the necessary resources (e.g., memory, storage, network bandwidth) to support the data processing framework.

Note: The above steps are high-level and the actual implementation would depend on various factors such as the complexity of the data processing tasks, the size of the data, and the hardware and software resources available.

Andy Pacheco

for step one, what if we make the structure format abstract?

If you make the structure format abstract in step 1, it means that you are decoupling the implementation of the data processing framework from the specific format of the data being transmitted. This allows you to change the data format in the future without affecting the implementation of the rest of the framework.

To make the structure format abstract, you can define an interface that specifies the methods and data fields required for processing the data. The implementation of the interface can then be changed as needed to accommodate different data formats.

For example, you can define an interface called "DataProcessor" with methods such as "processData" and "sendResult". The implementation of the interface would then handle the specific details of processing and transmitting the data in a specific format.

By making the structure format abstract, you can make the data processing framework more flexible and easier to maintain. It also allows for better separation of concerns, as the data processing logic can be implemented independently from the data format.

Andy Pacheco

for step two, i want to have peer to peer architecture

To create a peer-to-peer data processing framework over TCP/IP, you can follow these steps:

Define the communication protocol: Determine the structure and format of the data being transmitted between virtual computers, such as the message format, data encoding, and error handling.

Implement the peer-to-peer communication: Write the code for the peer-to-peer communication between virtual computers. Each virtual computer would act as both a client and a server, sending requests to and receiving requests from other virtual computers.

Use the TCP/IP socket API: To implement the communication between the peers, use the socket API provided by the operating system. The socket API provides a low-level interface for sending and receiving data over a network using TCP/IP.

Implement data processing: Implement the data processing logic in each virtual computer. This can include reading and parsing the data, performing calculations, and storing the results.

Test the framework: Test the data processing framework thoroughly to ensure that it works as expected, especially when it comes to processing large amounts of data and handling errors in a peer-to-peer network.

Deploy the framework: Finally, deploy the framework on multiple virtual computers. Make sure that the network configuration is correct and that the virtual computers have the necessary resources (e.g., memory, storage, network bandwidth) to support the data processing framework.

Note: The above steps are high-level and the actual implementation would depend on various factors such as the complexity of the data processing tasks, the size of the data, and the hardware and software resources available.

elaborate step three please

Step 3 of creating a peer-to-peer data processing framework over TCP/IP involves using the TCP/IP socket API provided by the operating system. The socket API provides a low-level interface for sending and receiving data over a network using TCP/IP.

Here is how you can use the socket API to implement peer-to-peer communication:

Create a socket: Use the `socket()` function to create a socket on each virtual computer. The socket is the endpoint for network communication.

Bind the socket to an address: Use the `bind()` function to bind the socket to an address (i.e., an IP address and a port number). This allows the socket to receive incoming data from other virtual computers.

Connect to a peer: Use the `connect()` function to connect the socket to a peer virtual computer. This allows the virtual computer to send data to the peer.

Send and receive data: Use the `send()` and `recv()` functions to send and receive data over the network. The data can be in any format specified in step 1, such as binary, text, or JSON.

Close the socket: Use the `close()` function to close the socket when communication is no longer needed.

Note: The socket API provides many other functions for controlling and monitoring network communication. The functions and their usage would depend on the specific requirements of the data processing framework.

Andy Pacheco

tell me more about the socket api extension

The socket API is an extension of the standard C programming library that provides a low-level interface for network communication. It is used for creating and managing network sockets, which are endpoints for network communication. The socket API is used by developers to build networked applications, such as web servers, chat applications, and data processing frameworks.

The socket API provides a wide range of functions for creating and managing sockets, including:

`socket()`: This function creates a socket. It takes a family (e.g., `AF_INET` for IPv4), a type (e.g., `SOCK_STREAM` for TCP), and a protocol (e.g., `IPPROTO_TCP`) as arguments and returns a file descriptor that can be used to identify the socket in future operations.

`bind()`: This function assigns an address (i.e., IP address and port number) to a socket. It is used by servers to listen for incoming connections.

`listen()`: This function puts the socket in a passive mode and listens for incoming connections. It is used by servers to wait for clients to initiate communication.

`accept()`: This function accepts an incoming connection request and creates a new socket for the client. It is used by servers to handle incoming connections.

`connect()`: This function establishes a connection to a server. It is used by clients to initiate communication.

`send()`: This function sends data over the network. It is used by both clients and servers to transmit data.

`recv()`: This function receives data from the network. It is used by both clients and servers to receive data.

`close()`: This function closes a socket and releases its resources. It is used by both clients and servers to close the socket when communication is no longer needed.

Note: The socket API is supported by most operating systems and is widely used for network programming. It provides a flexible and powerful interface for network communication, but also requires a good understanding of network protocols and programming concepts.

Andy Pacheco

how can i see the communication between processor and the ram

To see the communication between the processor and the RAM, you can use various tools and methods, including:

System Monitor: Most operating systems come with a system monitor tool that provides real-time information about the system's performance and resource utilization. You can use the system monitor to view the data transfer rate between the processor and the RAM.

Performance Profiling Tools: Performance profiling tools, such as Intel's VTune, allow you to monitor and analyze the performance of your system, including the communication between the processor and the RAM. These tools provide detailed information about the system's performance, including data transfer rate, memory utilization, and processor utilization.

Debugging Tools: Debugging tools, such as GDB, allow you to monitor the communication between the processor and the RAM by inspecting memory and register values. You can use these tools to step through the code and see how data is being transferred between the processor and the RAM.

Hardware Debugging Tools: Hardware debugging tools, such as logic analyzers, allow you to monitor the communication between the processor and the RAM at the hardware level. These tools provide a detailed view of the communication between the processor and the RAM, including data transfer rate, memory access patterns, and bus utilization.

Note: The specific tools and methods used to view the communication between the processor and the RAM depend on the platform, the operating system, and the specific requirements of the system.

how can I see the exact data transfer between the processor and the ram

To see the exact data transfer between the processor and the RAM, you can use various hardware debugging tools, such as:

Logic Analyzer: A logic analyzer is a device that captures and analyzes signals on a digital bus. You can use a logic analyzer to monitor the communication between the processor and the RAM and see the exact data being transferred.

Oscilloscope: An oscilloscope is a device that displays electrical signals over time. You can use an oscilloscope to monitor the signals on the bus between the processor and the RAM and see the exact data being transferred.

Bus Analyzer: A bus analyzer is a device that monitors and analyzes the communication on a digital bus. You can use a bus analyzer to monitor the communication between the processor and the RAM and see the exact data being transferred.

Note: These hardware debugging tools require a good understanding of digital circuits, digital bus protocols, and data transfer protocols. They are typically used by hardware engineers and system architects to debug and optimize system performance.

can I use these tools as a software

Yes, many of the hardware debugging tools can also be used as software tools. There are software-based versions of logic analyzers, oscilloscopes, and bus analyzers that provide similar functionality as their hardware counterparts. These software tools allow you to monitor

and analyze the communication between the processor and the RAM by capturing and decoding the signals on the bus.

Examples of software-based debugging tools include:

Saleae Logic: A software-based logic analyzer that provides a graphical interface for capturing and analyzing signals on a digital bus.

PulseView: An open-source software-based logic analyzer that provides a graphical interface for capturing and analyzing signals on a digital bus.

Virtual Oscilloscope: A software-based oscilloscope that provides a graphical interface for displaying electrical signals over time.

Note: These software-based debugging tools typically require a hardware interface, such as a USB-based logic analyzer, to capture the signals on the bus. They provide a convenient and cost-effective alternative to hardware-based debugging tools and are well suited for software development and debugging.

Swap on Commands

Here are the commands to set up a swap space on a network-based storage device in a Linux system:

1. Create a swap file on the network-based storage device. For example:

```
$ dd if=/dev/zero of=/path/to/swapfile bs=1024 count=1048576
```

This creates a 1 GB swap file on the network-based storage device at /path/to/swapfile. You can adjust the size of the swap file to meet the specific requirements of your system.

2. Format the swap file:

```
$ mkswap /path/to/swapfile
```

3. Enable the swap file:

```
$ swapon /path/to/swapfile
```

4. Verify that the swap space has been enabled:

```
$ free -h
```

You should see the new swap space listed under Swap in the output of the free command.

5. Add an entry for the swap file to the /etc/fstab file, to ensure that the swap space is automatically mounted on boot:

```
/path/to/swapfile swap swap defaults 0 0
```


How to use swapfiles over the network linux

Here are the commands to set up a swap space on a network-based storage device in a Linux system:

To use swapfiles over the network in Linux, you can follow these general steps:

Create a swapfile on the remote machine using the "fallocate" or "dd" command. For example:
`sudo fallocate -l 2G /path/to/swapfile`

Set appropriate permissions on the swapfile using the "chmod" command. For example: `sudo chmod 600 /path/to/swapfile`

Enable the swapfile using the "swapon" command. For example: `sudo swapon /path/to/swapfile`

On the local machine, create a network block device (NBD) using the "nbd-client" command. For example: `sudo nbd-client remote-machine-ip 2000 /dev/nbd0`

Map the remote swapfile to the local NBD using the "mkswap" command. For example: `sudo mkswap /dev/nbd0`

Enable the local NBD as a swap device using the "swapon" command. For example: `sudo swapon /dev/nbd0`

Note that using swapfiles over the network can have performance implications and may not be suitable for all use cases.

booting, virtual machine disk sharing, and remote swapfile usage. It allows networked storage to be treated as if it were directly attached to a local system, providing flexibility and reducing the need for physical storage devices.

While TCP can be used to transfer block device data over a network, it is not a direct alternative to NBD. NBD is a specialized protocol specifically designed for sharing block devices over a network, and it has several advantages over using generic TCP for this purpose.

Some of the advantages of NBD over TCP include:

Lower overhead: NBD is designed to minimize the amount of network overhead required to transfer block device data, which can result in better performance and lower network latency compared to using TCP.

Block-level access: NBD provides block-level access to the shared block device, allowing clients to access and modify individual blocks of data. TCP, on the other hand, is a stream-oriented protocol and does not provide direct access to individual blocks.

Security: NBD includes features for secure authentication and encryption of the block device data transferred over the network. While it is possible to implement security measures when using TCP for block device transfers, NBD provides a standardized and secure solution out-of-the-box.

That being said, it is technically possible to use TCP for block device sharing, and there may be situations where this approach is more appropriate or practical than using NBD. However, in general, if block-level access, low overhead, and secure data transfer are important requirements, NBD is likely the better choice.

Sharing a Swapfile using Network File Server:

Install NFS on both machines:

```
sudo apt-get install nfs-kernel-server nfs-common
```

Create a shared directory on the machine that will provide the swap file:

```
sudo mkdir /swap_share  
sudo chmod 777 /swap_share
```

Edit NFS exports file on the machine that will provide the swap file:

```
sudo nano /etc/exports
```

Add to the end:

```
/swap_share(rw, sync, no_subtree_check)
```

Start NFS server:

```
sudo systemctl start nfs-kernel-server
```

Check status:

```
sudo systemctl status nfs-kernel-server
```

On second machine create directory:

```
sudo mkdir /swap_mount
```

Mount:

```
sudo mount <ip-address-of-the-provider-machine>:/swap_share /swap_mount
```

Sharing folder between host machine and virtual machine:

1. Turn off virtual machine
2. Setting, Shared Folders, Add folder, Folder Path
3. Start virtual machine
4. `sudo mount -t vboxsf <folder_name> <mount point>`
5. `cd /<mount point>`

```
echo "Y" | apt-get install nfs-kernel-server nfs-common
```

```
#Create Shared Folder  
mkdir /swap_share  
chmod 777 /swap_share
```

```
#Edit Exports File  
echo "/swap_share 192.168.1.239(rw, sync, no_subtree_check) >> /etc/exports"
```

#Start Network File System Servers

systemctl start nfs-kernel-server

ifconfig | grep inet | grep -oE '\b([0-9]{1,3}\.){3}[0-9]{1,3}\b'