1. VIRTUAL MACHINES, MULTITASKING & IPC

Duration: 1 hour

MY notes: the speech is with green colour as background

Good [morning/afternoon], everyone!
*Welcome to today's seminar on **Virtual Machines, Multitasking, and Inter-Process Communication (IPC)**. Over the next four hours, we will dive into key foundational and advanced topics in system-level computing. This seminar is designed to combine theoretical knowledge with practical, hands-on experience, equipping you with skills that are directly applicable in real-world scenarios.*
Let me set the stage for our journey today:

- *We'll begin by setting up the environment, ensuring our tools and configurations are ready.*
- *Next, we'll explore the architecture of Windows Subsystem for Linux (WSL), which bridges the gap between Linux and Windows environments.*
- *From there, we'll delve into process management and multitasking concepts, implementing and monitoring background processes.*
- *We'll then investigate IPC mechanisms like named pipes, moving into dynamic interactions between processes.*
- *Finally, we'll bring it all together by discussing Docker and containerized environments, which form the backbone of modern application deployment.*

By the end, you'll have both the theoretical foundation and the practical experience needed to tackle challenges in multitasking, IPC, and virtualization.
Let's get started!

OVERVIEW

This seminar provides students with practical experience in understanding and working with virtual machines (VMs), multitasking models, API integration, dynamic linking, and inter-process communication (IPC). Through hands-on exercises and detailed explanations, students will gain both theoretical knowledge and practical skills in these fundamental computing concepts.

LEARNING OBJECTIVES

- *Understand the architecture and operation of virtual machines in WSL*
- *Master basic multitasking concepts and process management*
- *Implement inter-process communication mechanisms*
- *Gain practical experience with API integration and dynamic linking*
- *Learn container management principles using Docker*

Part 0: Environment Setup (Duration: 10 minutes)

Our first task is to set up the environment for success. This step may seem routine, but it's critical. A properly configured environment saves us from unexpected roadblocks, especially when working with resource-intensive tasks like virtualization, multitasking, and Docker containers.

In this part, we will validate our system compatibility using a script called 01check_wsl.sh.

- *The script examines key components such as the WSL version, Linux kernel, available memory, CPU specifications, and disk space.*
- *Each command gives us specific insights:*
  - ***wsl.exe --version** ensures we're using WSL 2 or higher for full virtualization.*
  - ***free -h** and **df -h** verify we have enough memory and storage.*
  - ***lscpu** checks if our CPU supports virtualization.*

💡 ***HERE'S WHY THIS MATTERS:** Running resource-intensive applications on an under-configured environment will result in slowdowns or crashes.*

Demonstration:

Let's run the script now and interpret the output together. Look for any red flags, such as low memory or outdated WSL versions. If there are issues, we'll discuss quick fixes, such as adjusting .wslconfig for resource allocation.

PART 0: ENVIRONMENT SETUP

The goal (of this part) is to establish a functional development environment using Windows Subsystem for Linux (WSL). This includes validating system compatibility and installing essential tools. Each step ensures that the system is optimized for running virtual machines, multitasking scripts, and Docker containers.

INITIAL WSL SETUP CHECK

Script: 01check_wsl.sh

Purpose: This script verifies your Windows Subsystem for Linux (WSL) environment. Each command provides critical information:

- **wsl.exe --version:** Confirms the WSL version to ensure compatibility with VM and container setups.
- **uname -a:** Shows the Linux kernel version and architecture, validating the system configuration.
- **lsb_release -a:** Provides Ubuntu details (version, codename) to match system requirements.
- **free -h:** Checks available memory for running VMs and multitasking.
- **lscpu:** Lists CPU specifications, including support for virtualization.
- **df -h:** Displays disk space usage to confirm sufficient storage for tasks.

Understanding the Commands

1. wsl.exe --version
   - **Purpose**: Verifies WSL version
   - **Why Important**: Different WSL versions have different capabilities and limitations
   - **What to Look For**: Version should be WSL 2 or higher for full virtualization support
2. uname -a
   - **Purpose**: Displays detailed Linux kernel information
   - **Why Important**: Confirms kernel compatibility with virtual machine features
   - **How to Interpret**: Shows architecture (x86_64), kernel version, and build details
3. lsb_release -a
   - **Purpose**: Shows Ubuntu distribution details
   - **Why Important**: Ensures compatibility with provided scripts and tools
   - **Key Information**: Version number and codename help in package management
4. free -h
   - **Purpose**: Displays memory usage in human-readable format
   - **Why Important**: Virtual machines and containers need adequate memory

- **What to Check**: Available memory should be at least 4GB for smooth operation

5. lscpu
   - **Purpose**: Lists CPU specifications
   - **Why Important**: Virtual machines require specific CPU features
   - **Critical Features**: Look for "vmx" or "svm" flags indicating virtualization support
6. df -h
   - **Purpose**: Shows disk space usage
   - **Why Important**: VMs and containers need sufficient storage
   - **Minimum Requirements**: At least 10GB free space recommended

Expected Output: Students will observe detailed system information. Any discrepancies (e.g., insufficient resources) must be addressed before proceeding.

```
#!/bin/bash
# Save as 01check_wsl.sh
```

(possible) results of running 01check_wsl.sh

```
=== WSL Environment Check ===
WSL Version:
WSL version: 2.3.26.0
Kernel version: 5.15.167.4-1
WSLg version: 1.0.65
MSRDC version: 1.2.5620
Direct3D version: 1.611.1-81528511
DXCore version: 10.0.26100.1-240331-1435.ge-release
Windows version: 10.0.26100.2605
```

What This Tells Us:

- WSL version 2.3.26.0 indicates a modern installation capable of full virtualization
- Kernel 5.15 supports all required virtualization features
- WSLg 1.0.65 enables GUI application support

```
=== Linux Version ===
Linux Lenovo16 5.15.167.4-microsoft-standard-WSL2 #1 SMP Tue Nov 5 00:21:55 UTC 2024 x86_64 x86_64 x86_64 GNU/Linux
```

Key Information:

- Running WSL2 kernel (not WSL1)
- Microsoft-customized kernel optimized for Windows integration
- x86_64 architecture supporting full 64-bit operations

```
=== Ubuntu Version ===
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 24.04.1 LTS
Release:        24.04
Codename:       noble
```

Significance:

- LTS (Long Term Support) version ensures stability
- 24.04 provides latest package compatibility
- .1 indicates it includes latest security updates

```
=== Memory Available ===
               total        used        free      shared  buff/cache   available
Mem:            31Gi       809Mi        30Gi       3.0Mi       384Mi        30Gi
Swap:          8.0Gi          0B       8.0Gi
```

Analysis:

- 31GB total RAM is excellent for VM/container operations
- Only 809MB used shows plenty of headroom
- 384MB buff/cache indicates efficient memory management
- 30GB available ensures smooth operation of multiple containers

```
=== CPU Information ===
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         39 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  20
```

```
  On-line CPU(s) list:    0-19
Vendor ID:                GenuineIntel
  Model name:             13th Gen Intel(R) Core(TM) i7-13700H
    CPU family:           6
    Model:                186
    Thread(s) per core:   2
    Core(s) per socket:   10
    Socket(s):            1
    Stepping:             2
    BogoMIPS:             5836.80
    Flags:                fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse s
                          se2 ss ht syscall nx pdpe1gb rdtscp lm constant_tsc rep_good nopl xtopology tsc_reliable nonst
                          op_tsc cpuid pni pclmulqdq vmx ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadl
……..
                          1d arch_capabilities
Virtualization features:
  Virtualization:         VT-x
  Hypervisor vendor:      Microsoft
  Virtualization type:    full
Caches (sum of all):
  L1d:                    480 KiB (10 instances)
  L1i:                    320 KiB (10 instances)
  L2:                     12.5 MiB (10 instances)
  L3:                     24 MiB (1 instance)
Vulnerabilities:
  Gather data sampling:   Not affected
  Itlb multihit:          Not affected
  L1tf:                   Not affected
  Mds:                    Not affected
  Meltdown:               Not affected
  Mmio stale data:        Not affected
  Reg file data sampling: Mitigation; Clear Register File
  Retbleed:               Mitigation; Enhanced IBRS
  Spec rstack overflow:   Not affected
  Spec store bypass:      Mitigation; Speculative Store Bypass disabled via prctl and seccomp
  Spectre v1:             Mitigation; usercopy/swapgs barriers and __user pointer sanitization
  Spectre v2:             Mitigation; Enhanced / Automatic IBRS; IBPB conditional; RSB filling; PBRSB-eIBRS SW sequence;
                          BHI BHI_DIS_S
  Srbds:                  Not affected
  Tsx async abort:        Not affected

=== Disk Space ===
Filesystem      Size  Used Avail Use% Mounted on
none            16G     0   16G   0% /usr/lib/modules/5.15.167.4-microsoft-standard-WSL2
none            16G  4.0K   16G   1% /mnt/wsl
drivers        1.9T  507G  1.4T  28% /usr/lib/wsl/drivers
/dev/sdc       1007G  1.5G  955G   1% /
none            16G   76K   16G   1% /mnt/wslg
none            16G     0   16G   0% /usr/lib/wsl/lib
rootfs          16G  2.2M   16G   1% /init
none            16G  536K   16G   1% /run
none            16G     0   16G   0% /run/lock
none            16G     0   16G   0% /run/shm
tmpfs          4.0M     0  4.0M   0% /sys/fs/cgroup
none            16G   76K   16G   1% /mnt/wslg/versions.txt
none            16G   76K   16G   1% /mnt/wslg/doc
C:\            1.9T  507G  1.4T  28% /mnt/c
D:\            918G  440G  479G  48% /mnt/d
F:\            477G  223G  255G  47% /mnt/f
Z:\             14G   43M   14G   1% /mnt/z
tmpfs          3.2G   16K  3.2G   1% /run/user/1000
```

REQUIRED PACKAGE INSTALLATION

Script: 02setup_environment.sh

Purpose: This script ensures the environment is equipped with tools necessary for development and multitasking. Key components include:

- build-essential: Provides compilers and libraries.
- Docker: Enables container management, vital for multitasking and API demonstrations.
- htop, tree, jq: Facilitate resource monitoring and JSON parsing.

Understanding the Installation Process

1. **Essential Tools Installation**
    - **build-essential**: Provides compilation tools (gcc, g++, make)

- **htop**: Interactive process viewer
- **net-tools**: Network configuration utilities
- **procps**: Process monitoring tools
- **Why Important**: These tools form the foundation for development and system monitoring

2. **Development Tools Installation**
   - **git**: Version control system
   - **make**: Build automation tool
   - **Why Important**: Essential for managing code and building projects
3. **Container Tools Installation**
   - **Docker**: Container platform
   - **Why Important**: Enables containerization and isolated environments
   - **Post-Installation**: User added to docker group for permissions

Troubleshooting:

- If Docker permissions issues arise, run sudo usermod -aG docker $USER and restart the session.

```
#!/bin/bash
# Save as 02setup_environment.sh
```

(possible) results of running 02setup_environment.sh

```
=== Updating Package Lists ===
[sudo] password for clim:
Get:1 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Hit:2 http://archive.ubuntu.com/ubuntu noble InRelease
Get:3 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:4 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [585 kB]
….
….
Get:29 http://archive.ubuntu.com/ubuntu noble-backports/multiverse amd64 Components [212 B]
Fetched 6128 kB in 2s (2556 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
4 packages can be upgraded. Run 'apt list --upgradable' to see them.
=== Installing Essential Tools ===
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
net-tools is already the newest version (2.10-0.1ubuntu4).
procps is already the newest version (2:4.0.4-4ubuntu3.2).
procps set to manually installed.
iproute2 is already the newest version (6.1.0-1ubuntu6).
iproute2 set to manually installed.
tmux is already the newest version (3.4-1ubuntu0.1).
tmux set to manually installed.
tree is already the newest version (2.1.1-2ubuntu3).
curl is already the newest version (8.5.0-2ubuntu10.6).
curl set to manually installed.
wget is already the newest version (1.21.4-1ubuntu4.1).
wget set to manually installed.
The following additional packages will be installed:
  bzip2 cpp cpp-13 cpp-13-x86-64-linux-gnu cpp-x86-64-linux-gnu dpkg-dev fakeroot g++ g++-13 g++-13-x86-64-linux-gnu
  g++-x86-64-linux-gnu gcc gcc-13 gcc-13-base gcc-13-x86-64-linux-gnu gcc-x86-64-linux-gnu libalgorithm-diff-perl
  libalgorithm-diff-xs-perl libalgorithm-merge-perl libaom3 libasan8 libatomic1 libc-dev-bin libc-devtools libc6-dev
  libcc1-0 libcrypt-dev libde265-0 libdpkg-perl libfakeroot libfile-fcntllock-perl libgcc-13-dev libgd3 libgomp1
  libheif-plugin-aomdec libheif-plugin-aomenc libheif-plugin-libde265 libheif1 libhwasan0 libisl23 libitm1 libjq1
  liblsan0 libmpc3 libnl-3-200 libnl-genl-3-200 libonig5 libquadmath0 libstdc++-13-dev libtsan2 libubsan1 libxpm4
  linux-libc-dev lto-disabled-list make manpages-dev rpcsvc-proto
Suggested packages:
  bzip2-doc cpp-doc gcc-13-locales cpp-13-doc debian-keyring g++-multilib g++-13-multilib gcc-13-doc gcc-multilib
  autoconf automake libtool flex bison gdb gcc-doc gcc-13-multilib gdb-x86-64-linux-gnu lm-sensors strace glibc-doc
  bzr libgd-tools libheif-plugin-x265 libheif-plugin-ffmpegdec libheif-plugin-jpegdec libheif-plugin-jpegenc
  libheif-plugin-j2kdec libheif-plugin-j2kenc libheif-plugin-rav1e libheif-plugin-svtenc libstdc++-13-doc make-doc
The following NEW packages will be installed:
  build-essential bzip2 cpp cpp-13 cpp-13-x86-64-linux-gnu cpp-x86-64-linux-gnu dpkg-dev fakeroot g++ g++-13
  g++-13-x86-64-linux-gnu g++-x86-64-linux-gnu gcc gcc-13 gcc-13-base gcc-13-x86-64-linux-gnu gcc-x86-64-linux-gnu
  htop jq libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl libaom3 libasan8 libatomic1
  libc-dev-bin libc-devtools libc6-dev libcc1-0 libcrypt-dev libde265-0 libdpkg-perl libfakeroot
```

```
  libfile-fcntllock-perl libgcc-13-dev libgd3 libgomp1 libheif-plugin-aomdec libheif-plugin-aomenc
  libheif-plugin-libde265 libheif1 libhwasan0 libisl23 libitm1 libjq1 liblsan0 libmpc3 libnl-3-200 libnl-genl-3-200
  libonig5 libquadmath0 libstdc++-13-dev libtsan2 libubsan1 libxpm4 linux-libc-dev lto-disabled-list make manpages-dev
  rpcsvc-proto
0 upgraded, 60 newly installed, 0 to remove and 4 not upgraded.
Need to get 72.0 MB of archives.
After this operation, 244 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu noble/main amd64 libnl-3-200 amd64 3.7.0-0.3build1 [55.6 kB]
Get:2 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 libc-dev-bin amd64 2.39-0ubuntu8.3 [60.8 kB]
Get:3 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 linux-libc-dev amd64 6.8.0-51.52 [1769 kB]
….
….
Get:59 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 libheif-plugin-aomenc amd64 1.17.6-1ubuntu4.1 [14.7 kB]
Get:60 http://archive.ubuntu.com/ubuntu noble/main amd64 manpages-dev all 6.7-2 [2013 kB]
Fetched 72.0 MB in 7s (11.1 MB/s)
Extracting templates from packages: 100%
Selecting previously unselected package libnl-3-200:amd64.
(Reading database ... 40919 files and directories currently installed.)
Preparing to unpack .../00-libnl-3-200_3.7.0-0.3build1_amd64.deb ...
Unpacking libnl-3-200:amd64 (3.7.0-0.3build1) ...
Selecting previously unselected package libc-dev-bin.
Preparing to unpack .../01-libc-dev-bin_2.39-0ubuntu8.3_amd64.deb ...
…
…
Preparing to unpack .../59-manpages-dev_6.7-2_all.deb ...
Unpacking manpages-dev (6.7-2) ...
Setting up libaom3:amd64 (3.8.2-2ubuntu0.1) ...
Setting up manpages-dev (6.7-2) ...
…
…
Setting up libheif-plugin-aomenc:amd64 (1.17.6-1ubuntu4.1) ...
Processing triggers for libc-bin (2.39-0ubuntu8.3) ...
Processing triggers for man-db (2.12.0-4build2) ...
Processing triggers for hicolor-icon-theme (0.17-2) ...
=== Installing Development Tools ===
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
git is already the newest version (1:2.43.0-1ubuntu7.1).
git set to manually installed.
make is already the newest version (4.3-4.1build2).
make set to manually installed.
bc is already the newest version (1.07.1-3ubuntu4).
bc set to manually installed.
gawk is already the newest version (1:5.2.1-2build3).
gawk set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 4 not upgraded.
=== Installing Container Tools ===
Installing Docker...
# Executing docker install script, commit: 4c94a56999e10efcf48c5b8e3f6afea464f9108e

WSL DETECTED: We recommend using Docker Desktop for Windows.
Please get Docker Desktop from https://www.docker.com/products/docker-desktop/


You may press Ctrl+C now to abort this script.
+ sleep 20
+ sh -c apt-get -qq update >/dev/null
+ sh -c DEBIAN_FRONTEND=noninteractive apt-get -y -qq install ca-certificates curl >/dev/null
+ sh -c install -m 0755 -d /etc/apt/keyrings
+ sh -c curl -fsSL "https://download.docker.com/linux/ubuntu/gpg" -o /etc/apt/keyrings/docker.asc
+ sh -c chmod a+r /etc/apt/keyrings/docker.asc
+ sh -c echo "deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu noble stable" >
/etc/apt/sources.list.d/docker.list
+ sh -c apt-get -qq update >/dev/null
+ sh -c DEBIAN_FRONTEND=noninteractive apt-get -y -qq install docker-ce docker-ce-cli containerd.io docker-compose-plugin docker-
ce-rootless-extras docker-buildx-plugin >/dev/null
+ sh -c docker version
Client: Docker Engine - Community
 Version:           27.4.1
 API version:       1.47
 Go version:        go1.22.10
 Git commit:        b9d17ea
 Built:             Tue Dec 17 15:45:46 2024
 OS/Arch:           linux/amd64
 Context:           default
```

```
Server: Docker Engine - Community
 Engine:
  Version:          27.4.1
  API version:      1.47 (minimum version 1.24)
  Go version:       go1.22.10
  Git commit:       c710b88
  Built:            Tue Dec 17 15:45:46 2024
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.7.24
  GitCommit:        88bf19b2105c8b17560993bee28a01ddc2f97182
 runc:
  Version:          1.2.2
  GitCommit:        v1.2.2-0-g7cb3632
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0

================================================================================

To run Docker as a non-privileged user, consider setting up the
Docker daemon in rootless mode for your user:

    dockerd-rootless-setuptool.sh install

Visit https://docs.docker.com/go/rootless/ to learn about rootless mode.


To run the Docker daemon as a fully privileged service, but granting non-root
users access, refer to https://docs.docker.com/go/daemon-access/

WARNING: Access to the remote API on a privileged Docker daemon is equivalent
         to root access on the host. Refer to the 'Docker daemon attack surface'
         documentation for details: https://docs.docker.com/go/attack-surface/

================================================================================

=== Verifying Installations ===
docker: Installed ✓
git: Installed ✓
make: Installed ✓
htop: Installed ✓
curl: Installed ✓
jq: Installed ✓
```

Key Learning Outcomes

1. **Why Setup Validation Matters:**
   Ensures compatibility and avoids runtime issues.
2. **Real-World Connection:**
   These tools and steps mimic the process professionals follow to configure cloud environments like AWS or Azure VMs.

Part 1: Understanding WSL Architecture (Duration: 10 minutes)

Now that our environment is ready, let's explore WSL.
Windows Subsystem for Linux is an incredible integration of Linux tools into the Windows ecosystem. It's more than just a terminal – it's a lightweight virtual machine running a real Linux kernel.

- *WSL offers **network interoperability**, **file system integration**, and **dynamic process management**.*

- *You can seamlessly switch between Linux and Windows environments, which is invaluable for developers and system administrators.*

To illustrate this, we'll run the S1.wsl_info.sh script.

- *It demonstrates network interfaces (ip addr show), mounted drives (mount), and resource usage (/proc/meminfo).*
- *One of the most exciting features of WSL is the ability to execute Windows programs like notepad.exe directly from Linux.*

Hands-On Exercise:

- *Try running the script, navigate to /mnt/c/ to view Windows files, and run a Windows program from the Linux terminal.*
- *Reflect: How does WSL blur the line between two traditionally distinct operating systems?*

Key Takeaway: *WSL is a prime example of dynamic linking and interoperability in action.*

PART 1: UNDERSTANDING WSL ARCHITECTURE

(The goal for this section is to)Understand how WSL bridges Linux and Windows systems to enable multitasking, dynamic linking, and inter-process communication (IPC).
Windows Subsystem for Linux (WSL) integrates Linux tools and applications into Windows. It serves as a lightweight virtual machine that:

- *Provides a Linux kernel on top of the Windows platform.*
- *Offers seamless interoperability for file systems, network configurations, and process management.*

WSL ARCHITECTURE OVERVIEW (WSL Integration Points)

Script: S1.wsl_info.sh
Purpose: This script explores WSL integration points, including network interfaces, file system mounts, and system resource access.

- ip addr show: Displays active network interfaces, including WSL.
- $PATH: Lists paths, demonstrating Windows-Linux interoperability.
- mount: Verifies mounted Windows drives, ensuring seamless file access.
- /proc/meminfo, /proc/cpuinfo: Provide resource details.

Understanding WSL Integration

1. **Network Integration**
   - Shows how WSL implements virtual network interfaces
   - Demonstrates communication between Windows and Linux
   - Explains NAT networking in virtual environments
2. **File System Integration**
   - Maps Windows drives to Linux mount points
   - Shows how dynamic linking works across systems
   - Explains virtual file system implementation

```
#!/bin/bash
# Save as S1.wsl_info.sh

echo "=== WSL Integration Points ==="
echo "1. Network Information:"
ip addr show
```

##lo: Loopback interface for internal communications.

```
##eth0: The main virtual interface used for external communication in WSL.
echo -e "\n2. Windows Path Integration:"
echo $PATH | tr ':' '\n' | grep -i 'windows'
##Purpose: Lists Windows paths included in the Linux $PATH.
##Significance: Demonstrates dynamic linking between Windows and Linux executables.
###/mnt/c/Windows/System32
###/mnt/c/Program Files
##Key Concept: Windows tools like notepad.exe or powershell.exe can be executed directly from WSL.
echo -e "\n3. Mount Points:"
mount | grep -i 'windows'
##Displays how Windows drives (e.g., C:, D:) are mounted into WSL.
##Key Concept: WSL uses /mnt/<drive_letter> to make Windows drives accessible from Linux.
echo -e "\n4. Process Integration:"
ps aux | grep -i '[w]slhost'
##Purpose: Identifies the WSL host process responsible for Linux-Windows integration.
##Relevance: Illustrates how WSL manages processes within the Windows ecosystem.
echo -e "\n5. System Resources:"
cat /proc/meminfo | head -n 5
##Displays memory details. Demonstrates how WSL provides Linux processes with direct access to host system resources.
cat /proc/cpuinfo | grep -E "processor|model name" | head -n 4
##Lists CPU specifications. Demonstrates how WSL provides Linux processes with direct access to host system resources.
```

(possible) results of running 02setup_environment.sh

```
=== WSL Integration Points ===
1. Network Information:
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet 10.255.255.254/32 brd 10.255.255.254 scope global lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:15:5d:52:6d:c7 brd ff:ff:ff:ff:ff:ff
    inet 172.27.158.203/20 brd 172.27.159.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::215:5dff:fe52:6dc7/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:18:6e:ac:7a brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever

2. Windows Path Integration:
/mnt/c/Windows/system32
/mnt/c/Windows
/mnt/c/Windows/System32/Wbem
/mnt/c/Windows/System32/WindowsPowerShell/v1.0/
/mnt/c/Windows/System32/OpenSSH/
/mnt/c/WINDOWS/System32/OpenSSH/
/mnt/c/WINDOWS/system32
/mnt/c/WINDOWS
/mnt/c/WINDOWS/System32/Wbem
/mnt/c/WINDOWS/System32/WindowsPowerShell/v1.0/
/mnt/c/Users/ASE/AppData/Local/Microsoft/WindowsApps

3. Mount Points:

4. Process Integration:

5. System Resources:
MemTotal:       32622948 kB
MemFree:        31812848 kB
MemAvailable:   31713912 kB
Buffers:            2180 kB
Cached:           232136 kB
processor       : 0
```

```
model name      : 13th Gen Intel(R) Core(TM) i7-13700H
processor       : 1
model name      : 13th Gen Intel(R) Core(TM) i7-13700H
```

Hands-On Activity for Students

1. Run the Script S1.wsl_info.sh
2. Discuss the output with the class, focusing on:
   - Network integration (ip addr show).
   - File system integration (mount).
3. Experiment:
   - Try running notepad.exe or explorer.exe directly from the WSL terminal.
   - Navigate to /mnt/c/ and view files on the Windows C:\ drive.
4. Analysis Question:
   - How does WSL ensure seamless communication between Windows and Linux?

KEY LEARNING OUTCOMES

1. Windows-Linux Interoperability:
   Students will appreciate how WSL combines the strengths of both platforms.
2. Dynamic Linking in Action:
   Direct execution of Windows tools from Linux highlights the power of WSL.
3. Resource Awareness:
   Understanding system resources (CPU, memory) is vital for multitasking and IPC.

Part 2: Process Management and Multitasking (Duration: 10 minutes)

Moving forward, we'll explore multitasking and process management.
Multitasking allows multiple processes to run simultaneously, a cornerstone of modern operating systems. In this part, you will:

1. ***Create background tasks.***
2. ***Monitor their behavior.***
3. ***Control them using commands like pkill or kill.***

Let's begin by running S2.process_demo.sh.

- *This script creates three background tasks that log their activity to files.*
- *You'll monitor these tasks using ps and jobs.*
- *After observing, you'll terminate them and clean up resources.*

💡 *Why this is important: Mismanaged background processes can lead to system slowdowns, memory leaks, and inefficiencies.*

Interactive Activity:

- *Modify the script to start five tasks instead of three.*
- *Use tail -f to monitor logs in real-time and analyze the differences between tasks.*

Question for reflection: What happens if we don't clean up processes properly? How would this affect system performance?

PART 2: PROCESS MANAGEMENT AND MULTITASKING

Goal:

Learn the fundamentals of process management and multitasking by creating and monitoring background processes, analyzing system resource usage, and using signals for process control.

BACKGROUND PROCESS DEMONSTRATION

a. *Multitasking is the ability of an OS to run multiple processes simultaneously.*
b. *Processes can be run in the **foreground** (interactive) or **background** (non-interactive).*
c. *Important tools:*
   - *ps: Shows running processes.*
   - *jobs: Displays background processes for the current session.*
   - *pkill or kill: Stops specific processes.*

Script: S2.process_demo.sh
Purpose: This script demonstrates multitasking by:

- Creating background tasks (& runs processes in the background).
- Using jobs and ps to monitor active processes.
- Stopping tasks with pkill to show resource cleanup.

```
#!/bin/bash
# Save as S2.process_demo.sh

# Create a background task
create_background_task() {
……..
}
echo "=== Starting Background Tasks ==="
# Start 3 background tasks
for i in {1..3}; do
   create_background_task $i &
   echo "Started task $i with PID $!"
done
##The function create_background_task writes logs to a file every 5 seconds.
##The & symbol after a command runs it in the background.
##PID $!": Displays the process ID (PID) of the background task.
##Students see how tasks can run in the background without blocking the terminal.
echo -e "\n=== Process Status ==="
ps aux | grep "[c]reate_background_task"
##Lists all processes containing "create_background_task" in their name.
##The [c] syntax prevents grep itself from appearing in the results.
echo -e "\n=== Using jobs command ==="
jobs
……..
read
##Lists background jobs started in the current session.
# Cleanup
pkill -f "create_background_task"
##The pkill command stops all processes matching the pattern "create_background_task".
rm -f task_*.log
##Logs (task_*.log) are deleted with rm -f.
```

(possible) results of running 02setup_environment.sh

```
=== Starting Background Tasks ===
Started task 1 with PID 2835
Started task 2 with PID 2836
Started task 3 with PID 2838
```

```
=== Process Status ===

=== Using jobs command ===
[1]   Running                 create_background_task $i &
[2]-  Running                 create_background_task $i &
[3]+  Running                 create_background_task $i &
Tasks are logging to task_*.log files
Press Enter to stop all tasks...
```

After Enter the system returns the control to user:

```
clim@Lenovo16:~/THEORY/1VirtualM,Multitask$
```

Key learning point:

Resource cleanup is essential in multitasking to avoid memory leaks or system slowdowns.

Running the script

1. *Save the script as S2.process_demo.sh and make it executable:*
   chmod +x S2.process_demo.sh
2. *Execute the script:*
   ./S2.process_demo.sh
3. *Observe outputs:*
   - *Monitor process logs with tail -f task_1.log.*
   - *Check running tasks with ps aux.*
4. *Press Enter to stop all tasks and clean up.*

Activity for students

1. ***Experiment with Background Jobs:***
   - *Modify the script to start 5 tasks instead of 3.*
   - *Discuss how this impacts system resource usage.*
2. ***Analyze Logs:***
   - *Use cat task_1.log to view task outputs.*
   - *How does the output differ between tasks?*
3. ***Troubleshoot Stuck Processes:***
   - *What happens if a task isn't cleaned up?*
   - *Introduce the kill command:*
     kill -9 <PID>

PROCESS MONITORING TOOL

Script: S3.monitor.sh

Purpose: Provides real-time monitoring of system resources and processes, essential for managing multitasking efficiently.

- ps aux: Lists processes sorted by CPU usage.
- netstat: Displays active network connections, useful for IPC diagnostics.

```
#!/bin/bash
# Save as S3.monitor.sh

monitor_processes() {
……..
    ps aux --sort=-%cpu | head -n 6
        ##Lists top 5 CPU-consuming processes.
    echo -e "\n2. Memory Usage:"
    free -h
        ##Displays used and available memory in a human-readable format.
    echo -e "\n3. Disk Usage:"
```

```
    df -h | grep -v "loop"
        ##Shows available disk space, excluding loopback devices.
    echo -e "\n4. Network Connections:"
    netstat -tun | head -n 5
        ##Lists active network connections.
    echo -e "\nPress Ctrl+C to exit..."
    sleep 2
  done
}

monitor_processes
```

(possible) results of running S3.monitor.sh

```
=== System Monitor ===
Time: Sat Jan 11 13:58:46 EET 2025

1. Top CPU Processes:
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root        2373  0.1  0.1 2170004 52096 ?       Ssl  13:43   0:01 /usr/bin/containerd
clim        3141  0.1  0.0   4752  3436 pts/0    S+   13:56   0:00 /bin/bash ./S3.monitor.sh
root           1  0.0  0.0  22088 13420 ?        Ss   13:33   0:00 /sbin/init
root        2560  0.0  0.2 2285364 85156 ?       Ssl  13:43   0:00 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
clim        2836  0.0  0.0   4752  2016 pts/0    S    13:52   0:00 /bin/bash ./S2.process_demo.sh

2. Memory Usage:
               total        used        free      shared  buff/cache   available
Mem:            31Gi       898Mi        30Gi       3.0Mi       287Mi        30Gi
Swap:          8.0Gi          0B       8.0Gi

3. Disk Usage:
Filesystem      Size  Used Avail Use% Mounted on
none            16G     0   16G   0% /usr/lib/modules/5.15.167.4-microsoft-standard-WSL2
none            16G  4.0K   16G   1% /mnt/wsl
drivers        1.9T  506G  1.4T  28% /usr/lib/wsl/drivers
/dev/sdc       1007G  2.3G  954G   1% /
none            16G   80K   16G   1% /mnt/wslg
none            16G     0   16G   0% /usr/lib/wsl/lib
rootfs          16G  2.2M   16G   1% /init
none            16G  556K   16G   1% /run
none            16G     0   16G   0% /run/lock
none            16G     0   16G   0% /run/shm
tmpfs          4.0M     0  4.0M   0% /sys/fs/cgroup
none            16G   76K   16G   1% /mnt/wslg/versions.txt
none            16G   76K   16G   1% /mnt/wslg/doc
C:\            1.9T  506G  1.4T  28% /mnt/c
D:\            918G  440G  479G  48% /mnt/d
F:\            477G  223G  255G  47% /mnt/f
Z:\             14G   43M   14G   1% /mnt/z
tmpfs          3.2G   16K  3.2G   1% /run/user/1000

4. Network Connections:
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State

Press Ctrl+C to exit...
```

After Ctrl+C the system returns the control to user:

```
^C
clim@Lenovo16:~/THEORY/1VirtualM,Multitask$
```

Activity for Students

1. ***Run the Script:***

   ./S3.monitor.sh

2. ***Experiment:***

   - *Observe changes in CPU usage while running multiple background tasks.*
   - *Discuss how multitasking affects memory and disk usage.*

3. ***Analysis Question:***

- *What metrics are most critical when managing multitasking in real-world scenarios?*

Key Learning Outcomes

1. ***Process Awareness:***
   Students learn how to monitor, start, and stop processes efficiently.
2. ***Multitasking in Action:***
   Hands-on experience demonstrates the impact of multitasking on system resources.
3. ***Practical Debugging:***
   Using tools like ps, free, and netstat equips students to troubleshoot resource bottlenecks.

Part 3: Inter-Process Communication (IPC) (Duration: 10 minutes)

We now move to IPC, which enables processes to share data.
Imagine two processes working in parallel – one needs to send data to the other. IPC mechanisms like ***named pipes*** *enable such communication.*

- *A* ***named pipe*** *is a file that persists in the file system, allowing unrelated processes to exchange messages.*

Demonstration 1: *Unidirectional Communication*
We'll use the S4.pipe_demo_receiver.sh script to set up a receiver. In a second terminal, send messages using echo commands. Observe how messages are logged by the receiver.
Demonstration 2: *Bidirectional Communication*
We'll then run S5.pipe_demo2.sh, which allows two-way communication. You'll see how processes interact dynamically, sending and receiving messages.
💡 *Reflection: How does bidirectional communication differ from unidirectional communication? Where might each be useful?*
Key Learning Outcome: *Students will appreciate the role of IPC in building distributed systems and managing process interactions.*

PART 3: INTER-PROCESS COMMUNICATION (IPC)

NAMED PIPES EXAMPLE (only with RECEIVER)

Script: S4.pipe_demo_receiver.sh
Purpose: Demonstrates unidirectional communication using named pipes. The sender writes messages to the pipe, which can be read in another terminal.

```
#!/bin/bash
# Save as S4.pipe_demo_receiver.sh

PIPE_NAME="/tmp/demo_pipe"

# Cleanup any existing pipe
rm -f "$PIPE_NAME"

# Create named pipe
mkfifo "$PIPE_NAME"
```

##Creates a special file /tmp/demo_pipe that processes can use to exchange messages.

```
##Why Important: Named pipes persist in the file system and allow communication between unrelated processes.
Echo “=== Named Pipe Demo ===”
echo “Open a new terminal and run: cat $PIPE_NAME”
echo “This terminal will send messages to the pipe.”
Echo “Press Ctrl+C when done.”

# Send messages
count=1
while true; do
    echo “Message $count from sender ($(date))” > “$PIPE_NAME”
    echo “Sent message $count”
    count=$((count + 1))
    sleep 2
done
##Writes a message to the named pipe every 2 seconds.
##Messages are stored in the pipe until read by another process.
# Note: The cleanup below will only execute if you Ctrl+C
# Cleanup
rm -f "$PIPE_NAME"tmux kill-window -t "Receiver"
```

Running the script:

```
clim@Lenovo16:~/THEORY/1VirtualM,Multitask$ ./S4.pipe_demo.sh
=== Named Pipe Demo ===
Open a new terminal and run: cat /tmp/demo_pipe
This terminal will send messages to the pipe.
Press Ctrl+C when done.
```

In a second terminal, read the pipe:

```
clim@Lenovo16:~$ cat /tmp/demo_pipe
Message 1 from sender (Sat Jan 11 14:12:13 EET 2025)
##The cat command reads messages from the pipe as they arrive.
```

We will see in the main terminal a new line:

```
Sent message 1
```

NAMED PIPES EXAMPLE (both with SENDER+RECEIVER)

Script: S5.pipe_demo2.sh

This script demonstrates bidirectional communication using a named pipe. Both the sender and receiver processes interact through the same pipe.

```
#!/bin/bash
# Save as S5.pipe_demo2.sh

PIPE_NAME="/tmp/demo_pipe"
TIMEOUT=10  # Timeout in seconds for receiver

# Cleanup function to remove pipe and terminate processes
cleanup() {
……..
}

# Set cleanup on exit or interrupt
trap cleanup EXIT SIGINT SIGTERM

# Remove any existing pipe
rm -f "$PIPE_NAME"

# Create a new named pipe
if ! mkfifo "$PIPE_NAME"; then
    echo "Error: Could not create named pipe at $PIPE_NAME"
    exit 1
```

```
fi

echo "=== Named Pipe Demo ==="
echo "Receiver will listen for messages for $TIMEOUT seconds."
echo "To send a message, open another terminal and use: echo 'Your message' > $PIPE_NAME"
echo "To stop the communication, send 'quit'."

# START THE RECEIVER IN THE BACKGROUND
{
   while true; do
……..
       else
……
       fi
   done
} &
```

##Command: read -t $TIMEOUT message < "$PIPE_NAME"

```
#           #Reads a message from the pipe with a timeout of 10 seconds.
#           #If no message is received, a timeout message is displayed.
# Get the background receiver process's PID
……..

# Allow the receiver a moment to start
……

# Sender: Simulate sending 5 messages
……..
```

Running the script:

```
clim@Lenovo16:~/THEORY/1VirtualM,Multitask$ ./S5.pipe_demo_both.sh
=== Named Pipe Demo ===
Will send 5 messages with timeouts
Sending message 1...
“Open a new terminal and run: cat /tmp/demo_pipe”
Sent message 1
Receiver got: Message 1 at Sat Jan 11 14:32:07 EET 2025
Receiver timeout - no message received
Sending message 2...
“Open a new terminal and run: cat /tmp/demo_pipe”
Sent message 2
Sending message 3...
“Open a new terminal and run: cat /tmp/demo_pipe”
Sent message 3
Sending message 4...
“Open a new terminal and run: cat /tmp/demo_pipe”
Sent message 4
Sending message 5...
“Open a new terminal and run: cat /tmp/demo_pipe”
Sent message 5
Sending quit message...
Demo completed successfully.

Cleaning up...
```

In a second terminal, send messages:

```
clim@Lenovo16:~$ echo "Hello from Terminal 2" > /tmp/demo_pipe
```

Stop the communication by sending "quit":

```
clim@Lenovo16:~$ echo "quit" > /tmp/demo_pipe
```

Expected Output in Main Terminal:

SIGNAL HANDLING

Script: S6.signal_demo.sh
Purpose: This script demonstrates how processes respond to signals such as SIGINT, SIGTERM, and custom signals like SIGUSR1, demonstrating controlled process termination.

```
#!/bin/bash
# Save as S6.signal_demo.sh

# Cleanup function
…….
# Signal handler function (triggered later in traps)
……..
# Register signal handlers
trap 'handle_signal SIGINT' SIGINT
##Triggers the function handle_signal when a SIGINT (Ctrl+C) signal is received.
trap 'handle_signal SIGTERM' SIGTERM
trap 'handle_signal SIGUSR1' SIGUSR1

……..
# Counter for dots
……..
   # Start new line after max_dots
…….
```

Running the script:

```
clim@Lenovo16:~/THEORY/1VirtualM,Multitask$ ./S6.signal_demo.sh
=== Signal Demonstration ===
PID: 7819
Try these commands in another terminal:
kill -SIGUSR1 7819
kill -SIGTERM 7819
Or press Ctrl+C in this terminal
```

In another terminal, send signals:

```
clim@Lenovo16:~$ kill -SIGUSR1 7819
clim@Lenovo16:~$ kill -SIGTERM 7819
Received SIGUSR1
Received SIGTERM
Cleaning up and exiting...
```

We exit with Ctrl+C:

Key Learning Outcomes

1. **Unidirectional and Bidirectional Pipes:**
   Students experience real-world IPC mechanisms and see how data flows between processes.
2. **Signal Handling in Processes:**
   Controlled process termination and custom actions illustrate signal importance.
3. **Practical Debugging:**
   Using tools like kill and read, students learn to troubleshoot communication and control mechanisms.

Part 4: Docker Container Management in WSL (Duration: 10 minutes)

We conclude with Docker, one of the most impactful tools in modern computing.
*Docker allows us to create **isolated environments**, or containers, to run applications. It is indispensable for multitasking and API integration.*

Using the S7.docker_manager.sh script, you will:

1. ***Create a container running an NGINX web server.***
2. ***Monitor its CPU and memory usage using docker stats.***
3. ***Clean up all containers when finished.***

💡 *Why Docker? Containers ensure that applications run consistently, regardless of the host environment. They also optimize resource utilization by isolating processes.*

Interactive Exercise:

- *Access the running container via http://localhost:8080 in your browser.*
- *Experiment with starting, stopping, and restarting containers.*

Troubleshooting Note: If Docker permissions issues arise, we'll fix them with a quick setup script (03docker_setup_TOavoidWSLissues.sh).

Key Takeaway: *Docker's ability to integrate with WSL demonstrates multitasking, resource management, and API communication at scale.*

PART 4: DOCKER CONTAINER MANAGEMENT IN WSL

DOCKER MANAGEMENT SCRIPT

Script: S7.docker_manager.sh

Purpose: Simplifies Docker container setup, monitoring, and cleanup. Shows the integration of API support and multitasking.

```
#!/bin/bash
# Save as S7.docker_manager.sh

# Ensure Docker is running
ensure_docker() {
.....
}
##Function: ensure_docker
#       #Checks if Docker is installed and running.
#       #If the Docker service is inactive, attempts to start it.

# Basic container management
create_test_container() {
.....
}
##Function: create_test_container
#       #Runs an NGINX container mapped to port 8080.
#       #Ensures that old containers named test-nginx are removed to prevent conflicts.
show_containers() {
.....
}

container_stats() {
.....
}
##Function: container_stats
#       #Uses docker stats --no-stream to display CPU, memory, and network usage for active containers.
cleanup_containers() {
.....
}
##Function: cleanup_containers
#       #Stops all running containers and removes them using docker stop and docker rm.
```

```
# Menu
……
```

Running the script:

```
clim@Lenovo16:~/THEORY/1VirtualM,Multitask$ ./S7.docker_manager.sh

=== Docker Management ===
1. Create test container
2. Show containers
3. Show container stats
4. Cleanup containers
5. Exit
Choose option: 1
=== Creating Test Container ===
Container 'test-nginx' already exists. Removing it...
test-nginx
a260bc68fa92998c901b4ecd248603cc4384900741fdc108080fe93873dee2e2
Container created successfully!
Access nginx at http://localhost:8080

=== Docker Management ===
1. Create test container
2. Show containers
3. Show container stats
4. Cleanup containers
5. Exit
Choose option: 2
=== Active Containers ===
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS
          NAMES
a260bc68fa92   nginx:latest   "/docker-entrypoint.…"   3 seconds ago   Up 3 seconds   0.0.0.0:8080->80/tcp,
[::]:8080->80/tcp   test-nginx

=== Docker Management ===
1. Create test container
2. Show containers
3. Show container stats
4. Cleanup containers
5. Exit
Choose option: 3
=== Container Statistics ===
CONTAINER ID   NAME         CPU %     MEM USAGE / LIMIT     MEM %     NET I/O     BLOCK I/O
PIDS
a260bc68fa92   test-nginx   0.00%     15.71MiB / 31.11GiB   0.05%     806B / 0B   0B / 0B     21

=== Docker Management ===
1. Create test container
2. Show containers
3. Show container stats
4. Cleanup containers
5. Exit
Choose option: 4
=== Cleaning Up ===
a260bc68fa92
a260bc68fa92
All containers removed.

=== Docker Management ===
1. Create test container
2. Show containers
3. Show container stats
4. Cleanup containers
```

```
5. Exit
Choose option: 2
=== Active Containers ===
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

=== Docker Management ===
1. Create test container
2. Show containers
3. Show container stats
4. Cleanup containers
5. Exit
Choose option: 5
clim@Lenovo16:~/THEORY/1VirtualM,Multitask$
```

1. Execute the script:
   ./S7.docker_manager.sh
2. Follow the menu prompts to:
   - Create a test container.
   - Monitor active containers.
   - Clean up all containers.

Interactive Activity:

- Access the running container by opening http://localhost:8080 in a browser.

IF THERE ANY ERROR (…common Docker permissions issue in WSL). Let's fix this with a proper setup script:

```
#!/bin/bash
# Save as 03docker_setup_TOavoidWSLissues.sh

……
```

after run the command newgrp docker
return now to the script S7.**docker_manager.sh**

TROUBLESHOOTING COMMON WSL ISSUES

1. Docker Service Issues:

```
# Reset Docker service
sudo service docker stop
sudo rm -rf /var/lib/docker
sudo service docker start
```

there can be more issues so S8.docker_service_manager.sh could be a good idea to handle issues:

```
#clean up the current state:
sudo service docker stop
sudo killall -9 dockerd containerd
sudo rm -f /var/run/docker.sock
```

```
#install Docker properly if not already done:
sudo apt update
sudo apt install -y docker.io containerd

#add your user to the docker group:
sudo usermod -aG docker $USER
newgrp docker
```

For you future management try the S8.docker_service_manager.sh script.

This script manages common Docker service issues, ensuring smooth operation in WSL.

```
#!/bin/bash
# Save as S8.docker_service_manager.sh

……..
# Function to check if running in WSL
……..
}

# Function to check service status
check_status() {
…….
}
```

##service docker status: Checks the status of the Docker daemon.

```
# Function to stop Docker completely
stop_docker() {
……..
}

# Function to start Docker
start_docker() {
……
}

# Function to restart Docker
restart_docker() {
……
}
```

##service docker restart: Restarts the Docker service to resolve issues.

```
# Function to check if Docker is functioning
test_docker() {
……
      docker run --rm hello-world
```

##Tests if Docker is functioning by running a simple container.

```
   else
…….
…….
}

# Menu
……
```

Key Learning Outcomes

1. **Containerized Environments:**
   Students understand how to create and manage containers for isolated multitasking.
2. **Monitoring and Debugging:**
   Hands-on practice with docker stats and service management equips students to troubleshoot real-world issues.
3. **Practical Application:**
   The integration of containers with WSL demonstrates dynamic linking, multitasking, and API support in a professional setting.

2. WSL Memory Issues:

```
# Create/edit .wslconfig in Windows home directory
[wsl2]
memory=8GB
processors=4
```

3. Network Connectivity:

```
# Reset WSL network
wsl.exe --shutdown
netsh winsock reset
```

4. Process Management:

```
# Kill hung processes
wsl.exe --terminate Ubuntu
```

Additional Resources

- WSL Documentation: https://docs.microsoft.com/en-us/windows/wsl/
- Docker WSL Integration: https://docs.docker.com/desktop/windows/wsl/
- Bash Manual: $ man bash
- WSL GitHub Issues: https://github.com/microsoft/WSL/issues

Homework Assignment

1. Create a comprehensive system monitoring dashboard using the concepts learned
2. Implement a multi-container application with proper resource management
3. Write a report on WSL limitations and workarounds for virtualization tasks

Conclusion and Homework (Duration: 5 minutes)

What have we learned today?

1. *The role of **WSL** in bridging Linux and Windows environments.*
2. *How to manage **background processes** and multitask effectively.*
3. *The importance of **IPC** for process communication.*
4. *The power of **Docker containers** for isolated multitasking.*

Before you go, here are some homework assignments to reinforce your learning:

1. ***System Monitoring Dashboard:** Build a real-time monitoring script using Bash.*
2. ***Multi-Container Application:** Create a microservices architecture using Docker Compose.*
3. ***WSL Limitations Report:** Research challenges with WSL and propose workarounds.*

Thank you for your attention and active participation today. I hope you've found this seminar insightful and practical. If you have any questions or need clarification, feel free to ask during the remaining time. Happy coding, and I look forward to seeing your projects!

Thank you, and have a great day!

PRACTICE EXERCISES AND HOMEWORK ASSIGNMENTS

Reinforce the concepts of virtual machines, multitasking, inter-process communication (IPC), and Docker container management through hands-on exercises and creative problem-solving.

Exercise 1: Process Management

Objective: Create a script to manage and monitor multiple background processes.

Tasks

1. Write a script to:
    - Start five background tasks that log their activity to separate files.
    - Monitor their CPU and memory usage every 10 seconds.
2. Provide options to:
    - Start new tasks.
    - Stop specific tasks by entering their process ID (PID).

Example Workflow:

1. Run the script:
2. ./process_manager.sh

Menu:

1. Start a new task
2. Stop a task by PID
3. Show running tasks
4. Exit

3. Expected Output (while monitoring tasks):
4. Task 1: CPU 5.3%, Memory 12.8MB
5. Task 2: CPU 3.2%, Memory 8.4MB

Exercise 2: IPC Implementation

Objective: Build a client-server system using named pipes for communication.

Tasks

1. Create a server script that:
    - Waits for incoming messages from clients via a named pipe.
    - Logs received messages with timestamps.
2. Create a client script that:
    - Sends user input messages to the server via the named pipe.
    - Accepts a "quit" command to stop communication.

Example Interaction:

Server Output:

Received: "Hello from Client 1" at Sat Jan 11 15:00:00 EET 2025
Received: "quit" at Sat Jan 11 15:01:00 EET 2025
Shutting down server...

Client Interaction:

Enter message: Hello from Client 1
Message sent.
Enter message: quit

Exercise 3: Container Deployment

Objective: Create a multi-container application and manage it with Docker.

Tasks

1. Write a script to:
    - Set up two containers:
        - **Container 1:** A simple web server (NGINX).
        - **Container 2:** A database server (MySQL).
2. Add features for:
    - Health checks for the containers.
    - Real-time resource monitoring using docker stats.
    - Cleanup of unused containers and images.

Expected Script Menu:

1. Create web and database containers
2. Show container stats
3. Run health checks
4. Stop and remove all containers
5. Exit

Assignment 1: System Monitoring Dashboard

Objective: Build a comprehensive monitoring dashboard using Bash scripts to track CPU, memory, disk, and network usage in real-time.

Requirements:

- Display:
  - Top CPU-consuming processes.
  - Memory and swap usage.
  - Disk I/O statistics.
  - Active network connections.

Example Display:

=== System Monitoring Dashboard ===
1. CPU Usage: 35% | Top Process: chrome (15%)
2. Memory Usage: 8GB / 32GB | Swap: 512MB used
3. Disk I/O: Read 5MB/s | Write 2MB/s
4. Active Connections: 10 (5 inbound, 5 outbound)

Assignment 2: Multi-Container Application

Objective: Build a multi-container application simulating a microservices architecture.

Requirements:

1. Containers:
   - **Frontend:** React or NGINX server.
   - **Backend:** Node.js or Python Flask application.
   - **Database:** MySQL or PostgreSQL.
2. Features:
   - Set up Docker Compose to manage the services.
   - Ensure containers communicate over a shared network.
   - Implement container health checks and logging.

Deliverables:

- A functional docker-compose.yml file.
- Health check scripts for each service.
- Documentation on how to deploy and test the system.

Assignment 3: WSL Limitations and Workarounds

Objective: Research and document the limitations of WSL for virtualization tasks and propose practical solutions.

Key Points to Address:

1. WSL memory and CPU usage constraints.
2. Docker integration challenges in WSL.
3. Alternatives to WSL for heavy virtualization workloads.

Deliverables:

- A detailed report (1-2 pages) with examples and references.
- Recommendations for overcoming common issues.

Key Learning Outcomes

1. ***Process Management:***
   Students gain practical skills in managing background tasks and multitasking effectively.
2. ***IPC Mastery:***
   Understanding and implementing communication between processes prepares students for complex systems design.
3. ***Containerized Applications:***
   Real-world experience with Docker fosters an understanding of microservices and virtualization concepts.

Troubleshooting Tips

1. **Docker Service Issues:**
    - Use the script S8.docker_service_manager.sh to check and restart the Docker service.
    - Common fix for permissions:
    - sudo usermod -aG docker $USER
    - newgrp docker
2. **WSL Resource Management:**
    - Adjust .wslconfig in the Windows home directory to allocate more resources:
    - [wsl2]
    - memory=8GB
    - processors=4
3. **Stuck Processes:**
    - Use kill or pkill to terminate unresponsive processes:
    - pkill -f <process_name>