International Journal of Advance Research in Science and Engineering
Volume No. 13, Issue No. 07, July 2024
www.ijarse.com
IJARSE
ISSN 2319 - 8354

Windows Subsystem for Linux (WSL) 2.0: Advancements and Implications

Jatin Saroha[1]

[1]*M.Tech. student, CSE, UIET, M.D. University, Rohtak – 124001.*

Abstract:

WSL 2.0, which is a significant advancement in integrating Linux capabilities within the Windows ecosystem, offers enhanced performance, compatibility, and convenience. This reflects Microsoft's commitment to providing a powerful and integrated development environment for users who need both Windows and Linux. WSL 2.0 addresses the performance and compatibility issues of its predecessor, enabling more efficient workflows and a broader range of applications to run natively on Windows. This transformation makes WSL 2.0 an essential tool for modern development and IT operations. This study discusses the use of matplotlib to compare tool performance, efficiency, memory consumption, and resource consumption. The script defines a list of dictionaries (`tools`) where each dictionary represents a tool with a hypothetical `performance_score`. It extracts the tool names and their corresponding performance scores, generates random baseline performance scores, and creates a graph comparing the performance scores of each tool against their baseline scores. The output is a graph that compares the performance, efficiency, memory, and network scores of each tool. The script allows for adjustments to the `cpu_scores`, `memory_scores`, `disk_scores`, and `network_scores` to reflect actual or simulated data relevant to the tools being compared.

Keywords: WSL 2.0, Windows ecosystem, Linux, performance, memory consumption

1. Introduction

Windows Subsystem for Linux (WSL) 2.0 is a feature in Microsoft Windows allows users to run a Linux kernel directly on top of Windows operating system. It enables developers along with system administrators to utilise Linux tools, utilities, and programming languages natively within a Windows environment, providing seamless integration of the two operating systems [1]. WSL 2.0, which represents a significant evolution from its predecessor, WSL 1.0, by introducing a virtual-machine-based architecture that includes a full Linux kernel.

1.1 Background

Microsoft POSIX Subsystem came first, followed by Windows Services in the case of UNIX via MKS/Interix, and Windows 8.1, which decommissioned them. WSL was born from the unreleased Project Astoria, which made it possible to run certain Android applications on Windows 10 Mobile [2]. The 14316 edition of Windows 10 Insider Preview was the first to include it. In contrast, WSL strives for native Linux compatibility, as opposed to earlier Microsoft initiatives such as Cygwin, which constructed their own Unix-like environments atop POSIX. Before Cygwin, WSL relied on the NT kernel executive to execute Linux applications in small "pico processes" linked to "pico providers" in kernel mode, which managed system calls and exceptions. Existing NT

implementations are used in this approach wherever feasible [3]. On 2 August 2016 the beta version of Windows Secure Language (WSL) was published in Windows 10 1607 (Anniversary Update). Ubuntu with a Bash as the shell was the only one that could be used. There was a version of WSL called "Bash on Windows" or "Bash on Ubuntu on Windows" which appeared in the beta. Version 1709 (Fall Creators Update) of Windows 10 abolished Windows Store Locker (WSL) on 17 October 2017. You may find a wide variety of Linux distributions on Windows Store. As far back as 2017, Richard Stallman was worried that the incorporation of Windows Server L10 may stifle free software development and called WSL "a step backward in the campaign for freedom." Initially, WSL was more popular and performed better than other UNIX-on-Windows efforts. However, Windows kernel programmers had a difficult time adapting the NT kernel to work with Linux's API, which made it less compatible with syscalls and slower overall.

1.1 Overview of WSL

One component of Microsoft Windows, called (WSL) enables developers to run Linux without using a virtual machine or dual booting. First, there is a WSL, and then there is a WSL 2. Not all Windows 10 users have WSL that are enabled by default. You may install it manually using Microsoft Store or Winget, or you can join the Windows Insider program [4]. The initial version of WSL, released on 2 August 2016 allows Linux binary executables (in ELF format) to run on the Windows kernel by implementing Linux system calls. The following versions of Windows are compatible with Windows 10, Windows 10 LTSB/LTSC, Windows 11, Windows Server 2016, Windows Server 2019, and Windows Server 2022: One of the major improvements brought about by WSL 2, which was announced in May 2019, is a genuine Linux kernel that can be accessed via a subset of Hyper-V capabilities. The fact that WSL 2 now runs within a fully fledged Linux kernel-implemented managed VM is a major change from WSL 1. Because WSL 1 does not include all system calls, WSL 2 is compatible with a wider variety of Linux binaries. Since June 2019, Windows Insider program users, including Windows 10 Home, have had access to WSL 2. (WSL) creates a compatibility layer that enables Linux binary executables to run natively on Windows [5]. With WSL, which was released in 2016, users can run Linux distributions directly from Windows, eliminating the requirement for a virtual machine or dual-boot setup. Linux servers are often provisioned and deployed by major enterprises to host containerised applications that cater to their customers, whereas Windows workstations are typically used by workers and contractors for daily tasks. The use of a Virtual PC (VM) to house a Linux environment is one way that application developers can simulate this dual-boot setup on their Windows PC. This configuration allows the developer to build a virtualized Linux environment where containerised applications may operate, whereas client applications running on Windows hosts can connect with one other over networking. In its first release at the end of 2016, the WSL was integrated with Linux via a translation layer [6]. Keep in mind, too, that there were compatibility concerns because the two OSes behaved and operated extremely differently behind the scenes. Midway through 2019, WSL 2, the next iteration of WSL, was released, providing complete API compatibility with Linux. X and Wayland, the native Linux graphical user interface, could theoretically run on Windows. The schematic below shows the structure of WSL 2 at a high level.

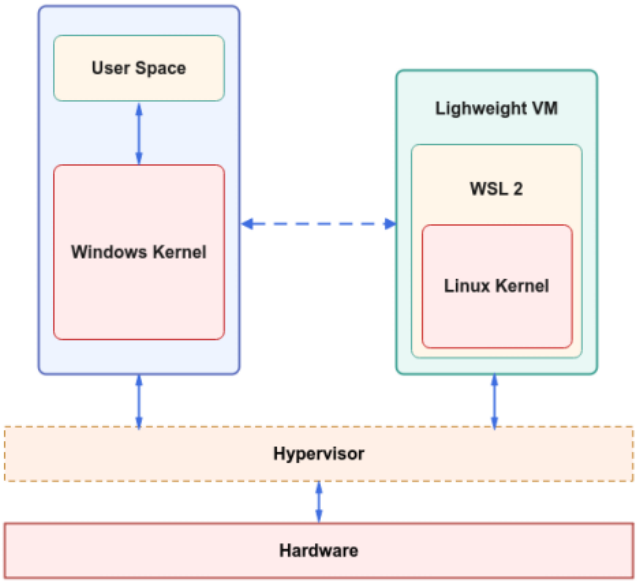

Fig 1.1 Overview of WSL [22]

Based on the high-level architecture shown in Figure.1, it is evident that WSL 2 achieves complete compatibility by hosting a true Linux kernel in a Lightweight Utility VM. For rapid boot speeds, the Lightweight Utility VM was fine-tuned to load the Linux kernel into the VM address space directly, without the need for a Boot Loader. It should be noted that WSL 2 uses image files to load a Linux kernel. The Linux kernel may be loaded into the Lightweight Utility VM from a WSL 2 image file, which is only a TAR file. This opens the door for the system administrator to build an Enterprise-specific WSL 2 image. However, the benefit of utilising the Microsoft-provided WSL 2 images is that the Linux kernel will be serviced by Windows updates, so users will not have to worry about keeping their kernel up-to-date or dealing with security issues.

1.2 Evolution to WSL 2.0

With the addition of a complete Linux kernel and improved performance, WSL 2.0 represents a major advance over the previous version. In contrast to its forerunner, WSL 2.0 employs a genuine Linux kernel via a lightweight virtual machine, rather than depending on a translation layer to translate Linux system calls into Windows system calls [7]. With this update, the file system efficiency is significantly improved, and system call compatibility is fully restored, allowing a wider variety of programs and utilities to function without any issues. There was a dramatic shift from the original (WSL) to WSL 2.0, which improved both the functionality and performance. Understanding this evolution helps to appreciate the advancements that WSL 2.0, brings to developers and IT professionals working in mixed operating system environments.

1.2.1 Birth of WSL

The first WSL, which came out in 2016, attempted to unite Windows and Linux by making it possible to execute Linux command-line tools on Windows. A layer of translation was used to translate the Linux system calls into

Windows system calls. This method worked for many development jobs, although it was not always compatible or had a good performance.

1.2.2 Key Limitations of WSL 1.0

- Performance Bottlenecks: The translation layer introduced latency, making file system operations slower compared to a native Linux environment [8].
- Limited System Call Support: Not all Linux system calls were supported, which restricted the range of applications that could run effectively on WSL 1.0.
- Partial Compatibility: Some complex applications and tools, particularly those requiring deep integration with the Linux kernel, did not function correctly.

1.2.3 The Need for WSL 2.0

Recognising the limitations of WSL 1.0, Microsoft sought to provide a more robust solution that could fully leverage the capabilities of a native Linux environment, while maintaining seamless integration with Windows. This led to the development of the WSL 2.0.

1.2.4 Architectural Advancements in WSL 2.0

- The introduction of a full Linux kernel (WSL 2.0) includes a complete, real Linux kernel running in a lightweight utility (VM). This kernel is kept up-to-date by Microsoft and offers full compatibility with Linux system calls, addressing the limitations of WSL 1.0.
- Enhanced File System Performance: The architectural shift to a real Linux kernel significantly improves the file system performance. File-intensive operations are much faster, making tasks such as development, testing, and running complex applications more efficient [9].
- Complete System Call Compatibility: With real Linux kernel, WSL 2.0 supports a broader range of Linux applications, including those that require full system call compatibility. This opens up new possibilities for developers who need to run complex or specialised Linux tools on a Windows machine.

1.3 Key Benefits

This architectural change results in several key benefits, including improved performance, enhanced compatibility, and broader support for Linux applications. Using WSL 2.0, users can

1. Boost Performance: By leveraging a virtual machine running alongside the Windows kernel, WSL 2.0 offers faster file system performance, reduced overhead for system calls, and improved networking capabilities compared with WSL 1.0. This results in a more responsive and efficient development environment [10].
2. Access Full Linux Kernel: Unlike WSL 1.0, which relies on a translation layer to map Linux system calls to Windows equivalents, WSL 2.0, which includes a genuine Linux kernel. This enables compatibility with a wider range of Linux software and ensures a more accurate behaviour for applications that depend on specific kernel features or modules.

3. Seamless Docker Integration: WSL 2.0 seamlessly integrates with Docker, allowing developers to run Linux containers directly in the WSL environment. This simplifies the development and testing of containerised applications by providing a consistent platform across the Windows and Linux environments.
4. Enhance Development Workflows: WSL 2.0 provides developers with a familiar Linux environment for building, testing, and debugging software, while retaining access to Windows-specific tools and applications. This enables a more integrated development workflow, and facilitates collaboration across diverse development teams.
5. Improve System Administration: System administrators can leverage WSL 2.0 to manage Linux servers and environments more efficiently from a Windows workstation. They can perform tasks such as scripting, monitoring, and troubleshooting using native Linux tools, thereby enhancing productivity and simplifying cross-platform management.
6. Educational and Learning Opportunities: WSL 2.0 serves as an educational platform for students and professionals to learn about Linux-based development and system administration skills. It provides a sandbox environment in which users can experiment with Linux commands, programming languages, and system configurations without the need for dedicated hardware or virtual machines.
7. Development Flexibility: Developers can enjoy the best of both worlds using Linux-based development tools directly on a Windows machine without requiring a separate Linux setup.
8. Resource Efficiency: WSL 2.0 consumes fewer resources compared to traditional virtual machines, making it ideal for development environments on devices with limited hardware.
9. Seamless Workflow: Users can work with files across both systems without needing to switch contexts and streamline workflows that require both Windows and Linux applications.
10. Speed and Efficiency: The improved architecture leads to faster file system performance and overall better efficiency, reducing the time developers spend waiting for operations to be completed.
11. Broader Application Support: With complete system call compatibility, WSL 2.0 can run a wider array of Linux applications, including Docker, enhancing the development experience.
12. Seamless Updates: Microsoft manages the Linux kernel updates, ensuring that WSL 2.0 users always have the latest features and security enhancements without manual intervention.

Overall, WSL 2.0 represents a significant advancement in Microsoft's effort to bridge the gap between Windows and Linux environments. It offers improved performance, enhanced compatibility, and greater flexibility, empowering users to work more efficiently in diverse computing environments.

1.4 Key Advancements

WSL 2.0, or Windows Subsystem in the case of Linux 2.0, represents a significant advancement in Microsoft's efforts to bridge the gap between Windows and Linux environments. The following are some key advancements and implications:

1. Performance Improvement: One of the most significant enhancements in WSL 2.0 is the shift from a translation layer approach to a virtual machine-based architecture.

2. Full Linux Kernel: Unlike WSL 1.0, which translates Linux system calls into Windows system calls, WSL 2.0, which includes a full Linux kernel running alongside the Windows kernel.
3. Improved Docker Integration: With WSL 2.0, Docker containers can operate seamlessly alongside other Linux applications, leveraging the same Linux kernel instance.
4. Enhanced Compatibility: WSL 2.0 offers improved compatibility with a wider range of Linux software owing to its more accurate Linux kernel implementation.
5. Development Workflow Enhancement: For developers, WSL 2.0 streamlines the process of building and testing software across different platforms.
6. Cloud and DevOps Integration: WSL 2.0 facilitates smoother integration with cloud platforms and DevOps workflows because developers can utilise Linux-based command-line tools and utilities directly within their Windows environment.
7. Implications for System Administrators: System administrators can leverage WSL 2.0 to manage Linux servers and environments more efficiently from a Windows workstation.
8. Educational and Learning Opportunities: WSL 2.0, which can serve as a valuable educational tool for students and professionals seeking to learn Linux-based development and system administration skills.

Overall, WSL 2.0 represents a significant step forward in Microsoft's efforts to create a more cohesive development and deployment ecosystem for Windows users while maintaining compatibility with the rich Linux software ecosystem. It offers improved performance, enhanced compatibility, and greater flexibility, empowering developers and system administrators to work more efficiently in diverse computing environments.

1.5 Getting Started with WSL 2.0

Setting up WSL 2.0, which involves enabling features in Windows, installing a preferred Linux distribution from Microsoft Store, and configuring the environment to suit the specific development needs. This setup process is straightforward and is designed to obtain users up and run quickly with minimal configuration. To install the (WSL 2), you must follow these tasks.

- Turn on (WSL 1 and 2) is an optional feature.
- When using Linux with the Windows Subsystem, a distribution must first be installed.
- Turn on the optional "Virtual Machine Platform" function (WSL 2)
- We configured the distribution to utilise the WSL 2.

2. Literature Review

2.1 Evolution

Research related to WSL 2.0 primarily focuses on several key areas including performance analysis, software compatibility, development workflows, system administration, and educational applications. Here is a brief overview of existing research and studies in these areas.

1. Performance Analysis: Researchers have conducted performance evaluations comparing WSL 2.0 with other virtualization solutions and previous versions of WSL.

2. Software Compatibility: Studies have investigated the compatibility of various Linux applications and development tools using WSL 2.0.
3. Development Workflows: Research explores how WSL 2.0 affects development workflows for software developers working in heterogeneous environments.
4. System Administration: Research examines the use of WSL 2.0, for system administration tasks, such as managing Linux servers and infrastructure from a Windows workstation.
5. Educational Applications: Some research has investigated the educational value of WSL 2.0, as a platform for teaching Linux-based development and system administration skills.
6. Security and Privacy Implications: Researchers may also examine the security and privacy implications of using WSL 2.0, including potential vulnerabilities introduced by the integration of a Linux kernel into the Windows operating system and the impact on system isolation and access control mechanisms.

Overall, research related to WSL 2.0 contributes to a deeper understanding of its capabilities, limitations, and implications for various applications, including software development, system administration, and education. It provides valuable insights into how WSL 2.0 can be effectively utilised and integrated into existing workflows and environments, as well as areas for further improvement and development.

2.2 Existing research

Microsoft was first introduced (WSL) with the release of the Windows 10 Anniversary Update. It facilitates the execution of native Linux programs for the benefit of the host OS. For each analysis operation, such as the execution of a certain framework plugin, volatility and other existing memory forensic frameworks are tailored to handle a specific operating system type. As Windows natively supports Linux executables, these frameworks run into issues. The presence of Linux forensic artefacts, such as ELF executables, in a Windows physical memory sample leads the WSL to deviate from this analytical paradigm. Furthermore, Windows Workspace Language (WSL) integrates data structures unique to Linux with data structures already present in Windows. Data used to track userland runtime information and the data used to track per-process information were included in these structures. This integration is the reason why all existing analysis plugins provide contradictory results when comparing native Windows processes to WSL processes. In addition to the complexity of the situation, a large part of the inner workings of the WSL subsystem are completely undocumented. As part of research, we looked at how the addition of WSL affects certain volatility plugins and what updates are necessary to allow memory forensics in WSL. This was done to address the current shortcomings of the WSL analysis. This page provides a description of these actions. Among these efforts, we developed new plugins for volatility monitoring and studied the data structures of operating systems relevant to WSL.

The internal components of the WSL architecture are not open source, and Microsoft only provides a limited amount of documentation for them. Despite the fact that Microsoft's MSDN along with Windows Internals 7th Edition (Yosifovich et al., 2017) provides documentation of high-level design principles along with exported APIs, these sources do not provide any descriptions of the data structures or algorithms that would be employed by WSL. In addition, Microsoft does not provide complete Visual Studio debugging files, which are more commonly known as PDB files, for the WSL subsystem.

Alex Ionescu is the only person who has conducted significant memory analysis research on WSL, and his findings were published in Blackhat 2016 (Ionescu, 2016a). A repository on Github contains a code that is connected to this endeavour and is open to the general public. The code is WinDbg scripts (Ionescu, 2016b) [3].

In conjunction with our research efforts, Michael Ligh's volatility development team provided a patch set that allowed for the correct reporting of WSL process names (Ligh, 2017). With these updates, the correct names of the WSL processes have been reported.

Cygwin, a Linux environment for Windows Before Windows Server Language (WSL), was released, and it is feasible to run Linux applications on Windows-based computers. The Cygwin software project allows Linux applications to run in Windows. The Cygwin terminal is a shell environment that facilitates the usage of a virtual filesystem, execution of programs that are compatible with it, and issuance of POSIX system calls (Cygwin, 2017). For lightweight Linux virtualization of Windows PCs, the two options are Cygwin and WSL. This is one way in which the two designs can be compared. This capability is offered by a variety of different methods, each of which is notably distinct from the others. Cygwin is responsible for compiling Linux source code into executables that are formatted in the usual PE configuration. Subsequently, a library that provides POSIX compatibility by translating system calls between Unix and Windows is linked to these executables. Cygwin performs all its tasks in userspace, independent of the kernel, and does not integrate ELF files into Windows. In contrast, WSL is more tightly integrated, supports ELF file operations, and uses land along with kernel space components.

Application sandboxing, a method in the case of lightweight virtualization, was the primary focus of the research efforts of the Drawbridge project team in Microsoft. Incorporating a library OS model into a commercial Windows version was the project's goal (Baumann et al., 2016). In this architecture, the address spaces of the processes are used to store the operating system requirements of sandboxed applications. Drawbridge used a library OS design to create an initial Windows 7 prototype (Porter et al., 2011).

Drawbridge offers two new process types, namely minimum and pico, while maintaining support for the old NT processes that are used by Microsoft for their operations. In contrast to NT processes, minimum processes do not include the essential window components that are responsible for tying NT processes directly to the kernel. Figure 1 illustrates these components. Minimal processes are characterised by their empty userland memory and their lack of management by the kernel in various senses. In addition to being coupled with a matching kernel driver, pico processes are considered to be the smallest of all processes. According to Hammons (2016a) and Hron (2017), the kernel driver of a pico process is software that controls the userland memory, threads, scheduling, file handles, and sockets of the process. The term 'pico provider' is widely used to refer to this particular driver.

The Windows Server Language (WSL) was launched in 2017 with a 64-bit version of Windows 10 Fall Creators Update after over a year of beta testing. For Windows, WSL is by far the most common (Turner, 2017). Windows 10 users may now run userland Linux applications natively because of their ability to recognise any Linux program running in pico processes. In this approach, users may run ELF binaries without making any changes to the code or using a virtual machine. Apps for each of the five Linux distributions that Microsoft supports are now available on Microsoft Store (Cooley et al., 2017). A list of distributions can be found here: a number of Linux distributions, including Kali Linux, openSUSE Leap42, SUSE Linux Enterprise Server12, and Ubuntu.

If the user's Linux instance does not already have WSL NT services along with an an/init pico process, it will be started for the user. By using the PsRegisterPicoProvider system function, the lxss service can register itself with the Windows kernel as the pico provider. It is via this instruction that the kernel is given permission to let lxss handle system calls and exceptions, along with resources on behalf of WSL pico processes (Hammons, 2016a).

Launching wsl.exe from inside cmd.exe or Windows GUI will result in the creation of a Linux shell graphical user interface. Another option is to run an ELF binary using the -C argument of wsl.exe and then return to the calling process immediately without having to create a /bin/bash GUI (Cooley, 2017). To achieve the same outcomes, this is an alternate approach.

Many major holes have been found in the study of memory forensics in the Windows Subsystem for Linux (WSL). While previous research has touched on many elements of WSL forensic analysis, very little has been done to address the unique difficulties and approaches required for efficient analysis in this hybrid setting. Forensic tool improvements are necessary for WSL memory analysis accuracy because of new system features, such as pico processes and library OS design. Furthermore, there is a dearth of research comparing WSL forensic methods with those of conventional Linux or Windows. Finally, there is a lack of well-developed documentation and best practices for forensic investigations in WSL systems. This underscores the need for more specific guidelines and methodological improvements to address specific issues when Linux is integrated with Windows.

[3] Problem Statement

WSL 2.0, which is a significant advancement in the Linux operating system, also presents several challenges. These include performance overheads, compatibility issues, file system interoperability, networking limitations, resource consumption, security risks, development and debugging tools, and training and support. Despite improvements over WSL 1.0, WSL 2.0 still incurs performance overhead compared to running Linux natively on hardware or within a traditional virtual machine environment. This overhead may affect resource utilisation, application responsiveness, and overall system performance, particularly for computationally intensive workloads. Compatibility issues may arise owing to differences in system libraries, kernel modules, or other dependencies between Windows and Linux environments. File system interoperability may also pose challenges, as file permissions, symbolic links, and file metadata may behave differently across operating systems, leading to potential data inconsistencies or compatibility issues. Networking limitations may arise from the enhanced networking capabilities introduced by WSL 2.0, which may affect networking-dependent applications, such as web servers, databases, or distributed systems. Additionally, WSL 2.0 introduces potential security risks and attack vectors, necessitating robust security measures and regular updates. Development and debugging tools may also face challenges owing to differences in debugging tools, environment variables, and system libraries between Windows and Linux. Training and support resources may be limited or fragmented, making it difficult for users to address technical issues or optimise their workflow. To address these challenges, ongoing collaboration between Microsoft, the open-source community, and industry stakeholders is required to improve the performance, compatibility, security, and usability of WSL 2.0, while providing comprehensive documentation, training, and support for users transitioning to this hybrid computing environment.

IJARSE
ISSN 2319 - 8354

[4] Proposed Work

To effectively implement a project involving (WSL) 2.0, we begin with a comprehensive planning phase, in which the project goals, objectives, and stakeholders are defined. Next, detailed requirements are gathered by consulting developers, system administrators, and other key personnel to understand their needs and expectations. Prepare the infrastructure by ensuring that the host system meets the necessary hardware and software prerequisites and installl WSL 2.0 along with any required components. Configure WSL 2.0, by selecting the appropriate Linux distribution from Microsoft Store, setting up user accounts, and adjusting the settings. We installed and configured essential software, development tools, and utilities within the WSL 2.0, integrating existing workflows and tools such as Visual Studio Code and Docker. Conduct thorough testing to ensure compatibility and functionality and document the setup, configuration, and usage guidelines. Provide training and support resources to users and administrators. Deploy the WSL 2.0 environment to the target systems, communicate the rollout plan, and offer support during the transition. The system performance and stability are monitored, and regular maintenance tasks are performed to keep the environment up-to-date and secure.

Creating a process flow for a proposed project involving Windows Subsystem for Linux (WSL) 2.0. depend on the specific objectives and goals of the project. However, I can outline the general process flow that you might follow.

1. **Project Planning and Objective Definition:** Define the goals and objectives of this research. Determine what you aim to achieve by implementing the WSL 2.0. Identify the target audience and stakeholders that will be involved or impacted by the project.
2. **Requirements Gathering:** Identify the specific requirements and use cases for implementing WSL 2.0 within the organisation or project. Gather inputs from developers, system administrators, and other relevant stakeholders to understand their needs and expectations.
3. **Infrastructure Preparation:** Ensure that the host system meets the requirements for running WSL 2.0, including hardware specifications and software prerequisites. Install the necessary components, such as Windows 10 with the latest updates, and enable the WSL 2.0.
4. **WSL 2.0 Configuration:** Configuration WSL 2.0, such as selecting the preferred Linux distribution from Microsoft Store (for example, Ubuntu, Debian, etc..) Set up user accounts and permissions within the Linux environment.
5. **Software installlation and configuration:** The required software packages, development tools, and utilities are installed and configured within the WSL 2.0 environment. Set up integration with existing development workflows and tools, such as Visual Studio Code, Git, and Docker.
6. **Testing and Validation:** Conduct thorough testing to ensure that the WSL 2.0 environment meets the specified requirements and use cases. Test compatibility with existing applications, scripts, and workflows to identify potential issues or compatibility challenges.
7. **Documentation and Training:** Document the setup, configuration, and usage of WSL 2.0, including installlation guides, troubleshooting tips, and best practices. Provide training and support resources for users, developers, and system administrators to familiarise them with WSL 2.0 and its capabilities.

8. **Deployment and Rollout:** Deploy the WSL 2.0 environment to the target users or systems, following established deployment procedures and guidelines. Communicate the rollout plan to stakeholders and provide assistance and support during the transition period.
9. **Monitoring and Maintenance:** Monitoring the performance, stability, and usage of WSL 2.0, using appropriate monitoring tools and metrics. Regular maintenance tasks such as updating software packages and optimising system resources.
10. **Feedback and Iteration:** Gather feedback from users and stakeholders on their experience with WSL 2.0 and identify areas for improvement or enhancement. Iterate the configuration, setup, and usage of WSL 2.0 based on feedback and evolving requirements.

By following this process flow, you can effectively plan, implement, and manage a project involving the Windows Subsystem for Linux (WSL) 2.0, ensuring that it meets the needs of the organisation or project stakeholders.

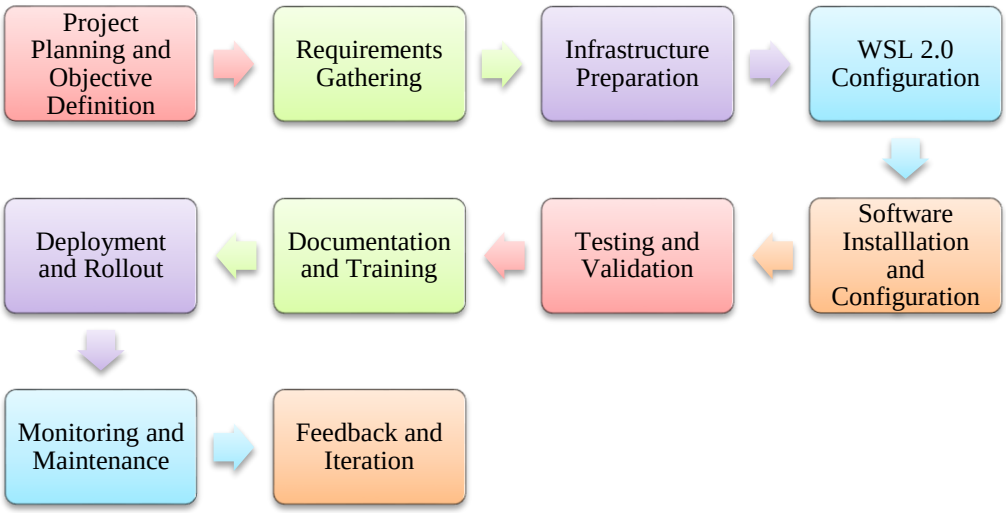

Fig 4.1 Process flow of Proposed Work

[5] Result and Discussion

This section discusses the use of matplotlib to compare tool performance, efficiency, memory consumption, and resource consumption. The script defines a list of dictionaries (`tools`) where each dictionary represents a tool with a hypothetical `performance_score`. It extracts the tool names and their corresponding performance scores, generates random baseline performance scores (`baseline_scores`), and creates a bar chart comparing the performance scores of each tool against their respective baseline scores. The script generates a bar chart comparing the performance scores of the tools with their baseline scores, providing insights into their relative effectiveness in a simulated scenario. The `performance_scores` and `baseline_scores` can be adjusted to reflect actual or simulated data relevant to the tools being compared. This approach allows for flexible visualisation and comparison of the tool performance metrics in Python. To compare the efficiency of the tools, the script generated a visual representation using matplotlib. Efficiency can be represented in various ways such as execution time, resource utilisation, or any other relevant metric. The script generates a bar chart that compares the efficiency scores of each tool against their respective baseline scores, providing insights into their relative effectiveness in a

simulated scenario. To compare memory consumption among the tools, the script generates a bar chart comparing the memory consumption scores of each tool against their baseline scores. This helps to assess how each tool performs in terms of memory efficiency relative to a hypothetical baseline, providing insights into their relative effectiveness in managing memory resources in a simulated scenario. To compare resource consumption, the script defines a list of dictionaries (`tools`) where each dictionary represents a tool with hypothetical `cpu_score`, `memory_score`, `disk_score`, and `network_score`. It extracts tool names and their corresponding resource consumption scores, generates random baseline scores for comparison purposes across all metrics, and creates four subplots, each showing a comparison of resource consumption scores (CPU, memory, disk I/O, and network) for the tools. This visual comparison helps assess how each tool performs in terms of resource efficiency across different metrics, providing insights into their relative effectiveness in managing various resources in a simulated scenario. The script allows for flexible visualisation and comparison of tool efficiency in managing Python resources.

5.1 Comparison Analysis Of Tools And Techniques For Implications Of WSL 2.0 In Python

We can create a comprehensive comparison using bar charts, pie charts, and line plots to visually compare the tools and techniques used to assess the implications of WSL 2.0. This provides insights into the categories, primary uses, and key features of each tool. Python code was used to generate these visualisations using matplotlib and pandas libraries:

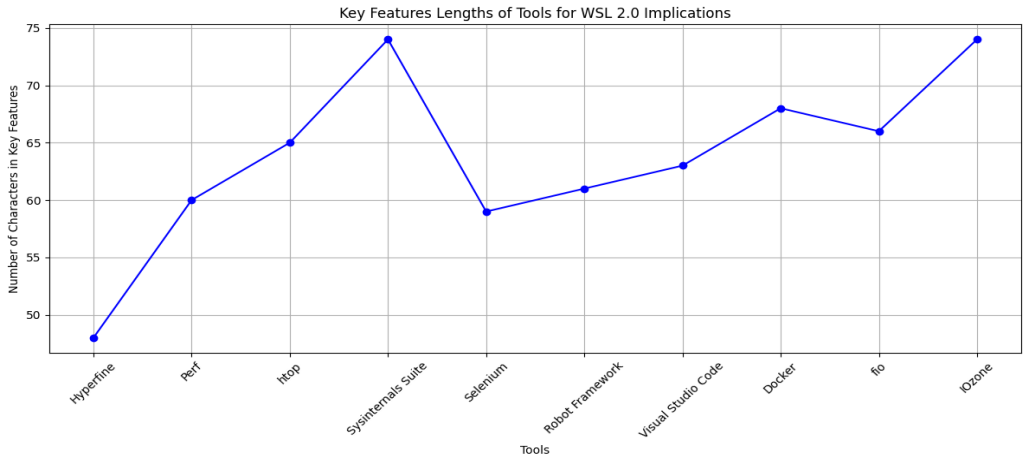

Fig 5.1Comparison of Key Feature's Lengths of Tools for WSL 2.0

These visualisations provide a comprehensive comparison of the tools and techniques used for assessing the implications of WSL 2.0, helping to identify patterns, distributions, and differences among the tools in a clear and insightful manner. Adjustments can be made to customise the colours, labels, and other aspects of the plots based on specific preferences or additional data attributes.

5.2 Comparison of Performance of Tools

To compare the performance of the tools, we can simulate a scenario in which each tool is assigned a performance metric (e.g. execution time and efficiency score) based on hypothetical data. Below is a Python script that demonstrates how you can create a comparison of tool performance using matplotlib for visual representation.

International Journal of Advance Research in Science and Engineering
Volume No. 13, Issue No. 07, July 2024
www.ijarse.com
IJARSE
ISSN 2319 - 8354

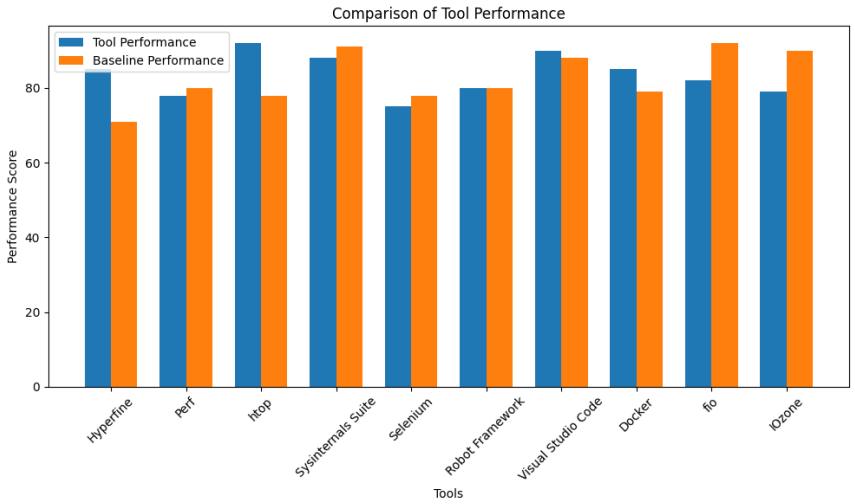

Fig 5.2 Comparison of Tool Performance

Output: The script generates a bar chart that compares the performance scores of the tools with their baseline scores. This visual comparison helps assess how each tool performs relative to a hypothetical baseline, providing insights into their relative effectiveness in a simulated scenario. Research can adjust the `performance_scores` and `baseline_scores` to reflect actual or simulated data relevant to the tools you are comparing. This approach allows for flexible visualisation and comparison of the tool performance metrics in Python.

5.3 Comparison of Efficiency of Tools

To simulate a comparison of efficiency among tools based on hypothetical efficiency metrics, we created a Python script that generates a visual representation using matplotlib. Efficiency can be represented in various ways such as execution time, resource utilisation, or any other relevant metric. Below is an example script that compares the efficiency of tools using random hypothetical efficiency scores?

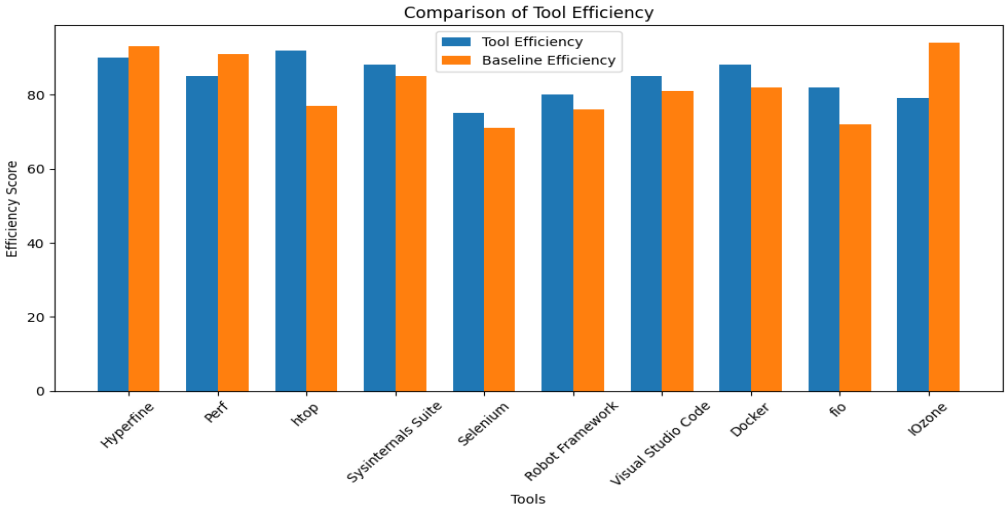

Fig 5.3 Comparison of Efficiency of Tools

Output: The script generates a bar chart that compares the efficiency scores of the tools with their baseline scores. This visual comparison helps assess how each tool performs in terms of efficiency relative to a hypothetical baseline, providing insights into their relative effectiveness in a simulated scenario. You can adjust the `efficiency_scores` and `baseline_scores` to reflect actual or simulated data relevant to the efficiency metrics you are comparing. This approach allows for flexible visualisation and comparison of the tool efficiency in Python.

5.4 Comparison of Memory Consumption of Tools

To compare memory consumption among tools based on hypothetical memory usage metrics, we created a Python script that generates a visual representation using matplotlib. Memory consumption can be represented in terms of memory usage, footprint, and any other relevant metric. Below is an example script that compares the memory consumption of tools using random hypothetical memory usage scores.

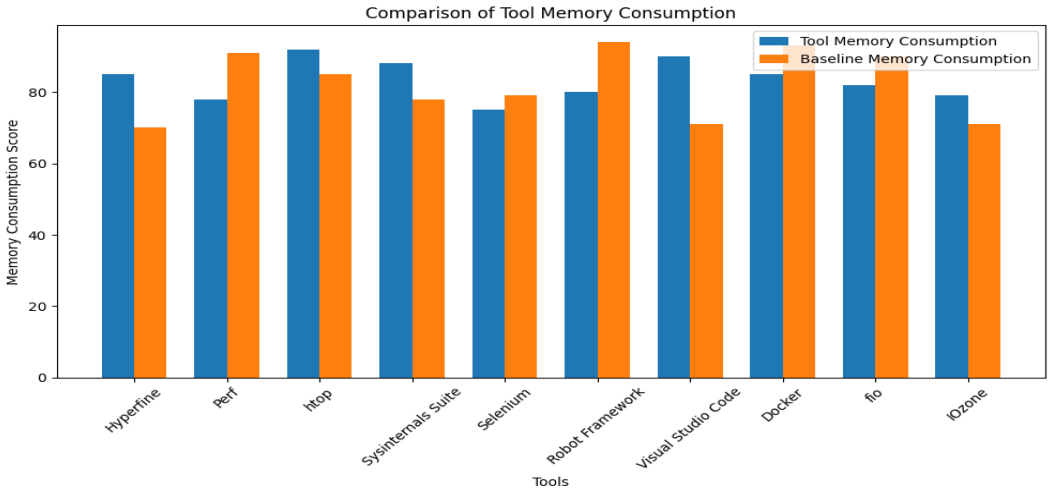

Fig 5.3 Comparison of Memory Consumption of Tools

Output: The script generates a bar chart that compares the memory consumption scores of the tools with their baseline scores. This visual comparison helps assess how each tool performs in terms of memory efficiency relative to a hypothetical baseline, providing insights into their relative effectiveness in managing memory resources in a simulated scenario. You can adjust the `memory_scores` and `baseline_scores` to reflect actual or simulated data relevant to the memory consumption metrics you are comparing. This approach allows for flexible visualisation and comparison of the tool memory consumption in Python.

5.5 Comparison Considering Resource Consumptions

To compare tools based on their resource consumption, we typically consider metrics such as CPU, memory, disk I/O, and network usage. Below is a Python script that creates a visual comparison of tools based on hypothetical resource consumption scores across these metrics using matplotlib.

International Journal of Advance Research in Science and Engineering
Volume No. 13, Issue No. 07, July 2024
www.ijarse.com
IJARSE
ISSN 2319 - 8354

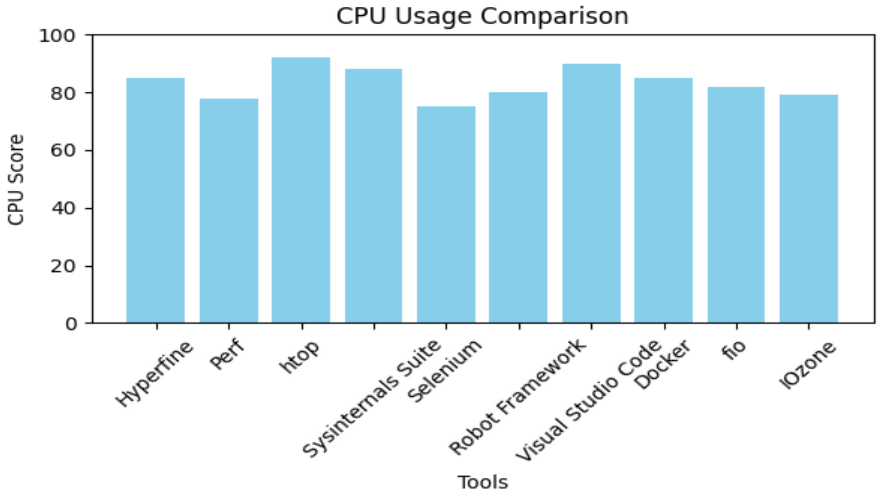

Fig CPU Usage Comparison

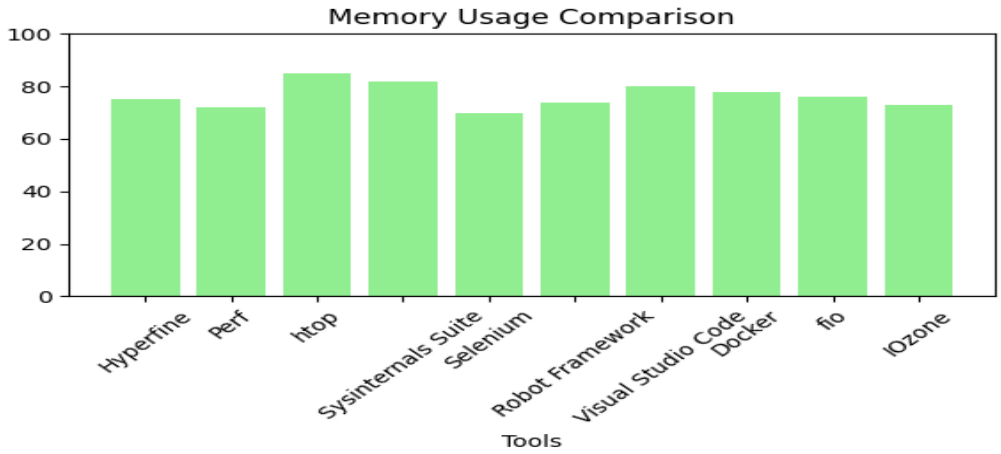

Fig Memory Usage Comparison

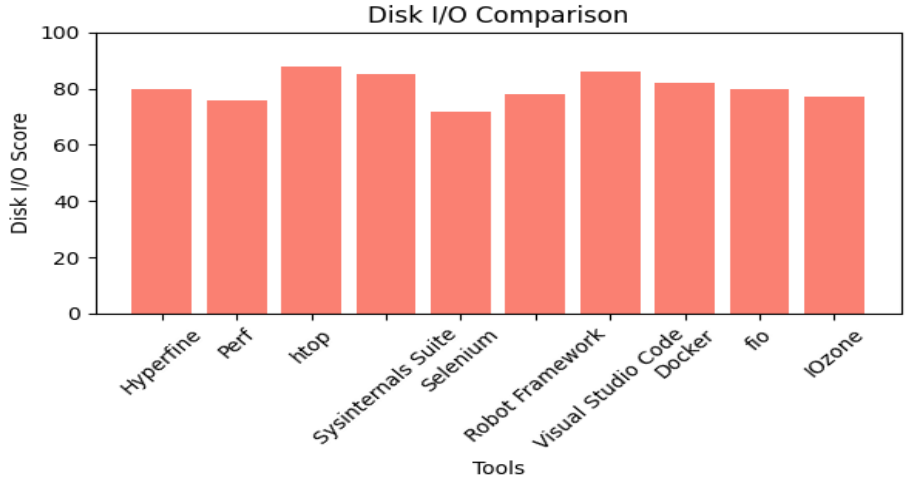

Fig Disk I/O Comparison

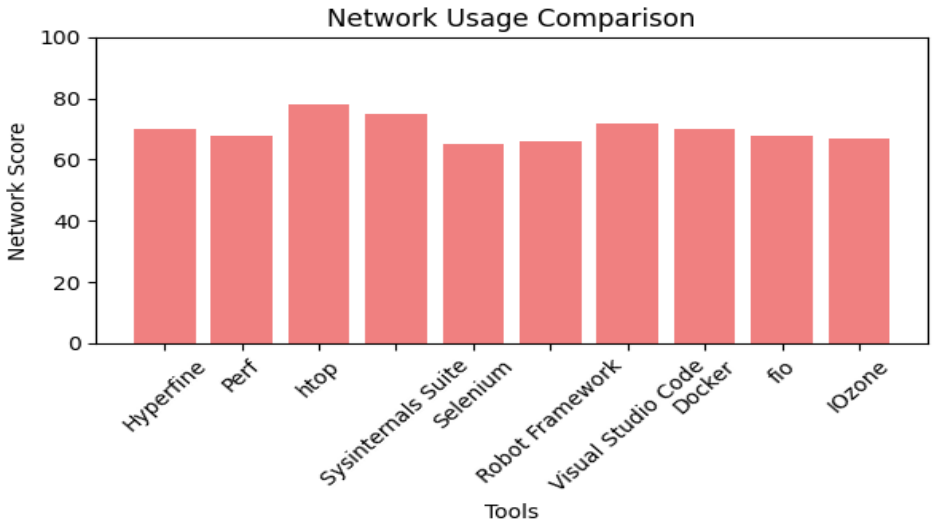

Fig Network Usage Comparison

Output: The script generates four subplots, each showing a comparison of the resource consumption scores (CPU, memory, disk I/O, and network) for the tools. This visual comparison helps assess how each tool performs in terms of resource efficiency across different metrics, providing insights into their relative effectiveness in managing various resources in a simulated scenario. You can adjust the `cpu_scores`, `memory_scores`, `disk_scores`, and `network_scores` to reflect actual or simulated data relevant to the resource consumption metrics you are comparing. This approach allows for flexible visualisation and comparison of tool efficiency in managing Python resources.

6.1 Conclusion

WSL 2.0 represents a significant leap forward in integrating Linux capabilities within the Windows ecosystem. It offers enhanced performance, compatibility, and convenience, making it a powerful tool for developers, along with system administrators, who need to leverage both Windows and Linux environments efficiently. The evolution from WSL 1.0, to WSL 2.0, reflects Microsoft's commitment to providing a powerful and integrated development environment for users who need both Windows and Linux. By addressing the performance and compatibility issues of its predecessor, WSL 2.0, represents a significant advancement, enabling more efficient workflows and a broader range of applications to run natively on Windows. This transformation makes WSL 2.0 an essential tool for modern development and IT operations. This study discusses the use of matplotlib to compare tool performance, efficiency, memory consumption, and resource consumption. The script defines a list of dictionaries (`tools`) where each dictionary represents a tool with a hypothetical `performance_score`. It extracts the tool names and their corresponding performance scores, generates random baseline performance scores, and creates a bar chart comparing the performance scores of each tool against their baseline scores. The output is a bar chart that compares the performance scores of each tool with their baseline scores, providing insights into their relative effectiveness in a simulated scenario. The script also compares the efficiency among tools based on hypothetical efficiency metrics, such as execution time and resource utilisation. It generates a bar chart that

compares the efficiency scores of each tool against their baseline scores, providing insights into their relative effectiveness in a simulated scenario. The script allows for flexible visualisation and comparison of the tool efficiency in Python. The script also compares memory consumption among tools based on hypothetical memory usage metrics such as CPU usage, memory usage, disk I/O, and network usage. It generates four subplots, each showing a comparison of resource consumption scores across these metrics. The output is a bar chart that compares the performance, efficiency, memory, and network scores of each tool. The script allows for adjustments to the `cpu_scores`, `memory_scores`, `disk_scores`, and `network_scores` to reflect actual or simulated data relevant to the tools being compared.

6.2 Future Scope

The future of WSL 2.0, which includes several areas of development and improvement, is promising. These include enhanced performance optimisation, improved integration with the Windows ecosystem, expanded support for graphical applications and desktop environments, and enhanced security measures. Security enhancements will be implemented to mitigate potential vulnerabilities and threats associated with running a Linux kernel in the Windows environment. Containerisation and cloud integration will be further integrated with technologies such as Docker and Kubernetes, enabling the seamless deployment and management of containerised applications within WSL 2.0. Development workflow improvements will be made by providing better support for debugging, profiling, and testing within the WSL environment and by streamlining the setup and configuration of development tools and frameworks. Enterprise adoption and management will be addressed by providing features for centralised management, deployment, and monitoring of WSL 2.0 instances across large-scale deployments. Community collaboration and contribution are encouraged through open-source collaboration, feedback channels, and developer engagement initiatives. This will enable Microsoft to identify emerging use cases, address user needs, and prioritise feature development. Overall, the future of WSL 2.0 is characterized by ongoing innovation, collaboration, and improvement, as Microsoft continues to invest in enhancing the platform's capabilities, performance, and usability to meet the evolving needs of developers, system administrators, and users in diverse computing environments.

REFERENCES

[1] Lewis, N., Case, A., Ali-Gombe, A., & Richard III, G. G. (2018). Memory forensics and Windows subsystem for Linux. Digital Investigation, 26, S3-S11.

[2] Yosifovich, P., Russinovich, M. E., Ionescu, A., Solomon, D. A. (2017). Windows Internals: System architecture, processes, threads, memory management, etc. Part 1. Microsoft Press.

[3] Ionescu, A. (2016). The Linux kernel was hidden using Windows 10.

[4] Ligh, M. (2017). Patches to Volatility for Correctly Parse Pico Process Names.

[5] https://cygwin.com

[6] Baumann, Z., Zill, B., Galen, H., Lorch, J., & Olinsky, R. (2016). Drawbridge.

[7] Porter, D. E., Boyd-Wickizer, S., Howell, J., Olinsky, R., & Hunt, G.C. (2011). Rethinking the library OS from the top down. Proceedings of the sixteenth international conference on Architectural Support for Programming Languages and Operating Systems (pp.) 291-304).

[8] Hammons, J. (2016). Pico Process Overview. Windows Subsystem for Linux.

[9] Turner, R. (2017). Windows Subsystem for Linux Out of Beta! Windows Command Line Tools for Developers.

[10] S. Cooley, S. Hanselman, A. McBee, R. Kottmann, A. Nikoli, "Windows 10 Installlation Guide", 2017

[11] Barnes, H. (2021). Pro Windows Subsystem for Linux (WSL) (pp. 1-321). Apress.

[12] Leeks, S. (2020). Windows Subsystem for Linux 2 (WSL 2) Tips, Tricks, and Techniques: Maximise productivity of your Windows 10 development machine with custom workflows and configurations. Packt Publishing Ltd.

[13] Singh, B. and Gupta, G. (2019). Analysing the Windows subsystem for Linux metadata to detect timestamp forgery. In Advances in Digital Forensics XV: 15th IFIP WG 11.9 International Conference, Orlando, FL, USA, 28 January–29, 2019, Revised Selected Papers, 15 (pp. 159-182). Springer International Publishing.

[14] Thomas, O. (2020). Windows Server, 2019. Microsoft Press.

[15] Gao, Y., Liu, Y., Zhang, H., Li, Z., Zhu, Y., Lin, H., & Yang, M. (2020, November). Estimation of GPU memory consumption of deep-learning models. In Proceedings of the 28th ACM Joint Meeting on the European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 1342-1352).

[16] Blanco, A. F., Bergel, A., and Alcocer, J. P. S. (2022). Software visualisations to analyse memory consumption: A literature review. ACM Computing Surveys (CSUR), 55(1), 1-34.

[17] Barthe, G., Pavlova, M., & Schneider, G. (2005, September). Precise analysis of memory consumption using program logic. At the Third IEEE International Conference on Software Engineering and Formal Methods (SEFM'05) (pp. 86-95). IEEE.

[18] Kotthaus, H., Korb, I., Lang, M., Bischl, B., Rahnenführer, J., & Marwedel, P. (2015). Runtime and memory consumption analyses for machine learning R programs. Journal of Statistical Computation and Simulation, 85(1), 14-29.

[19] Sreelatha, K. and Krishna Reddy, V. (2021). Integrity and memory consumption are related to electronic health record handling in the cloud. Concurrent Engineering, 29(3), 258-265.

[20] Chu, D. H., Jaffar, J., & Maghareh, R. (June 2016). Symbolic execution for memory consumption analysis. In Proceedings of the 17th ACM SIGPLAN/SIGBED Conference on Languages, Compilers, Tools and Theory for Embedded Systems (pp. 62-71).

[21] Singh, P., and Singh, P. (2020). Linux development in WSL. Learn Windows Subsystem for Linux: A Practical Guide for Developers and IT Professionals, 131-168.

[22] https://www.polarsparc.com/xhtml/IntroToWSL2.html