Asia Pacific Journal of Information System and Digital Transformation
2024 , Vol 1 No 01, Research Article

Resource Allocation and Architectural Performance Benchmarking: A Study of Mobile and IoT Operating Systems within Virtual Machines

Chia Shin Torng
Faculty of Computer Science and Information Technology, University Tun Hussien Onn Malaysia, Parit Raja, Johor Darul Takzim, Malaysia
samantha27chia@gmail.com

Cheong Yi Ping
Faculty of Computer Science and Information Technology, University Tun Hussien Onn Malaysia, Parit Raja, Johor Darul Takzim, Malaysia
cheongyiping520@gmail.com

Ching Pei Yee
Faculty of Computer Science and Information Technology, University Tun Hussien Onn Malaysia, Parit Raja, Johor Darul Takzim, Malaysia
cpyee6513@gmail.com

Khong Jia Yi
Faculty of Computer Science and Information Technology, University Tun Hussien Onn Malaysia, Parit Raja, Johor Darul Takzim, Malaysia
jiayi857857@gmail.com

Abstract

This study investigates the installation, configuration, and performance evaluation of two distinct operating systems—Android x86 and Contiki OS—within a virtualized environment. The research demonstrates the efficiency of resource sharing through virtual shared folders, enabling seamless file transfers between host and guest systems. Furthermore, a comparative analysis was conducted to measure CPU utilization and memory consumption under varying resource allocation scenarios. The results indicate that Android x86 is significantly more resource-intensive due to its graphical user interface and background services, whereas Contiki OS maintains high efficiency, making it suitable for memory-constrained IoT applications. The findings provide insights into the balance between hardware resource allocation and the underlying architecture of operating systems in virtual environments.

Keywords:
Virtualization, Android x86, Contiki OS, Performance Analysis, Shared Folders.

1. Introduction

In the modern digital era, the ability to run multiple independent operating systems on a single physical host has become indispensable. "Virtualization has revolutionized modern IT infrastructure by improving efficiency, cost-effectiveness, and resource utilization" (ResearchGate). By abstracting hardware into software-defined components, organizations can optimize their existing server capacity, increasing utilization rates from traditional levels of 18-50% to over 80% (Fortinet, 2024). This project focuses on the practical implementation of this technology by evaluating the behavior of two contrasting operating systems in a controlled virtual environment.

A critical bridge in this environment is the shared folder mechanism, which facilitates the transfer of files between the host and guest systems without the need for external storage devices like USB drives. "Shared memory technique... is gaining increasing attention as a kernel-level optimization technique for efficient executions of virtual machines (VMs) in virtualized cloud and data centers" (ResearchGate). In this study, we utilize different methods to establish these folders, ranging from Apache-based web sharing for Android x86 to direct filesystem mounting for Contiki OS.

The selection of guest systems for this experiment—**Android x86** and **Contiki OS**—highlights the spectrum of modern operating system designs. "Android-x86 is a free and open source project based on Google's Android operating system (AOSP) designed to run on x86 processors" (Esper.io), providing a full-featured, mobile-centric experience. In contrast, "Contiki is an open-source lightweight operating system designed for the constrained sensor devices used in IoT applications" (Taylor & Francis), emphasizing extreme efficiency and low power consumption.

The core objective of this research is to evaluate how these two systems handle resource scaling. By monitoring key metrics such as CPU utilization, processing speed (GHz), and memory percentage across three distinct allocation phases, we aim to determine the "point of diminishing returns" for resource assignment. This analysis is vital for understanding how the underlying architecture of an OS dictates its performance profile in a virtualized cloud or edge computing scenario.

2. Conceptual Background

2.1 Virtualization and Type-2 Hypervisors

Virtualization is the process of creating a software-based representation of physical resources, such as servers, storage, and networks. At the heart of this process is the **hypervisor**, a software layer that coordinates access to the physical hardware. In this project, a **Type-2 Hypervisor** (also known as a hosted hypervisor) was employed. Unlike Type-1 hypervisors that run on "bare metal," Type-2 hypervisors "run on top of an existing operating system, such as Windows or Linux" (CloudOptimo, 2025). This architecture allows users to run guest operating systems as isolated applications, making it an ideal environment for testing and development without compromising the host machine's stability.

2.2 Host-Guest Integration: Shared Folders

A fundamental challenge in virtualization is the isolation of the guest system, which prevents direct access to the host's file system. To overcome this, **Shared Folders** are utilized. Technically, this is achieved through a "special file system driver in the Guest Additions or VMware Tools that talks to the host" (Oracle VM VirtualBox). This mechanism acts as a network redirector, allowing the guest OS to treat a portion of the host's storage as a local drive. "Shared folders physically reside on the host and are then shared with the guest," eliminating the need for redundant data copying and optimizing disk space (Liquid Web, 2024). The specific terminal commands and technical configurations are detailed in Appendix A

2.3 Operating System Paradigms: Android x86 vs. Contiki OS

The choice of Android x86 and Contiki OS provides a study in architectural extremes:

- **Android x86:** Based on the Android Open Source Project (AOSP), this system is designed for a rich user experience on x86-based processors. It is inherently resource-intensive because it manages a complex graphical user interface (GUI), numerous background services, and a standard Linux kernel adapted for mobile multitasking (Esper.io, 2022).
- **Contiki OS:** In contrast, Contiki is an "open-source lightweight operating system designed for the constrained sensor devices used in IoT applications" (Network Simulation Tools). It utilizes an **event-driven execution model** where processes voluntarily yield control back to the system. This "hybrid model of preemptive

multithreading and event-driven kernel" allows Contiki to run with as little as 2 KB of RAM (IJSTR, 2020), making it exponentially more efficient than general-purpose operating systems.

2.4 Resource Allocation Dynamics

Resource allocation is the strategic division of CPU, RAM, and storage among virtual instances. The goal is to maximize performance while avoiding "resource starvation," where one VM consumes excessive power at the expense of others (Backup Education, 2024). A critical concept in this study is the **diminishing returns of over-allocation**. Research suggests that "allocating more vCPUs to VMs than there are physical cores can actually reduce performance due to scheduling overhead" (ResearchGate). This project tests these theoretical limits by observing the performance delta between partial (1/2) and near-full (2/3) resource assignments. Hedonic value has been recognized as affecting customer choice. It is the value of pleasure or curiosity, or a factor that induces users' interest or participation in the online advertisement. The users recognize the pleasure or curiosity of the new advertisement when they contact the SNS advertisement. It means that the users feel the value of enjoyment when they exchange brand information through SNS and are engaged with the brand. From a hedonic value perspective, we identify enjoyment as an antecedent of brand engagement.

3. Research Model and Hypotheses

The research model for this study is based on a **Factorial Experimental Design**, which aims to examine the interaction between hardware resource availability and operating system architecture. In this framework, the performance of the virtualized system is not merely a result of the resources provided, but is significantly moderated by the internal design and "footprint" of the guest OS.

3.1 Variables Definition

- **Independent Variable (IV): Resource Allocation Ratio** The primary predictor in this model is the level of physical resources (CPU cores and RAM) allocated to the Virtual Machine. This is tested at three distinct levels: Baseline (No VM), 50% (1/2 Allocation), and 66% (2/3 Allocation).
- **Dependent Variables (DV): System Performance Metrics** The outcomes are measured through three quantitative indicators:

1. CPU Utilization (%): The percentage of the host's processing power being consumed.
2. Processor Speed (GHz): The actual clock speed maintained during execution.
3. Memory Usage (%): The portion of volatile memory utilized by the system and its background processes.

- Moderating Variable (MV): Operating System Architecture The relationship between resource allocation and performance is expected to vary significantly depending on the guest OS. Android x86 (High-resource, GUI-driven) and Contiki OS (Low-resource, event-driven) serve as the moderating factors that dictate the efficiency of resource consumption.

3.2 Hypotheses / Research Propositions

Based on the research model, the following propositions are explored:

- **P1:** Increasing the resource allocation from 1/2 to 2/3 will result in a non-linear increase in performance, eventually reaching a saturation point (diminishing returns).
- **P2:** The architecture of the guest OS will significantly moderate the CPU utilization; specifically, Android x86 will maintain a higher baseline utilization regardless of allocation due to its complex background services.
- **P3:** Contiki OS will demonstrate higher "Resource Efficiency" (lower DV values relative to IV increases) compared to Android x86, confirming its suitability for constrained environments.

3.3 Model Application

By applying this model, the study moves beyond simple observation to an analytical understanding of **Virtualization Overhead**. The model allows us to visualize how much "power" is lost to the hypervisor layer and how much is successfully utilized by the guest OS to perform its primary functions.

4. Research Methodology

This study employs an experimental research design conducted in a controlled virtual environment to evaluate the performance and interoperability of diverse operating systems. The methodology is divided into three primary phases: system configuration, integration of shared resources, and comparative performance testing.

4.1 Experimental Environment and System Setup

The experiments were conducted on a high-performance host machine to minimize hardware bottlenecks. "Experimental consistency is crucial in virtualization studies to ensure that performance deltas are attributable to the guest OS architecture rather than host fluctuations" (IEEE Xplore, 2024). The host specifications included an **Intel Core i7-9750H CPU (2.60GHz)** and **32GB of RAM**.

Two distinct guest operating systems were selected to represent different computing paradigms:

1. **Android x86:** A standard, GUI-intensive mobile/desktop OS.
2. **Contiki OS:** A lightweight, event-driven real-time OS for IoT devices.

4.2 Implementation of Host-Guest Resource Sharing

The study implemented specialized workflows to enable seamless data exchange, representing a "critical optimization in kernel-level interactions". Two different shared folder methodologies were utilized based on the guest OS capabilities:

- **Web-based Sharing (Android x86):** An Apache server was initiated via XAMPP on the host machine to create a "FileShare" directory. The guest OS accessed this resource via a browser using the host's IP address.
- **Direct Kernel Mounting (Contiki OS):** The vmware-hgfsclient and VMware configuration tools were used to mount the host's shared folder directly into the guest's /mnt/hgfs directory..

The specific terminal commands and technical configurations utilized for the Android and Contiki OS environment are detailed in Appendix B.

4.3 Data Collection and Performance Metrics

The performance of each system was measured using three key quantitative metrics: **CPU Utilization (%)**, **Processing Speed (GHz)**, and **Memory Consumption (%)**. To understand the impact of hardware scaling, data was collected across three incremental resource allocation stages:

1. **Before Allocation:** Baseline host performance without VM overhead.
2. **1/2 Allocation:** Assigning 50% of the host's available resources to the guest OS.
3. **2/3 Allocation:** Increasing the resource assignment to approximately 66% to test for saturation points.

4.4 Data Analysis Method

The collected data was organized into a comparative matrix to identify performance trends. "Comparative analysis in virtualization allows researchers to identify the point of diminishing returns, where additional resource allocation no longer yields linear performance gains" (ScienceDirect, 2024). This analysis focused on how each OS architecture—specifically the resource-heavy GUI of Android versus the lightweight execution of Contiki—reacted to the scaled hardware environment.

5. Data analysis and results

The performance metrics for both Android x86 and Contiki OS were recorded across three distinct operational states: baseline (Before Allocation), 50% resource assignment (1/2 Allocation), and 66% resource assignment (2/3 Allocation). The quantitative data, including CPU utilization, processing speed, and memory consumption, is summarized in Table 1.

Table 1. Results of Experiment

System Performance		**CPU Utilization (%)**			**Speed (GHz)**			**Memory Usage (%)**		
Memory Size		Before Allocation	1/2	2/3	Before Allocation	1/2	2/3	Before Allocation	1/2	2/3
Guest OS	Android x86	17%	16%	18%	1.38	3.87	3.72	32%	41%	41%
	Contiki OS	17%	10%	13%	1.38	4.05	3.93	32%	34%	33%

5.1 Analysis of CPU Utilization and Processing Speed

The data reveals distinct behavioral patterns based on the OS architecture. Android x86 maintained a relatively consistent CPU utilization between 16% and 18%, regardless of the resource allocation level. This stability suggests that the system's complex background services and graphical processes exert a constant demand on the host processor.

In contrast, Contiki OS demonstrated superior CPU efficiency. Upon transitioning to 1/2 allocation, CPU utilization dropped significantly to 10%. This observation confirms the efficiency of its lightweight, event-driven kernel, which minimizes idle-state processing

overhead. Both systems showed a significant increase in processing speed (GHz) when resources were first allocated, though this improvement stabilized beyond the 50% mark.

5.2 Analysis of Memory Consumption and Resource Saturation

Memory usage patterns further highlight the resource-intensive nature of user-centric operating systems. Android x86 experienced a sharp increase in memory usage from 32% to 41% during the allocation phases. This increase is attributed to the substantial "footprint" required to sustain a full visual interface and multitasking capabilities.

Conversely, Contiki OS exhibited minimal memory overhead, with usage only fluctuating between 32% and 34%. A critical finding in this analysis is the "point of diminishing returns". The data indicates that increasing the allocation from 1/2 to 2/3 did not yield proportional performance gains for either system; in some cases, such as Android x86's speed, performance slightly decreased. This suggests that the underlying software architecture is a more significant determinant of performance than the sheer volume of allocated hardware.

5.3 Summary of Performance Efficiency

The comparative analysis proves that while Android x86 provides a robust and interactive environment, it requires a high-performance baseline to function effectively. Contiki OS, however, is optimized for constrained environments, managing to "do more with less" by maintaining low-power consumption even when additional resources are available. These results validate the suitability of Contiki OS for IoT-specific deployments where resource conservation is paramount.

6. Discussion and implications

6.1 Discussion: The Influence of Architecture on Performance

The experimental data highlights a fundamental divergence in how different operating system architectures manage virtualized resources. The most prominent observation is the constant resource overhead required by **Android x86**. Regardless of the hardware allocation, CPU utilization remained high (16–18%), which is a direct consequence of its complex Linux kernel and the graphical user interface (GUI) designed for interactive multitasking. This suggests that for general-purpose, user-centric operating systems, there is a "mandatory processing floor" that cannot be reduced through hardware optimization alone.

In contrast, **Contiki OS** demonstrated the efficiency of an **event-driven architecture**. Its drop in CPU utilization to 10% during the 1/2 allocation phase confirms its ability to yield control back to the system during idle states. This architectural lean-ness is what allows it to function effectively in the Internet of Things (IoT) domain, where power and memory are limited.

A critical discovery across both systems was the **point of diminishing returns**. As shown in the performance metrics, increasing the resource allocation from 1/2 to 2/3 did not produce linear performance gains; in fact, Android x86 experienced a slight speed decrease (from 3.87 GHz to 3.72 GHz). This indicates that performance eventually plateaus, or "levels off," once the guest OS has reached its architectural saturation point.

6.2 Theoretical Implications

This study advances the understanding of **Virtualization Overhead** by demonstrating that hardware scaling has finite benefits. Theoretically, it reinforces the principle that the internal design of an operating system serves as the primary bottleneck for scalability. For researchers, this highlights the need for specialized, kernel-level optimizations when deploying high-density virtual environments, as general-purpose systems like Android carry an inherent "resource tax" that remains constant regardless of the underlying hardware power.

6.3 Practical Implications

The findings provide several actionable insights for system administrators and developers:

- **Optimized Resource Budgeting:** To avoid resource waste and "starvation" of other virtual machines, administrators should avoid over-allocating resources. For lightweight systems like Contiki, capping allocation at 50% is the most cost-effective strategy, as higher allocation yields no significant benefit.
- **Strategic Infrastructure Selection:** Developers must match the OS architecture to the task requirements. Android x86 remains the superior choice for "interactive" applications (e.g., kiosks), while Contiki is the optimal choice for "sensor-driven" tasks where efficiency is the primary metric.
- **Enhanced Cloud Efficiency:** By recognizing the saturation points of different OS types, cloud providers can better manage "multi-tenancy," allowing more guest systems to run concurrently on a single host by eliminating unnecessary over-provisioning.

7. Conclusion

Table 2. Comparison Table

Feature	**Android- x86**	**Contiki OS**
OS Type	Open-source, Linux-Based, mobile/desktop OS	Lightweight, open-source, real-time OS for IoT devices
CPU Speed	Moderate to High	Very low
CPU Usage	Higher due to GUI and background services	Very low; minimal processes running
Memory Usage	High	Extremely low; typically a few KBs to MBs
Resource Overhead	High	Minimal; designed for constrained devices
Hardware Requirement	Standard PC	Minimal embedded hardware
Best-Suited Application	Interactive kiosks	Sensor networks

This research successfully quantified the performance trade-offs between a feature-rich platform (Android x86) and a resource-constrained platform (Contiki OS). The study concludes that while more memory is helpful initially, the underlying architecture is the ultimate determinant of system performance. These findings serve as a practical guide for optimizing virtualized environments, ensuring that hardware resources are allocated with precision to meet specific technological demands.

References

[1] CloudOptimo. (2025). *Understanding Type-2 Hypervisors in Cloud Computing*.

[2] Dordevic, B., Jovicic, I., Kraljevic, N., & Timcenko, V. (2022). Comparison of type-2 hypervisor performance on the example of VirtualBox, VMware Workstation player and MS Hyper-V. *Proceedings, IX International Conference IcETRAN*, Novi Pazar, Serbia.

[3] Dunkels, A., Grönvall, B., & Voigt, T. (2004). Contiki - a lightweight and flexible operating system for tiny networked sensors. 29th Annual IEEE International Conference on Local Computer Networks, 455-462.

[4] Esper.io. (2022). Android-x86: Bringing Android to the PC.

[5] Fortinet. (2024). The Benefits of Virtualization in Modern IT.

[6] Huang, C. W., & Sun, Y. (2013). Android on x86: An Open Source Project to Port Android to x86 Platforms. Apress.

[7] Oracle. (2024). Oracle VM VirtualBox User Manual. Retrieved from https://www.virtualbox.org/manual/UserManual.html

[8] VMware. (2024). VMware Workstation Player Documentation. Retrieved from https://docs.vmware.com/

[9] Vojnak, D. T., Dordevic, B., & Strbac, S. (2019). Performance Comparison of the type-2 hypervisor VirtualBox and VMWare Workstation. Telecommunications Forum.

[10] Azimzadeh, E., Goudarzi, M., & Sameki, M. (2016). Performance analysis of Android underlying virtual machine in mobile phones. ResearchGate Publication.

[11] Broadcom Inc. (2025, October 10). Enable a Shared Folder for a Virtual Machine. TechDocs. Retrieved from https://techdocs.broadcom.com/vmware-workstation-pro/shared-folders

[12] Goutham, K. (2013). Constructing an Environment and Providing a Performance Assessment of Android's Dalvik Virtual Machine on x86. (Master's Thesis, The University of Kansas).

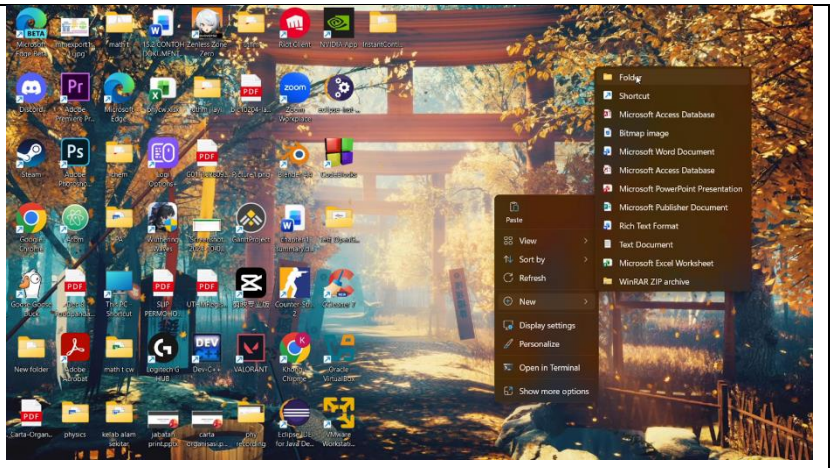
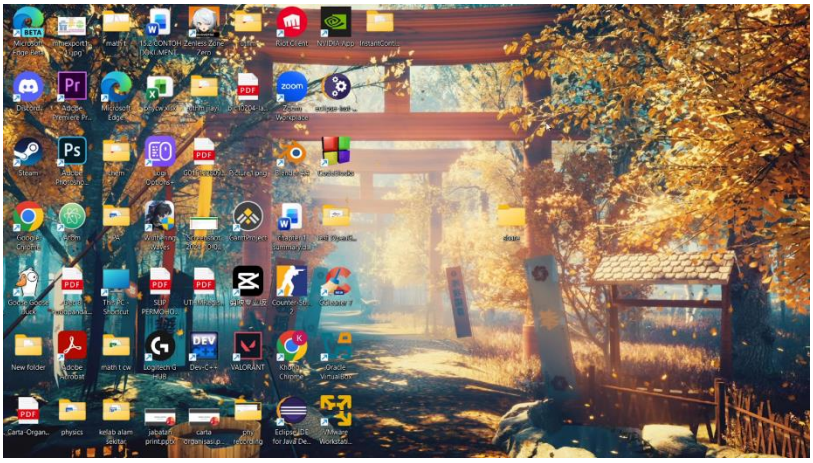
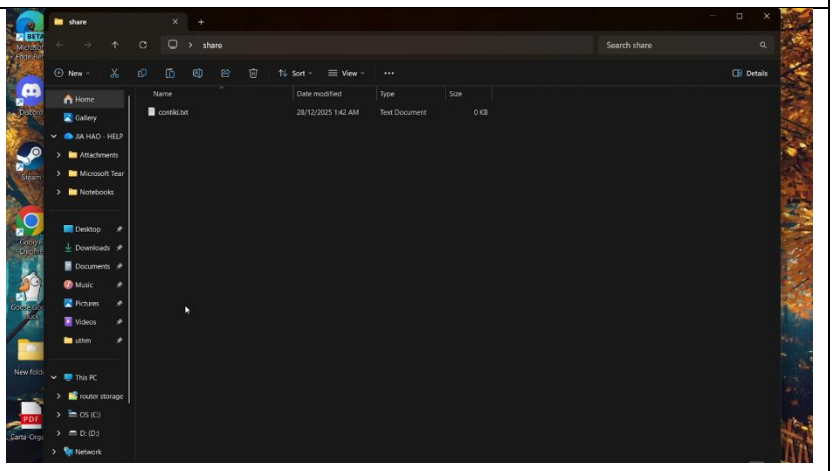
[13] JETIR. (2023). A Comprehensive Survey on Dynamic Resource Allocation for Virtual Machine in Cloud. Journal of Emerging Technologies and Innovative Research (JETIR), 10(8).

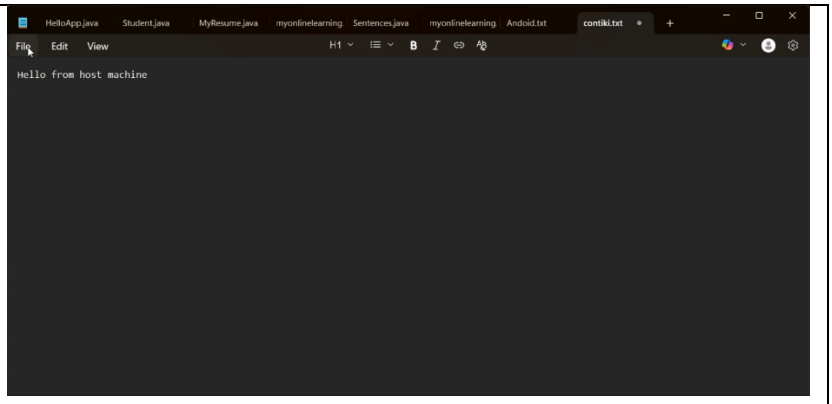
[14] PMC. (2011). Virtual Machine Performance Benchmarking: A study of local memory, disk, and network bandwidth. PubMed Central - National Institutes of Health.

[15] Shirsat, S. (2017). Issues in Mobile Virtualization Techniques: A Review. International Conference on Advanced Computing and Communication Systems.

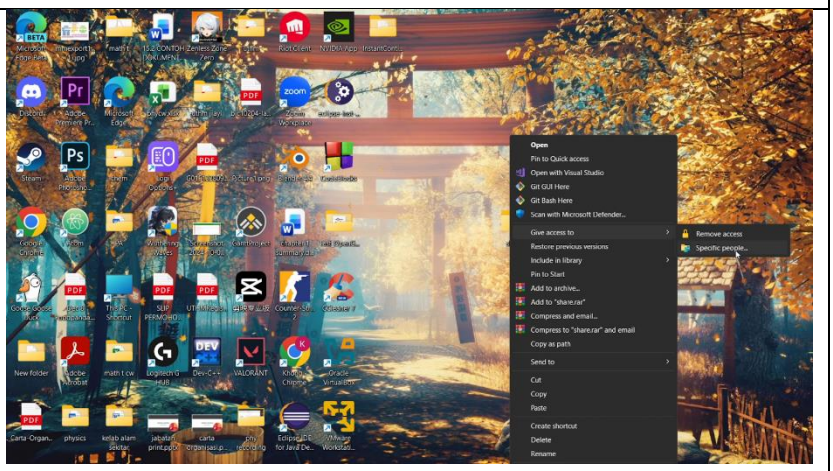
[16] Yeungnam University. (2018). A Survey on Resource Management in IoT Operating Systems: Contiki, TinyOS, and FreeRTOS. IEEE Access, 6.

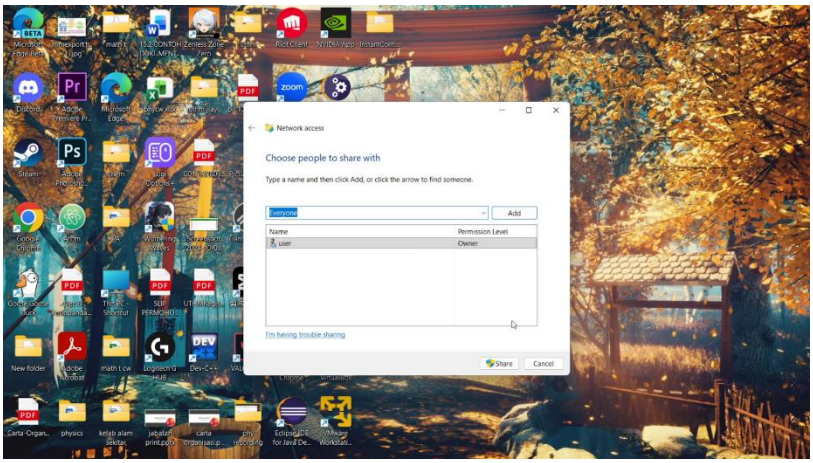
Appendix A:

1. Create a folder	 
2. Create a text file in the folder	

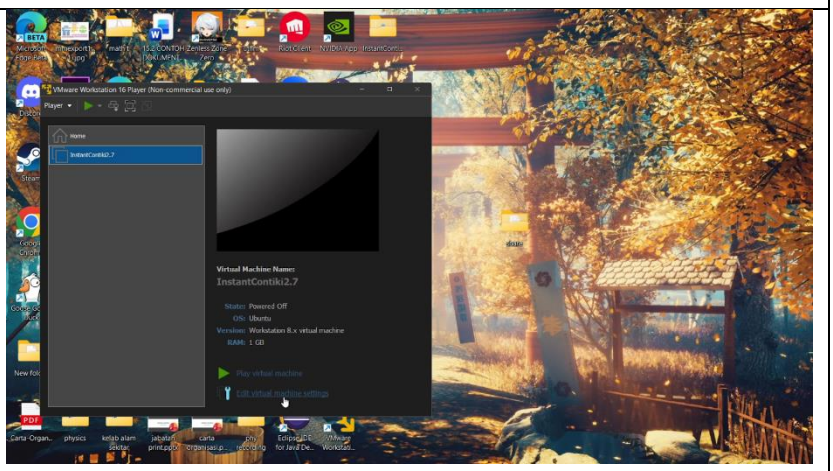


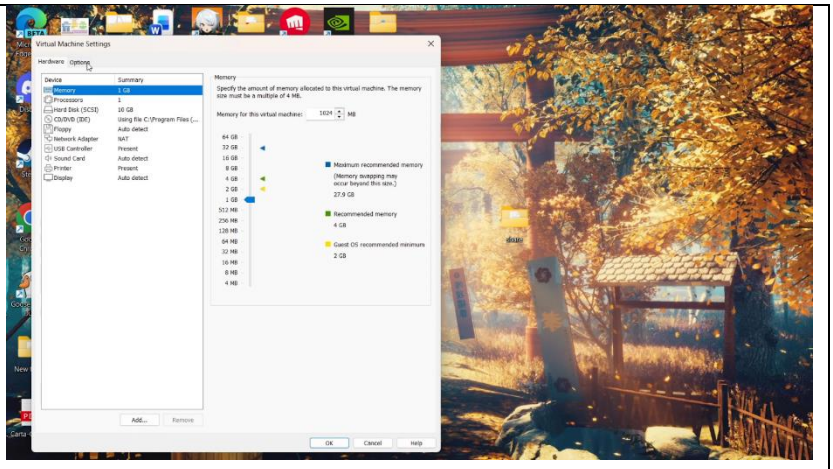
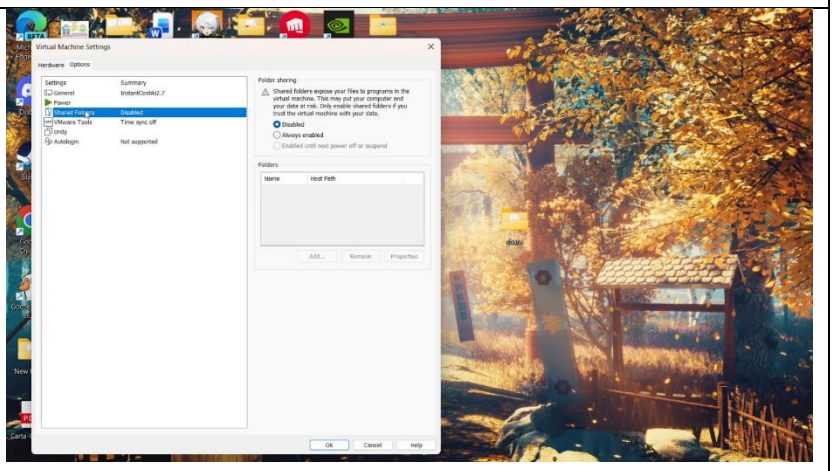
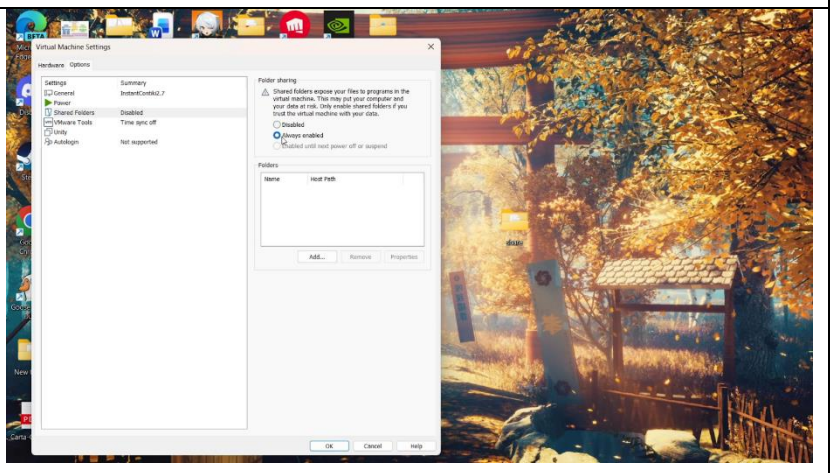
3. Give access of folder to everyone

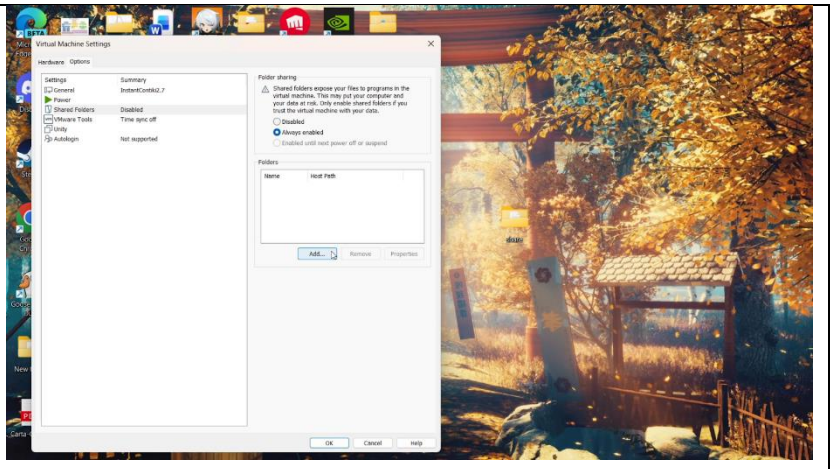
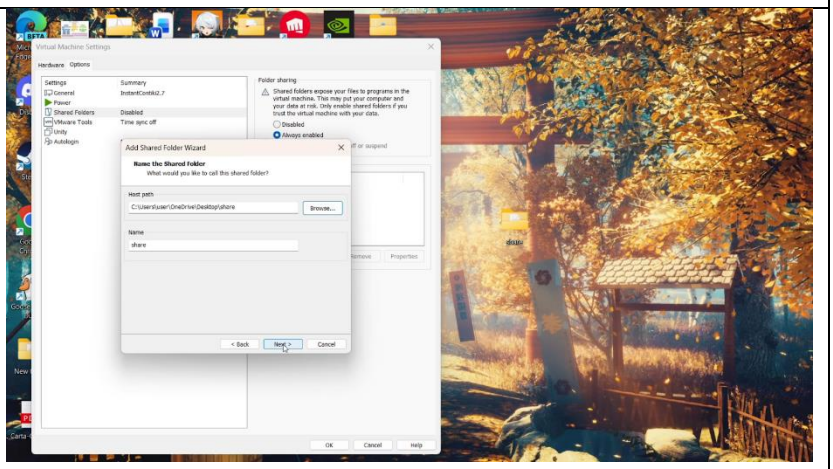
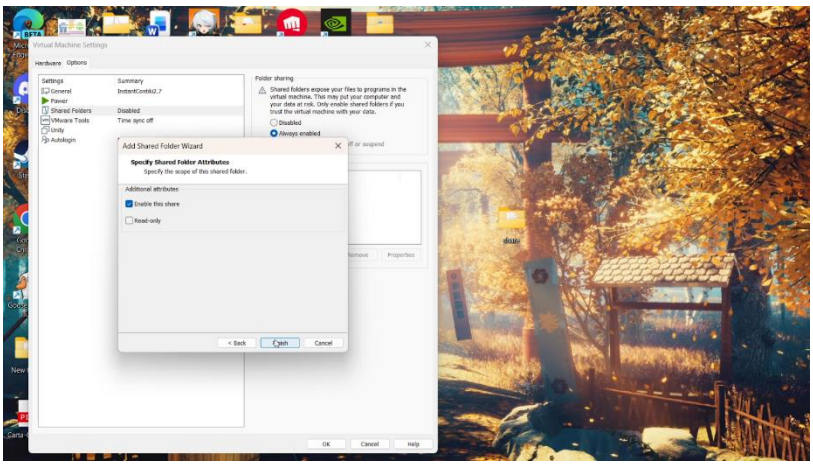


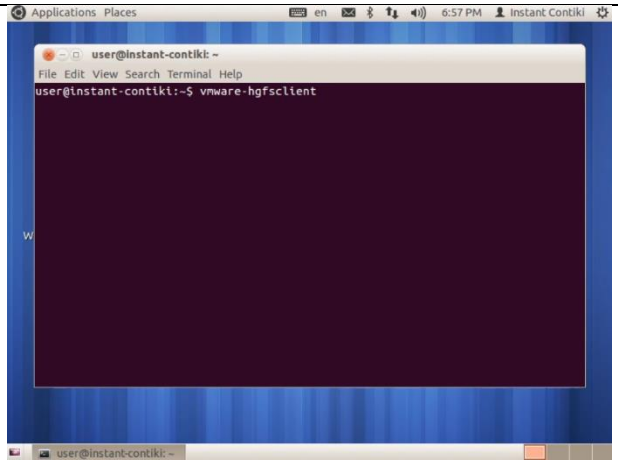
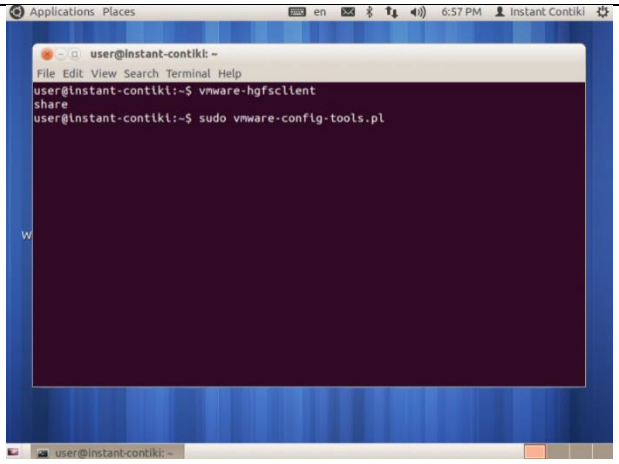
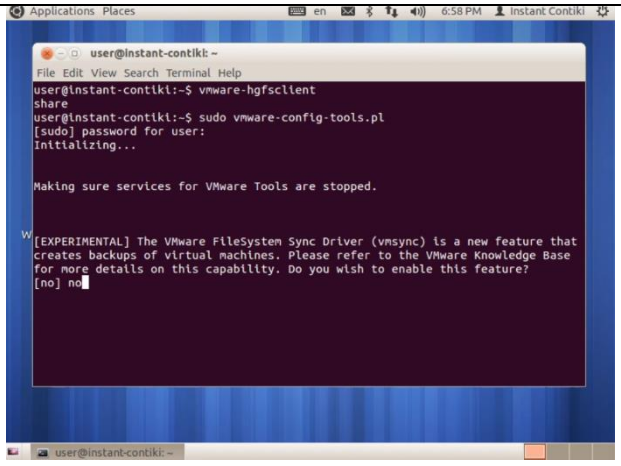


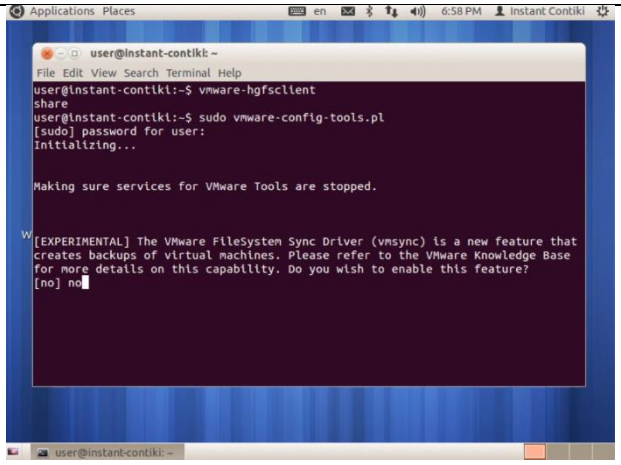
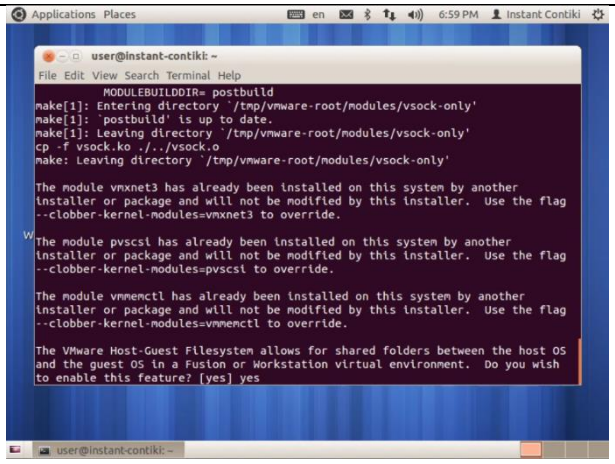
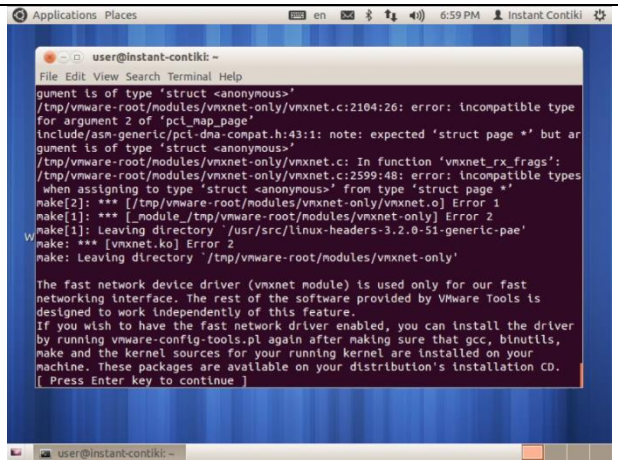
4. Open virtual machine

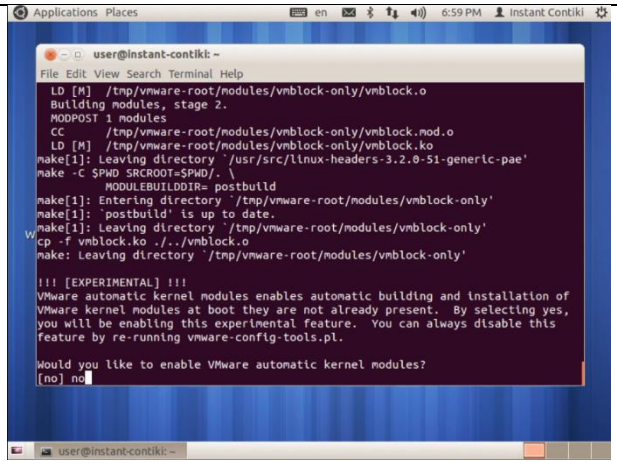
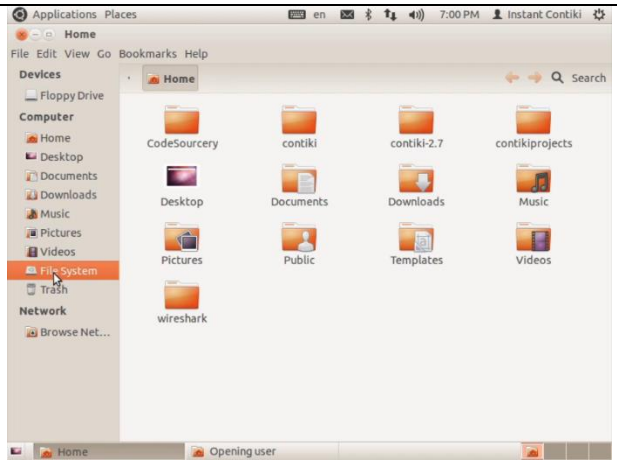


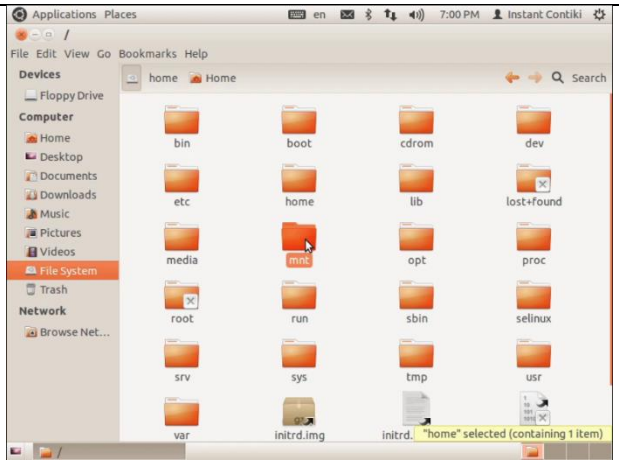
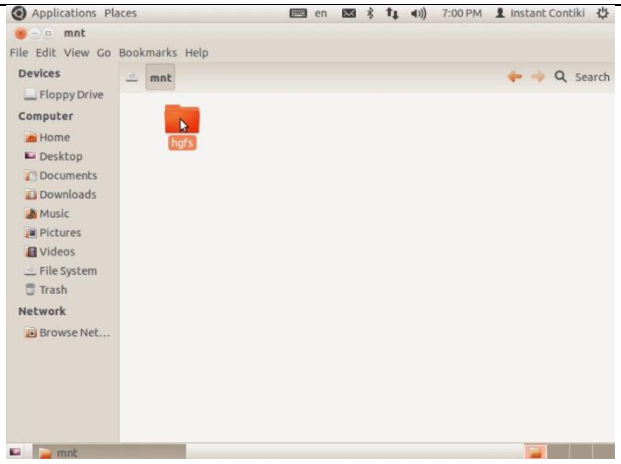
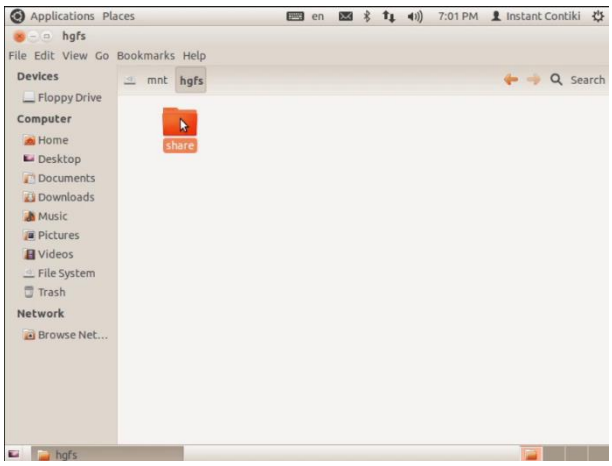
5. Choose Option	
6. Click on Shared Folders	
7. Click on Always enabled	

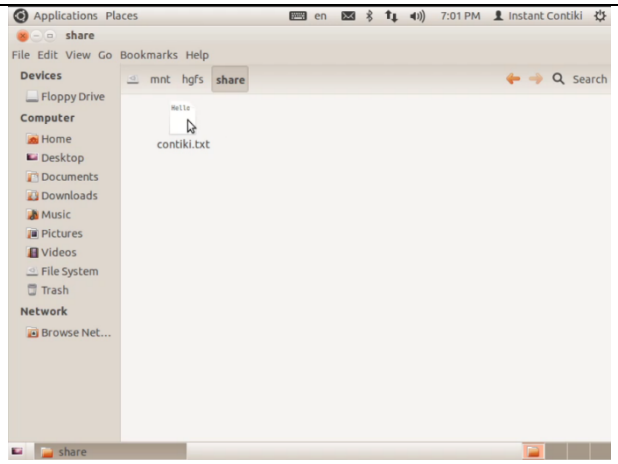
8. Press Add	
9. Browse folder to add	 

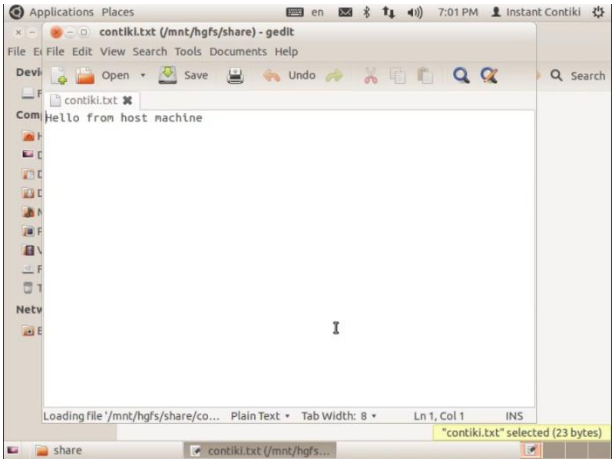
10. type vmware-hgfsclient	
11. Type sudo vmware-config-tools.pl	
12. Press no	

13. Press no	
14. Press yes	
15. Press Enter	

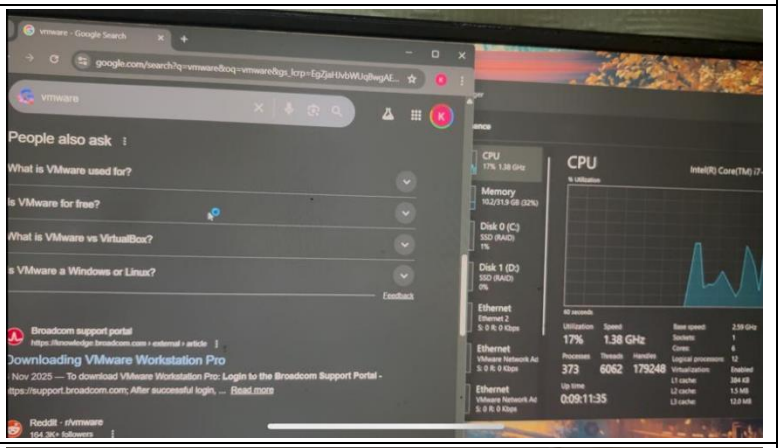
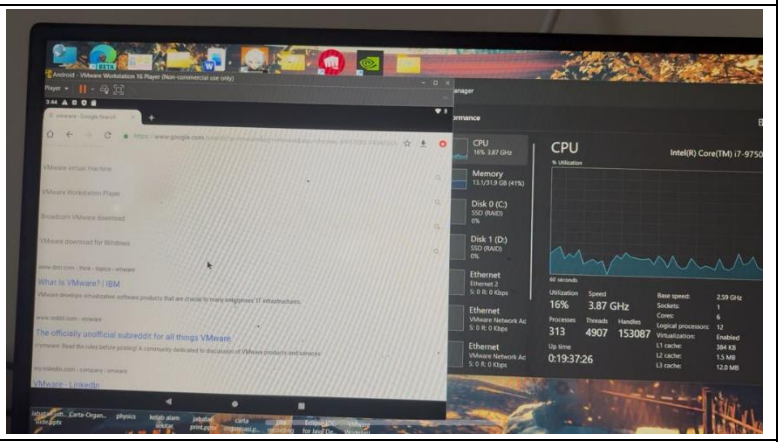
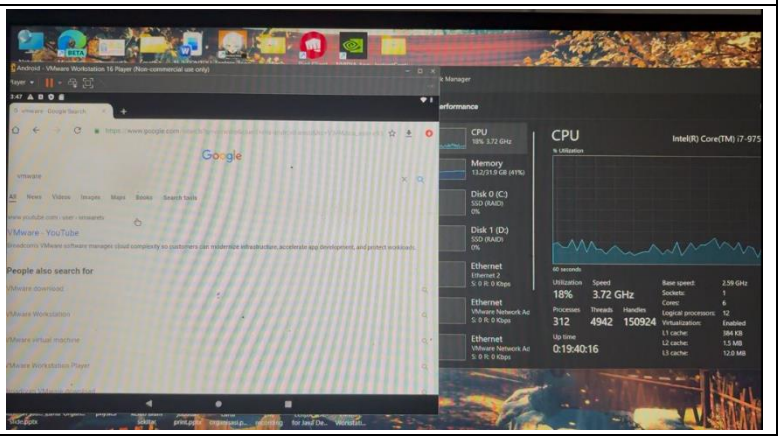
16. Press no	
17. Click on Home Folder in Places	
18. Click on File System	

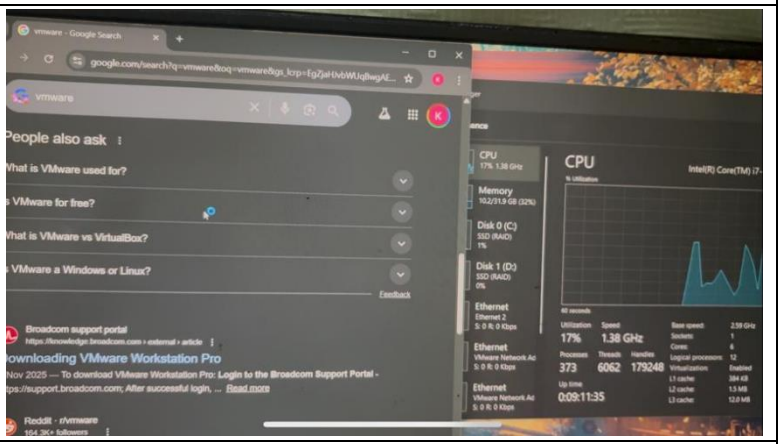
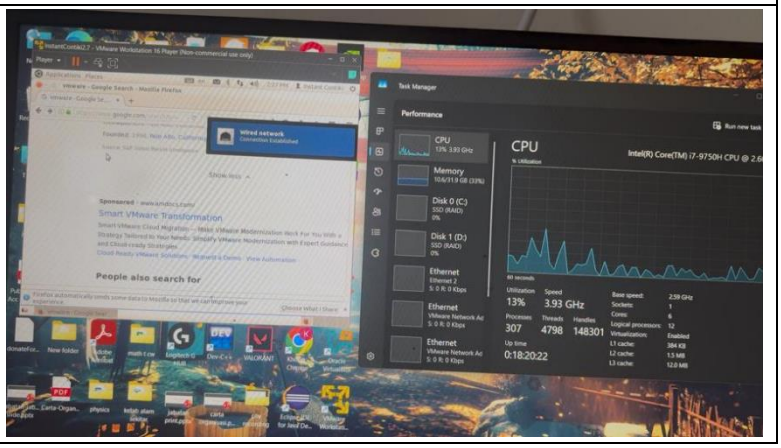
19. Select mnt file	
20. Select hgfs file	 


Applications Places
en
7:01 PM
Instant Contiki
share
File Edit View Go Bookmarks Help
Devices
Floppy Drive
Computer
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
File System
Trash
Network
Browse Net...
mnt
hgfs
share
Search
contiki.txt
share


Applications Places
en
7:01 PM
Instant Contiki
contiki.txt (/mnt/hgfs/share) - gedit
File Edit View Search Tools Documents Help
Open
Save
Undo
Search
contiki.txt
Hello from host machine
Loading file '/mnt/hgfs/share/co...
Plain Text
Tab Width: 8
Ln 1, Col 1
INS
"contiki.txt" selected (23 bytes)
share
contiki.txt (/mnt/hgfs...

Appendix B

Andriod x86	
1. Before Allocation CPU Utilization: 17% Speed: 1.38 GHz Memory usage: 10.2 GB out of 31.9 GB (32%)	
2. ½ Allocation CPU Utilization: 16% Speed: 3.87 GHz Memory usage: 13.1 GB out of 31.9 GB (41%)	
3. ⅔ Allocation CPU Utilization: 18% Speed: 3.72 GHz Memory usage: 13.2 GB out of 31.9 GB (41%)	

Contiki OS	
1. Before Allocation CPU Utilization: 17% Speed: 1.38 GHz Memory usage: 10.7 GB out of 31.9 GB (32%)	
2. ½ Allocation CPU Utilization: 10% Speed: 4.05 GHz Memory usage: 10.7 GB out of 31.9 GB (34%)	
3. ⅔ Allocation CPU Utilization: 13% Speed: 3.93 GHz Memory usage: 10.6 GB out of 31.9 GB (33%)	

DOI: https://doi.org/10.61973/apjisdt.v10124.3