ANKARA YILDIRIM BEYAZIT UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

RAM IMAGE RETRIEVAL IN LINUX USING PROTECTED MODE ARCHITECTURE'S PAGING TECHNIQUE

M.Sc. Thesis by

SEDAT AKTAŞ

Department of Computer Engineering

February, 2023

ANKARA

RAM IMAGE RETRIEVAL IN LINUX USING PROTECTED MODE ARCHITECTURE'S PAGING TECHNIQUE

A Thesis Submitted to the

Graduate School of Natural And Applied Sciences of

Ankara Yildirim Beyazit University

In Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering, Department of Computer Engineering

by

SEDAT AKTAŞ

February, 2023

ANKARA

M.Sc. THESIS EXAMINATION RESULT FORM

We have read the thesis entitled "**RAM IMAGE RETRIEVAL IN LINUX USING PROTECTED MODE ARCHITECTURE'S PAGING TECHNIQUE**" completed by **SEDAT AKTAŞ** under supervision of **PROF. DR. REMZİ YILDIRIM** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

PROF. DR. REMZİ YILDIRIM

Supervisor

DR. MUSTAFA YENİAD

Jury Member

PROF. DR. NURETTİN TOPALOĞLU

Jury Member

PROF. DR. SADETTİN ORHAN

Director

Graduate School of Natural and Applied Sciences

ETHICAL DECLARATION

I hereby declare that, in this thesis which has been prepared in accordance with the Thesis Writing Manual of Graduate School of Natural and Applied Sciences,

- All data, information and documents are obtained in the framework of academic and ethical rules,
- All information, documents and assessments are presented in accordance with scientific ethics and morals,
- All the materials that have been utilized are fully cited and referenced,
- No change has been made on the utilized materials,
- All the works presented are original,

and in any contrary case of above statements, I accept to renounce all my legal rights.

Date: 2023, 21 February **Signature:** _____

Name & Surname: SEDAT AKTAŞ

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to my advisor Prof. Dr. Remzi YILDIRIM for his continued support and encouragement. He supported my ideas during my thesis and spent time with me during challenging times. Without his assistance, I would not have been able to complete my thesis, and I appreciate his help.

Besides my advisor, I would also like to thank Binalyze, the company I work for, and my colleagues for valuable ideas and support for my research work.

And lastly, I would like to thank my family, who have always supported me and been by my side throughout the research process. They have always motivated me throughout my master's education.

February, 2023 **SEDAT AKTAŞ**

RAM IMAGE RETRIEVAL IN LINUX USING PROTECTED MODE ARCHITECTURE'S PAGING TECHNIQUE

ABSTRACT

Collecting evidence from cyber sources in forensic science is critical and essential work. One of these methods is collecting evidence from RAM (Random Access Memory) in digital devices. Since data is temporarily stored in RAM, evidence collection methods such as acquisition for the hard disk may not be sufficient. Therefore, RAM image acquisition techniques are frequently used. In this research, a RAM image was taken from a computer with a Linux operating system. Operating systems are divided into two user space and kernel space, and access from user space to kernel space is subject to certain restrictions. Kernel space has been developed to overcome this limitation. In the research, the protected mode architecture in the Linux operating system was examined, and the paging technique of the architecture was used. Using this technique, a kernel driver has been developed that reaches the addresses of the RAM and saves them to the disk as a file, and this kernel driver has been tested between kernel versions 2.6 and 5.18 and successfully copied to RAM. The tables and access methods used in the paging technique in the operating system are explained in detail. During the research, the Ubuntu 20.04 version operating system, one of the most widely used distributions of Linux, was preferred.

Keywords: Linux, paging, addressing, ram-image, memory dumping, kernel modules, forensic.

KORUMALI MOD MİMARİSİNİN SAYFALAMA TEKNİĞİNDEN YARARLANILARAK LINUX'TA RAM IMAGE ALMA

ÖZ

Forensic biliminde, siber kaynaklardan delil toplamak çok kritik ve önemli bir çalışmadır. Bu delil toplama yöntemlerinden bir tanesi de, dijital cihazlarda bulunan RAM (Random Access Memory)'den delil toplamaktır. RAM'de veriler geçici olarak depolandığı için, sabit disk için uygulanan acquisition gibi delil toplama yöntemleri yeterli olmayabilmektedir. Bundan dolayı RAM image alma teknikleri sıklıkla kullanılmaktadır. Bu çalışmada Linux işletim sistemli bilgisayardan RAM imajı alınmıştır. İşletim sistemleri user space ve kernel space olmak üzere ikiye ayrılmaktadır ve user space'ten kernel space'e ulaşım belirli kısıtlamalara tabidir. Bu kısıtlamayı aşmak için kernel space'te geliştirme yapılmıştır. Araştırmada, Linux işletim sistemindeki korumalı mod mimarisi incelenmiş ve mimarinin sayfalama tekniğinden yararlanılmıştır. Bu tekniği kullanarak RAM'in addreslerine ulaşan ve onları bir dosya olarak diske kaydeden bir kernel sürücüsü geliştirilmiş ve bu kernel sürücüsü, kernel 2.6 ve 5.18 versiyon aralığında test edilmiş ve başarılı bir şekilde RAM kopyalaması yapılmıştır. İşletim sistemindeki sayfalama tekniğinde kullanılan tablolar ve erişim yöntemleri ise ayrıntılı açıklanmıştır. Araştırma süresince Linux'un en yaygın kullanılan dağıtımlarından olan Ubuntu 20.04 versiyonlu işletim sistemi tercih edilmiştir.

Anahtar kelimeler: Linux, sayfalama, adresleme, ram görüntüsü, bellek kopyalama, kernel module, adli bilişim.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

With the introduction of digital systems into all areas of life, cybercrime has increased rapidly. This increase required forensic experts to take precautions for digital environments. This increasing led to the birth of a new science, digital forensics. Forensics, on the other hand, is divided into many branches. Increasing crime rates in digital environments have led to the development of many techniques in digital forensics. One of the techniques developed is memory dumping, which is the main subject of this research and is frequently used in forensics. The memory dumping process participates in crime detection by allowing the processes mapped to memory to be examined. In this research, RAM (Random Access Memory) copy was made by using the Memory Mapping [1] technique included in Intel's Protected Mode Architecture. An LKM (Loadable Kernel Module) has been developed for this copying process. In Figure 1.1, the block structure of the paging technique and 32-bit linear address conversion, which is the main subject of the research, is given:

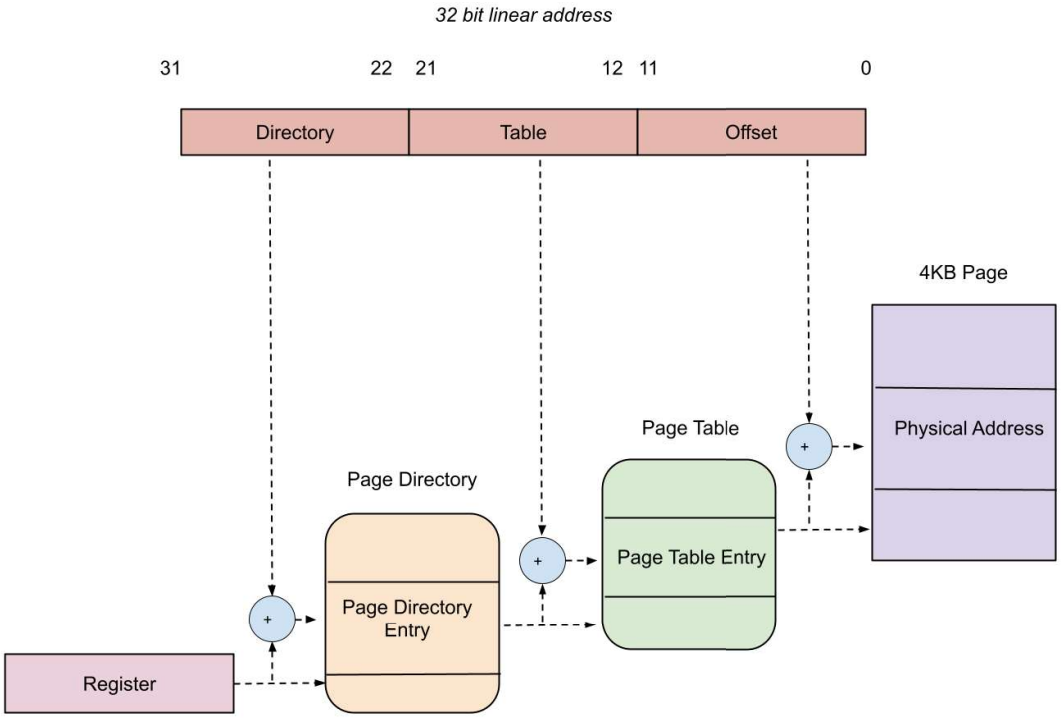

Figure 1.1 Methods of 32-bit linear addressing [2].

The kernel module was developed and tested on a PC which has CPU with 8 cores

and 16GB of RAM. Ubuntu 20.04 was chosen as the operating system. During the development process, C language was preferred because of its suitability for low-level operations. Each operating system has its unique design. The method applied in the research only works on Linux-based operating systems. Since other operating systems have different architectures, different methods must be followed.

1.1 Purpose of Research

This research aims to make a RAM copy using the paging technique of Intel protected mode architecture in the Linux-based Ubuntu 20.04 operating system to keep the working range of the module developed for this comprehensive and to explain the problems encountered, and the methods followed. The applied technique has also been tested in current kernel versions. Tests on new kernel versions revealed whether any changes were required for the new versions. The studies in the literature were scanned, the differences between these studies were revealed, and it was aimed to give an idea to those who want to work in this field. RAM Image acquisition, an important area in forensics science, has been the primary motivation for research as it requires operating system-level improvements.

1.2 Organization Chart of Research

Since the research is in the field of forensics science, in Chapter 1, information about the science of forensics, which is the science in which RAM Image is used, is given. In Chapter 2, the operating system concept and working principles are explained. Detailed information about the history and working logic of the Linux operation, which is the main subject of this research, is given. In Chapter 3, what the Loadable Kernel Module is, how it is developed and how it is loaded into the system are explained. In Chapter 4, explanations were made about the materials and methods used in the research, and schematic representations were included. The research results were discussed in the conclusion, and the problems encountered during the research were included.

1.3 Forensics

Forensic informatics, or criminology, is a science that examines the solution of crimes and cases by tracing them. Forensics is not a field but a discipline combining many

different fields. Today, many branches of science alone are not enough. Therefore, all sciences now have to work in multi-disciplinary. Forensic science also works together with various sciences including medicine, physics, chemistry, computer science, technology, robotics, biology, and geology.

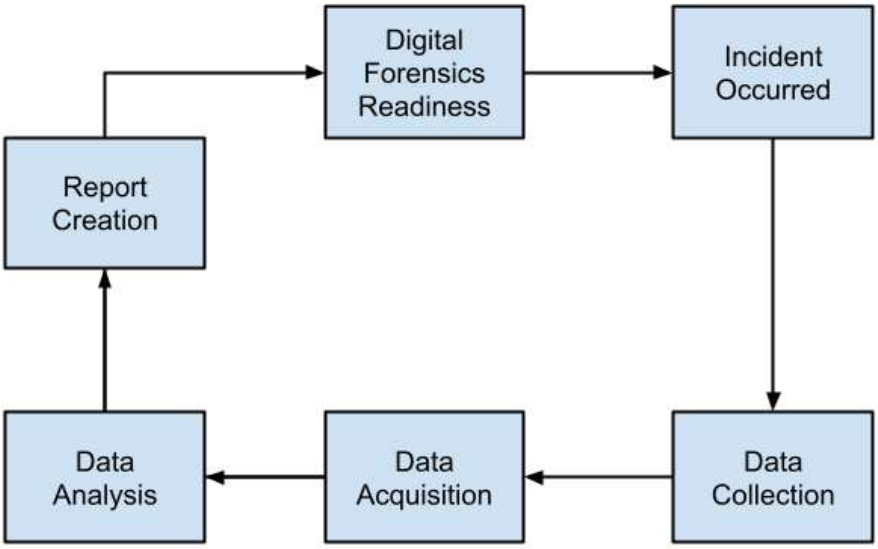

Figure 1.2 Forensics variants and their relationship representation [3].

Forensic informatics is essential in the correct conclusion of cases and the punishment of criminals. Forensic informatics also helps to prevent the innocent from being falsely accused or punished, even if there is no crime. Every piece of evidence taken from the crime scene is treated as a case within the framework of forensic computing. Each of them is followed along with related sciences. Legal evaluations and judgments here are evaluated according to the reports and evaluations of experts in forensic informatics.

1.3.1 Cyber Forensics

Cyber forensics is a science that gathers evidence from cyber sources. These can be computers, mobile phones, and tablets. Evidence is collected and analyzed from these devices by applying techniques developed in Forensics. The methods developed in Forensics are aimed at determining exactly what it finds in an electronic device. With the development and spread of digital systems, this branch of forensics has started to gain significant importance. The vast majority of crimes have slowly started to shift to digital environments. Therefore, detection and analysis methods developed in this field are also changing and diversifying daily. Thus, more reliable evidence is obtained, and a safer investigation is conducted [4].

1.3.1.1 History of Cyber Forensics

The first cybercrime was enacted in the Florida Computer Crime Act in 1978. As the level of computer crimes increased, American state laws were enacted dealing with copyright and privacy. In the 1980s, computer crime was incorporated into federal law. In the following years, countries regulated this crime with their laws. Extensive regulations have been made regarding these crimes in America and Europe. Since 2000, various institutions and organizations have published digital forensics manuals to address the need for standardization, and there is still a broad debate on this issue in these countries.

Forensics is divided into many sub-branches in the field of cyber. The fact that there is a lot of hardware to be examined in computers has created many sub-branches for forensics. The types of cybercrime we can encounter at any time in our daily life have become a sub-branch of forensics. For example, the email method, frequently used in daily life, is one of the places to collect forensic evidence.

Disks used for storage in computers and memory used temporarily are critical data sources for forensic reviewers. In addition, telephones and the networks used can be given as examples. Telephones are subject to inspection methods, just like computers. On the other hand, networks are subject to different methods to follow the traces left by network surfing.

1.3.2 Processes of Digital Forensics

Some stages should be followed during the evidence-collection process. These stages are the standards that must be followed for forensics science. These can be summarized as follows:

- *"Identification":* To differentiate according to the purpose of the research.
- *"Preservation":* Taking action to protect data.
- *"Analysis":* Using the necessary tools to extract appropriate data.
- *"Documentation":* Collecting and documenting results.

- *"Presentation":* Presenting the results found and actions taken.

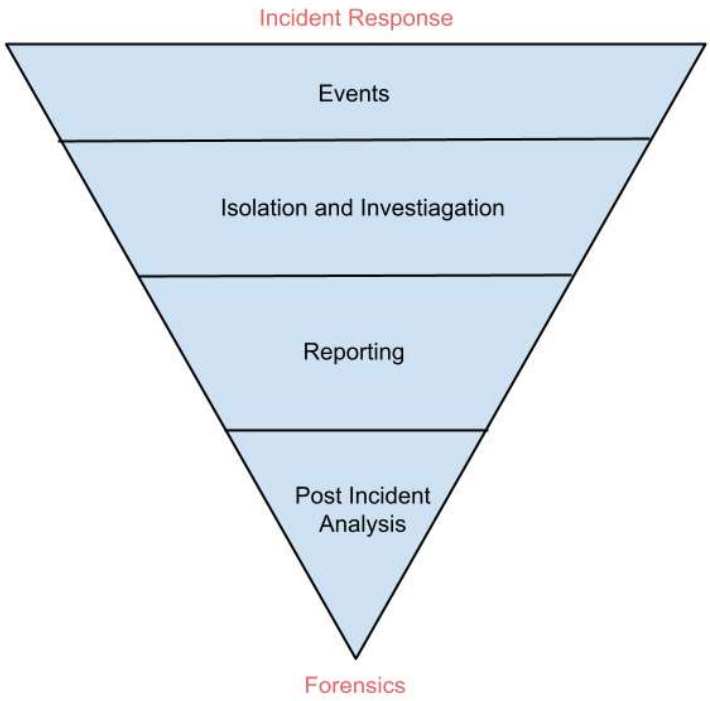

Figure 1.3 Forensics to incident response diagram [5].

1.3.3 DFIR (Digital Forensics and Incident Response)

The science of digital forensics collects data from digital systems. However, data collection alone may not be sufficient. Therefore, in case of any attack on cyber systems, techniques have been developed to detect this and take action accordingly. The science that combines these two methods is called DFIR. With this science, intervention can be made at the time of the incident and detected after the incident. This reduces the attack space.

1.3.4 Memory Forensics

Memory forensics is a sub-branch of digital forensics, one of the branches of forensics science. Studies on memory in the forensic field are gaining importance day by day. In addition to collecting data from the disk, information about applications mapped to RAM is also obtained. All running applications allocate memory by using processes at runtime and are mapped to memory on processes. Accessing this information instantly falls within the scope of memory forensics.

The most critical techniques known in this field are acquisition and memory dumping. The acquisition is the process of scanning the entire memory instantly. Allows scanning of applications currently running in RAM. Memory dumping is the current picture of RAM.

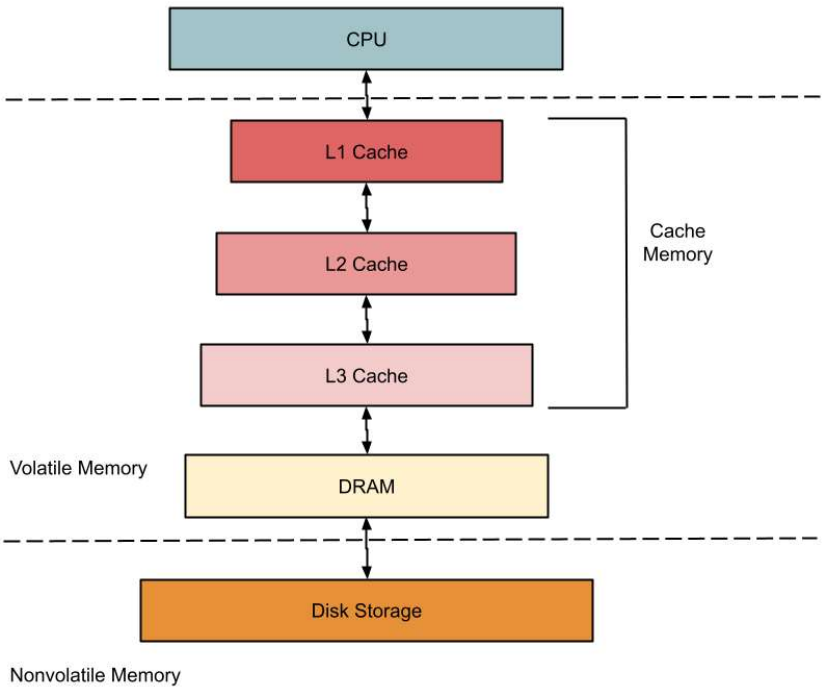

Figure 1.4 Volatile and nonvolatile memory diagram [6].

1.3.4.1 Volatile Data

Temporary data; These are data that are difficult to detect, capture, and work with. Because they are data lost when the computer is turned off, it must be captured and analyzed before the computer is shut down or it flies out of memory. Therefore, forensic techniques on memory are more complicated than operations on disk. Temporary data has an important place in every incident response. If critical operations are performed on a compromised device, it is sometimes preferable to turn off the device directly. Like all applications, malicious applications are mapped onto memory. Therefore, for them to be detected, it is necessary to have information on the RAM. Capturing and analyzing temporary data has an important place in the detection of malware.

1.3.4.2 Linux RAM Extraction

The memory dumping process can be likened to taking a snapshot of RAM from an open device. Different scenarios may arise when a cyber security incident is intervened. For example, the machine to be intervened may have lost contact with SIEM (Security Information And Event Management). In such a case, before capturing the data, the device is verified as malware-free and reverse-engineered the malware using special tools.

Obtaining a disk image from the device can take a long time. Many problems can be encountered when transferring an image, which can be 100 GB in size, to a scannable location. While the server hard drive can be over 100GB, the device's RAM is much smaller, typically between 16GB and 32GB. This is why RAM image is preferred over disk image. Dumping RAM from the device is much faster and takes less space. Common approaches are to image the hard disk and scan the RAM dump for IOCs (intrusion indicators), prioritizing RAM over complex disk images when triggering events.

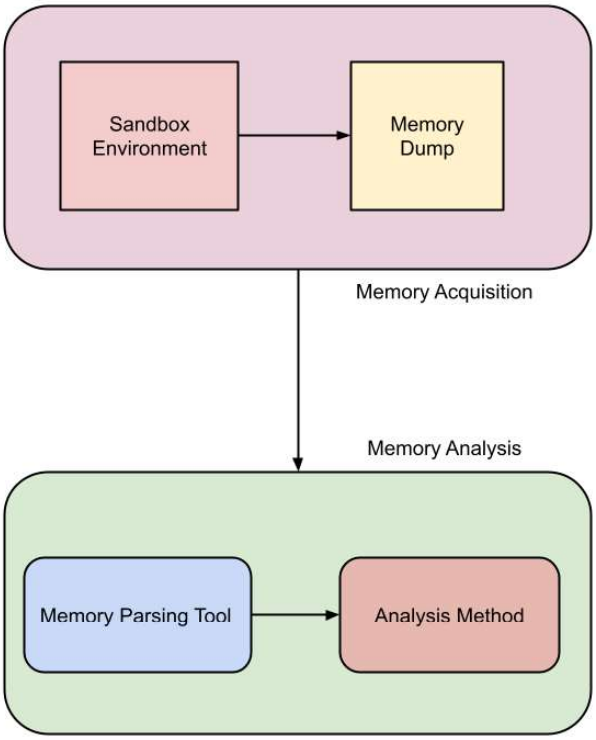

Figure 1.5 Memory dumping and analyzing diagram [7].

1.3.5 Memory Forensics Tools

Many tools were developed in the field of Memory Forensics [8]. These are tools that can be found both commercially and open-sourced. These tools include analysis methods such as acquisition and triage. In addition, software is available that combines specialized open-source software for scanning and discovery, such as Yara and Sigma. Sigma and Yara is a software tool that detects malware. Using them together increases the probability of success.

1.3.6 Literature Review

The most commercially successful product in the RAM Image technique is the LIME [9] kernel module. This research aims to develop and test a LIME-like module in the 2.6 to 5.18 kernel version ranges. It's explained on Lime's github page as follows: "A Loadable Kernel Module (LKM) which allows for volatile memory acquisition from Linux and Linux-based devices, such as Android. This makes LiME unique as it is the first tool that allows for full memory captures on Android devices. " Lime usage:

```
"insmod ./lime.ko path=<outfile | format=<raw|padded|lime>"
```

Different techniques are applied for Linux, Windows and MacOS. It is easier to get a ram image in Windows operating systems than in macOS and Linux operating systems. The kernel of the Linux operating system undergoes many changes. This creates a problem for backward compatibility. Therefore, for the developed tools to work in a wide range of versions, they must be tested very well and adapted to the kernel version changes. Also, having many distros is another challenge. Differences between distros also complicate development. In terms of malware, this unstable situation also makes it difficult for malware software. Therefore, antimalware applications are less common than Windows.

On the other hand, because Windows was developed by a single company and is widely used, backward attaches importance to compatibility. Applications can run without being affected by many version changes. Accessing Windows kernel APIs is the same as accessing Linux kernel APIs. They enable documentation and access. For

macOS, the situation is different. MacOS is not an operating system as documented and accessible to others as Linux and Windows. Access to kernel APIs is restricted.

There are studies and tools related to RAM Image in Windows operating system. Commercial tools can be used as well as step-by-step instructions in academic studies [10]. Here's how to use WinPMem [11], a multi-platform memory acquisition tool:

```
"winpmem_mini_x64.exe physmem.raw"
```

In case of any crash in Windows already, it allows complete dump of memory with start and recovery settings [12]. Another tool is the tool developed by CyberTest [13]. This tool works on both 32 bit and 64 bit. The last of the commonly used tools developed in this field for Windows is the tool developed by Belkasoft [14]. It can both do live capture with its acquisition feature and can do memory dumping.

We mentioned that there are software and hardware restrictions for macOS. That's why MacOS doesn't have as many tools as in Windows and Linux. The tool used in macOS is called OSXPMem [15]. This tool, on the other hand, works on Intel-based Macs, as can be seen in the summary section, but does not work on ARM-based Macs: "The OSX Memory Imager is an open source tool to acquire physical memory on an Intel based Mac." With OSXPMem, both memory acquire and memory dump can be done.

There have been various changes in Linux kernel versions over time. These changes have affected both the software in the upper layers and the low-level processes. These changes have always been less in user space than in kernel space. For example, forensic memory tools had to change when the kernel version was changed from 2.4 to 2.6. A thesis study conducted in Perl in 2006 explained how to use kernel macros [16] according to this change. However, some changes in the kernel versions in the meantime caused this study to be limited at specific points. In this research, it has been observed that the kernel module works successfully within the specified 4 kernel version ranges.

The tool AVML (Acquire Volatile Memory for Linux) [17] developed by Microsoft has

been affected by kernel version updates. This tool tries to access RAM from 3 sources:

- "/dev/crash"
- "/proc/kcore"
- "dev/mem"

2 of these 3 resources were affected by the kernel version changes and are no longer usable. As stated in the tool's Readme.md page, the tool tries to access all 3 sources in order and tries to pull data from the available source. However, there is a possibility that the last source will be closed in the new kernel version. In this case, this tool will be unavailable for this job.

In addition to Lime and AVML, a few different tools and working principles should be mentioned. The first of these is "memdump". "memdump" is a command that comes with the operating system, used in Unix-based operating systems. This command reads the "/dev/mem" file like AVML tool and prints its output to the screen. Ubuntu's official page provides the following information about the command: "This program dumps system memory to the standard output stream, skipping over holes in memory maps. By default, the program dumps the contents of physical memory (/dev/mem)." Installation:

```
"## Debian / ubuntu Linux ##
sudo apt-get install memdump

## FreeBSD ##
pkg_add -r -v memdupm"
```

Since this tool reads the /dev/mem file, processes mapped by the kernel are missing. Usage of "memdump" [18]:

```
"memdump [-kv] [-b buffer_size] [-d dump_size]
```

```
[-m map_file] [-p page_size]"
```

There are also tools that work using the memory acquisition technique. The most well-known of these tools is the "Volatility" [19] tool that works on both Windows, Linux and macOS. This tool successfully performs the acquisition process. It is widely used in the field of fornesics. One of the most well-known features of this tool is that it can analyze dumped memory files. It allows meaningful searches according to the operating system in files created in raw or lime format. It can be said that Volatility does not use the Ram Image technique because it performs acquisition and does not provide a dumping. While Volatility Foundation used the Python2 version in their first release, they have recently released Volatility3 as a beta version using the Python3 version with the work done. On volatility's github site, it says the following about volatility: "The Volatility Framework is a completely open collection of tools, implemented in Python under the GNU General Public License, for the extraction of digital artifacts from volatile memory (RAM) samples."It works on Windows versions 7, XP, 8, 10, Server 2003, Server 2008 and server 2016. It also works on 32-bit and 64-bit Linux kernels 2.6.11 to 5.5 and Ubuntu, Debian, CentOS, Fedora and OpenSuse. Volatility community has also created many plugins for this tool [20]. Usage of volatility3:

```
python vol.py imageinfo -f aybu_test.lime
```

Developed exclusively for Linux systems, "volatilitux" [21] is a simplified version of the Volatility tool for Linux. It serves to analyze physical dump files and tested with these operating systems: Android 2.1, Fedora 5 and 8, Debian 5, CentOS 5, Ubuntu 10.10 with and without PAE.

Another command that runs on Linux version 2.2 and uses "mmap" is the memfetch command. The official github page mentions memfetch's work as follows: "Memfetch is a very simple utility that can be used to dump process memory of any user space process running on the system without affecting its execution." [22] There is no output feature in Lime format, and since it is not loaded as a kernel module, access to kernel processes is restricted.

Another tool is the crash-utility tool created by the Redhat community. This tool can be used in Redhat and Fedora versions of Linux. This restriction means that the tool can be used on rpm-based Linux versions, but not on debian-based operating systems such as Ubuntu. There is a description about the tool on github: "The core analysis suite is a self-contained tool that can be used to investigate either live systems, kernel core dumps created from dump creation facilities such as kdump, kvmdump, xendump, the netdump and diskdump packages offered by Red Hat, the LKCD kernel patch, the mcore kernel patch created by Mission Critical Linux, as well as other formats created by manufacturer-specific firmware." [23] While it is a good feature of the tool that it works as of kernel version 2.2, it is a big shortcoming that it only works on rpm-based operating systems. Installation:

```
"## RHEL / CentOS ##
yum install crash

## Novell / Suse / OpenSUSE ##
zypper install yast2-kdump"
```

Besides dumping and acquisition techniques, there are also tools that use the grepping technique. These tools do what the grep command in Linux does in processes. In this respect, it can be said that it is closer to the acquisition method by leaving the dumping method. One of these tools is memgrep. The description of the command is as follows: "memgrep is a grep for /proc/pid/mem. It's licensed under the MIT license." [24] Usage:

```
"memgrep [FLAGS] [OPTIONS] [--] [regex]"
```

For those who work in forensic science, the event does not end with memory dumping. The file resulting from the dumping of the memory is not a file that can be opened and read. Some tools are needed to read RAM Image files. These tools support certain file formats. These file formats are:

Raw file format is the most used file format without headers, metadata or magic values. All modern analysis tools support this format. The other file format is Windows crash dump format [25]. It is the dump file format created by Windows when Windows crashes. This file format is divided into several sub-branches for itself. These can be listed as small [26], kernel and complete [27]. The Complete file format contains everything in memory. Kernel file format contains only memory information in kernel space. Expert Witness Format (ewf) is a format used by EnCase and is not as widely used as other formats [28].

When it is desired to copy the memory of a virtual machine, in this case, a file with the ".vmem" extension can be created using the dumpit tool. The instantaneous state of the virtual machine can be paused and resumed in another environment. This file format can be analyzed by EnCase.

The last file format is the ".dmp" file format. It is called MemoryDump or MiniDump file format. Used primarily by Microsoft. It also contains information about drivers and kernels. The file can be read with Bulk Extractor.

Other utilities are also needed to read these file formats. These utilities are:

Volatility tool is the most used in this utility. Since it has the ability to do acquisition about Volatility, it was mentioned in the tools section. After Volatility, the most used analysis tool is FireEye [29]. In addition to having many features, it also offers the opportunity to analyze memory dump files. Sans Sift, which became available for Ubuntu packages in 2014, is an important tool in Forensics science for the Linux community. The HxD [30] tool, on the other hand, is a tool that only works on the Windows operating system and allows to open the raw memory dump file as hex, examine and modify it.

There are more analysis tools in Linux and Windows than in the MacOS operating system, as in memory dumping tools. The limitations in analysis tools for the MacOS operating system also continue. While the most used MacOS tool is volatility, Volafox [31] is also among the analysis tools used. These tools allow to read the resulting file

after the ram image is taken. Therefore, the existence of many analysis tools and their comparisons contribute to forensic science.

CHAPTER 2

LINUX OPERATING SYSTEM

2.1 Operating System

The operating system is a layer that provides communication between the hardware and the user. In the hardware part, there are physical chips, cards, displays, and electromechanical items such as disks, keyboards, and printers. Hardware alone is limited in its usefulness. Software is needed to introduce the hardware to the system. The components created by this software that serve a specific purpose and that are meaningful are called programs. Programs control hardware, and each operating system comprises organized programs.

An operating system takes specific inputs, compiles them, and produces results. Every operating system consists of two main layers. One of them is user space, and the other is kernel space. Access to kernel space from user space is restricted. The operating system uses software to facilitate communication there. Software is developed with algorithms. Operating systems also work by taking advantage of these algorithms. Each program consists of two main layers. One is input; the other is output. Computers also work based on input and output. Therefore, the operating systems work by this input-output principle.

The hardware involved in the input/output processes has its unique designs. For example, reading a disk block with data from the drive known to be held in which block is an operation that must be done in the same way by all application programs. The operations performed have a fixed structure, and all parts working in that system must comply. Therefore, programs that do the same job have different codes from different operating systems. These programs provide convenience to the user. Thanks to these programs, users can focus only on what they need to do without knowing what kind of system is running in the background.

The operating system dominates the hardware components. Another operating system feature is the ease of use by providing an interface to users. Hardware components such

as processors, main memory, and input/output units that make up a computer system also participate in the computer system's operation. Each has its unique structure and design. An operating system works in harmony with all these hardware devices.

In each operation made from the user or kernel space, hardware and software are used. For example, a user developing a program using a high-level language needs a text editor to write the program and a compiler to convert his program to machine language. These are all conveniences for writing and using programs. All these programs, made available to the user in the user space, are also under the operating system's control.

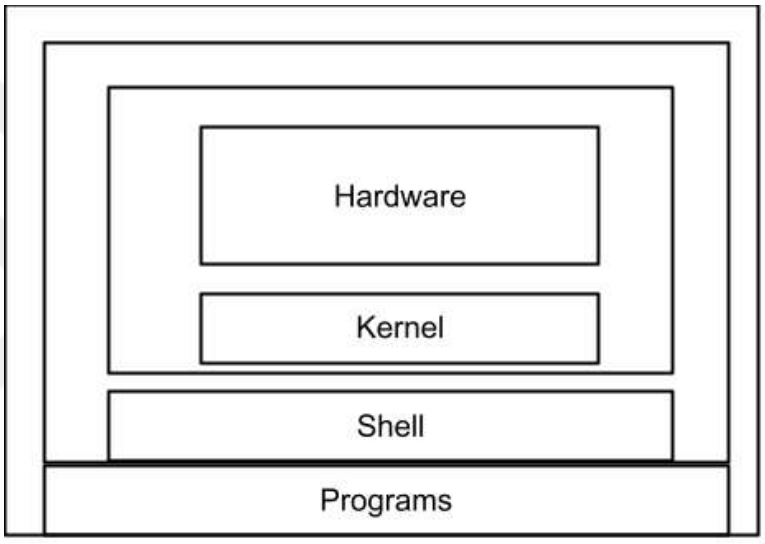

Figure 2.1 Schematic representation of the layers of the operating system [32].

Another function of operating systems is ensuring system resources' efficient use. Users prefer a fast but resource-intensive system. They expect the program to give the most accurate result as soon as possible. Nevertheless, resources are limited, and not every running program can run and terminate with the fastest runtime. In this case, the operating system tries to optimize the resources and allocate them. Efficient resource use requires allocating resources among the user as much as possible, keeping idle times to a minimum. In this way, optimizations create possibilities to produce more and more efficient services with the same number of resources. However, when examined in terms of programs, it is seen that this approach reduces operating speeds due to resource waiting. Therefore, optimization can be considered one of the most challenging functions of operating systems.

With the opening of the BIOS(Basic Input Output System), operating systems start to work by allocating memory space for themselves. Together with the BIOS, they install themselves into the system. Operating systems contain functions for processes that match the hardware, and they facilitate the work of users through these ready-made functions.

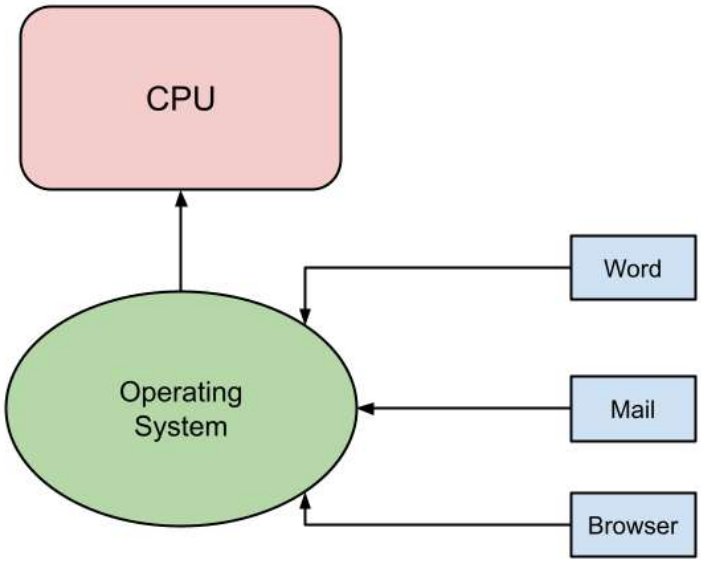

Figure 2.2 The relation of the operating system to the CPU and external software [33].

2.1.1 Functions of the Operating System

We can divide the essential functions of operating systems into two resource sharing and transferring those working on the computer to the screen:

Resource sharing; It is the process of sharing resources on the computer between users and programs. Examples of shared resources are processors, memory, I/O units, and data. Increasing optimization in resource sharing; helps to increase resource availability and utilization rates.

Functions [34]:

- Allows system calls by defining an interface to the user.
- It undertakes the function of regulating hardware sharing and usage in multi-user systems. This allows a possible race condition to be edited.

- Allows users to share data.
- Regulates input-output operations.

Image transfer; It is an example of operating systems communicating with the hardware. Even if multiple users use the computer simultaneously, it appears to be using it alone. In order to perform these operations, control of I/O units, memory, file system, and programs is required. Input-output units require hardware-like programming.

2.1.2 Operating System Types

Operating systems have been divided into different types over time. Emerging computer science and industry are the main actors in this differentiation. Since not every operating system can be used for the same job, engineers have designed unique operating systems for specific jobs. Operating systems can be divided into types according to resource usage, processing types, and timer usage patterns. Some of these are:

2.1.2.1 Time-Sharing Operating System

People from many regions can simultaneously utilize a specific computer system thanks to time-sharing operating systems. Multiple users simultaneously share the processor's time in these systems. The CPU on these systems runs a large number of useful apps. The CPU runs each user application in a brief window of time. These systems allocate a modest amount of time for each user program using CPU scheduling and multiprogramming. The system reacts fast to commands given by the user.

Shared Time Operating systems have a lot of benefits, like cutting down on CPU wait time. Users can acquire prints immediately since the response time is short. Additionally, it avoids program duplication. A time-sharing operating system has some restrictions, including security challenges and data transfer problems [35].

2.1.2.2 Real-Time Operating System

Compared to conventional operating systems, these operating system types allow functions to run safely at a defined time and order. Due to the functions being in a defined time and order, It is used in defense, aviation, space, automotive, nuclear

power plants, weapon systems, missile systems, air traffic control systems, and many more critical sectors.

2.1.2.3 Batch Operating System

This type of operating system runs a large number of I/O-intensive tasks. It has three main functions:

- It can handle multiple jobs.
- Responds to incoming requests in large numbers of small units. The reservation system in airlines can be given as an example.
- Has time-sharing capability. Multiple remotely connected users can run jobs on the system. An example is a database system.

In MS-DOS operating systems, it is written to a ".bat" file and processed collectively by the operating system.

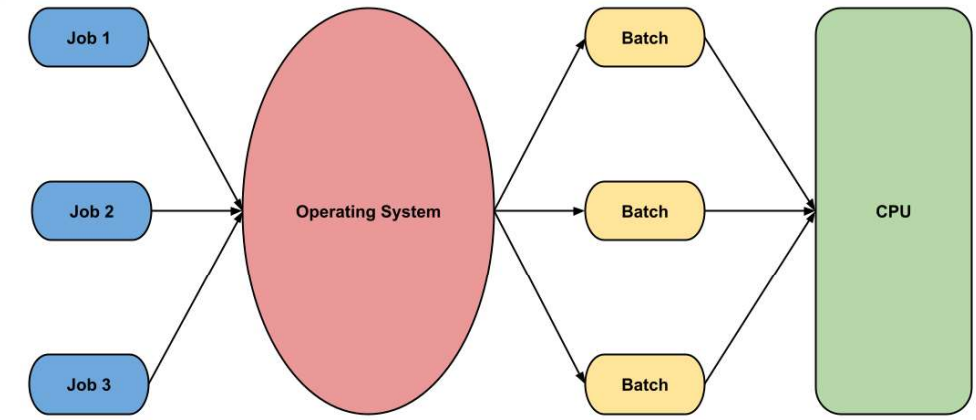

Figure 2.3 Schematic representation of batch systems [36].

2.1.2.4 Distributed Operating System

In distributed operating systems, multiple machines behave as a single, integrated machine. Unlike network operating systems, distributed operating systems users are unaware of the presence of more than one computer in the environment. Examples include Amoeba, Plan 9, Chorus, and Mungi. It can be preferred in processes that require high CPU and GPU, such as artificial intelligence and image processing.

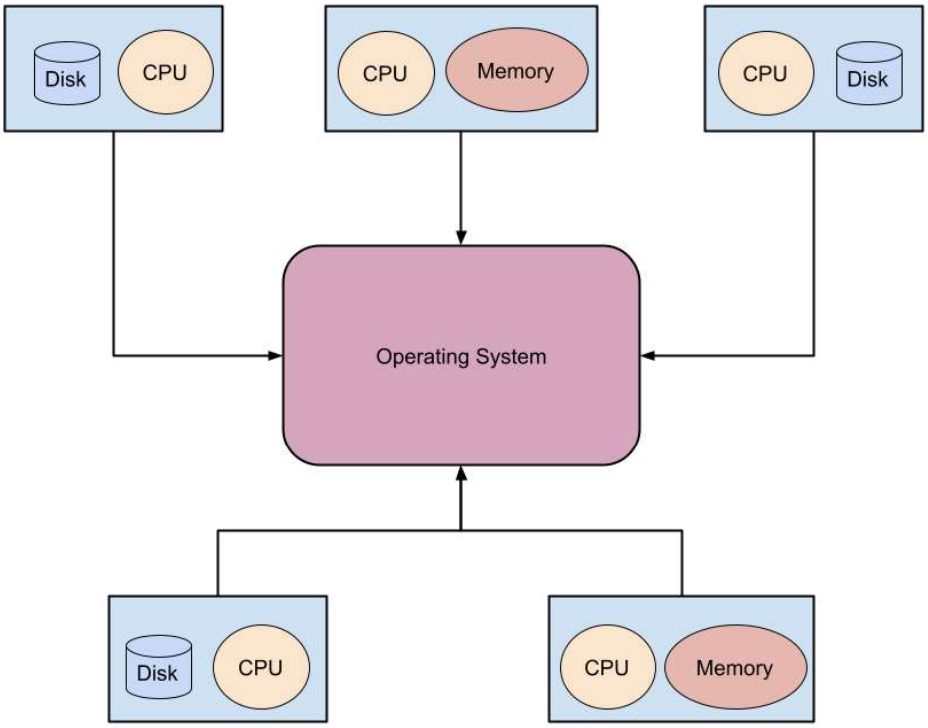

Figure 2.4 Schematic representation of distributed systems [37].

2.2 Linux

Operating systems are divided into two open-source and closed-source codes. Linux is considered among the open-source operating systems. It was developed by evolving on UNIX. Unlike Windows or macOS operating systems, it was developed by independent and non-commercial individuals, not by a commercial vendor. This has allowed many features to be easily added and removed from Linux. However, this also has negative consequences, such as the differentiation of UNIX applications over time and the fact that it is increasingly difficult to write applications that run on all UNIX systems [38].

2.2.1 History

In the late 1960s, the Multics project began to be developed at MIT (Massachusetts Institute of Technology) with the joint initiative of General Electric, AT&T (American Telephone and Telegraph Company) Bell Laboratories. This project was a project where the foundations of a new operating system were laid. As a result of these studies, Ken Thompson developed the first UNIX operating system in 1969 at AT&T Bell laboratories. The UNIX name is based on the MULTICS (Plural Information and Computing Service) [39] project. The basis of the Linux operating system is UNIX,

and today it is the basis of Linux and some other operating systems.

In 1970, Dennis Ritchie developed the C programming language, so UNIX was recompiled in C. C programming language; This language is preferred in an operating system kernel because it can directly access the hardware with program instructions. In the following years, UNIX was recompiled with minor changes with a structure known as the standard POSIX (Portable Operating System Interface for Computer Environments). It has gained a structure that can work on systems with small and different architectures.

All these improvements have allowed other UNIX-based operating systems to be tested. These include operating systems such as BSD, Xenix, and NetWare. As a college student in 1991, Linus Torvalds came up with the idea of a machine-based operating system that anyone could use in their home. Torvalds developed Linux, a UNIX-based operating system, with the MINIX(Mini UNIX) program. It attracted great attention worldwide, and many improvements were made with the support of everyone.

2.2.1.1 Different Linux Versions

The term Linux, contrary to popular belief, is the name of a kernel, not an operating system, and was developed by Linus Torvalds. Different Linux versions have emerged because it is open source, its source codes are accessible to everyone, and it allows anyone to create systems that do customized work for the job they want. Due to this opportunity, various organizations have developed their versions of Linux and presented them to users over the internet. The most commonly used Linux versions can be listed as follows: Redhat, Debian, Ubuntu, Centos, Suse, Kali, Pardus, and ArchLinux.

2.2.1.2 POSIX (Portable Operating System Interface for Unix) Standards

POSIX is a project developed by Richard Stallman in 1985. Because there are so many Linux distributions, there is also a differentiation between kernels. Since each developer can make changes in different mixes of Linux according to his needs, backward compatibility may not be fully ensured. An application that worked in a previous Linux kernel version may sometimes not work in a newly released Linux kernel version. To eliminate these problems, some standards have been established

that everyone tries to comply with at important points.

2.2.2 Processes

Processes are the basic building blocks of operating systems. They can be considered specialized workers who receive, process, and terminate a transaction. Every program needs processes that the operating system allocates to it to run. Programs are customized according to needs. Therefore, programs can use more than one process or a single process. The operating system keeps the information about processes in unique data structures. It allocates resources and memory to them and terminates them when the time comes. When a process terminates, all the resources it uses are released and made available for reuse by other processes. Figure 2.5 is a schematic representation of the states of these processes. In the Linux operating system, the communication between the process and the operating system is via signals.

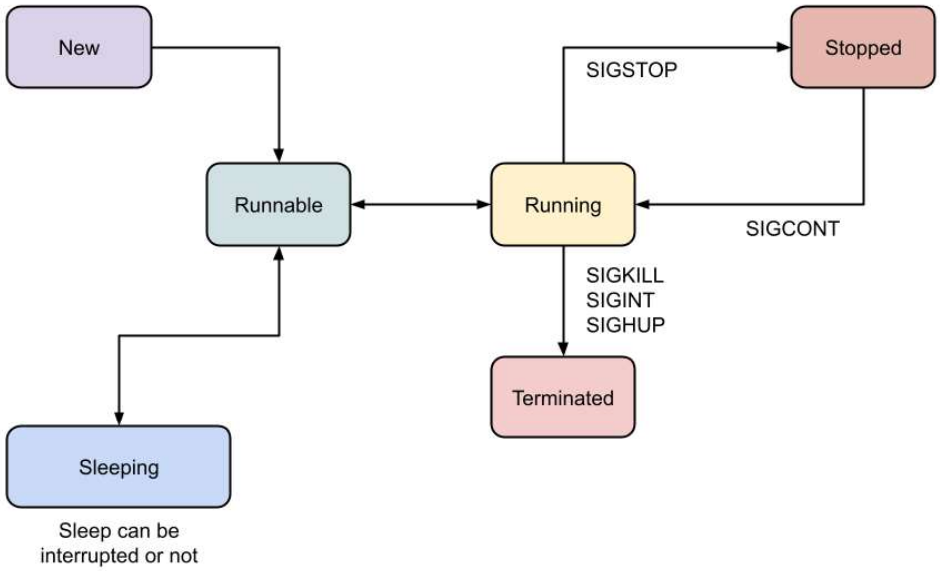

Figure 2.5 Schematic representation of the life cycle of processes [40].

A few signals used as examples:

- *"SIGSTOP"*: Stops the process.
- *"SIGKILL"*: Terminates the process and frees all resources allocated to the process.

- *"SIGCONT"*: It is a command that allows a process that has been stopped with the "SIGSTOP" command to continue running.
- *"SIGINT"*: Sends an interrupt to the process.

Other resources, such as CPU, may be made available to other processes, but all resources must be shared fairly. Otherwise, a process accesses all resources and prevents others from using it. This is known as deadlock.

The "ps" command is used to view the running processes in Linux. Here are a few options for this command:

- "aux": "ps -aux" command set is used to see all processes in the system. When the command is used as "ps -aux" with aux parameters, it shows all processes and process owners running on the system and terminals.
- "-u [user]": Example command usage showing running processes by username: ps -u [user]
- "-p [int]": It gives process information according to the process number.

Another command where processes can be viewed is the pstree command. pstree command for hierarchical display of processes available. This command can also be used as the tree in the Windows operating system.

Another command is "pgrep". It allows running processes to be sorted according to specific criteria. To make changes to the process, the process to be changed must first be found. Here the "pgrep" command also gives the needed running process numbers: "pid". The process number is the number that enables the processes to communicate in the system and is assigned to be unique for all processes.

In addition to the commands that allow processes to be displayed, some commands allow us to operate with processes. Example of these commands is the "kill" command. The "kill" command is used to kill processes. Processes can be terminated with

their names or process ids with this command. Sometimes processes do not respond to these commands. That is why force-like helper commands are implemented in these commands. For the "`kill`" command, the force parameter is -9.

2.2.3 Kernel

The kernel takes over brain function in an operating system. It acts as a bridge between the driver and the user layer and forms the basis of computer systems. User mode and kernel mode are found in modern processor architectures' designs. It is made to prevent users from accessing the kernel directly. The kernel has access to the driver and user layers. Figure 2.6 shows the location of the kernel between the hardware and the user layer.

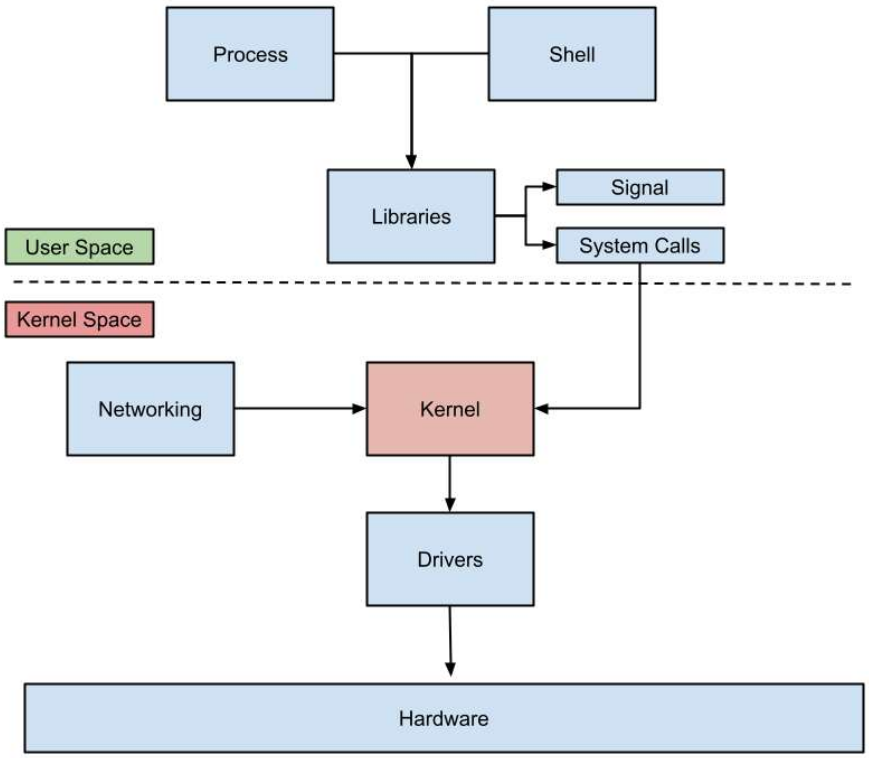

Figure 2.6 Kernel's hardware and user layer access scheme [41].

The kernel has the authority to schedule processes. It has the knowledge of all running processes, keeps this information within itself, and constantly updates it according to the change. As seen in Figure 2.7, the communication between peripheral elements is done through the kernel. Kernel responds to a request from a process, can create new processes, or terminate an already-standing process. The kernel also maps each process's virtual memory to the computer's physical memory. The usage authority

of the disk swap area is also in the kernel, and it provides communication between input-output devices.

System calls; are means by which programs interact with the operating system. A system call is made when a computer program sends a request to the operating system kernel. As can be seen in Figure 2.6, the kernel can be accessed directly through system calls. Shell or processes use system calls through libraries provided by the operating system. A shell can be thought of as a program. It is designed as a command interpreter. It receives the commands entered by the user, reads them, and runs the necessary programs. The shell's function is to pass the inputs from the user to the kernel as an intermediary layer between the user and the kernel. What to do in the system changes depending on the shell's flexible structure, that is, its functions. In this context, it can be thought that Shell is one of the keys to dominating the system. Different developers develop multiple shell programs. However, bash is the most preferred shell program due to its capabilities.

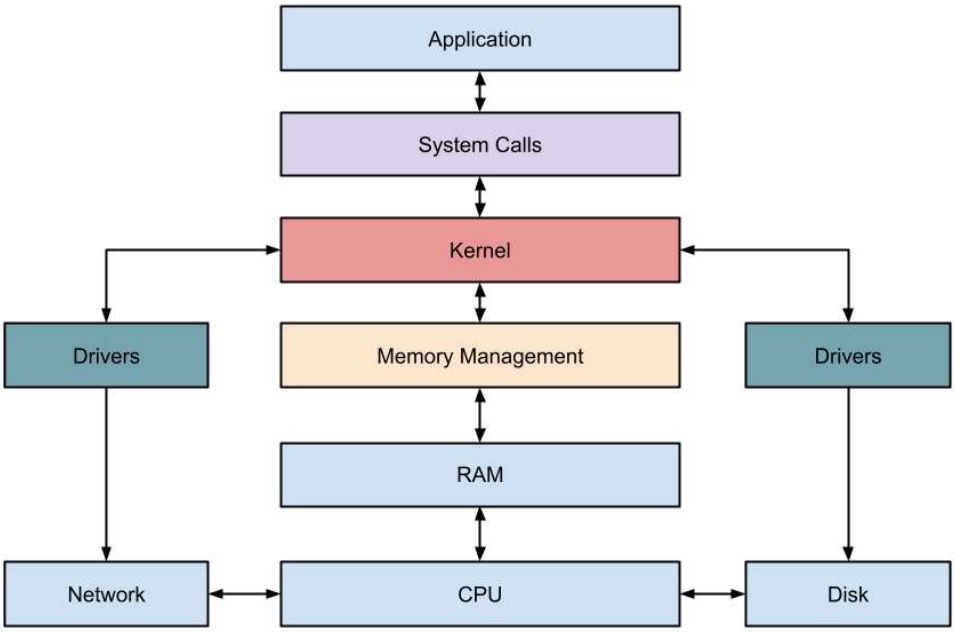

Figure 2.7 Schematic representation of the communication of the kernel and its surrounding elements [42].

There is another utility where we write these commands, and it is called a console. A console is a tool with a graphical and command line interface that allows the user to enter commands by taking place between the user and the shell. Commands are delivered to the shell via this tool.

2.2.4 Linux File System

Linux operating system is a file-based operating system. Many operations, such as device accesses, I/O processes, and inter-process communication, are performed as files. Each process maintains a file descriptor, and this file descriptor points to the file it is currently using. This filing system can be compared to a tree, and its root is denoted by a slash(/).

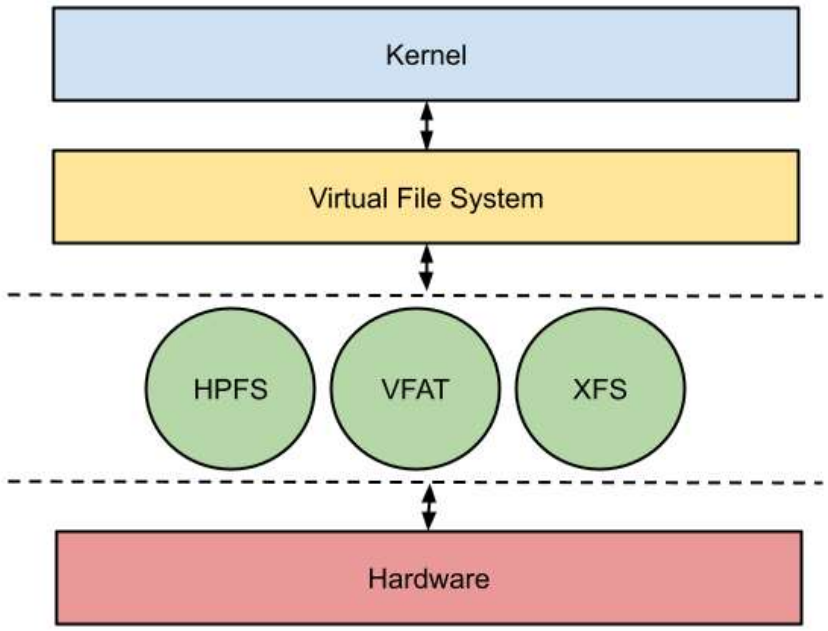

Figure 2.8 Linux file system diagram [43].

Under root are folders specialized for certain processes, and under each folder, there are subfolders and files. Examples of these folders are:

- *"bin":* is where commands run in the shell are kept. Commands to be used as builtin when a shell is opened are kept as files under this folder. Bash is an example.
- *"boot":* It is responsible for loading the operating system.
- *"sbin":* The commands to be executed at the system level are kept.
- *"dev":* All I/O devices are kept here. Mouse, keyboard and speaker are examples of these I/O devices. `"/dev/dsp"`, `"/dev/usb"`, `"/dev/sda"` is an example of a folder hierarchy.

- *"etc":* This folder contains the configuration files from the operating system. For example DNS(Domain Name System) settings `"/etc/resolve.conf"` kept in this file.
- *"proc":* Instant information about processes is kept here.
- *"var":* Logs of the operating system and other running applications are located here.
- *"tmp":* This is where all kinds of temporary files are located.
- *"mnt":* This is where mounted files are kept temporarily.
- *"opt":* This is where the applications' library files are located.
- *"srv":* All service files in the operating system are located here.
- *usr":* This is where binaries of applications are kept.
- *"lib":* This is where system libraries are located rather than application libraries.
- *"home":* Under this folder is the user folders.

2.2.4.1 File Descriptors

File descriptors are used for low-level file operations. There are 3 file descriptors in Linux:

- *Standard input:* Its numeric value is 0 and is interpreted as input. It is represented by stdin in the stdio library.
- *Standard output:* Its numeric value is 1 and is interpreted as output. It is represented by stdout in the stdio library.
- *Standard error:* Its numeric value is 2 and is interpreted as an error. It is represented by stderr in the stdio library.

2.2.4.2 Symbolic and hard links

Symbolic links can be thought of as shortcuts. Allows pointing from one file to another. It is different from normal links. Normal links add a pointer to the filename in the directory list, but symbolic links have a sign containing another filename. This means that the inode values are the same on hard links. The `"ln -s"` command is used to create a symbolic link. Only the `"ln"` command is used for hard links.

2.2.4.3 Inode

Operating systems that use the Linux kernel use a proprietary file system. An inode is a unique number assigned to a file or directory. Inodes are data structures in UNIX-based file systems such as ext3 and ext4. Ext3 and ext4 keep an array of these nodes called the inode table. This table contains a list of all files in the file system. Each node in the inode table has a unique inode number.

As can be seen in Figure 2.10, which shows the Inode structure, the following information is held in the Inode:

- Filename
- File type and size
- Number of links: Number of hard links per node.
- Pointers from the file to each disk block where data is held.
- Access permissions to the file.
- The owner of the file.
- Timestamp: Access time and modification time.
- The group the file belongs to.
- Other metadata about the file.

There are also commands related to the inode. The most used of these are:

- "df -i": Prints the inode table
- "ls -li": Prints the inode numbers at the current location

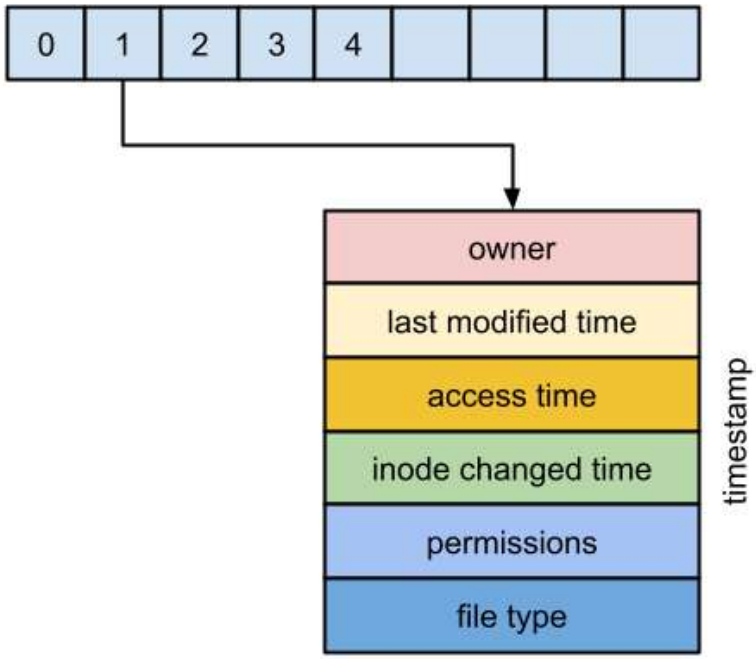

Figure 2.9 Inode structure [44].

2.2.5 Memory Management

As much memory as feasible is mapped at a fixed offset on Linux's operating system architecture's X86-derived CPUs. Since legacy devices can map at a lower level in the lowest 16M, that area is referred to as "zone dma." All virtual memory used by users is accessible to the Linux kernel via self-mapping. Virtual memory has been mapped to physical memory. High memory is the term used to describe any physical memory that cannot be temporarily mapped. Not all of the user's 4GB of virtual address space is given to them. The kernel can be built without normal memory to allocate a total 4GB to user space, although this harms speed.

When the standard kernel memory allocators are not yet operational in the early boot period, one approach for managing memory regions is by using memblock. The Linux kernel utilizes this approach for the "x86_64" architecture. "/linux/memblock".h is a header file that contains definitions for memblocks.

Conversion between physical and logical addresses is required to access directly mapped pages. There is a structure page for each physical page. There are certain established memory management structures. Each mm structure it references specifies the address area where the thread is currently operating. For this operation, "mm_struct" has a pointer in the page table—a red-black tree and a doubly linked list store this pointer match. Every "VMA"(Virtual Memory Area) specifies the region where every page is produced by the same object and has the same flags. A fresh "VMA" is created with each "mmap()" function.

Operating systems can use the hard disk as RAM in cases where RAM is insufficient. The part of the hard disk used like RAM is called swap space. This situation has some advantages and disadvantages. While it is a disadvantage that the hard disk is not as fast as the RAM, it is an advantage to prevent the RAM from crashing by filling up. In Windows and Linux operating systems, the user can configure swap space. In Linux, information about swap space is kept in the "/etc/fstab" file and is manually configured through this file.

2.2.5.1 Struct Page

All system physical page information is stored on the structure page. For all system pages, the kernel has a build page structure. This structure and several functions interact:

- The virtual address's corresponding page is returned by the item verb "virt_to_page()".
- The "pfn_to_page()" component returns the page that corresponds to the page frame number.
- The page frame number related to the page structure is returned by the item verb "page_to_pfn()".
- The "page_address()" component delivers the build page's virtual address. Only low-mem pages receive calls from these functions.

2.2.6 Interprocess Communication

Communication between processes is among the most critical functions of the operating system and the kernel. This communication can be from the kernel to the user side or from the user to the kernel side. When it is desired to access the kernel with the written codes, communication between the processes must be used. Therefore, like all modern UNIX-based systems, Linux provides various mechanisms for inter-process communication. There are 4 basic ways for this. The most important of these is `"procfs"`.

`"procfs"` is a pseudo-file system. Files that appear in this directory can be thought of as kernel variables. The kernel uses this directory to display internal information in files corresponding to `"/proc"`. `"procfs"` provides access to the kernel by directly accessing the virtual file system. Most files on the proc filesystem are read-only, but some files can be written, and kernel variables modified. It usually comes mounted on the system, but if it is not mounted on the system, it must be mounted with the command: `"mount -t proc proc /proc"`

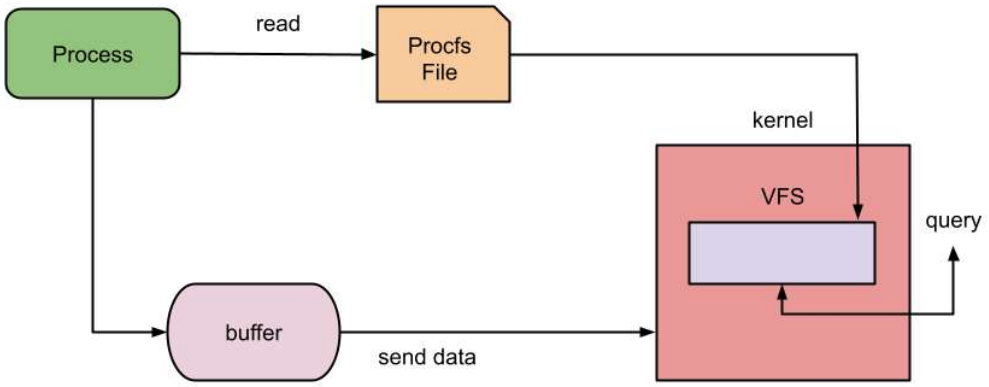

Figure 2.10 "Procfs" diagram [45].

2.2.7 Linux Command Structure

Commands run in Linux are run in the shell. There are several types of shells. There are certain standards that every shell must comply with. The features of these commands can be listed as follows [46]:

- All UNIX-based operating systems are case-sensitive.

- Commands are taken from the keyboard, and the outputs are printed to the screen. Some commands ensure that the output is not printed to the screen or redirected elsewhere.
- As in any operating system, the use of some characters is prohibited.
- Each command has options and arguments. Multiple operations can be done with each command.
- Each command has help and man pages.

The commands used for Linux kernel modules are:

"insmod" utility is used to add packages to the Linux kernel. Users can create kernel modules on Linux systems and load them into the kernel. LKMs (Loadable Kernel Modules) often provide system calls, software, and file system support to new devices. With or without parameters, it used to load an executable module with the extension ".ko" [47]. Insmod command usage:

```
"insmod [file name] [module options...]"
```

No matter how the Ubuntu system is loaded, modules cannot be loaded in the same place over and over. It cannot be used multiple times even if enabled in two places. To be able to load a component, dependency prototypes between kernel and driver must be provided. Just installing the module is not enough. The "modprobe" command is used to load the module to be loaded together with its components. This is how it differs from the "insmod" command. Installed kernel components, "lsmod" can be viewed using the command. Information about this is also "/proc" located under. Before using the "insmod" directive in the shell, it is necessary to make sure that the system is up to date and that no updates are needed.

"modinfo" command, on the other hand, gives information about the modules loaded into the system. Modinfo command usage:

```
"modinfo [-0] [-F field] [-k kernel] [modulename...]"
```

/"proc/modules" where information about the status of modules installed on the system is kept. "lsmod" reads the file and prints its contents onto the screen. It takes no arguments. When using these Linux commands, in some cases, it is desirable to redirect the output of one command to another command. In this case, "pipe" character is used. It is often used for operations from the command line.

2.2.8 Library Functions

Library functions are often used in kernel development. These functions are used to perform the operations allowed by the system on the kernel. For example, when trying to open a file, "fopen()" library using the "open()" system call is made. C libraries have different implementations on UNIX. The most commonly used is the GNU C Library. These common operations can be divided into two groups:

- Library function calls.
- System function calls.

Functions that belong to the standard C library are defined as library functions. "strcmp()" and "strlen()" functions are library functions that allow them to operate on arrays. Functions that do not make system calls are functions that do not change the internal functioning of the system. They are primarily used in coding arrangements. For example, the "strlen()" function, which returns the size of the array, is in this category. The other type is functions that make system calls. An example is the "fopen()" function, which uses the "open()" system call, a standard library function. The difference between open and "fopen" is that "fopen()" buffers a file for I/O operations, while "open()" is a system call that provides unbuffered I/O.

In general, if there is a library function that corresponds to a system call, applications should use the library function because system calls are more complex. Because library functions work on all systems that use library functions, an application running on one system will thus be able to run on another. Since system calls can vary from system to system, they are more disadvantageous than library functions. Because library

functions buffer processes, they switch less from user to kernel mode. For example, when writing an application that will read a lot of files, using "fread()" instead of "read()" will provide buffered I/O. This means that not every call to "fread()" will result in a system call. This is a situation that contributes significantly to the performance. There is a misunderstanding with "malloc()", the most popular library function. It is also that "malloc()" is not a system call. "malloc()" is a library function that calls "brk()" or "sbrk()" in the background Here are a few of the differences between a system and a library call:

- A library function is user program based and runs in user space. If it is a system call, it is not dependent on a user program and is executed in kernel space, not user space.
- A system call execution time is assumed to be part of system time because it runs in kernel space, while a library function execution time is at the user level as it runs in user space.
- Library functions can be easily debugged using the debugger because they run on the user side. System calls cannot be debugged because the kernel executes them.

2.2.9 Networking

The network subsystem is not an essential component of the operating system kernel. This means that the Linux kernel can be compiled without network support. In Linux, files related to network settings are located in the following directories:

- "/etc/sysconfig/network"
- "/etc/sysconfig/network-scripts"
- "/etc/hosts"
- "/etc/resolv.conf"
- "/etc/nsswitch.conf"
- "/etc/services"

Each of these settings files is used for different operations. Files in the network-scripts directory are used for network settings. Network settings are kept in files starting with ifcfg, with names such as eth0, eth1 according to the network card.

"struct socket", "struct sock", and "struct sk_buff" are the three fundamental constructions for handling network packets on the kernel side. A socket's first two elements are abstractions. The network representation of a socket, known as an "INET" socket in Linux, is represented by a "struct socket". This programming abstraction is similar to user space sockets, such as BSD sockets. The "struct socket" and "struct sock" are connected because the "struct socket" has an INET socket domain, and the "struct sock" holds a BSD socket. A network packet's status and representation are shown in the "struct sk buff". The fabric is constructed when a kernel packet is received from the user space or network interface.

CHAPTER 3

LINUX KERNEL MODULE DEVELOPMENT

3.1 Linux Kernel Module

In Linux-based operating systems, there are ready-made modules installed in the kernel. These create the kernel itself. Apart from the existing modules, the modules to be loaded into the kernel can be developed and loaded into the system. Modules are often confused with drivers. Drivers have direct hardware access, while modules do not have direct hardware access [48]. Since the modules work in the kernel space, every operation is critical because while the operating system restricts what the user can do while working in the user space, there is no such restriction in the kernel space. For this operation, commands such as verb|"modporbe"| "`insmod`" and "`depmod`" must be installed in the system. They are installed differently in different Linux distributions. Command to install kmod on Debian-based Linux:

```
"sudo apt-get install build-essential kmod"
```

Modules loaded in the kernel are in the "`/proc/modules`" file. Kernel modules to start with "`init_module()`" contains "`cleanup_module`" at the end. "`init_module()`" when using for start "`cleanup_module`" is called last. Cleanup process "`rmmod`" is called before the command. "`rmmod`" is the opposite of "`insmod`". While the module is loaded with "`insmod`", the module is deleted with "`rmmod`". Therefore, by calling after the cleaning processes are completed, it is guaranteed that the cleaning processes are completed. Usage of rmmod:

```
"rmmod [-f] [-s] [-v] [modulename]"
```

In the Linux system, most log files are kept in the log folder under the /var directory. In addition, other subdirectories contain the logs of certain programs and services in the log directory. "`printk()`" module is used instead of "`printf()`". This is because the modules do not write anything on the screen. Instead of printing something to the screen, logging is done, and these logs are saved in "`/var/log/kern.log`". This

approach means that any situation that may occur can be seen from the logs, and action can be taken accordingly. Otherwise, the developer will not be aware of the error conditions. As in any programming language, there are levels of logging:

- `"KERN_WARNING"`: Indicates a warning about a problematic condition that does not cause serious system problems.
- `"KERN_DEBUG"`: Used to debug messages.
- `"KERN_CRIT"`: Represents a critical condition associated with a serious hardware or software failure.
- `"KERN_NOTICE"`: Represents the normal state but is something to be aware of.
- `"KERN_INFO"`: Represents the information message. Many drivers print hardware-related information they find at this level at boot time.
- `"KERN_EMERG"`: Usually used for pre-execution emergency messages.

The module will not be loaded if the header cannot be loaded in the modules. For this, it is necessary to load the header beforehand. Installing headers [49]:

```
"apt-cache search linux-headers-'uname -r'"
```

3.1.1 Writing Modules for Multiple Kernel Versions

Modules use system calls. These system calls must remain the same in varying kernel versions. Otherwise, the module that works in one version may not work in the other. Therefore, even if new system calls are added, not making any changes to the old system calls is necessary for the stable operation of the modules. This condition must be provided in the software for backward compatibility. In principle, new kernel versions specify that they work without interrupting normal operations.

In most cases, device files also remain the same. It is crucial for kernel modules where kernel versions of the written module are requested to run. Tests are made on the specified kernel versions. When a different kernel is booted, if the

"CONFIG_MODVERSIONS" option is not enabled, the modules compiled for it will not be loaded. There is module versioning for this. The standard Linux distribution kernel is preconfigured. If there is a problem loading module due to versioning errors, the kernel should be tested with mod versioning disabled.

3.2 Modules vs Programs

A module always calls "init_module" or "module_init", then starts with the defined function call. This is the import function of the module. It relays to the kernel what functions the module starts. Functions are used in the codes inside the modules. For example, when the C program uses "printf" function, it requests the system's libraries in the background.

3.3 Name Space

When writing code in any programming language, variable names should also be meaningful to someone else. Defined global variables or externally accessed functions must not conflict with other externally accessible variables. Some variable names can cause conflicts. When writing code to the kernel, this can be a huge problem because even the smallest module depends on the entire kernel. When everything is declared static, writing it to the symbol table and storing it in the kernel is what name spaces do.

3.4 Device Drivers

Multiple ways are used to access devices. Access via "/proc" is one of the most preferred methods. In operating systems, communication is bidirectional. Just as the kernel can send information, processes can also send information to the kernel. Device information is kept under the "/proc" file system. "file_operations" used in function calls structure represents how the char driver makes this connection. Processes are primarily responsible for executing system calls. Some of the functions defined in file_operations struct are as follows:

```
"void (*show_fdinfo)(struct seq_file *m, struct file *f);"
"int (*open) (struct inode *, struct file *);"
"ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);"
```

```
"ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);"
"ssize_t (*ioctl) (struct inode *, unsigned int, unsigned long);"
```

3.4.1 Registering A Device

Device files frequently found at "/dev" are used to access char devices. It is good to place the device file in the current directory while developing a driver. The driver manages which device file is indicated by the significant number. "register_chrdev" function define:

```
"int register_chrdev(unsigned int major,
                     const char *name,
                     struct file_operations *fops);"
```

When something is not permitted, the function often returns an error code. However, a counter that counts the number of processes utilizing module is available. By reading "/proc/modules" or using the "sudo lsmod" tool, you may determine its value. If this value does not equal zero, "rmmod" will not work. Because the "sys_delete_module" syscall specified in "include/linux/syscalls.h" performs the check, the counter in cleanup module also has to be examined. "include/linux/module.h" contains these routines:

- "try_module_get" : Increase the current module's reference count.
- "module_put" : Reduce the current module's reference count.
- "module_refcount" : The current module's reference count is returned.

3.5 System Calls

Processes in user space cannot access kernel memory and cannot call kernel functions. This is a necessary approach to security. This technique is employed when a process asks the kernel for a service. Syscall is used to modify the behavior of the kernel. The "strace <arguments>" command can be used in the meantime to view the system calls that a program is utilizing. The place in the kernel that a process can access is

called "system_call". The working logic of this "system_call" can be seen in Figure 3.1.

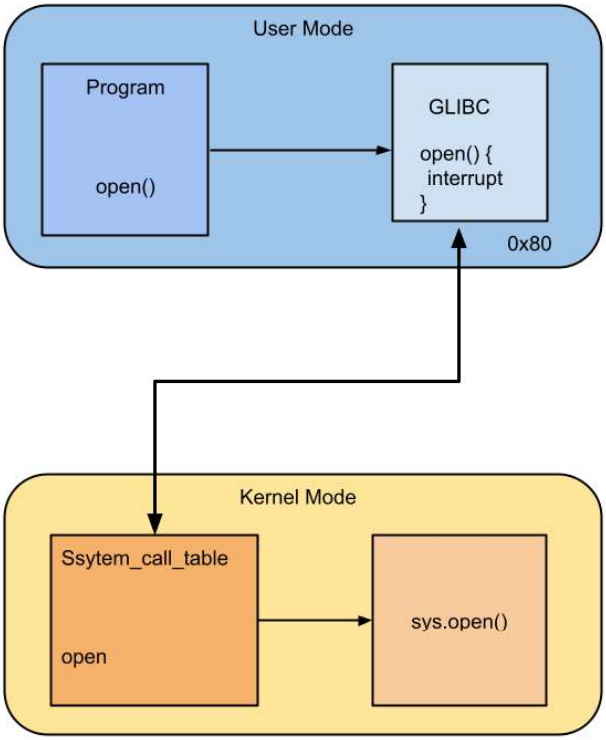

Figure 3.1 Sycall diagram [50].

The program first makes a macro call. As seen in Figure 3.1, a file open call has been made. GLIBC(GNU C Library) puts it as a "0x80" transfers it to "sys_call_table" in kernel space via the interrupt. According to the match in this table, the call at the system's kernel level is completed. It may be desirable to change the operations of system calls. To do this, a custom function must be defined by the user. This is usually done by writing a new program with defined functions. "sys_call_table" indicates the pointer function inside.

One must build a unique function and then edit it to alter how a specific system call behaves. The function is pointed to by the pointer in the "sys_call_table". So that it may be deleted later and the system is not left in an unstable condition, "cleanup" module must return the table to its initial state. The control record must be considered if the contents of "sys_call_table" are altered. Making a "system_call" and changing the records on the table is a severe operation. So every step should be done carefully. A control register is a processor register that alters or regulates the CPU's

general behavior. For the "x86" architecture, the "cr0" register contains several control flags that alter how the processor's core functions. Write protection is denoted by the "WP" flag in "cr0". After the WP flag has been set, the processor will not let any more attempts to write to the read-only partitions. As a result, disabling the WP bit is required before altering the "sys_call_table". Since Linux v5.3, the write "cr0" function has been disabled owing to sensitive "cr0" bits being corrected due to a security flaw. However, an attacker may still deactivate CPU protections like write protection by writing to CPU control registers.

3.6 Interrupt Handlers

The types of communication between the peripheral elements and the processor are categorized in two different ways. The communication here is the communication between the CPU and the peripherals. An interrupt is used in data exchange between the CPU and peripherals. This data communication is bidirectional. When an interrupt is used when peripheral elements need to communicate with the CPU, it is much more complex than the messages going from the CPU to the peripheral aspects since the communication here is from the peripheral elements to the CPU. The closer you get to the environmental elements layer, the more complex all processes become. Peripherals often have tiny RAM, and when available, information is lost if not read.

If an operation needs to be executed and the kernel module is performing another operation while that operation is being performed, then the process can be put to sleep mode. It is defined under "/proc/sleep" in kernel modules. It can only be used by one process at a time. "tail -f" is the most common command used to stream the file. This command marks the status of the process as "TASK_INTERRUPTIBLE" and stays in this state until the command terminates. Other processes that want to access are queued. When the command is terminated, "module_close" is called, and all processes in the queue are woken up. Calling "module_close" alone does not wake processes waiting to access the file. Another common method is to send a "Ctrl+c" signal. This is interpreted as "SIGINT" in Linux operating systems. This is because we are using "module_interruptible_sleep_on". If "module_sleep_on" is used instead, "Ctrl+c" becomes dysfunctional. This may not be good for user experience.

In some cases, processes do not process the sleep command. In these cases, the file uses the "O_NONBLOCK" flag when opening. The kernel returns with the "-EAGAIN" error code for actions such as preventing the file from being opened.

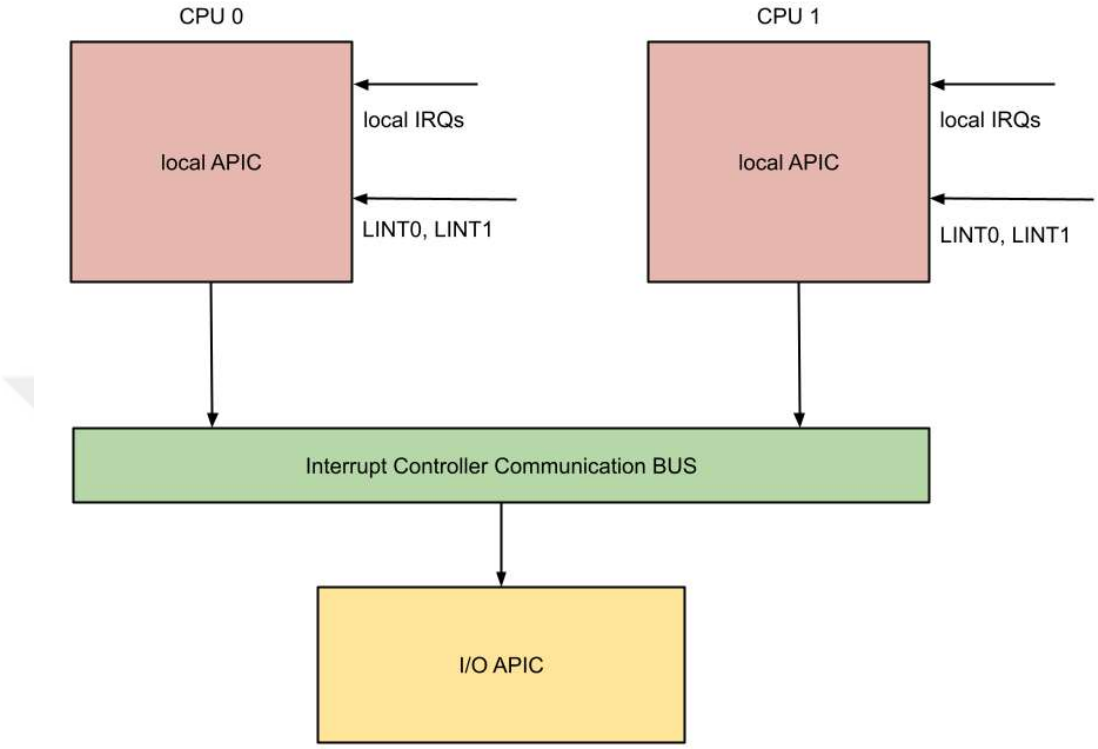

Figure 3.2 Interrupt diagram [51].

3.6.1 Request Access to I/O Ports

I/O accesses are always at the user's disposal. Therefore, access is requested, ensuring it can be allocated to a user. "request_region()" function is used for this purpose. After the process is finished, "release_region()" function must be used. This is necessary for freeing the allocated space. The function definition in the C code is as follows [52]:

```
"void release_region(unsigned long start, unsigned long n);"
```

For example, the base address of the COM1 serial port is "0x3F8". Requests to these link addresses "request_region" done with the function. The function definition is as follows [52]:

```
"#include <linux/ioport.h>
```

```
#define MY_BASEPORT 0x3F8
#define MY_NR_PORTS 8

if (!request_region(MY_BASEPORT, MY_NR_PORTS, "com1")) {
    return -EN0DEV;
}"
```

Port requests are typically made during driver initialization, and port freeing occurs when a device or module is removed. All port requests from user space "/proc/ioports" appear through the file. The output of this file is as follows:

```
"$ cat /proc/ioports
0000-001f: dma1
0020-0021: pic1
0040-005f: timer
0060-006f: keyboard
0070-0077: rtc
0080-008f: dma page reg
00a0-00a1: pic2
00c0-00df: dma2"
```

3.7 Task Scheduling

There are two ways to run processes on computer systems: tasklets and queue workers. When running a single process, tasklets are the fastest and easiest way to do this. Queue workers are used when scheduled operations are performed on more than one process. Queue workers are more complex than tasklets.

Tasklets are used to queue future tasks. Tasklets can be run in parallel, but one of the most significant disadvantages of tasklets is that the same Tasklet cannot be run on multiple CPUs simultaneously. Another downside is that the tasklet callback runs in

an atomic context, in a software interrupt. Tasklet struct definition:

```
"struct tasklet_struct
{
    struct tasklet_struct *next;
    unsigned long state;
    atomic_t count;
    void (*func)(unsigned long);
    unsigned long data;
};"
```

The job queues in the Linux kernel version 2.6 are another task scheduling method. The task code added to the job queue in this method has the properties of the process context. Job queues can be programmed and put to sleep when not needed. When deciding between using tasklet and work queue, it is necessary to consider whether the job can sleep or not. If it needs to match, a work queue is used. Work queue initialization define:

```
"DECLARE_WORK(name, void (*func)(void *))"
```

Allocate the work to the queue function:

```
"int schedule_work( struct work_struct *work );"
```

If a job hasn't previously been queued, this "schedule_work" method adds it to the kernel-global workqueue; otherwise, it leaves it in the same spot.

Some computer tasks need to be done at a specific time or regularly. If a process performs the job, this "crontab" is included in the file. This process is open in user space and can be defined by all users. "crontab" available on Linux. It is possible to create tasks that are set to run at specific times by using the unique syntax of the file. This process is primarily used in server-based systems. It runs as a service on Linux. Our service allows us to perform scheduled tasks that provide these routine repetitions is called cron. A few crontabs command options defined in Linux:

- Editing can be done by opening the file with the "`crontab -e`" command.
- The "`crontab -l`" command can be used if it is desired to list the specified scheduled tasks. The "`-l`" parameter represents the word list.
- The "`crontab -r`" command can be used to delete all created scheduled tasks. The "`-r`" parameter here represents the word remove.

The first method is to define a function to "`crontab`" using "`crontab`"'s own syntax. This is not a very preferred method because it is inefficient. File opening and writing operations are required. This is a very costly operation. It is also one of the biggest problems for concurrency. The second method is "`workqueue_struct`". A task is created using necessary functions called from this queue.

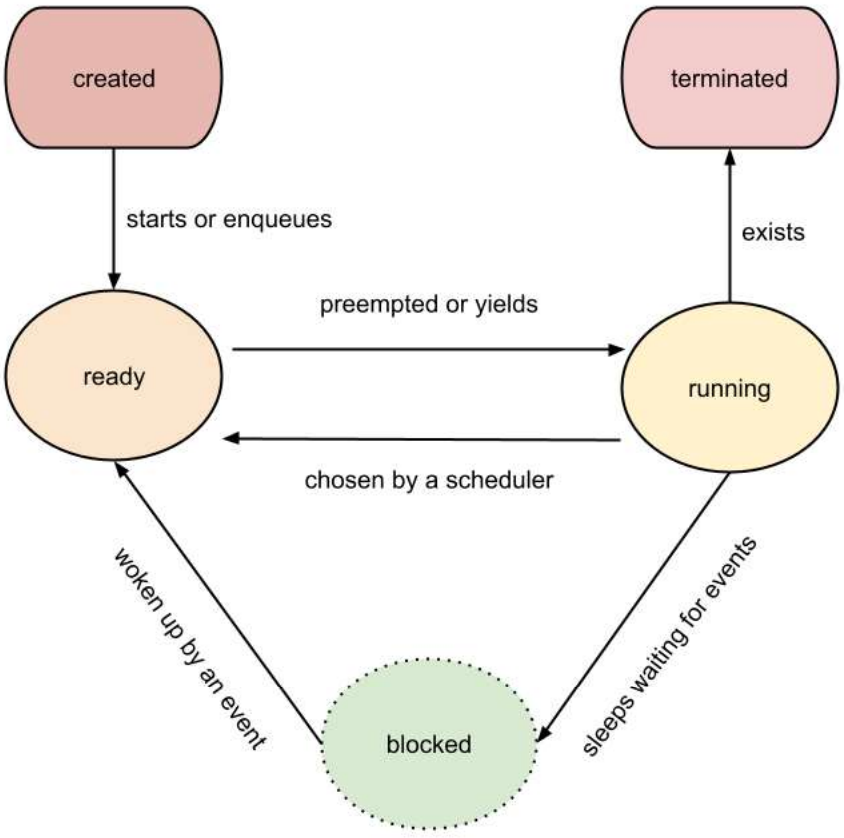

Figure 3.3 Task scheduler diagram [53].

3.8 Multi-Processing

One of the ways followed to increase the performance of computers is to produce software that works more efficiently. The second is to improve hardware performance.

Multi-processing allows multiple processes to be performed on multiple cores simultaneously. This allows different processors to run different jobs. This is also called parallelism and allows the same job to be done in less time, and faster.

The kernel can be called by different processes running on different processors. "dmesg" command is used on Linux to see and better describe the filtering outputs. Since the system is booted, all messages produced by the kernel and records about the kernel are kept in the "/proc/kmsg" directory. Instead of all kernel records, the boot notes written at system boot can be viewed with the "dmesg" command. The use of this command is often used to detect problems reported at system startup and to detect other system warnings. "Dmesg" helps examine the kernel ring buffer. Ring buffer, on the other hand, was born from the definition of circular buffer. Linux provides many functions where the circular buffer can be used in its application area. There are two groups of such features:

- Memory barrier created when creators and consumers of objects in buffers do not want to share locks.
- The lock mechanism that only one producer and one consumer can use.

The circular buffer is a fixed-size buffer with two indexes.

- *"head index":* The point at which the renderer added the item to the buffer.
- *"queue index":* The point at which the consumer finds the next item in the buffer.

The buffer is empty when the tail pointer is equal to the head pointer. Adding an item to the buffer increases the head index, and removing an item from the buffer increases the tail index. The queue index should never skip the head index, and both indexes should drop to 0 when they reach the end of the buffer, allowing an infinite amount of data to pass through the buffer.

If the buffer contains multiple items or items of variable size, the indices can be increased by more than 1 as long as one of the indices does not exceed the others.

Calculating the occupancy or remaining capacity of an arbitrary-size ring buffer requires using the module instruction, which is usually slow. However, the much faster "AND" command can be used if the buffer is doubled. Linux provides a set of macros for working with ring buffers for use in different methods.

Using memory barriers with ring buffers prevents:

- A single lock is used to manage access to both ends of the item buffer.
- Atomic counter operations are used.

This has two aspects. A producer fills the buffer, and a consumer empties the buffer. Only one thing should fill the buffer at any one time, and only one thing should empty it at any given time, but both sides can run simultaneously. In these cases, the lock mechanism is used. This indicates to the CPU that the new item's content should be written before the header index is presented to the consumer, and then the revised header index should be written to the CPU before the consumer wakes up. Wake_up() macro is used to determine if an item exists. Other than that, there is no way to guarantee this.

3.9 Avoiding Collisions and Deadlocks

One of the problems computer systems try to solve is when processes using several CPUs or threads try to access the same memory. Some techniques have been implemented in the kernel to stop this. These are used to lock or unlock that thread. As a result, the undesirable effects of simultaneous startup attempts do not occur. A mutex is the first method listed in the kernel's definitions.

Mutex defining:

```
"DEFINE_MUTEX(name);"
```

Mutex initializing dynamically:

```
"mutex_init(mutex);"
```

Uninterruptible mutex acquire:

```
"void mutex_lock(struct mutex *lock);"
"void mutex_lock_nested(struct mutex *lock, int subclass);"
"int  mutex_trylock(struct mutex *lock);"
"int mutex_lock_interruptible(struct mutex *lock);"
```

Unlock the mutex:

```
"void mutex_unlock(struct mutex *lock);"
```

Test if the mutex is locked:

```
"int mutex_is_locked(struct mutex *lock);"
```

Another method is to use a spinlock. It seizes all of the resources from the code's CPU and locks it. Because it is a costly procedure for the processor to save all the state of the relevant thread and return it when it wakes up, the thread is not put to sleep in situations when the semaphore is anticipated to be released fast. Until the semaphore is released, it is kept alive for a time, and a few empty rounds are produced. It's known as busy waiting. These semaphores, known as spinlocks, are frequently employed successfully in low-level activities. If a thread in a spinlock semaphore does not get the semaphore, it has been waiting for a predetermined amount of time. The processor suspends it like a regular thread. In multi-core computers, spinlocks have a purpose.

The library related to spinlock in Linux kernel is "<linux/spinlock.h>". It initiates a spinlock, and the initialization at compile time is:

```
"spinlock_t my_lock = SPIN_LOCK_UNLOCKED;"
```

or at runtime:

```
"void spin_lock_init(spinlock_t *lock);"
```

To lock:

```
"void spin_lock(spinlock_t *lock);"
```

When "spin_lock" is called, it waits until the lock becomes available.

To unlock:

"void spin_unlock(spinlock_t *lock);"

CHAPTER 4

MATERIALS AND METHODS

4.1 RAM Image

Memory dump analysis allows the investigation of unwanted activities performed by malicious software on the system with specialized tools [54]. It is considered an important malware analysis method in the forensic [55] field, as it contains the output of all transactions directly on memory.

Applications run on computer systems using processes. The operating system does this work by loading it into memory. Some malware may use methods such as injecting itself into memory areas of other running processes, but they still need to be handled in memory. All modern processors pull codes and instructions from their caches to run them. If the required space is not loaded into the cache, this space is loaded from memory. For this reason, the cache and memory proceed simultaneously; therefore, the activities of the processes running in the system can be determined by performing memory analysis. After the Ram Image is taken, the resulting file should be scanned with ram image analysis tools.

4.2 Memory Acquisition

Memory acquisition is a method that allows the scanning of processes that are mapped live on memory. Thanks to this method, information about processes mapped to memory can be obtained, similar to the one in this study. However, since this method, which is used with conventional methods, works in user space, the entire memory is not scanned. This differs from the method used in memory dumping in this study [56]. Since an application that does not run in kernel space will have limited memory access, the method of working on the kernel side while memory dumping is preferred. Thus, the memory restriction is bypassed by using the kernel's macros.

Considering that each operating system has different structures, there are different methods for memory acquisition. Studies on the Windows operating system [57] are more frequent considering the usage rate of the windows operating system. In the

Linux operating system, separate collectors must be written for each data. In addition, these operations can be performed on embedded [58] and Android [59] systems.

4.2.1 Linux Resource Structure

In operating systems, Drivers read I/O ports in some cases and I/O memory in others. These operations are read-and-write operations. These read and write operations are done at machine startup and during other operations. These processes are two different builds exported in the Linux kernel and facilitate communication between the driver and the device.

The Linux operating system uses a generic linked list to create in-memory data structures. These structures define the resources available in the system and allow the kernel code to manage and allocate resources. Related structure; As seen in Figure 4.1, it can be iterated between siblings as a one-way linked list, while at the same time, each child is linked to its parent by a bi-directional linked list. Source structures; It is exported in "`kernel/resource.c`" under the GPL (General Public License), available for use by other drivers. Thus, there is no problem accessing C code.

4.2.2 PCI (Peripheral Component Interconnect) I/O

Developed by Intel Corporation, the PCI standard is the industry standard [60] found in almost all desktop computers. PCIs have different access methods on different processors and have 32-bit I/O field addressing. Each processor has its I/O area access methods. If a processor does not have special instructions, access is provided by the host bridge [61]. The port source addressing distance is limited to "`IO_SPACE_LIMIT`" in kernel side. Specifies the limited memory space via the base address of the respective I/O structure:

```
Base IO address + limit = maximum address limit of matching IO
```

4.2.2.1 MEM I/O

I/O memory is a bus made available to external devices. It is possible to transfer data from inside to outside and externally mounted devices. This memory can be thought of as serving as a RAM-like structure. The bus provided by I/O memory provides access

to the CPU [62].

The memory source addressing distance is -1: "0xffffffffffffffff". This means it is empty when the 64-bit maximum addressing space is reached. As seen in Figure 4.1, within the I/O source structures;

- Each child can have only one parent.
- Parents can have any number of children.
- All children linked to the same parent are linked to the sibling list.

Siblings are structures with a set of memory areas directly interconnected in one direction. A sibling's address range can have sub-hierarchy, and a sibling's memory space can be defined within sub-parents and children.

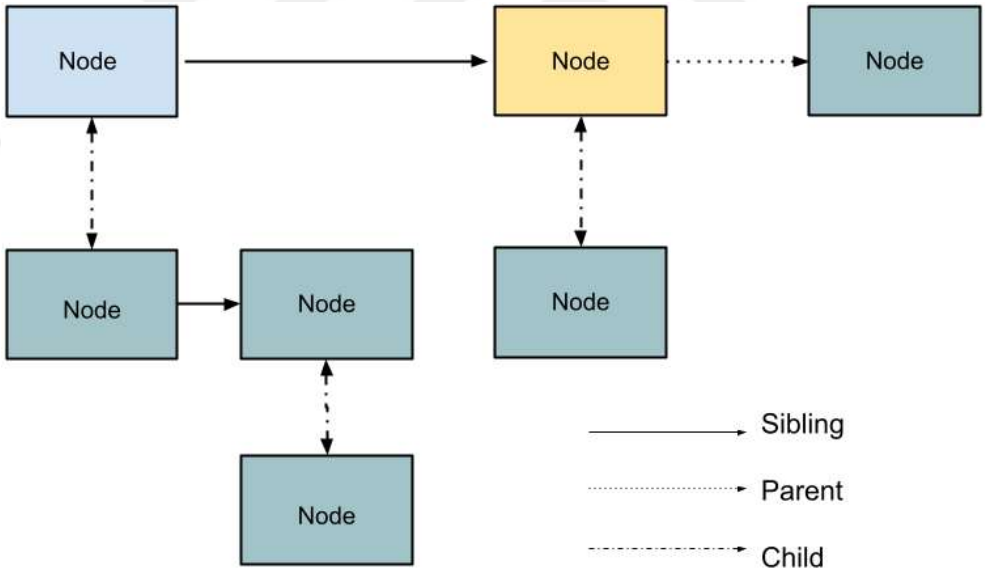

Figure 4.1 Resource structure hierarchy and linked list structure that handles system resources [63].

4.2.3 I/O Memory

I/O memory is available on the system, and Linux "/proc/iomem" covers the memory addressing included in the resource list. All allocated I/O addresses "iomem_resource" is in the kernel with its structure. Structure "resource.c", "ioport.h" is exported globally with macros such as representation of "iomem_resource" via code reference [64]:

```
"struct resource iomem_resource = {
     .name = "PCI-mem",
     .start = 0,
     .end = -1,
     .flags = IORESOURCE_MEM
 };
 "EXPORT_SYMBOL(iomem_resource);"
```

Flag information is required to filter the resources of the system's main memory. Since the physical memory regions in the system need to be allocated and accessed, "IOResource_Mem" is chosen as the flag variable here. Work is being done on the address ranges registered in "/proc/mem" , and the listed ranges represent physical addresses. These ranges correspond to the start and end of the code.

Linux distributions also use the "ioremap" value and do not have to use the return value as a pointer. This type of usage is not very effective, and kernel developers are making more and more efforts to stop it because it causes memory redundancy problems. There are three different addressing supported by Intel protected mode architecture [65]. These are logical, linear, and physical addresses. Address conversions are performed by the MMU (Memory Management Unit).

4.2.4 Linux Memory Addressing

Each program takes up memory space. The memory management algorithm takes care of this organization. These programs are mapped onto linear memory. Programs access a memory area of their own in the system. The writing of these programs takes place via virtual memory. These addresses generated by programs are called virtual addresses or logical addresses. The set of these virtual addresses is called the address space.

The places where processes are kept in memory are distributed in physical memory. Address conversion is required when the operating system wants to access data in the virtual address space. Physical address and virtual address conversions are the conversions for this access. For this, a table is kept by the Memory Management Unit,

Table 4.1 Memory address ranges addressed on the resource structure.

Address Range	Address Meaning
98c5d000 - 9974dfff	System Ram
9974e000 - 9974efff	ACPI Non-volatile Storage
9974f000 - 9974ffff	Reserved
99750000 - 99770fff	System Ram
99771000 - 99771fff	Reserved
99772000 - 9d614fff	System Ram
9d615000 - 9de14fff	Reserved
9de15000 - 9df44fff	ACPI Tables
9df45000 - 9e144fff	ACPI Non-volatile Storage
9e145000 - 9effefff	Reserved
9efff000 - 9effffff	System Ram
9f000000 - 9fffffff	Reserved
a0000000 - dfffffff	PCI Bus 0000:00
a0000000 - b1fffff	PCI Bus 0000:01
a0000000 - afffffff	0000:01:00.0
a0000000 - a086ffff	efifb

and this transformation is made over this table. These conversions are done by the MMU, as seen in Figure 4.2. Virtual memory is divided into pages for easy access after conversion. This page table is kept in physical memory. If the table to be accessed does not exist in memory, it will throw an error. The operating system transfers data between the disk and memory, which is done through pages by taking part in DMA (Direct Memory Access). When a program creates an address, the memory management unit is responsible for locating the page in the main memory by converting that address to a physical address. This architecture is in Intel 8086 [66] and is known as processor segment architecture.

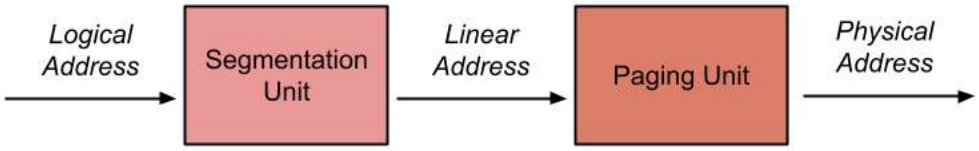

Figure 4.2 Indication of address conversion realized by the MMU [67].

4.2.5 Logical Addressing

The logical address space is divided into units of different lengths called segments. Segments consist of a different number of pages. Each program can create segments. The sections and data of the programs can be accessed by the conversion method shown in Figure 4.3.

In protected mode, the information contained in the segment registers is called by the selector. The selector specifies an index in the descriptor table. Selector; RPL (Requested Privilege Level) over 15 bits is addressed with the "TI (Table Index)" and "INDEX" value. The two low-significant bits of the selector to be loaded into the segment register are called RPL. This RPL (Requested Privilege Level) is tested in various ways. In these tests, cases such as segment access interrupted by a general protection fault may also occur. This algorithm forms the basis of the basic protected mode architecture.

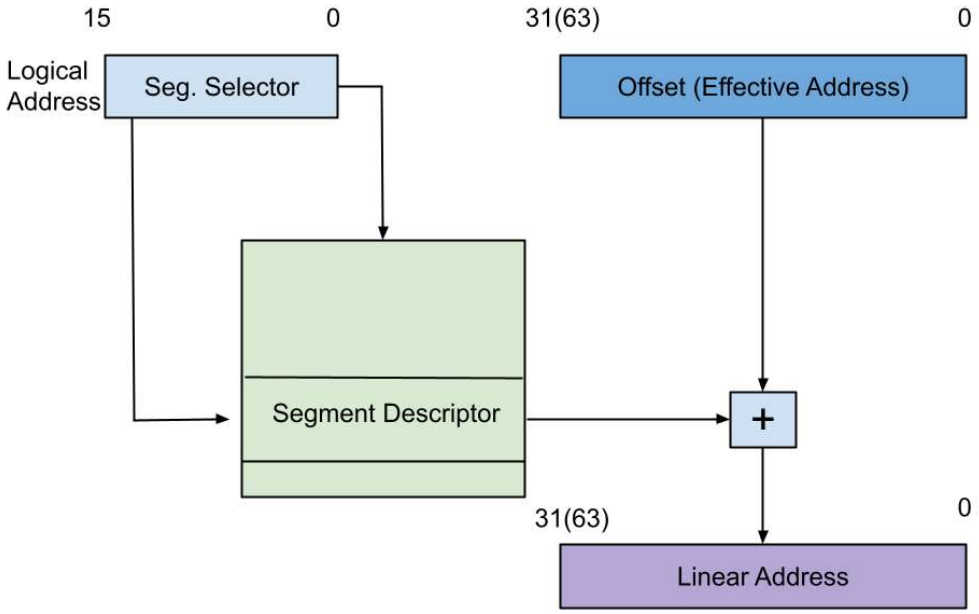

Figure 4.3 Indication of conversion of logical address to linear address [68].

The first two bits of the selector are used for privilege level control of x86 CPUs in protected mode. If the TI value is 0, the global descriptor table is checked; if the TI value is 1, it is indexed in the local descriptor table. As a result of the conversion shown in Figure 4.3, after accessing the relevant descriptive table, it is summed with the offset value, and the Linear Address, also known as the Virtual Address, is obtained. The linear address space refers to all addresses that can be mapped and used in the system. This linear address obtained is the output to be taken from this cycle.

4.2.6 RAM Image Stages

In the research, Section **??** describes the Linux kernel-specific resource structure and linear addressing based on protected mode architecture (IA-32). In contrast, two other forms of addressing (linear address and physical address) are also explained.

The first stage is filtering addresses mapped to the system RAM resource. There are different filtering methods at this stage. Filtering methods:

- Comparison of the related resource with its registered name in the "proc" file system over string.
- Bitwise comparison of flag information of enumerated resource structures.
- Reading the address range by the user and transmitting it to the driver as "modparam"

The preferred method is the second method. "IORESOURCE_SYSTEM_RAM" is used to mask and filter the bits pointed to in the resource flag. The definitions of I/O resource types and their usage according to the C language implementation are as follows:

```
"/*I/O Resource extended types*/
#define IORESOURCE_SYSTEM_RAM
(I/O RESOURCE_MEM|I/ORESOURCE_SYSRAM)"
```

In the "iomem_resource" struct, "IORESOURCE_SYSRAM", so the modifier bit is "IORESOURCE_SYSTEM_RAM" is set as.

Table 4.2 shows the "demsg" output of the filtering. "dmesg" output shows driver actions. "Dmesg" output can be interpreted according to the information given below:

- *"0x00000 - 0x9FFFF":* The portion of the addressing range is known in different sources as conventional memory or 64KB barrier. This space, which emerged with the Intel [69] 8088 CPU architecture and is used in IBM PCs, is now 64KB or 1MB.
- *"0x00000000 - 0x3FFFFFFF":* This 1GB address range is placed in kernel space. Relevant kernel symbols are handled within the field.

Table 4.2 Dmesg command output.

System Ram	End	Flag
1000	9dfff	81000200
9f000	9ffff	81000200
10000	3fffffff	81000200
40400000	97dfe017	81000200
97dfe018	97e0b857	81000200
97e0b858	97e0c017	81000200
97efc018	97e2b457	81000200
97e2b458	97ed4fff	81000200
97f32000	97f55fff	81000200
97f57000	98c5bfff	81000200
98c5d000	9974dfff	81000200
99750000	99770fff	81000200
99772000	9d614fff	81000200
9efff000	9effffff	81000200

- *"0x3FFFFFFF - 0xFFFFFFFF":* This 3GB address range is allocated to ring 3 processes, each process has a dedicated 3GB address space, 1GB kernel space is mapped to the process's virtual memory [70], but not accessible by the process.

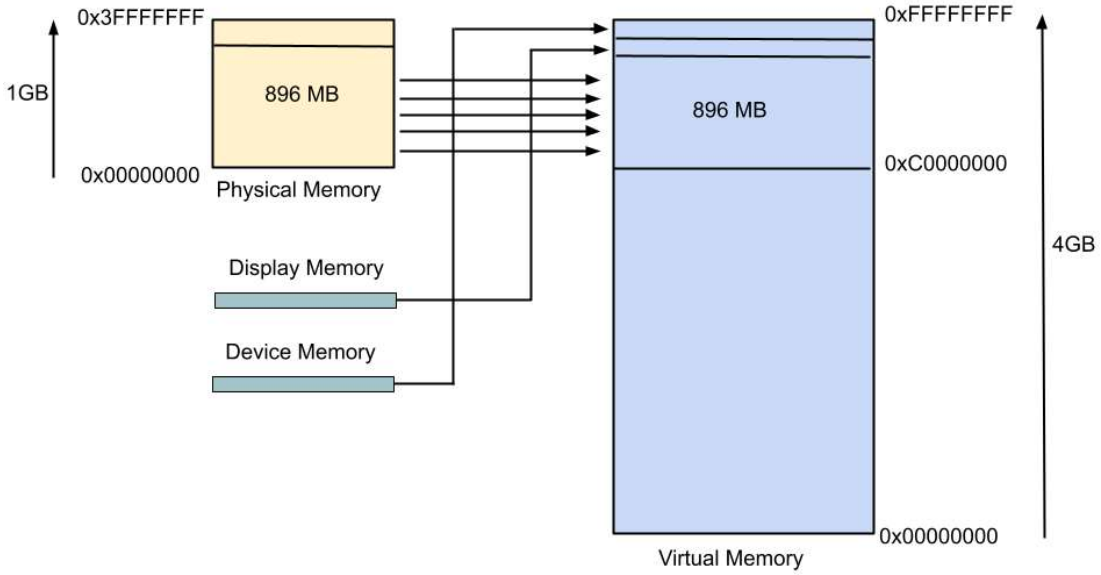

Figure 4.4 Linux kernel memory mapping [63].

Looking at the 896MB delta of the 1GB space allocated to the core for physical memory usage, it is seen that there is 128 B of space not mapped to the linear address on the DRAM (Dynamic Random Access Memory). This area is defined as high memory. This high memory is used when physical memory reaches its maximum. Replication

is performed between siblings via the I/O MEM source. The parent element contains the children's address range in the child hierarchy. So no additional iteration is needed for the sub-hierarchy.

The second stage is; It is the conversion of physical addresses in the "PAGE_SIZE" size of the system memory with the help of PFN (Physical Page Frame Number) and addressing the related virtual page. In kernel 2.6 and above, the page length is fixed and exported as 4KB as a global variable unless HUGE PAGES and HUGE TLB are configured in "meminfo".

TLB (Translation Lookaside Buffer) is a structure that stores the last loop addresses and makes future loops faster. It allows caching of paging cycles. The pagination here is active in all today's operating systems. It is updated if a particular piece of data in a TLB table cannot be located on the TLB. An update is not required if the data is already stored in the TLB. However, if the TLB is lost for whatever reason, the page table or disc will be used to update the TLB table.

Table 4.3 Architecture based page shift and page size table.

Architecture	PAGE SHIFT	PAGE SIZE
i386	12	4K
MIPS	12	4K
Alpha	13	8K
m68k	12	4K
m68k	13	8K
ARM	12	4K
ARM	14	16K
ARM	15	32K
IA-64	12	4K
IA-64	13	8K
IA-64	14	16K
IA-64	16	64K

For example, `97f57000-98c5bfff` If there are no partial pages in the range:

`(98c5bfff - 97f57000 / 1000 (range 4096)) = 3332` page must be.

- The source range start address is added by the iterative method with "PAGE_SIZE". As can be seen in Table 4.3, different intervals may be encountered in different

architectures.

- A callback is created with "walk_system_ram_range" [71]. The second method is not passed to the kernel, the procedure address is read from "System.map" and passed to the driver or "kallsyms_lookup_name()" [72]. "kallsyms_lookup_name()" has been exported in the kernel since kernel version 5.7.

For "walk_system_ram_range" in the "kernel/resource.c" file it says: "This function calls the @func callback against all memory ranges of type. System RAM which are marked as "IORESOURCE_SYSTEM_RAM" and "IORESOUCE_BUSY". It is to be used only for System RAM." "walk_system_ram_range" function macro define:

```
"int walk_system_ram_range(unsigned long start_pfn,
    unsigned long nr_pages,
    void *arg,
    int (*func)(unsigned long, unsigned long, void *)"
```

The first method was preferred because the second method is more risky and cannot be used for versions before kernel 5.7. PFE number was obtained by the. "Address >> PAGE_SHIFT" calculation, and the 12-bit address offset was omitted. An example loop can be shown as follows:

```
10010111111000101011010001011000 (0x97e2b458) => Physical Address
00000000000010010111111000101011 (0x00097e2b) => PFN
```

Figure 4.5 shows the 32-bit address conversion. This conversion can be done bidirectionally from a linear address to a physical address or from a physical to a linear address. To obtain the PFN number, the PFN is obtained by reversing the index and table index by shifting 12 bits to the right, as in the above example cycle. The page number was first converted to linear address with the "pfn_to_virt" macro and then the linear page address was obtained with the "virt_to_page" macro. These two processes summarize what the "pfn_to_page" macro does in the background.

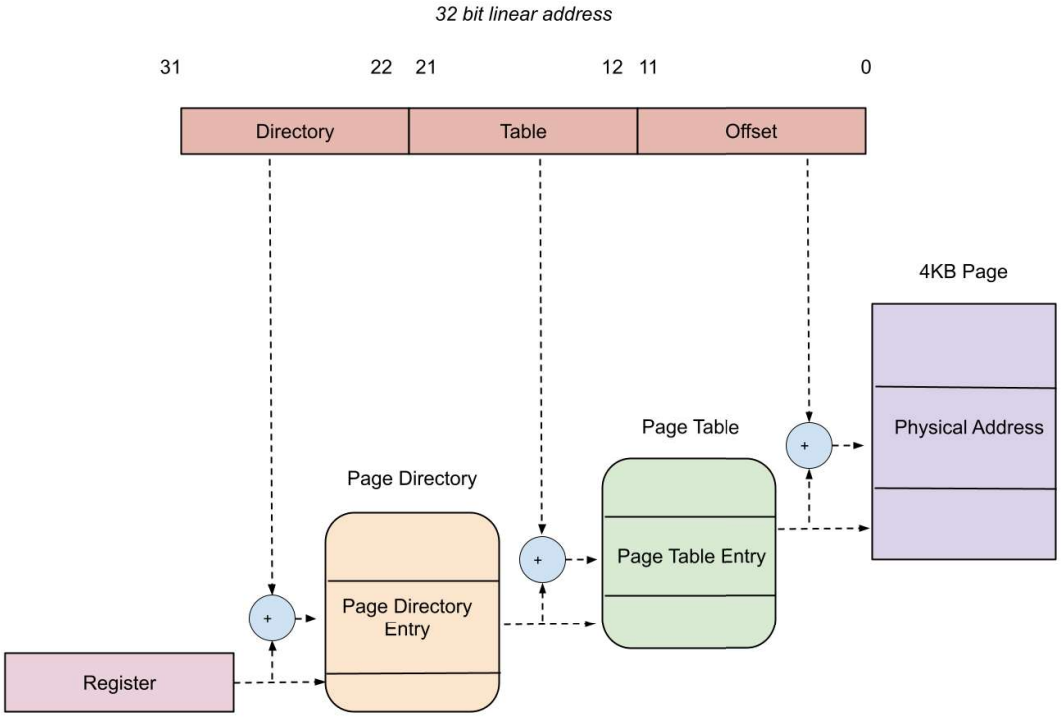

Figure 4.5 Methods of 32-bit linear addressing [2].

After the virtual page is obtained, it is mapped to the core area. The atomic operation has been implemented since kernel 2.6.37 and above. Otherwise, the core map is performed non-atomically. "kmap_atomic()" [73] works without the need for spinlock. "kmap_atomic" definition:

```
"void *kmap_atomic(struct page *page, enum km_type idx);"
```

The slot numbers not shared among the processors are determined after the type entry, and the operation is performed quickly and without splitting into a single processor. The idx value determines the processor slot used in kmap atomic execution. Since it is known that the process will not be blocked by another process, "smp_processor_id()" is used for idx calculation. "idx" The calculation was made as follows:

```
"idx = type + KM_TYPE_NR * smp_processor_id()"
```

After the paging, the relevant page is written to the file path with VFS (Virtual File System). In Kernel 2.3 and 4.14 version ranges, the memory access distance had to be changed to access the parameters on the user side in case of a system call requirement by

the kernel. The method to be used for this is standard. "get_fs" with the temporary current access limit and "set_fs" limit has been extended. The relevant call was made, and the limit was set to temp again. In addition to these; user access range "0x3FFFFFFF - 0xFFFFFFFF" is limited to 3GB. To achieve this, the address limit of the active thread is "0xFFFFFFFF" is set to. The macro used for this is as follows:

```
"cur_thread_info()->addr_limit = 0xFFFFFFFF"
```

4.2.7 Results

Following these steps, kernel builds were taken for different kernel versions on the machine with Ubuntu 20.04 operating system. The computer is 8GB RAM 4 cores and its architecture is amd64. In the kernel versions that were built later, the build of the application was taken. A Makefile has been written to compile the module and it is compiled with the "make" command. After that, the built module "insmod" is uploaded to the system. After these were done successfully, the application was run, and it was observed that the RAM Image output was obtained. Ram image file is created in lime format.

Command:

```
"sudo insmod binalyze-$(uname -r).ko \
    "path=/tmp/mem.lime format=lime""
```

With the "lsmod" command, it was confirmed that the module was created. Since the "lsmod" command will list all existing modules, the output of the "lsmod" command is piped to the "grep" command to see the binalyze module. The "grep" command performs a search. If there is a matching item, it prints the result to the screen.

Command:

```
lsmod | grep binalyze
```

For 4 kernel versions, in this case, it is transferred to the table:

Table 4.4 Test result printout.

Kernel	Build	Insmod	Ram Image
5.4.0-107-generic	Success	Success	Success
5.10.0-11-amd64	Success	Success	Success
4.4.0-142-generic	Success	Success	Success
3.3.4-5.fc17.x86	Success	Success	Success

The driver, developed in Kernel 2.6 and the current 5.18 version ranges, works without problems. Loadable Kernel Module flow chart:

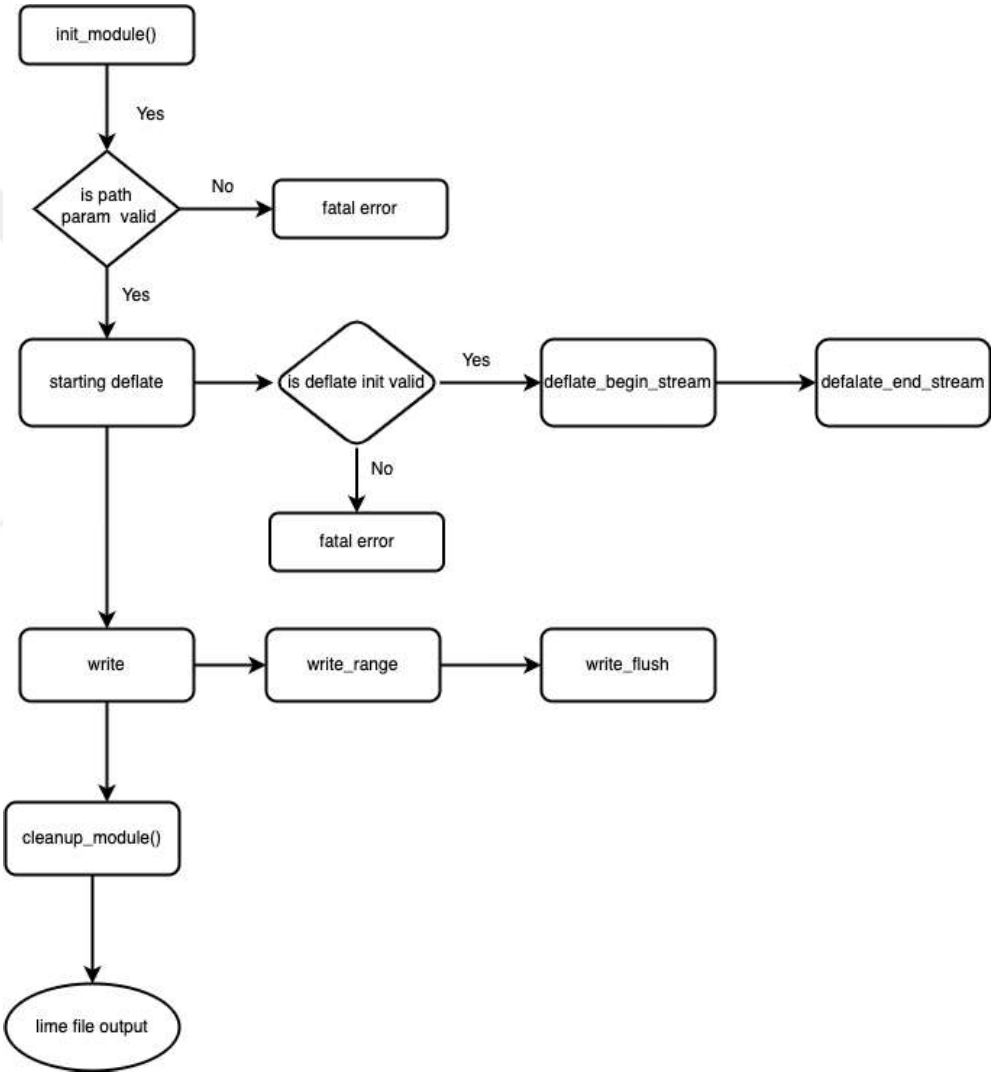

Figure 4.6 Kernel module flow chart.

Figure 4.7 shows "`modinfo`" output of the loaded driver module. As seen in Figure 4.7, GPL(General Public License) is used as the license. In the vermagic part, it is seen that SMP(Symmetric Multi Processing) mode configurations are done successfully. For the compilation to be successful, the kernel "`CONFIG_SMP`"

configuration must be made. Otherwise, the compilation will fail. The retpoline output indicates that the kernel module is protected for Spectre vulnerability. No action has been taken for this. While the module is being compiled, the compilation commands are done internally automatically.

```
filename:       /home/cmaestro/Documents/ramimage/ramimage.ko
description:    RAMIMAGE DRIVER MODULE
author:         BINALYZE LLC
license:        GPL
srcversion:     87FA828E6B5B55AF13E41FF
depends:
retpoline:      Y
name:           ramimage
vermagic:       5.11.0-22 generic SMP mod unload modversions
```

Figure 4.7 Modinfo output for the kernel module.

The application runs smoothly on operating systems used on modern computers such as Ubuntu, Debian, Red-hat, Fedora, Centos, and SuseLinux using the Linux kernel. Intel-protected mode architecture (IA32) is available on all. In addition, the Wind River Linux operating system belonging to VxWorks [74], one of the real-time operating systems, is an operating system found in embedded systems. It uses the Linux kernel as its kernel. A specialized project for embedded systems called Linux-yocto [75] is used as a kernel and uses the paging technique of protected mode architecture as in other Linux kernels. The application runs successfully on the Wind River Linux LTS 10.21.20.15 operating system [76]. Wind River Linux LTS 21 operating system uses Linux LTS 5.10 kernel version [77].

CHAPTER 5

CONCLUSION

In this research, a kernel module has been developed using the paging technique in a Linux-based operating system. Kernel development has been done in two stages. The first stage is filtering addresses mapped to the system RAM resource. At this stage, different filtering methods are available. The first method compares the source's registered name in the "proc" file system via string. The second method is the bitwise comparison of flag information of enumerated resource structures. The third method is to read the address range by the user and transmit it to the driver as "modparam". The second method is preferred because the "IORESOURCE_SYSTEM_RAM" macro used to mask and filter the bits in the resource flag provides ease of use. The second stage is the conversion of the physical addresses in the "PAGE_SIZE" size of the system memory with the help of PFN (Physical Page Frame Number) and the related virtual page address. At this stage, there are two methods. The first method is iteratively added with the source range start to address "PAGE_SIZE". The second method creates a callback with "walk_system_ram_range". The second method is not passed to the kernel; the procedure address "System.map" is read and given to the driver or "kallsyms_lookup_name()". Since the second situation is riskier because it is the process of sending parameters to the driver, the first method was preferred. After the paging, the relevant page is written to the file path with VFS (Virtual File System). Since the kernel allows atomic operations, a mutex is not used. The ram image file created after the program runs is in lime format and is the same size as the RAM. The reason for using the lime format is to provide convenience to cyber forensic engineers who will research the file. With tools such as Volatility, it is possible to search on a lime formatted file with the help of many plugins. The development was done on a computer with Ubuntu 20.04 operating system, 8GB Ram, and four cores.

Thus, unlike the memdump, volatilitux, memmap, crash and memgrep tools; Memory dumping is done by the kernel. This made it possible to access the memory pages that are mapped to memory by the kernel and cannot be accessed from user space. Thus, It

is aimed to contribute to the detection of cyber crimes by taking a copy of RAM. The development of RAM image analysis tools also increases the importance of RAM Image acquisition. In the developed kernel module, the kernel version range has been kept broad, ensuring that it can work in different kernel versions. This has increased the number of Linux systems covered. C programming language was used in the research due to its proximity to low-level operations. This made it easier to use kernel macros.

During the research, two main problems were encountered:

A partial ram image is obtained by freeing up the space of all processes running on the processor by the user process, or "/dev/mem" system memory is mapped, but access to this area is restricted as of kernel version 4.16. This restriction caused the RAM image to be automatically retrieved to be missing. This restriction is made for security reasons. Kernel 4.17 and higher kernel versions are disabled by disabling "CONFIG_STRICT_DEVMEM", and the restriction is overcome.

Another problem is that due to the "insmod" filter, the driver binary built in one kernel cannot enter the system in another kernel version. The driver developed to solve this problem installed the driver in the pre-start phase and when the ram image came. It has been confirmed that the driver was installed via "insmod" .

Future research may focus on performing RAM Image retrieval in different operating systems. As a first step, it is aimed to enable the tool to cover the Android operating system. Afterward, the macOS operating system can be focused on due to the limitations in the operating system and fewer tools than other operating systems. The next step after macOS should be iOS. The diversity in real-time and embedded systems can be increased. Applications can be tested and updated in more real-time and embedded systems, as their usage is increasing day by day, and they are of critical importance.

Another subject to work on is swap space. The operating system can use the disk as memory when the memory is not enough. This space on the disk is called swap space.

The subject of swap space was not mentioned in the research. In future work, studies on swap space will ensure that the operations mapped on the disk are not missing.

Another feature that can be added is that the hash of the RAM Image file is calculated and presented to the user. This feature is important for the user to know if the file has been changed or corrupted. Some tools calculate the hash that comes ready-made among the Linux commands, but this calculation of the developed tool will provide convenience to the users.

REFERENCES

[1] Harris D. M. and Harris S. L., "3 - sequential logic design," in *Digital Design and Computer Architecture (Second Edition)*, second edition ed., Harris D. M. and Harris S. L., Eds. Boston: Morgan Kaufmann, 2013, pp. 108–171. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123944245000033

[2] Jacob B. and Mudge T., "Software-managed address translation," in *Proceedings Third International Symposium on High-Performance Computer Architecture*, 1997, pp. 156–167.

[3] Luthfi A. N. and Prayudi Y., "Process model of digital forensics readiness scheme (dfrs) as a recommendation of digital evidence preservation," *2015 Fourth International Conference on Cyber Security, Cyber Warfare, and Digital Forensic (CyberSec)*, pp. 117–122, 2015.

[4] Pollitt M., "A history of digital forensics," in *Advances in Digital Forensics VI*, Chow K.-P. and Shenoi S., Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 3–15.

[5] Pătraşcu A. and Patriciu V.-V., "Beyond digital forensics. a cloud computing perspective over incident response and reporting," in *2013 IEEE 8th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, 2013, pp. 455–460.

[6] Osbourne G., "Memory forensics: Review of acquisition and analysis techniques." [Online]. Available: https://apps.dtic.mil/sti/citations/ADA594490

[7] Nyholm H., Monteith K., Lyles S., Gallegos M., DeSantis M., Donaldson J., and Taylor C., "The evolution of volatile memory forensics," *Journal of Cybersecurity and Privacy*, vol. 2, no. 3, pp. 556–572, 2022. [Online]. Available: https://www.mdpi.com/2624-800X/2/3/28

[8] Kamal K. M. A., Alfadel M., and Munia M. S., “Memory forensics tools: Comparing processing time and left artifacts on volatile memory,” in *2016 International Workshop on Computational Intelligence (IWCI)*, 2016, pp. 84–90.

[9] LIME, “About lime,” 2022, liME (formerly DMD) is a Loadable Kernel Module (LKM), which allows the acquisition of volatile memory from Linux and Linux-based devices, such as those powered by Android. [Online]. Available: https://github.com/504ensicsLabs/LiME

[10] A. A. Suzen K. T. and Kucuksille E. U., “Development of kernel mode ram driver for ram image on windows,” *Suleyman Demirel University, Journal of Natural and Applied Sciences ,Volume 23, Issue 2*, p. 498–504, 2019.

[11] Velocidex W., “Velocidex/winpmem: The multi-platform memory acquisition tool.” [Online]. Available: https://github.com/Velocidex/WinPmem

[12] Deland-Han, “Read small memory dump files - windows client.” [Online]. Available: https://learn.microsoft.com/en-us/troubleshoot/windows-client/performance/read-small-memory-dump-file

[13] “Ram memory dumper tool.” [Online]. Available: https://www.cybertest.com/physical-ram-memory-dumper

[14] “belkasoft ram capturer: volatile memory acquisition tool.” [Online]. Available: https://belkasoft.com/ram-capturer

[15] “Google code archive - long-term storage for google code project hosting.” [Online]. Available: https://code.google.com/archive/p/pmem/wikis/OSXPmem.wiki

[16] Urrea J. M., “An analysis of linux ram forensics,” Ph.D. dissertation, 2006.

[17] Microsoft, “Microsoft/avml: Avml - acquire volatile memory for linux.” [Online]. Available: https://github.com/microsoft/avml

[18] “memdump.” [Online]. Available: https://manpages.ubuntu.com/manpages/bionic/en/man1/memdump.1.html

[19] Volatilityfoundation, “Volatilityfoundation/volatility: An advanced memory forensics framework.” [Online]. Available: https://github.com/volatilityfoundation/volatility

[20] ——, “Volatilityfoundation/community: Volatility plugins developed and maintained by the community.” [Online]. Available: https://github.com/volatilityfoundation/community

[21] SecComm, “Seccomm/volatilitux: Volatilitux is a memory forensics framework to help analyzing linux physical memory dumps.” [Online]. Available: https://github.com/SecComm/volatilitux

[22] Citypw, “Citypw/lcamtuf-memfetch: Memfetch is a simple utility to dump all memory of a running process, either immediately or when a fault condition is discovered. it is an attractive alternative to the vastly inferior search capabilities of many debuggers and tracers - and a convenient way to grab "screenshots" from many types of text-based interactive utilities.” [Online]. Available: https://github.com/citypw/lcamtuf-memfetch

[23] Crash-Utility, “Crash-utility/crash: Linux kernel crash utility.” [Online]. Available: https://github.com/crash-utility/crash

[24] Eras, “Eras/memgrep: Tool for grepping the memory of processes.” [Online]. Available: https://github.com/eras/memgrep

[25] “Understanding crash dump files,” Feb 2022. [Online]. Available: https://techcommunity.microsoft.com/t5/ask-the-performance-team/understanding-crash-dump-files/ba-p/372633

[26] Domars, “Small memory dump - windows drivers.” [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/small-memory-dump?redirectedfrom=MSDN

[27] ——, “Complete memory dump - windows drivers.” [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/complete-memory-dump?redirectedfrom=MSDN

[28] Ligh M. H., Case A., Levy J., and Walters A., *The art of memory forensics: Detecting malware and threats in windows, linux and mac memory*. Wiley, 2014.

[29] "Welcome to the fireeye market." [Online]. Available: https://fireeye.market/

[30] Hörz M., "Hxd - freeware hex editor and disk editor." [Online]. Available: https://mh-nexus.de/en/hxd/

[31] n0fate, "N0fate/volafox: Mac os x memory analysis toolkit." [Online]. Available: https://github.com/n0fate/volafox

[32] Wang Y., *Operating Systems*, 06 2004, pp. 133.1–15.

[33] AYDIN S. H., *Linux İşletim Sistemi*. Ankara: Orta Doğu Teknik Üniversitesi Bilgi İşlem Daire Başkanlığı, 2002.

[34] Milenkovic M., *Operating Systems: Concepts and Design*. USA: McGraw-Hill, Inc., 1992.

[35] White C., "Real time operating systems - 2023." [Online]. Available: https://tr.strephonsays.com/time-sharing-and-vs-real-time-operating-system-9629

[36] "Types of operating systems," Sep 2021. [Online]. Available: https://www.geeksforgeeks.org/types-of-operating-systems/

[37] Levy E. and Silberschatz A., "Distributed file systems: Concepts and examples," *ACM Comput. Surv.*, vol. 22, no. 4, p. 321–374, dec 1990. [Online]. Available: https://doi.org/10.1145/98163.98169

[38] Kerrisk M., *The Linux Programming Interface*. San Fransisco: No Starch Press, Inc., 2010.

[39] Multicians Project, "About multics," 1969, multics is a operating system project. [Online]. Available: multicians.org/about-multics.html

[40] Abdullahi I., "Process scheduling in longest job first (ljf) algorithm. a proposed framework for starvation problem," Ph.D. dissertation, 01 2012.

[41] Burguera I., Zurutuza U., and Nadjm-Tehrani S., "Crowdroid: Behavior-based malware detection system for android," 10 2011, pp. 15–26.

[42] Sârbu C., Johansson A., Suri N., and Nagappan N., "Profiling the operational behavior of os device drivers," *Empirical Software Engineering*, vol. 15, pp. 380–422, 11 2008.

[43] Lu L., Arpaci-Dusseau A. C., Arpaci-Dusseau R. H., and Lu S., "A study of linux file system evolution," in *11th USENIX Conference on File and Storage Technologies (FAST 13)*. San Jose, CA: USENIX Association, Feb. 2013, pp. 31–44. [Online]. Available: https://www.usenix.org/conference/fast13/technical-sessions/presentation/lu

[44] Rodeh O., Bacik J., and Mason C., "Btrfs: The linux b-tree filesystem," *ACM Trans. Storage*, vol. 9, no. 3, aug 2013. [Online]. Available: https://doi.org/10.1145/2501620.2501623

[45] "Linux proc man page." [Online]. Available: https://man7.org/linux/man-pages/man5/proc.5.html

[46] Chapelle G., "A practical guide to linux commands, editors, and shell-programming, third edition by mark g. sobell," *SIGSOFT Softw. Eng. Notes*, vol. 38, no. 4, p. 38, jul 2013. [Online]. Available: https://doi.org/10.1145/2492248.2492251

[47] geeksforgeeks, "About insmod," 2018, insmod defining. [Online]. Available: https://www.geeksforgeeks.org/insmod-command-in-linux-with-examples/

[48] Peter Jay SALZMAN, Michael BURIAN, O. P., *The Linux Kernel Module Programming Guide*. USA: Open Source, e-book, 2002.

[49] De Goyeneche J.-M. and De Sousa E., "Loadable kernel modules," *IEEE Software*, vol. 16, no. 1, pp. 65–71, 1999.

[50] Bagherzadeh M., Kahani N., Bezemer C.-P., Hassan A. E., Dingel J., and Cordy J. R., "Analyzing a decade of linux system calls," *Empirical Software*

Engineering, vol. 23, no. 3, pp. 1519–1551, Jun 2018. [Online]. Available: https://doi.org/10.1007/s10664-017-9551-z

[51] O'Gorman J., *The linux process manager: The internals of scheduling, interrupts, and signals*. Wiley, 2003.

[52] 0xAX, "Linux-insides/linux-mm-2.md at master · 0xax/linux-insides." [Online]. Available: https://github.com/0xAX/linux-insides/blob/master/MM/linux-mm-2.md

[53] Bharadwaj R., *Mastering linux kernel development: A kernel developer's reference manual*. Packt Publishing, 2017.

[54] Kollár, Ivor, "Forensic ram dump image analyzer," 2010, analyzer tools. [Online]. Available: https://dspace.cuni.cz/handle/20.500.11956/34206

[55] Petroni N. L. W., "Fatkit: A framework for the extraction and analysis of digital forensic data from volatile system memory. digital investigation," *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, p. 197–210, 2006.

[56] Aljaedi A., Lindskog D., Zavarsky P., Ruhl R., and Almari F., "Comparative analysis of volatile memory forensics: Live response vs. memory imaging." 10.1109/PASSAT/SocialCom.2011.68: IEEE, 2011.

[57] Seokhee L. and Hyun-Sang K., "A study of memory information collection and analysis in a view of digital forensic in window system," 2006.

[58] Seo J., Lee, Soekjun, and Taeshik S., "A study on memory dump analysis based on digital forensic tools," *Peer-to-Peer Netw. Appl. 8*, vol. 8, p. 694–703, 2015.

[59] Sylve J., Case A., Marziale L., and Richard G. G., "Acquisition and analysis of volatile memory from android devices," *Digital Investigation*, vol. 8, no. 3, pp. 175–184, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1742287611000879

[60] Dell, “What is pci and pci express?” 2022, what is PCI? [Online]. Available: https://www.dell.com/support/kbdoc/tr-tr/000123517/pci-ve-pci-express-nedir-kb-makalesi-161614

[61] Oracle, “Pci i/o address space,” 2022, pCI. [Online]. Available: https://docs.oracle.com/cd/E19683-01/806-5222/hwovr-30/index.html

[62] Corbet J. and Rubini A., *Linux Device Drivers*. San Fransisco: O’Reilly Media, Inc, 2001.

[63] “Memory mapping.” [Online]. Available: https://linux-kernel-labs.github.io/refs/heads/master/labs/memory_mapping.html

[64] Bovet D. P. and Cesati M., *Understanding the linux kernel*. O’Reilly, 2001.

[65] Shanley T., *Protected Mode Software Architecture*. Harlow, England: Addison-Wesley Professional, 1996.

[66] Mathur S., *Microprocessor 8086: Architecture, Programming and Interfacing*. USA: PHI, 2011.

[67] Brey B., *The Intel Microprocessors 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro Processor, Pentium II, Pentium III, Pentium 4, and Core2 with 64-Bit Extensions Architecture, Programming, and Interfacing*. Upper Saddle River, New Jersey: The Intel Microprocessors 8086, 2009.

[68] Abbott D., “Chapter 3 - introducing linux,” in *Linux for Embedded and Real-Time Applications (Fourth Edition)*, fourth edition ed., Abbott D., Ed. Newnes, 2018, pp. 29–54. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128112779000031

[69] Aslan K., *İntel İslemcileri Korumali Mod Yazılım Mimarisi*. Turkiye: Pusula Yayincilik, 1997.

[70] Gorman M., *Understanding the Linux® Virtual Memory Manager*. Upper Saddle River, New Jersey: Pearson Prentice Hall, 2004.

[71] Patchwork, “About system ram,” 2019, mm resource: let walk systemram range searcchild resources. [Online]. Available: https://patchwork.kernel.org/project/linux-nvdimm/patch/20190225185738.F6C24E62@viggo.jf.intel.com/

[72] Bootlin, “About kallsyms,” 2022, kallsyms.c - kernel/kallsyms.c. [Online]. Available: https://elixir.bootlin.com/linux/latest/source/kernel/kallsyms.c

[73] Kernel, “About high memory,” 2022, high Memory Handling — The Linux Kernel documentation. [Online]. Available: https://www.kernel.org/doc/html/latest/vm/highmem.html

[74] “Vxworks rtos.” [Online]. Available: https://www.windriver.com/products/vxworks

[75] WindRiver-Labs, “Windriver-labs/wr-kernel: Wr-kernel layer for wind river linux.” [Online]. Available: https://github.com/WindRiver-Labs/wr-kernel

[76] [Online]. Available: https://docs.windriver.com/bundle/Wind_River_Linux_Getting_Started_LTS_22_tki1589820771450/page/mmo1403548787756.html

[77] Wind-River-Blog-Network, “Wind river linux lts 21 now available,” Oct 2021. [Online]. Available: https://blogs.windriver.com/wind_river_blog/2021/06/wind-river-linux-lts-21-now-available/

[78] Aljaedi A., Lindskog D., Zavarsky P., Ruhl R., and Almari F., “Comparative analysis of volatile memory forensics: Live response vs. memory imaging,” in *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*, 2011, pp. 1253–1258.

[79] Firoozjaei M. D., Lashkari A. H., and Ghorbani A. A., “Memory forensics tools: a comparative analysis,” *Journal of Cyber Security Technology*, vol. 6, no. 3, pp. 149–173, 2022. [Online]. Available: ttps://doi.org/10.1080/23742917.2022.2100036

[80] Bmc-Msft, “How-to capture an image.” [Online]. Available: https://learn.microsoft.com/en-us/security/research/project-freta/how-to-capture-an-image

[81] [Online]. Available: https://www.binary-zone.com/2019/06/20/acquiring-linux-memory-using-avml-and-using-it-with-volatility/

[82] Microsoft, "Error while running on amazon linux 2 microsoft/avml." [Online]. Available: https://github.com/microsoft/avml/issues/18

[83] Barrygolden, "Mmmapiospace function (wdm.h) - windows drivers." [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/nf-wdm-mmmapiospace

[84] "Ftk imager." [Online]. Available: https://www.exterro.com/ftk-imager

[85] Zeltser L., the AuthorI design security solutions A., shepherd them to a sustainable state. I used to be hands-on in many areas of cybersecurity, focus on strategy I. N. I., and leadership, "One-click windows memory acquisition with dumpit," Jan 2017. [Online]. Available: https://zeltser.com/memory-acquisition-with-dumpit-for-dfir-2/

[86] "Alison kim," Dec 2022. [Online]. Available: https://www.sans.org/blog/mac-os-forensics-how-to-simple-ram-acquisition-and-analysis-with-mac-memory-reader-part-

[87] Poling J., "Osx (mac) memory acquisition and analysis using osxpmem and volatility." [Online]. Available: https://ponderthebits.com/2017/02/osx-mac-memory-acquisition-and-analysis-using-osxpmem-and-volatility/

[88] "Dc3dd: Kali linux tools," Nov 2022. [Online]. Available: https://www.kali.org/tools/dc3dd/

[89] Matteo A., "Simple forensics imaging with dd, dc3dd; dcfldd," Oct 2020. [Online]. Available: https://www.forensics-matters.com/2020/10/20/simple-forensics-imaging-with-dd-dc3dd-dcfldd/

[90] Ruff N., "Windows memory forensics," *Journal in Computer Virology*, vol. 4, no. 2, p. 83–100, 2007.

[91] Vömel S. and Freiling F. C., "A survey of main memory acquisition and analysis techniques for the windows operating system," *Digital Investigation*, vol. 8,

no. 1, pp. 3–22, 2011. [Online]. Available:
https://www.sciencedirect.com/science/article/pii/S1742287611000508

[92] [Online]. Available: http://halilozturkci.com/
adli-bilisim-volafox-ile-mac-os-x-sistemlerde-hafiza-analizi/

APPENDIX A

This appendix contains information about memory analysis techniques and tools. After giving information about memory analysis techniques, it includes a detailed analysis of the tools that use them.

1. Memory Analyzing

Every application running on the computer uses processes. These processes need space allocated by the computer to run. This memory allocation is done on RAM. Therefore, if you want to get information about the applications running on the computer, you should work on the RAM. With the development of computer systems day by day, there has been an increase in the number of harmful applications, which have become more and more threatening to the security of the systems.

Therefore, it is necessary to detect these dangerous applications and take the required precautions. This has revealed the science of forensics and caused the judicial authorities to shift their direction to digital channels. This has led to the emergence of a sub-discipline called memory forensics. The science of memory forensics provides information about harmful applications by making examinations on RAM. Different analysis techniques have emerged.

1.1 Memory Analyzing Processes

The memory analysis process can be grouped under 3 headings [78].

- Acquisition of memory
- Transferring memory to file by dumping
- Analyzing the information obtained
- Taking action in the light of the information collected

In the memory acquisition technique, the ram is scanned live. The user determines the area they want to examine. It allows for the collection of evidence quickly. The user

can customize his analysis according to his wishes, focusing on various types of evidence such as MFT, RAM Image, Prefetch Files, Shim Cache.

The memory dumping technique is different from the live analysis of RAM. Dumping takes the pages in RAM and saves them to a file. Since the application runs on user space in a live investigation, the information obtained is limited to user space. When the application that will create the RAM Image file has a kernel module running in the kernel space, not in the user space, it is also possible to obtain information about the applications mapped to the RAM in the kernel space.

The resulting dump file is not a meaningful file to open and read. Therefore, analysis tools have been developed for this. These tools allow the user to extract meaningful data from the collected evidence. This way, the user can be informed about malicious applications and intervene in the event.

1.2 Memory Analyzing Tools

Different tools use techniques for collecting evidence over memory. The implementation of the methods is different for each operating system. Some tools only do dumping, only acquisition, after the image file is taken, to analyze that file or to perform both dumping and analysis [79].

1.2.1 AVML (Acquire Volatile Memory for Linux)

AVML is a tool that can take RAM Images on systems with 32-bit and Linux operating systems. Developed by Microsoft. Tested on Ubuntu, CentOS, RHL, Debian, Oracle Linux, and CBL-Mariner. It tries to access the /dev/crash, /proc/kcore, /dev/mem files on Linux, trying to get a RAM Image from the accessible one. If `kernel_lockdown` is enabled, the operation will fail. The output form supports lime format. Page comparison can be made with Snappy. In addition, the output file can be sent to Azure Blob. There is a retry mechanism for this sending to be done successfully.

AVML is written in Rust language, and Rust language dependencies must be installed on the system in order to compile the program [80].

```
"# Install MUSL
$ sudo apt-get install musl-dev musl-tools musl

# Install Rust via rustup
$ curl https://sh.rustup.rs -sSf | sh -s -- -y

# Add the MUSL target for Rust
$ rustup target add x86_64-unknown-linux-musl

# Build
$ cargo build --release --target x86_64-unknown-linux-musl"
```

When AVML is compiled with these steps, it extracts the binary to the path `"/avml/target/x86_64-unknown-linux-musl/release"`. AVML offers two half file extraction options, compressed and uncompressed [81]. The file extracted using the `"sudo ./avml --compress output.lime.compressed"` command becomes 4.9GB on an 8GB system. The `"sudo ./avml output.lime"` command gives the following output:

```
sedat@ubuntu:~/avml/target/x86_64-unknown-linux-musl/release$ sudo ./avml output.lime
Error: error: unable to read memory
caused by:
    0: unable to create memory snapshot:
        error: unable to create memory snapshot from source: /dev/crash
        caused by:
            0: unable to create memory snapshot
            1: unable to read memory
            2: No such file or directory (os error 2)

        error: unable to create memory snapshot from source: /proc/kcore
        caused by:
            0: unable to create memory snapshot
            1: write block failed: 4294967296..9932111871

        error: unable to create memory snapshot from source: /dev/mem
        caused by:
            0: unable to create memory snapshot
            1: write block failed: 1048576..2952708095
```

Figure 5.1 Uncompressed AVML output error.

This error occurs because the system has insufficient space to create an output file [82]. After making enough room, the output was successfully created. The resulting file is 8GB in size because it is uncompressed. However, the output shows which resources AVML uses and reads block by block from the sources. While it is the

advantage of AVML that it can create compressed files, it is one of the disadvantages of AVML that it uses these three resources in user space. AVML fails on systems where access to these three resources is closed.

1.2.1 Memdump

Memdump tool is developed by IBM and can be installed via Ubuntu packages. It does the dump operation by reading the "/dev/mem" file. memdump takes the dump of physical memory, but with the -k option, it also has the feature of getting the kernel dump. However, this feature only works correctly for some kernel versions. As written on the official page, it produces fake results in kernel 2.2. Since "/dev/mem" is not accessible in hardware firmware such as boot PROM, BIOS, dump operation cannot be performed.

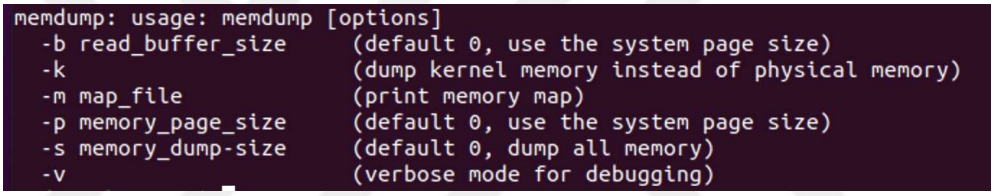

Figure 5.2 memdump usage.

1.2.1 Volatilitux

Volatilitux is a tool developed by the Volatility organization to analyze physical memory. It can run on x86 and ARM machines. It works on Linux and Windows XP and later. Like memdump, Volatilitux also allows the image of both physical and kernel memory to be received. But while memdump cannot load the kernel module, Volatilitux also allows it to load it. Python 2.6 and above must be installed on the system. The disadvantage of this tool is that the offset information to be obtained from the "vm_flags" field is incorrect.

1.2.1 Memfetch

Memfetch produces a process-based output by reading the gold of the "/proc" file. This means that it can only access processes running in user space, as in other tools. Using mmap() in Linux 2.2 kernel version causes the system to hang. This tool, unlike other tools, attaches to a single process. This means that the dump file of the entire

RAM could not be retrieved. It writes the process hex memory information and attaches parts to the screen and 64KB files. This can also be said as a disadvantage.

```
sedat@ubuntu:~/lcamtuf-memfetch$ sudo ./memfetch -h
memfetch 0.05b by Michal Zalewski <lcamtuf@coredump.cx>
Usage: ./memfetch [ -sawm ] [ -S xxx ] PID
  -s      - wait for fault signal before generating a dump
  -a      - skip non-anonymous maps (libraries etc)
  -w      - write index file to stdout instead of mfetch.lst
  -m      - avoid mmap(), helps prevent hanging on some 2.2 boxes
  -S xxx  - dump segment containing address xxx (hex) only
```

Figure 5.3 memfetch usage.

The information written to the screen by the memfetch tool, which is attached to the firefox process with id 171582 using sawm options, is as follows:

```
sedat@ubuntu:~/lcamtuf-memfetch$ sudo ./memfetch  -awm 171582
memfetch 0.05b by Michal Zalewski <lcamtuf@coredump.cx>
[+] Attached to PID 171582 (/usr/lib/firefox/firefox).
PAGE_SIZE is 4096
[*] Writing master information to standard output.

[000] mem-000.bin:
     Memory range 0xb6800000 to 0xb6900000 (1048576 bytes)
     3d2b6800000-3d2b6900000 rw-p 00000000 00:00 0

[S] [001] mem-001.bin:
     Memory range 0x49700000 to 0x49800000 (1048576 bytes)
     5f349700000-5f349800000 rw-p 00000000 00:00 0

[S] [002] mem-002.bin:
     Memory range 0x0fe00000 to 0x0ff00000 (1048576 bytes)
     61b0fe00000-61b0ff00000 rw-p 00000000 00:00 0

[S] [003] mem-003.bin:
     Memory range 0x96600000 to 0x96700000 (1048576 bytes)
     b0596600000-b0596700000 rw-p 00000000 00:00 0

[S] [004] mem-004.bin:
     Memory range 0x45200000 to 0x45300000 (1048576 bytes)
     c4445200000-c4445300000 rw-p 00000000 00:00 0
```

Figure 5.4 memfetch output.

1.2.1 WinPMem

WinPMem is a physical memory acquisition tool with 32-bit and 64-bit support, running on Windows operating systems between WinXP and Win 10. It has the opportunity to work both on the userspace side and as a drive. It has the opportunity to work both on the userspace side and as a drive. WinPMem does not support the Lime file format. It only supports the raw file format. WinPMem also supports the MmMapIoSpace [83] method. It allows the acquisition of ram image using this

method.

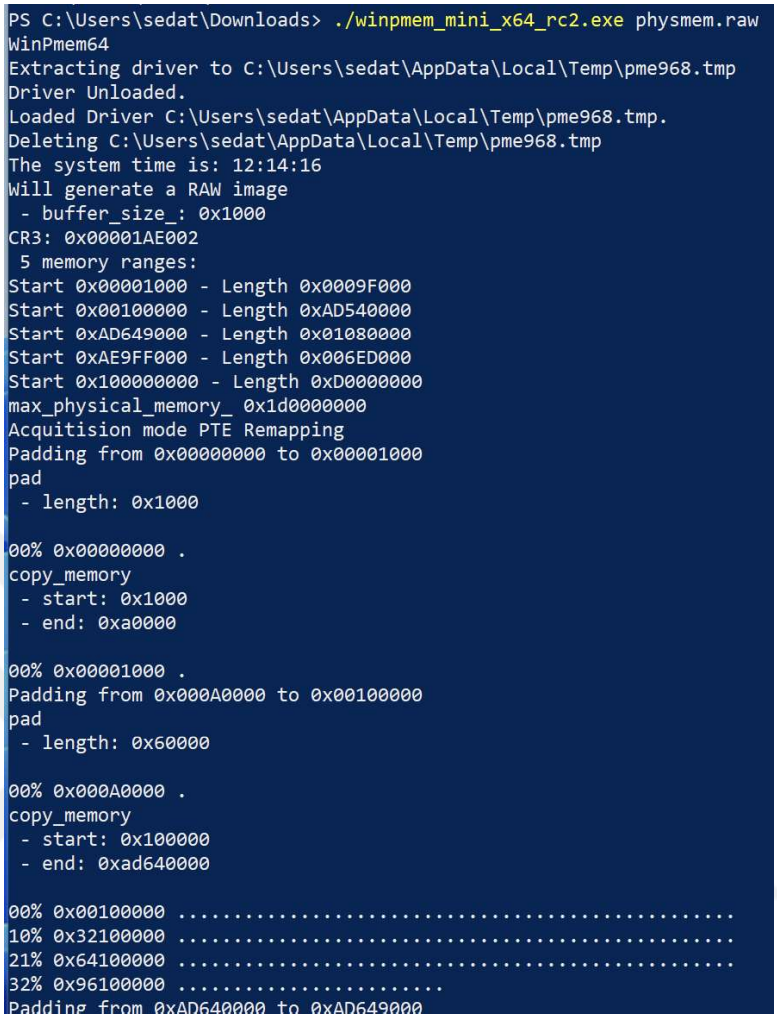

Figure 5.5 winpmem output.

As seen in Figure 5.5, the exe builds a driver and installs it on the system. It writes memory ranges and padding information to the screen. It creates an 8GB dump file on Windows 11 with 8GB RAM. After the process, the program automatically unloads the driver it has installed. In the Windows operating system, this process is much faster than the dumping processes in the Linux operating system.

1.2.1 Belkasoft Live RAM Capturer

Belkasoft is a tool that enables volatile memory capture by offering 32-bit and 64-bit support. It works on machines with the Windows operating system. It creates a .mem file. In addition, unlike other tools, it also allows for analysis of the output file.

It works in kernel space by generating a Belka driver, as shown in Figure 5.6. Page

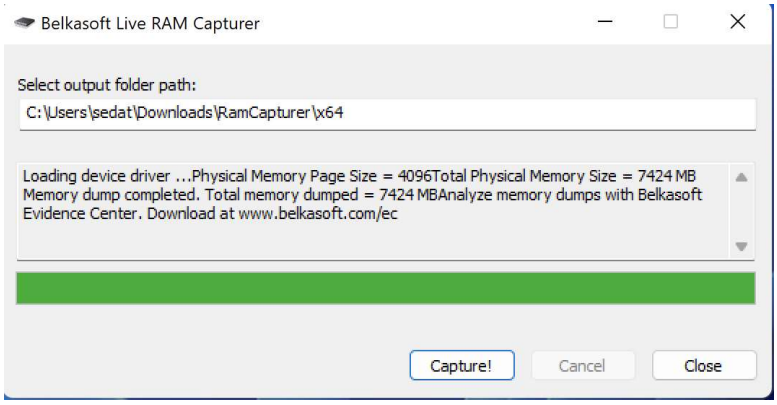

Figure 5.6 belka output.

size appears as 4096. The total RAM size of the machine is 7424MB, and Belka also created a 7424MB file with a .mem extension.

1.2.1 Ftk Imager

A single file, folders, zip drives, CDs, and DVDs may all be imaged with the free Ftk Imager program from the AccessData business. FTK Imager also enables previewing of digital evidence in specific settings without capturing an image. It also offers a disk image analysis.

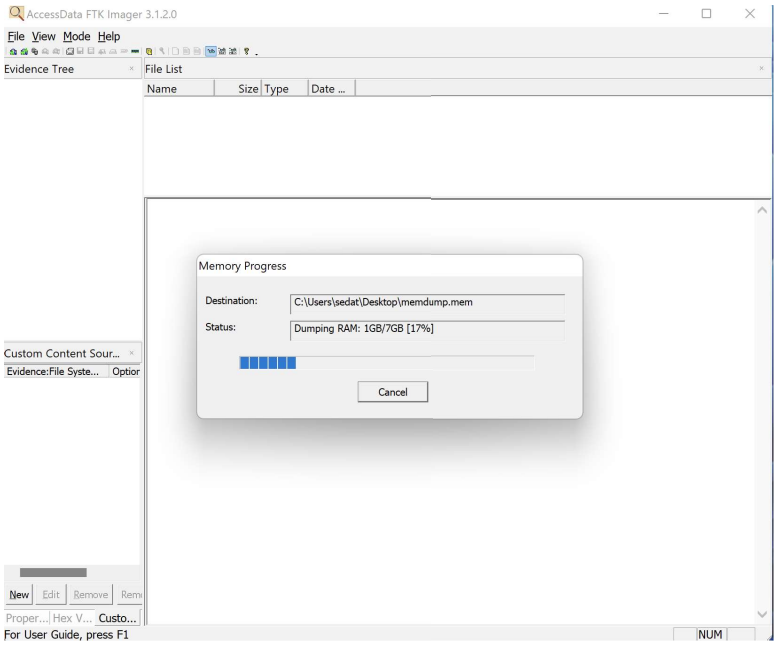

Figure 5.7 Ftk Imager output.

As seen in Figure 5.7, it creates a 7GB dump image file. It also can create page files. It also has the feature of encrypting the image files to be analyzed with MD5 and

Sha1. In addition to being used on machines with the Windows operating system, some versions also work on Ubuntu, Fedora, and Mac [84]. Unlike others, Ftk Imager has the feature of recovering deleted files if no other data has been written on them.

1.2.1 LiMe

LiMe is a tool to get a ram image by loading it into the kernel as a kernel module in Linux-based systems. It is the most successful tool in Linux-based systems among commercial tools. It can create files in 3 different image formats raw, lime, and padded. It has the feature of encrypting the resulting file and getting its hash. It can perform acquisition on the network interface. As of Linux kernel version 2.6.35, it can provide a timeout feature [9].

1.2.1 Dumpit

DumpIt software was a free software developed by Moonsols company that can be used to take images of the physical memory of both 32-bit and 64-bit Windows operating systems. The DumpIt.exe application is an application used to take a physical image of the memory. This application can also be used in the process of obtaining a memory image on the suspicious system by copying it to a USB memory. The DumpIt app does not have any analysis capabilities [85].

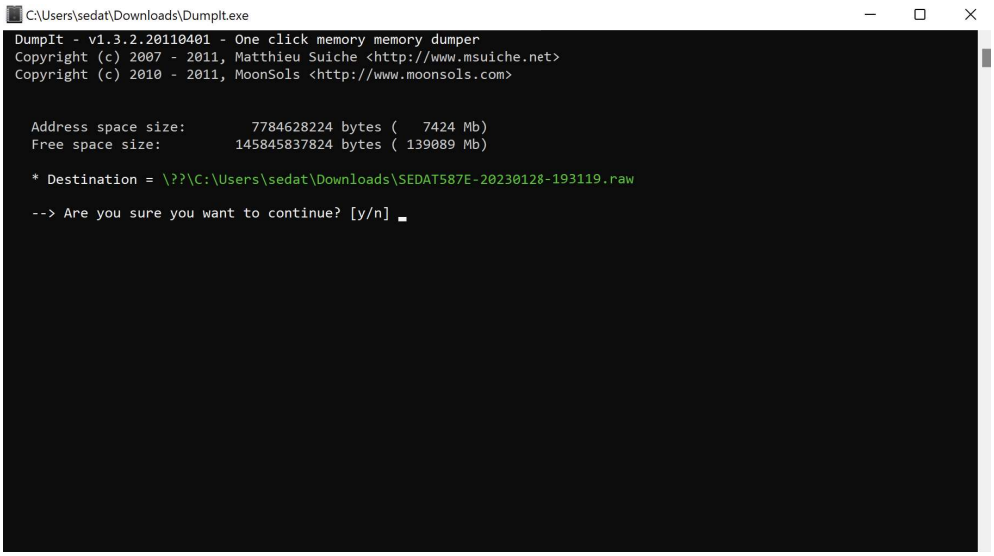

Figure 5.8 Dumpit output.

1.2.1 Memory MAC Reader

An application for Mac's command line that can be downloaded for free is called Mac Memory Reader. The utility generates a dump file in Apple's Mach-O format that contains the offsets and lengths of every accessible chunk of physical RAM and outputs it to a USB drive or any other associated device [86]. Usage:

```
"sudo ./MacMemoryReader -v -H SHA-256 tactical-ram.img"
```

The -v command is a command that allows displaying the info and debugs level steps. The -H command is an option to encrypt the image file. MD5 and SHA formats are supported as encryption algorithms.

1.2.1 OSXPMem

OSX Memory Imager is an open-source tool for obtaining physical memory on an Intel-based Mac. It does not work on ARM-based MACs. Osxpmem works in both user space and kernel space. Access to physical memory is provided with the pmem.kext file in the project. The tool is reading from "/dev/pmem". Because MAC operating systems are more restrictive than other operating systems, certain pitfalls can be encountered while using the tool. While untaring toll on the official page, it is stated that it must be done with "sudo su".

OsxPMem supports Mach-O, ELF and zero-padded RAW file formats. Default format is ELF but can be selected as an option [87].

```
"$ sudo su"
"$ tar xvf OSXPMem.tar.gz"
"$ cd OSXPMem"
"$ ./osxpmem memory.dump"
"$ ./osxpmem --format mach memory.dump"
```

1.2.1 Dc3dd

Unlike other tools, dc3dd allows the files in the system to be saved in image format. This enables the analysis of the file with the tools that perform image analysis. Like other image techniques, there is no possibility of live acquisition on RAM. But, for example, the image of the /proc file can be taken. dc3dd, which is used more in the Kali community, is also available in Ubuntu. It is specialized for the Linux operating system [88].

```
"sudo dc3dd if=/var/dev/sda1 of=/tmp/dc3dd hash=sha512"
```

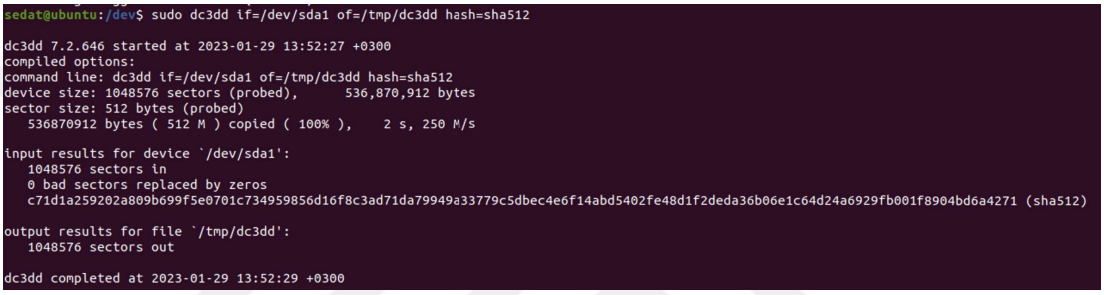

Figure 5.9 Dc3dd output.

Command options:

- if=/dev/sda1 is the source in this case is sda1
- of=/tmp/dc3dd is where the output file is saved
- log=image.log is the output path for the log
- hash=sha256 on the fly hashing algorithm
- bs=4k is the block size [89]

1.3 RAM Image Files Analysing Tools

For forensics engineers, the process doesn't end with getting a RAM Image. The resulting RAM Image files should be examined. Since these files are not files that can be opened and read and are large files, additional tools are used for analysis. Some tools allow analysis for different file formats.

There are also certain techniques in file scanning. These are string search and process object search. Those working in forensic science make username and password searches and network address searches. String search is an analysis method that can easily access them.

The other method is the analysis method related to the processes. For this, detection can be made on process objects and signatures. Analysis is done in the EPROCESS structure for Windows [90]. However, these structure analyzes are difficult to detect as they vary between Windows versions. In signature scanning, it is based on the logic of finding an unwanted signature in the signature pool [91].

1.3.1 Volatility

The Volatility tool is the most widely used commercial tool and allows the analysis of RAM image files in lime and raw formats.

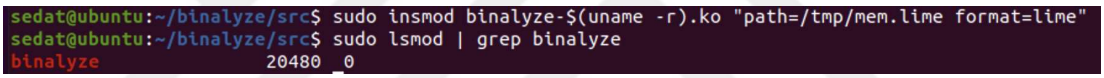

Figure 5.10 Insmod kernel module.

Two different products are offered, namely Volatility2 and Volatility3. Since Volatility3 is newer, Volatility3 was used to analyze the 8GB file extracted by the kernel module. Volatility provides ready-made plugins to each operating system for analysis. Using these plugins, the RAM Image file was analyzed.

The first of these plugins is the "`Linux.lsmod.Lsmod`" plugin enables the lsmod command in Linux to be run in the RAM Image file. This plugin lists the kernel modules saved in the RAM Image file. When this command is run combined with the grep command, the loaded kernel module must be found. As seen in Figure 5.9, the kernel module named binalyze was found in the RAM Image file.

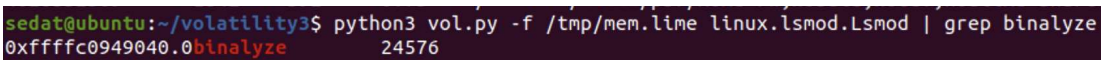

Figure 5.11 Lsmod plugin output.

The other command is the pslist command in Linux. Volatility offers a plugin to run the pslist command on the file. Firefox was open when the module was running, so firefox output should be seen when the pslist command is grappled with Firefox.

```
sedat@ubuntu:~/volatility3$ python3 vol.py -v -f /tmp/mem.lime linux.pslist.PsList | grep firefox
INFO    volatility3.cli: Volatility plugins path: ['/home/sedat/volatility3/volatility3/plugins', '/home/sedat/volatility3/volatility3
/framework/plugins']
INFO    volatility3.cli: Volatility symbols path: ['/home/sedat/volatility3/volatility3/symbols', '/home/sedat/volatility3/volatility3
/framework/symbols']
INFO    volatility3.framework.automagic: Detected a linux category plugin
INFO    volatility3.framework.automagic: Running automagic: ConstructionMagic
INFO    volatility3.framework.automagic: Running automagic: SymbolCacheMagic
INFO    volatility3.framework.automagic: Running automagic: LayerStacker
INFO    volatility3.schemas: Dependency for validation unavailable: jsonschema
INFO    volatility3.framework.automagic: Running automagic: SymbolFinder
INFO    volatility3.framework.automagic: Running automagic: LinuxSymbolFinder
INFO    volatility3.schemas: Dependency for validation unavailable: jsonschema
INFO    volatility3.framework.automagic: Running automagic: KernelModule
0x9c4987d29900  403974  403974  2079    firefox
```

Figure 5.12 Pslist plugin output.

Another plugin allows the mountinfo command to be run. The Mouninfo command lists what is mounted on Linux. Since the working system is a virtual machine and files can be mounted from the original Mac device to the Ubuntu virtual machine, it is expected that the home file will be mounted and the home file will be seen as output.

```
sedat@ubuntu:~/volatility3$ python3 vol.py -f /tmp/mem.lime linux.mountinfo.MountInfo | grep media
4026531841 100.0980     29      0:53king/attempt/media/psf/iCloud       rw,nosuid,nodev,noatime shared:546      prl_fs
4026531841      1516    29      0:72    /       /media/psf/AllFiles     rw,nosuid,nodev,noatime shared:714      prl_fs
w,sync
4026531841      1839    29      0:73    /       /media/psf/Home rw,nosuid,nodev,noatime shared:727      prl_fs  Home
4026532239      1000    91      0:53    /       /media/psf/iCloud       rw,nosuid,nodev,noatime shared:556 master:546
w,sync
4026532239      1838    91      0:72    /       /media/psf/AllFiles     rw,nosuid,nodev,noatime shared:726 master:714
s       rw,sync
4026532239      1866    91      0:73    /       /media/psf/Home rw,nosuid,nodev,noatime shared:739 master:727   prl_fs
4026532366      992     548     0:53    /       /media/psf/iCloud       rw,nosuid,nodev,noatime shared:552 master:546
w,sync
4026532366      1813    548     0:72    /       /media/psf/AllFiles     rw,nosuid,nodev,noatime shared:725 master:714
s       rw,sync
4026532366      1864    548     0:73    /       /media/psf/Home rw,nosuid,nodev,noatime shared:738 master:727   prl_fs
4026532367      990     637     0:53    /       /media/psf/iCloud       rw,nosuid,nodev,noatime shared:551 master:546
w,sync
4026532367      1749    637     0:72    /       /media/psf/AllFiles     rw,nosuid,nodev,noatime shared:724 master:714
```

Figure 5.13 Mountinfo plugin output.

1.3.2 Mandiant Memoryze

It is a tool developed by FireEye company that allows to analyze dump files running on Windows. It is specialized for detecting malware from dump files. Besides, it can scan the full system memory range. It can encrypt the output file with hash algorithms. Besides memory scanning, it can also scan full disk. It can make forensic reporting in 12 TCP states.

1.3.2 Volafox

Volafox is open-source analysis software used to analyze memory images taken from Mac OS X systems. This software can analyze memory images taken from Snow Leopard, Lion, Mountain Lion, and Mavericks systems. The memory image to be analyzed must be in flat format. If the memory image was taken with the Mac Memory Reader application, then the relevant memory image needs to be converted.

The flatten.py script that comes with Volafox is used for this process. Usage:

```
"flatten.py <MAC MEMORY READER IMAGE> <FLAT IMAGE>"
```

Volafox needs the overlay files for the corresponding Mac OS X version to perform analysis processing on the memory file for that Mac OS X system. Volafox can be used after this overlay file is created. The most straightforward command that can be run with the Volafox tool is the "system_porfiler" command. This command gives information about the respective MacOS and how many CPUs and memory it had when it was last hibernated.

Another command is the mount command. This command lists the mounted drives on the MacOS system when the relevant memory image is taken, the directory where these drives are mounted, and the file system types on these partitions. Usage [92]:

```
"python vol.py macos-mem.vmem -o mount"
```

The ps command determines which processes are running on the relevant MacOS when the memory image is taken. In light of the information obtained in the output of this command, it can be determined whether unwanted and suspicious applications are running on the system.

```
"python vol.py macos-mem.vmem -o ps"
```

Another important command is the lsof command. This command provides information about which files the specified process is using. With the -p option, it is determined which process will be accessed.

```
"python vol.py macos-mem.vmem -o lsof -p 277"
```

1.3.2 wxHexEditor