



Bahir Dar University

Software engineering department

Documentation on OpenBSD

Name: Mezaselasie Tadele
ID No: 1601988
Course: OSSP
Submitted to: Wendimu B.

Contents

1. Introduction (Background and Motivation).....	3
2. OBJECTIVE	4
3. Requirement.....	4
4. Installation Process.....	5
Conclusion	9
5. Issues and problems	9
6. File System Support	11
7. Advantage and Disadvantage.....	11
Advantage of openBSD	11
Disadvantage of openBSD.....	12
8. Conclusion	12
9. Future Outlook and Recommendation	12
10. Virtualization in modern operating system.....	13
11. System Call Implementation	14
References	17

1. Introduction (Background and Motivation)

Operating systems are the backbone of computing because they manage system resources like memory, processes, storage, and devices while facilitating user-hardware communication. Their stability and security determine the reliability of any computer system as they deliver the foundation in which all applications run. As a result of its design philosophy, modularity, and portability, UNIX-based systems continue to be the most meaningful among the vast number of operating systems prevalent today. Among the most popular UNIX-like operating systems, OpenBSD is one of the derivatives of the Berkeley Software Distribution (BSD) and is distinguished by its focus on security, correctness, and code simplicity. OpenBSD is held in an extremely high reputation by academic communities, research communities, security communities, and cyber security communities but not widely used by user communities compared to Linux and Windows. OpenBSD being clean and open in its code base, is an excellent platform through which one can learn The fundamental ideas behind how an operating system functions including memory management, process scheduling, system calls, and user permissions. It is based on standard UNIX fundamentals and therefore gives excellent insight into the design and operation of secure and efficient operating systems.

Learning OpenBSD does not just teach us design of the operating system but also focuses on security and simplicity in systems today. OpenBSD's strict concern for correctness and openness of code provides a unique glimpse at how a secure and effective operating system is created from scratch. By starting from its modularity and examining its internal mechanisms, it becomes clear how the operating system manages to balance user requirements with system performance. OpenBSD's design invites the user to think in terms of performance, security, and maintainability at depth and presents a rich learning experience for anyone who wishes to learn about the inner workings of operating systems.

2. OBJECTIVE

The primary objective of this assignment was to install and set up the OpenBSD operating system and acquiring real hands-on experience with system internals of this OS. Specifically, the assignment focused on developing a better understanding of UNIX-based systems through system-level programming and hands-on experimentation. One of the basic exercises was to implement or learn how to use the `madvise` system call, which provides advice to the kernel on patterns of use of memory by an executing program. This consisted of working with system level APIs and learning the way memory is managed at the kernel level. Although the task did not imply the installation of a specific virtualization platform, I personally installed it inside Oracle Virtual Box. This option allowed me to securely separate the system environment, without affecting the host machine, and gave higher flexibility and reversibility to the installation and testing process. With the execution of this project, I intended to strengthen my understanding of the operating system fundamentals, kernel structure, and system configuration problems in the real world.

The following specific objectives were also pursued:

- To get an insight into UNIX-based systems' inner workings by working with OpenBSD hands-on.
- To learn the management of memory at the system level using the `madvise()` system call.
- To install and set up a secure virtual environment under Oracle Virtual Box for experimentation purposes.
- To study how kernel-level operations affect performance and behavior of the system.
- To have enhanced hands-on experience of system calls and their importance in resource management in an operating system.

3. Requirement

In the aim to perform the installation of OpenBSD the following couple of essential hardware and software requirements needed to be present. Concerning hardware, a personal computer that has a minimum dual-core processor and 4 GB of RAM was sufficient enough to support both the host operating system and the virtual machine at the same time. Although OpenBSD is a lightweight and efficient operating system, virtualization still takes some additional system resources to effectively emulate hardware. A minimum of 20 GB of available storage space was also needed to install the Oracle Virtual Box software package, the OpenBSD ISO file, and the virtual hard disk employed during installation.

Other Hardware Requirements:

- A solid power supply to avoid shutdowns during the installation.
- A working mouse and keyboard for interacting with the virtual machine.
- A monitor of sufficient resolution to appropriately display the OpenBSD installer.

From the software perspective, two dominant factors were needed. The first was Oracle VM Virtual Box, a powerful virtualization tool available for free that allows one to create and run virtual machines on more than one host operating system. It was installed from the official website of Oracle and worked without any additional plug-in or extensions being installed. The second was the ISO installation image of OpenBSD, downloaded from the OpenBSD official download site. This ISO image has all the system files necessary to create a complete installation of the operating system within the virtual environment.

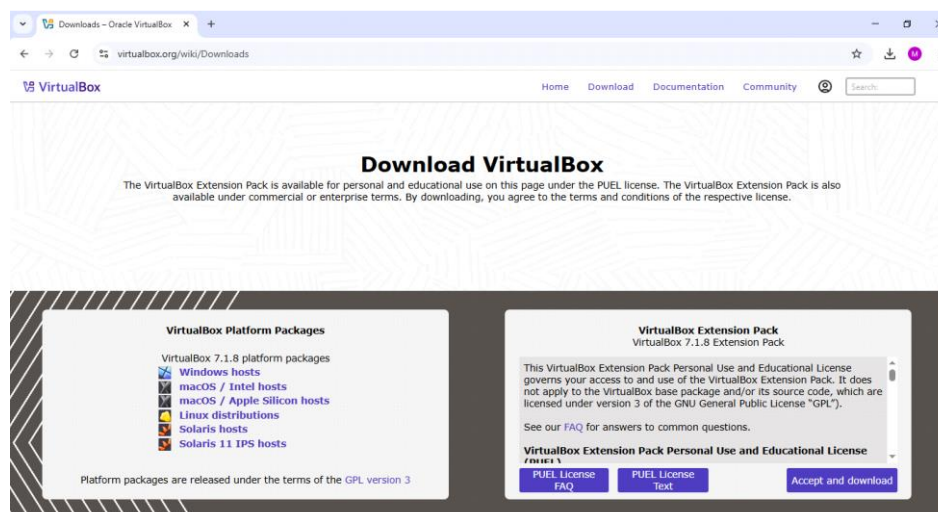
Other Software Needs:

- A stable internet connection for installing required installation files.
- Files extract utility (optional) in case the ISO is packed in an archive form.

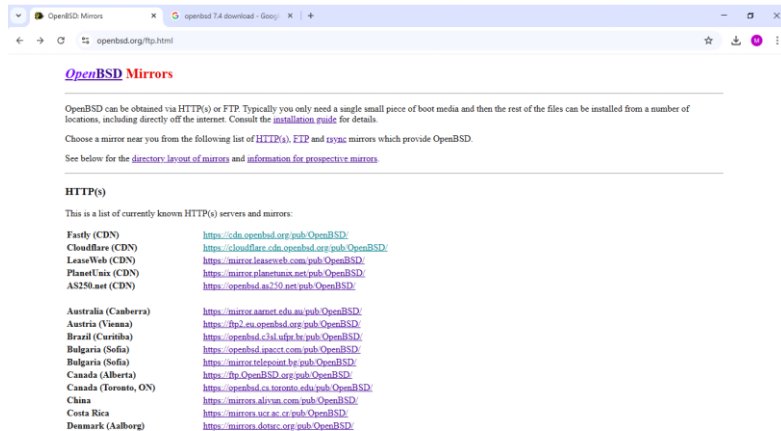
4. Installation Process

Step by step explanation

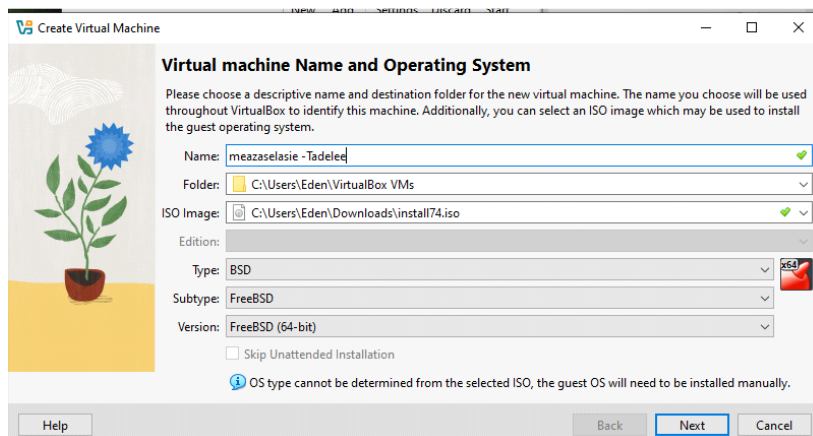
- First, I searched for Oracle Virtual Box and downloaded it from the official website.



- Then I downloaded the OpenBSD ISO from OpenBSD mirror.



- Once installed on my computer, I opened Virtual Box and created a virtual machine. I chose OpenBSD (64-bit) as the operating system and named the VM. I then mounted the ISO file to the virtual machine's optical drive from the VM settings.



- I started the virtual machine, and it booted into OpenBSD's text-based installer. The installer asked what I wanted to do I pressed i to start a fresh installation. I accepted the default keyboard layout since it matched my system. Then I entered a hostname of my choice for the system.

```

softraid0 at root
scsibus1 at softraid0: 256 targets
root on rd0a swap on rd0b dump on rd0b
WARNING: CHECK AND RESET THE DATE!
erase ^?, werase ^W, kill ^U, intr ^C, status ^T

Welcome to the OpenBSD/amd64 7.4 installation program.
(I)nstall, (U)pgrade, (A)utoinstall or (S)hell? I
At any prompt except password prompts you can escape to a shell by
typing '!'. Default answers are shown in []'s and are selected by
pressing RETURN. You can exit this program at any time by pressing
Control-C, but this can leave your system in an inconsistent state.

Choose your keyboard layout ('?' or 'L' for list) [default]
System hostname? (short form, e.g. 'foo') meaza

```

- I left the default interface (network) at em0, ran autoconf for both IPv4 and IPv6, and the system configured DNS automatically.

```

Choose your keyboard layout ('?' or 'L' for list) [default]
System hostname? (short form, e.g. 'foo') meaza

Available network interfaces are: em0 vlan0.
Network interface to configure? (name, lladdr, '?', or 'done') [em0] ?
Available network interfaces are: em0 vlan0.
    em0: lladdr 08:00:27:dd:9b:ba
Network interface to configure? (name, lladdr, '?', or 'done') [em0]
IPv4 address for em0? (or 'autoconf' or 'none') [autoconf]
IPv6 address for em0? (or 'autoconf' or 'none') [none] autoconf
Available network interfaces are: em0 vlan0.
Network interface to configure? (name, lladdr, '?', or 'done') [done]
Using DNS domainname my.domain
Using DNS nameservers at 10.0.2.3

```

- Then I set a root password for admin access. I chose not to allow SSH login for root, for security reasons.

```

Password for root account? (will not echo)
Password for root account? (again)
Start sshd(8) by default? [yes] no
Do you expect to run the X Window System? [yes] yes
Do you want the X Window System to be started by xenodm(1)? [no] yes
Setup a user? (enter a lower-case loginname, or 'no') [no]

```

- When asked if I wanted to start the X Window System with xenodm, I selected yes to enable a graphical login screen. Then, I was prompted to create a user account. I entered mezaselasie as the username and typed my full name as mezaselasie tadele. I set a password for the user account by entering it twice. After that, I confirmed the timezone as Africa/Addis_Ababa.

```

Do you expect to run the X Window System? [yes] yes
Do you want the X Window System to be started by xenodm(1)? [no] yes
Setup a user? (enter a lower-case loginname, or 'no') [no] meazaselasie
Full name for user meazaselasie? [meazaselasie] meazaselasie tadele
Password for user meazaselasie? (will not echo)
Password for user meazaselasie? (again)
What timezone are you in? ('?' for list) [Africa/Addis_Ababa]

Available disks are: wd0.
Which disk is the root disk? ('?' for details) [wd0] ?
    wd0: UBOX HARDDISK (16.0G)
Available disks are: wd0.
Which disk is the root disk? ('?' for details) [wd0]
Encrypt the root disk with a passphrase? [no] yes

Configuring the crypto chunk wd0...

No valid MBR or GPT.
Use (W)hole disk MBR, whole disk (G)PT or (E)dit? [whole]
Setting OpenBSD MBR partition to whole wd0...done.
New passphrase: _

```

The installer showed that wd0 was the available disk, and I selected it as the root disk. I chose to encrypt the root disk with a passphrase for security. Since no valid MBR or GPT was found, I selected to use the whole disk with MBR. Then, I entered a new passphrase to finish setting up the encryption.

- The pre-selected system sets from the ISO were installed by the installer, e.g., `bsd`, `base74.tgz`, `comp74.tgz`, etc.

```

/dev/sd0h (0569335437ee3fa0.h) on /mnt/usr/local type ffs (rw, asynchronous, local, nodev)
/dev/sd0j (0569335437ee3fa0.j) on /mnt/usr/obj type ffs (rw, asynchronous, local, nodev, nosuid)
/dev/sd0i (0569335437ee3fa0.i) on /mnt/usr/src type ffs (rw, asynchronous, local, nodev, nosuid)
/dev/sd0e (0569335437ee3fa0.e) on /mnt/var type ffs (rw, asynchronous, local, nodev, nosuid)

Let's install the sets!
Location of sets? (cd0 disk http nfs or 'done') [cd0]
Pathname to the sets? (or 'done') [/4/amd64]

Select sets by entering a set name, a file name pattern or 'all'. De-select sets by prepending a '-', e.g.: '-game*'. Selected sets are labelled '[X]'.
[X] bsd [X] comp74.tgz [X] xbase74.tgz [X] xserv74.tgz
[X] bsd.rd [X] man74.tgz [X] xshare74.tgz
[X] base74.tgz [X] game74.tgz [X] xfont74.tgz
Set name(s)? (or 'abort' or 'done') [done]
Directory does not contain SHA256.sig. Continue without verification? [no] yes
Installing bsd 100% |*****| 24750 KB 00:00
Installing bsd.rd 100% |*****| 4550 KB 00:00
Installing base74.tgz 100% |*****| 368 MB 00:00
Extracting etc.tgz 100% |*****| 257 KB 00:00
Installing comp74.tgz 0% | | 0 --:-- ETA

```

- Once installation finished, the installer asked to reboot the system. I rebooted the virtual machine, and it started from the virtual hard drive. The OpenBSD login prompt appeared, showing that the installation was successful.



Conclusion

The installation of OpenBSD was successfully done in Virtual Box. The installation involved easy system configuration, user creation, and system set installation. It gave an appropriate idea about installing OpenBSD and booting it online. As a whole, it was an easy but informative process.

5. Issues and problems

The biggest problem I encountered during the installation process was immediately after the very first reboot. The system emerged with a warning telling me, "check your date and reset it," preventing it from entering normal running state. This was because of the virtual machine booting continuously into the ISO image instead of booting from the installed system. The system clock of OpenBSD was not also synchronized with the hardware clock of the host, and this caused a time and date difference. This is a common issue of virtualized configurations, particularly UNIX-like systems install, which highly rely on system time for scheduling.

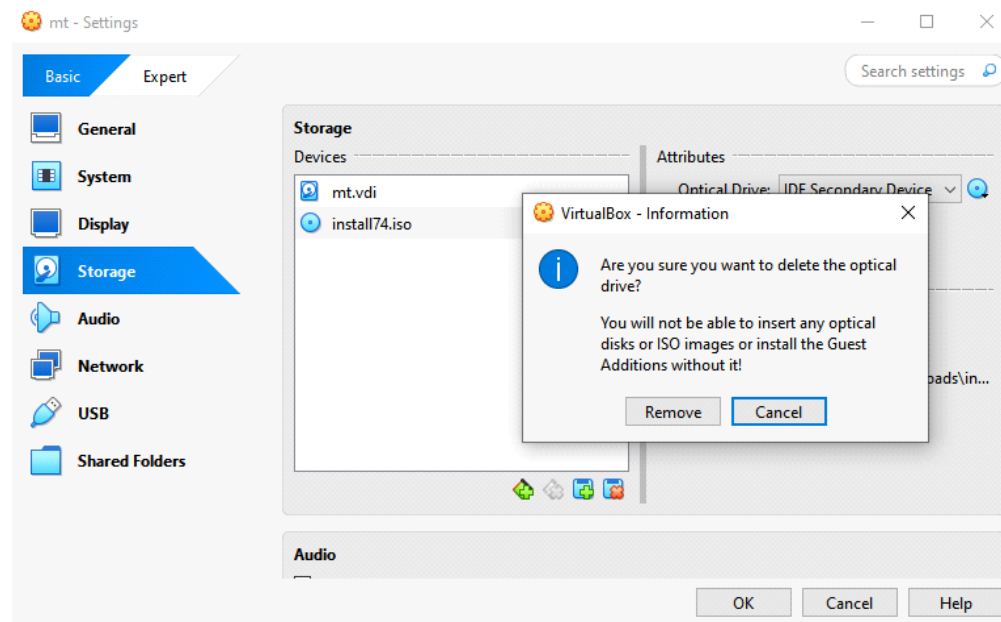
It took some trial and error, but I figured out that the cause of the problem was the virtual CD/DVD drive still using the OpenBSD ISO file. As long as the ISO was mounted, the system continued to think that each reboot was a failed installation attempt and was interfering with the boot loader and resulting in the time-related error. After I removed the iso file from virtual drive, the virtual machine booted directly into the installed OpenBSD environment, and the time warning no longer appeared. The system continued to operate as usual, with me being able to log in and making it possible for me to continue with further setup and system call implementation.

This problem provided me with a glimpse of how virtualized systems are complex and how operating systems handle them. It demonstrated that how important learning system configuration is and how a minute failure, e.g., an ISO mount, will create ginormous disruptions. The experience made me realize the importance of synchronizing system clock and hardware clock so that a system may work fine, especially in UNIX-like systems. It also emphasized the need for troubleshooting expertise, patience, and cautions to track each and every step of the configuration and installation process.

```
usb0 at ehci0: USB revision 2.0
uhub0 at usb0 configuration 1 interface 0 "Intel EHCI root hub" rev 2.00/1.00 ad
dr 1
usb1 at ohci0: USB revision 1.0
uhub1 at usb1 configuration 1 interface 0 "Apple OHCI root hub" rev 1.00/1.00 ad
dr 1
isa0 at mainbus0
pckbc0 at isa0 port 0x60/5 irq 1 irq 12
pckbd0 at pckbc0 (kbd slot)
wskbd0 at pckbd0: console keyboard, using wsdisplay0
softraid0 at root
scsibus1 at softraid0: 256 targets
root on rd0a swap on rd0b dump on rd0b
WARNING: CHECK AND RESET THE DATE!
erase ^?, werase ^W, kill ^U, intr ^C, status ^T

Welcome to the OpenBSD/amd64 7.4 installation program.
(I)nstall, (U)pgrade, (A)utoinstall or (S)hell? S
# date
Mon Apr 7 06:34:23 GMT 2025
# date 040719302025
date: illegal time format
usage: date [-a jul] [-f pformat] [-r seconds]
       [-z output_zone] [+format] [[[[[[cc]yy]mm]dd]HH]MM[.SS]]]
#
```

This figure explains the issue I faced during the installation of openbsd.



As you can observe from above, I fixed the problem by dismounting the mounted ISO image prior to rebooting, upon completing installation.

6. File System Support

OpenBSD supports several file systems, most notably its native FFS (Fast File System), also widely referred to as UFS (Unix File System). It is the one used by default upon installation and was designed to provide stable operation, fast access times, and reliable metadata journaling. OpenBSD's implementation includes support for features such as soft updates to keep the file system consistent, especially on abrupt (sudden) shutdowns.

Apart from its native file system, OpenBSD also has read-only support for a number of external file systems such as FAT32 to allow users to read USB storage devices and other removable media which employ that universal format. Support for writing on FAT32 is not really advisable since there is no journaling and hence corruption of data could be possible. Ext2, a file system that is widespread in Linux distributions, is supported in read-only mode as well, mainly for interoperability and data recovery.

These file systems such as NTFS, exFAT, ZFS, and APFS are not natively supported in OpenBSD because of the system's focus on security, simplicity, and BSD-licensed software. The goal is to reduce dependence on closed technology and maintain the code base tidy and auditable.

7. Advantage and Disadvantage

Advantage of openBSD

1. One of the most advantages of OpenBSD is that it places an utmost focus on security.
2. OpenBSD is renowned within the UNIX community for being ahead of the game when it comes to security features, secure-by-default configurations, extensive code auditing, and a low attack surface.
3. OpenBSD employs a number of security technologies such as W^X (Write XOR Execute) memory protections, `pledge()`, and `unveil()`, which function to minimize program capabilities and narrow the exploitability. These make it an excellent candidate to be employed in configurations that need utmost reliability and less vulnerability.
4. Simplicity and stability of the system are another significant advantage.
5. OpenBSD's codebase is clean, well-documented, and the developer keeps it in excellent consistency. It lacks unnecessary features and targets correctness, thereby easier to learn and construct with, especially for learning and system-level learning.
6. Running OpenBSD in a virtualized context like Oracle Virtual Box also brings the benefit of hardware separation as well as trying things out without having an effect on the host system.
7. OpenBSD is documented extensively, to the extent of having its own install manual.
8. It is simple to learn and debug, even for OpenBSD beginners.

Disadvantage of openBSD

9. OpenBSD is not without flaws. The most apparent among them is the scarce hardware support compared to more popular operating systems such as Linux or Windows. Since the project does not accept non-open or proprietary drivers, newer hardware devices, especially graphics or wireless cards, may not function at all or as intended.
10. OpenBSD also does not have support for modern desktop environments and multimedia applications, so it is less suitable for daily desktop usage or users who require large graphical interfaces and application support. Its use is mainly meant for servers, firewalls, or educational and development purposes.
11. Moreover, lack of adequate support for third-party file systems such as NTFS or exFAT might make sharing of data with other systems a bit cumbersome, especially in multi-platform setups.

8. Conclusion

Through this project, I have gained hands-on experience in installing and setting up an operating system from scratch. OpenBSD running on a virtual machine taught me more about how UNIX-like systems work, especially how systems are set up, user accounts are set up, and initial setup procedures. I became more comfortable using command-line tools and navigating through a text-based installation procedure. I also learned about how to deal with real-world issues, like boot failures because of installed installation media, and how to effectively fix them. Researching OpenBSD taught me about its file system structure, security-focused design, and system behavior in general. This experience has made me more confident in dealing with system-level operations and has prepared me to learn more about advanced (complex) topics like system calls and memory management.

9. Future Outlook and Recommendation

I will further learn about OpenBSD in the future, i.e., by using the command line interface and exploring with its pre-installed tools. I would also like to continue writing simple shell scripts so that they are able to support simple tasks and help me understand how the system handles things, users, and services. Because I don't know it yet, I will spend some time learning step-by-step each part and try to refer to the official documents more and more. As a recommendation, having new users start with easy step-by-step guides, especially for installations. Certain sections are confusing without experience. It is also recommended to test OpenBSD initially in a virtualized environment, which I did, so that changes are not applied on a live machine head on. Constant practice and setting up small tests within the system might also boost confidence and build a stronger foundation in the long run.

Aside from experimentation, I also intend to examine OpenBSD's source code in order to gain a

clearer picture of its internal design and system design decision-making. This can allow me to gain a clearer understanding of how its basic components interact, especially in memory protection and process isolation. I am also interested in knowing how OpenBSD implements security at the kernel level because it is a valuable lesson for system programming and cyber security. By learning how OpenBSD obtains minimalism without losing robustness, I hope to appreciate effective OS engineering principles more.

10. Virtualization in modern operating system

Virtualization is the technology that allows one physical computer to run many different operating systems or environments at the same time. Virtualization accomplishes this by creating virtual machines (VMs), separate systems that run inside software but behave like real computers. A virtual machine can possess an operating system, applications, hard drive space, and network configuration, but draw on the host machine's physical resources (CPU, memory, disk, etc.). The reason virtualization is so important in modern operating systems is that it provides many benefits. One of the greatest advantages is resource efficiency. Instead of needing a separate physical computer for each operating system or function, virtualization allows all to be done on one machine. This conserves hardware costs, power, and space. It also improves testing and development because users are able to create virtual environments where they can test different operating systems or setups securely without compromising the main system. For example, in this project, OpenBSD was installed inside a virtual machine using Oracle Virtual Box, which made it easy to write on the OS without having to change the host system. Virtualization is done with a program called a hypervisor. This hypervisor straddles the hardware and the virtual machines, controlling the allocation of system resources. There are two main types: Type 1 runs directly on the hardware (like VMware ESXi), and Type 2 runs on a second OS (like Oracle Virtual Box or VMware Workstation). There, Oracle Virtual Box is a Type 2 hypervisor running over the host operating system such that OpenBSD as a guest OS can be used.

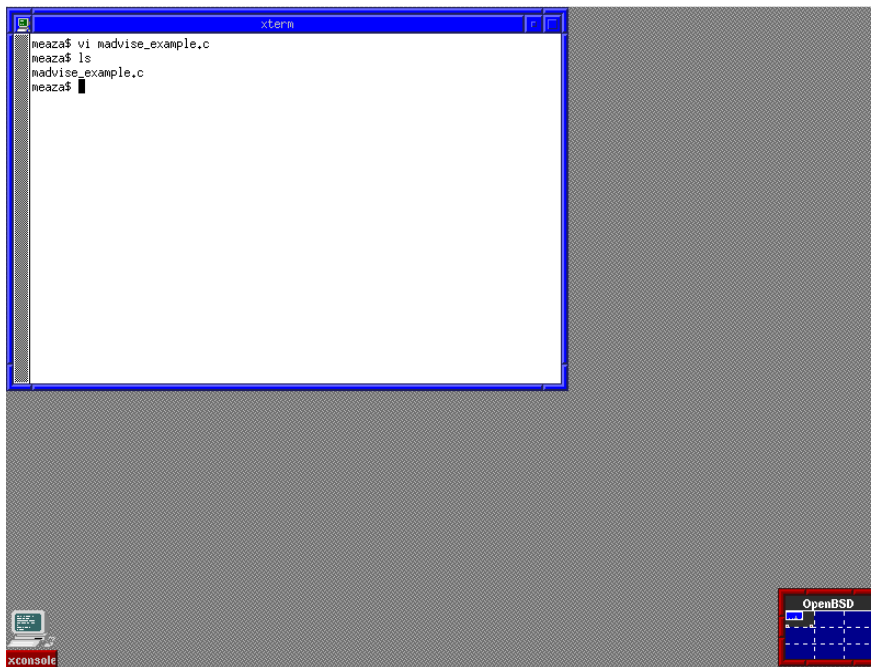
Virtualization in Modern Operating Systems (Why It Is Necessary)

- Virtualization makes the system more isolated and secure by encapsulating faults or malicious behavior into other virtual machines.
- Virtualization also aids backup and recovery of the system easily through the process of creating snapshots of virtual machines, which can be easily recovered if they fail.
- Virtual machines are easily cloned or migrated from one physical machine to another, and this makes it easy to deploy and maintain the systems.
- Virtualization enables legacy software to be executed using older operating systems running inside virtual machines without the need for special hardware.

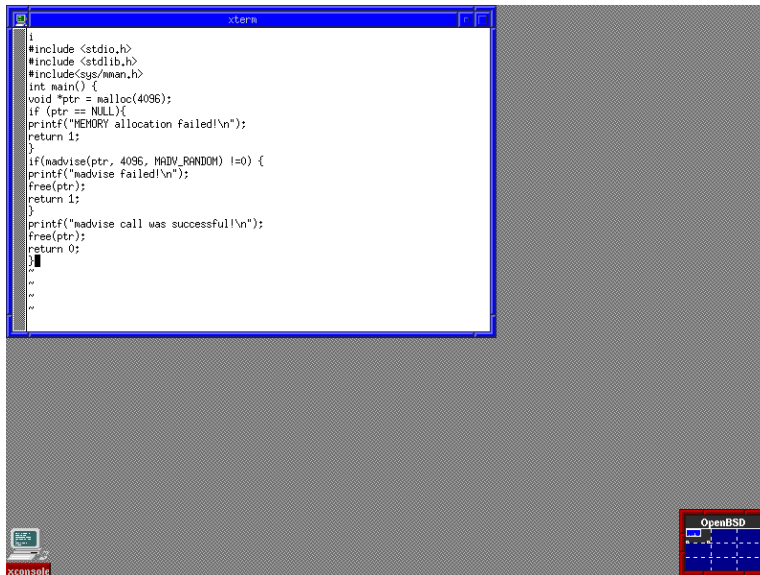
11. System Call Implementation

The `madvise()` system call is employed to convey the operating system hints regarding how a program plans to access a particular piece of memory. This offers the OS sufficient information to alter memory treatment for better performance depending on the anticipated access pattern. It alters nothing in the program's behavior but assists in allowing the kernel to make more effective paging and caching decisions. In this case, I employed the `MADV_RANDOM` flag to advise the system that the memory will be accessed in an unpredictable or non-linear way, something which leads the system to bypass unwanted read-ahead reads.

Once the system was installed, i implemented and tested the `madvise` system call. To do this, I used a terminal and executed the command `vi madvise_example.c` to open and edit a C source file.



In the file, I have included a simple C program that allocates 4096 bytes of memory with malloc, and then uses the madvise system call to inform the operating system that the memory is to be accessed in a random order (MADV_RANDOM).



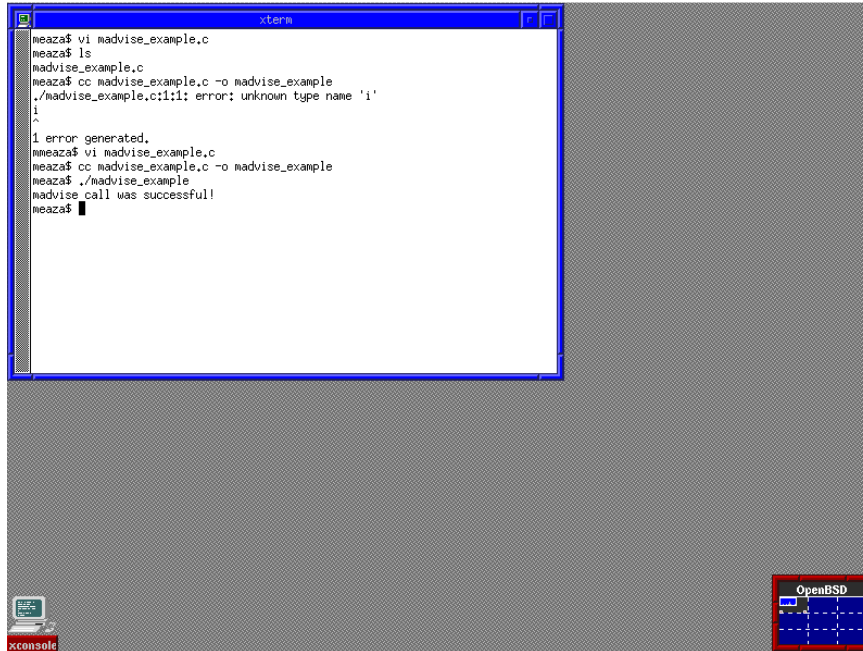
```
1
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
int main() {
    void *ptr = malloc(4096);
    if (ptr == NULL) {
        printf("MEMORY allocation failed\n");
        return 1;
    }
    if (madvise(ptr, 4096, MADV_RANDOM) != 0) {
        printf("madvise failed\n");
        free(ptr);
        return 1;
    }
    printf("madvise call was successful\n");
    free(ptr);
    return 0;
}
```

The program tests if the memory allocation and madvise are successful and prints a message accordingly. After saving the file, I compiled it, but initially had an issue where there was a character in the code that was unrecognized.



```
meaza$ vi madvise_example.c
meaza$ ls
madvise_example.c
meaza$ cc madvise_example.c -o madvise_example
./madvise_example.c:1:1: error: unknown type name 'i'
1
^
1 error generated.
```

I recognized the issue, fixed it, and recompiled the program without issues. When running the executable, it gave the following output: "madvise call was successful!", which meant that the system call worked as intended



```
xterm
neaza$ vi madvise_example.c
neaza$ ls
madvise_example.c
neaza$ cc madvise_example.c -o madvise_example
./madvise_example.c:1:1: error: unknown type name 'i'
1
~
1 error generated.
neaza$ vi madvise_example.c
neaza$ cc madvise_example.c -o madvise_example
neaza$ ./madvise_example
madvise call was successful!
neaza$
```


References

1. openbsd project. (n.d.). <https://www.openbsd.org/ftp.html>
2. Oracle Corporation. <http://www.virtualbox.org/>
3. Security features. https://en.wikipedia.org/wiki/OpenBSD_security_features
4. Andrath of the abyss youtube tutorial.