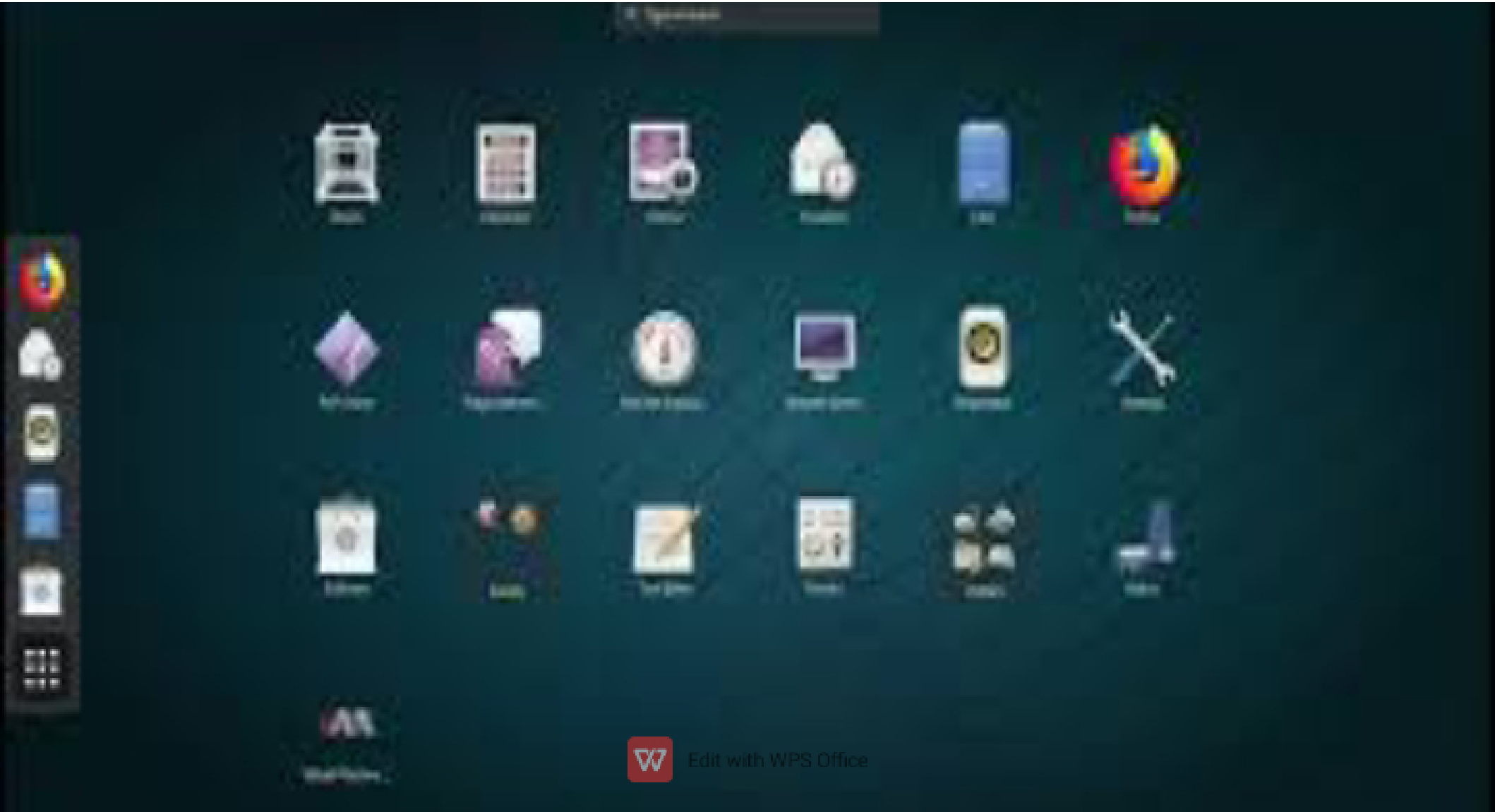
Assignment By Tigist Dejen Fentie

Id 1602589

section B

Table of Contents pages

Red Hat Enterprise Linux (RHEL)



1. Introduction

Red Hat Enterprise Linux (RHEL) is a leading enterprise Linux platform developed by Red Hat Inc., designed for performance, security, and scalability. It is used in servers, cloud environments, and workstations, and is known for its stability and long-term support. RHEL is subscription-based, offering enterprise-grade support and updates to customers.

2. Objective

The primary objective of RHEL is to provide:

A stable and secure operating system for enterprise-level computing

Support for mission-critical applications with high performance and reliability.

A standardized platform for development, deployment, and innovation in hybrid cloud environments.

Long-term maintenance and certified compatibility with hardware and software vendors

Hardware Requirements for RHEL

Minimum Requirements (for basic use or minimal installation):

CPU: 1.1 GHz 64-bit processor (x86_64 architecture)
RAM: 2 GB

Storage: 20 GB hard disk space

Display: 800x600 resolution

Boot Mode: BIOS or UEFI supported

Network: Ethernet or WiFi (for updates and

Recommended Requirements (for GUI or server environments):

CPU: 2 GHz dual-core or higher

RAM: 4–8 GB or more

Storage: 50 GB or more (especially for GUI, applications, and logs)

Graphics: Basic supported GPU for graphical interface

Peripherals: USB port or DVD-ROM (for boot media), internet connection

Software Requirements for RHEL

Supported Architectures:

x86_64 (64-bit Intel/AMD)

ARM 64-bit (aarch64) — for servers and embedded devices

IBM Power and IBM Z (enterprise-grade servers)

Required Software for Installation:

RHEL ISO image — Downloaded from Red Hat Customer Portal

5. Common Issues and Problems Faced

Subscription Issues: Failure to register or activate the subscription may block access to updates and repositories.

Driver Compatibility: Some hardware may lack drivers (especially Wi-Fi or GPU).

Network Configuration: Post-installation network issues due to misconfigured interfaces.

SELinux Restrictions: SELinux can block services if not configured properly.

Package Dependency Errors: When using yum or dnf, some packages may conflict or be unavailable.

Software Limitations: Some open-source tools may not be readily available in the official repos.

Dual Boot Issues: Problems can arise when installing RHEL alongside another OS, like Windows.

6. Solutions to Common Issues in RHEL

Issue	Solution
Subscription Issues	Ensure system is connected to the internet. Use the command `subscription-manager register` and `subscription-manager attach` to activate the subscription.
Driver Compatibility	Use `lspci` or `lsmod` to identify missing drivers. Install necessary drivers using `dnf`, or get vendor-specific drivers if needed.

SELinux Blocking Services	Check logs using `ausearch -m avc`. Temporarily disable with `setenforce 0` (not recommended long-term). Permanently allow access with `semanage` or proper SELinux policies.
Package Dependency Errors	Enable required repos via `subscription-manager repos --enable=<repo-name>`. Clean cache with `dnf clean all` and retry.

Network Configuration

Use `nmcli` or `nmtui` to manage network connections. Restart networking service: `systemctl restart NetworkManager`.

Software Availability	Use EPEL repository for extra packages: `dnf install epel-release`. Also consider compiling from source if trusted.
Dual Boot Issues	Use `grub2-mkconfig` to regenerate boot menu: `grub2-mkconfig -o /boot/grub2/grub.cfg`. Ensure secure boot is configured properly.

7. File System Support in RHEL

RHEL supports a variety of file systems optimized for different purposes:

File System	Use Case	Features
XFS	Default in RHEL 7 and later	High performance, scalable, supports journaling and large files
ext4	Legacy support and general use	Journaling, wide compatibility, stable

Advantages of RHEL

Stability and Reliability: Enterprise-grade OS with long-term support and regular updates.

Security: Built-in SELinux, regular security patches, and compliance tools.

Support: Professional support from Red Hat with detailed documentation and community forums.

Performance: Optimized for server and cloud workloads.

Certification: Certified with a wide range of hardware and enterprise applications.

Btrfs *(limited/ experimental)*	Snapshotting, advanced features	Not officially supported in RHEL 9
VFAT/ FAT32, NTFS	External devices, dual-booting	Limited support; use `ntfs-3g` for full NTFS support
NFS	Network file sharing	Supports sharing between Linux systems
GFS2	Cluster file systems	Used in high-availability

Disadvantages of RHEL

Cost: Requires a paid subscription, which may not be ideal for individuals or small businesses.

Proprietary Control: Some features are restricted or only accessible with Red Hat subscription.

Learning Curve: May be complex for beginners compared to user-friendly Linux distros like Ubuntu.

Limited Free Access: Unlike CentOS (now CentOS Stream), full RHEL is not freely available without registration.

Container Support: Native support for Podman, Buildah, and Kubernetes.

Scalability: Suitable for small servers to large-scale data centers.

Virtualization

Virtualization is the process of creating virtual versions of physical components such as servers, storage devices, or operating systems. It allows multiple virtual machines (VMs) to run on a single physical system, each with its own OS and applications.

Why Virtualization? (Purpose)

1. Better Resource Utilization – Maximizes the use of CPU, RAM, and storage.

Cost Efficiency – Reduces hardware costs and energy consumption.

3. Isolation & Security – Each VM is isolated; one crash doesn’t affect others.

4. Flexibility & Scalability – Easy to scale up/down or clone systems.

5. Simplified Management – Centralized control using virtualization tools

6. Disaster Recovery – Easier backup and restoration of entire systems.

How Virtualization Works (Technology)

1. Hypervisor: A software layer that enables virtualization

Type 1 (bare-metal): Runs directly on hardware (e.g., VMware ESXi, Microsoft Hyper-V).

Type 2 (hosted): Runs on a host OS (e.g., VirtualBox, VMware Workstation

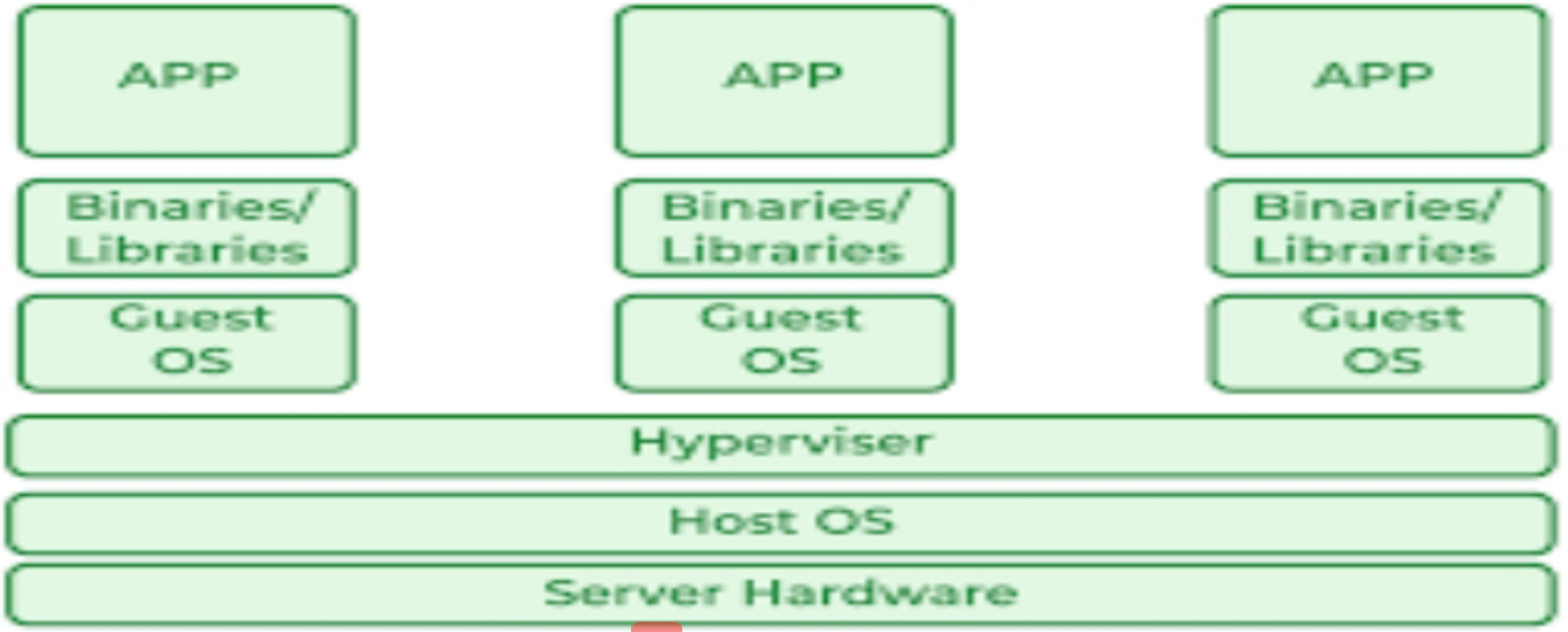
2. Virtual Machines (VMs): Software-based emulations of physical computers, with their own OS and resources.

3. Virtualization in OS:

OS-level tools (e.g., KVM in Linux, Hyper-V in Windows)

Container-based virtualization (e.g., Docker) for lightweight, fast-deploying apps

---



Virtualization

1. UNIX Standardization Overview

UNIX systems needed standardization to ensure compatibility across different vendors and versions. Standards help unify system behavior, APIs, and data types so software can run across different UNIX platforms.

2. Key Standards

ISO C:
The international standard for the C programming language. It defines syntax, data types, and the standard C library, providing portability across systems.

IEEE POSIX (Portable Operating System Interface):
A family of standards defined by IEEE to maintain compatibility across UNIX-like systems. It standardizes system APIs, shell interfaces, and utilities. E.g., POSIX.1 defines system calls like fork(), exec(), etc.

FIPS (Federal Information Processing Standards):
U.S. government standards for computing, including versions of POSIX adopted for federal use. FIPS 151-2, for example, aligns with POSIX.1.

3. UNIX System Implementations

UNIX System V Release 4 (SVR4):
Developed by AT&T, it merged features from System V, BSD, and Xenix. It influenced many commercial UNIX versions.

4.4BSD (Berkeley Software Distribution):
Developed at UC Berkeley, introduced features like TCP/IP stack, vi, and csh. A major influence on later systems like FreeBSD, NetBSD, and macOS.
Solaris
A System V-based UNIX developed by Sun Microsystems. Known for features like ZFS, DTrace, and zones (containers).

macOS

Apple’s OS, based on Darwin (a UNIX-compliant OS derived from 4.4BSD and Mach kernel). Certified as UNIX by The Open Group.

4. Limits and Options in UNIX

Limits: Defined constants in limits.h and sysconf() specify system capabilities (e.g., max file size, number of open files).

Options: POSIX defines optional features that systems may or may not support (e.g., real-time extensions, asynchronous I/O).

5. Primitive System Data Types

Defined by C and POSIX standards (e.g., int, char, pid_t, size_t, off_t) to ensure consistent data sizes and function signatures across UNIX systems

6. Conflict Between Standards

Inconsistencies: Different vendors added non-standard extensions, causing software compatibility issues.

POSIX vs. System V vs. BSD: APIs and command behavio sometimes differ (e.g., ps or echo syntax).

Resolution: Unified standards (like SUS — Single UNIX Specification) emerged to harmonize these differences.

Definition:

In C, the nanosleep() function is used to suspend execution of a program for a specified duration in seconds and nanoseconds

nanosleep() is a system call (or function in some programming languages) that allows a program to pause or sleep for a specified amount of time with a higher precision than traditional sleep functions.

```c
#include <stdio.h>
#include <time.h>

int main() {
    struct timespec req, rem;

    req.tv_sec = 1; // 1 second
    req.tv_nsec = 500000000; // 500 milliseconds

    int ret = nanosleep(&req, &rem);

    if (ret == 0) {
        printf("Slept for 1.5 seconds\n");
    } else {
        printf("Sleep interrupted\n");
    }
```

```c
include <stdio.h>
include <time.h>
include <errno.h>

oid greetWithDelay() {
    // Initialize the requested time
    struct timespec req, rem;

    req.tv_sec = 2;          // Sleep for 2 seconds
    req.tv_nsec = 500000000; // 500 milliseconds (0.5 seconds)

    printf("Hello, Tigist Dejen! Let's take a short nap...\n");

    // Call nanosleep
    if (nanosleep(&req, &rem) == 0) {
        printf("We're back after the nap!\n");
    } else {
        // Handle interruptions
        if (errno == EINTR) {
            printf("Sleep was interrupted! Remaining time: %ld seconds,
                   rem.tv_sec, rem.tv_nsec);
        } else {
            perror("nanosleep failed");
        }
    }
```

```c
#include <stdio.h>
#include <time.h>

int main() {
    struct timespec tim, tim2;
    tim.tv_sec = 1;  // Sleep for 1 second
    tim.tv_nsec = 500000000L;  // Sleep for an additional 500 milliseconds

    if (nanosleep(&tim, &tim2) < 0) {
        printf("Nano sleep system call failed\n");
        return -1;
    }

    printf("Nano sleep successful\n");
    return 0;
}
```

A system call is a mechanism that allows user-level programs (applications) to request services from the operating system's kernel. The operating system provides these services to manage hardware, resources, and other system-level tasks. System calls act as the interface between a program and the kernel, allowing user applications to perform operations such as file handling, process management, memory allocation, and device input/output (I/O)

In a Unix-like operating system (including Linux), system calls are typically invoked by a special instruction that causes a context switch from user mode to kernel mode. This ensures that the kernel can execute privileged

Implemented System Call: nanosleep

The nanosleep system call allows a process to suspend its execution for a specified time. It is a more precise way of sleeping than the standard sleep function, as it takes time in nanoseconds rather than seconds

Steps to Implement and Use the nanosleep System Call

To implement and use the nanosleep system call in a Linux environment, follow these steps

Step 2: Define the Required Header Files

In order to use the nanosleep system call, you need to include the necessary header files in your C program.

time.h: To define the timespec structure for specifying the sleep time.

unistd.h: For system call access.

```
#include <stdio.h>
#include <time.h>
#include <unistd.h>
```

Step 3: Define the timespec Structure

The nanosleep system call requires a timespec structure to define the sleep time in seconds and nanoseconds. The structure has two fields:

tv_sec: Seconds to sleep.

tv_nsec: Nanoseconds to sleep (ranges from 0 to 1,000,000,000)

```
struct timespec req;
```

Step 4: Set the Sleep Duration

You need to specify the time for which the program should "sleep" by assigning values to the tv_sec and tv_nsec fields of the timespec structure.

Step 5: Call the nanosleep Function

The nanosleep function takes two arguments:

Step 6: Compile the Program

Once you have written the program, compile it using the GCC compiler:

Step 8: Output Results

The program will pause for the specified duration (in this case, 2.5 seconds). After that, it will print the message indicating that it has woken up.

Step 9: Check for Errors

In case of an error, the nanosleep function returns -1, and the global errno variable is set accordingly. You can check for errors using the perror() function to display an error message.

1. Download RHEL ISO from the Red Hat Customer Portal.

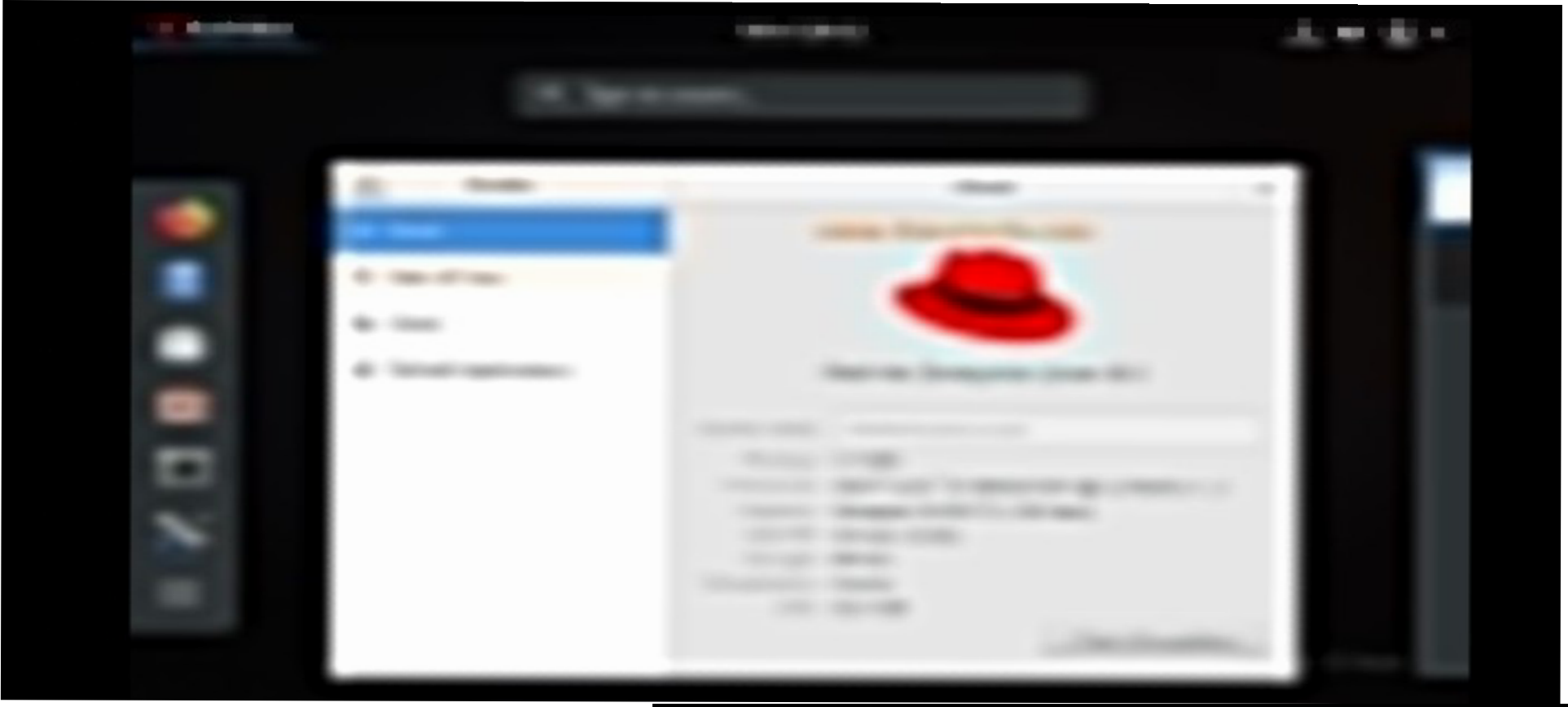
2. Create bootable USB using tools like Fedora Media Writer or dd command.
3. Boot from USB on the target machine.
4. Select “Install Red Hat Enterprise Linux” from the boot menu.
5. Choose language and keyboard, then click “Continue”.
6. Configure settings:
Set time zone
Choose installation destination (disk)
Select software environment (e.g., Server with GUI)
Configure network and hostname

7. Set root password and optionally create a user.

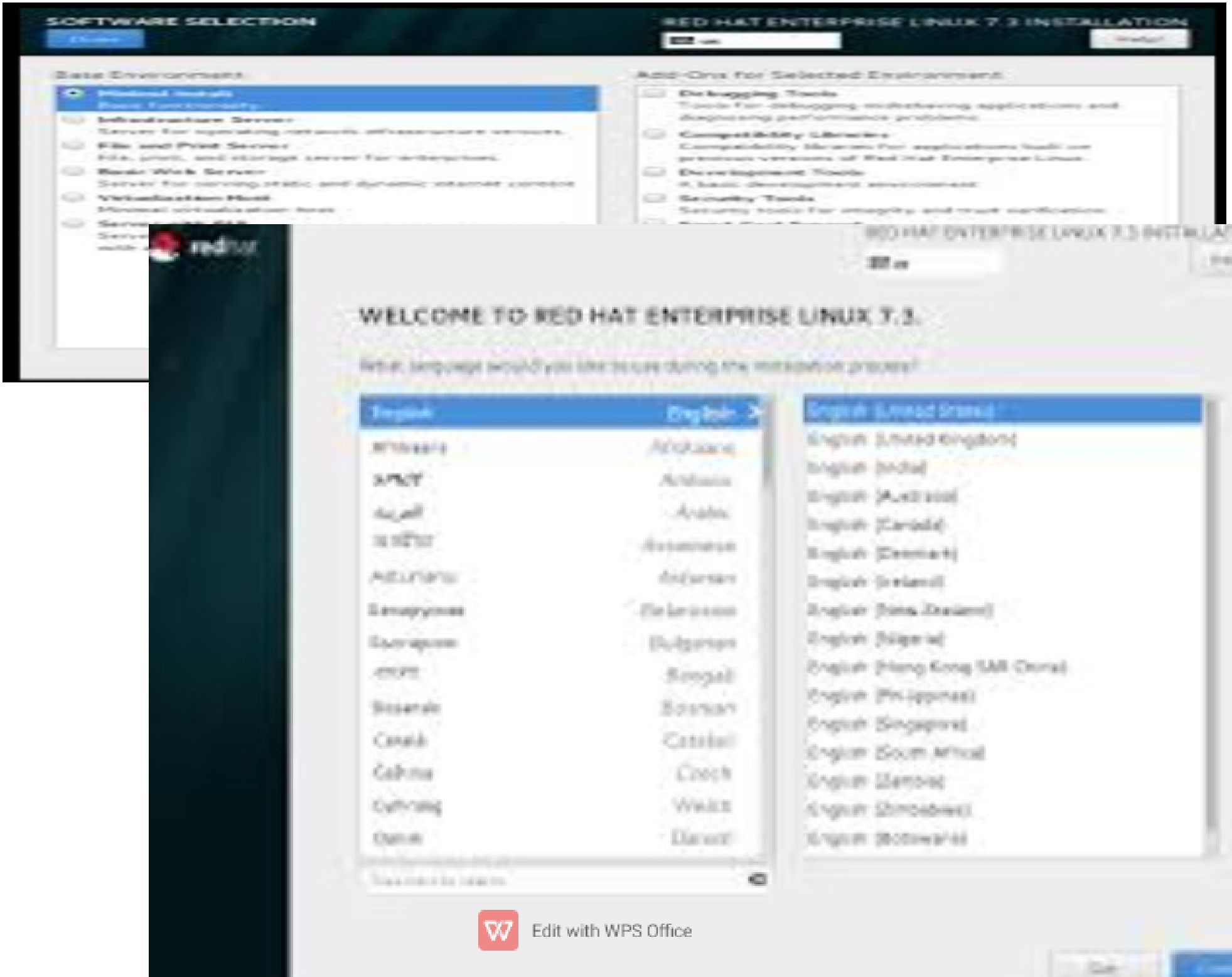
8. Click “Begin Installation”.

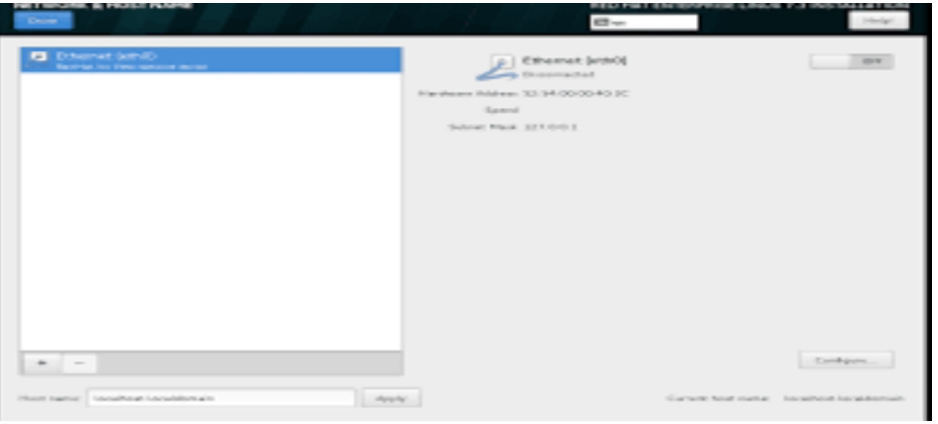
9. After installation, click Reboot.

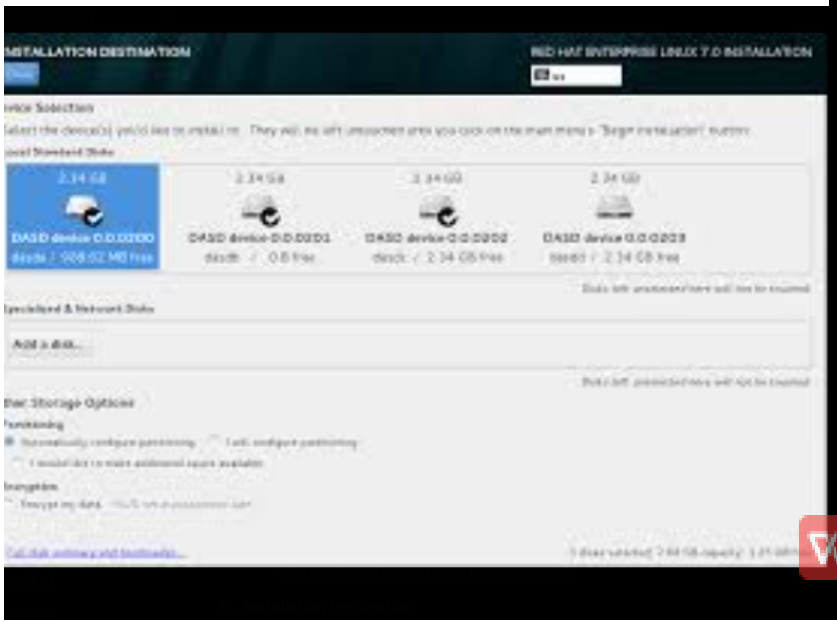
10. On first boot, register with Red Hat and complete setup.




CONFIGURATION
USER SETTINGS
Root Password
User Creation
Edit with WPS Office
WHEN IS FREE MORE EXPENSIVE?


SOFTWARE SELECTION
RED HAT ENTERPRISE LINUX 7.3 INSTALLATION
WELCOME TO RED HAT ENTERPRISE LINUX 7.3.
Edit with WPS Office





Conclusion:

In this project, we explored the nanosleep system call, which allows a process to pause or "sleep" for a specified amount of time with nanosecond precision. Unlike the traditional sleep function, which only allows sleep durations in seconds, nanosleep provides more fine-grained control over the duration of sleep, making it useful for real-time applications that require high-precision timing, such as multimedia processing, simulations, and scientific computing.

By following the steps outlined, we successfully implemented a custom system call using nanosleep to pause a process for a precise amount of time (2.5 seconds in this case). This implementation highlights

the power of system calls in enabling programs to interact efficiently with the operating system for various tasks, such as timing, resource management, and hardware interaction.

References:

1. The Linux Programming Interface by Michael Kerrisk

2 Google