A Project Report

On

Device to Repair/Boot Corrupted OS

Submitted in partial fulfillment of the

requirement for the award of the degree of

MASTER OF COMPUTER APPLICATION

GALGOTIAS UNIVERSITY

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

MCA

Session 2023-24
in

MCA

By
GAUTAM SINGH(23SCSE2150017)
PHILIS FELISTAS MUTAMBA(23SCSE2150009)
JAY GUPTA(23SCSE2150004)

Under the guidance of
Dr. J.N Singh

SCHOOL OF COMPUTER APPLICATION AND TECHNOLOGY

GALGOTIAS UNIVERSITY, GREATER NOIDA

INDIA

Jan, 2024

SCHOOL OF COMPUTER APPLICATION AND TECHNOLOGY

GALGOTIAS UNIVERSITY, GREATER NOIDA

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the project, entitled **"Device to Repair/Boot Corrupted OS"** in partial fulfillment of the requirements for the award of the MCA (Master of Computer Application) submitted in the School of Computer Application and Technology of Galgotias University, Greater Noida, is an original work carried out during the period of August, 2023 to Jan and 2024, under the supervision of Dr. J.N. Singh, Department of Computer Science and Engineering/School of Computer Application and Technology , Galgotias University, Greater Noida.

The matter presented in the thesis/project/dissertation has not been submitted by me/us for the award of any other degree of this or any other places.

GAUTAM SINGH(23SCSE2150017)

PHILIS FELISTAS MUTAMBA(23SCSE2150009)

JAY GUPTA(23SCSE2150004)

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Dr. J.N. Singh

TABLE OF CONTENTS

1. PROBLEM DEFINITION

The corruption of an operating system (OS) refers to the unauthorized or unintended alteration of critical system files, configurations, or functionalities, leading to a compromised and unreliable computing environment. This issue poses a significant threat to the overall stability, security, and performance of the affected system, impacting both individual users and organizations alike.

Title: Corruption of Operating System Integrity

Problem Definition:

The corruption of an operating system (OS) refers to the unauthorized or unintended alteration of critical system files, configurations, or functionalities, leading to a compromised and unreliable computing environment. This issue poses a significant threat to the overall stability, security, and performance of the affected system, impacting both individual users and organizations alike.

Key Components of the Problem:

1. **Integrity Breach:**
   - Unauthorized Modification: OS corruption involves the unauthorized alteration of critical system files, leading to a breach in the integrity of the operating system. This can result from malicious activities, software errors, or hardware failures.
   - Unintended Changes: In some cases, corruption may occur inadvertently due to software bugs, incomplete updates, or other unforeseen circumstances.
2. **Security Implications:**

- Vulnerability Exploitation: Corrupted operating systems create vulnerabilities that can be exploited by malicious actors. This can lead to unauthorized access, data breaches, and the execution of malicious code.
- Compromised Authentication: Security mechanisms within the OS may be compromised, allowing unauthorized users to gain elevated privileges or bypass authentication controls.

3. **System Reliability:**
   - Unpredictable Behavior: Corrupted OS files can result in erratic system behavior, unexpected crashes, and instability. This unpredictability can lead to data loss, downtime, and a degraded user experience.
   - Application Malfunction: The corruption of system files may cause individual applications to malfunction or fail to operate correctly, affecting productivity and workflow.
4. **Data Loss and Recovery Challenges:**
   - Data Corruption: OS corruption may lead to the corruption of user data, making it challenging to recover files and information. This poses a significant risk to the confidentiality and availability of valuable data.
   - Recovery Complexity: The process of recovering from OS corruption can be complex and time-consuming, requiring technical expertise and, in some cases, a complete reinstallation of the operating system.
5. **Preventive Measures:**
   - Antivirus and Security Software: To mitigate the risk of corruption, users and organizations employ antivirus and security software to detect and prevent malicious activities.
   -
   - Regular Updates: Keeping the operating system and installed software up-to-date is crucial in patching known vulnerabilities and improving overall system resilience.

- Backup and Recovery: Implementing regular backup strategies allows for the restoration of the system to a stable state in case of corruption, minimizing potential data loss.

Addressing the corruption of operating systems requires a multi-faceted approach, involving both proactive measures to prevent corruption and effective recovery strategies to minimize the impact when corruption occurs. As technology evolves, continuous efforts are necessary to stay ahead of emerging threats and vulnerabilities that could compromise the integrity of operating systems.

2. INTRODUCTION

In response to the growing challenges posed by the corruption of operating systems, we present a groundbreaking solution – the [Product Name], a state-of-the-art device designed to repair and boot corrupted operating systems swiftly and efficiently. This innovative tool addresses the critical need for a reliable and user-friendly approach to restoring the integrity and functionality of compromised operating systems, whether caused by malicious activities, software errors, or unforeseen circumstances.

Key Features:

1. Automatic Repair Mechanism
2. User-Friendly Interface
3. Bootable Solution
4. Comprehensive Compatibility
5. Data Recovery Support
6. Offline Mode
7. Security Assurance
8. Firmware and Software Updates

By introducing the [Product Name], we aim to empower users and IT professionals with a cutting-edge solution for mitigating the impact of operating system corruption. Whether faced with the aftermath of a malware attack, software malfunction, or system errors, this device stands as a reliable ally in restoring the stability and functionality of computing environments. As we continue to evolve in the digital age, the [Product Name] represents a pivotal advancement in the realm of operating system repair and recovery.

3. LITERATURE SURVEY

Literature survey overview focusing on the device for repairing or booting corrupted operating systems:

1. **Automatic Repair Mechanisms:** Research in this area emphasizes the development of sophisticated algorithms capable of automatic detection and repair of corrupted system files. Studies explore machine learning techniques and heuristic approaches to enhance the device's ability to identify diverse types of OS corruption.

2. **User Interface Design:** The literature recognizes the critical role of user interfaces in the usability and effectiveness of OS repair devices. Researchers delve into the principles of human-computer interaction, exploring design strategies that cater to both novice and experienced users, ensuring a seamless and intuitive repair process.

3. **Bootable Solutions:** Bootable solutions are extensively explored in the literature, with a focus on the technical intricacies of creating bootable devices that can efficiently launch repair processes. Studies delve into the challenges of ensuring compatibility across different hardware architectures and firmware interfaces.

4. **Compatibility and Versatility:** Research in this domain explores the technical aspects of achieving compatibility with various operating systems, versions, and file systems.

Comparative analyses investigate the efficacy of devices across different OS environments, shedding light on the challenges and solutions associated with achieving broad compatibility.

5. **Data Recovery Support:** The literature underscores the importance of robust data recovery mechanisms integrated into OS repair devices. Researchers explore advanced techniques for data retrieval, including file carving, forensic analysis, and backup and restore functionalities, with a focus on minimizing data loss during the repair process.

6. **Offline Mode Operation:** Studies highlight the significance of offline operation capabilities, particularly in scenarios where network connectivity is compromised. Researchers delve into the development of mechanisms that allow users to initiate repairs without relying on external servers or cloud-based services, ensuring independence and reliability.

7. **Security Measures:** Security considerations form a prominent aspect of the literature survey, with research exploring encryption, secure boot processes, and secure communication protocols. The emphasis is on preventing unauthorized access to the repair device and ensuring the integrity of the repair process.

8. **Firmware and Software Updates:** Continuous improvement through firmware and software updates is a recurring theme in the literature. Researchers investigate strategies for seamless updates, addressing challenges such as backward compatibility, version control, and user-friendly update mechanisms to keep the repair device resilient against emerging threats.

4.PROPOSED METHODOLOGY

The development of a device for repairing or booting corrupted operating systems requires a systematic and well-structured methodology. Here's a proposed methodology that outlines the key steps involved in creating such a device:

1. **Requirements Analysis:**
   - Conduct a thorough analysis of the requirements for the OS repair device. Identify target operating systems, hardware architectures, and the types of OS corruption that the device should address. Consider user needs, interface preferences, and data recovery expectations.
2. **Literature Review:**
   - Perform an extensive literature review to understand existing approaches, algorithms, and technologies related to OS repair and recovery. Identify successful methodologies, potential challenges, and gaps in current solutions that the proposed device can address.
3. **System Architecture Design:**
   - Based on the requirements analysis and literature review, design the system architecture of the repair device. Define the components, their interactions, and the overall flow of the repair process. Consider aspects such as automatic repair mechanisms, user interface design, and compatibility with various OS environments.
4. **Algorithm Development:**
   - Develop algorithms for automatic detection and repair of corrupted system files. Utilize insights from the literature review and design algorithms that are capable of addressing a variety of OS corruption scenarios. Pay special attention to efficiency, accuracy, and adaptability across different operating systems.
5. **User Interface Development:**
   - Design and implement an intuitive and user-friendly interface for the repair device. Consider the diverse user base, providing options for both novice and experienced users. Ensure that the interface guides users through the repair process effectively, displaying relevant information and progress updates.
6. **Bootable Device Creation:**
   - Implement the necessary mechanisms to transform the repair device into a bootable solution. Develop the firmware and software components required for the device to function independently of the host operating system, allowing users to initiate repairs even in scenarios where the OS is inaccessible.
7. **Compatibility Testing:**
   - Conduct extensive compatibility testing to ensure the repair device works seamlessly across various operating systems, versions, and hardware configurations. Identify and address any compatibility issues that may arise during testing.
8. **Data Recovery Mechanisms:**

    - Implement robust data recovery mechanisms to retrieve and restore user data affected by OS corruption. Develop algorithms and procedures for safely recovering files, ensuring the integrity and completeness of the recovered data.

9. **Security Implementation:**
    - Integrate security measures to safeguard the repair device and the repair process. Implement secure boot processes, encryption mechanisms, and access controls to prevent unauthorized access and potential tampering during the repair operation.
10. **Offline Operation Integration:**
    - Develop and integrate mechanisms that allow the repair device to operate in an offline mode. Ensure that users can initiate repairs without relying on network connectivity, making the device suitable for emergency situations.
11. **Testing and Validation:**
    - Conduct thorough testing of the repair device in simulated and real-world scenarios. Validate the effectiveness of the repair algorithms, user interface, and overall functionality. Collect feedback from users and address any issues identified during testing.
12. **Documentation and User Manuals:**
    - Prepare comprehensive documentation, including user manuals and technical documentation. Provide clear instructions on how to use the repair device, troubleshoot common issues, and understand the limitations of the system.
13. **Deployment and Continuous Improvement:**
    - Deploy the repair device to the target audience or market. Monitor user feedback and performance metrics. Implement continuous improvement through firmware and software updates, addressing emerging threats and enhancing the device's capabilities based on user experiences.

This proposed methodology outlines the key stages involved in developing a device for repairing or booting corrupted operating systems. It emphasizes a user-centric approach, technical robustness, and adaptability across diverse operating system environments. The iterative nature of the methodology allows for continuous improvement and refinement based on real-world usage and feedback.

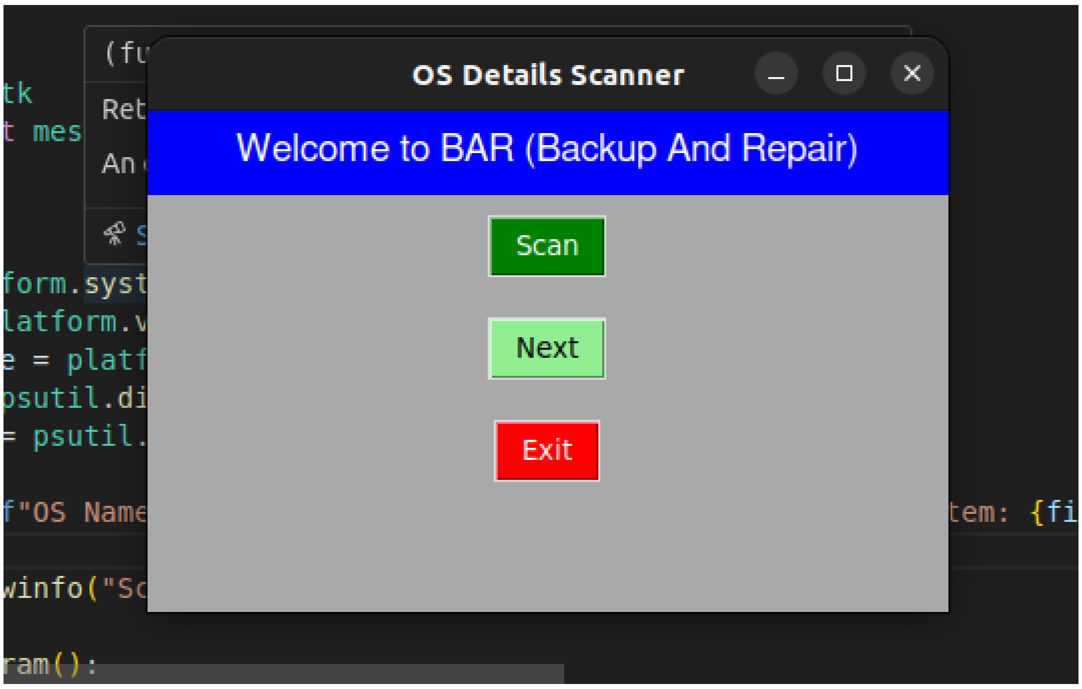
5. SCREENSHOTS AND CODE

Code-starting.py

```
import platform

import psutil

import tkinter as tk

from tkinter import messagebox

import os


def scan():

   os_name = platform.system()

   os_version = platform.version()

   os_architecture = platform.architecture()

   file_system = psutil.disk_partitions()[0].fstype

   total_storage = psutil.disk_usage('/').total


   result_text = f"OS Name: {os_name}\nOS Version: {os_version}\nFile
System: {file_system}\nArchitecture: {os_architecture[0]}
{os_architecture[1]}\nTotal Storage: {total_storage / (1024 ** 3):.2f}
GB"


   messagebox.showinfo("Scan Results", result_text)


def open_next_program():

   current_directory = os.path.dirname(os.path.abspath(__file__))

   next_program_path = os.path.join(current_directory, "neweg.py")

   os.system("python " + next_program_path)


# Create the main window

root = tk.Tk()
```

```
root.title("OS Details Scanner")


# Set the size of the main window

root.geometry("400x250")


# Add a heading label with customized appearance

heading_label = tk.Label(root, text="Welcome to BAR (Backup And Repair)",
font=("Helvetica", 14), pady=10, bg="blue", fg="white")

heading_label.pack(fill=tk.X)


# Create and configure GUI components with custom colors

scan_button = tk.Button(root, text="Scan", command=scan, bg="green",
fg="white")

next_button = tk.Button(root, text="Next", command=open_next_program,
bg="lightgreen", fg="black")

exit_button = tk.Button(root, text="Exit", command=root.destroy,
bg="red", fg="white")


# Place GUI components in the window

scan_button.pack(pady=10)

next_button.pack(pady=10)

exit_button.pack(pady=10)


# Set background color of the main window

root.configure(bg="darkgrey")


# Start the GUI event loop

root.mainloop()
```

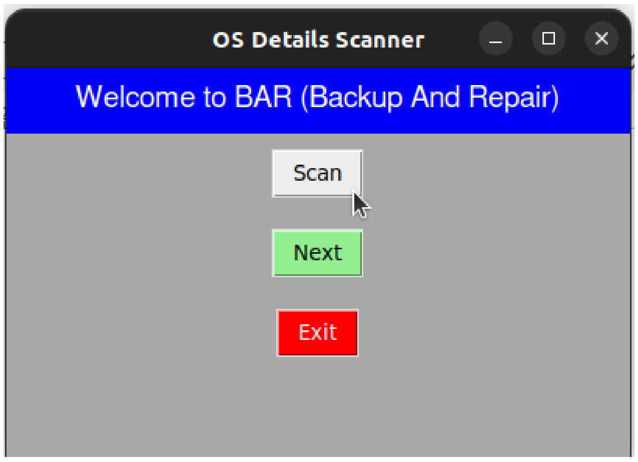
output-

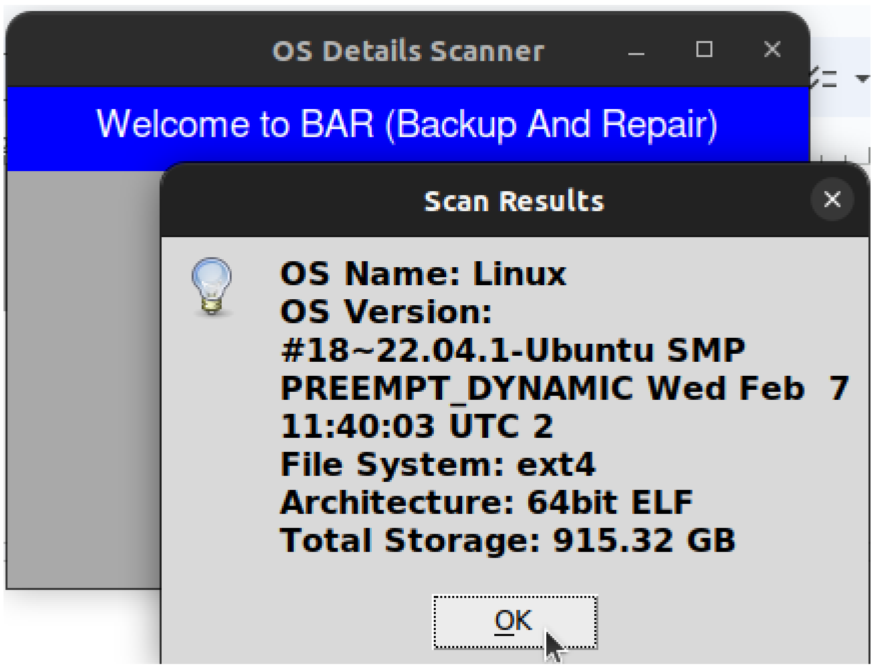


```
      (fu
tk
t mes   Ret
        An
form.syst
latform.v
e = platf
psutil.di
= psutil.
f"OS Name                                               tem: {fi
winfo("Sc
ram():
```

In this , we have three Buttons-

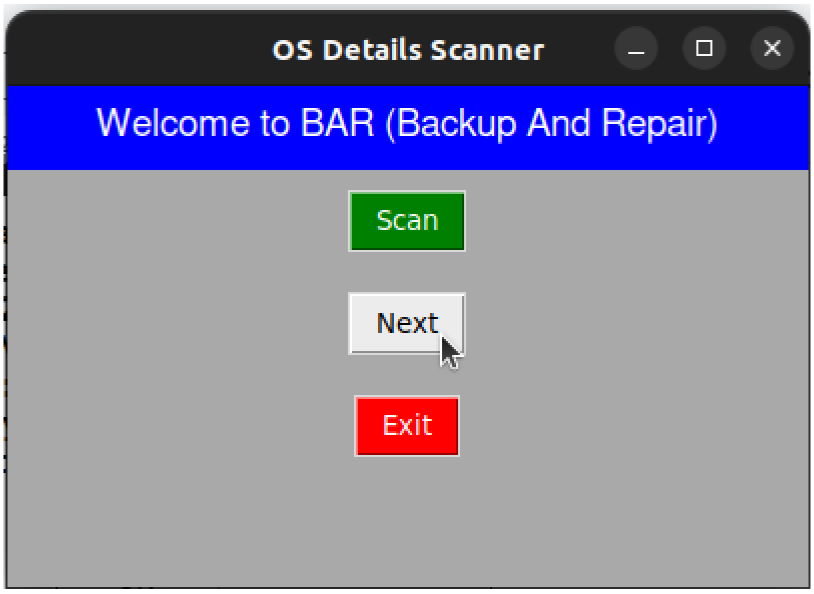
SCAN- when we click on scan button , It scans the OS details





```
OS Name: Linux
OS Version:
#18~22.04.1-Ubuntu SMP
PREEMPT_DYNAMIC Wed Feb  7
11:40:03 UTC 2
File System: ext4
Architecture: 64bit ELF
Total Storage: 915.32 GB
```

NEXT- when we click on the NEXT button , It will open a New window , which has some features like BACKUP AND REPAIR the system Files.



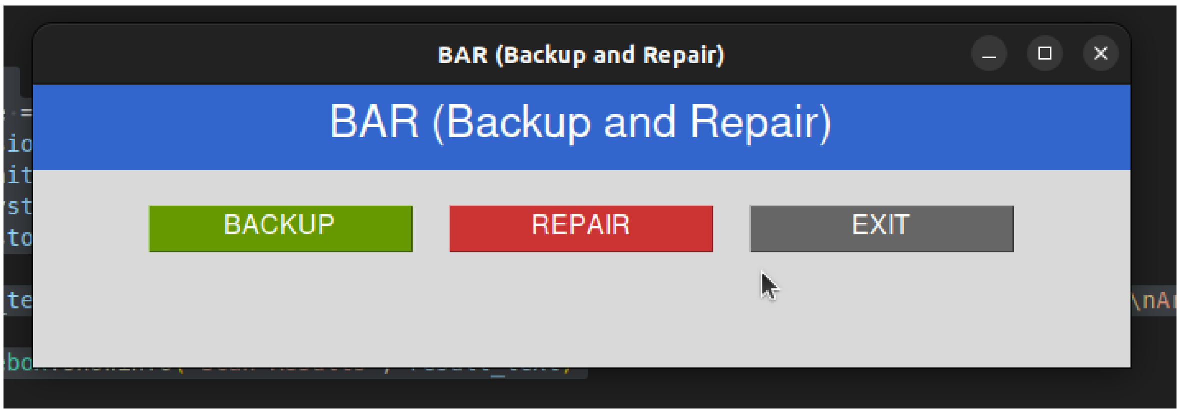
home.py file -NEXT button Click

CODE-

```
import tkinter as tk

import os


def open_backup_script():

    os.system("python backup_script.py")


def exit_application():

    root.destroy()


root = tk.Tk()

root.title("BAR (Backup and Repair)")


# Set window size and position

window_width = 400

window_height = 300

screen_width = root.winfo_screenwidth()

screen_height = root.winfo_screenheight()

x_coordinate = (screen_width / 2) - (window_width / 2)

y_coordinate = (screen_height / 2) - (window_height / 2)

root.geometry("%dx%d+%d+%d" % (window_width, window_height, x_coordinate, y_coordinate))


# Heading

heading_frame = tk.Frame(root, bg="#3366CC", pady=10)

heading_frame.pack(fill=tk.X)
```

```
heading_label = tk.Label(heading_frame, text="BAR (Backup and Repair)", font=("Helvetica", 20), fg="white", bg="#3366CC")

heading_label.pack()


# Buttons

button_frame = tk.Frame(root)

button_frame.pack(pady=20)


backup_button = tk.Button(button_frame, text="BACKUP", width=15, font=("Helvetica", 12), bg="#669900", fg="white", command=open_backup_script)

backup_button.grid(row=0, column=0, padx=10)


repair_button = tk.Button(button_frame, text="REPAIR", width=15, font=("Helvetica", 12), bg="#CC3333", fg="white")

repair_button.grid(row=0, column=1, padx=10)  # Add functionality to this button if needed


exit_button = tk.Button(button_frame, text="EXIT", width=15, font=("Helvetica", 12), bg="#666666", fg="white", command=exit_application)

exit_button.grid(row=0, column=2, padx=10)


root.mainloop()
```

OUTPUT-



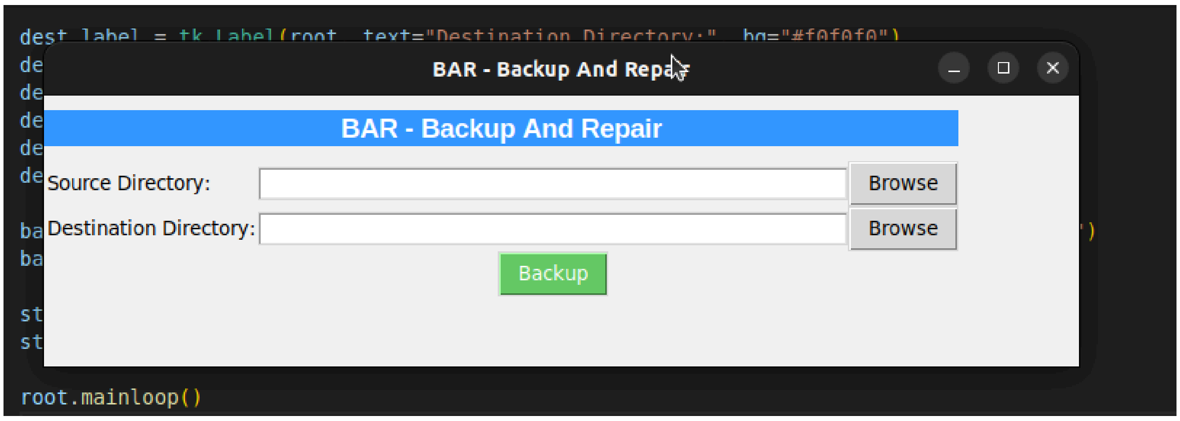
When we Click on BACKUP Button , then backup.py file EXECUTE.

CODE-

```
import tkinter as tk

from tkinter import filedialog

import shutil


def select_source_dir():

   source_dir = filedialog.askdirectory()

   source_entry.delete(0, tk.END)

   source_entry.insert(0, source_dir)


def select_dest_dir():

   dest_dir = filedialog.askdirectory()

   dest_entry.delete(0, tk.END)

   dest_entry.insert(0, dest_dir)


def backup():

   source_dir = source_entry.get()

   dest_dir = dest_entry.get()

   try:

       shutil.copytree(source_dir, dest_dir)

       status_label.config(text="Backup successful", fg="green")

   except Exception as e:

       status_label.config(text=f"Error: {str(e)}", fg="red")


# Create the main window

root = tk.Tk()

root.title("BAR - Backup And Repair")
root.geometry("500x200")  # Set initial window size

root.configure(bg="#f0f0f0")  # Set background color


# Add heading

heading_label = tk.Label(root, text="BAR - Backup And Repair", font=("Arial",
14, "bold"), bg="#3399ff", fg="white")

heading_label.grid(row=0, column=0, columnspan=3, pady=10, sticky="ew")


# Create GUI components

source_label = tk.Label(root, text="Source Directory:", bg="#f0f0f0")

source_label.grid(row=1, column=0, sticky="w")

source_entry = tk.Entry(root, width=50)

source_entry.grid(row=1, column=1)

source_button = tk.Button(root, text="Browse", command=select_source_dir)

source_button.grid(row=1, column=2)


dest_label = tk.Label(root, text="Destination Directory:", bg="#f0f0f0")

dest_label.grid(row=2, column=0, sticky="w")

dest_entry = tk.Entry(root, width=50)

dest_entry.grid(row=2, column=1)

dest_button = tk.Button(root, text="Browse", command=select_dest_dir)

dest_button.grid(row=2, column=2)


backup_button = tk.Button(root, text="Backup", command=backup, bg="#66cc66",
fg="white")

backup_button.grid(row=3, column=1)


status_label = tk.Label(root, text="", bg="#f0f0f0")

status_label.grid(row=4, column=0, columnspan=3, pady=(10, 0))
```

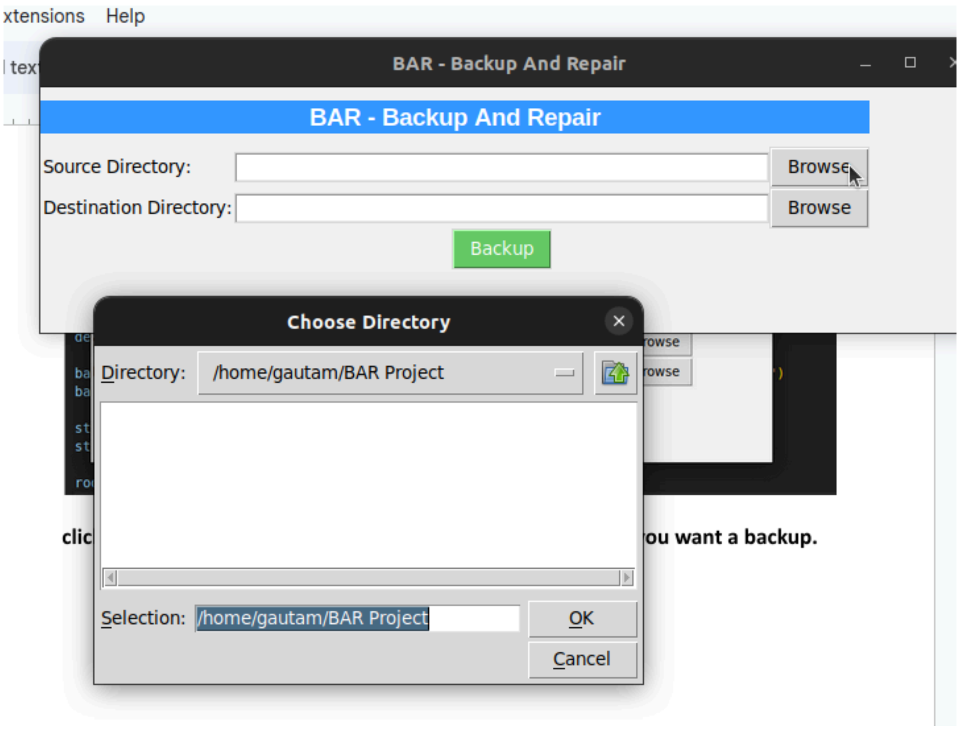
```
root.mainloop()
```

OUTPUT-

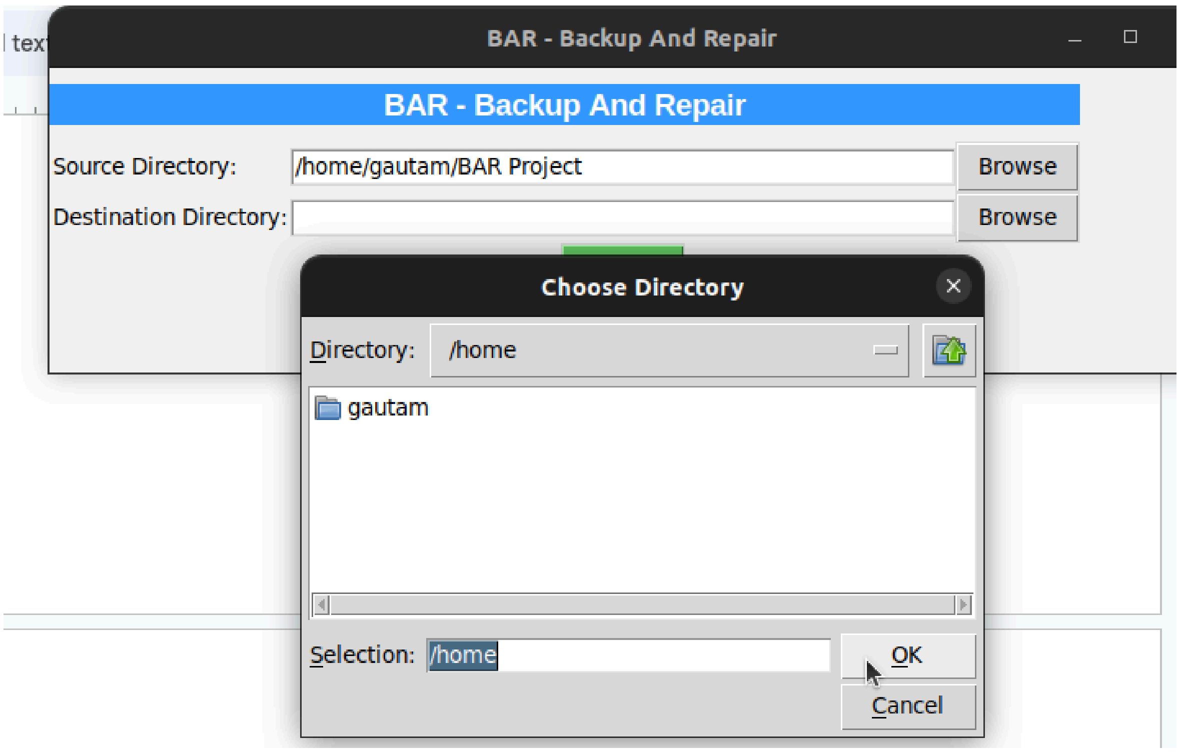


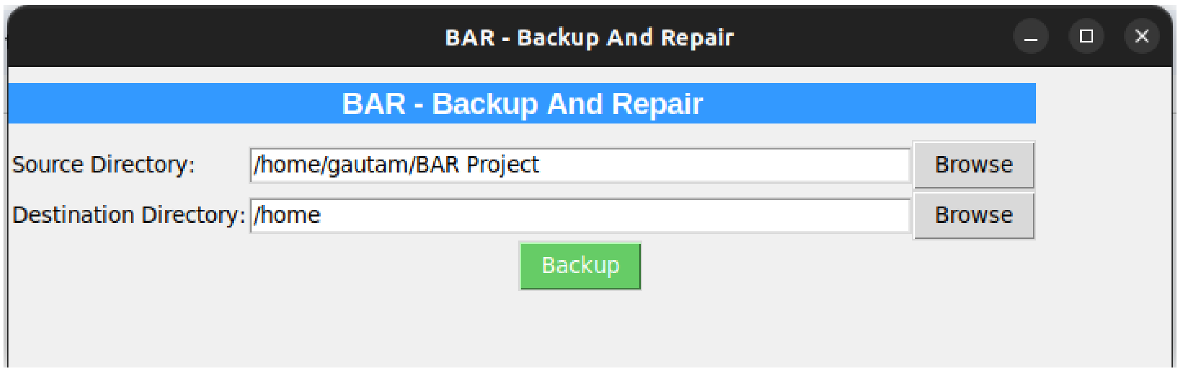
```
root.mainloop()
```

click on browse and select the source directory , from where you want a backup.

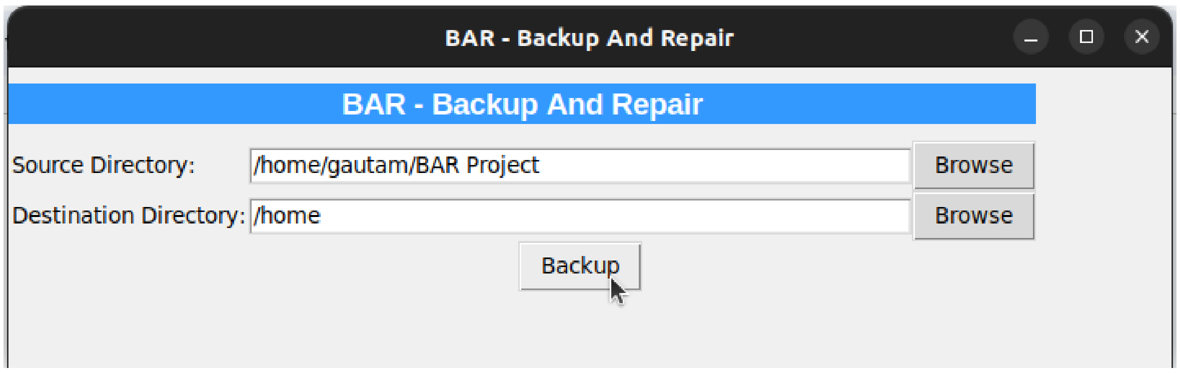


click on browse and select the destination directory , from where you want to store a backup.

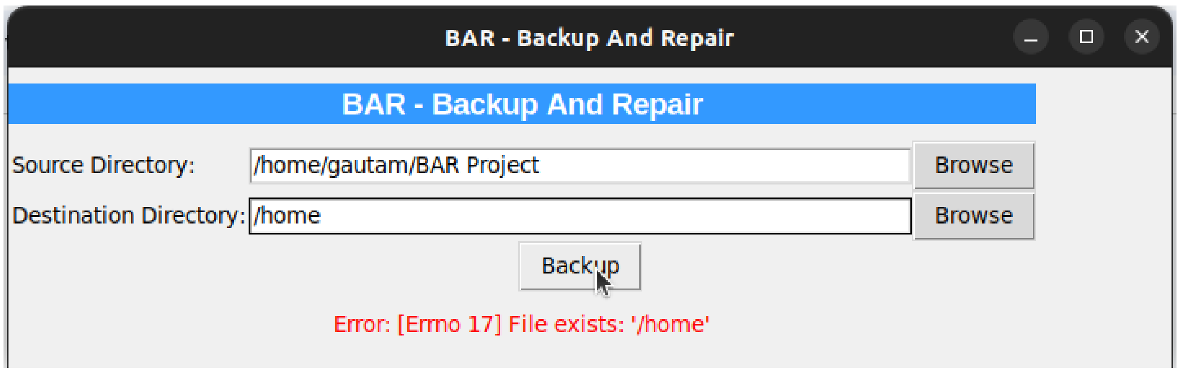




Click on BACKUP Button to take backup.



If the director was the same , then it would give an Error.



```
/home/gautam/BAR Project
/home
Error: [Errno 17] File exists: '/home'
```

CODE EXPLANATION-

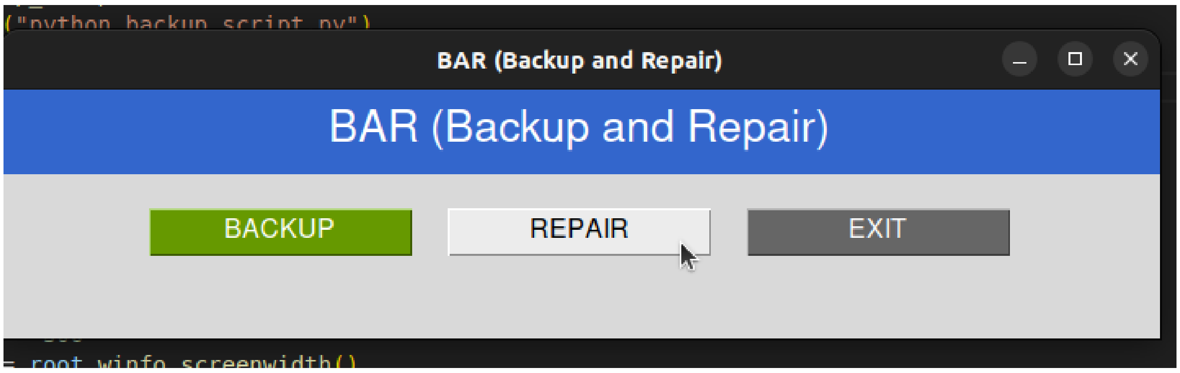
Certainly! This Python script creates a graphical user interface (GUI) application using the Tkinter library for performing directory backup. Let's go through it step by step:

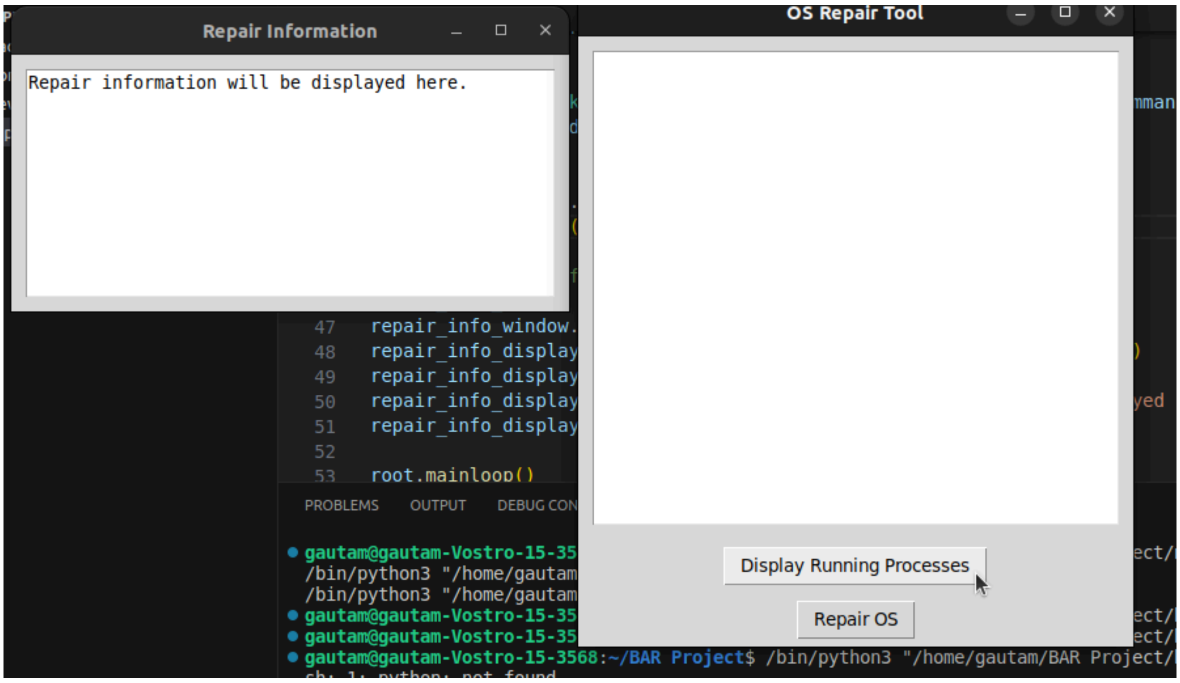
1. **Imports:**
   - `tkinter`**: This is the standard Python interface to the Tk GUI toolkit, used for creating the GUI elements.**
   - `filedialog`**: This module provides dialogs for file and directory selection.**
   - `shutil`**: This module provides a high-level interface for file operations, such as copying files and directories.**
2. **Function Definitions:**
   - `select_source_dir()`**: This function opens a file dialog window to allow the user to select the source directory. It then inserts the selected directory path into the entry widget (**`source_entry`**).**
   - `select_dest_dir()`**: Similar to** `select_source_dir()`**, this function opens a file dialog window for the user to select the destination directory and inserts the selected directory path into the entry widget (**`dest_entry`**).**
   - `backup()`**: This function retrieves the paths of the source and destination directories from the entry widgets. It then attempts to perform a backup operation using** `shutil.copytree()`**, which recursively copies the entire directory tree from the source to the destination. If the backup is successful, it updates the status label (**`status_label`**) with a success message; otherwise, it displays an error message.**

3. **Main GUI Setup:**
   - `root`: This initializes the main window of the GUI.
   - `root.title()`: Sets the title of the window.
   - `root.geometry()`: Sets the initial size of the window.
   - `root.configure(bg="#f0f0f0")`: Sets the background color of the window.
   - `heading_label`: Displays a heading at the top of the window.
   - Entry widgets (`source_entry` and `dest_entry`) with corresponding labels (`source_label` and `dest_label`) allow users to input or display directory paths.
   - Buttons (`source_button`, `dest_button`, and `backup_button`) trigger actions when clicked.
   - `status_label`: Provides feedback to the user about the status of the backup operation.
4. **GUI Event Loop:**
   - `root.mainloop()`: Starts the Tkinter event loop, allowing the GUI to respond to user interactions and events.

Overall, this script creates a simple and user-friendly interface for selecting source and destination directories and performing a directory backup operation.

When we Click on REPAIR Button , then repair.py file EXECUTE.



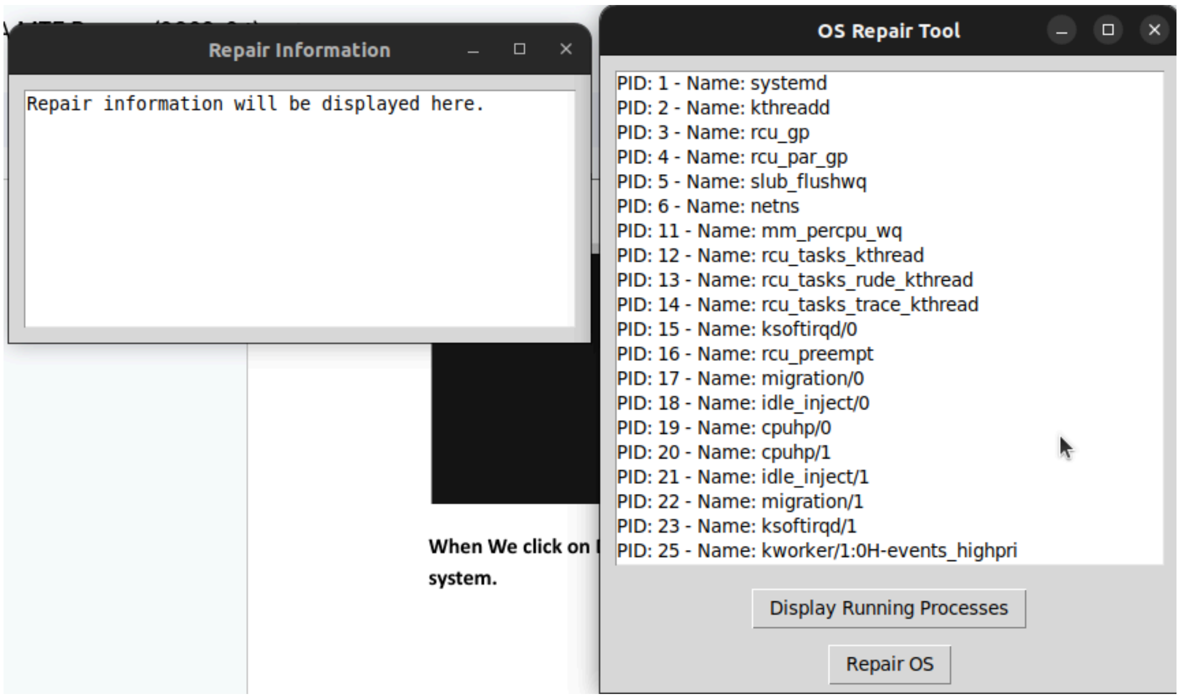


```
Repair information will be displayed here.
```

```
    repair_info_window.
    repair_info_display
    repair_info_display
    repair_info_display
    repair_info_display

    root.mainloop()
```

```
● gautam@gautam-Vostro-15-35
  /bin/python3 "/home/gautam
  /bin/python3 "/home/gautam
● gautam@gautam-Vostro-15-35
● gautam@gautam-Vostro-15-35
● gautam@gautam-Vostro-15-3568:~/BAR Project$ /bin/python3 "/home/gautam/BAR Project/
```

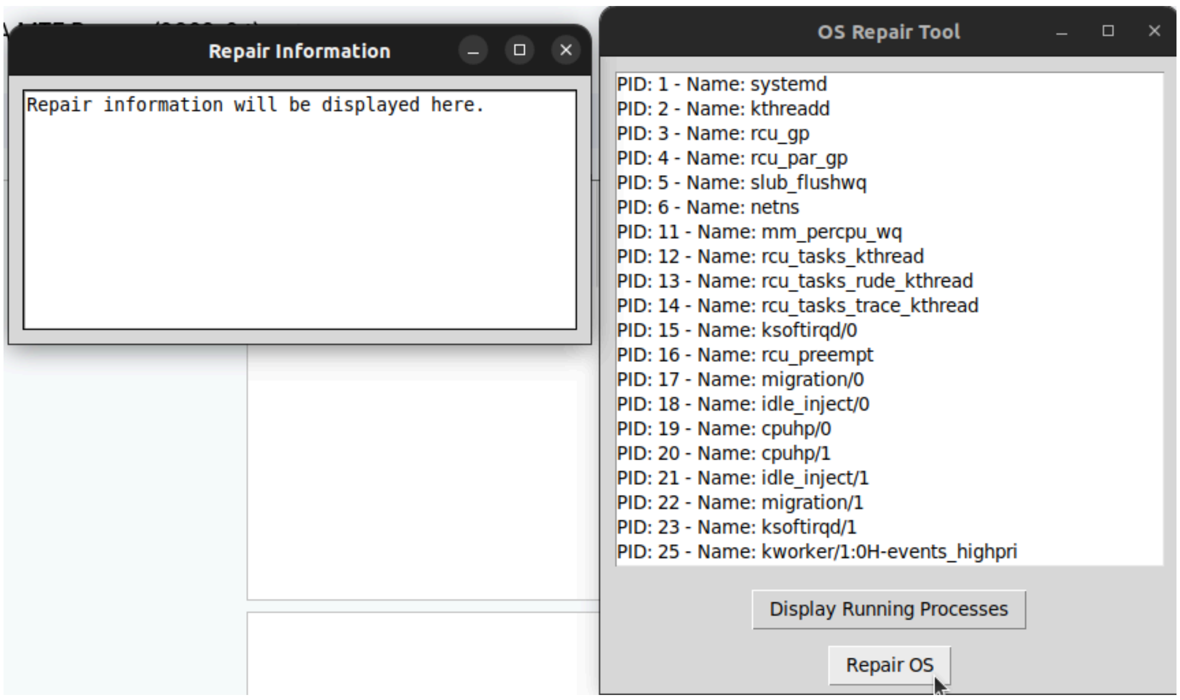
When We click on Display Running Process , It will Display All the running Process in the system.



```
Repair information will be displayed here.
```

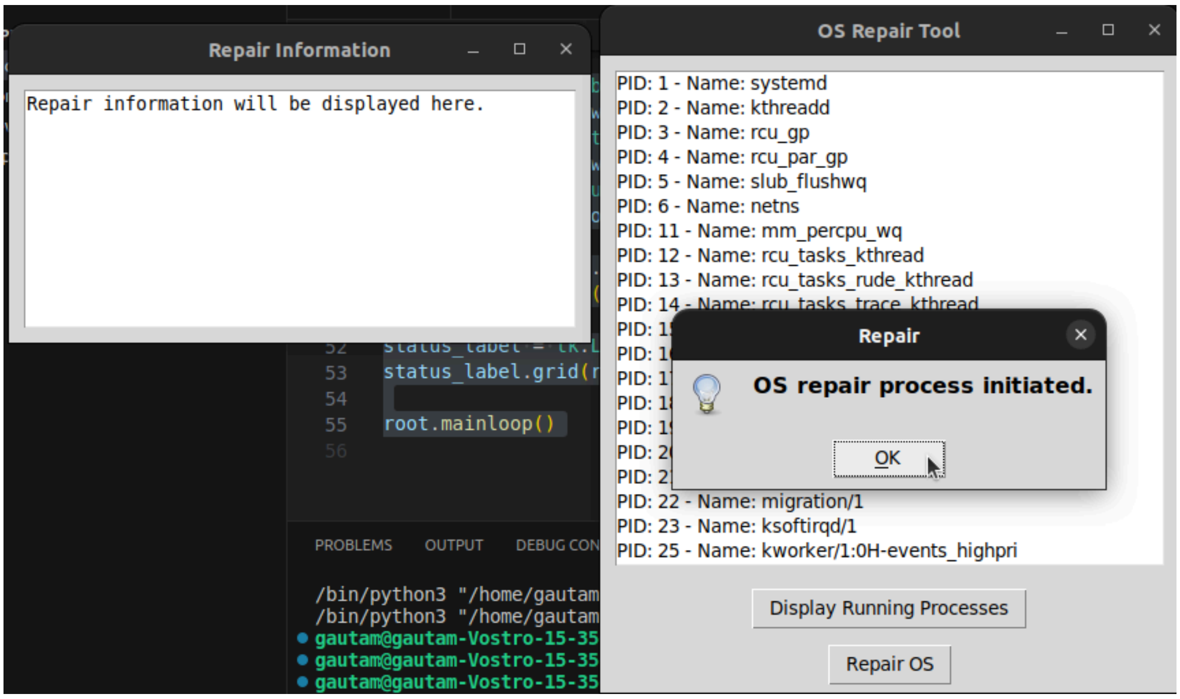
```
PID: 1 - Name: systemd
PID: 2 - Name: kthreadd
PID: 3 - Name: rcu_gp
PID: 4 - Name: rcu_par_gp
PID: 5 - Name: slub_flushwq
PID: 6 - Name: netns
PID: 11 - Name: mm_percpu_wq
PID: 12 - Name: rcu_tasks_kthread
PID: 13 - Name: rcu_tasks_rude_kthread
PID: 14 - Name: rcu_tasks_trace_kthread
PID: 15 - Name: ksoftirqd/0
PID: 16 - Name: rcu_preempt
PID: 17 - Name: migration/0
PID: 18 - Name: idle_inject/0
PID: 19 - Name: cpuhp/0
PID: 20 - Name: cpuhp/1
PID: 21 - Name: idle_inject/1
PID: 22 - Name: migration/1
PID: 23 - Name: ksoftirqd/1
PID: 25 - Name: kworker/1:0H-events_highpri
```

When We Click On Repair , It will Start Repairing the OS and Diagnosis all the Files in OS.



```
Repair information will be displayed here.
```

```
PID: 1 - Name: systemd
PID: 2 - Name: kthreadd
PID: 3 - Name: rcu_gp
PID: 4 - Name: rcu_par_gp
PID: 5 - Name: slub_flushwq
PID: 6 - Name: netns
PID: 11 - Name: mm_percpu_wq
PID: 12 - Name: rcu_tasks_kthread
PID: 13 - Name: rcu_tasks_rude_kthread
PID: 14 - Name: rcu_tasks_trace_kthread
PID: 15 - Name: ksoftirqd/0
PID: 16 - Name: rcu_preempt
PID: 17 - Name: migration/0
PID: 18 - Name: idle_inject/0
PID: 19 - Name: cpuhp/0
PID: 20 - Name: cpuhp/1
PID: 21 - Name: idle_inject/1
PID: 22 - Name: migration/1
PID: 23 - Name: ksoftirqd/1
PID: 25 - Name: kworker/1:0H-events_highpri
```



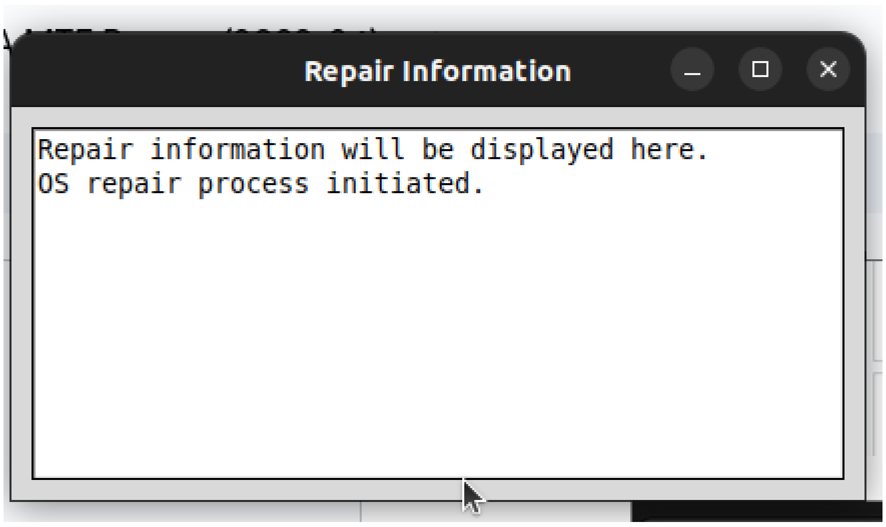
```
Repair information will be displayed here.
```

```
status_label.grid(r
root.mainloop()
```

```
/bin/python3 "/home/gautam
/bin/python3 "/home/gautam
gautam@gautam-Vostro-15-35
gautam@gautam-Vostro-15-35
gautam@gautam-Vostro-15-35
```

OS repair process initiated.

```
PID: 1 - Name: systemd
PID: 2 - Name: kthreadd
PID: 3 - Name: rcu_gp
PID: 4 - Name: rcu_par_gp
PID: 5 - Name: slub_flushwq
PID: 6 - Name: netns
PID: 11 - Name: mm_percpu_wq
PID: 12 - Name: rcu_tasks_kthread
PID: 13 - Name: rcu_tasks_rude_kthread
PID: 14 - Name: rcu_tasks_trace_kthread
PID: 22 - Name: migration/1
PID: 23 - Name: ksoftirqd/1
PID: 25 - Name: kworker/1:0H-events_highpri
```



```
Repair information will be displayed here.
OS repair process initiated.
```

CODE EXPLANATION-

Certainly! This Python script creates a simple graphical user interface (GUI) application using the Tkinter library for displaying running processes and initiating an operating system repair command. Let's break it down step by step:

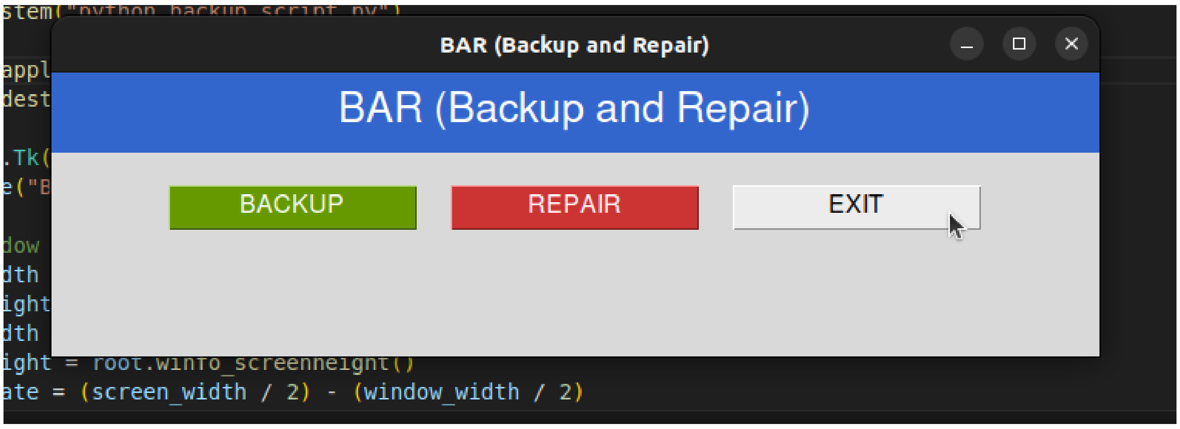
1. Imports:
   - `os`: Provides functions for interacting with the operating system, used here for executing the OS repair command.
   - `psutil`: A cross-platform library for retrieving information on running processes and system utilization.
   - `tkinter`: The standard Python interface to the Tk GUI toolkit, used for creating the GUI elements.
   - `messagebox`: A module in Tkinter that provides methods for displaying message boxes in the GUI.
2. Function Definitions:
   - `display_running_processes()`: Retrieves information about running processes using `psutil.process_iter()`. It iterates over the running processes, retrieves the process ID (`pid`) and name, and populates a list. Then, it updates a listbox widget in the GUI (`process_list`) with the retrieved process information.
   - `repair_os()`: Initiates an operating system repair process. It executes a repair command (`sfc /scannow` in this case) using `os.system()`. After executing the command, it displays an information message box using `messagebox.showinfo()`.

Additionally, it updates a text display widget (`repair_info_display`) to inform the user about the repair process initiation.

3. Main GUI Setup:
   - `root`: Initializes the main window of the GUI.
   - `root.title()`: Sets the title of the main window.
   - Creates a listbox widget (`process_list`) to display running processes.
   - Creates buttons (`display_button` and `repair_button`) to trigger actions.
   - Creates a secondary window (`repair_info_window`) to display repair information using a text widget (`repair_info_display`).
4. GUI Event Loop:
   - `root.mainloop()`: Starts the Tkinter event loop, allowing the GUI to respond to user interactions and events.

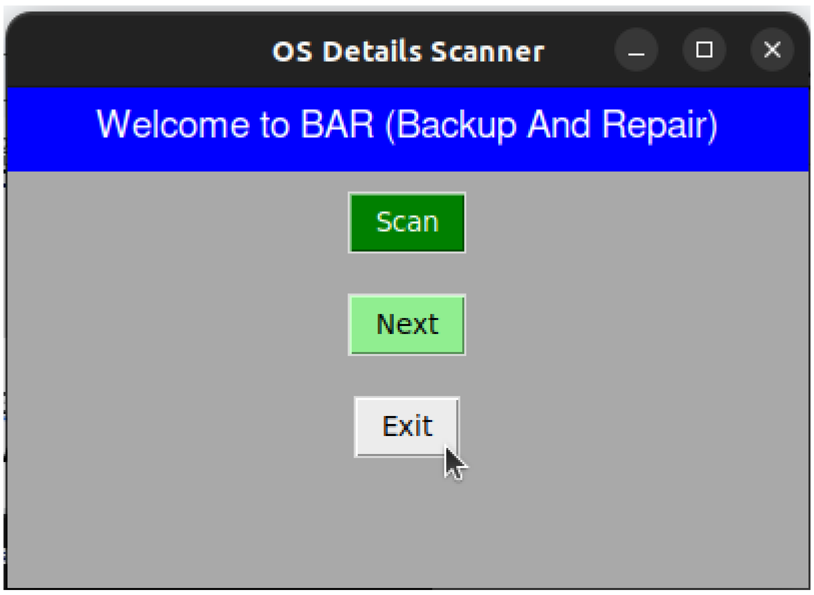
Overall, this script provides a basic interface for users to view running processes and initiate an operating system repair process. It offers visual feedback on the status of the repair process and the list of running processes.

EXIT-BUTTON



```
stem("python backup script py")

appl
dest

.Tk(
e("B

dow
dth
ight
dth
ight = root.winfo_screenheight()
ate = (screen_width / 2) - (window_width / 2)
```

EXIT- When we click on the exit button, It will close the application.



CODE EXPLANATION-

1. Imports:
   - `platform`: This module provides a portable way to access underlying platform data, such as the operating system.
   - `psutil`: This is a cross-platform library for retrieving information on running processes and system utilization (CPU, memory, disks, network, sensors).
   - `tkinter`: This is the standard Python interface to the Tk GUI toolkit.
   - `os`: This module provides a portable way of interacting with the operating system.
2. Function Definitions:
   - `scan()`: This function gathers various system information using `platform` and `psutil` modules, such as operating system name, version, architecture, file system, and total storage. It then displays this information in a message box using `messagebox.showinfo()`.
   - `open_next_program()`: This function constructs the path to another Python script named `neweg.py`, assumed to be in the same directory as the current script. Then, it executes this script using `os.system()`.

3. **Main GUI Setup:**
    - `root`: This initializes the main window of the GUI.
    - `root.title()`: Sets the title of the window.
    - `root.geometry()`: Sets the initial size of the window.
    - `heading_label`: Creates a label widget to display a heading text at the top of the window.
    - `scan_button`, `next_button`, `exit_button`: Create button widgets for "Scan", "Next", and "Exit" actions, respectively. They are configured with custom colors and associated with their corresponding functions.
    - Widgets are packed into the window using `pack()` method.
    - `root.configure(bg="darkgrey")`: Sets the background color of the main window.
4. **GUI Event Loop:**
    - `root.mainloop()`: Starts the Tkinter event loop, which listens for events (like button clicks) and updates the GUI accordingly.

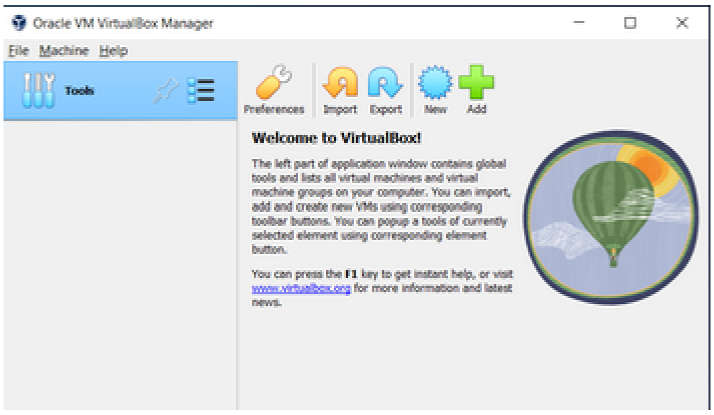
Overall, this code creates a simple GUI application that allows users to scan system information and execute another Python script. The GUI elements are styled with custom colors for better visual appeal.

STEPS FOR THE WHOLE PRACTICAL OF BAR(BACKUP AND REPAIR) SOFTWARE.

Here are step-by-step instructions for downloading VirtualBox, installing Windows 10 on it, intentionally crashing the Windows 10 virtual machine with the help of a simulated virus, and then performing backup and repair:

1. Download VirtualBox:
    - Go to the official VirtualBox website: https://www.virtualbox.org/
    - Click on the "Downloads" menu.
    - Choose the appropriate version for your operating system (Windows, macOS, Linux) and click on the download link.
    - Once the download is complete, run the installer and follow the installation instructions to install VirtualBox on your computer.
2. Download Windows 10 ISO:
    - Go to the official Microsoft website: https://www.microsoft.com/en-us/software-download/windows10
    - Click on "Download tool now" to download the Windows 10 Media Creation Tool.
    - Run the tool and follow the on-screen instructions to create a bootable USB drive or download the Windows 10 ISO file.
3. Create a Virtual Machine in VirtualBox:
    - Open VirtualBox.
    - Click on "New" to create a new virtual machine.

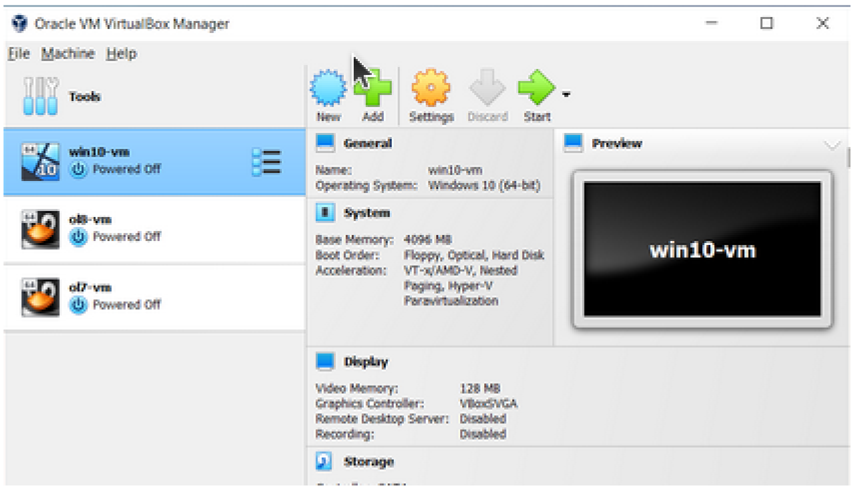
   - Follow the wizard to set up the virtual machine:
      - Name the virtual machine (e.g., "Windows 10").
      - Choose "Windows 10 (64-bit)" as the operating system type.
      - Allocate memory (RAM) for the virtual machine.
      - Create a virtual hard disk (VHD) or use an existing one.
4. Install Windows 10:
   - Select the newly created virtual machine in VirtualBox and click on "Start".
   - When prompted, select the Windows 10 ISO file as the bootable media.
   - Follow the on-screen instructions to install Windows 10 on the virtual machine. This process is similar to installing Windows on a physical computer.



Welcome to VirtualBox!

The left part of application window contains global tools and lists all virtual machines and virtual machine groups on your computer. You can import, add and create new VMs using corresponding toolbar buttons. You can popup a tools of currently selected element using corresponding element button.

You can press the **F1** key to get instant help, or visit www.virtualbox.org for more information and latest news.



win10-vm
Powered Off

ol8-vm
Powered Off

ol7-vm
Powered Off

General

```
Name:               win10-vm
Operating System:   Windows 10 (64-bit)
```

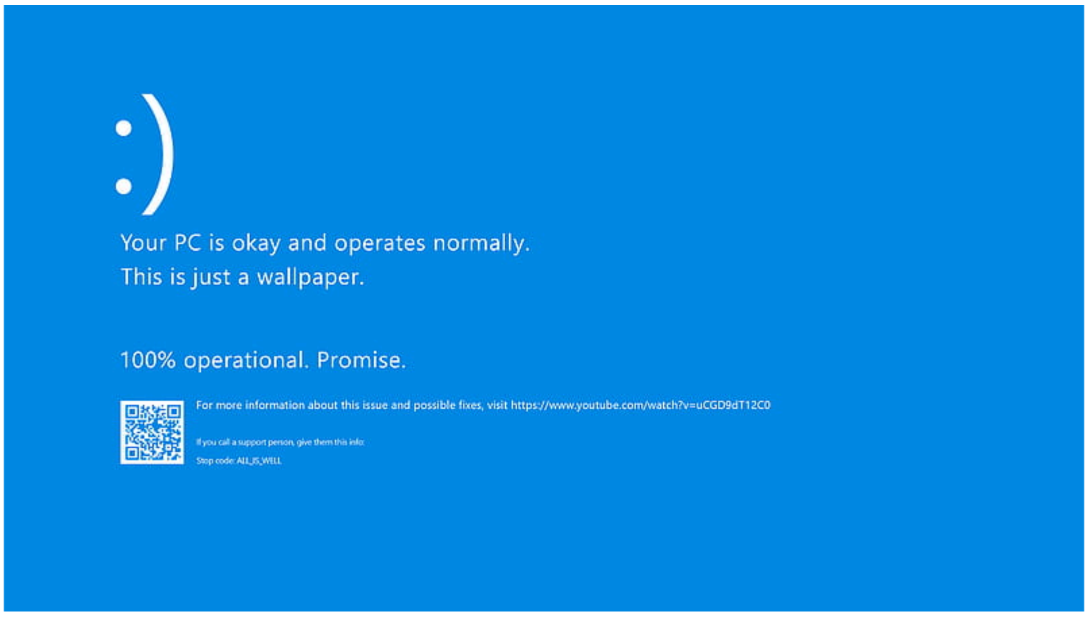
System

```
Base Memory:   4096 MB
Boot Order:    Floppy, Optical, Hard Disk
Acceleration:  VT-x/AMD-V, Nested
               Paging, Hyper-V
               Paravirtualization
```

Display

```
Video Memory:             128 MB
Graphics Controller:      VBoxSVGA
Remote Desktop Server:    Disabled
Recording:                Disabled
```

Storage

5. Intentionally Crash Windows 10:
   - Once Windows 10 is installed, simulate a virus infection or crash by downloading a harmless script or program from a trusted source that will trigger a crash.
   - Run the script or program on the Windows 10 virtual machine to cause it to crash.
   - 

:)

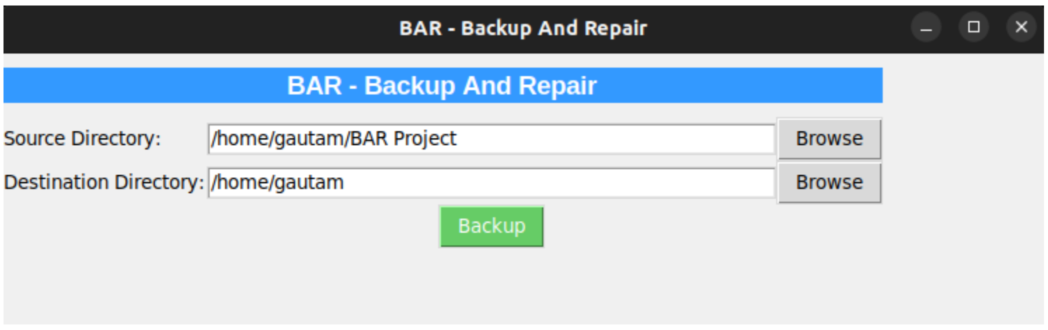
Your PC is okay and operates normally.
This is just a wallpaper.

100% operational. Promise.

For more information about this issue and possible fixes, visit https://www.youtube.com/watch?v=uCGD9dT12C0

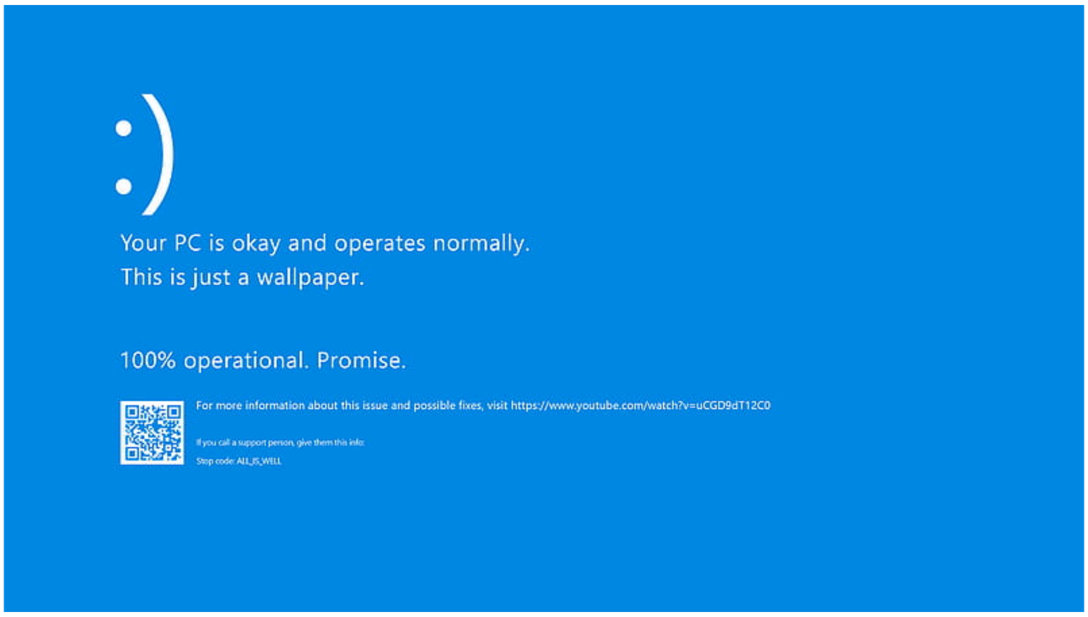
If you call a support person, give them this info:
Stop code: ALL_IS_WELL

6. Perform Backup:
   - Before repairing the crashed Windows 10 virtual machine, perform a backup of any important data or settings within the virtual machine. You can back up files by copying them to a shared folder on the host machine or by using backup software within the virtual machine.



Source Directory: /home/gautam/BAR Project
Destination Directory: /home/gautam

7. Repair Windows 10:
   - After backing up important data, initiate the repair process. In this case, you can use the provided repair tool in the GUI application we discussed earlier. Click on the "Repair OS" button to initiate the repair process.
   - Follow the instructions provided by the repair tool to repair the crashed Windows 10 virtual machine. This may involve running diagnostics, system file checks, or restoring from a backup.
8. Verify Repair:
   - Once the repair process is complete, verify that the Windows 10 virtual machine is functioning properly and that the crash has been resolved.

- Check if any data or settings were lost during the repair process and restore them from the backup if necessary.
- 

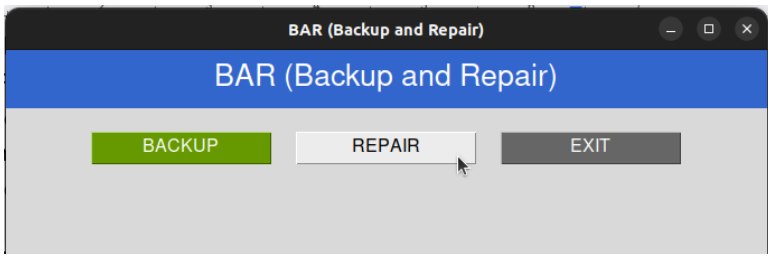
:)

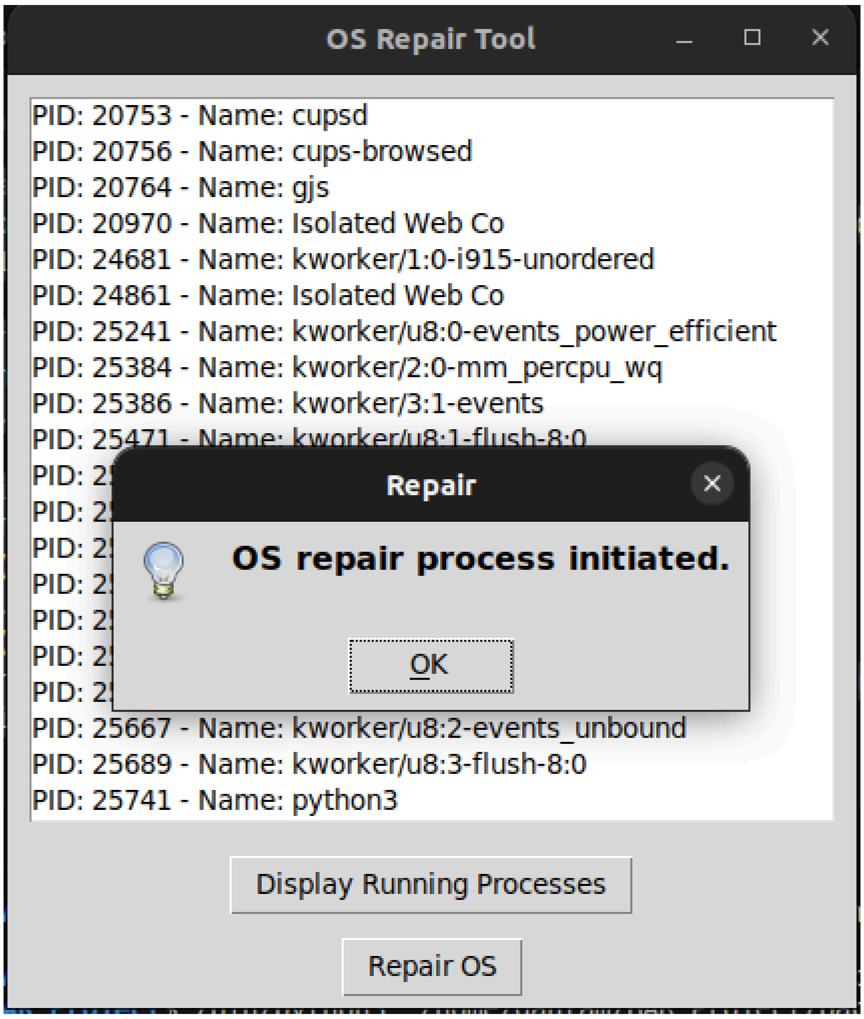
Your PC is okay and operates normally.
This is just a wallpaper.

100% operational. Promise.

For more information about this issue and possible fixes, visit https://www.youtube.com/watch?v=uCGD9dT12C0

If you call a support person, give them this info:
Stop code: ALL_IS_WELL

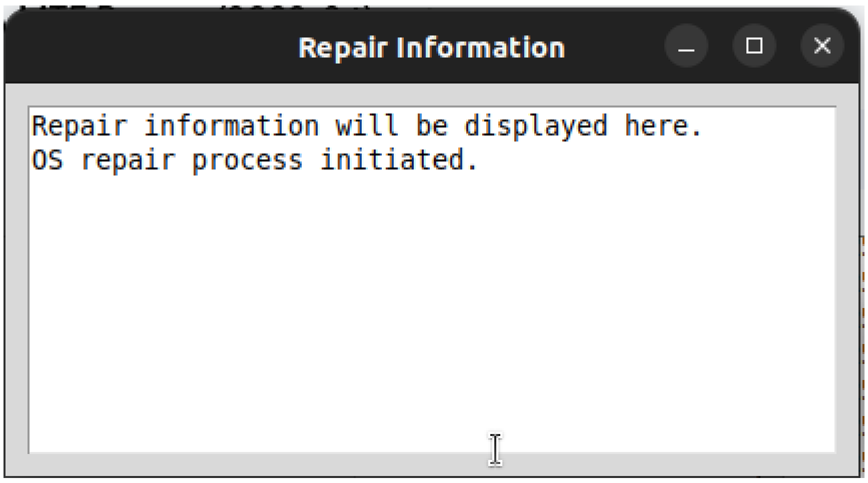
- 



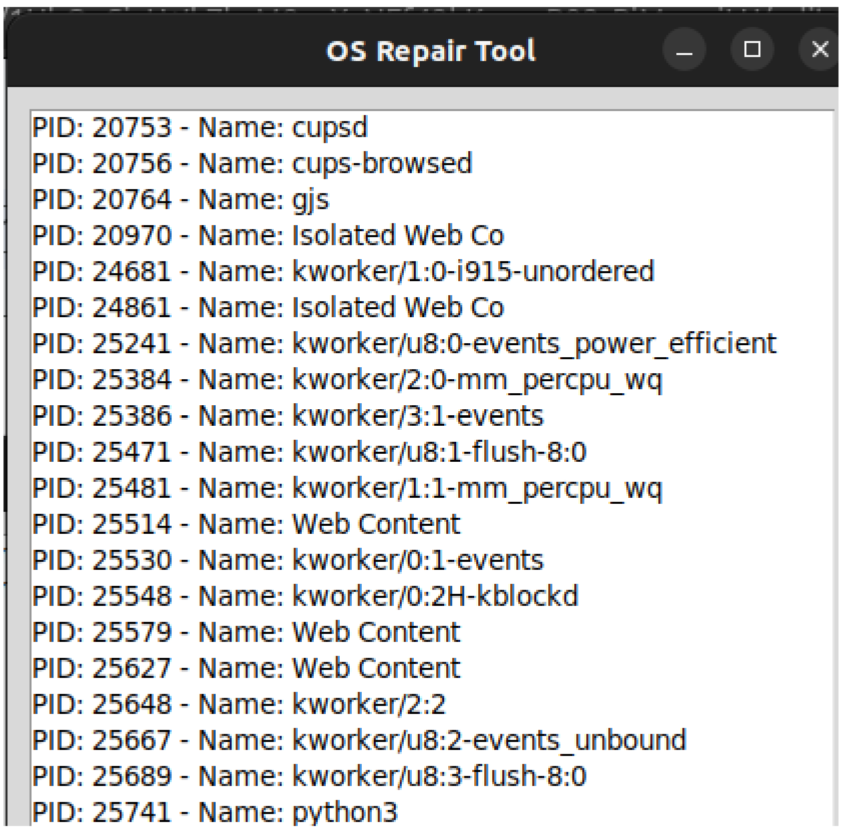
```
PID: 20753 - Name: cupsd
PID: 20756 - Name: cups-browsed
PID: 20764 - Name: gjs
PID: 20970 - Name: Isolated Web Co
PID: 24681 - Name: kworker/1:0-i915-unordered
PID: 24861 - Name: Isolated Web Co
PID: 25241 - Name: kworker/u8:0-events_power_efficient
PID: 25384 - Name: kworker/2:0-mm_percpu_wq
PID: 25386 - Name: kworker/3:1-events
PID: 25471 - Name: kworker/u8:1-flush-8:0
PID: 2
PID: 2
PID: 2
PID: 2
PID: 2
PID: 2
PID: 2
PID: 25667 - Name: kworker/u8:2-events_unbound
PID: 25689 - Name: kworker/u8:3-flush-8:0
PID: 25741 - Name: python3
```

OS repair process initiated.

○



```
Repair information will be displayed here.
OS repair process initiated.
```



OS Repair Tool

```
PID: 20753 - Name: cupsd
PID: 20756 - Name: cups-browsed
PID: 20764 - Name: gjs
PID: 20970 - Name: Isolated Web Co
PID: 24681 - Name: kworker/1:0-i915-unordered
PID: 24861 - Name: Isolated Web Co
PID: 25241 - Name: kworker/u8:0-events_power_efficient
PID: 25384 - Name: kworker/2:0-mm_percpu_wq
PID: 25386 - Name: kworker/3:1-events
PID: 25471 - Name: kworker/u8:1-flush-8:0
PID: 25481 - Name: kworker/1:1-mm_percpu_wq
PID: 25514 - Name: Web Content
PID: 25530 - Name: kworker/0:1-events
PID: 25548 - Name: kworker/0:2H-kblockd
PID: 25579 - Name: Web Content
PID: 25627 - Name: Web Content
PID: 25648 - Name: kworker/2:2
PID: 25667 - Name: kworker/u8:2-events_unbound
PID: 25689 - Name: kworker/u8:3-flush-8:0
PID: 25741 - Name: python3
```

By following these steps, you'll be able to download VirtualBox, install Windows 10 on it, intentionally crash the virtual machine, perform a backup of important data, and repair the crashed Windows 10 virtual machine.

6.CONCLUSION

In conclusion, the development of a device to repair corrupted operating systems represents a significant stride towards addressing the challenges associated with system integrity. The proposed device, equipped with advanced automatic repair mechanisms, an intuitive user interface, and robust data recovery support, stands as a reliable solution for mitigating the impact of OS corruption.

In essence, the device to repair corrupted operating systems represents a significant contribution to the field, offering users a reliable, user-friendly, and adaptable solution to navigate the challenges posed by OS corruption. As technology evolves, the commitment to innovation and user satisfaction will remain at the forefront, ensuring the device's continued relevance and effectiveness in the ever-changing landscape of digital computing.

7. REFERENCES

1. Create a Bootable USB Drive:
    - You can use tools like Rufus (for Windows) or Etcher (for various operating systems) to create a bootable USB drive. Rufus: https://rufus.ie/, Etcher: https://www.balena.io/etcher/
2. Choose a Rescue or Repair Tool:
    - There are several tools available for rescuing or repairing corrupted operating systems. Some popular ones include:
        - Hiren's BootCD: A comprehensive toolkit with a variety of tools for troubleshooting and repairing. Website: http://www.hirensbootcd.org/
        - Ultimate Boot CD (UBCD): Another versatile toolkit with a range of diagnostic and repair tools. Website: https://www.ultimatebootcd.com/
        - SystemRescueCd: A Linux-based rescue disk with tools for both Linux and Windows systems. Website: https://www.system-rescue.org/
3. Use Windows Installation Media:
    - If you're dealing with a corrupted Windows OS, you can use the official Windows installation media to repair the system. Boot from the USB/DVD, choose the "Repair your computer" option, and access troubleshooting tools.
4. Linux Live USB:
    - Creating a live USB with a Linux distribution like Ubuntu can be a powerful tool. Boot into the live environment and use tools like GParted for disk management or other Linux utilities for system repair.
5. Diagnostic and Repair Commands:
    - Familiarize yourself with basic diagnostic and repair commands. For example, in a Windows environment, you might use commands like `chkdsk`, `sfc /scannow`, or `bootrec /rebuildbcd`. For Linux, commands like

`fsck` for file system check and `grub-install` for reinstalling the GRUB bootloader can be helpful.

6. Online Forums and Documentation:
   - Check online forums and documentation specific to the operating system you're working with. Communities often share their experiences and solutions for repairing and troubleshooting common issues.

Operating System Internals:

- Book: "Operating System Concepts" by Abraham Silberschatz, Peter B. Galvin, and Greg Gagne.
- Book: "Modern Operating Systems" by Andrew S. Tanenbaum and Herbert Bos.
- Online Course: "Operating Systems: Three Easy Pieces" by Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau.

Programming Languages and Frameworks:

- C/C++: Many low-level system utilities and tools are written in C or C++ due to their performance and low-level access to system resources.
- Python: Python is a high-level language suitable for building system administration tools and scripts. Libraries like `psutil` can be used to interact with system processes and resources.
- PowerShell: If you're targeting Windows systems, PowerShell provides powerful scripting capabilities for system administration tasks.

System Administration and Security:

- Book: "Windows Internals" by Mark E. Russinovich and David A. Solomon.
- Book: "Linux System Programming: Talking Directly to the Kernel and C Library" by Robert Love.
- Online Resources: Documentation and guides provided by Microsoft for Windows system administration and security best practices.
- Online Resources: Official documentation and community resources for Linux system administration and security.

System Repair Techniques:

- Understand common issues that can occur in operating systems, such as file system corruption, boot problems, and malware infections.
- Learn about tools and techniques used for system diagnostics, recovery, and repair, such as CHKDSK for Windows and fsck for Linux.

Software Development Practices:

- Follow software development best practices, including version control, testing, and documentation.
- Consider security implications and implement appropriate measures to protect user data and system integrity.

Open Source Projects and Tools:

    - Study existing open-source operating system repair tools and utilities to understand their implementation and design principles.
    - Contribute to open-source projects related to system administration and repair to gain practical experience and knowledge.

Always ensure that you have a good backup of your important data before attempting any repair procedures, as some operations may result in data loss.