

TABLE OF CONTENTS

Chapter 1: Introduction

1.1	Introduction.....	Page No.5
1.1.1	User Account Management.....	Page No.5
1.1.2	Privilege Management.....	Page No.5
1.1.3	Root Privilege.....	Page No.6
1.1.4	System Security.....	Page No.6
1.1.5	Inter Process Communication.....	Page No.7
1.2	Objectives.....	Page No.7
1.3	Scope of the project.....	Page No.9
1.4	Proposed System.....	Page No.10

Chapter 2: Implementation

2.1	Program Coding	Page No.11
2.2	Output.....	Page No.22

Chapter 3: Conclusion

References

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

In an increasingly digital world, the effective management of user accounts within software systems has become paramount. From ensuring data security and privacy to providing seamless user experiences, a robust User Account Management System (UAMS) plays a pivotal role in modern software applications. This report presents the design, development, and implementation of a UAMS tailored to address the evolving needs of our organization's software ecosystem. In an increasingly digital world, the effective management of user accounts within software systems has become paramount. From ensuring data security and privacy to providing seamless user experiences, a robust User Account Management System (UAMS) plays a pivotal role in modern software applications. This report presents the design, development, and implementation of a UAMS tailored to address the evolving needs of our organization's software ecosystem.

1.1.1 USER ACCOUNT MANAGEMENT

User account management is a fundamental aspect of operating system administration. In this project, handling user creation and deletion is achieved using the `useradd` and `userdel` commands. These commands allow the system administrator to add new users and remove existing users, respectively.

useradd:

- This command is used to create a new user account. It sets up the new user's home directory, default shell, and other configurations.

userdel:

- This command removes a user account and can optionally remove the user's home directory and mail spool.

In the project, the `on_create_user` function utilizes `useradd` to add a new user. It builds a command string that includes the root password (retrieved securely) and executes it using the `system()` function. Similarly, `on_delete_user` constructs a command with `userdel` to remove a user.

1.1.2 PRIVILEGE MANAGEMENT

Privilege management involves assigning specific permissions and rights to users or groups of users. This ensures that users have the necessary access to perform their roles without compromising system security.

Group Membership:

- Adding users to groups like sudo allows them to execute commands with elevated privileges.

Usermod:

- The usermod command is used to modify a user's account, including their group memberships.

In the project, the function `ask_root_permission_and_privileges` handles privilege management. It allows the root user to grant additional privileges to new users, such as adding them to the sudo group, which is then executed via the usermod command within the `on_create_user` function if specified.

1.1.3 ROOT PRIVILEGE

Root privileges are necessary for executing administrative tasks that regular users cannot perform. Secure handling of the root password and execution of commands with sudo is crucial for maintaining system integrity.

Root Authentication:

- The project prompts for the root password when performing tasks that require elevated privileges.

Sudo Command:

- Using sudo, the system executes commands with root privileges without exposing the root account to regular users.

The `get_root_password` function requests the root password securely using a GTK dialog. The password is then used to execute commands with sudo in a safe manner, ensuring that only authorized users can perform sensitive operations.

1.1.4 SYSTEM SECURITY

System security is paramount, especially when dealing with user credentials and executing system commands. The project ensures secure password handling and command execution to prevent unauthorized access and potential security breaches.

Password Handling:

- User passwords are entered securely via GTK entries, and root passwords are not stored or displayed.

Secure Execution:

- Commands are constructed carefully to avoid injection attacks and are executed using `sudo` to ensure only authorized actions are performed.

The project employs GTK dialogs for secure password input and verifies root permissions before executing any sensitive operations. This prevents unauthorized users from gaining elevated privileges.

1.1.5 INTER PROCESS COMMUNICATION

Interprocess communication (IPC) refers to the mechanisms an operating system provides to allow processes to manage shared data. In this project, IPC is used to execute system commands and retrieve their output.

system() Function:

- This function is used to execute shell commands from within the program. It is used for both creating and deleting users.

popen() Function:

- This function opens a process by creating a pipe, forking, and invoking the shell. It is used to list users by executing a command that reads the `/etc/passwd` file.

In the project, the `on_list_users` function uses `popen()` to execute the `cut -d: -f1 /etc/passwd` command, which extracts and displays usernames. The output is read through a pipe and displayed in the GTK text view widget, providing a real-time list of users on the system.

1.2 OBJECTIVES OF THE PROJECT

The User Account Management System (UAMS) project aims to develop a secure, efficient, and user-friendly GUI application for managing user accounts on a Linux system. The system will provide an intuitive interface that simplifies the creation, deletion, and listing of user accounts, making these tasks accessible even to administrators with limited command-line experience. Security is a primary focus, with the implementation of robust authentication mechanisms to ensure that only authorized users can perform privileged operations. This includes secure handling of root passwords and user credentials.

Additionally, the UAMS will facilitate efficient assignment and modification of user privileges, allowing for flexible and granular control over user permissions. The system will provide clear feedback through error and success messages, enhancing the user experience and keeping administrators informed about their actions. Seamless integration with existing Linux system commands and tools will ensure compatibility and leverage the strengths of the underlying OS. Comprehensive documentation will be developed for both users and developers, ensuring ease of use and maintenance. By achieving these objectives, the UAMS project will streamline user account management, enhance security, and improve administrative efficiency on Linux systems.

1.3 SCOPE OF THE PROJECT

The User Account Management System (UAMS) project involves designing, developing, implementing, and testing a software application that provides a graphical user interface for managing user accounts on a Linux system. The project will include functionalities for creating new user accounts, specifying usernames and passwords, and assigning privileges. It will also provide functionality for deleting existing user accounts and displaying a list of all current user accounts on the system.

The project will support assigning various privileges to user accounts, such as adding users to groups like sudo for administrative rights, and specifying additional usermod options for customizing user privileges. A secure mechanism for root user authentication will be implemented to perform privileged operations, which includes prompting for and securely handling the root password.

A user-friendly graphical interface will be developed using the GTK library, featuring windows, dialog boxes, entry fields, buttons, and text views. The application will provide clear and informative error messages for failed operations and success messages for completed operations. Integration with the Linux system's command-line tools for user management (e.g., useradd, userdel, usermod, chpasswd) will be achieved by securely constructing and executing system commands.

Security considerations will include securely handling and transmitting passwords and ensuring that only authorized users can perform privileged operations through proper authentication and permission checks. The project will include user documentation with instructions on using the UAMS and developer documentation for codebase maintenance and future enhancements.

Testing and validation will encompass functional testing to ensure all functionalities work as expected, security testing to verify proper securing of sensitive operations and data, and usability testing to ensure the interface is intuitive and easy to use. Potential future enhancements may include additional features like more granular role-based access control, integration with other authentication systems (e.g., LDAP), and enhanced reporting features.

Overall, the scope of this project ensures effective user account management on Linux systems with a focus on security, usability, and system integration.

1.4 PROPOSED SYSTEM

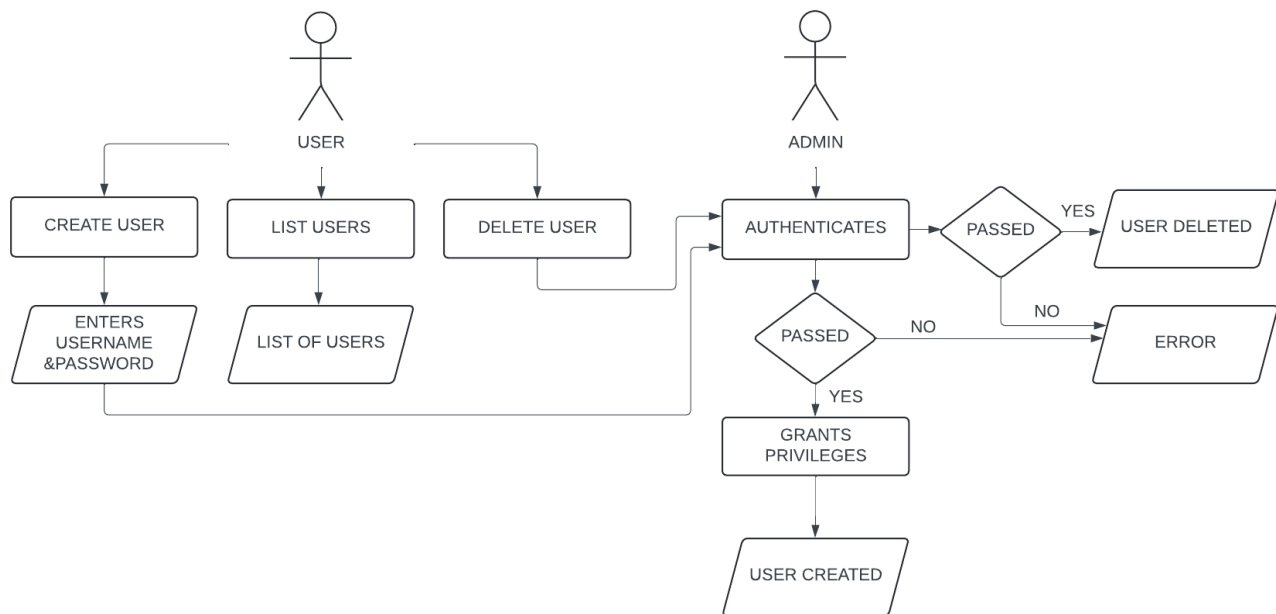


Figure 1.4.1

CHAPTER 2

IMPLEMENTATION

2.1 PROGRAM CODING :

```
#include <gtk/gtk.h>
#include <stdlib.h>
#include <string.h>

// Function prototypes
void on_create_user(GtkWidget *widget, gpointer data);
void on_delete_user(GtkWidget *widget, gpointer data);
void on_list_users(GtkWidget *widget, gpointer data);
char* get_root_password();
gboolean ask_root_permission_and_privileges(const char *username, char *privileges, size_t
priv_size);

void on_create_user(GtkWidget *widget, gpointer data) {
    GtkWidget **entries = (GtkWidget **)data;
    const char *username = gtk_entry_get_text(GTK_ENTRY(entries[0]));
    const char *password = gtk_entry_get_text(GTK_ENTRY(entries[1]));

    char privileges[256];
    // Ask for root permission and privileges
    if (!ask_root_permission_and_privileges(username, privileges, sizeof(privileges))) {
        GtkWidget *dialog = gtk_message_dialog_new(NULL,
GTK_DIALOG_DESTROY_WITH_PARENT,
GTK_MESSAGE_ERROR, GTK_BUTTONS_OK,
"Permission denied by root user.");
        gtk_dialog_run(GTK_DIALOG(dialog));
        gtk_widget_destroy(dialog);
    }
}
```



```

    return;
}

// Get root password
char *root_password = get_root_password();
if (root_password == NULL) {
    GtkWidget *dialog = gtk_message_dialog_new(NULL,
GTK_DIALOG_DESTROY_WITH_PARENT,
                                GTK_MESSAGE_ERROR, GTK_BUTTONS_OK,
                                "Root authentication failed.");

    gtk_dialog_run(GTK_DIALOG(dialog));
    gtk_widget_destroy(dialog);
    return;
}

// Create user
char command[512]; // Increased buffer size
snprintf(command, sizeof(command), "echo %s | sudo -S useradd %s", root_password, username);
if (system(command) == 0) {
    snprintf(command, sizeof(command), "echo %s | sudo -S bash -c 'echo \"%s:%s\" | chpasswd'",
root_password, username, password);
    if (system(command) == 0) {
        // Grant privileges if specified
        if (strlen(privileges) > 0) {
            snprintf(command, sizeof(command), "echo %s | sudo -S usermod %s %s", root_password,
privileges, username);
            if (system(command) != 0) {
                GtkWidget *dialog = gtk_message_dialog_new(NULL,
GTK_DIALOG_DESTROY_WITH_PARENT,
                                GTK_MESSAGE_WARNING, GTK_BUTTONS_OK,
                                "User %s created, but failed to set some privileges.", username);

                gtk_dialog_run(GTK_DIALOG(dialog));

```

```

        gtk_widget_destroy(dialog);
    }
}

GtkWidget *dialog = gtk_message_dialog_new(NULL,
GTK_DIALOG_DESTROY_WITH_PARENT,
        GTK_MESSAGE_INFO, GTK_BUTTONS_OK,
        "User %s created successfully with password %s.",
        username, password);

gtk_dialog_run(GTK_DIALOG(dialog));
gtk_widget_destroy(dialog);
} else {
    GtkWidget *dialog = gtk_message_dialog_new(NULL,
GTK_DIALOG_DESTROY_WITH_PARENT,
        GTK_MESSAGE_ERROR, GTK_BUTTONS_OK,
        "Failed to set password for user %s.", username);

    gtk_dialog_run(GTK_DIALOG(dialog));
    gtk_widget_destroy(dialog);
}
} else {
    GtkWidget *dialog = gtk_message_dialog_new(NULL,
GTK_DIALOG_DESTROY_WITH_PARENT,
        GTK_MESSAGE_ERROR, GTK_BUTTONS_OK,
        "Failed to create user %s.", username);

    gtk_dialog_run(GTK_DIALOG(dialog));
    gtk_widget_destroy(dialog);
}

// Clean up
free(root_password);
}

char* get_root_password() {

```

```

GtkWidget *dialog, *content_area;
GtkWidget *label, *password_entry;
GtkDialogFlags flags = GTK_DIALOG_MODAL | GTK_DIALOG_DESTROY_WITH_PARENT;
char *password;

dialog = gtk_dialog_new_with_buttons("Root Authentication",
                                     NULL,
                                     flags,
                                     ("_OK"),
                                     GTK_RESPONSE_OK,
                                     ("_Cancel"),
                                     GTK_RESPONSE_CANCEL,
                                     NULL);

content_area = gtk_dialog_get_content_area(GTK_DIALOG(dialog));
label = gtk_label_new("Enter root password:");
password_entry = gtk_entry_new();
gtk_entry_set_visibility(GTK_ENTRY(password_entry), FALSE);
gtk_container_add(GTK_CONTAINER(content_area), label);
gtk_container_add(GTK_CONTAINER(content_area), password_entry);
gtk_widget_show_all(dialog);

gint response = gtk_dialog_run(GTK_DIALOG(dialog));
if (response == GTK_RESPONSE_OK) {
    const char *text = gtk_entry_get_text(GTK_ENTRY(password_entry));
    password = strdup(text);
} else {
    password = NULL;
}

gtk_widget_destroy(dialog);
return password;

```

```
}
```

```
gboolean ask_root_permission_and_privileges(const char *username, char *privileges, size_t
priv_size) {
    GtkWidget *dialog, *content_area;
    GtkWidget *label, *admin_check, *sudo_check, *other_entry;
    GtkDialogFlags flags = GTK_DIALOG_MODAL | GTK_DIALOG_DESTROY_WITH_PARENT;

    dialog = gtk_dialog_new_with_buttons("Root Permission",
                                         NULL,
                                         flags,
                                         ("_Grant"),
                                         GTK_RESPONSE_OK,
                                         ("_Deny"),
                                         GTK_RESPONSE_CANCEL,
                                         NULL);

    content_area = gtk_dialog_get_content_area(GTK_DIALOG(dialog));
    char message[256];
    snprintf(message, sizeof(message), "Do you grant permission to create user '%s'? Select privileges:",
username);
    label = gtk_label_new(message);
    gtk_container_add(GTK_CONTAINER(content_area), label);

    admin_check = gtk_check_button_new_with_label("Admin (add to sudo group)");
    gtk_container_add(GTK_CONTAINER(content_area), admin_check);

    sudo_check = gtk_check_button_new_with_label("Sudo (grant sudo privileges)");
    gtk_container_add(GTK_CONTAINER(content_area), sudo_check);

    label = gtk_label_new("Other (specify usermod options):");
    gtk_container_add(GTK_CONTAINER(content_area), label);
```

```

other_entry = gtk_entry_new();
gtk_container_add(GTK_CONTAINER(content_area), other_entry);

gtk_widget_show_all(dialog);

gint response = gtk_dialog_run(GTK_DIALOG(dialog));
if (response == GTK_RESPONSE_OK) {
    privileges[0] = '\0'; // Initialize privileges string

    if (gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(admin_check))) {
        strncat(privileges, "-aG sudo", priv_size - strlen(privileges) - 1);
    }

    if (gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(sudo_check))) {
        strncat(privileges, " -aG sudo", priv_size - strlen(privileges) - 1);
    }

    const char *other_privs = gtk_entry_get_text(GTK_ENTRY(other_entry));
    if (strlen(other_privs) > 0) {
        strncat(privileges, " ", priv_size - strlen(privileges) - 1);
        strncat(privileges, other_privs, priv_size - strlen(privileges) - 1);
    }
} else {
    gtk_widget_destroy(dialog);
    return FALSE;
}

gtk_widget_destroy(dialog);
return TRUE;
}

```

```

void on_delete_user(GtkWidget *widget, gpointer data) {
    GtkWidget *entry = (GtkWidget *)data;
    const char *username = gtk_entry_get_text(GTK_ENTRY(entry));

    // Get root password
    char *root_password = get_root_password();
    if (root_password == NULL) {
        GtkWidget *dialog = gtk_message_dialog_new(NULL,
GTK_DIALOG_DESTROY_WITH_PARENT,
                                GTK_MESSAGE_ERROR, GTK_BUTTONS_OK,
                                "Root authentication failed.");
        gtk_dialog_run(GTK_DIALOG(dialog));
        gtk_widget_destroy(dialog);
        return;
    }

    // Delete user
    char command[512]; // Increased buffer size
    snprintf(command, sizeof(command), "echo %s | sudo -S userdel %s", root_password, username);
    if (system(command) == 0) {
        GtkWidget *dialog = gtk_message_dialog_new(NULL,
GTK_DIALOG_DESTROY_WITH_PARENT,
                                GTK_MESSAGE_INFO, GTK_BUTTONS_OK,
                                "User %s deleted successfully.", username);
        gtk_dialog_run(GTK_DIALOG(dialog));
        gtk_widget_destroy(dialog);
    } else {
        GtkWidget *dialog = gtk_message_dialog_new(NULL,
GTK_DIALOG_DESTROY_WITH_PARENT,
                                GTK_MESSAGE_ERROR, GTK_BUTTONS_OK,
                                "Failed to delete user %s.", username);
        gtk_dialog_run(GTK_DIALOG(dialog));
    }
}

```

```

        gtk_widget_destroy(dialog);
    }

    // Clean up
    free(root_password);
}

void on_list_users(GtkWidget *widget, gpointer data) {
    (void)widget; // Unused parameter

    GtkTextBuffer *buffer = (GtkTextBuffer *)data;
    FILE *fp = popen("cut -d: -f1 /etc/passwd", "r");
    if (fp == NULL) {
        perror("popen");
        return;
    }

    char buffer_text[256];
    gtk_text_buffer_set_text(buffer, "", -1); // Clear existing text

    while (fgets(buffer_text, sizeof(buffer_text), fp) != NULL) {
        GtkTextIter end;
        gtk_text_buffer_get_end_iter(buffer, &end);
        gtk_text_buffer_insert(buffer, &end, buffer_text, -1);
    }
    pclose(fp);
}

int main(int argc, char *argv[]) {
    GtkWidget *window;
    GtkWidget *notebook;
    GtkWidget *grid;

```

```

GtkWidget *label;
GtkWidget *username_entry;
GtkWidget *password_entry;
GtkWidget *button;
GtkWidget *delete_entry;
GtkWidget *scrolled_window;
GtkWidget *text_view;
GtkTextBuffer *buffer;
GtkWidget *entries[2];

gtk_init(&argc, &argv);

// Main window setup
window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
gtk_window_set_title(GTK_WINDOW(window), "User Management");
gtk_window_set_default_size(GTK_WINDOW(window), 400, 300);
g_signal_connect(window, "destroy", G_CALLBACK(gtk_main_quit), NULL);

notebook = gtk_notebook_new();
gtk_container_add(GTK_CONTAINER(window), notebook);

// Create User Tab
grid = gtk_grid_new();
gtk_container_set_border_width(GTK_CONTAINER(grid), 10);
gtk_grid_set_row_spacing(GTK_GRID(grid), 5);
gtk_grid_set_column_spacing(GTK_GRID(grid), 5);

label = gtk_label_new("Username:");
gtk_grid_attach(GTK_GRID(grid), label, 0, 0, 1, 1);

username_entry = gtk_entry_new();
gtk_grid_attach(GTK_GRID(grid), username_entry, 1, 0, 1, 1);

```



```

label = gtk_label_new("Password:");
gtk_grid_attach(GTK_GRID(grid), label, 0, 1, 1, 1);

password_entry = gtk_entry_new();
gtk_entry_set_visibility(GTK_ENTRY(password_entry), FALSE);
gtk_grid_attach(GTK_GRID(grid), password_entry, 1, 1, 1, 1);

button = gtk_button_new_with_label("Create User");
gtk_grid_attach(GTK_GRID(grid), button, 0, 2, 2, 1);

entries[0] = username_entry;
entries[1] = password_entry;
g_signal_connect(button, "clicked", G_CALLBACK(on_create_user), entries);

gtk_notebook_append_page(GTK_NOTEBOOK(notebook), grid, gtk_label_new("Create User"));

// Delete User Tab
grid = gtk_grid_new();
gtk_container_set_border_width(GTK_CONTAINER(grid), 10);
gtk_grid_set_row_spacing(GTK_GRID(grid), 5);
gtk_grid_set_column_spacing(GTK_GRID(grid), 5);

label = gtk_label_new("Username:");
gtk_grid_attach(GTK_GRID(grid), label, 0, 0, 1, 1);

delete_entry = gtk_entry_new();
gtk_grid_attach(GTK_GRID(grid), delete_entry, 1, 0, 1, 1);

button = gtk_button_new_with_label("Delete User");
gtk_grid_attach(GTK_GRID(grid), button, 0, 1, 2, 1);

```

```

g_signal_connect(button, "clicked", G_CALLBACK(on_delete_user), delete_entry);

gtk_notebook_append_page(GTK_NOTEBOOK(notebook), grid, gtk_label_new("Delete User"));

// List Users Tab
grid = gtk_grid_new();
gtk_container_set_border_width(GTK_CONTAINER(grid), 10);
gtk_grid_set_row_spacing(GTK_GRID(grid), 5);
gtk_grid_set_column_spacing(GTK_GRID(grid), 5);

scrolled_window = gtk_scrolled_window_new(NULL, NULL);
gtk_widget_set_size_request(scrolled_window, 380, 250);
gtk_grid_attach(GTK_GRID(grid), scrolled_window, 0, 0, 2, 1);

text_view = gtk_text_view_new();
buffer = gtk_text_view_get_buffer(GTK_TEXT_VIEW(text_view));
gtk_container_add(GTK_CONTAINER(scrolled_window), text_view);

button = gtk_button_new_with_label("List Users");
gtk_grid_attach(GTK_GRID(grid), button, 0, 1, 2, 1);

g_signal_connect(button, "clicked", G_CALLBACK(on_list_users), buffer);

gtk_notebook_append_page(GTK_NOTEBOOK(notebook), grid, gtk_label_new("List Users"));

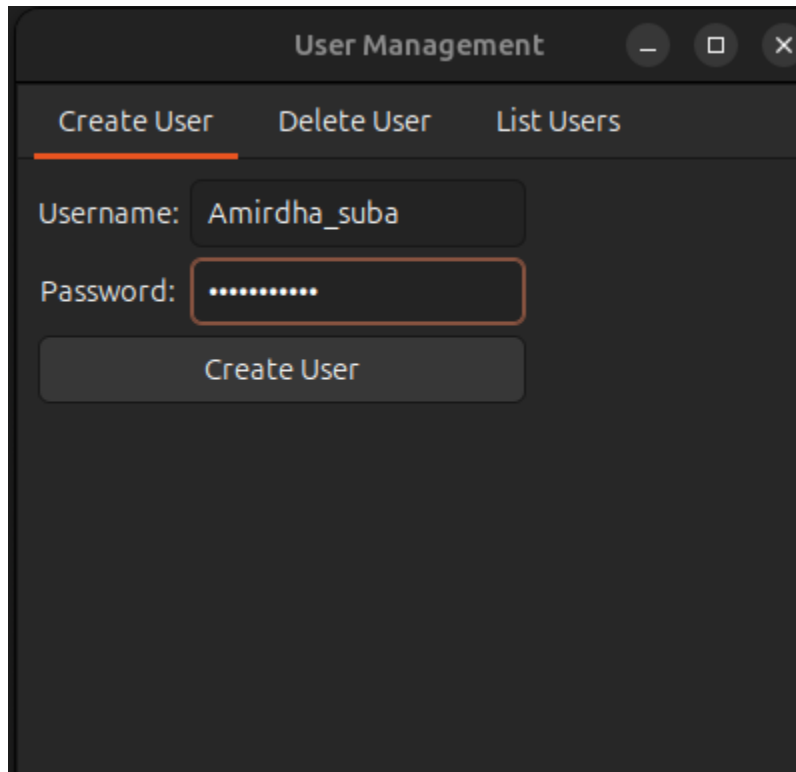
gtk_widget_show_all(window);

gtk_main();

return 0;
}

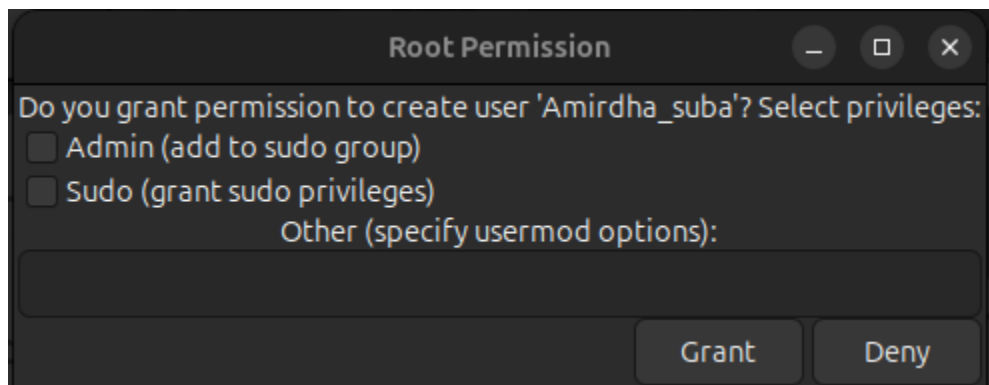
```

2.2 OUTPUT :



The image shows a window titled "User Management" with three tabs: "Create User", "Delete User", and "List Users". The "Create User" tab is selected and highlighted with an orange underline. Below the tabs, there are two input fields: "Username:" with the text "Amirdha_suba" and "Password:" with a masked password ".....". Below these fields is a button labeled "Create User".

Figure 2.2.1 : Creating User



The image shows a window titled "Root Permission" with a question: "Do you grant permission to create user 'Amirdha_suba'? Select privileges:". Below the question are three checkboxes: "Admin (add to sudo group)", "Sudo (grant sudo privileges)", and "Other (specify usermod options):". There is a text input field below the "Other" option. At the bottom right, there are two buttons: "Grant" and "Deny".

Figure 2.2.2 : Granting privilege

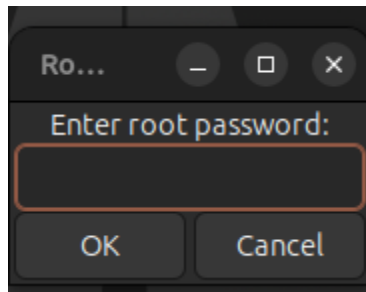


Figure 2.2.3 : Authentication from root user

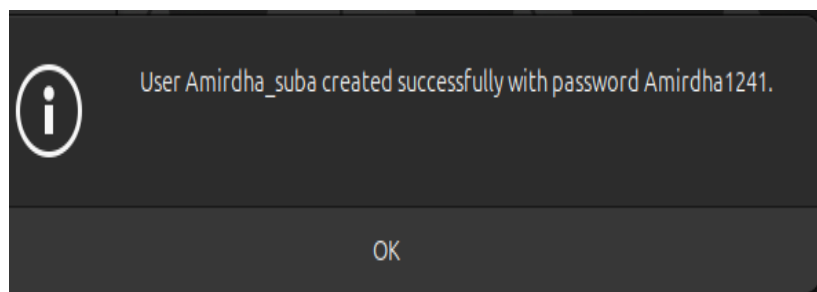


Figure 2.2.4 User Created

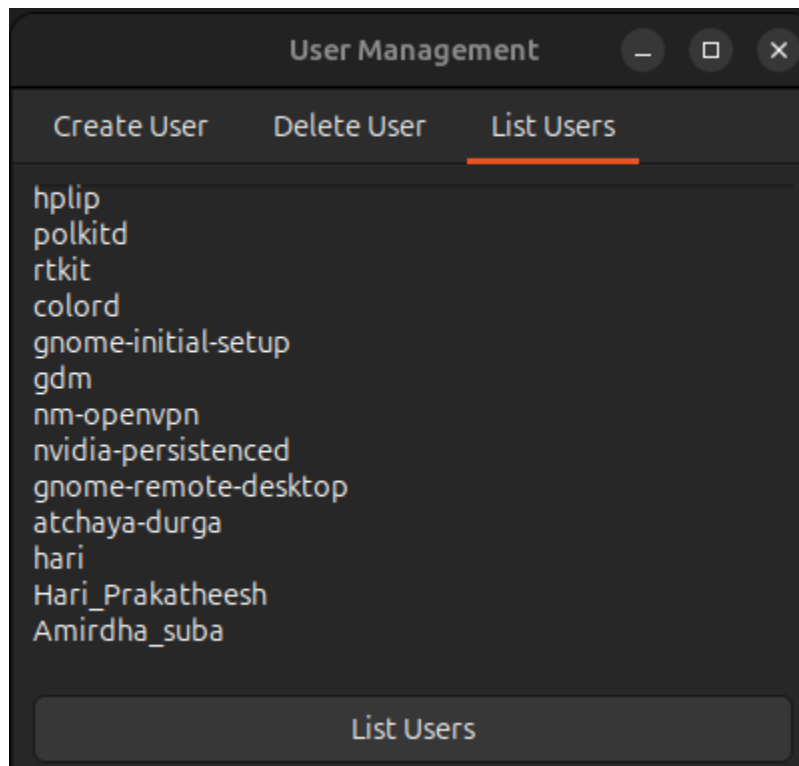


Figure 2.2.5 Listing User

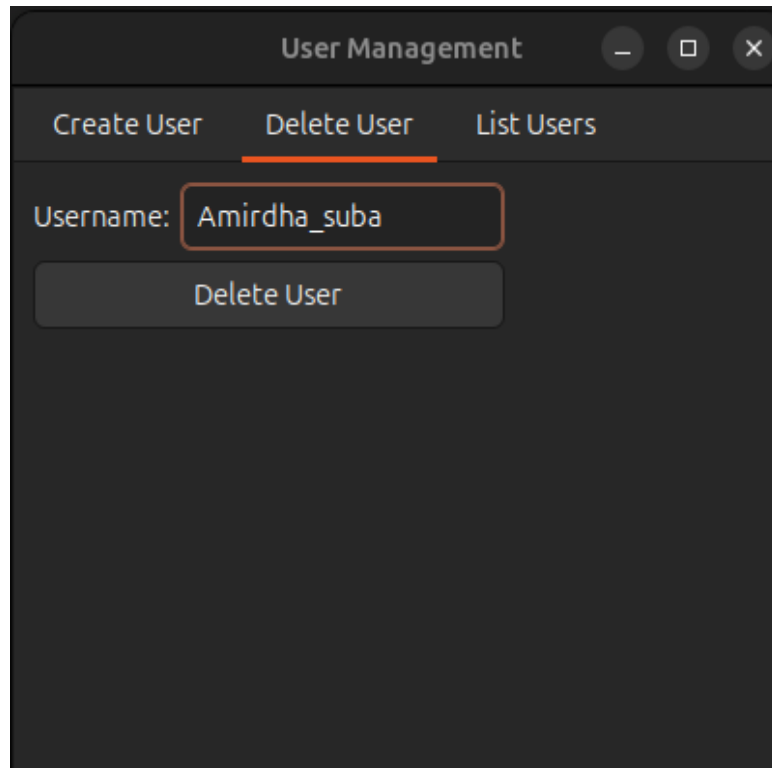


Figure 2.2.6 : Deleting Users

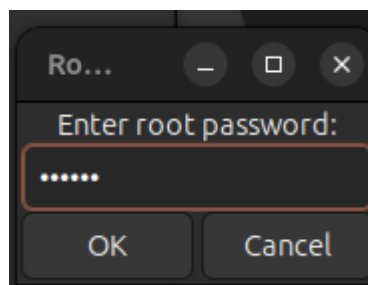


Figure 2.2.7 Authentication From Root to Delete

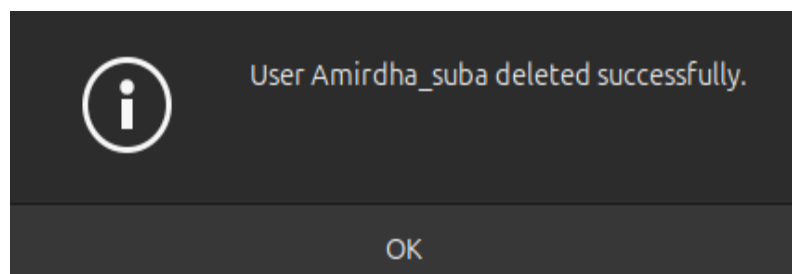


Figure 2.2.8 User Deleted

CHAPTER 3

CONCLUSION

The User Account Management System (UAMS) project offers a comprehensive solution for simplifying and securing user account administration on Linux systems. By providing a user-friendly graphical interface, robust authentication mechanisms, and efficient privilege management, the UAMS streamlines administrative tasks while ensuring the integrity and security of user data. The project's focus on clear feedback, seamless integration with existing system tools, and comprehensive documentation enhances usability and maintainability. With its ability to facilitate user account creation, deletion, and privilege assignment, the UAMS empowers administrators to effectively manage user accounts with confidence.

REFERENCES

1. Linux System Programming: Talking Directly to the Kernel and C Library by Robert Love
2. The Linux Programming Interface: A Linux and UNIX System Programming Handbook by Michael Kerrisk
3. GTK Documentation: <https://www.gtk.org/docs/>
4. Linux useradd Man Page: <https://man7.org/linux/man-pages/man8/useradd.8.html>
5. Linux sudo Man Page: <https://man7.org/linux/man-pages/man8/sudo.8.html>
6. GCC Documentation: <https://gcc.gnu.org/onlinedocs/>