PROCESS MONITOR

MINOR PROJECT REPORT

By

MOHAMED IRSATH (RA2211056010055)
JENIL JAIN (RA2211056010070)

Under the guidance of

DR. SHRI BHARATHI SV

In partial fulfilment for the Course

of

21CSC202J – OPERATING SYSTEMS

in **Data Science and Business Systems**

FACULTY OF ENGINEERING AND TECHNOLOGY

SCHOOL OF COMPUTING

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR

NOVEMBER 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this minor project report for the course **21CSC202J OPERATING SYSTEMS** entitled in **" PROCESS MONITOR "** is the bonafide work **of Mohamed Irsath (RA2211056010055) and Jenil Jain (RA2211056010070)** who carried out the work under my supervision.

DR. SHRI BHARATHI SV
SUPERVISOR
Assistant Professor
Department of Data Science and Business Systems

R. M. LAKSHMI
HEAD OF THE DEPARTMENT
Department of Data Science and Business Systems

Department of Data Science and Business Systems

SRM Institute of Science and Technology

Own Work Declaration Form

Degree/ Course **:** B.Tech in Computer Science and Engineering (Data Science)

Student Names **:** Mohamed Irsath, Jenil Jain

Registration Number: RA2211056010055, RA2211056010070

Title of Work **:** PROCESS MONITOR

We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct, as listed inthe University Website, Regulations, and the Education Committee guidelines.

We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc.)
- Given the sources of all pictures, data etc. that are not my own

- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:	
I am aware of and understand the University's policy on Academic misconduct and I certify that this assessment is our own work, except where indicated by referring, and that I have followed the good academic practices noted above.	
Mohamed Irsath - RA2211056010055	Jenil Jain - RA2211056010070

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support. We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr. T.V.Gopal**, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We are incredibly grateful to our Head of the Department, **Dr. M. Lakshmi,** Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology for her suggestions and encouragement at all the stages of the project work.

Our inexpressible respect and thanks to our guide, **Dr. Shri Bharathi SV**, Assistant Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology for providing us with an opportunity to pursue our project under her mentorship. She provided us with the freedom and support to explore the research topics of our interest. Her passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank the Data Science and Business Systems Department staff and students, SRM Institute of Science and Technology, for their help during our project. Finally, we would like to thank parents, family members, and friends for their unconditional love, constant support, and encouragement.

Mohamed Irsath [RA2211056010055]

Jenil Jain [RA2211056010070]

TABLE OF CONTENTS

ABSTRACT

The "Process Monitor" is a robust yet straightforward process management tool designed to facilitate the monitoring and control of running processes within a Linux-based operating system. This project utilizes a Binary Search Tree (BST) data structure to efficiently organize and manage process information, including Process IDs (PIDs), names, and statuses. The primary objective of the "Process Monitor" is to offer users an intuitive interface for managing processes effectively. Upon execution, the tool accesses the /proc directory to gather process details, constructing a BST data structure to organize and facilitate swift retrieval and manipulation of process information. The functionality of the "Process Monitor" includes process creation, deletion, display, and search capabilities. Users can create new processes by providing the desired command to execute, delete specific processes by their respective PIDs, display a comprehensive list of active processes, and search for processes by their names. The tool incorporates a user authentication system to ensure secure access, prompting users to enter predefined username and password credentials before allowing utilization of its functionalities. Implemented with careful consideration for efficiency and usability, the "Process Monitor" offers a menu-driven interface, simplifying the user experience and enabling seamless navigation through its features. While providing a convenient means for process management, it is important to exercise caution while utilizing the tool, especially when terminating processes, as abrupt termination may impact system stability.

In summary, the "Process Monitor" offers a user-friendly interface, leveraging a BST data structure to provide efficient process management functionalities, enhancing the user's ability to oversee and control processes within the Linux environment.

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

PID	Process ID
BST	Binary Search Tree
CPU	Central Processing Unit
OS	Operating Systems
DT_DIR	Directory type
SIGKILL	Signall that causes immediate termination of a process
GUI	Graphical User Interface
DFS	Depth First Search
DOI	Digital Object Killer
IEEE	Institute of Electrical and Electronics Engineering
ASCII	American Standard Code for Information Interchange

CHAPTER 1

INTRODUCTION

1.1 General

The "Process Monitor" project stands as an ambitious and meticulously crafted software initiative tailored explicitly for Linux-based operating systems. In the intricate ecosystem of Linux environments, the management and oversight of processes serve as pivotal components of system administration. Processes, acting as the fundamental units of execution, operate autonomously, exerting substantial influence on system performance and resource allocation. The "Process Monitor" project endeavors to provide an innovative and robust software solution that simplifies, streamlines, and optimizes the intricate process management landscape within Linux environments.

Within the dynamic realm of Linux-based systems, managing processes efficaciously remains a critical endeavor. The "Process Monitor" project emerges against the backdrop of the escalating complexity of process management tasks, ranging from overseeing resource utilization to ensuring system stability. Addressing these challenges requires a comprehensive tool that harmoniously amalgamates efficient process management capabilities, intuitive user interfaces, and robust security measures, all tailored explicitly for the Linux ecosystem.

1.2 Purpose

The purpose of the "Process Monitor" project is multi-faceted, aiming to address several critical objectives within the context of process management and system administration:

- Efficient Process Management: The primary purpose is to create a tool that facilitates efficient management and monitoring of running processes within a Unix-based operating system. By leveraging a Binary Search Tree (BST) data structure, the tool organizes process information systematically, enabling quick retrieval and manipulation of process details.

- User-Friendly Interface: To provide a user-friendly interface that simplifies the process management tasks for users. Through a menu-driven system, users can perform essential actions such as creating new processes, terminating existing ones, displaying comprehensive process lists, and searching for specific processes based on their names.

- Enhanced System Stability and Performance: By providing a means to oversee and control processes effectively, the "Process Monitor" contributes to system stability and performance. Users can identify and manage processes efficiently, preventing issues related to resource utilization, conflicts, or undesired processes.

- Security Measures: Implementation of user authentication ensures secure access to the process manager functionalities. This helps maintain system integrity and prevents unauthorized access, ensuring that only authorized users can manage processes within the system.

- Educational and Learning Purposes: The project serves as a learning opportunity for developers, allowing them to understand and implement data structures like Binary Search Trees in a practical scenario. It also offers insights into system-level programming, process management, and user interface design.

- Contribution to System Administration: It aims to contribute to the toolkit available for system administrators, enabling them to have an additional resource for managing processes effectively.

1.3 Scope

The scope of the "Process Monitor" project spans across various pivotal domains within Linux system administration and process management. Its primary aim lies in orchestrating, monitoring, and governing processes within the system landscape. Core functionalities revolve around the comprehensive handling of processes, encompassing their inception, termination, and precise attribute-based querying. Leveraging directory traversal techniques within the /proc directory, this initiative adeptly mines essential process-centric information, enabling nuanced and informed process management. Employing the Binary Search Tree (BST) data structure, the project engenders a streamlined organization of process data, fostering expeditious access, incorporation, expulsion, and categorization of processes based on their unique Process IDs (PIDs). Furthermore, the inclusion of rudimentary user authentication fortifies the tool's security framework, ensuring authorized access to system functions. In anticipation of future advancements, the envisioned trajectory integrates real-time monitoring capabilities, augmented process metrics, interface refinements, and potential harmonization with established system monitoring tools. This holistic outlook mirrors an adaptive evolution, aligning with the ever-evolving exigencies of Linux system administration paradigms. Overall, the scope of the "Process Monitor" encompasses a comprehensive suite of functionalities geared towards effective process management, system monitoring, and potential expansions to cater to evolving system administration requirements in Linux environments.

CHAPTER 2

LITERATURE REVIEW

In this section, we discuss the previous research that has been conducted in an effort to create a Process Monitor. The information present in this section includes Process creation , Process Termination , Process Searching and Saving Processes details in a Binary Search Tree are used in our study.

2.1. Process Management

- Research Paper on Operating System | ISSN: 2320-2882
  Authors: Anjalee Sahu (1st), Asst. Prof. Shrikant Singh (2nd), HOD Rahul Chawda (3rd)
  Department of Computer Science, Kalinga University, Naya Raipur, Raipur Chhattisgarh 492101, India

In this paper we learned about how OS works and how it act as a Intermediate between a user and Hardware. It also states that the operating system handles system resources such as the computer's memory and sharing of the central processing unit (CPU) time by various applications or peripheral devices. Programs and input methods are constantly competing for the attention of the CPU and demand memory, storage and input/output bandwidth. The operating system ensures that each application gets the necessary resources it needs in order to maximize the functionality of the overall system.

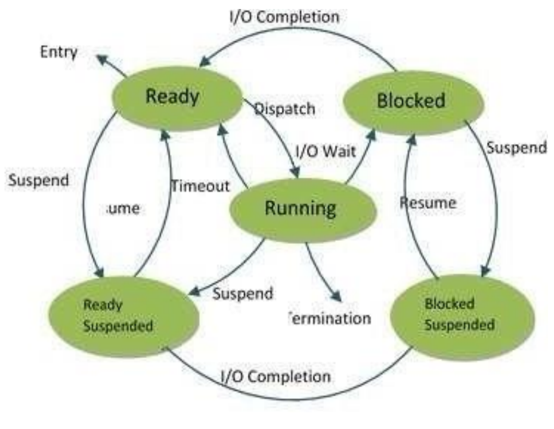


Fig 2.1.1

- Operating System Research Paper in Course: Machine Learning (INT 246)
  Research Paper No: 11913362
  Lovely Professional University

In this paper we learned the features of Operating System and the jobs that Operating System manages like Device Management which Allocates and deallocates resources and determines who gets them. Job Accounting which Tracks time and resources used by various jobs or users. Memory Management Tracks primary memory.Who uses which parts, which parts do not, etc. It also allocates memory when a process or program requests it. Processor Management Allocate a processor to a process and deallocate it when it is no longer needed or when a task is completed. System Performance Control: Records service requests and delays from the system. It also looks for passwords or some other protection technique to prevent unauthorized access to programs and data.

2.2 Data Structure:

- A Study on the Usage of Data Structures in Information Retrieval
  Authors: V. R. Kanagavalli (Associate Professor), G. Maheeja (PG Student)
  Department of Computer Applications, Sri Sai Ram Engineering College

In this Paper we learned ablout many Data Structures but we implemented Binary Search Tree because of it’s Efficient Searching and Retrieval:A tree is a data structure has a data item as its root and the subtrees are generated with a node as a parent node. In a search tree generally the solution is found as leaf. There are various kinds of trees depending upon the way of arrangement and the way of traversal of the tree.

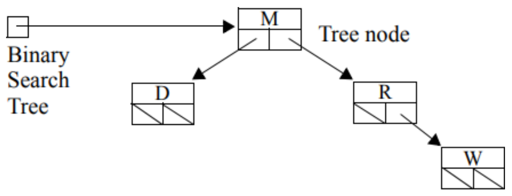


Fig 2.2.1

- Thinking about binary trees in an object-oriented world
  Authors: A. Michael Berman from Rowan College of New Jersey and Robert C. Duvall from Brown University."

In this paper we learned how a Binary Search Tree can be used in this PROCESS MONITOR more efficiently This paper provides ways of programming the BST that more nearly implement the definition directly into the code, and eliminate the problems with the “traditional” implementation. We believe that by creating code that closely mimics our abstract notions, we can reduce that conceptual gap between the abstract and the concrete, making it easier for our students to understand the implementation

2.3 Monitoring and Security:

- A Continuous Improvement and Monitoring Performance System: Monitor - Analysis Vol. 2011
  Article ID: 917557, 15 pages
  DOI: 10.5171/2011.917557
  Authors: Vito Romaniello, Paolo Renna, and Vincenzo Cinque

In this paper we learned that A monitoring system is necessary to measure the achievement of targets, and the deviation from the objectives; The instrument will be more effective and accurate if it can track the individual contributors that affect directly and indirectly objectives. The difficulties of setting up a system of performance measurement are not few. For example, an initial problem is to determine the period of controlling and planning.

- Recent Trends in User Authentication - A Survey
  Digital Object Identifier: 10.1109/ACCESS.2019.DOI
  Authors: Syed W. Shah and Salil S. Kanhere (Senior Member, IEEE)
  School of Computer Science and Engineering, The University of New South Wales, Australia

In this paper we learned that user authentication remains critically important in today's digital world. With the increasing reliance on digital systems, networks, and online services, ensuring secure access to information and protecting sensitive data has become a paramount concern across various industries and everyday activities

CHAPTER 3

PROPOSED METHODOLOGY

3.1. Process Enumeration:

- **Directory Traversal**: The code begins by opening and reading the contents of the /proc directory using the opendir and readdir functions provided by the <dirent.h> library. It iterates through each entry in the /proc directory using the readdir function to access information about individual processes.

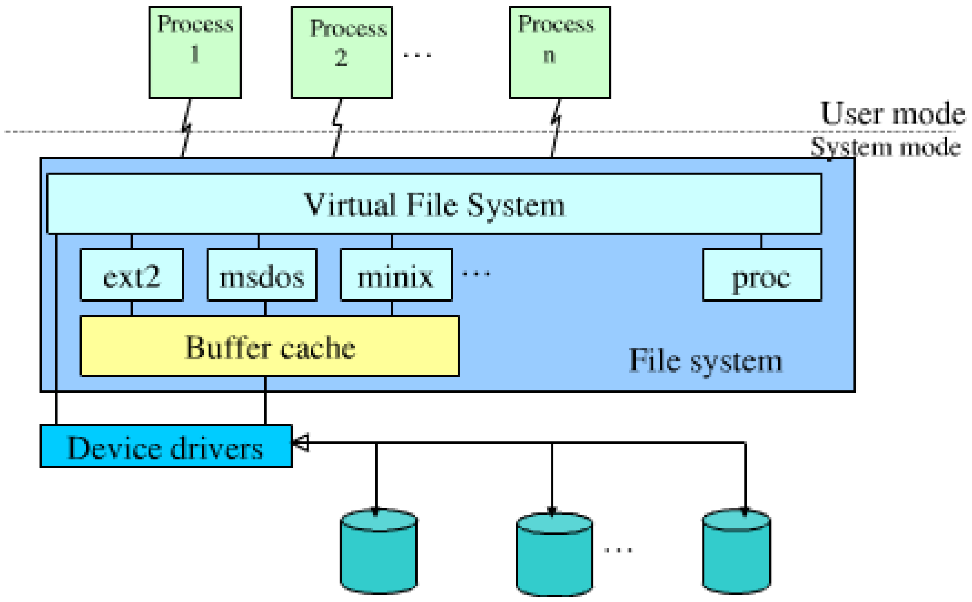


Fig 3.1.1

- **Process Information Retrieval**:

  For each entry in the /proc directory, the code checks if it represents a process by verifying its type (DT_DIR) using the d_type attribute. If the entry corresponds to a process (identified by a numeric directory name), the code constructs a file path to access specific information about that process, such as its status. The atoi() function converts the directory name (which represents the process ID - PID) to an integer for further processing.

It constructs a path to the /proc/{pid}/status file, where crucial process information is stored, and opens this file using fopen. Within this file, the code reads different lines containing process details, such as the process name and status, using fgets. Information like the process name and status is extracted from these lines by searching for specific keywords (e.g., "Name:", "State:") and extracting the corresponding data. Extracted process details, such as the PID, name, and status, are then utilized to create nodes in the Binary Search Tree (BST) data structure, allowing for organized storage and management of this information.

3.2 Process creation, deletion and searching:

- **Process creation:**

  In this project,create_process function utilizes fork() to spawn a new child process. After calling fork(), the child process gets a copy of the parent process's memory, file descriptors, and execution context. The child process then executes the specified command or program using execlp() to replace its own process image with the specified command. The execlp() function replaces the current process's image with a new one specified by the command provided, effectively starting a new program or process.

- **Process deletion:**

  In this project, the kill_process function is responsible for terminating a process by sending the SIGKILL signal to the specified PID obtained from the BST. SIGKILL is a signal that causes immediate termination of the target process without allowing it to perform any cleanup operations. It forcefully terminates the process. This termination process ensures that the identified process is forcefully killed, removing it from the list of running processes.

- **Process searching:**

  The search_process_by_name_in_bst function performs a depth-first traversal of the Binary Search Tree (BST) data structure populated with process information. This function searches for processes whose names match a given search query or substring provided by the user. It follows a recursive approach to traverse the BST. It first traverses the left subtree, then checks if the name of the current node's process matches the provided search query. If a match is found, it prints details about the process (such as PID, name, and status) to the console. Finally, it traverses the right subtree, continuing the search process.

3.3 Binary Search Tree (BST) Implementation:

The Binary Search Tree (BST) implemented in the "Process Monitor" project serves as a data structure to organize and store information about running processes retrieved from the /proc directory in a Linux-based system. Here's an overview of the BST implementation in the provided code

- **Purpose of BST:**
  I. The BST is used to efficiently organize process information (such as PID, process name, and status) obtained during directory traversal and process information retrieval.
  II. It allows for quick insertion, deletion, and searching of processes based on their PIDs, facilitating efficient process management within the "Process Monitor" tool.

- **Structure of BST Node:**
  Each node in the BST represents an individual process and contains the following attributes:
  pid: Represents the Process ID (PID) of the process.
  name: Stores the name of the process.
  status: Holds the status of the process (e.g., "Running," "Sleeping").
  left and right: Pointers to the left and right child nodes, respectively.

- **BST's Advantages:**
  I. The BST provides an efficient way to organize process information, enabling quick access and manipulation of processes based on their PIDs.
  II. It allows for sorted storage and retrieval of process details, facilitating effective process management operations.

- **Limitations:**
  I. While BSTs offer efficient search and insertion operations, their performance can degrade if the tree becomes unbalanced, leading to suboptimal time complexities in certain scenarios.
  II. Overall, the BST implementation in the "Process Monitor" project offers an organized and efficient means to manage and access process-related data within the tool.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <string.h>
#include <signal.h>
#include <sys/types.h>
#include <unistd.h>

#define USERNAME "Mohamed"
#define PASSWORD "@Irsath06"

struct ProcessNode {
    int pid;
    char name[256];
    char status[256];
    struct ProcessNode* left;
    struct ProcessNode* right;
};

void add_process_to_bst(struct ProcessNode** root, int pid, const char* name, const char* status) {
    if (*root == NULL) {
        struct ProcessNode* newProcess = (struct ProcessNode*)malloc(sizeof(struct ProcessNode));
        newProcess->pid = pid;
        strncpy(newProcess->name, name, sizeof(newProcess->name) - 1);
        newProcess->name[sizeof(newProcess->name) - 1] = '\0';
        strncpy(newProcess->status, status, sizeof(newProcess->status) - 1);
        newProcess->status[sizeof(newProcess->status) - 1] = '\0';
        newProcess->left = newProcess->right = NULL;
        *root = newProcess;
    } else {
        if (pid < (*root)->pid) {
            add_process_to_bst(&(*root)->left, pid, name, status);
        }
 else {
            add_process_to_bst(&(*root)->right, pid, name, status);
        }
    }
}

void search_process_by_name_in_bst(struct ProcessNode* root, const char* targetName) {
    if (root != NULL) {
        search_process_by_name_in_bst(root->left, targetName);
        if (strstr(root->name, targetName) != NULL) {
            printf("PID: %d, Name: %s, Status: %s\n", root->pid, root->name, root->status);
        }
        search_process_by_name_in_bst(root->right, targetName);
    }
}

void kill_process_in_bst(struct ProcessNode** root, int dpid) {
    if (*root == NULL) {
        return;
    }

    if (dpid < (*root)->pid) {
        kill_process_in_bst(&(*root)->left, dpid);
    } else if (dpid > (*root)->pid) {
        kill_process_in_bst(&(*root)->right, dpid);
    } else {
        struct ProcessNode* temp = *root;
        if ((*root)->left == NULL) {
            *root = (*root)->right;
        } else if ((*root)->right == NULL) {
            *root = (*root)->left;
        } else {
            struct ProcessNode* minRightSubtree = (*root)->right;
 while (minRightSubtree->left != NULL) {
                minRightSubtree = minRightSubtree->left;
            }
            (*root)->pid = minRightSubtree->pid;
            strcpy((*root)->name, minRightSubtree->name);
            strcpy((*root)->status, minRightSubtree->status);
            kill_process_in_bst(&(*root)->right, minRightSubtree->pid);
        }
        free(temp);
    }
}

void print_processes_in_bst(struct ProcessNode* root) {
    if (root != NULL) {
        print_processes_in_bst(root->left);
        printf("PID: %d, Name: %s, Status: %s\n", root->pid, root->name, root->status);
        print_processes_in_bst(root->right);
    }
}

int create_process(char* cmd) {
    pid_t pid = fork();
    if (pid == 0) {
        if (cmd != NULL) {
            execlp(cmd, cmd, NULL);
            exit(0);
        }
    } else if (pid > 0) {
        printf("Process %d created successfully.\n", pid);
        return pid;
    } else {
        printf("Error creating process.\n");
    }
}
```

```
int authenticate_user() {
    char username[256];
    char password[256];
    printf("Enter username: ");
    scanf("%255s", username);
    printf("Enter password: ");
    scanf("%255s", password);
    if (strcmp(username, USERNAME) == 0 && strcmp(password, PASSWORD) == 0) {
        printf("Authentication successful!\n");
        return 1;
    } else {
        printf("Authentication failed. Access denied.\n");
        return 0;
    }
}

void free_bst(struct ProcessNode* root) {
    if (root != NULL) {
        free_bst(root->left);
        free_bst(root->right);
        free(root);
    }
}

int main() {
    DIR* dir;
    struct dirent* entry;
    struct ProcessNode* bstRoot = NULL;
    char cmd[256];
    int dpid, newpid;

    dir = opendir("/proc");
    if (dir == NULL) {
```

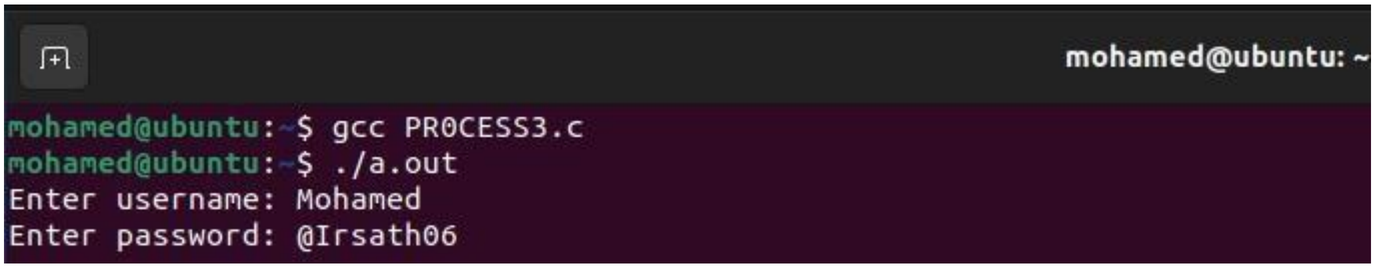
```
perror("opendir");
    exit(1);
  }

  while ((entry = readdir(dir)) != NULL) {
    if (entry->d_type == DT_DIR) {
        char path[300];
        int pid = atoi(entry->d_name);
        if (pid != 0) {
            snprintf(path, sizeof(path), "/proc/%s/status", entry->d_name);
            FILE* file = fopen(path, "r");
            if (file) {
                char line[256];
                char* name = NULL;
                char* status = NULL;
                while (fgets(line, sizeof(line), file)) {
                    if (strstr(line, "Name:")) {
                        name = strdup(line + 6);
                    } else if (strstr(line, "State:")) {
                        status = strdup(line + 7);
                    }
                }
                if (name != NULL && status != NULL) {
                    add_process_to_bst(&bstRoot, pid, name, status);
                }
                free(name);
                free(status);
                fclose(file);
            }
        }
    }
  }
  closedir(dir);
```

```
if (authenticate_user()) {
        while (1) {
            printf("\nMenu:\n");
            printf("1. Create a new process (c)\n");
            printf("2. Delete a process (d)\n");
            printf("3. Display the list of processes (p)\n");
            printf("4. Search a Process (s)\n");
            printf("5. Quit (q)\n");
            printf("Enter your choice:");

            char choice;
            scanf(" %c", &choice);

            switch (choice) {
                case 'c':
                    printf("Enter the command to run: ");
                    scanf("%255s", cmd);
                    newpid = create_process(cmd);
                    add_process_to_bst(&bstRoot, newpid, cmd, "Running");
                    break;
                case 'd':
                    printf("Enter the PID of the process to delete: ");
                    scanf("%d", &dpid);
                    kill_process_in_bst(&bstRoot, dpid);
                    break;
                case 'p':
                    print_processes_in_bst(bstRoot);
                    break;
                case 's':
                    printf("Enter the name of the process to search: ");
                    char targetName[256];
                    scanf("%255s", targetName);
                    search_process_by_name_in_bst(bstRoot, targetName);
                    break;
                case 'q':
                    free_bst(bstRoot);
                    return 0;
                default:
                    printf("Invalid choice. Please enter 'c'/'d'/'p'/'s'/'q'\n");
            }
        }
    }

    free_bst(bstRoot);
    return 0;
}
```

CHAPTER 4

RESULTS

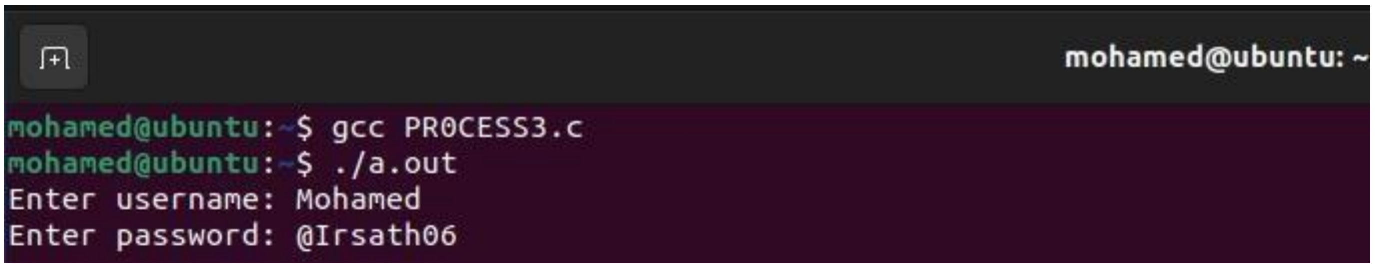
User Authentication:



```
mohamed@ubuntu: ~
mohamed@ubuntu:~$ gcc PR0CESS3.c
mohamed@ubuntu:~$ ./a.out
Enter username: Mohamed
Enter password: @Irsath06
```

Fig4.1.1

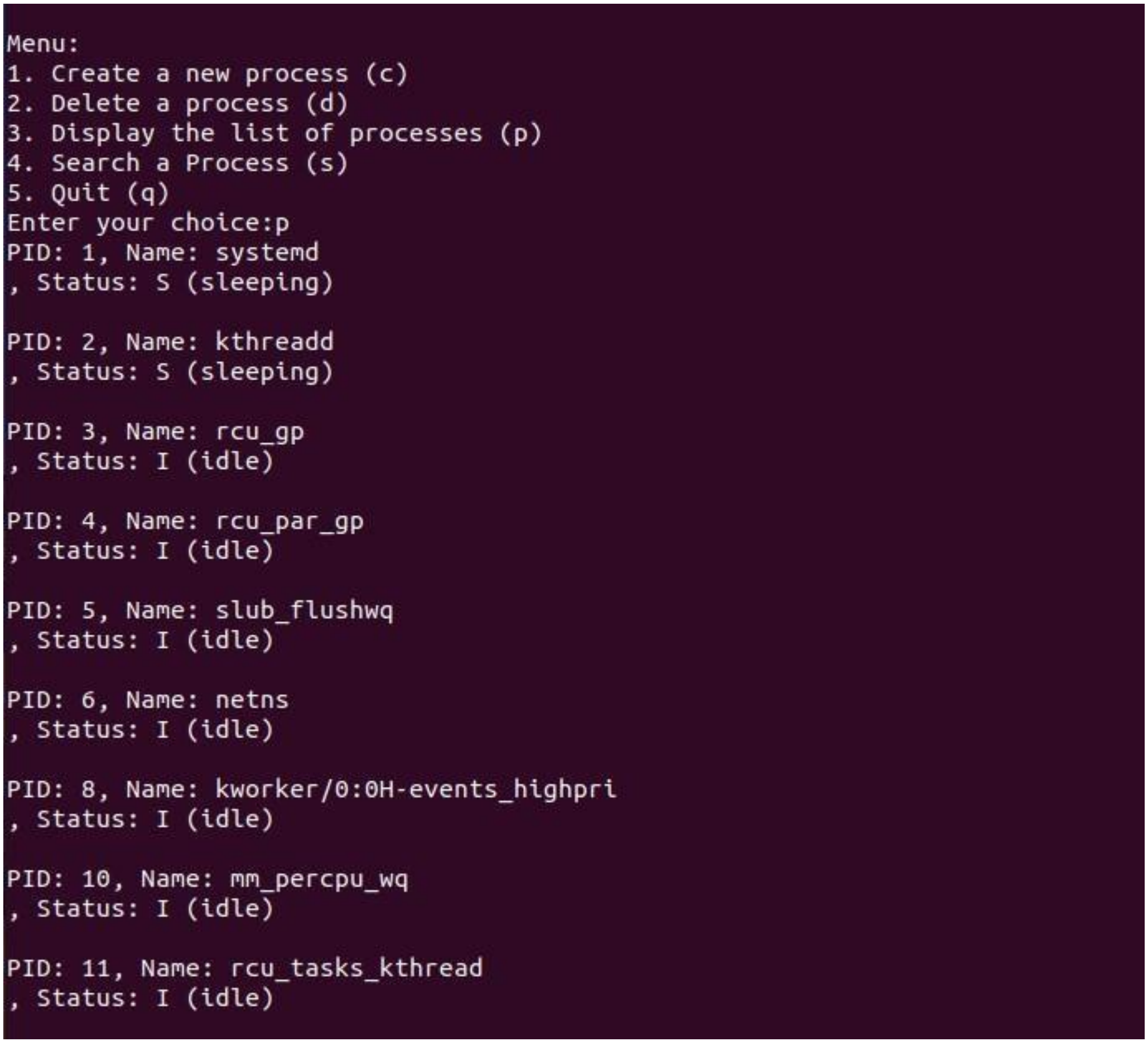
Main Module:



```
mohamed@ubuntu: ~
mohamed@ubuntu:~$ gcc PR0CESS3.c
mohamed@ubuntu:~$ ./a.out
Enter username: Mohamed
Enter password: @Irsath06
```

Fig4.2.1

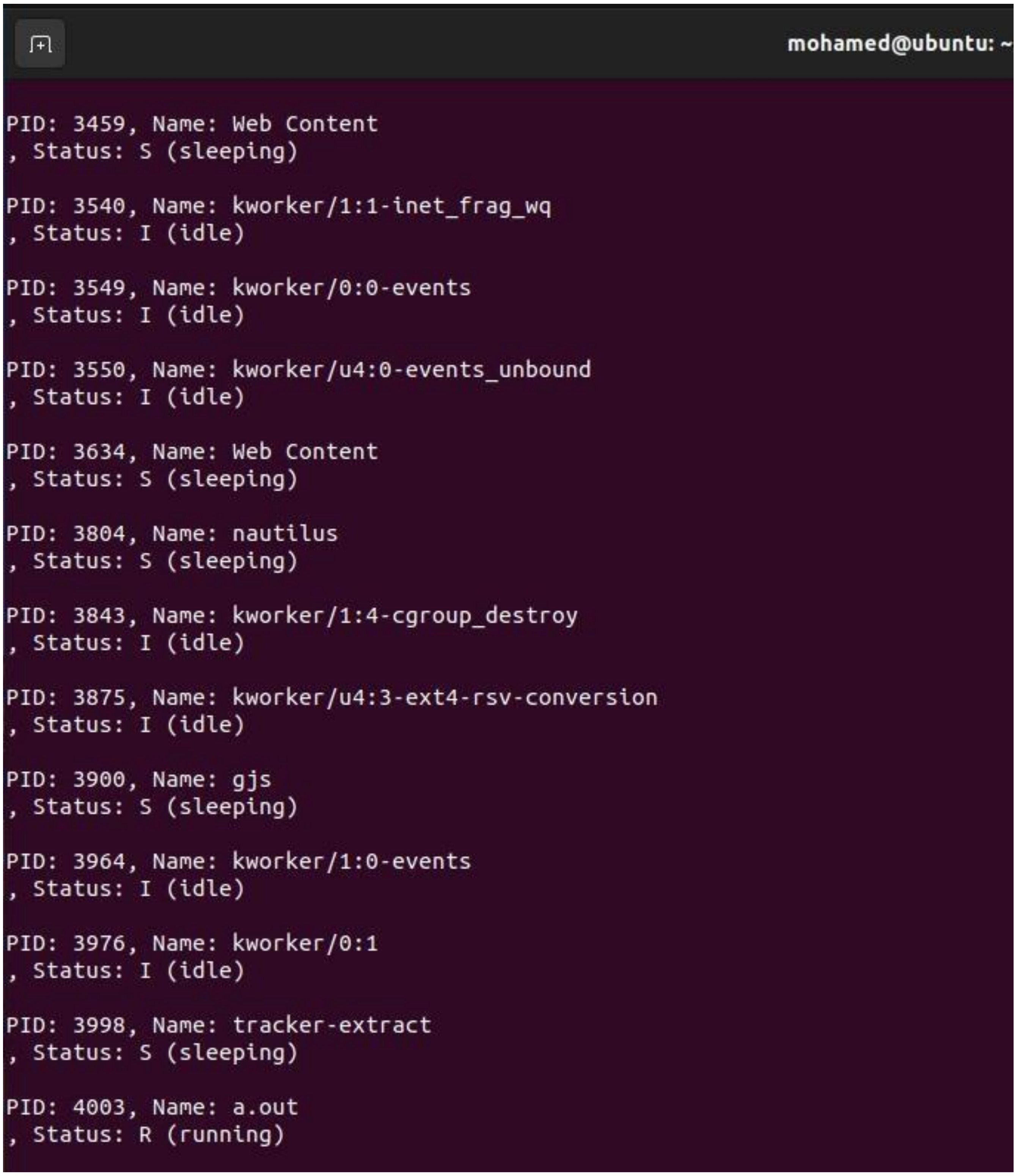
Display Processes:



```
Menu:
1. Create a new process (c)
2. Delete a process (d)
3. Display the list of processes (p)
4. Search a Process (s)
5. Quit (q)
Enter your choice:p
PID: 1, Name: systemd
, Status: S (sleeping)

PID: 2, Name: kthreadd
, Status: S (sleeping)

PID: 3, Name: rcu_gp
, Status: I (idle)

PID: 4, Name: rcu_par_gp
, Status: I (idle)

PID: 5, Name: slub_flushwq
, Status: I (idle)

PID: 6, Name: netns
, Status: I (idle)

PID: 8, Name: kworker/0:0H-events_highpri
, Status: I (idle)

PID: 10, Name: mm_percpu_wq
, Status: I (idle)

PID: 11, Name: rcu_tasks_kthread
, Status: I (idle)
```

Fig 4.3.1

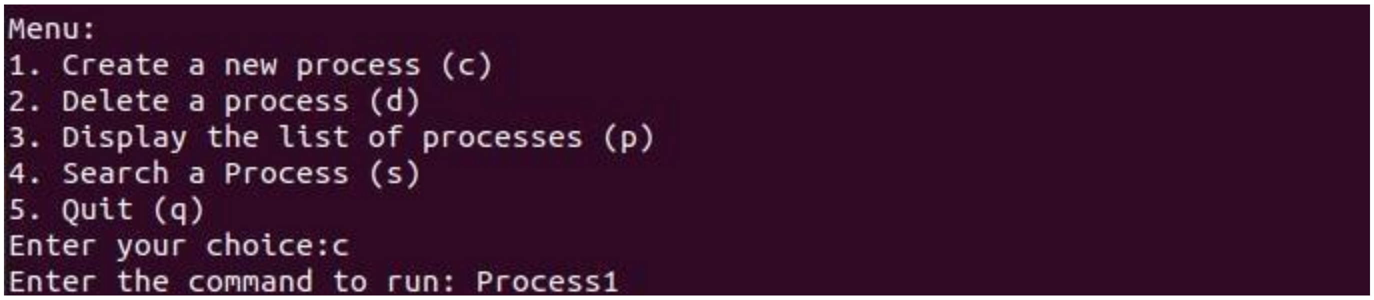
(Process between PID 12 to 3458)


mohamed@ubuntu: ~

```
PID: 3459, Name: Web Content
, Status: S (sleeping)

PID: 3540, Name: kworker/1:1-inet_frag_wq
, Status: I (idle)

PID: 3549, Name: kworker/0:0-events
, Status: I (idle)

PID: 3550, Name: kworker/u4:0-events_unbound
, Status: I (idle)

PID: 3634, Name: Web Content
, Status: S (sleeping)

PID: 3804, Name: nautilus
, Status: S (sleeping)

PID: 3843, Name: kworker/1:4-cgroup_destroy
, Status: I (idle)

PID: 3875, Name: kworker/u4:3-ext4-rsv-conversion
, Status: I (idle)

PID: 3900, Name: gjs
, Status: S (sleeping)

PID: 3964, Name: kworker/1:0-events
, Status: I (idle)

PID: 3976, Name: kworker/0:1
, Status: I (idle)

PID: 3998, Name: tracker-extract
, Status: S (sleeping)

PID: 4003, Name: a.out
, Status: R (running)
```

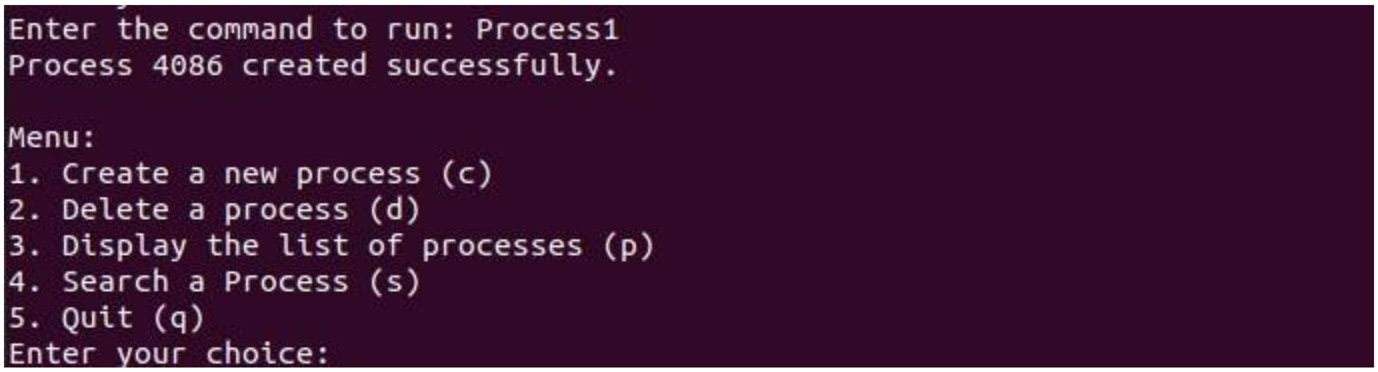
Fig 4.3.2

Create Process



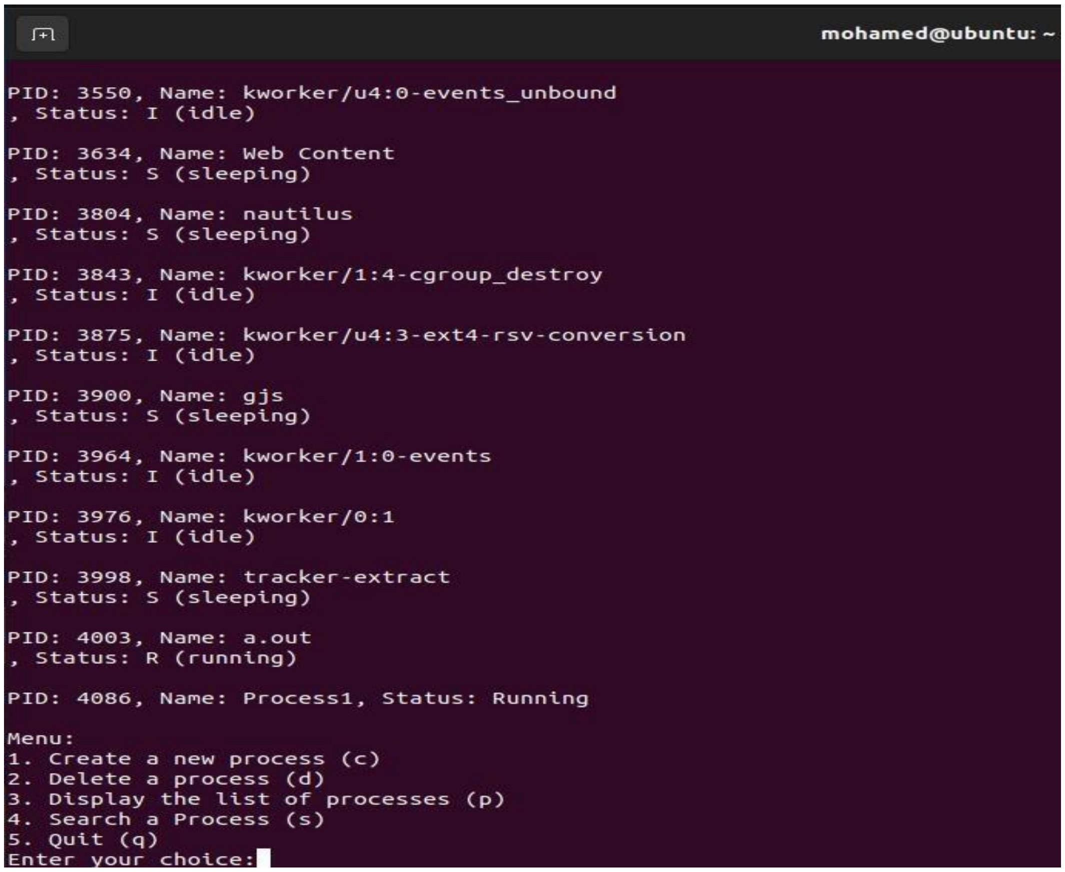
```
Menu:
1. Create a new process (c)
2. Delete a process (d)
3. Display the list of processes (p)
4. Search a Process (s)
5. Quit (q)
Enter your choice:c
Enter the command to run: Process1
```

Fig 4.4.1



```
Enter the command to run: Process1
Process 4086 created successfully.

Menu:
1. Create a new process (c)
2. Delete a process (d)
3. Display the list of processes (p)
4. Search a Process (s)
5. Quit (q)
Enter your choice:
```

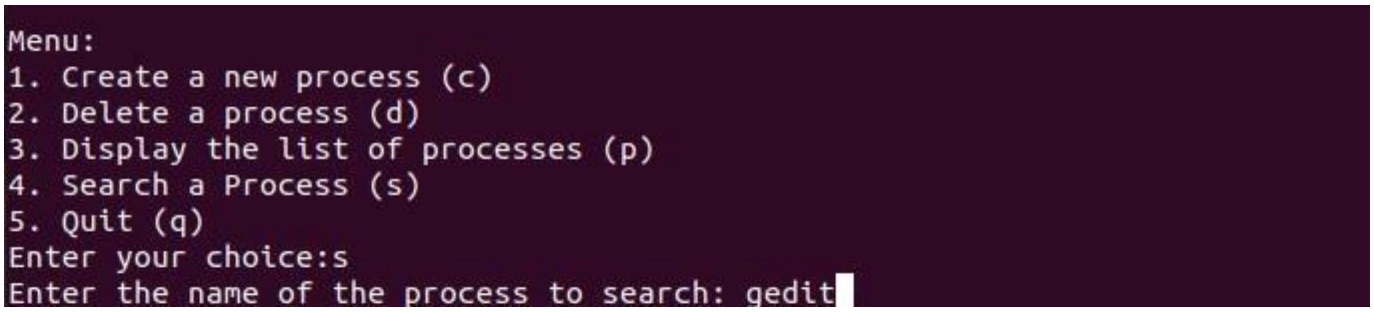
Fig 4.4.2



```
mohamed@ubuntu: ~

PID: 3550, Name: kworker/u4:0-events_unbound
, Status: I (idle)

PID: 3634, Name: Web Content
, Status: S (sleeping)

PID: 3804, Name: nautilus
, Status: S (sleeping)

PID: 3843, Name: kworker/1:4-cgroup_destroy
, Status: I (idle)

PID: 3875, Name: kworker/u4:3-ext4-rsv-conversion
, Status: I (idle)

PID: 3900, Name: gjs
, Status: S (sleeping)

PID: 3964, Name: kworker/1:0-events
, Status: I (idle)

PID: 3976, Name: kworker/0:1
, Status: I (idle)

PID: 3998, Name: tracker-extract
, Status: S (sleeping)

PID: 4003, Name: a.out
, Status: R (running)

PID: 4086, Name: Process1, Status: Running

Menu:
1. Create a new process (c)
2. Delete a process (d)
3. Display the list of processes (p)
4. Search a Process (s)
5. Quit (q)
Enter your choice:
```

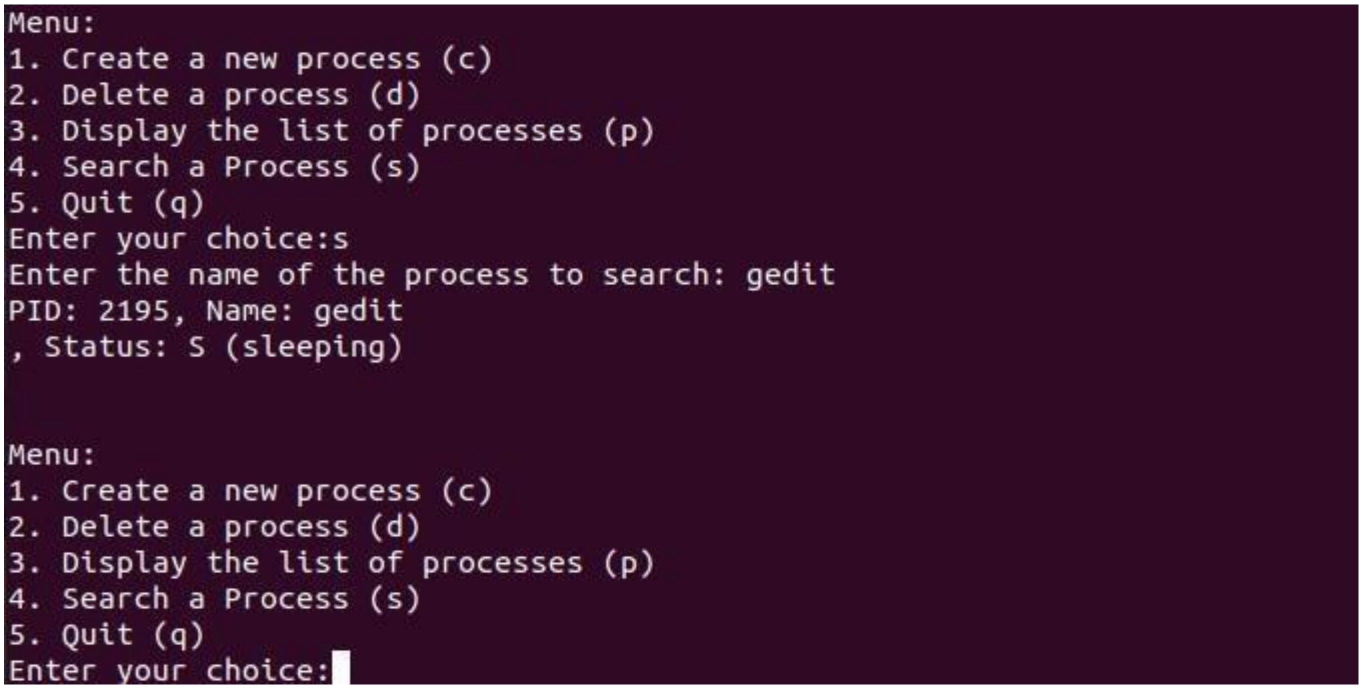
Fig 4.4.3**(Displaying after creation)**

Search a process:



```
Menu:
1. Create a new process (c)
2. Delete a process (d)
3. Display the list of processes (p)
4. Search a Process (s)
5. Quit (q)
Enter your choice:s
Enter the name of the process to search: gedit
```

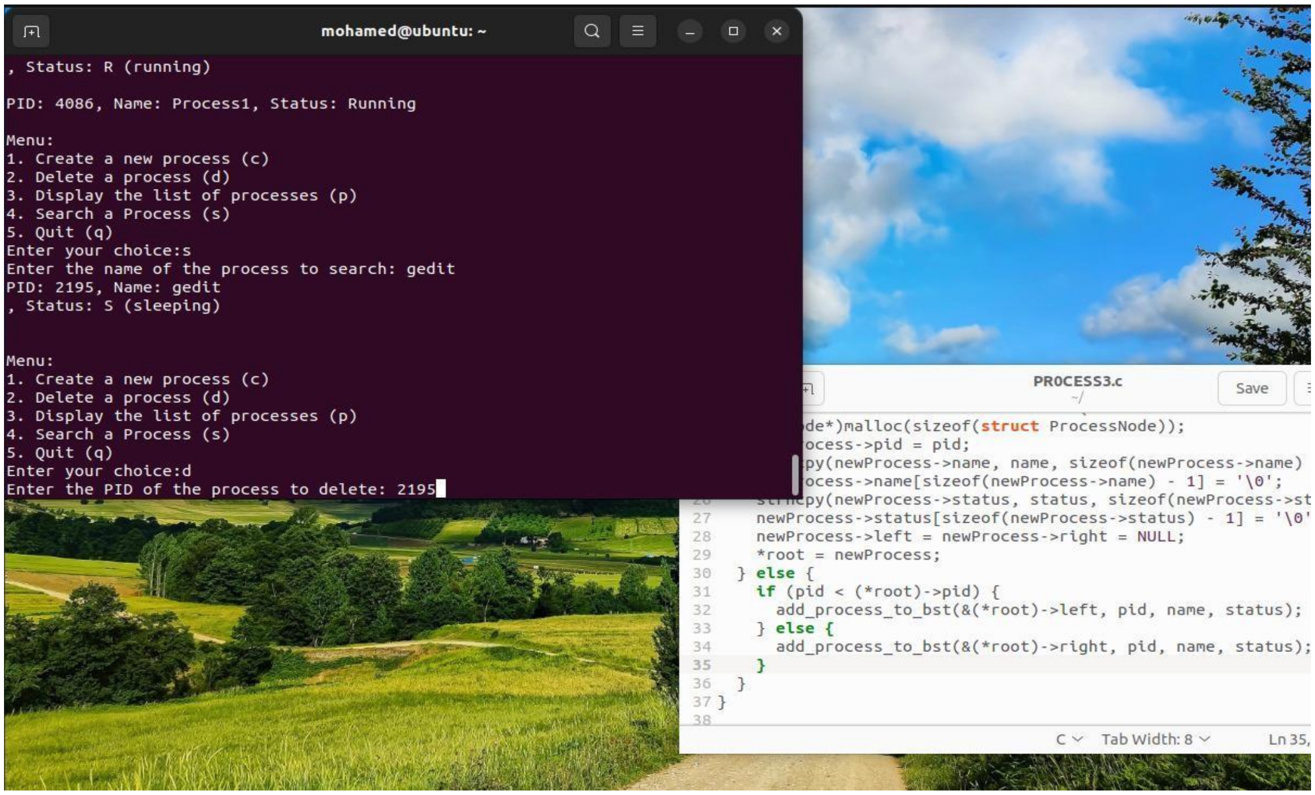
Fig 4.5.1



```
Menu:
1. Create a new process (c)
2. Delete a process (d)
3. Display the list of processes (p)
4. Search a Process (s)
5. Quit (q)
Enter your choice:s
Enter the name of the process to search: gedit
PID: 2195, Name: gedit
, Status: S (sleeping)


Menu:
1. Create a new process (c)
2. Delete a process (d)
3. Display the list of processes (p)
4. Search a Process (s)
5. Quit (q)
Enter your choice:
```

Fig 4.5.2

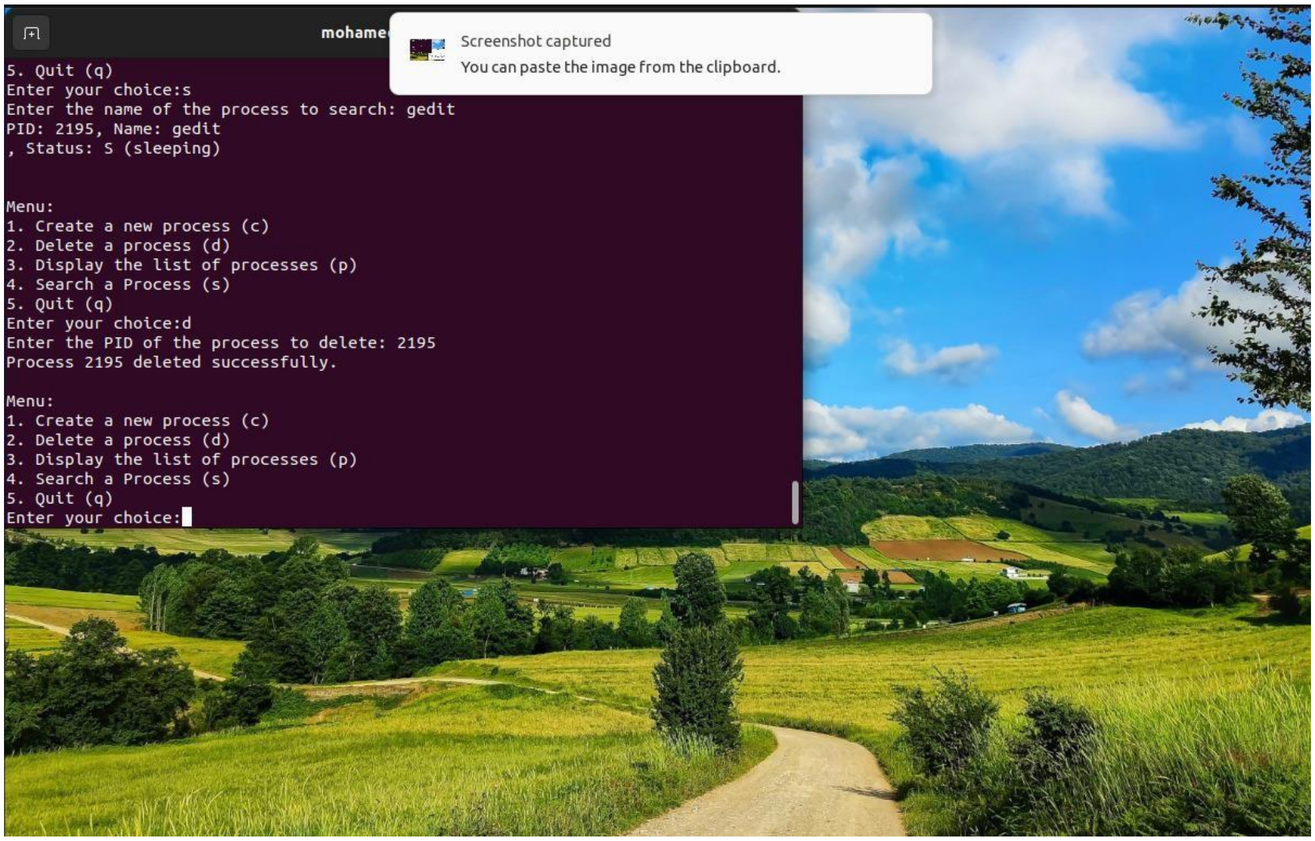
Delete a process:



```
, Status: R (running)

PID: 4086, Name: Process1, Status: Running

Menu:
1. Create a new process (c)
2. Delete a process (d)
3. Display the list of processes (p)
4. Search a Process (s)
5. Quit (q)
Enter your choice:s
Enter the name of the process to search: gedit
PID: 2195, Name: gedit
, Status: S (sleeping)


Menu:
1. Create a new process (c)
2. Delete a process (d)
3. Display the list of processes (p)
4. Search a Process (s)
5. Quit (q)
Enter your choice:d
Enter the PID of the process to delete: 2195
```

```
de*)malloc(sizeof(struct ProcessNode));
ocess->pid = pid;
py(newProcess->name, name, sizeof(newProcess->name)
ocess->name[sizeof(newProcess->name) - 1] = '\0';
strncpy(newProcess->status, status, sizeof(newProcess->st
    newProcess->status[sizeof(newProcess->status) - 1] = '\0'
    newProcess->left = newProcess->right = NULL;
    *root = newProcess;
  } else {
    if (pid < (*root)->pid) {
      add_process_to_bst(&(*root)->left, pid, name, status);
    } else {
      add_process_to_bst(&(*root)->right, pid, name, status);
    }
  }
}
```

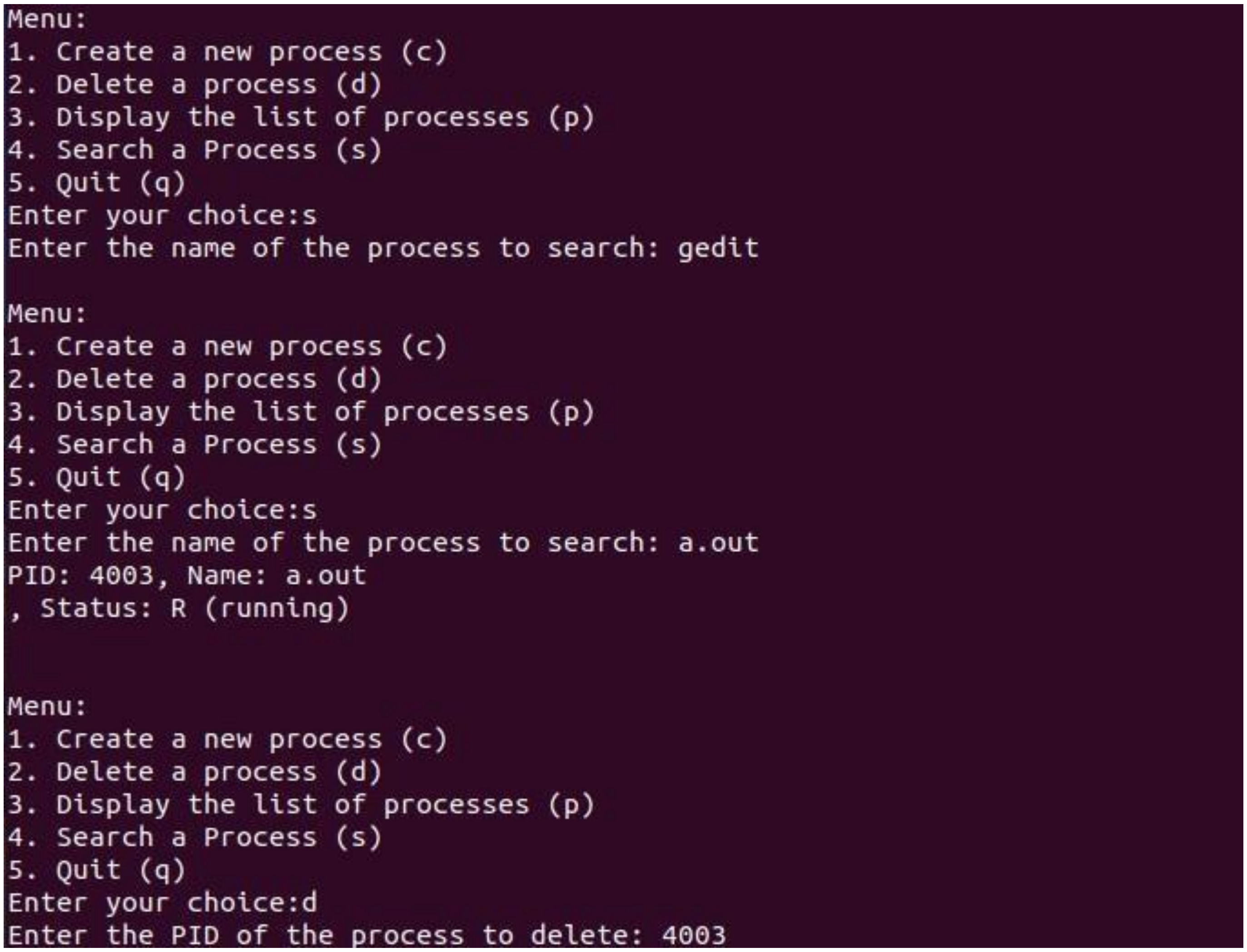
Fig 4.6.1 (**Before Deletion**)



```
5. Quit (q)
Enter your choice:s
Enter the name of the process to search: gedit
PID: 2195, Name: gedit
, Status: S (sleeping)


Menu:
1. Create a new process (c)
2. Delete a process (d)
3. Display the list of processes (p)
4. Search a Process (s)
5. Quit (q)
Enter your choice:d
Enter the PID of the process to delete: 2195
Process 2195 deleted successfully.

Menu:
1. Create a new process (c)
2. Delete a process (d)
3. Display the list of processes (p)
4. Search a Process (s)
5. Quit (q)
Enter your choice:
```

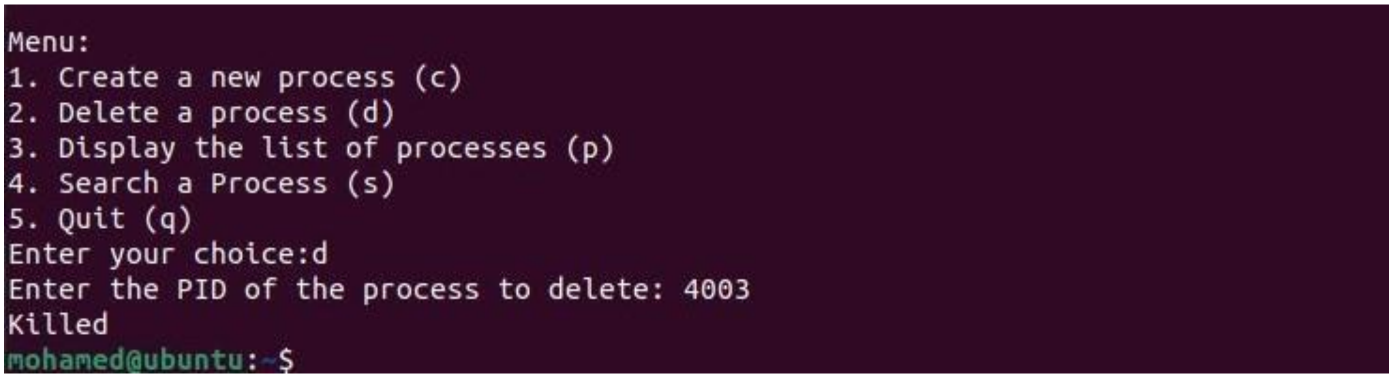
Fig 4.6.2 **(After Deletion)**

Killing the Program running process



```
Menu:
1. Create a new process (c)
2. Delete a process (d)
3. Display the list of processes (p)
4. Search a Process (s)
5. Quit (q)
Enter your choice:s
Enter the name of the process to search: gedit

Menu:
1. Create a new process (c)
2. Delete a process (d)
3. Display the list of processes (p)
4. Search a Process (s)
5. Quit (q)
Enter your choice:s
Enter the name of the process to search: a.out
PID: 4003, Name: a.out
, Status: R (running)


Menu:
1. Create a new process (c)
2. Delete a process (d)
3. Display the list of processes (p)
4. Search a Process (s)
5. Quit (q)
Enter your choice:d
Enter the PID of the process to delete: 4003
```

Fig 4.6.3



```
Menu:
1. Create a new process (c)
2. Delete a process (d)
3. Display the list of processes (p)
4. Search a Process (s)
5. Quit (q)
Enter your choice:d
Enter the PID of the process to delete: 4003
Killed
mohamed@ubuntu:~$
```

Fig 4.6.4

CHAPTER 5

CONCLUSION

The "Process Monitor" project presents an impressive and intricate solution for Linux process management. Its foundation lies in navigating the system's /proc directory, employing a Binary Search Tree (BST) structure to systematically organize and extract essential process details. This design choice enables a wide array of functionalities crucial for efficient process oversight within the Linux environment.

The BST implementation serves as the backbone of the tool, facilitating key operations such as addition, deletion, searching, and systematic display of processes based on their unique Process IDs (PIDs). By leveraging this structure, the tool preserves hierarchical order when adding new processes, enabling seamless management while maintaining system integrity.

One of the notable features of the "Process Monitor" is its ability to perform detailed searches, extracting specific processes based on user-defined names or substrings. This functionality enhances usability by allowing users, including system administrators, to pinpoint and focus on particular processes for monitoring or management purposes.

Moreover, the tool provides the capability to terminate identified processes while efficiently removing their entries from the BST. This feature empowers users to take decisive actions when needed, ensuring responsive and effective process control.

The structured and sorted presentation of stored processes through an intuitive menu-driven interface enhances user interaction. This design choice simplifies navigation and control, empowering both system administrators and regular users to interact seamlessly with the operational processes of the system.

Looking toward future developments, potential enhancements could include real-time monitoring capabilities. Integrating features that provide live updates on running processes could significantly benefit system administrators in monitoring system performance and identifying potential issues promptly. Additionally, enhancing the tool's security through robust user authentication mechanisms would be essential. Strengthening access controls and authentication procedures can safeguard the "Process Monitor" and the system it operates within, ensuring that only authorized individuals have control over critical processes.

Overall, the "Process Monitor" stands as a comprehensive and robust tool that streamlines process management and aids in system monitoring within a Linux environment. Its current functionalities provide a strong foundation for potential future improvements aimed at enhancing real-time monitoring capabilities and fortifying system security.

CHAPTER 6

FUTURE SCOPE

As technology advances at an unprecedented pace, the "Process Monitor" project not only lays a robust foundation for contemporary system process management but also sets the stage for an expansive and dynamic future scope. The trajectory ahead envisions a convergence of cutting-edge capabilities, addressing the evolving needs of complex computing environments. From real-time monitoring enhancements to fortified security measures, the project charts a course towards a horizon marked by sophisticated features and comprehensive system optimization. In this journey, the integration of advanced functionalities such as process grouping, cross-platform adaptability, and a user-friendly graphical interface promises to redefine how we perceive and manage system processes. As we delve into the future, the "Process Monitor" is poised to be a pivotal cornerstone, orchestrating and optimizing system processes across a diverse technological landscape, ensuring adaptability and relevance in the ever-evolving realm of information technology.

- Real-Time Monitoring: Implement dynamic live updates to provide real-time information on process statuses, resource consumption, and overall system performance.

- Enhanced Security Measures: Integrate sophisticated user authentication mechanisms like multi-factor authentication to fortify access control and elevate system security.

- Exhaustive Process Analysis: Extend functionalities to perform comprehensive process analysis, including deep dives into resource utilization metrics.

- Performance Bottleneck Identification: Develop capabilities to identify performance bottlenecks and provide optimization insights for better system efficiency.

- Process Grouping: Introduce process grouping features for holistic management, categorizing processes based on commonalities or functionalities.

- Cross-Platform Adaptability: Ensure compatibility with various Linux distributions and potentially extend support to other Unix-like systems for broader applicability.

- Graphical User Interface (GUI): Provide a visual facelift through the implementation of a user-friendly GUI, enhancing accessibility and comprehension of intricate process data.

- Automated Process Handling: Integrate automated process handling capabilities, such as auto-restart for failed processes or dynamic resource allocation, to streamline operational efficiency.

- Integration with Existing Frameworks: Facilitate seamless integration with existing system monitoring frameworks, creating a comprehensive ecosystem for system management.

- Technological Landscape Optimization: Position the "Process Monitor" as a pivotal cornerstone for orchestrating and optimizing system processes across diverse technological landscapes.

The "Process Monitor" project aims to revolutionize system process management by incorporating real-time monitoring, robust security measures, exhaustive process analysis, and advanced features like process grouping and cross-platform adaptability. The envisioned GUI and automated process handling capabilities add user-friendliness and operational efficiency to the tool. The ultimate goal is to integrate it seamlessly with existing frameworks, creating a comprehensive ecosystem for system management across diverse technological landscapes.

CHAPTER 7

REFERENCE

- Research Paper on Operating System | ISSN: 2320-2882
  Authors: Anjalee Sahu (1st), Asst. Prof. Shrikant Singh (2nd), HOD Rahul Chawda (3rd)
  Department of Computer Science, Kalinga University, Naya Raipur, Raipur Chhattisgarh 492101, India

- Operating System Research Paper in Course: Machine Learning (INT 246)
  Research Paper No: 11913362
  Lovely Professional University

- A Study on the Usage of Data Structures in Information Retrieval
  Authors: V. R. Kanagavalli (Associate Professor), G. Maheeja (PG Student)
  Department of Computer Applications, Sri Sai Ram Engineering College

- Recent Trends in User Authentication - A Survey
  Digital Object Identifier: 10.1109/ACCESS.2019.DOI
  Authors: Syed W. Shah and Salil S. Kanhere (Senior Member, IEEE)
  School of Computer Science and Engineering, The University of New South Wales, Australia

- A Continuous Improvement and Monitoring Performance System: Monitor - Analysis Vol. 2011
  Article ID: 917557, 15 pages
  DOI: 10.5171/2011.917557
  Authors: Vito Romaniello, Paolo Renna, and Vincenzo Cinque

- Thinking about binary trees in an object-oriented world
  Authors: A. Michael Berman from Rowan College of New Jersey and Robert C. Duvall from Brown University."