Process Monitor

using Binary Search Tree and Linux OS

Presented By:
Mohamed Irsath-055
Jenil Jain-070

Guided By:
Dr.Shri Bharathi

Agenda

- Problem Statement
- Abstract
- Requirements
- Algorithm
- Papers

Problem Statement

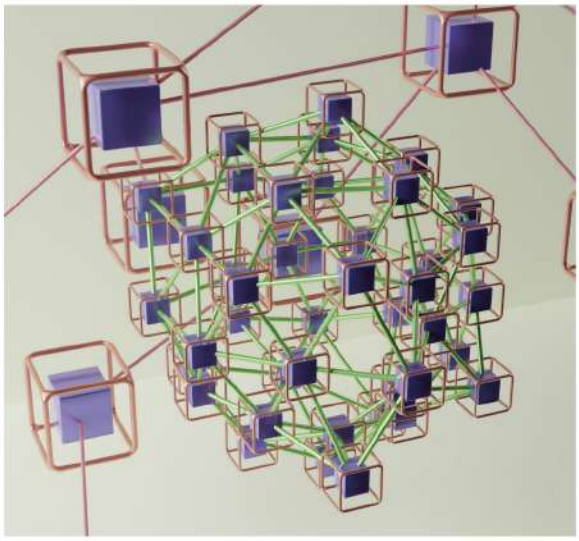
Process monitoring is an important task for system administrators. It allows them to identify and troubleshoot performance problems, detect and respond to security threats, and ensure that critical processes are running and available.

Abstract

This project aims to develop a process monitor in C on Linux. The process monitor will track all running processes on the system and display their information, such as process ID (PID), process name, parent process ID, CPU usage, memory usage, and start time. It will also allow users to search for specific processes, kill processes, and start new processes.

The process monitor will be implemented using the Linux system calls `getpid()`, `getppid()`, `getrusage()`, `kill()`, `fork()`, and `execvp()`. It will use a linked list to track all running processes on the system and a binary search tree to allow users to search for specific processes. The process monitor will be a command-line tool. The following command will start the process monitor: ./process_monitor. The process monitor will display a list of all running processes on the system. The user can use the arrow keys to navigate through the list and the S, K, and R keys to search for specific processes, kill processes, and start new processes, respectively.

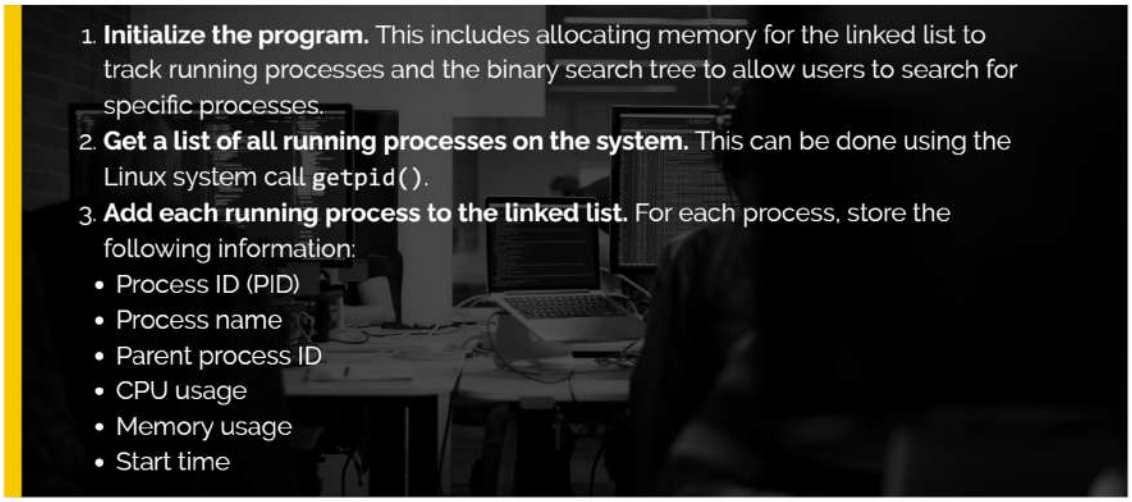
This project will develop a useful tool for monitoring and managing processes on Linux systems. The process monitor will be easy to use and will provide users with valuable information about running processes.



Requirements to run Process Monitor

- C or C++ Compiler
- Linux

Algorithm



1. **Initialize the program.** This includes allocating memory for the linked list to track running processes and the binary search tree to allow users to search for specific processes.
2. **Get a list of all running processes on the system.** This can be done using the Linux system call `getpid()`.
3. **Add each running process to the linked list.** For each process, store the following information:
   - Process ID (PID)
   - Process name
   - Parent process ID
   - CPU usage
   - Memory usage
   - Start time

1. **Build the binary search tree from the linked list.** This will allow users to efficiently search for specific processes.
2. **Start the main loop.** The main loop will continuously display the list of running processes and allow users to search for specific processes, kill processes, and start new processes.
3. **When the user wants to search for a specific process, use the binary search tree to find the process.**
4. **When the user wants to kill a process, use the Linux system call `kill()` to send a SIGKILL signal to the process.**
5. **When the user wants to start a new process, use the Linux system calls `fork()` and `execvp()` to create and execute a new process.**
6. **Update the linked list and binary search tree to reflect any changes to the list of running processes.**
7. **Repeat steps**

Application

Papers

- **Papers:**
- "Process Monitoring Using Machine Learning: A Survey" by Mohamed Medhat Gaber and Mohamed Salah Mohamed (2020)
- "A Comprehensive Survey on Process Monitoring Techniques" by Jialiang Zhang, Li Guo, and Hui Wang (2021)
- "Prescriptive Process Monitoring for Cost-Aware Cycle Time Reduction" by Tobias Jung, Manuel Karg, and Mathias Weske (2021)