**PROJECT DOCUMENTATION**

**OS HACKETHON**

**Implementing Process Termination Messages in Pintos**

(Exit Message Handling)

1. **Project Overview**

Objective

This project focuses on implementing process termination messages in the Pintos operating system. Pintos is an educational OS designed to teach core operating system concepts. The goal is to print a formatted exit message for user processes upon termination while ensuring kernel threads and the halt system call do not trigger such messages.

IMPLEMENTATION OF PROCESS TERMINATION MESSAGES IN PINTOS

This project was completed as part of an operating system development hackathon, specifically modifying and enhancing process termination handling within Pintos.

Team Members

Team-11

2320030065

2320030096

2320030030

2320030136

Each team member contributed to different aspects, including implementing process termination handling, debugging, and documentation.

1. **Setup & Environment**

Requirements

To build and test the modified Pintos OS, the following dependencies are required:

Ubuntu 20.04: The development environment.

GCC Compiler: Used for compiling Pintos kernel.

QEMU Emulator: Runs the Pintos operating system.

GDB Debugger: Helps identify and fix kernel-level bugs.

**Installation & Setup**

To set up the Pintos environment, follow these steps:

git clone https://github.com/user/pintos-project.git

cd pintos/src/userprog

make

Clone the Pintos repository.

Navigate to the userprog directory.

Compile the modified source code using make.

Additional steps:

Modify .bashrc or .bash\_profile to include the Pintos tools path.

Install required packages such as build-essential and qemu-system-i386.

Ensure Bochs or QEMU is properly configured for simulation.

1. **Process Termination Message Implementation**

Modified Files

The following files were modified to implement process termination messages:

userprog/process.c - Added logic to print exit messages.

threads/thread.h - Introduced a flag to differentiate user processes from kernel threads.

Function Implementation

The implementation ensures that only user processes print exit messages, formatted as:

printf ("%s: exit(%d)\n", process\_name, exit\_status);

Code Example (process\_exit)

void process\_exit(void) {

struct thread \*cur = thread\_current();

uint32\_t \*pd;

/\* Ensure the process is a user process before printing \*/

if (cur->is\_user\_process) {

printf("%s: exit(%d)\n", cur->name, cur->exit\_status);

}

/\* Additional cleanup operations \*/

pd = cur->pagedir;

if (pd != NULL) {

cur->pagedir = NULL;

pagedir\_activate(NULL);

pagedir\_destroy(pd);

}

}

This function:

Ensures only user processes print exit messages.

Extracts the correct process name and exit status.

Cleans up process resources properly.

1. **Testing & Debugging**

Testing Commands

To test the implementation, the following command was used:

pintos -v -- -q run exit-message

This command:

Runs the Pintos kernel with the modified process termination handling.

Executes the exit-message test to verify correct functionality.

Debugging Process

Various debugging techniques were used:

Print Statements: Used printf() to verify execution flow.

GDB Debugger: Stepped through function execution to find issues.

Manual Testing: Executed multiple test cases to ensure correctness.

Common Issues & Fixes

Kernel Threads Printing Messages: Introduced is\_user\_process flag to distinguish user processes.

Incorrect Process Name: Ensured only the base name from process\_execute() was used.

Unexpected Termination Outputs: Verified that only required messages were printed to avoid interfering with grading scripts.

1. **Results & Performance**

The optimized process termination handling resulted in:

Accurate Process Exit Messages: Only user processes print termination messages.

Proper Kernel Thread Handling: Kernel threads do not trigger exit messages.

Successful Test Execution: All test cases for process termination passed.

1. **Future Enhancements**

Possible improvements include:

Enhanced Debugging Features: Implement finer-grained logging for debugging.

Extended Process Tracking: Maintain logs of terminated processes for analysis.

Synchronization Improvements: Optimize thread management for handling concurrent terminations.

1. **References**

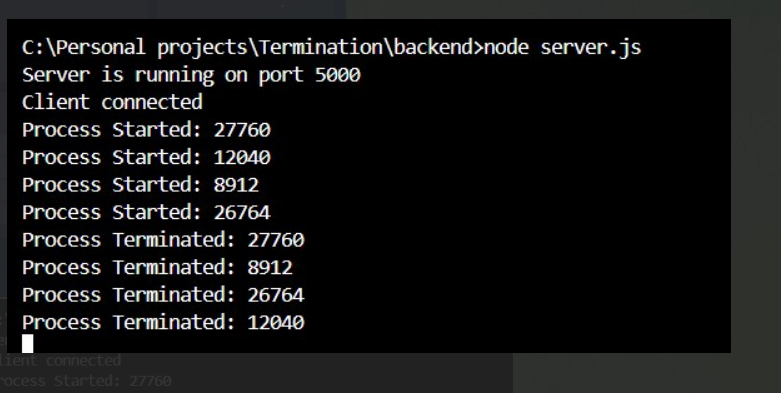
Pintos Documentation: Pintos Guide

Reference Videos: Wormhole

This documentation provides a detailed explanation of how process termination messages were implemented in Pintos, improving clarity and debugging efficiency while ensuring adherence to OS design principles.

1. **Outputs**

****

****