IM3014 Linux System Administration Practice
Homework 1: Smart Trash Can System

Introduction

You are a junior system administrator at **IM Corp**. Recently, several of your colleagues have complained about accidentally deleting important files using the `rm` command. Since `rm` removes files permanently without confirmation, even a small typo can lead to data loss.

To address this issue, your manager has asked you to design a safer alternative: A **Smart Trash Can System** on Linux. Witch consist of a smart remove command called `srm`. Instead of deleting files immediately, `srm` will move them into a special "trash" directory, along with metadata about their original location and deletion time. A companion tool, `srm-restore`, will allow users to recover mistakenly removed files, and another tool, `srm-empty`, will automatically purge files that have been in the trash for too long.

This project will help you practice several key skills in Linux system administration:

- Writing robust Bash scripts with error handling.
- Managing filesystems, metadata, and permissions.
- Understanding how system utilities like `rm`, `mv`, `du` and `tar` work together.
- Using `systemd` user services and timers to automate recurring maintenance tasks.

By the end of this assignment, you will have built a simple but practical tool that makes Linux environments safer for everyday users while deepening your understanding of process control, filesystem management, and service automation.

Problem Specification

Your task is to design and implement the **Smart Trash Can System**, which includes three command-line utilities and a background systemd service to periodically clean up deleted files and directories. The expected behavior of the system is as follows:

Overview

The system maintains a dedicated directory structure inside the user's home directory, rooted at `<home-directory>/.trash`. Within this root, all deleted payloads are stored in `<home-directory>/.trash/files`, while the corresponding metadata log files are stored in `<home-directory>/.trash/meta`.

Whenever a file or directory is removed with `srm`, a unique identifier `<key>` must be generated to avoid collisions. The key has the format `<basename>.<date>.<pid>.<counter>`, where each component is defined as follows:

- `<basename>` — the basename of the file or directory being removed.
- `<date>` — the current date when `srm` is executed, expressed as `YYYY-MM-DD`.
- `<pid>` — the process ID of the shell running the `srm` command.
- `<counter>` — an integer starting at 0 and incremented until a collision-free key is found.

For each removal operation, a metadata file `<home-directory>/.trash/meta/<key>_meta.csv` is created. This file records information about the trashed item in three columns:

- `base_path` — the base path of the file or directory before deletion.
- `deleted_iso8601` — the timestamp of the deletion operation in ISO 8601 format (e.g., `2025-09-06T10:21:33+08:00`).
- `size_bytes` — the total size of the deleted file or directory in bytes.

In addition, the removed content is archived into a compressed payload file, stored as `.trash/files/<key>.tar.gz`. Note that when running in superuser mode (sudo), the files should still be directed to the current user's home directory.

Command: `srm`

The command `srm` serves as a safer alternative to the traditional `rm` utility. Its primary function is to move compressed files or, when explicitly requested, directories into the dedicated trash area rather than deleting them permanently. By default, `srm` only accepts files as arguments. If the user provides the `-r` flag, entire directories may also be removed recursively. The operation of `srm` ensures that no data is irreversibly lost at the time of execution.

Command: `srm-restore`

The command `srm-restore` provides users with the ability to recover files or directories that have previously been removed using `srm`. Instead of immediately restoring all items, the program operates in an interactive fashion: it enumerates the contents of the trash, displays them in a tabular form, and prompts the user to select one item to restore. If the original location still exists at the time of restoration, the program avoids overwriting by appending a suffix of the form `.restored.<YYYYmmdd-HH:MM>` to the restored file or directory name.

Here is an example of such interface:

```
$ srm-restore
Index | Deleted ISO8601           | Size   | Type | Key                    | Original Path
------+---------------------------+--------+------+------------------------+-----------------------------
    0 | 2025-09-06T10:21:33+08:00 |  48201 | FILE | report.md.1756...0     | /home/ntuim/Documents/report.md
    1 | 2025-09-06T11:03:05+08:00 |1048576 | DIR  | project.1756...0       | /home/ntuim/Projects/project
    2 | 2025-09-06T12:44:10+08:00 |   4096 | FILE | notes.txt.1756...1     | /home/ntuim/notes.txt

Enter Index to restore: 1
Original exists; restoring to: /home/ntuim/Projects/project.restored.20250906-12:48
Restored to: /home/ntuim/Projects/project.restored.20250906-12:48
```

Command: `srm-empty`

The command `srm-empty` permanently removes items from the trash that have exceeded a defined retention period. It scans the metadata directory `<home-directory>/.trash/meta`, parses the recorded deletion timestamps, and compares them against the current time. Items older than the configured threshold are deleted from both `files/` and `meta/`. If a timestamp cannot be parsed, the entry is skipped with a warning.

The retention period is set with the environment variable `TRASH_MAX_AGE_DAYS`, which defaults to seven days. At the end of its run, the program prints a summary line such as `Purged <N> item(s) older than <D> day(s).`.

Background Service

To automate the cleanup process, the program `srm-empty` is designed to run periodically as a background service under `systemd`. This is accomplished by defining a user-level service unit, `srm-empty.service`, which executes the purge operation once, and a companion timer unit, `srm-empty.timer`, which schedules it to run on a regular basis. In this assignment, the timer is configured to trigger **once per day** to prevent unnecessary disk usage.

Self-Testing Guide

After implementing the three programs and the background service, you should verify that each component behaves as expected. The following steps outline a minimal self-test procedure.

```
# Step 1: Setup
$ mkdir -p ~/lab
$ echo "alpha" > ~/lab/a.txt
$ echo "beta"  > ~/lab/b.txt

# Step 2: Testing srm
$ srm ~/lab/a.txt
# Expect: Trashed: /home/<user>/lab/a.txt
```

```
ls -l ~/.trash/files
ls -l ~/.trash/meta

# Step 3: Testing srm -r
$ srm ~/lab
# Expect: srm: cannot remove ~/lab, is a directory (use -r to trash recursively)

$ srm -r ~/lab
# Expect: Trashed: /home/<user>/lab

# Step 4: Testing srm-restore
$ srm-restore
# Expect: table of entries and prompt to restore

# Step 5: Testing srm-empty
$ TRASH_MAX_AGE_DAYS=0 srm-empty
# Expect: Purged <N> item(s) older than 0 day(s).

# Step 6: Testing systemd service
$ systemctl --user daemon-reload
$ systemctl --user enable --now srm-empty.timer
$ systemctl --user list-timers srm-empty.timer
# Expect: next run scheduled (daily)
```

Deliverables

You are required to setup the Smart Trash Can system on your virtual machine during the designated time period (from 2025-09-29 to 2025-10-6) and submit the following materials:

1. **Source Code:** Provide the complete source code for the three commands (`srm`, `srm-restore`, and `srm-empty`) and the configuration file for `systemd` service. You may implement them in any programming language of your choice, but they must be installed and callable as standard commands. For example, invoking them with `srm file.txt` is acceptable, whereas running them via an interpreter such as `python srm.py` is NOT!
2. **System Setup Report:** Write a report (maximum length: 3 pages) that documents:
   - how you configured the system;
   - how the commands were integrated into your environment so that they behave like native utilities;
   - screenshots or command outputs that demonstrate the system running correctly, including deletion, restoration, and background cleanup.

Your submission should be concise, technically clear, and sufficient to allow another system administrator to reproduce your setup.

Words from the TA

You may notice that the specification for this assignment is less detailed than other programming tasks you may have encountered in the past. For example, the exact approach to error handling or the precise format of the user interface. This reflects the reality of software development: requirements in the real world are often incomplete or ambiguous, and many design decisions are ultimately left to the engineer.

We encourage you to embrace this flexibility, make reasonable design choices, and document them clearly in your report. In addition, you are strongly encouraged to use modern generative AI tools, such as ChatGPT, to guide you through this assignment, whether for brainstorming, debugging, or refining your implementation.