Họ Tên: Hà Cao Đức Minh

MSSV: 2312059

LAB 1

Introduction to Linux Programming

REPORT

CONTENTS

ANSWER QUESTIONS

Question 1:

Compare the Output Redirection (>/>>) with the Piping (|) technique.

Answer:

Feature	Output Redirection (>/>>)	Piping (|)
Primary Goal	Save command output to a file	Chain commands for data processing
Output Destination	File	Standard input of another command
Input Source	Standard input of the command is unchanged	Standard output of the preceding command
Data Flow	Command Output → File	Command Output → Next Command Input
Operators	>: (overwrite) >>: (append)	|: (pipe)
Use Cases	Logging, saving results, creating files	Filtering, transforming data, complex workflows

Use Output Redirection (> or >>) when we want to store the results of a command in a file for later use.

Use Piping (|) when we want to use the output of one command as the input for another command to perform a series of operations in a single command line.

Question 2:

Compare the sudo and the su command.

Answer:

Feature	sudo (superuser do)	su (substitute user)
Primary Goal	Execute a single command with superuser privileges.	Become the superuser (root user) or another user.
Privilege Scope	Command-specific privilege elevation.	Full shell session with the privileges of the target user.
User Context	Executes command in the current user's environment.	Switches the entire user context to the target user.
Authentication	Typically authenticates the current user's password.	Authenticates with the target user's password (usually root).
Configuration	Controlled by sudoers file, allowing fine-grained permissions.	Less configurable, relies on standard user account permissions.
Logging/Auditing	Logs each sudo command, enhancing security and accountability.	Auditing can be less straightforward as it involves user switching.
Security	Generally considered more secure. Principle of least privilege.	Can be less secure if used unnecessarily. Full root access.
Use Cases	- Running administrative commands occasionally.	- Becoming root for a longer administrative session (less common now). - Switching to another user account.
Example	sudo apt update (updates package lists as root)	su - (becomes root user in a new login shell) su username (switches to user 'username')

Question 3:

Discuss about the 777 permission on critical services (web hostings, sensitive databases, ...).

Answer:

The 777 permissions are a major security concern for critical services that should be completely avoided. It creates wide-open access for attackers and can lead to complete compromise of your systems and data. Instead of 777 permissions, web hosting could use 765 and sensitive databases should use 750 permissions.

Question 4.1:

What are the advantages of Makefile? Give examples?

Answer:

First, Makefile provides us a method to automatically build processes. Second, it automates the build process, from compiling source code to linking object files and creating executable files. Third, Makefile also only recompiles what is necessary and manages dependencies for you. Finally, customization is easier with Makefile, define your own rules, variables and functions to tailor the build process.

Question 4.2:

Compiling a program for the first time usually takes a longer time in comparison with the next re-compiling. What is the reason?

Answer:

The first time compilation is a full build, Makefile processes every single source file and goes through all stages of compilation. Recompilation, on the other hand, identifies only changed files and recompiles those.

Question 4.3:

Is there any Makefile mechanism for other programming languages? If it has, give an example?

Answer:

There are Makefile mechanisms for other programming languages:

- In Rust, Cargo is the build system and package manager.

```
# Configuration for Rust
# Cargo.toml
[package]
name = "my-rust-project"
version = "0.1.0"
edition = "2021"

[dependencies]
regex = "1.0"

[dev-dependencies]
criterion = "0.3"

[[bench]]
name = "my_benchmark"
harness = false
```

- In Python, Makefile can be used, but tox or invoke is more common.

```
# Makefile for Python project

PROJECT_NAME = my_python_project
VENV_DIR = venv
PYTHON = $(VENV_DIR)/bin/python  # Path to Python in
virtual environment
PIP = $(VENV_DIR)/bin/pip       # Path to pip in virtual
environment

# Default target (what happens when you just run `make`)
.DEFAULT_GOAL := all

all: venv lint test  # Build target: set up venv, lint,
then test

venv: $(VENV_DIR)/bin/activate  # Target to create
virtual environment
```

```
$(VENV_DIR)/bin/activate: requirements.txt
	@echo "⟶ Creating virtual environment..."
	python3 -m venv $(VENV_DIR)
	$(PIP) install -r requirements.txt
	touch $@ # Update timestamp of venv target

lint: venv  # Target to run linting (depends on venv)
	@echo "⟶ Running linting (flake8)..."
	$(PYTHON) -m flake8 my_module tests

test: venv  # Target to run tests (depends on venv)
	@echo "⟶ Running tests (pytest)..."
	$(PYTHON) -m pytest tests

clean:  # Target to clean up build artifacts
	@echo "⟶ Cleaning up..."
	rm -rf $(VENV_DIR)
	find . -name "*.pyc" -delete
	find . -name "__pycache__" -delete

.PHONY: all venv lint test clean # Declare phony targets
```

```
# Configuration for Python using invoke
from invoke import task

@task
def lint(c):
    c.run("flake8 my_python_project")

@task
def test(c):
    c.run("pytest")

@task(lint, test)
def build(c):
    c.run("python setup.py sdist bdist_wheel")

@task
def clean(c):
    c.run("rm -rf build dist *.egg-info")
```

EXERCISES OUTPUT

Exercise 3.6

```
gslay@admin:~/projects/BKU/OS_labs/lab_1$ ./calc.sh
>> 2 + 5
7
>> EXIT
gslay@admin:~/projects/BKU/OS_labs/lab_1$ ./calc.sh
>> ANS + 3
10
>> 1 + 3
4
>> 1 + 2
3
>> ANS - 3
0
>> 3 x 5
15
>> 9 % 6
3
>> 25 / 6
4.17
>> HIST
1 + 2 = 3
ANS - 3 = 0
3 x 5 = 15
9 % 6 = 3
25 / 6 = 4.17
>> EXIT
gslay@admin:~/projects/BKU/OS_labs/lab_1$ ./calc.sh
>> 1 / 0
MATH ERROR
>> 1 @ 2
SYNTAX ERROR
>> EXIT
```

Exercise 5.3

```
gslay@admin:~/projects/BKU/OS_labs/lab_1/Exercise_5.3$ make
gcc -Wall -g calc.c -o calc
gslay@admin:~/projects/BKU/OS_labs/lab_1/Exercise_5.3$ ./calc
>> 2 + 5
7
>> EXIT
gslay@admin:~/projects/BKU/OS_labs/lab_1/Exercise_5.3$ ./calc
>> ANS + 3
10
>> 1 + 2
3
>> ANS - 4
-1
>> 3 x 5
15
>> 9 % 6
3
>> 25 / 6
4.17
>> HIST
SYNTAX ERROR
>> EXIT
gslay@admin:~/projects/BKU/OS_labs/lab_1/Exercise_5.3$ ./calc
>> 1 / 0
MATH ERROR
>> 1 @ 2
SYNTAX ERROR
>> EXIT
```