System Programming - Linux
Week 15 - Review of Course Project (Answers)

Lecturer: Dr. Aggrey Obbo (PhD)

June 17, 2025

Contents

This exercise is designed to assess the student's practical and theoretical understanding of fundamental Linux system programming concepts. The student is required if possible to use a Linux virtual machine or environment such as Ubuntu to answer the practical questions.

Instructions

1. Complete all sections. For practical questions, execute the commands and briefly describe the outcome or relevant file changes.
2. For theoretical questions, provide clear and concise explanations.
3. Assume you are working in a fresh user directory.
4. The highrest obtainable mark is 100

1 Section 1: Linux Utilities and Installation

(a) Install the htop utility (a powerful interactive process viewer) on your Linux system.

(i) Provide the command(s) you would use.
sudo apt update # For Debian/Ubuntu
sudo apt install htop # For Debian/Ubuntu
# OR
sudo dnf install htop # For Fedora/CentOS/RHEL

(ii) Briefly explain what htop shows that ps doesn't by default.
htop provides an interactive, color-coded view of processes, showing CPU/memory usage per core/process, process trees, and easy sorting/filtering, unlike ps which gives a static snapshot by default.

(2.5 Marks for each right answer)

(b) Display your system's total RAM, used RAM, and free RAM in a human-readable format.

(i) Provide the command.
Command: free -h

(ii) Capture a snippet of its output.
Example Output Snippet: total used free shared buff/cache available Mem: 3.8Gi 1.2Gi 1.5Gi 100Mi 1.1Gi 2.3Gi Swap: 2.0Gi 0.0Ki 2.0Gi

(2.5 Marks for each right answer)

(c) Create a new user named devuser with a home directory and a default shell. Set a password for devuser (e.g., "DevP@pass").

(i) Provide the command(s).
Create User and Set Password: sudo adduser devuser # Follow prompts: Enter password "DevP@ss123", confirm, and accept defaults for other info.

(ii) Explain how you would verify the user was created.
Verification: You could check /etc/passwd or /etc/group for the new entry, or attempt to switch to the user: su - devuser (then enter password).

(2.5 Marks for each right answer)

2 Section 2: File Systems and Directories

(a) Create a directory structure as follows:

(i) ~/project_work/module1/src
(ii) ~/project_work/module1/doc
(iii) ~/project_work/module2/src
(iv) ~/project_work/module2/bin

(5 Marks)

(b) Provide the single command to achieve this. Inside ~/project_work/module1/src/, create an empty file named main.c. Then, change its permissions so that only the owner can read and write, while group and others have read-only access.

(i) Provide the command(s).
mkdir -p ~/project_work/module1/src,doc,module2/src,bin
(5 Marks)

(ii) Verify the permissions using ls -l.
Verification (ls -l output): -rw-r–r– 1 youruser yourgroup 0 Jun 16 20:00 main.c

(2.5 marks for each right answer)

(c) Change the ownership of the ~/project_work/module2/bin/ directory and all its contents to devuser (from Section 1) and its primary group.

(i) Provde the command(s)
sudo chown -R devuser:devuser ~/project_work/module2/bin/

(ii) Explain how you would verify the ownership change.
Verification: Use ls -lR ~/project_work/module2/bin/. The output should show devuser as the owner.

(2.5 marks for every tru answer)

(d) From your home directory (~), create a symbolic link named my_project that points to ~/project_work/module1/src/.

(i) Provide the command(s).
ln -s ~/project_work/module1/src/ ~/my_project

(ii) Provide the command and demonstrate that my_project behaves like a pointer by trying to cd into it.
cd ~/my_project pwd # Output: /home/youruser/project_work/module1/src

(2.5 Marks for each right answer)

3 Section 3: Inter-Process Communication and Redirection

(a) Use a pipe (|) to count the number of lines in the /etc/passwd file that contain the word "bash".

(i) Provide the command.
grep "bash" /etc/passwd — wc -l

(ii) State the output you get.
Example Output: (Varies by system, typically 1 or 2)
1

(2.5 marks for every right anwer)

(b) Redirect the standard output of the ls -l command to a file named file_list.txt in your current directory. Then, append the output of date to the same file_list.txt.

(i) Provide the command(s).
ls -l > file_list.txt date >> file_list.txt

(ii) Show the content of file_list.txt after both operations.
file_list.txt Content (Example): total 4 -rw-r–r– 1 youruser yourgroup 0 Jun 16 20:00 file_list.txt
drwxr-xr-x 3 youruser yourgroup 4096 Jun 16 20:00 project_work
lrwxrwxrwx 1 youruser yourgroup 32 Jun 16 20:00 my_project -> /home/youruser/project_work/module1/src/
Mon Jun 16 20:05:00 EAT 2025

(2.5 marks for every right answer)

(c) Execute a command that will intentionally produce an error (e.g., cat /nonexistent_file), but redirect its standard error to a file named error_log.txt. Ensure no error message appears on your terminal.

(i) Provide the command.
cat /nonexistent_file 2> error_log.txt

(ii) Show the content of error_log.txt.
error_log.txt Content is as follows:
cat: /nonexistent_file: No such file or directory

(2.5 marks for every right answer)

4 Section 4: Message Queues

(a) Describe a scenario where using a message queue would be a suitable IPC mechanism. Explain why it's suitable for that scenario compared to shared memory or pipes.
A logging system where multiple application processes need to send log messages to a single dedicated logging daemon process.
Why suitable: Message queues provide asynchronous communication and decoupling. Applications don't need to wait for the logger to be ready; they just drop messages into the queue. This is better than shared memory because it handles discrete messages easily, and better than pipes for multiple writers/readers (message queues manage access and ordering more robustly).
(5 Marks)

(b) Briefly explain the purpose of the following key POSIX message queue functions: mq_open(), mq_send(), and mq_receive().
mq_open(): Used to create a new message queue or open an existing one. It returns a message queue descriptor, similar to a file descriptor. You specify the queue's name, flags (e.g., read/write, create if not exists), and permissions.
mq_send(): Used by a process to add a message to the end of a message queue. You specify the queue descriptor, the message buffer, its size, and a priority value. Messages are typically placed on the queue based on priority.
mq_receive(): Used by a process to retrieve the highest-priority (or oldest, if priorities are equal) message from a message queue. It blocks if the queue is empty until a message arrives or a timeout occurs.
(5 Marks for all and 1 mark for any right answer less than 3)

5 Section 5: Semaphores

(a) Explain the primary problem that semaphores are designed to solve in concurrent programming.(5 Marks - Theoretical) Provide a simple analogy.
Primary Problem Solved by Semaphores: Semaphores are designed to solve the problem of synchronization and mutual exclusion in concurrent programming, specifically when multiple processes or threads need to access a shared resource (e.g., a shared variable, a file, a printer). Without synchronization, concurrent access can lead to race conditions and corrupted data.
Analogy: Imagine a single-stall restroom (the shared resource). A semaphore acts like a "key" or "occupancy counter". If the restroom is empty (semaphore value > 0), you take the key (decrement semaphore) and enter. If it's occupied (semaphore value $= 0$), you wait outside until someone exits and returns the key. (10 Marks)

(b) Describe the behavior of the two fundamental semaphore operations, often referred to as "P" (or wait()) and "V" (or post()). How do these operations ensure synchronization?
"P" (Proberen / Test, also known as wait() or sem_wait()): This operation decrements the semaphore's value. If the semaphore's value becomes negative after decrementing (meaning the resource is unavailable), the calling process/thread is blocked until the semaphore's value becomes non-negative (i.e., the resource becomes available). This ensures that a process waits if the resource is in use. "V" (Verhogen / Increment, also known as post() or sem _post()): This operation increments the semaphore's value. If there are processes/threads blocked waiting on this semaphore, one of them will be unblocked (made ready to run). This signals that a resource has been released and is now available. Synchronization: By using P before accessing a shared resource and V after releasing it, semaphores enforce mutual exclusion (only one process accesses the critical section at a time) and coordinate access, preventing race conditions. (10 Marks)

6 Section 6: Sockets

(a) Briefly compare and contrast TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) in terms of reliability and connection state.
TCP (Transmission Control Protocol): Reliable (guarantees delivery, retransmits lost data), connection-oriented (establishes a dedicated logical connection via a three-way handshake), provides ordered data delivery.
UDP (User Datagram Protocol): Unreliable (no delivery guarantee, no retransmissions), connectionless (sends discrete packets without prior setup), provides unordered data delivery.
(5 Marks)

(b) Outline the logical sequence of operations (system calls) a TCP client typically performs to establish a connection and send data to a server.
TCP Client Workflow:
socket(): Create a stream socket.
Prepare sockaddr_in (server's IP and port, in network byte order).
connect(): Initiate connection to the server.
send() / recv(): Exchange data.
close(): Close the socket.
(5 Marks)

(c) Outline the logical sequence of operations (system calls) a TCP server typically performs to listen for, accept, and handle a single client connection.
socket(): Create a stream socket (listener).
bind(): Assign local IP and port to the listener socket.
listen(): Put the listener socket into a passive listening state.
accept(): Block and wait for a client connection, returning a new socket for that client.
send() / recv(): Exchange data with the accepted client.
close(): Close the client-specific socket. (Loop back to accept() for next client).
(5 Marks)

(d) You are trying to connect to a web server (e.g., example.com on port 80), but your client program consistently receives a "Connection refused" error. List two common reasons for this error and suggest a Linux command-line utility you could use to diagnose each reason.
Reason 1: No server listening on the specified port. The server application might not be running, crashed, or

listening on a different port. Diagnosis Command: ss -tuln | grep <port_number> (e.g., ss -tuln | grep 80). If no output, nothing is listening.
Reason 2: Firewall blocking the connection. A firewall on either the client or server host (or an intermediate network firewall) is dropping incoming connection attempts to that port. Diagnosis Command: From the server, check firewall rules using sudo ufw status (Ubuntu) or sudo firewallcmd -list-all (Fedora), or sudo iptables -L -n -v. Look for rules that deny traffic to the target port. (5 Marks)

End.