

National University of Computer and Emerging Sciences, Lahore Campus



Course:	Operating Systems	Course Code:	CS 2006
Program:	BSCS- 4E	Semester:	Spring 2025
Due Date	16 February, 2025 11:59 PM	Total Marks:	100 marks
Instructor	Mubashar Hussain	TA	L226598@lhr.nu.edu.pk
Type:	Assignment 1	Questions:	3

Important Instructions:

1. Submit the zip file with your roll number with section (e.g., 23L-1011_4E) as folder name.
2. **Heavy penalties will be given to all students involved in plagiarism.**
3. Late submission of your solution is not allowed.
4. Your codes will be evaluated thoroughly so make sure you understand all the concepts well.
5. Viva maybe conducted hence it is advised to come prepared
6. In case of any queries, you can always reach out to me via email.

Question 1: Secure Data Transformation

[40 marks]

Scenario:

Imagine you are working as a security analyst in a high-stakes investigation firm. Your team has received a critical piece of evidence in the form of a text file named "classified.txt" containing sensitive information. The file is highly confidential and gets tampered with due to a bug, making it unreadable in its current state. To ensure confidentiality and data integrity, you must develop a C/C++ program for secure data transformation

Your Tasks:

- **Decrypt** - Your team is finally able to find the decryption key and is ready for a secure data decryption process.
(Decryption Key: Shift 1 character backward (Caesar cipher))
Encrypted Data: "Uifsf jt b tfdsfu npofz"
Decrypted Data: "There is a secret message"
- **Redact** - After successful decryption your team agreed to scan confidential information such as SSN (in format "###-##-###-#") and securely redact them, replacing them with a 12 letter word **"CONFIDENTIAL"**
Original Data: "Please find enclosed the SSN: 123-45-678-9"

Redacted Data: "Please find enclosed the SSN: CONFIDENTIAL"

- **Report** - In accordance to rule and regulation, names of the ones responsible should be appended at the end of the file.

Requirements:

Process Creation (fork())

- Child A handles the decryption phase and **writes** the decrypted text to a newfile called **"decrypted.txt"**.
- Child B uses the decrypted file and **appends** the redacted data to a new file **"output.txt"**.
- Parent process finally **appends** to **"output.txt"** the Child PID labelling the task handled by them.

Process Coordination (waitpid())

- The Parent Process must wait for Child A to complete before Child B starts.
- The Parent Process must wait for Child B before writing PIDs to "output.txt".

File Handling (fstream)

- Use **only** system calls (open,read,write,etc).
- **Do not use** ifstream, ofstream, fscanf(), fgets(), fprintf(), or fputs().

Process Termination (exit())

- Each child process should call exit() after completing its assigned task.

Example:

classified.txt:

Kpio Epf, TTO: 2340560789a, Beesftt: 234 Nbjo Tu
Kbof Tnjud, TTO: a9807605432, Beesftt: 567 Fmn Tu

decrypted.txt:

John Doe, SSN: 123-45-6789, Address: 123 Main St
Jane Smith, SSN: 987-65-4321, Address: 456 Elm St

output.txt:

John Doe, SSN: CONFIDENTIAL, Address: 123 Main St
Jane Smith, SSN: CONFIDENTIAL, Address: 456 Elm St
Decrypted by child PID: 3456, Redacted by child PID:3457, Parent PID: 1234

Question 2: ShellDuel: Beat the System! 🏆

[20 marks]

Scenario:

Your Operating Systems professor has thrown down a challenge: "Can you build your own custom shell that executes real Linux commands?"

Develop "ShellDuel"—a lightweight but powerful custom shell that executes commands like a pro while also explaining what they do!

❖ How It Works:

- The shell continuously prompts the user for a command such as "cp ./OS ../newOS".
 - Focus on the following 4 commands (cp, mkdir, grep, ls)
- The command will be stored in a character array or a string object.
- The shell will perform tokenization and separate the command and its arguments.
- The shell will then create a child process and wait for the child to finish the execution.
- Then, the child will use the **execvp** system call to execute the command.
- After executing, it displays a short **explanatory message** about what the command does (make it generic).
- The shell **only stops when the user types "exit"**—until then, the duel continues!

❖ Example:

Enter Command: cp ./OS ../newOS

Executing: cp ./OS ../newOS

cp copies the file or directory from ./OS to ../newOS.

Enter Command: mkdir projectX

Executing: mkdir projectX

mkdir creates a new directory named projectX.

Enter Command: grep "error message" logs.txt

Executing: grep "error message" logs.txt

grep searches for the word "error message" in logs.txt.

Enter Command: exit

Goodbye, challenger! See you in the next duel.

Can you **prove you're the ultimate shell master?**

Question 3: The Ultimate Car Assembly Line Simulation 🚗

[40 Marks]

Scenario:

Welcome to **Turbo AutoWorks**, a cutting-edge car manufacturing plant! Unfortunately, the production line has been facing **critical failures**, and you, as an **Operating Systems Expert**, have been assigned to **optimize process management** to ensure smooth assembly.

The Four Stages of Production

1. Chassis Manufacturing (Main Stage - 2 Subtasks)

- **Subtask 1:** Welding the frame
- **Subtask 2:** Assembling the base structure
- Each subtask is handled by a separate child process.
- Both tasks must be completed successfully before moving forward.

2. Engine Installation (Main Stage - 3 Subtasks)

- **Subtask 1:** Mounting the engine
- **Subtask 2:** Connecting fuel and electrical systems
- **Subtask 3:** Testing engine functionality
- Each subtask is handled by a separate child process.
- All tasks must succeed for engine installation to be considered complete.

3. Painting (Main Stage - No Subtasks, 50% Failure Chance)

- A single process paints the car.
- Painting has a 50% failure rate.
- If Painting fails twice, the entire assembly line shuts down.

4. Final Inspection & Delivery (Main Stage - No Subtasks, 50% Failure Chance)

- A single process performs a quality check and prepares the car for delivery.
- Final Inspection has a 50% failure rate.
- If Inspection fails twice, the entire assembly line shuts down.

Program Requirements:

1. Process Creation (fork())

- i) The parent process creates a child process for each main stage.
- ii) Chassis Manufacturing and Engine Installation require additional child processes to handle subtasks.

2. Process Coordination (waitpid())



- i) The parent process waits for each stage to complete before moving forward.

3. Random Failures & Error Handling

- i) **Only 2 out of 4 main stages have a defined failure risk:**
 - ❖ **Chassis Manufacturing & Engine Installation:** No failure risk.
 - ❖ **Painting & Final Inspection:** 50% failure rate per attempt.
- ii) Each process gets TWO attempts before failure is considered critical.
- iii) If a main stage fails, the entire assembly line is shut down immediately.

Note: Use `srand()` to make the randomness more dynamic across runs. Visit [rand\(\) and srand\(\) in C++ - GeeksforGeeks](#) to learn more.

4. Logging & Status Reporting

- i) Every attempt must be **logged to the console**, e.g.:
 - ❖  *"Welding attempt 1: Success"*
 - ❖  *"Painting attempt 1: FAILED"*
- ii) If a **main stage fails**, print: **"ASSEMBLY LINE SHUT DOWN"** and terminate the program.
- iii) If all **stages succeed**, print: **"Car successfully manufactured!"**
- iv) **At the end, print an "Assembly Summary" including:**
 - ❖ Number of attempts per stage.
 - ❖ Final success or failure status.

5. Process Termination (`exit()`)

- i) Each process exits with 0 on success and a non-zero value on failure.
- ii) The parent process tracks exit codes to determine if a retry is needed or if production should stop.

Example Output:

```
Starting Car Assembly...
Welding attempt 1: Success
Assembling frame attempt 1: Success
Engine mounting attempt 1: Success
Electrical systems attempt 1: Success
Engine testing attempt 1: Success
Painting attempt 1: FAILED
Painting attempt 2: Success
Final Inspection attempt 1: FAILED
Final Inspection attempt 2: FAILED
Final Inspection failed! ASSEMBLY LINE SHUT DOWN.
```