System Programming Lab #8
Command-Line Parsing and Process Creation

Computer Science Department
Engineering School
Central Asian University

Spring 2025

Objective

The objective of this lab is to provide students with hands-on experience in:

- Parsing command-line arguments using `getopt()` and `getopt_long()` in C programs.
- Understanding and implementing robust command-line options for real-world applications.
- Creating and managing child processes using `fork()` and using system calls (e.g., `write()`) to record results.

Practice Tasks

Practice the two programs discussed during the lecture from the Linux Environment chapter:

- Writing your own function to get command-line options and arguments manually.
- Using Linux `getopt()` (and later `getopt_long()`) for enhanced option parsing. Experiment with changes to see the differences.

Tasks

Task 1: Handling Command-Line Arguments

Write a C program that:

1. Checks if exactly 2 arguments (including the program name) are passed.
2. Prints a usage message if the argument count is incorrect.
3. Processes each argument:
   - If an argument starts with `-`, print it as an option and process it using a function.
   - Otherwise, treat it as a regular argument and display it.
4. Implements the `process_option()` function to handle:
   - `-help`: Displays help information.
   - `-version`: Prints version details.
   - `-info`: Shows relevant information.

Task 2: Parsing Options with getopt()

Write a C program that:

1. Uses `getopt()` to parse command-line options.
2. Takes two options:
   - `-n` followed by an integer N.
   - `-m` followed by an integer M.
3. Computes:
   - The sum of the first N natural numbers (i.e., $1 + 2 + \ldots + N$).
   - The product of the first M natural numbers (i.e., $1 \times 2 \times \ldots \times M$).
4. Prints a usage message if the required flags are missing.

Task 3: File Encryption Decryption using getopt()

Write a **C program** that encrypts or decrypts a file using **XOR encryption**. Your program should:

1. Use `getopt()` to handle command-line options:
   - `-E <input_file>` to encrypt the file.
   - `-D <input_file>` to decrypt the file.
   - `-o <output_file>` to specify the output file.
   - `-k <key>` (Optional) to specify an encryption key; defaults to `'K'` if not provided.
2. Implement XOR-based encryption/decryption where:
   - Each character is XORed with the given key.
   - The same function is used for both encryption and decryption.
3. Ensure proper error handling:
   - Display an error message if incorrect options are provided.
   - Handle file errors gracefully.

Example Usage:

```
./xor_tool -E input.txt -o encrypted.txt -k X
./xor_tool -D encrypted.txt -o decrypted.txt -k X
```

Incorrect Usage:

```
./xor_tool -E input.txt -D encrypted.txt -o output.txt
```

Expected Output:

```
Error: Specify either -E for encryption or -D for decryption, not both.
```

Task 4: Using fork() for Parallel Execution

Write a C program that:

1. Uses `getopt()` to parse options:
   - `-n` followed by an integer N.
   - `-m` followed by an integer M.
   - `-f` followed by a filename to save results.
2. Creates two child processes using `fork()`:
   - One child computes the sum of N and M.
   - Another child computes the product of N and M.
3. Saves results to the specified file using the `write()` system call.
4. Displays a usage message if any required option is missing.