Course	COMP 8005
Program	Bachelor of Science in Applied Computer Science
Term	September 2025

- This is an individual programming assignment.

Objective

- Develop a multi-threaded password cracker for UNIX systems that can handle various password hashing algorithms.
- Analyze the impact of multi-threading on performance and evaluate the scalability of the implementation.

Learning Outcomes

- Implement multi-threading for computational tasks.
- Measure and analyze performance scaling with multiple threads.
- Understand the limitations of multi-threaded speedup.
- Work with different password hashing algorithms.
- Present findings using tables, graphs, and written explanations.

Assignment Details

- You will create a password cracker that supports the following UNIX password hashing algorithms:
    - yescrypt
    - bcrypt
    - SHA256
    - SHA512
    - MD5

- Your program must:
    - Parse the `/etc/shadow` file (or a copy of it) to get the algorithm, salt, and hash for the specified user.
    - Test cracking performance with 1, 2, 3, and 4 threads.
    - Test with a 3-character password.
    - Test with the 79 common characters given in the notes (06-legal-characters).
    - Test with N threads, where N is the number of CPU cores you have.
    - Predict the time required for N threads based on your results.

    - Run the test with N threads and compare the actual results to your prediction.
    - Explain why the prediction likely differed from reality.
    - Explain why using N threads is likely not N times faster.
    - Present your results using graphs/tables alongside written explanations.

- To find the number of CPU cores on Linux, run:

```
# logical cores
nproc

# physical cores
lscpu | awk '/^Core\(s\) per socket:/ {cores=$4}
/^Socket\(s\):/ {sockets=$2} END {print cores * sockets}'
```

- To find the number of CPU cores on macOS, run:

```
# logical cores
sysctl -n hw.ncpu

# physical cores
sysctl -n hw.physicalcpu
```

- If your physical and logical cores differ, then predict and run with both values.

Command Line

- Your program must be able to be run like;

```
cracker <shadow file> <username> <number of threads>
```

- The `shadow file` is either `/etc/shadow` or a copy of it
- The `username` is the name of the user whose password is to be cracked in the `shadow file`.
- The `number of threads` is the number of threads to create (1, 2, 3, 4, or N, where N is the logical or physical cores)

Requirements

- The program must be implemented in a language.
    - ***Note: If you are using Python, ensure you use at least version 3.13, which enables the removal of the Global Interpreter Lock (GIL).***
- The password cracker must support all five hashing algorithms.
- Include a structured report analyzing your results.

- Graphs and tables must be used to illustrate findings.
- Clear and well-documented code.

Constraints

- Ensure your implementation correctly utilizes multi-threading.
- Handle different hashing algorithms efficiently.
- Avoid unnecessary resource contention.
- Ensure correctness and avoid race conditions.

Resources

- UNIX crypt(3) documentation
- Libraries supporting password hashing and multi-threading in your chosen language
- System profiling tools (e.g., time, perf, htop)

Submission

- Ensure your submission meets all the guidelines, including formatting, file type, and submission.
- Follow the AI usage guidelines.
- Be aware of the late submission policy to avoid losing marks.
- ***Note: Please strictly adhere to the submission requirements to ensure you don't lose any marks.***

Evaluation

Topic	Value
Design	25
Testing	25
Implementation	30
Report	20
Total	100%

Hints

- Use appropriate synchronization mechanisms to avoid race conditions.

- Consider a method to avoid using locks (there is at least one way, but I will ***not*** provide any hints).
- If desired, consider CPU affinity and scheduling policies for enhanced performance tuning.