Course	COMP 80005
Program	Bachelor of Science in Applied Computer Science
Term	September 2025

- This is an individual programming assignment.

Objective

- Develop a distributed password-cracking system that builds upon your previous work with multi-threaded password-cracking.
- Introduce distributed programming concepts, focusing on fault tolerance, dynamic scalability, and work distribution across multiple worker nodes.

Learning Outcomes

- Implement a distributed computing system for password cracking.
- Design a fault-tolerant system that allows worker nodes to join or leave at any time.
- Use network programming to manage communication between a server and multiple clients.
- Implement dynamic work allocation.
- Handle partial progress reporting through configurable checkpointing.

Assignment Details

Requirements

Central Server

- The server is unaware of the nodes; nodes must register with the server.
- Connects to each node after it has registered.
- Dynamically distributes work and a checkpoint interval (N password guesses) to each node.
- Nodes can register or disappear/disconnect at any time.
- Tracks remaining work and reassigns unfinished tasks when nodes disappear/disconnect.
- Stops distributing work once the password is found, but allows nodes to complete their currently working passwords.
- Prints the cracked password to the console when found.

- Must display whenever a node is registered, disappears/disconnects, what work each node is allocated, and checkpoint information for each node.

Worker Nodes

- Registers with the server upon startup.
- Waits for the server to connect to it.
- Receives:
    - The hashed password to crack.
    - A range of passwords to attempt.
    - The checkpoint interval (N) from the server.
- Periodically send checkpoint updates every N attempts (sent from the server).
- Continue working until they receive a stop signal from the server.
- Gracefully handle disconnections (e.g., signal handlers).

Constraints

- You can use any language(s) and libraries that you like, as long as they do not take care of the distributed programming for you.

Command-line Parameters

- The server must accept the following configurable parameters:

Parameter	Description	Example
`--port <num>`	Port the server listens on	`--port 5000`
`--hash <hash>`	Hashed password to crack	`--hash <password hash from mkpasswd>`
`--work-size <num>`	Number of passwords assigned per node request	`--work-size 1000`
`--checkpoint <num>`	Number of attempts before a node sends a checkpoint	`--checkpoint 500`
`--timeout <num>`	Number of seconds to wait for a checkpoint from a client	`--timeout 600`

- The worker nodes must accept the following parameters:

Parameter	Description	Example
`--server <ip>`	IP address of the server	`--server 192.168.1.10`
`--port <num>`	Port to connect to	`--port 5000`

`--threads <num>`	Number of threads to use for cracking	`--threads 4`

Failure Handling & Dynamic Scaling

- Nodes may disconnect at any time (e.g., due to a crash, network failure, or being killed).
- The server must detect failures and redistribute unfinished work.
- Nodes should be able to join/leave at any time and receive work.

Communication Protocol

- The specific message format is up to you (e.g., JSON, plain text, binary).
- The protocol must support:
    - New node connections
    - Work requests
    - Checkpoint updates
    - Node disconnections (this can be by inferring a crash or by an optional disconnection message - optional)
    - Cancel found notifications
    - Password found notifications

Testing/Report

- Be sure to test with 1, 2, 3, 4, and 10 nodes.
- Test with 1, 2, 3, and 4 nodes first, then predict the speed of 10 nodes.
- Test with 10 nodes and document how your prediction was caused and what caused it to be off (most likely).
- Ensure that you use graphs for all of the above tests.

Resources

- UNIX socket programming tutorials
- Distributed computing references
- Password hashing and cracking theory

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
Correctness	30
Dynamically add/remove nodes	20
Design	20
Testing	30
Total	100%

Hints

- The server uses thread or async programming to handle multiple connections efficiently.
- Use efficient data structures to track work distribution and node status.
- Keep communication minimal to avoid unnecessary overhead.
- Test with different node counts to ensure stability under load.
- Gracefully handle node failures by detecting disconnections and redistributing work.