CS 440 – Project 2
Measuring Thread Creation and Destruction Overhead

Spring 2026
Due Monday, Feb 23, 2026 @11:59 PM
Points: 50 (+ up to 25 extra credit)
Teams: Individual or pairs (maximum 2 students)

Overview

In this project, you will experimentally measure the overhead of thread creation and destruction under different structural patterns. You will compare flat thread creation with two-level and three-level hierarchical thread creation.

Global Requirements

• Each experiment (A., B., C.) must create and destroy exactly 5,000 threads.
• Record start time before the first thread is created.
• Record end time after the final thread is destroyed.
• Run each experiment three times and compute the average.
• Print final counts for threads created and destroyed.

Experiment A. – Flat Threads

Create 5,000 threads with no hierarchy. Destroy (join) all threads after creation and measure total execution time.

Experiment B. – Two-Level Hierarchy

Create 50 parent threads. Each parent creates 99 child threads and waits for all children to terminate before terminating itself.

Thread count check:

- Parents: 50
- Children: 50 × 99 = 4,950
- **Total:** 5,000 threads

When printing identifiers, use lineage format:

parentID-childID

Experiment C. – Three-Level Hierarchy

Create 20 initial threads. Each initial thread creates 3 child threads, and each child thread creates 82 grandchild threads. All threads must wait for their descendants before terminating.

Thread count check:

- Initial: 20
- Children: 20 × 3 = 60
- Grandchildren: 20 × 3 × 82 = 4,920
- **Total:** 5,000 threads

When printing identifiers, use lineage format:

initialID-childID-grandchildID

Batching (If You Run Out of Memory)

If your program fails due to system resource limits, you must reduce concurrency using batching while still creating and destroying exactly 5,000 threads. A method for performing this is provided in each template. Document the error encountered and the batch sizes used.

Batching Strategy (Required if Limits Are Hit)

Batching means creating threads in small groups, waiting for them to finish, then continuing.

Examples:
A.: create 25 threads → join → repeat until 5,000 are created
B.: each parent creates children in batches (e.g., 25 at a time)
C.: each child creates grandchildren in batches

If batching is used, you must document:

- What error or limit you encountered
- The batch size(s) used
- Confirmation that 5,000 threads were still created and destroyed

Sample Output

Sample Output — A. Flat Thread Creation (5,000 threads)

```
=== A. Flat (UNBATCHED) ===
N_TOTAL: 5000
Output grouping: 25 threads

Start time: 1706908423123456789 ns  (2026-02-01 13:05:12.123456 -0700)

Created threads:   1-25
Created threads:   26-50
Created threads:   51-75
...
Created threads:   4976-5000

Joined threads:    5000-4976
Joined threads:    4975–4951
Joined threads:    4950–4926
...
Joined threads:    25–1

End time:   1706908423989123456 ns  (2026-02-01 13:05:12.989123 -0700)
Elapsed:    866.566 ms

Threads created:   5000
Threads destroyed: 5000
```

Sample Output — B. Two-Level Hierarchy (Parent → Child)

```
=== B. Two-level (UNBATCHED) ===
Parents: 50
Children per parent: 99
Output grouping: 25 threads
Total threads: 5000

Start time: 1706908430123987123 ns  (2026-02-01 13:05:13.123987 -0700)

Parent 1 started
Parent 1 created children: 1-1 … 1-25
Parent 1 created children: 1-26 … 1-50
Parent 1 created children: 1-51 … 1-75
Parent 1 created children: 1-76 … 1-99
Parent 1 joined   children: 1-99 … 1-1
Parent 1 completed

...

Parent 25 started
Parent 25 created children: 25-1 … 25-25
Parent 25 created children: 25-26 … 25-50
Parent 25 created children: 25-51 … 25-75
Parent 25 created children: 25-76 … 25-99
Parent 25 joined   children: 25-99 … 25-1
Parent 25 completed

...

Parent 50 completed

End time:   1706908431298845123 ns  (2026-02-01 13:05:14.298845 -0700)
Elapsed:    1174.858 ms

Threads created:   5000
Threads destroyed: 5000
```

Sample Output — C. Three-Level Hierarchy (Grandparent → Parent → Child)

```
=== C. Three-level (UNBATCHED) ===
Initial threads: 20
Children per initial: 3
Grandchildren per child: 82
Output grouping: 25 threads
Total threads: 5000

Start time: 1706908440039182736 ns  (2026-02-01 13:05:15.039182 -0700)

Initial 1 started
Initial 1 created child: 1-1

Child 1-1 created grandchildren: 1-1-1 … 1-1-25
Child 1-1 created grandchildren: 1-1-26 … 1-1-50
Child 1-1 created grandchildren: 1-1-51 … 1-1-75
Child 1-1 created grandchildren: 1-1-76 … 1-1-82
Child 1-1 joined   grandchildren: 1-1-82 … 1-1-1
Child 1-1 completed

Initial 1 created child: 1-2
...
Initial 1 completed

...

Initial 20 completed

End time:   1706908441824719023 ns  (2026-02-01 13:05:16.824719 -0700)
Elapsed:    1785.533 ms

Threads created:   5000
Threads destroyed: 5000
```

Sample Output — A. Flat (BATCHED, if needed)

```
=== A. Flat (BATCHED) ===
N_TOTAL: 5000
Batch size (internal): 10
Output grouping: 25 threads

Start time: 1706908423123456789 ns  (2026-02-01 13:05:12.123456 -0700)

Created threads:   1–25
Joined threads:    25–1
```

```
Created threads:   26–50
Joined threads:    50–26

Created threads:   51–75
Joined threads:    75–51

...
Created threads:   4976–5000
Joined threads:    5000–4976

End time:   1706908423989123456 ns  (2026-02-01 13:05:12.989123 -0700)
Elapsed:    866.566 ms

Threads created:   5000
Threads destroyed: 5000
```

Sample Output — B. Flat (BATCHED Children, if needed)

```
===  Two-level (BATCHED children) ===
Parents: 50
Children per parent: 99
Child batch size (internal): 10
Output grouping: 25 threads
Total threads: 5000

Start time: 1706908430123987123 ns  (2026-02-01 13:05:13.123987 -0700)

Parent 1 started
Parent 1 created children: 1-1 … 1-25
Parent 1 joined   children: 1-25 … 1-1

Parent 1 created children: 1-26 … 1-50
Parent 1 joined   children: 1-50 … 1-26

Parent 1 created children: 1-51 … 1-75
Parent 1 joined   children: 1-75 … 1-51

Parent 1 created children: 1-76 … 1-99
Parent 1 joined   children: 1-99 … 1-76
Parent 1 completed

...

Parent 25 started
Parent 25 created children: 25-1 … 25-25
Parent 25 joined   children: 25-25 … 25-1
...
Parent 25 completed
```

```
...

Parent 50 completed

End time:   1706908431298845123 ns  (2026-02-01 13:05:14.298845 -0700)
Elapsed:    1174.858 ms

Threads created:   5000
Threads destroyed: 5000
```

Deliverables

Source Code

- Complete, runnable source code
- Build/run instructions
- Any language with real OS threads (Java, Python, C#, C, C++, etc.) – use templates provided or go completely rogue with your own code.

Program Output

- Submit console output showing:
- Start time, end time, elapsed time
- Threads created/destroyed
- Three trial results for A., B., and C.

Written Report

Your report must include:

- System Information
- OS
- CPU
- RAM
- Language/runtime version
- Methodology
- How threads were created and joined
- Whether batching was used

Results Table

- Three trials + average for each experiment
- Analysis (1–2 pages)
- Compare A. vs B. vs C.
- Discuss why times differ despite identical thread counts
- Discuss scheduling and non-determinism
- Comment on the effect of hierarchy
- Batching Discussion (if applicable)
    - What failed
    - What you changed
    - Why batching solved the problem

Short Video Demonstration

- Submit a short (2-5 minute) narrated screen recording that:

- Runs A., B., and C.
- Shows final timing and counters
- Demonstrates batching (if used)

Extra Credit

EC1: Thread Pool / Executor Comparison (+10)

- Implement and demonstrate a version using a **fixed-size thread pool** instead of creating a new OS thread per task.
- Compare performance against your baseline experiments.

EC2: Additional Language Beyond Java/Python/C# (+5)

- Implement the project in a language **other than Java, Python, or C#**. Examples:
    - **C**
    - **C++**
    - Rust
    - Go
- Include build instructions and a brief comparison of results.

EC3: POSIX pthreads (+10)

- Implement using **POSIX pthreads** (C + pthread).
- Compare results against a managed language (Java/C#) or Python.

Grading Rubric (50 Points)

Correctness & Requirements (20 pts)

- Correct thread counts (5,000 created/destroyed for A/B/C): **10 pts**
- Correct hierarchy structure and joins: **10 pts**

Timing & Output (10 pts)

- Proper timing boundaries (start/end): **5 pts**
- Clear, correct printed results: **5 pts**

Analysis & Write-Up (15 pts)

- Clear methodology and environment description: **5 pts**
- Results table with averages: **5 pts**

- Thoughtful comparison and discussion: **5 pts**

Code Quality & Video (5 pts)

- Code clarity and organization: **3 pts**
- Clear video demonstration: **2 pts**

Extra Credit (up to +25 pts)

- EC1 Thread pool comparison: +10
- EC2 Additional language (not Java/Python/C#): +5
- EC3 POSIX pthreads: +10

Key Takeaway

This project is not about speeding things up; it is about understanding overhead, structure, and how the OS schedules threads. Your analysis matters just as much as your code.