Recitation 4: Threads

Department of Computer Science and Engineering
University of Minnesota

June 30, 2025

Overview

- Threads
- POSIX Thread API
- Basic Synchronization
- Exercise 1: Matrix addition
- **Please check the man pages for more information on any APIs discussion.**

Threads

- Lightweight execution units within a process.
- A thread has its own id, stack, program counter and a set of registers.
- Threads spawned within a process share data (global, static), code and heap.
- Threads communicate using shared memory.

POSIX Thread APIs

- A set of standard APIs for a thread lifecycle management.
- Thread creation, join, detach, exit and cancel to name a few.
- During code compilation use `-pthread` flag

```
#include <pthread.h>
pthread_t pthread_self(void);
// Returns id of calling thread
```

Thread Creation

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start_routine) (void *), void *arg);
// Creates a new thread that executes start_routine with argument arg
// thread: Uniquely identifies thread
// attr: Defines thread behavior, set to NULL unless default configs
         should change
// start_routine: Function pointer of routine to be executed
// arg: start_routine can accept only one argument, for multiple
        arguments use a structure
// Returns 0 on success, otherwise an error number
```

Thread join(samples/p1.c, p2.c)

```
int pthread_join(pthread_t thread, void **retval);
// Blocking call that waits for a thread to complete start_routine
// thread: Uniquely identifies thread (1st argument of pthread_create)
// retval: Return value from start_routine
// Returns 0 on success, otherwise an error number
```

Thread detach, exit and cancel(samples/p3.c)

```
int pthread_detach(pthread_t thread);
// Alternative to pthread_join, thread will release resources on
// start_routine completion. There is no blocking call in the code.
void pthread_exit(void *retval);
// Terminates the calling thread and returns a value via retval.
int pthread_cancel(pthread_t thread);
// Sends cancellation request to a thread
// detach and cancel returns 0 on success, otherwise an error number
```

Basic Synchronization

- Threads could share global/static variables. What happens when multiple threads access the same variable in parallel? **Race conditions**.
- Sections of code that are shared and modified by threads: **critical section**.
- We could use a variable with busy waiting for synchronization, however, it is highly inefficient, wasting a lot of CPU cycles.

Exercise

- Given two matrices A and B of size m*m. Calculate C = A + B, where the addition is split across **multiple threads**.
- **matrix.c** will take two arguments: value of **m** and the number of threads, **t**.
- You should split the matrices column-wise across **t** threads. Use the Arguments structure to pass the start and end indices of each thread along with other values (already populated).
- **m** may not be divisible by **t**. So the last thread may have more number of columns to handle.
- You could check whether your matrix is correct by writing a sequential code to perform C = A + B and the compare the values with parallel code.

Deliverables

- Submit the completed exercises folder as a **zip** to Gradescope by **July 6th, 11:59 pm**.
- The sample codes should have all the necessary information for completing all the above exercises. If you have any questions, please post it to Piazza.
- Autograder will not be used as the output from each student will be different depending on the reporting style. Also, the order of output is not predictable as the OS schedules the processes/threads spawned. Manual grading will be employed.