CSE 124
Week 2: Discussion Session
Jan 18, Friday
Outline

• Concurrency

• Threads to achieve concurrency in C++
What is Concurrency?

• Ability of an algorithm/program to run more than one task at a time.

• Different jobs can execute different parts that can run independently of each other at once.

• A Concurrent Web Server is able to handle multiple client requests at the same time.

• It’s different from parallelism (more hardware to actually run multiple tasks at the same time)
How to make web server concurrent?

- **Multi-threaded approach**: a new thread for each incoming connection
- **Multi-process approach**: will fork() a new process to handle an incoming web request
- **Event-driven concurrency**: will keep a list of active connections and loop over them, performing a little bit of work on behalf of each connection.
We’ll use threads!
What is a thread?

The smallest sequence of programmed instructions that can be managed independently by an OS scheduler

- When starting a program, a process is created.
- Every process has a unique set of virtual memory, files, code and resources.
- A process consists of one or more threads that run the code.
- Every thread in a process accesses the same memory while running.
- When changing the value of a global variable in one thread, all threads then see the new value.
Creating a thread earlier

Earlier we used POSIX threads.

```c
#include <pthread.h>
pthread_create (thread,
               attr,
               start_routine,
               arg)
```

Not anymore!
Now with C++11

- Use ‘thread’ library.
  `#include<thread>`

- ‘join’ threads
  Thread becomes non joinable after a call to this function and can be destroyed safely.
  Caller is blocked till the thread completes execution.

  Used to wait on the completion of threads
Example #1

```cpp
#include <iostream>
#include <thread>
using namespace std;

void print_num(int x)
{
    for (int i = 0; i < 1000; i++) {
        cout << x;
    }
}

int main()
{
    /* Spawning threads t1 and t2, both calling the same function but with different arguments. */
    thread t1(print_num, 0);
    thread t2(print_num, 1);
    cout << "Both threads are now executing concurrently...\n";

    /* Synchronizing the threads. 'join' makes sure that we wait for the threads to complete execution */
    t1.join();
    t2.join();
    cout << "\nThreads have been synchronized and stopped running.\n";
    return 0;
}
```
Notice that both the functions are being run simultaneously.
```cpp
#include <iostream>
#include <thread>
using namespace std;

void print_num(int x)
{
    for (int i = 0; i < 1000; i++) {
        cout << x;
    }
}

void multiply(int x, int y)
{
    for (int i = 0; i < 1000; i++) {
        cout << x * y;
    }
}

int main()
{
    /* Spawning threads t1 and t2, both calling the same function but with different arguments. */
    thread t1(print_num, 0);
    thread t2(multiply, 1, 3);
    cout << "Both threads are now executing concurrently...
";

    /* Synchronizing the threads. 'join' make sures that we wait for the threads to complete execution */
    t1.join();
    t2.join();

    cout << "
Threads have been synchronized and stopped running."
; return 0;
}
Both threads are now executing concurrently...

Threads have been synchronized and stopped running.
References

- https://www.geeksforgeeks.org/multithreading-in-cpp/
Questions?