15-440
Distributed Systems
Recitation 6

Ammar Karkour

Slides Adopted From:

Laila Elbeheiry

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Logistics

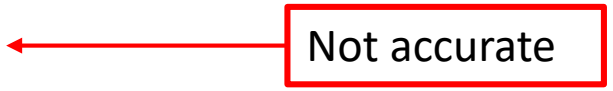
- Quiz 2 Graded (Average: 16, Stdev: 2.3, Max: 19)
- P1 Due Next Sunday
- PS3 Released (Due next Thursday)

In this Recitation..

- Study concurrent programming
  - Using Java as a language
  - Using an abstract shared memory model
- In a future lecture
  - Use C/C++ primitives (MPI)
  - Using a distributed memory machine

What is concurrency?

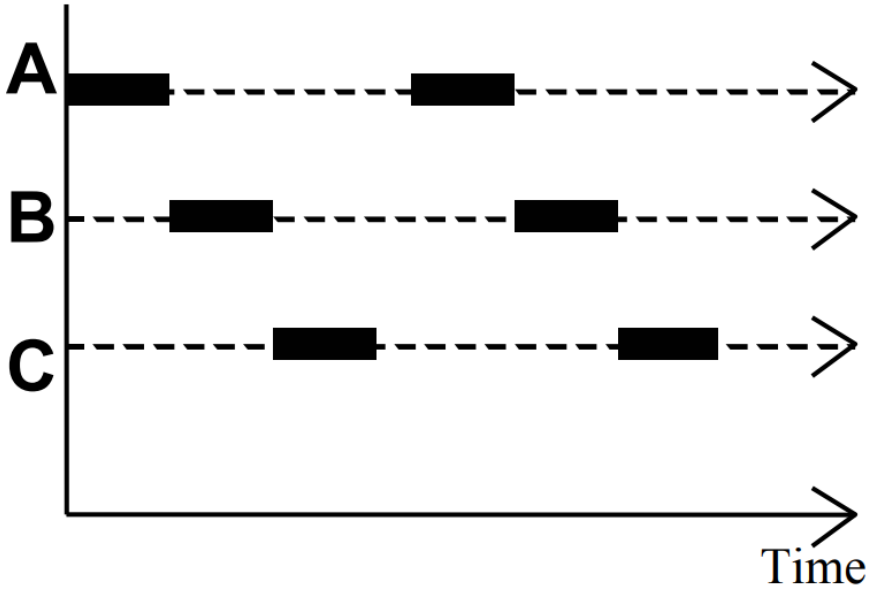
- Sequential Programs
  - Single thread of control
  - Executes one instruction at a time
  - (- pipelining + SIMD)
- Concurrent Programs
  - Multiple autonomous sequential threads, executing (logically) in parallel
- The implementation (i.e. execution) of the threads can be:
  - Multiprogramming – Threads multiplex their executions on a single processor.
  - Multiprocessing – Threads multiplex their executions on a multiprocessor or a system
  - Distributed Processing – Processes multiplex their executions on several different machines

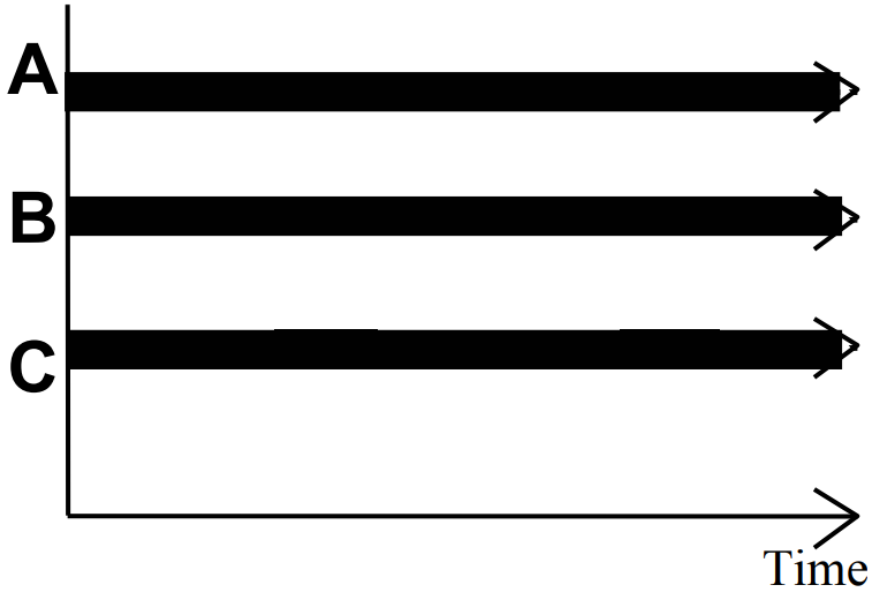


Concurrency and Parallelism

- Concurrency doesn't imply parallelism

Why?





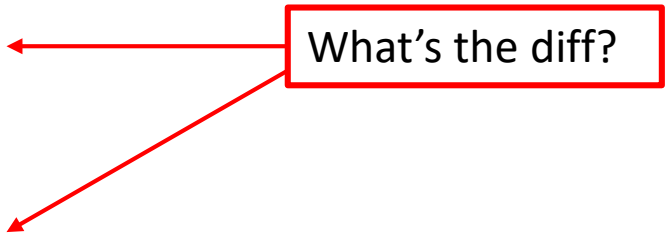
Concurrency in Java

Bank use case

Concurrency in Java

- Java has a predefined class `java.lang.Thread`

```
public class MyThread extends Thread {
    public void run() {
    }
}
```



- Java also provides a standard interface

```
public interface Runnable {
    public void run();
}
```

- Any class which wishes to express concurrent execution must implement this interface and the `run` method
- Threads do not begin their execution until the `start` method in the `Thread` class is called

Activity Trace 1 of ATMs

Account ID > Ammar

Password > 1234

your account balance is 200

Deposit or withdraw amount > -150

your balance is 50

Account ID > Sana

Password > 0000

your account balance is 250

Deposit or withdraw amount > -50

your balance is 200

Activity Trace 2 of ATMs

Time

Account ID > Ammar

Password > 1234

Your account balance is 200

Deposit or withdraw amount > -150

your balance is 50

200 – 150 – 150 = 50!!!

Account ID > Ammar

Password > 1234

Your account balance is 200

Deposit or withdraw amount > -150

your balance is 50

Synchronization

- Threads can be arbitrarily interleaved
- Some interleavings are NOT correct
- Java provides synchronization mechanism to restrict the interleavings
- Synchronization serves two purposes:
  - Ensure safety for shared updates – Avoid race conditions
  - Coordinate actions of threads – Parallel computation – Event notification

Safety of Concurrent Execution

- Multiple threads access shared resource simultaneously
  - Safe only if:
    - All accesses have no effect on resource, – e.g., reading a variable
    - All accesses are atomic
    - Only one access at a time: mutual exclusion

Mutual Exclusion

- Prevent more than one thread from accessing critical section at a given time
- Once a thread is in the critical section, no other thread can enter that critical section until the first thread has left the critical section.
- No interleavings of threads within the critical section
- Serializes access to section

```
synchronized int getbal() { return balance; }
synchronized void post(int v) { balance = balance + v; }
```

Activity Trace 2 of ATMs Zoom in

Time

```
int val = in.readLine();

if (acc.getbal() + val > 0)

        post(val);



out.println("your balance is " +
        acc.getbal());
```

your balance is 50

```
int val = in.readLine();

if (acc.getbal() + val > 0)

        post(val);

out.println("your balance is " +
        acc.getbal());
```

your balance is 50

Negative Bank Balance!

Atomicity

- Synchronized methods execute the body as an atomic unit
- May need to execute a code region as the atomic unit
- Block Synchronization is a mechanism where a region of code can be labeled as synchronized
- The `synchronized` keyword takes as a parameter an object whose lock the system needs to obtain before it can continue

```
synchronized (acc) {
    if (acc.getbal() + val > 0)
        acc.post(val);
    else
        throw new Exception();
    out.print("your balance is " + acc.getbal());
}
```

Activity Trace 2 of ATMs Zoom in

Time

```
out.println("your balance is " + acc.getbal());
```
your balance is 200

```
out.print("Deposit or withdraw amount > ");
```
Deposit or withdraw amount > -150

```
int val = in.readLine();

synchronized(acc)
if (acc.getbal() + val > 0) post(val);
out.println("your balance is " +
            acc.getbal());
```
your balance is 50

```
out.println("your balance is " + acc.getbal());
```
your balance is 200

```
out.print("Deposit or withdraw amount > ");
```
Deposit or withdraw amount > -150

```
int val = in.readLine();

synchronized(acc)



if (acc.getbal() + val > 0)
throw new Exception();
```

Balance shows 200 but couldn't withdraw!!

Activity Trace 2 of ATMs Zoom in

Time

Account ID > Ammar

Password > 1234

synchronized(acc)

out.println("your balance is " + acc.getbal());

your balance is 200

Deposit or withdraw amount >

Account ID > Ammar

Password > 1234

synchronized(acc)

NO RESPONSE!!!

Account Transfer Execution Trace

Sana wants to transfer 10 riyals to Abdalla
Abdalla wants to transfer 20 riyals to Sana
Will our code always work?

Time

Sana -> Abdalla

```
synchronized(from) {
if (from.getbal() > val)
from.post(-val);
```

```
synchronized(to)
```

DEADLOCKED!!!!

Abdalla -> Sana

```
synchronized(from) {
if (from.getbal() > val)
from.post(-val);
```

```
synchronized(to)
```

How to fix?

Avoiding deadlocks

- Cycle in locking graph = deadlock
- Standard solution: canonical order for locks
  - Acquire in increasing order
  - Release in decreasing order
- Ensures deadlock-freedom, but not always easy to do

Other types of synchronization in Java

- Semaphores
- Blocking & non-blocking queues
- Concurrent hash maps
- Copy-on-write arrays
- Exchangers
- Barriers
- Futures
- Thread pool support

Potential Concurrency Problems

- Deadlock
  - Two or more threads stop and wait for each other
- Livelock
  - Two or more threads continue to execute, but make no progress toward the ultimate goal.
- Starvation
  - Some thread gets deferred forever.
- Lack of fairness
  - Each thread gets a turn to make progress.
- Race Condition
  - Some possible interleaving of threads results in an undesired computation result

Interesting Ongoing Research on Concurrency

- Automatic parallelizers (e.g. Parsynt)
- Verification of concurrent programs (e.g. Duet)
- Concurrent program testing (e.g. Penelope)
- PL approached to deadlock freedom

Conclusion

- Concurrency and Parallelism are important concepts in Computer Science
- It can be very hard to understand and debug concurrent programs
- Parallelism is critical for high performance
  - From Supercomputers in national labs to Multicores and GPUs on your desktop
- Concurrency is the basis for writing parallel programs
- Next Recitation: Project 2

Credits

- The bank use case code and some slides are taken from 6.189 IAP 2007 MIT concurrent programming lecture