



THREADS & CONCURRENCY

Lecture 23- CS2110 – Fall 2018

CPU Central Processing Unit. Simplified view

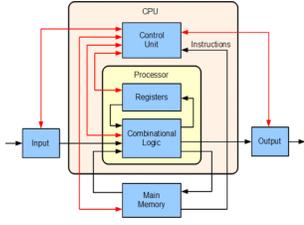
The CPU is the part of the computer that executes instructions.

Java: `x = x + 2;`

Suppose variable x is at Memory location 800, Instructions at 10

Machine language:

- 10: load register 1, 800
- 11: Add register 1, 2
- 12: Store register 1, 800



Basic uniprocessor-CPU computer. Black lines indicate data flow, red lines indicate control flow
From wikipedia

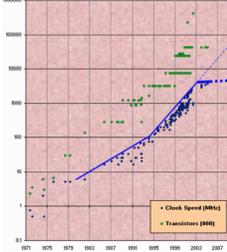
Part of Activity Monitor in Gries's laptop

>100 processes are competing for time. Here's some of them:

Process Name	% CPU	CPU Time	Threads
Grab	4.1	3.33	7
ReportCrash	2.3	0.78	6
Eclipse	1.5	1:48:30.07	54
SuperTab	1.4	1:40:44.59	5
Activity Monitor	1.4	10.57	10
https://www.wunderground.c...	1.1	1:34.19	23
Creative Cloud	0.8	58:32.81	27
Microsoft PowerPoint	0.6	3:24.02	9
Safari Networking	0.4	26:53.25	10
loginwindow	0.3	16:14.79	4
Google Drive	0.3	6.33	22
Safari	0.3	50:09.48	24

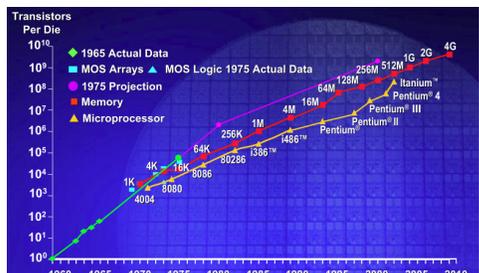
Clock rate

- Clock rate “frequency at which CPU is running”
- Higher the clock rate, the faster instructions are executed.
- First CPUs: 5-10 Hz (cycles per second)
- Today MacBook Pro 3.5GHz
- Your OS can control the clock rate, slow it down when idle, speed up when more work to do



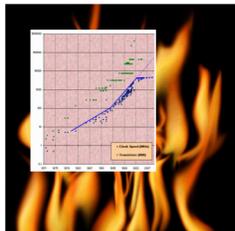
Why multicore?

- Moore's Law: Computer speeds and memory densities nearly double each year



But a fast computer runs hot

- Power dissipation rises as square of the clock rate
- Chips were heading toward melting down!
- Put more CPUs on a chip: with four CPUs on one chip, even if we run each at half speed we can perform more overall computations!



Today: Not one CPU but many

7

Processing Unit is called a **core**.

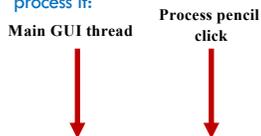
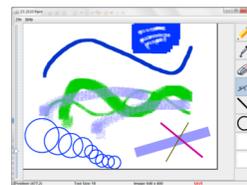
- Modern computers have “multiple cores” (processing units)
 - Instead of a single CPU (central processing unit) on the chip 5-10 common. Intel has prototypes with 80!
- We often run many programs at the same time
- Even with a single core (processing unit), your program may have more than one thing “to do” at a time
 - Argues for having a way to do many things at once

Many programs. Each can have several “threads of execution”

8

We often run many programs at the same time
And each program may have several “threads of execution”

Example, in a Paint program, when you click the pencil tool, a new thread of execution is started to call the method to process it:



Programming a Cluster...

9

- Sometimes you want to write a program that is executed on many machines!
- Atlas Cluster (at Cornell):
 - 768 cores
 - 1536 GB RAM
 - 24 TB Storage
 - 96 NICs (Network Interface Controller)



Many processes are executed simultaneously on your computer

10

- Operating system provides support for multiple “processes”
- Usually fewer processors than processes
- Processes are an abstraction: at hardware level, lots of multitasking
 - memory subsystem
 - video controller
 - buses
 - instruction prefetching

Concurrency

11

- Concurrency refers to a single program in which several processes, called threads, are running simultaneously
 - Special problems arise
 - They reference the same data and can interfere with each other, e.g. one process modifies a complex structure like a heap while another is trying to read it
- CS2110: we focus on two main issues:
 - Race conditions
 - Deadlock

Race conditions



12

- A “race condition” arises if two or more processes access the same variables or objects concurrently and at least one does updates
- Example: Processes t1 and t2 $x = x + 1;$ for some static global x.

Process t1	Process t2
...	...
$x = x + 1;$	$x = x + 1;$

But $x = x + 1;$ is not an “atomic action”: it takes several steps

Race conditions

13

- Suppose x is initially 5

Thread t1	Thread t2
LOAD x	...
ADD 1	LOAD x
STORE x	ADD 1
	STORE x

- ... after finishing, x = 6! We "lost" an update

Race conditions

14

- Typical race condition: two processes wanting to change a stack at the same time. Or make conflicting changes to a database at the same time.
- Race conditions are bad news
 - Race conditions can cause many kinds of bugs, not just the example we see here!
 - Common cause for "blue screens": null pointer exceptions, damaged data structures
 - Concurrency makes proving programs correct much harder!

Deadlock

15



- To prevent race conditions, one often requires a process to "acquire" resources before accessing them, and only one process can "acquire" a given resource at a time.
- Examples of resources are:
 - A file to be read
 - An object that maintains a stack, a linked list, a hash table, etc.
- But if processes have to acquire two or more resources at the same time in order to do their work, **deadlock** can occur. This is the subject of the next slides.

Dining philosopher problem

16



Five philosophers sitting at a table.

Each repeatedly does this:

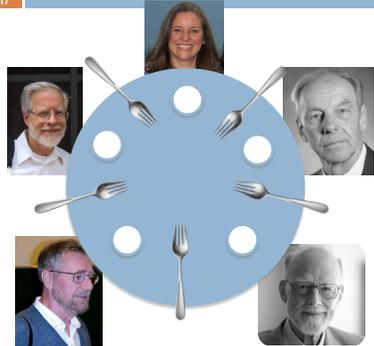
- think
- eat

What do they eat? spaghetti.

Need TWO forks to eat spaghetti!

Dining philosopher problem

17



Each does repeatedly :

- think
- eat (2 forks)

eat is then:

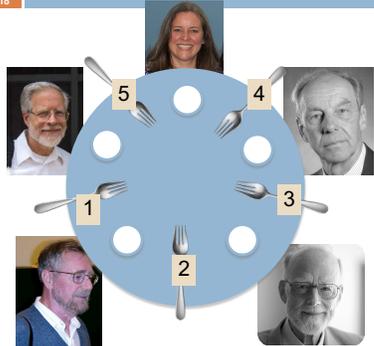
- pick up left fork
- pick up right fork
- pick up food, eat
- put down left fork
- put down right fork

At one point, they all pick up their left forks

DEADLOCK!

Dining philosopher problem

18



Simple solution to deadlock:

Number the forks. Pick up smaller one first

- think
- eat (2 forks)

eat is then:

- pick up smaller fork
- pick up bigger fork
- pick up food, eat
- put down bigger fork
- put down smaller fork

Java: What is a Thread?

- 19 20
- A separate “execution” that runs within a single program and can perform a computational task independently and concurrently with other threads
- Many applications do their work in just a single thread: the one that called main() at startup
 - But there may still be extra threads...
 - ... Garbage collection runs in a “background” thread
 - GUIs have a separate thread that listens for events and “dispatches” calls to methods to process them
- Today: learn to create new threads of our own in Java

Thread

- A thread is an object that “independently computes”
 - Needs to be created, like any object
 - Then “started” --causes some method to be called. It runs side by side with other threads in the same program; they see the same global data
- The actual executions could occur on different CPU cores, but don't have to
 - We can also simulate threads by *multiplexing* a smaller number of cores over a larger number of threads

Java class Thread

- 21 22
- threads are instances of class Thread
 - Can create many, but they do consume space & time
- The Java Virtual Machine creates the thread that executes your main method.
- Threads have a priority
 - Higher priority threads are executed preferentially
 - By default, newly created threads have initial priority equal to the thread that created it (but priority can be changed)

Creating a new Thread (Method 1)

```
class PrimeThread extends Thread {
    long a, b;

    PrimeThread(long a, long b) {
        this.a = a; this.b = b;
    }

    public void run() {
        //compute primes between a and b
        ...
    }
}

PrimeThread p = new PrimeThread(143, 195);
p.start();
```

overrides Thread.run()

Call run() directly? no new thread is used: Calling thread will run it

Do this and Java invokes run() in new thread

Creating a new Thread (Method 2)

```
23 24
```

```
class PrimeRun implements Runnable {
    long a, b;

    PrimeRun(long a, long b) {
        this.a = a; this.b = b;
    }

    public void run() {
        //compute primes between a and b
        ...
    }
}

PrimeRun p = new PrimeRun(143, 195);
new Thread(p).start();
```

Example

Thread name, priority, thread group

```
public class ThreadTest extends Thread {
    public static void main(String[] args) {
        new ThreadTest().start();
        for (int i = 0; i < 10; i++) {
            System.out.format("%s %d\n",
                Thread.currentThread(), i);
        }
    }

    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.format("%s %d\n",
                Thread.currentThread(), i);
        }
    }
}

Thread[Thread-0,5,main] 0
Thread[main,5,main] 0
Thread[main,5,main] 1
Thread[main,5,main] 2
Thread[main,5,main] 3
Thread[main,5,main] 4
Thread[main,5,main] 5
Thread[main,5,main] 6
Thread[main,5,main] 7
Thread[main,5,main] 8
Thread[main,5,main] 9
Thread[Thread-0,5,main] 1
Thread[Thread-0,5,main] 2
Thread[Thread-0,5,main] 3
Thread[Thread-0,5,main] 4
Thread[Thread-0,5,main] 5
Thread[Thread-0,5,main] 6
Thread[Thread-0,5,main] 7
Thread[Thread-0,5,main] 8
Thread[Thread-0,5,main] 9
```


Fancier forms of locking

31

- Java developers have created various synchronization abstract data types
 - Semaphores: a kind of synchronized counter (invented by Dijkstra)
 - Event-driven synchronization
- The Windows and Linux and Apple O/S have kernel locking features, like file locking
- But for Java, **synchronized** is the core mechanism

Summary

32

- Use of multiple processes and multiple threads within each process can exploit concurrency
 - Which may be real (multicore) or “virtual” (an illusion)
- When using threads, beware!
 - Synchronize any shared memory to avoid race conditions
 - Synchronize objects in certain order to avoid deadlocks
 - Even with proper synchronization, concurrent programs can have other problems such as “livelock”
- Serious treatment of concurrency is a complex topic (covered in more detail in cs3410 and cs4410)