Topic 12
Concurrency
CH 13 Scott
skip: 13.1.2, 13.2.3 (except fork/join and Box 13.3) 13.3 except 13.3.5, 13.4.1, 13.4.2, 13.4.4, 13.4.5

Why?
- Problem has a logically parallel structure
  - characters in a video game
- Exploit machine capabilities
  - SPEED!
- Physical distribution
  - the internet, some things arrive very slowly

Parallel and the web
- see firefox developer tools / network (rightmost column)
- imagine if all of the images were retrieved serially

Concurrent == 2 or more task may be active at any time (not that active does not imply actually doing anything. So you can have concurrent processing of threads on a single processor machine.

Parallel == more than one task can be running. So all parallel are concurrent. Ti get parallel need more than one processor.

Parallel Level

Instruction level — in the core recognizing that instructions take different route through memory (no need to anything at the application level.)

Vector — operations repeatedly on every element of a dataset (GPUs) The classic example of this form is weather forecasting … and various graphics operations. GPUs are essentially vector parallel devices.

Thread — do different things on different data through (possibly) different processors

“race condition” Consider the code below

//Thread One	// Thread 2
do something that takes a while	do something that takes a while
set a=1	set a=2

Assume that a is a common object share by the threads. Then, these two threads are in a race in that if thread 1 finishes first then the value of a will be 1 for a while then 2. OTOH, if thread 2 finishes first …

So in this case need a way of “synchronize” the threads in order to get a deterministic result. One of the principle things that a PL offers for parallel programming is synchronization tools.

Parallelism and Event-driven programming
Instead of parallel programming could achieve some of the same effect through a "dispatch loop" And this is basically what you have to do when using Javascript (and effectively when doing event driven programming).

Dispatch loop and Javascript

JS is single threaded. But it can send stuff off to things outside the language, and do other things while those dispatched items are being processed. …. "in effect the dispatch loop turns the program inside out, making the management of tasks explicit and the control flow within tasks implicit"
For instance, consider task "get a web page then ask user if it relevant, if yes, save local copy". in JS, need to write this as:

event Handler for web page loaded — ask if relevant
event Handler for user input about relevance — if yes save, if no download next web page
event Handler for save complete — download next web page

Main
    get list of web pages to ask about
    start web page download

Thread == every concurrent activity. So how many threads are running on computer??
== "active entity that the programmer thinks of as running concurrently with other threads"
UNIX: "ps -e | wc" is a good start.   But that undercounts as these are processes. Many processes are multi-threaded. So need to get information on a thread basis
-e == every

```
ps -eL | wc
     -L == show threads
```

heavyweight process == own address space (a unix process?)
lightweight process == groups share address space
'lightweight processes were added to Unix to accommodate shared-memory multiprocessors"
"One of the major characteristics of shared memory multiprocessors is that all processors have equally direct access to one large memory address space."
"In systems with no shared memory, each CPU must have its own copy
of the operating system, and processes can only communicate through message passing."

Communication and Synchronization

comm == "any method that allows one thread to obtain information from another thread"

shared memory   — some or all memory os accessible to threads
message passing — (pure) no shared memory "send" and receive

Java — shared and can do message passing (https://www.coderscampus.com/java-multithreading-java-util-concurrent/) but it really often ends up looking like shared memory
Go — shared and message passing (will discuss shared)
Elixir — message passing only

Example of the need for sync in shared memory system

Java **counter_java/BCounter.java**

massive undercounting

Go **counter_go/counter.go**

same issue, big undercount

Elixir — with no shared (updateable) memory this Java/Go issue cannot arise

see **Tasks_ex/spawn.ex** and **Tasks_ex/task.ex** for a simple example starting threads in elixir doing much the same thing as Java/go but no shared memory so no interaction among threads

synch mechanisms

busy-wait /spinning

```
while (1)
    sleep(100)
    check if there is something to do
```

blocking — idea give control to another thread and leave a note somewhere saying you are blocked; In the future, another thread sees the note and unblocks

Alternate: in one thread: in a section of code, put up a block. Run code. Unblock.

in Java definitely want to unblock in a try/finally. In Go, put unblock in a defer.

Java: label a whole function as "synchronized" then only one thread can do anything with it. Vector and Hashtable are synchronized but this is really done by making every writing method synchronized. Is it sufficient to have each method simply labeled "synchronized"?

events_java/GTReader.java

Here we have our own event and handler Idea is that we want to read from the internet. But we want to keep doing stuff. So do the read in a separate thread. When complete, tell the main thread that reading is complete. Done using busy/wait syncing

fork == start a new thread

join == merge two threads

Problem with shared memory — synchronization. What happens when two threads read/write to shared memory at the same time?

As in the above programs

More generally, shared-memory systems can be subject to all of these issues:

- *Data race,* when two or more threads attempt to access and write the same data.
  - (examples in Go and java above) Question: how can I reduce/eliminate atomiticity violations in the counter programs
- *Deadlock:* when two or more threads attempt to access a shared resource protected with locks (usually in an attempt to fix a data race!)
  - see **ThreadLock_java/TL.java**
- *Livelock:* similar to deadlock, except one thread is not actually locked — it executes continuously trying to acquire a shared resource.

- *Starvation:* in which a process is unable to complete in the allotted time because a resource is given to higher priority processes. (Can also occur when there are too many threads)
    - On the counter.go program
    - `1000000000 * 10` threads: 38.23s user 0.47s system 707% cpu 5.469 total
    - `100 * 100000000` threads: 91.51s user 12.24s system 484% cpu 21.413 total
    - `10 * 1000000000` threads: 704.96s user 126.58s system 385% cpu 3:35.83 total
- *Blocking suspension:* when one thread waits an unacceptably long time for a resource. Similar to starvation, but it eventually succeeds.
- *Order violation:* when the desired order of operations between multiple threads is violated.
- *Atomicity violation:* in which two code blocks in a thread are interleaved with code blocks in another thread, such that the result of the computation is inconsistent. In other words, a block of code that a developer intended or expected to be atomic is not executed atomically.
- https://blog.devgenius.io/how-elixir-helps-you-write-bug-free-concurrent-code-97dfd0577f19

Handling the starvation problem: limit threads
First issue after simply starting threads. How do you limit the number of threads running?
Why do you want to limit?
Java **counter_java/EPCounter.java**
effectively the same as CCounter, but stylistically better. Idea is to use java system that limits the number of active threads. Question: why is this a good idea?
Go **counter_go/LimitedThreadCounter.go**
elixir **taks_ex/limitedTasks.ex**

task parallel == break down problem into a set of "tasks". Run these in parallel. Problem: can only get as much parallelism as you can define tasks
data Parallel == break problem into smaller pieces by way of data. Then do very similar operations on each piece. (Similar to vector level parallel from above)

Implementing threads
at the OS level could 1-1 map thread to OS process
but OS processes are heavier (they are not "heavywieght")
Alternative 1 OS process shared among all threads
but then the language is effectively single-threaded
In practice language runtime pre-gets many processes and then shares those among threads

Shared memory handling
need a method of making sure that shared memory is only used by one thread at a time
Semaphors

basically a counter which you cannot get by unless it is 0 for you
        each thread is responsible for incrementing/decrementing the semaphor
    Monitor
        set a block of code/memory that only one thread at a time can use
        Java "synchronized" is essentially a monitor
    Lock
        when you set to a point in the code, check if "lock" has been set. If yes, wait for unlock

Java has at least 3
- ReentrantLock — much like a semaphor
- Atomic — a datatype with its own lock.
    - atomic == inseparable. imagine everything happens at same time
- Synchronized — much line monitor
    - see **counter_java/ExecCounter.java**
    - see **counter_go/atomiccounter.go**, **counter_go/lockcounter.go**

For real world problem using shared memory
        see **timeJSON.go**   getting info about all public lab machines.

Message Passing
    There is no shared memory
    Advantages? naturally applies to distributed systems
    Disadvantages?? speed

    Elixir cannot do shared memory (why?)
    MP requires knowing to whom you are passing the message!!

    A distinguished "manager thread"?? Or at least a post office thread

    see **Tasks/task.ex** for a simple example starting threads in elixir
    see **crawler_ex/crawler.ex** for message passing web crawler

Homework: perfect number finder

```
time elixir perfect.ex > perr.out
real    1466m55.366s
user    10852m14.119s
sys     5m49.169s
```

Speedup from parallel — over 7 on an 8 core machine

Results:

```
"AAA"
-6 ---> [815634435, 2102272, 1155, 592, 315, 52, 15, 7]
-5 ---> []
-4 ---> [73995392, 536920064, 15370304, 8394752, 116624, 8384, 2144, 884, 152, 44, 14,
 5]
-3 ---> []
-2 ---> [32896, 136, 10, 9, 3]
-1 ---> [536870912, 33554432, 8388608, 134217728, 4594592, 2097152, 780572, 524288,
 131072, 32768, 22952, 11990, 8192, 2048, 1892, 512, 182, 128, 32, 8]
0 ---> [33550336, 8128, 496, 28, 1]
```

```
1 ---> [4]
2 ---> [134193152, 8382464, 522752, 130304, 1952, 464, 104, 2]
3 ---> [18, 16]
4 ---> [8378368, 1848964, 521728, 32128, 5830, 4030, 1888, 88, 70]
5 ---> [6]
6 ---> [442365, 32445, 8925]
```

Checkpointing and long parallel runs.