Concurrency

- A PROCESS or THREAD is a potentially active execution context.
- Classic von Neumann (stored program) model of computing has a single thread of control.
- Parallel programs have more than one.
- A process can be thought of as an abstraction of a physical PROCESSOR.
- Processes/Threads can come from:
  - multiple CPUs
  - kernel-level multiplexing of single physical machine
  - language or library level multiplexing of kernel-level abstraction

Concurrent Processes

- They can run:
  - in true parallel
  - unpredictably interleaved
  - run-until-block
- Most work focuses on the first two cases, which are equally difficult to deal with.
- Common scenario:
  - the operating system multiplexes one or more processes on top of one or more physical processors.
  - and a library package or language run-time system multiplexes one or more threads on top of one or more OS processes.

Classes of Programming Notation

- Two main classes of programming notation:
  1. synchronized access to shared memory
  2. message passing between processes that don't share memory
- Both approaches can be implemented on hardware designed for the other.
- Although shared memory on message-passing hardware tends to be slow.
- We'll focus here on shared memory.
- The book covers both.
Process creation syntax
- static set
- co-begin: Algol 68, Occam, SR
- parallel loops
  - iterations are independent: SR, Occam, others
  - iterations are to run (as if) in lock step: F95
- launch-on-elaboration: Ada, SR
- fork (join?): Ada, Modula-3, Java
- implicit receipt: DP, Lynx, RPC systems
- early reply: SR, Lynx

Race Conditions
1. A race condition occurs when actions in two processes are not synchronized
2. And program behavior depends on the order in which the actions happen.

Leads to unexpected/unwanted behavior.
Example: 2 processes can access shared variable in Electronic Controlled Steering (e.g., torque sensors).
- amount of assisted steering is dependent upon the value of the variable
- What might happen?

Synchronization
- SYNCHRONIZATION:
  - act of ensuring that events in different processes happen in a desired order.
  - Synchronization can be used to eliminate race conditions.
  - Most synchronization can be regarded as either:
    - MUTUAL EXCLUSION: ensuring that only one process is executing a CRITICAL SECTION at a time
      - E.g.: touching a variable
    - CONDITION SYNCHRONIZATION: ensuring that a given process does not proceed until some condition holds (e.g. that a variable contains a given value).
Example:

index: 1..SIZE
buf: array [index] of data
nextfree, nextfull : index

procedure insert(d : data)
% put something into the buffer, wait if it's full
procedure remove : data
% take something out of the buffer, wait if it's empty

A solution requires
1. Only one process manipulate the buffer (or at least any given slot of the buffer) at a time.
   - Mutual Exclusion solution
2. Processes wait for non-full or non-empty conditions as appropriate.
   - Condition Synchronization

Common Concurrency Concepts

- Atomic actions: happens all at once
  - needed to implement synchronization
  - Ex: reads, writes of individual memory locations
- Spinning or Busy-waiting:
  - Repeatedly reading a shared location until it reaches a certain value
  - Spin lock: busy-wait mutual exclusion mechanism
  - Spin lock is better than put processor to sleep if expected spin time is less than rescheduling overhead.

Schedulers

- Give us the ability to “put a thread/process to sleep” and run something else on its process/processor.
  - Start with coroutines
  - make uniprocessor run-until-block threads
  - add preemption
  - add multiple processors
Scheduler

- Coroutine: multiple execution contexts, only one of which is active
- Preemption: Use timer interrupts (in OS) or signals (in library package) to trigger involuntary yields.
- See book for examples of how to implement scheduler

Alternative to Shared Memory

- Distributed memory
- Use message passing to communicate changes to memory

Architectures for Concurrency

- Multiprocessors (HPC)
  - Shared memory model
  - Efficiency in operations
  - Less flexible
- Clusters of workstations (NOWs)
  - Distributed memory
  - Overhead for communication
  - Flexibility in configuration