Parallel Prog

Louden
14 Parallel (1).

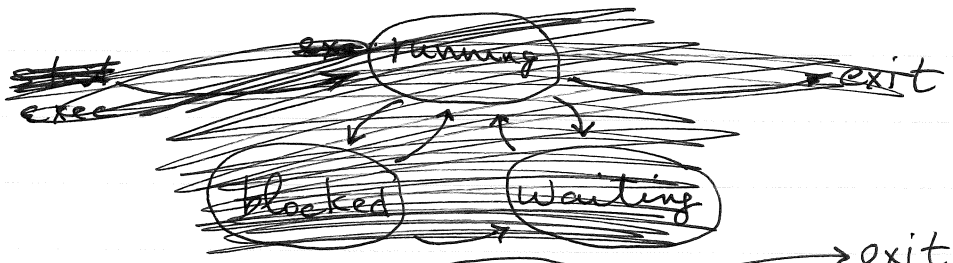
- parallel vs concurrent
- hardware vs software

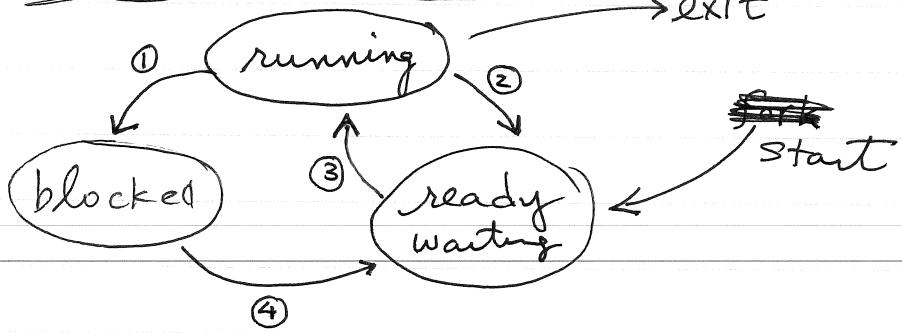
parallel = at same time
concurrent = context switching via softw.
  ↳ preemptive
  nonpreemptive = coroutines

Intro

Process = program in execution.

Process state





① sys req
② timeout (hardware) or yield (voluntary)
③ dispatch
④ sys req complete

process = mem + sys resources
thread shares process w/ other threads
  private regs + stack.

kernel thread vs green (user) thread
  pthreads

parallel w/ multiple CPUs.

Louden

contextswitch — 14 Parallel (2).

~~1. push regs on curr stack~~

1. store regs in curr thread context
2. load regs from new ctx
3. jmp to curr insn

(is ip a user reg?)

concurrent access:

shared mem problem

race condition

ex:

observers	reporters	shared
loop {	loop {	x.
waitfor(event);	~~wait~~ sleep(5 min)	
x = x + 1	~~private~~ y = x	
}	x = 0; ↔ priv y.	

- need mutual exclusion
- critical section = short as possible.
  - no print in crit.
- only a prob if time slicing

Synchronized

Deadlock:
1. mutex
2. hold & wait
3. non-preemption
4. circular wait

Louden

Lang - no explicit facilities — 14 Parallel (3)

~~func, objor~~

- func, logic, oo lang.
- implicit parallelism
- run on different processors

~~ex~~

Fortran : options to parallelize arrays.

14.2.2 Process Creation

- separate processes.

fork() vfork()
execl()

Granularity — process
program
thread
proceduP
stmt
operatri

Stmt-level parallel

FTN 95 → forall (i = 1:100 , j = 1:100)
⋮
end forall.

14.3 Threads

fine to medium grained parallel.

```
class P extends Thread {
    public void run() { ... }
}
Thread t = new P();
t.start();
```

OR

```
class P implements Runnable {
    pub void run() { ... }
}
P x = new P();
Thread t = new Thread(x);
t.start();
```

in addition to main thread (& possibly a gc thread).

green threads — do not use OS mechanism
— managed by JVM

native threads — use OS threads.
— slower (lightweight p).
↑
kernel.

Louden

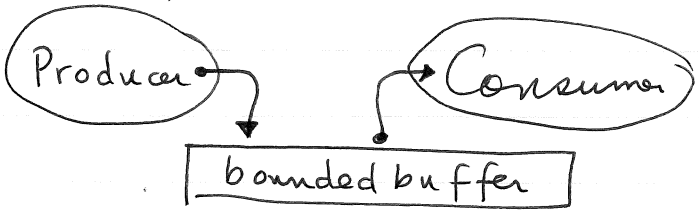
14 Parallel (5).

Done with thread

```
t.start()
  :
  :
t.join() - wait for thread
t.join(1000) - wait one sec, give up.
```

thread done when run() returns.

Producer / Consumer Example




```
class Producer implements Runnable {
    Produce (~~Blue~~ Buffer b) { ... }
    pub void run() { ~~loop~~ ... b.put(x) ... }
}
class Consumer implements Runnable {
    Consumer (Buffer b) { ... }
    pubvoid run() { loop{ ... =b.get() ... }
}
Buffer b = new Buffer();
Thread c = new Thread (new Consumer(b))
Thread p = new Thread (new Producer(b))
c.start();
p.start()
```

14.4 Semaphores

Semaphores by E.W. Dijkstra.

(proberen) P(s): if s>0 then s=s-1
else suspend

(verhogen) V(s): if proc waiting then wakeup
else s=s+1

atomic operations

shared
sema S=1
int x=0

observer
P(s)
x=x+1
V(s)

reporter
P(s)
y=x; x=0
V(s)
print y

Bounded Buffer w semaphors

shared buf[5], sema lock=1
sema free=5
sema used=0

producer
local
loop { x=make
P(free)
P(lock)
buf.put(x)
V(lock)
V(used)
}

consumer
loop { local x
P(used)
P(lock)
x = buf.get()
V(lock)
V(free)
use x
}

deadlock
if P in wrong order.

JAVA SEMAPHORES


```
class sema {
    int level = 1;
    sema () { }
    sema (int init) { level = init }
    synch void P() {
        [redacted]
        level --;
        if(level < 0)
            try { wait(); }
            catch ( Int Exn ) { }
    }
    synch void V() {
        level ++;
        if (value <=0) notify()
    }
}
```

---

pipeline sort: O(n) time
O(n) computers

14.5 Monitors

```
monitor buffer;
    condition notfull, notempty.
    synchronized entry put(x) {
        if (no space) then wait(notfull)
        put in queue
        signal(notempty)
    }
    synchronized get() {
        if (no data) then wait(notempty)
        t = get from queue
        signal(notfull)
        return t
    }
}
```

Java: notify()
notifyAll()
wait()
sleep(ms).

macro lang - shell, #cpp
- SQL

Message Passing

send (process to, message m)
receive (process from, message m).

- name both sender and receiver.
- or: send to any
  receive from any.
- usual: send to specific
  rec from any.

1. sender wait until receiver ready
   - or continue
2. receiver wait until msg ready.
   - or accept null msg.

Both wait ⇒ Rendezvous.
otherwise ⇒ mailbox mechanism