CS 744 — Lecture 16.

(*) 1 — Lab 4a — available
- 4b — ~~server~~ + thread pool + pthreads.
- 4c — cflask (http header parsing + function dispatch + pthreads)

(*) 2 — Projects
- ideas list available (some ongoing edits)
- groups of 1 or 2
- original ideas not listed encouraged (get 1000 bonus points)*

(*) not redeemable.

(*) 3 — Lab Exam #2
- 25th Oct. Fri° 8.30 am

(*) 4
- 9th Oct. Wed — pthreads lab session meet in SL2
- 11th Oct. Fri° — guest lecture (CC103) measurement, analysis & scalability reasoning of computing systems

(*) Threads & Synchronization.

Q. What is a thread?

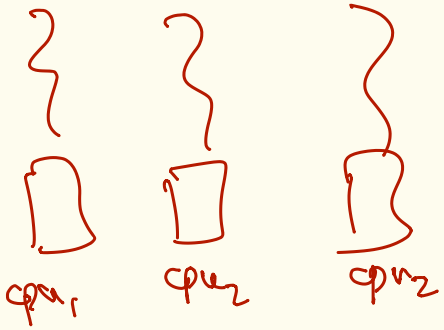
~ an independent ordering/sequence of instructions.

~ "thread of execution"

~ process is an example of thread of execution.

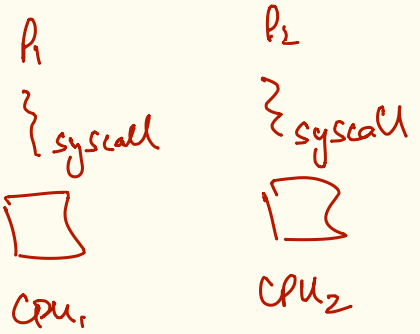
Q Why is multi-threaded execution interesting? challenging?

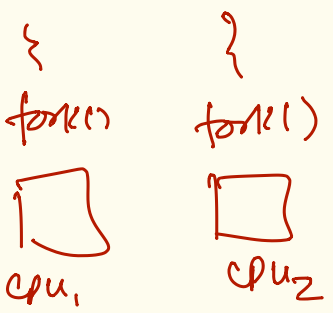
- multiple threads of execution may simultaneously access / or have access to shared memory regions / data objects.
- all in memory writes: read - update - write → interleaving of actions of multiple threads via can lead to inconsistency.



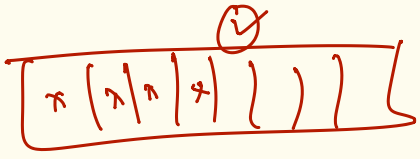
- shopping cart
- ticket service
- money transfer

Q does this happen in the Kernel?





fork executing on two or more cpus.

(i) 

(R/W) Shared kernel state

(ii) freelist access for memory allocation.

(iii) ready queue ~ list of ready processes for scheduling.

within the Kernel simultaneous access to shared state is possible ——

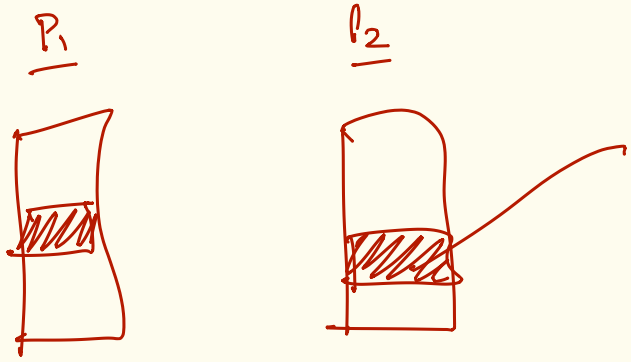
(i) system call + system call

(ii) system call + interrupt

(iii) interrupt + interrupt

Q ~ how about multiple threads of execution in user mode?

(i) processes —



shared memory regions need to be mapped to same physical pages by OS in the page tables.

(ii) threads (pthread) threads of execution in user space

threads — by def^n. share code, data & heap.

have independent stack.

- user space threads
- Kernel threads ✓
  - independent PCB w/ thread info
  - schedulable.

Q. how to "protect" / safely / correctly / consistently read / update shared state?

* critical section ~ where possibilities of getting in inconsistent state exist due to access of shared state.

(i) disable preemption → single CPU
depend on exit, yield, ...

(ii) disable interrupts → ok for multi-cpu
stalling all CPUs
doesn't address user VA accesses.

(iii) atomic updates → ISA supported
eg. x86. lock inc R0;
prefix locks the address bus till instruction is complete.

(iv) locks
locking based primitives

getlock(); — atomic
// critical section
releaselock();

Q What is a lock?

- lock is a variable (value at a memory address)
- lock ← 0 // available
  lock ← 1 // not available.

Spin lock: (busy loop)

Spins on the CPU till gets a lock.

(i)

Lock is the memory variable

```
getlock: MOV R0, 1
         TSL LOCK, R0
         CMP R0, 0
         JNZ getlock
         ret

unlock:  MOV LOCK, 0
         ret
```

TSL, compare & swap

xchg.

atomic {
- oldvalue = LOCK
- LOCK = R0
- return oldvalue in R0
}

lock — 0/1

mutex ~ yields CPU if lock not available & wakes up to test lock when lock released.

condition variables ~ any generalized condition for synchronization.

Semaphores (integers/conditions) ~ up V — increment & wakeup
down P — decrement & sleep

reader/writer lock —

barrier