

# Programming Paradigms

Note Title 1/20/2009

shared memory  
common addr space

proc 1 write to loc 0x2000

proc 2 read from loc 0x2000

sync is critical

message passing

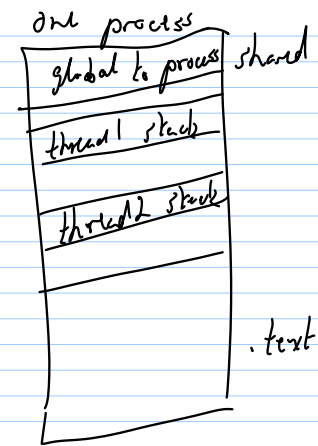
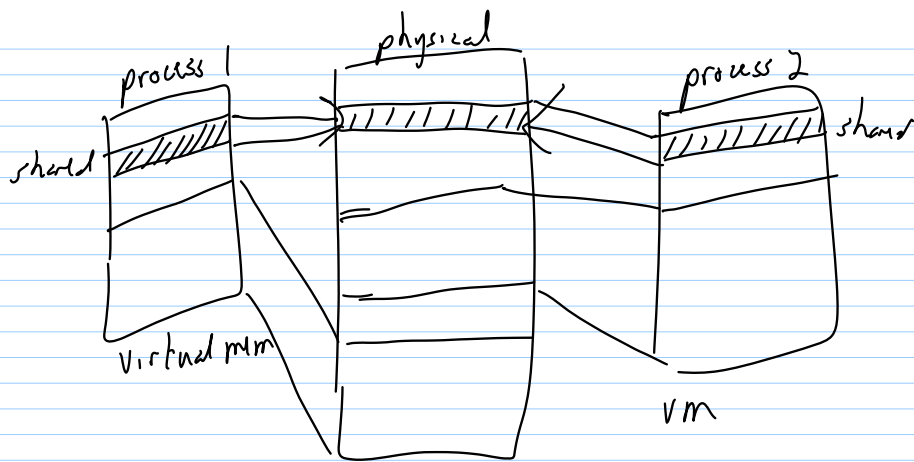
private memory space

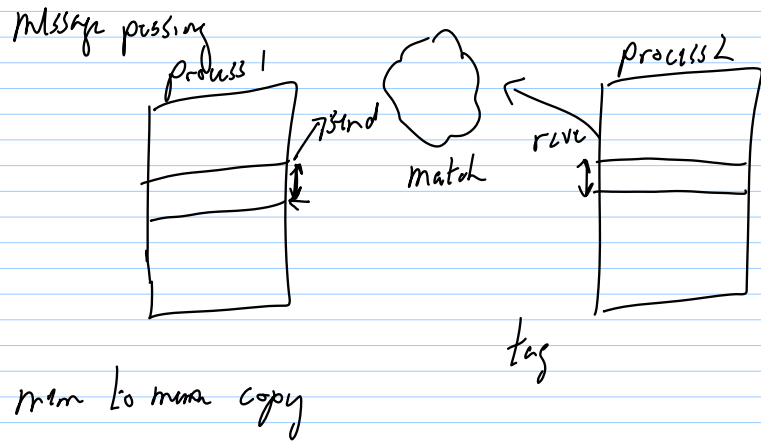
proc 1 issue send()

proc 2 issues read() / recv()

Not tied to architecture

e.g. Dist. Shared Mem. (DSM) systems

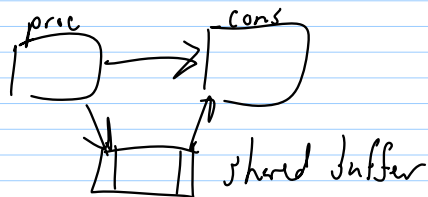




## Shared Memory

similar to time-sharing on uni processor  
 $\Rightarrow$  fundamental issue is concurrency  
 but instead of interleaved execution  
 we have parallel "

class.c problem producer / consumer



enqueue()

allocate entry  
 copy data into entry  
 update local ptrs  
 " global ptrs  
 return

dequeue()

follow ptrs  
 copy data from entry  
 update ptrs  
 free entry  
 return

need mech. for mutual exclusion

- mutex lock

mutex lock allows 1 thread access (owner),  
 and blocks all other threads until owner  
 releases lock

to ensure proper (correct) operation, place lock  
 around critical section

program 1) program 1)  
 lock (queue) lock (queue)  
 allocate follow ptrs  
 copy copy  
 update ptrs update ptrs  
 unlock (queue) free  
 return unlock (queue)  
 return return

critical section critical section

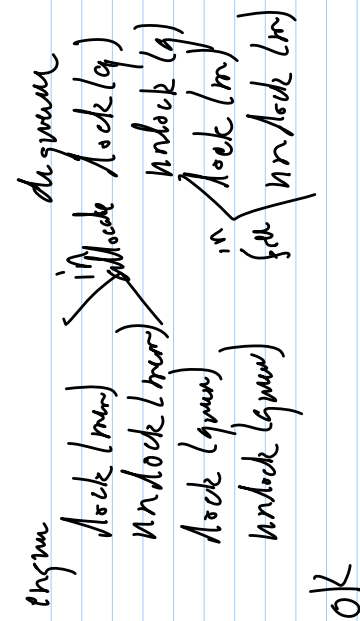
req. any thread that obtains a lock must release it, or deadlock may result

critical section is mutually exclusive  
 ⇒ limits concurrency  
 ⇒ " parallelism  
 ∴ minimize time in CS

rewrite

program 1) program 1)  
 allocate lock (queue)  
 copy data follow ptrs  
 lock (queue) update ptrs  
 update ptrs unlock (queue)  
 unlock (queue) copy data  
 return free  
 return return

allocate + free might use mutex ("mutex")



want:                    set:  
while (!flag);        flag = 1;  
or  
while (!flag)        signal (flag);  
  sleep();

but  
→ lock (m)        lock (g)  
   lock (g)        lock (m)  
   unlock (g)      unlock (m)  
   unlock (m)      unlock (g)  
deadlock

condition variables  
protected by mutex  
pthread  
= mutex\_lock (lock)  
= cond\_wait (cond, lock)  
= mutex\_unlock (lock)

another sync primitive  
spin lock or variable  
flag is initially 0, waiting  
thread doesn't proceed until flag  
is non-zero.

---

Thread B

- mutex\_lock(lock)

- cond\_signal(cond)

- mutex\_unlock(lock)