

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

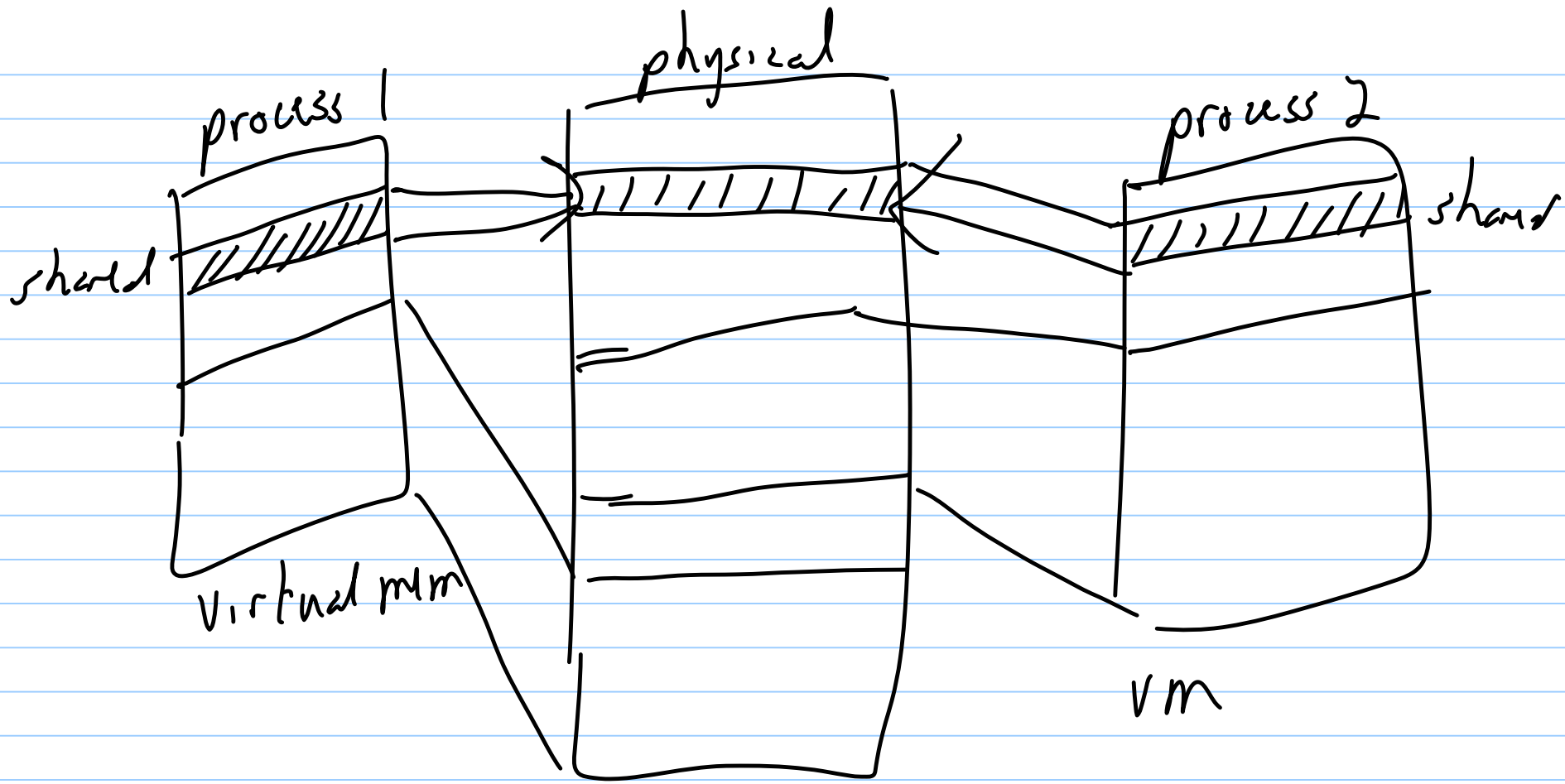
private memory space

proc 1 issues send()

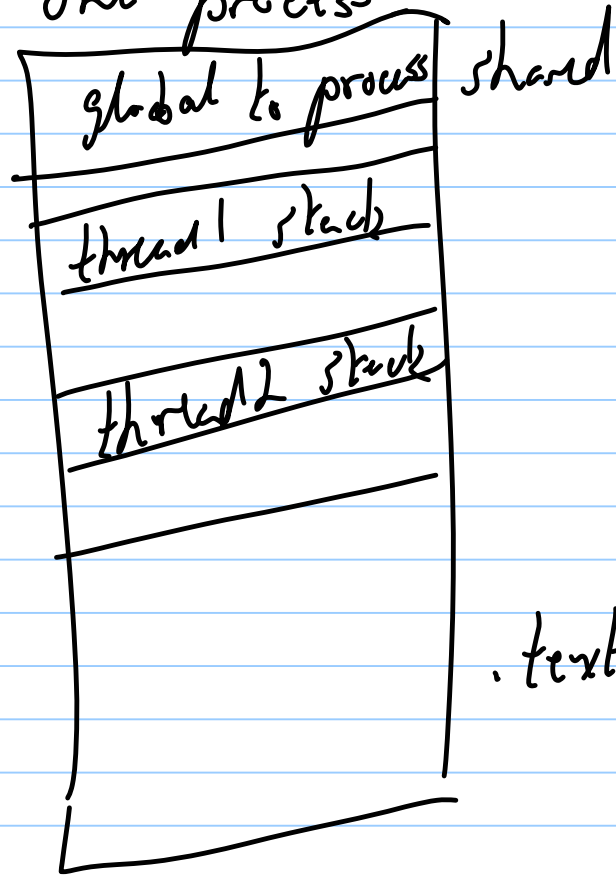
proc 2 issues read() / rcv()

Not tied to architecture

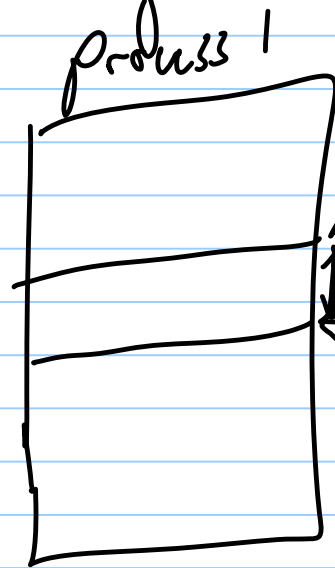
e.g. Dist. Shared Mem. (DSM) systems



one process



Message passing



send



Process 2



recv

tag

mem to mem copy

Shared Memory

similar to time-sharing on uni processor

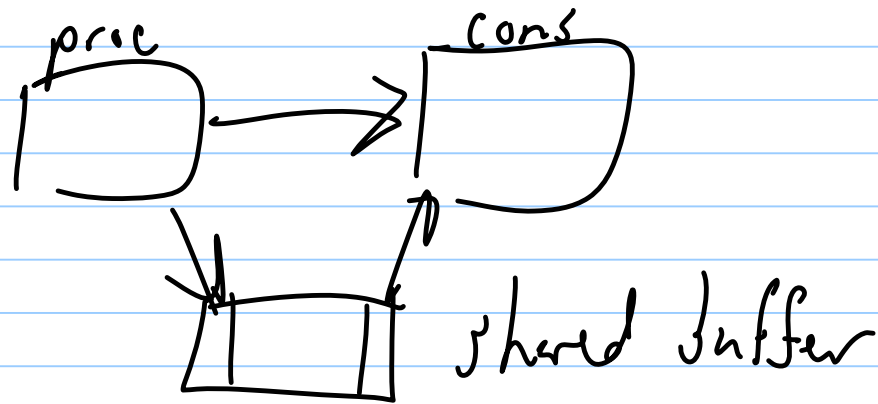
⇒ fundamental issue is concurrency

but

instead of interleaved execution

we have parallel " "

classic 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

enqueue ()
 lock (queue)
 allocate
 copy
 update ptrs
 unlock (queue)
 return

critical section

dequeue ()
 lock (queue)
 follow ptrs
 copy
 update ptrs
 free
 unlock (queue)
 return

critical section

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

critical section is mutually exclusive

\Rightarrow limits concurrency

\Rightarrow " parallelism

∴ minimize time in CS

rewrite

```
enqueue()
  allocate
  copy data
  lock (queue)
  [update ptrs
  unlock (queue)
  return
```

```
dequeue()
  lock (queue)
  [follow ptrs
  [update ptrs
  unlock (queue)
  copy data
  free
  return
```

allocate + free might use mutex ("mutex")

program

lock (mem)
unlock (mem)
lock (qmem)
unlock (qmem)

in
allocate

de program

lock (q)
unlock (q)
lock (m)
unlock (m)

ok

bwt

lock (m)
→ lock (g) →
unlock (g)
unlock (m)

~~lock (g)~~
lock (m)
unlock (m)
unlock (g)

deadlock

another sync primitive
spin lock on variable

flag is initially 0, waiting
thread doesn't proceed until flag
is non-zero.

want:

while (!flag);

set:

flag = 1;

OR

while (!flag)
sleep();

signal (&flag);

condition variables

protected by mutex

don't req. spin wait

thread A

= mutex_lock (lock)

= cond_wait (cond, lock)

= mutex_unlock (lock)

Thread B

- mutex_lock(lock)
- cond_signal(cond)
- mutex_unlock(lock)