Lecture 14

Official GATE Syllabus for Operating Systems

Section 8: Operating System

System calls, processes, threads, inter-process communication, concurrency and synchronization. Deadlock. CPU and I/O scheduling. Memory management and virtual memory. File systems.

Topics Covered So far

- System calls, processes, threads
- CPU and I/O scheduling.

Topics to be covered

- Inter-process communication, concurrency and synchronization.
- Memory management and virtual memory.
- Deadlock.
- File systems.

Official GATE Syllabus for Operating Systems

Section 8: Operating System

System calls, processes, threads, inter-process communication, concurrency and synchronization. Deadlock. CPU and I/O scheduling. Memory management and virtual memory. File systems.

Topics Covered So far

- System calls, processes, threads
- CPU and I/O scheduling.

Topics to be covered

- Inter-process communication, concurrency and synchronization.
- Memory management and virtual memory.
- Deadlock.
- File systems.

Next Topic:
Inter process Communication

Small topic

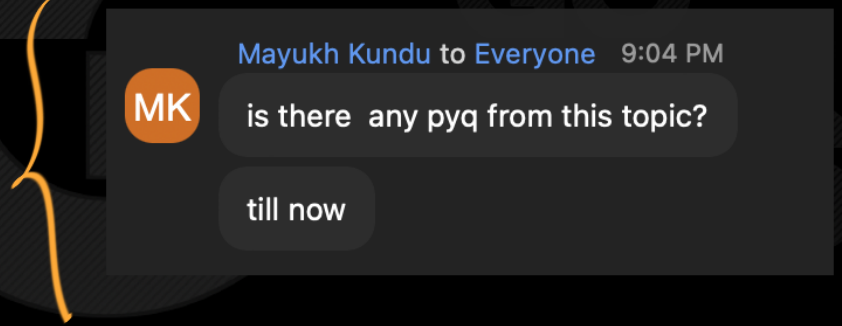
Next Topic:

Inter process Communication

Small topic

Next Topic:

Inter process Communication



NO

* Suppose two processes within a system wants to talk. ⇒ take help of OS

Two processes are in different systems

OS + CN both concepts are needed.

Operating systems provide facilities for inter-process communications (**IPC**), such as

- Shared memory
- Message queues
- Pipes

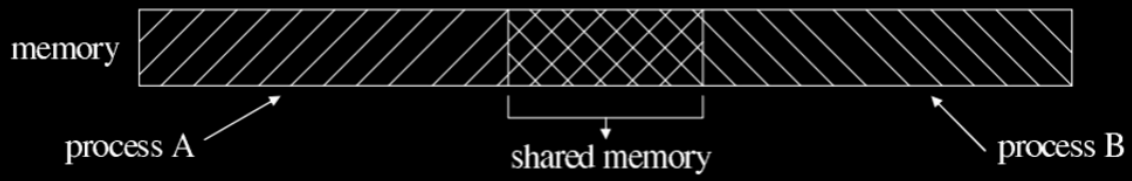
GO CLASSES

Shared memory

this is shared memory which is in user space

Shared Memory

- Easiest way for processes to communicate
  - at least some part of each processes' memory region overlaps the other processes'

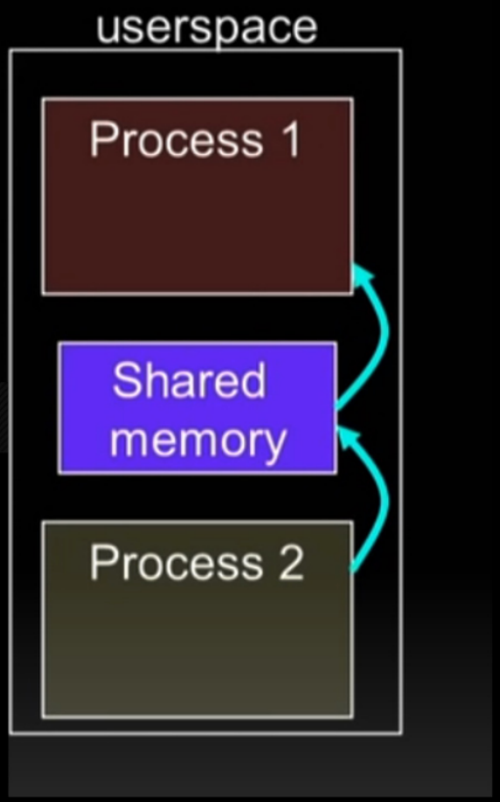


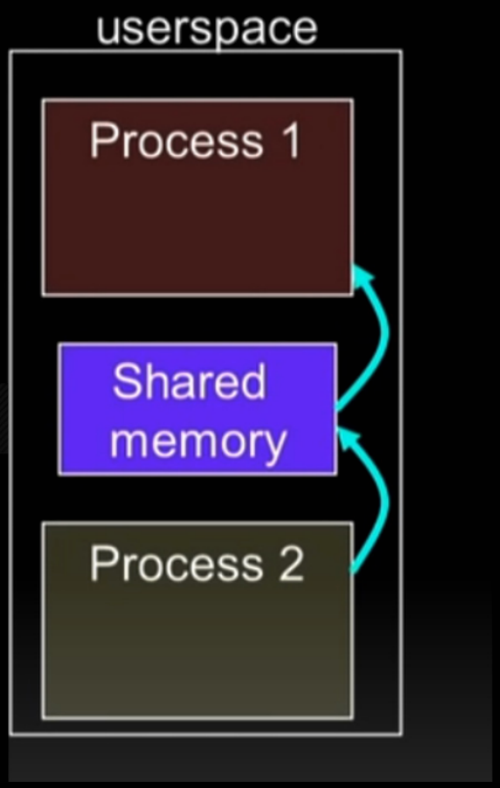
- If A wants to communicate to B and vice versa
  - A writes into shared region
  - B reads from shared region

https://pages.cs.wisc.edu/~mattmcc/cs537/notes/IPC.pdf

Shared Memory

- One process will create an area in RAM which the other process can access
- Both processes can access shared memory like a regular working memory
  - Reading/writing is like regular reading/writing
  - Fast



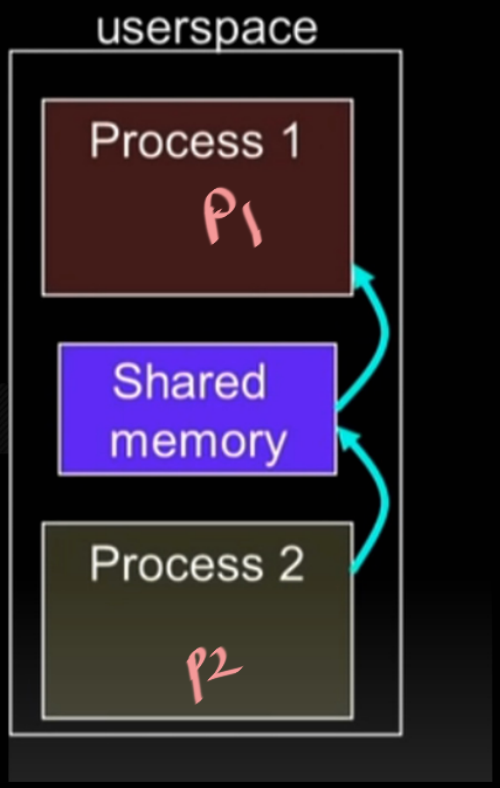


to create Shared memory we need OS support

↳ System call

But once we have the Shared memory we do not need OS support

← because this memory is in user space



P_1, P_2 can modify the memory just like normal memory

→ any of the process can read or write to/from memory.

GO CLASSES

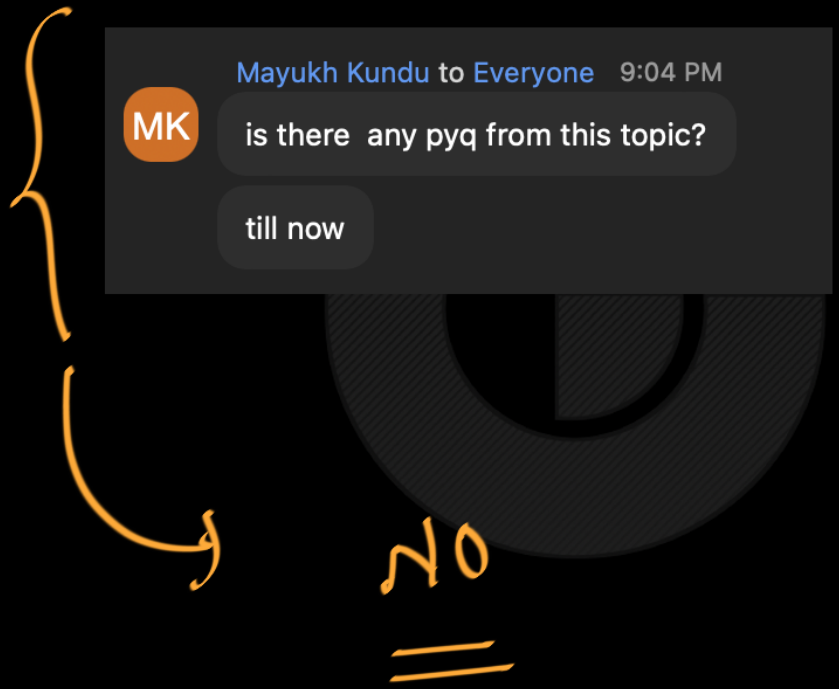
Where is this memory?

Neither in the stack nor in the heap.

(malloc() gives memory in heap)

* Shmget() gives memory in .....

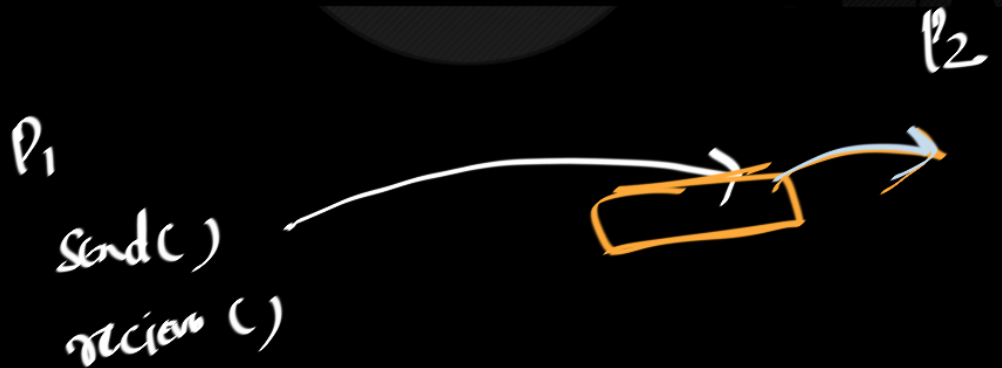
↑ it gives the different memory which in anone's stack or heap.

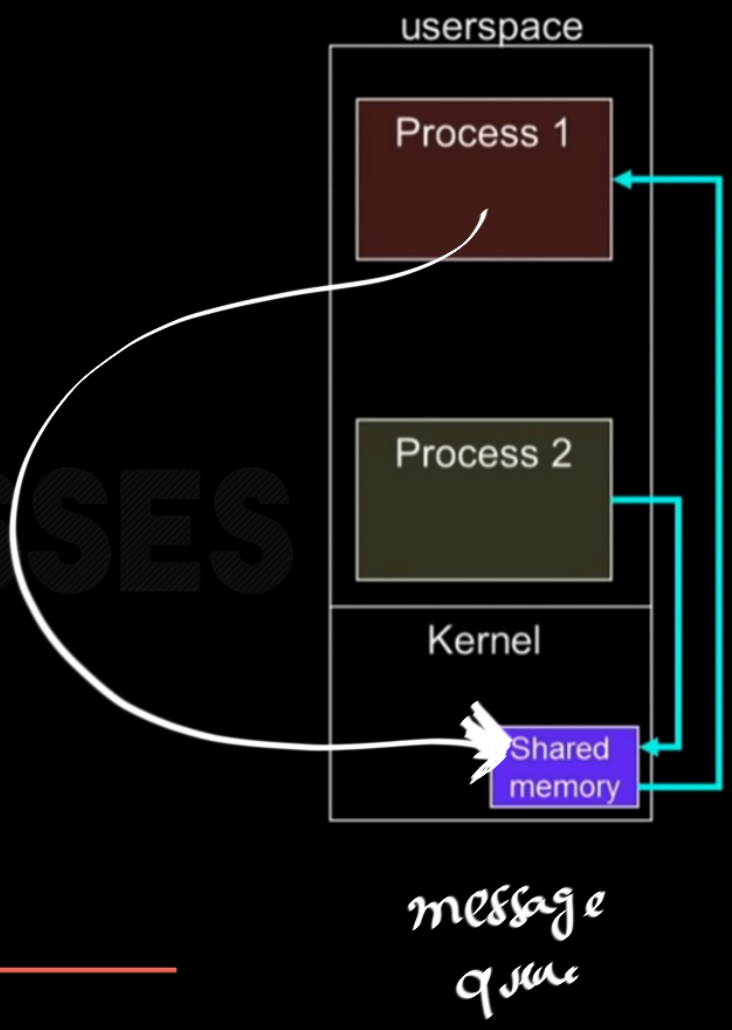

Mayukh Kundu to Everyone 9:04 PM
MK
is there any pyq from this topic?
till now
NO

Message passing

Message passing

- Shared memory created in the kernel
- System calls such as send and receive used for communication
  - Cooperating : each send must have a receive





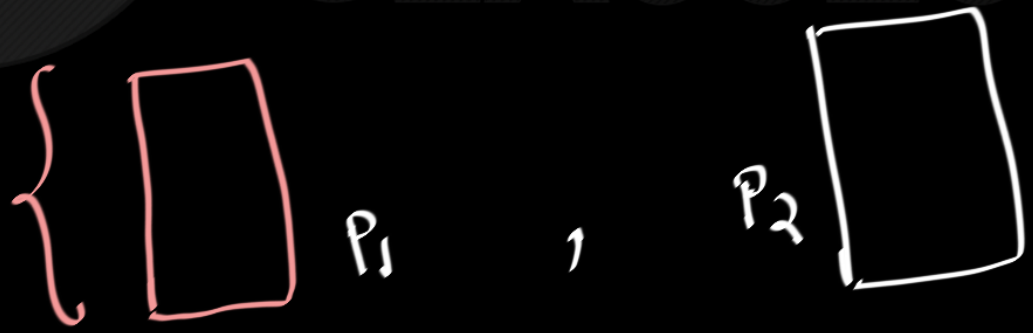
GO CLASSES

or illusion

So the memory isolation abstraction is maintained

OS provides many illusions to us.

LAS
Logical address space



Message Passing

- With message passing, processes do not share any address space for communicating
  - So the memory isolation abstraction is maintained
- Two fundamental operations:
  - send: to send a message (i.e., some bytes)
  - recv: to receive a message (i.e., some bytes)
- If processes P and Q wish to communicate they
  - establish a communication "link" between them
    - This "link" is an abstraction that can be implemented in many ways
      - even with shared memory!!
  - place calls to send() and recv()
  - optionally shutdown the communication "link"

System calls

in Kernel Space (message queue)

Question

c. (4 points) Why would two processes want to use shared memory for communication instead of using message passing? Your answer should be brief (no more than 5 sentences).

https://hkn.eecs.berkeley.edu/examfiles/cs162_sp06_mt1_sol.pdf

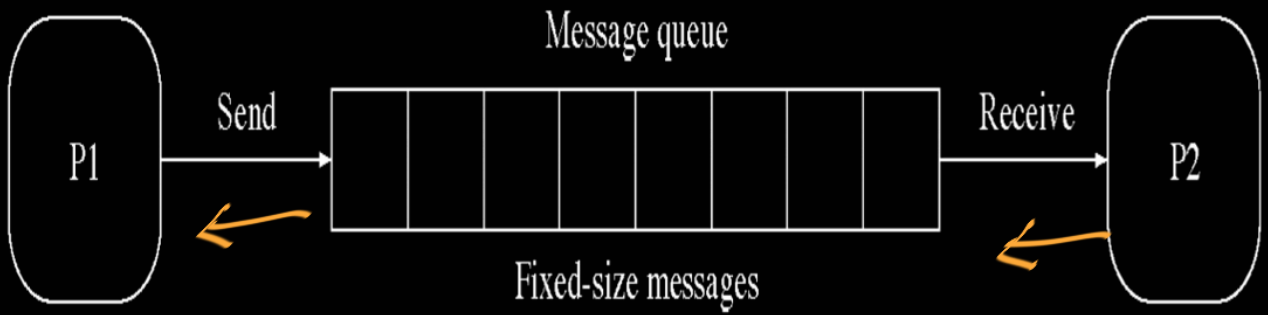
Question

c. (4 points) Why would two processes want to use shared memory for communication instead of using message passing? Your answer should be brief (no more than 5 sentences).

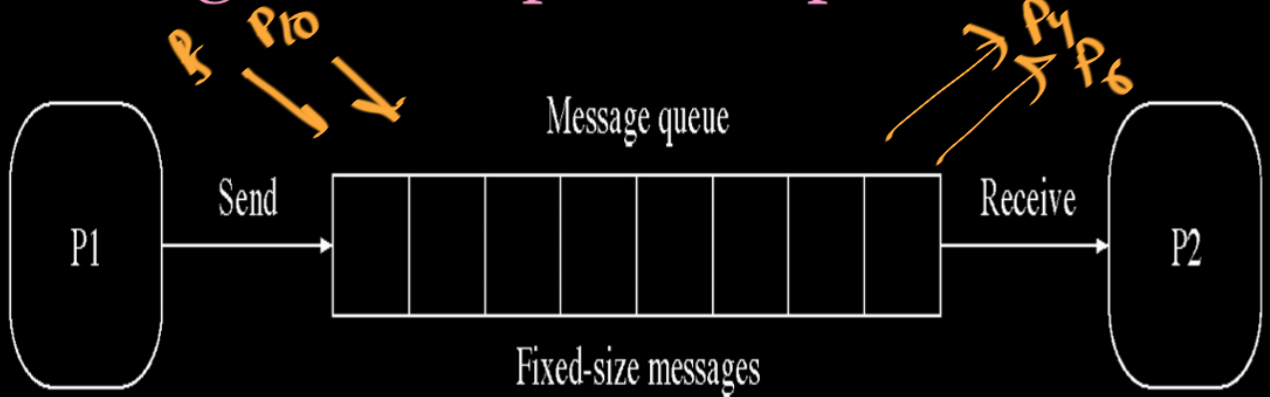
*Shared memory has **higher performance** because you **avoid the context switch overhead** of using message passing through the kernel. We gave full credit only if your answer included both performance and context switch overhead.*

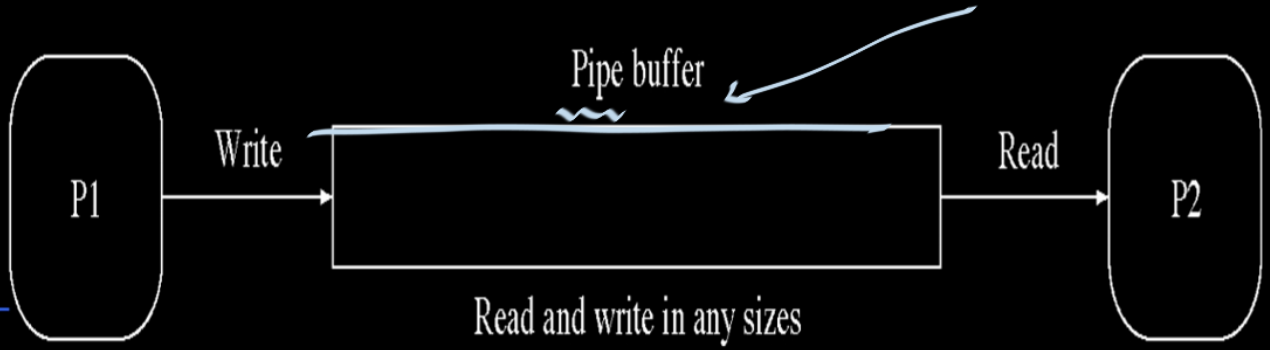
https://hkn.eecs.berkeley.edu/examfiles/cs162_sp06_mt1_sol.pdf

this is in Kernel space.



Messages and Pipes Compared





can only be implemented within parent and child

Operating systems provide facilities for inter-process communications (**IPC**), such as

- Shared memory
- Message queues
- Pipes

there are multiple ways to have IPC, all of these ways are having support from libraries

Next Topic:
Synchronization