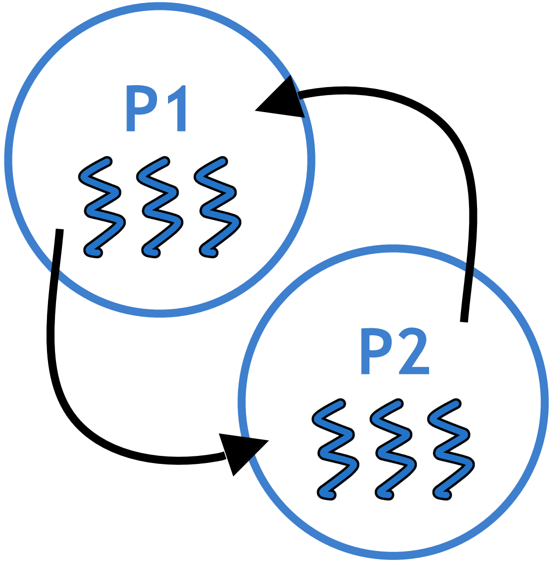
IPC and Shared Memory

INTER-PROCESS COMMUNICATION

Inter-Process Communication (IPC)

IPC == OS-supported mechanisms for interaction among processes (coordination and communication)

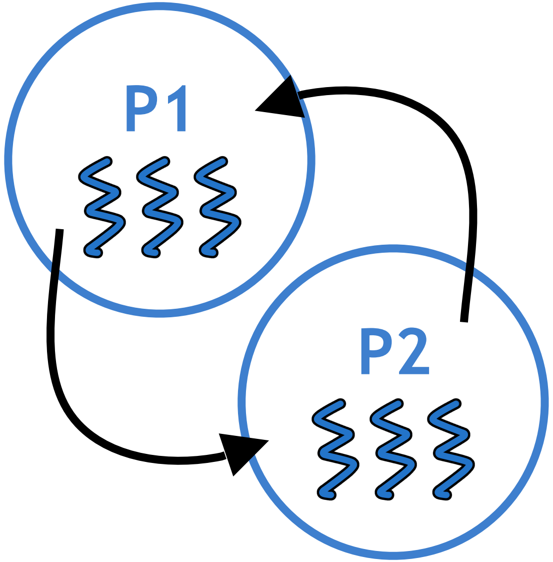
- message passing
- memory based IPC



Inter-Process Communication (IPC)

IPC == OS-supported mechanisms for interaction among processes (coordination and communication)

- message passing
  - sockets, pipes, message queues, …
- memory based IPC
  - shared memory, mem. mapped files, …

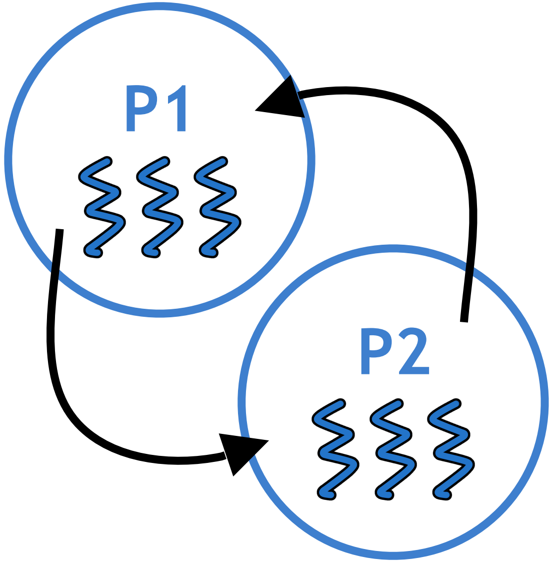


Inter-Process Communication (IPC)

IPC == OS-supported mechanisms for interaction among processes (coordination and communication)

- message passing
  - sockets, pipes, message queues, ...
- memory based IPC
  - shared memory, mem. mapped files, ...
- higher-level semantics
  - files, RPC...

MORE ON THIS LATER

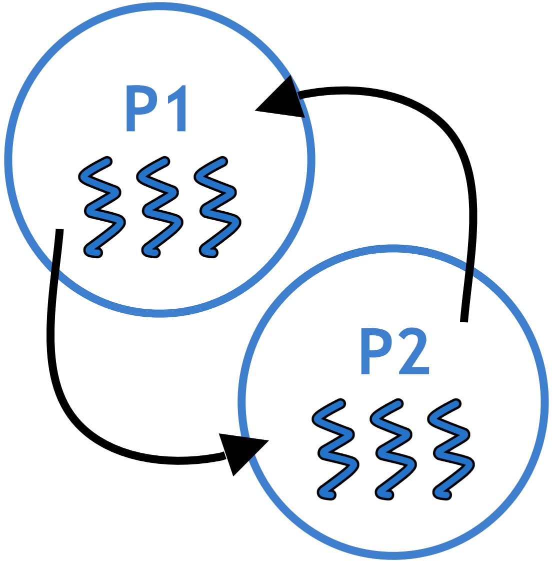


Inter-Process Communication (IPC)

IPC == OS-supported mechanisms for interaction among processes (coordination and communication)

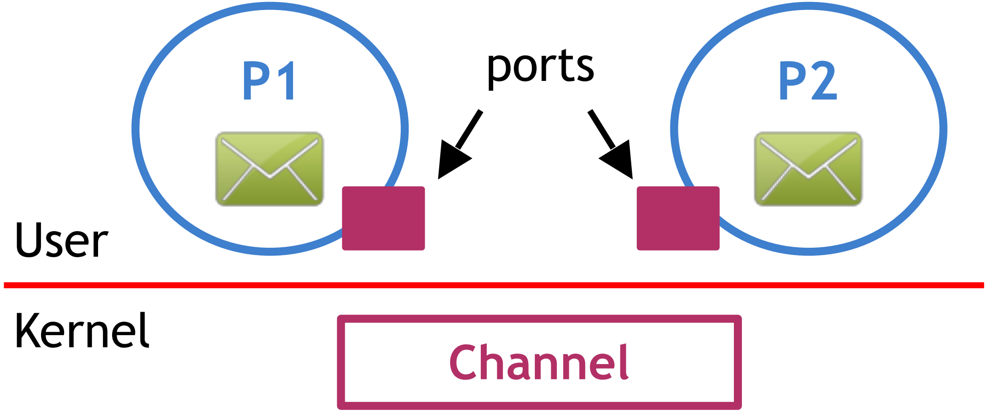
- message passing
  - sockets, pipes, message queues, …
- memory based IPC
  - shared memory, mem. mapped files, …
- higher-level semantics
  - files, RPC…
- synchronization primitives

MORE ON THIS LATER



MESSAGE-BASED IPC

Message-Passing



send/recv of messages

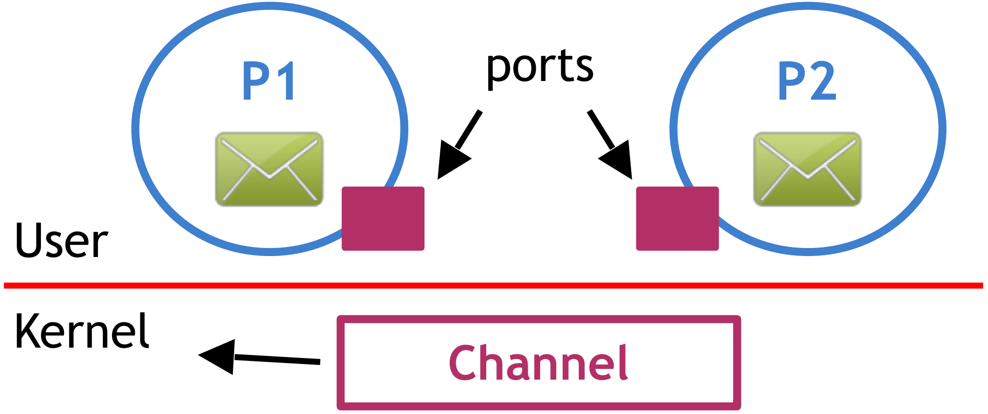
OS creates and maintains a channel

- e.g., buffer, FIFO queue, …

OS provides message interface to processes called a port

- processes send/write messages to port
- processes recv/read messages from port

Message-Passing



Kernel required to

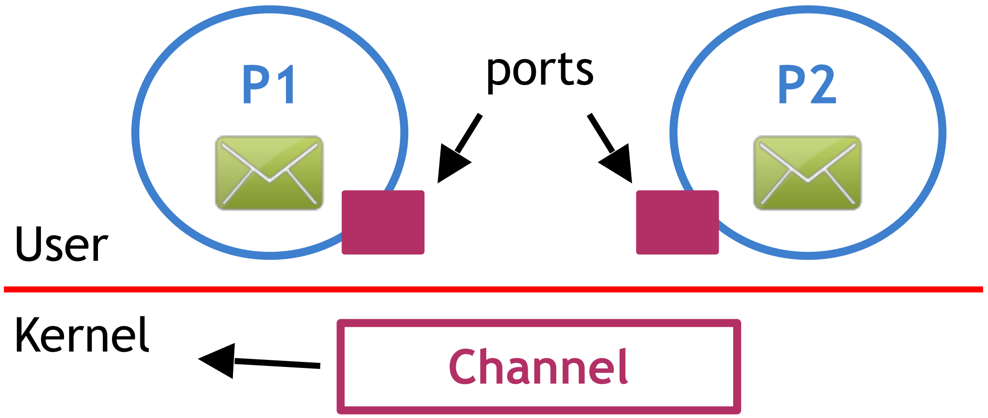
- establish communication
- perform each IPC operation

- send: system call + data copy
- recv: system call + data copy

=> request/response round trip

=> 4x user/kernel crossing + 4x data copy

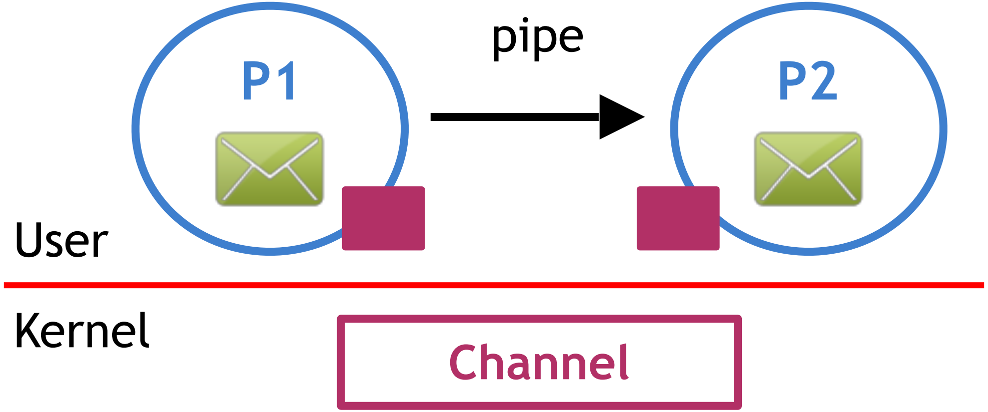
Message-Passing



- overheads
+ simplicity: kernel does channel management and synchronization

FORMS OF MESSAGE PASSING

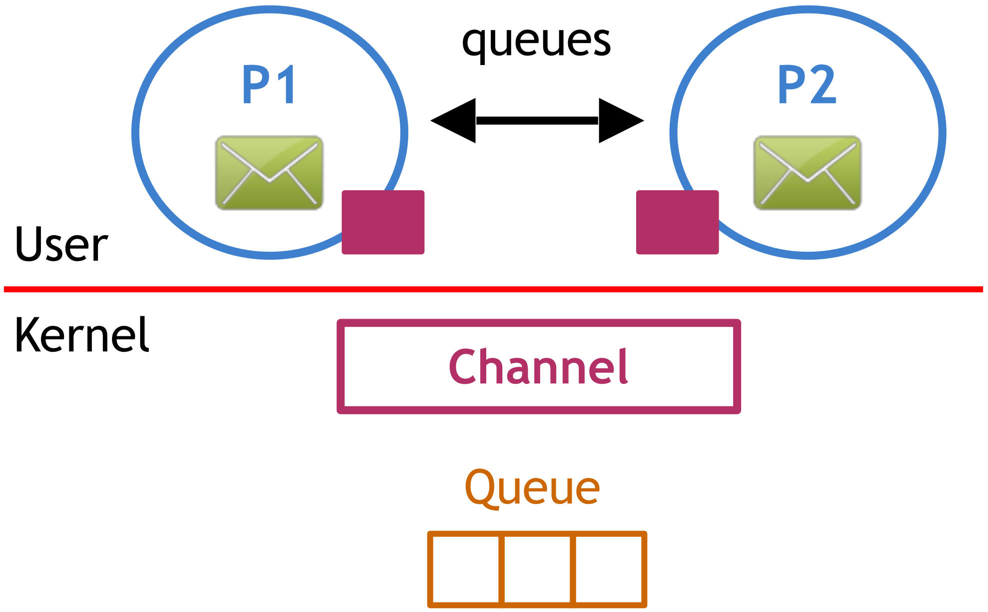
Pipes



Pipes

- carry byte stream between two processes
- e.g., connect output from one process to the input of another

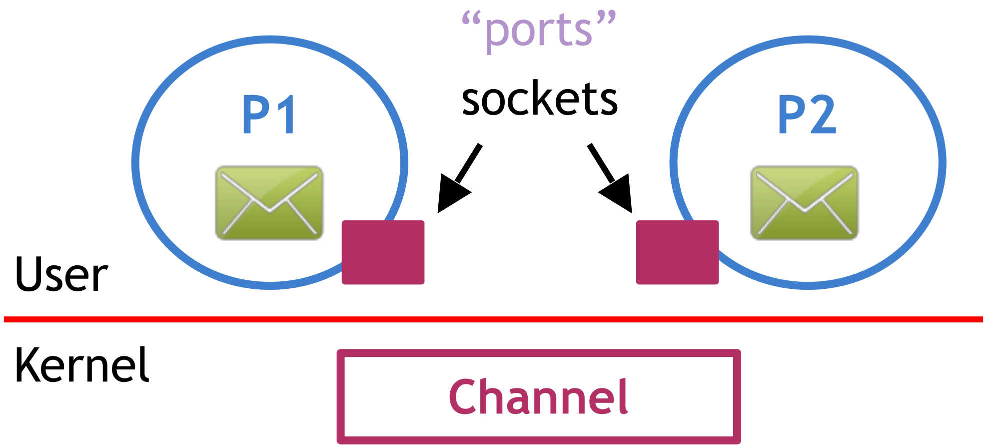
Message Queues



Message Queues

- carry "messages" among processes
- OS management includes priorities, scheduling of message delivery, ...
- Message APIs: System V and POSIX

Sockets



Sockets

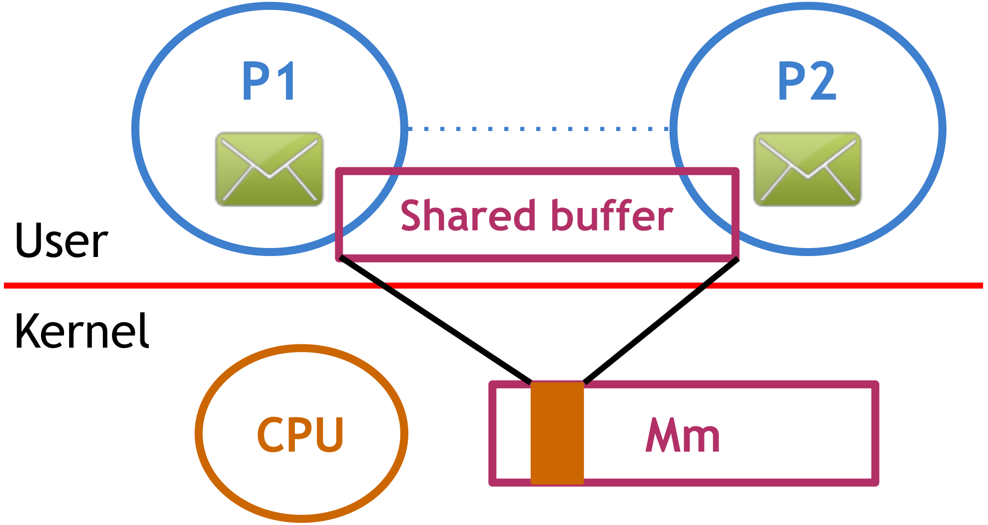
- send(), recv() == pass msg buffers
- socket() == create kernel-level socket buffer
- associate complex in-kernel processing (TCP/IP, …)

=> if different machines, channel between process and network device

=> if same machine, kernel bypasses full protocol stack

SHARED MEMORY IPC

Shared Memory IPC

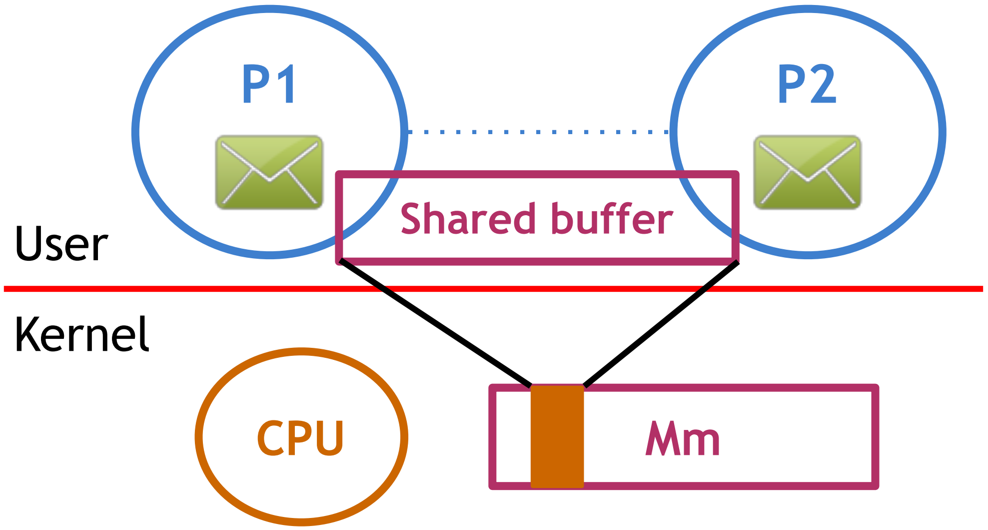


read and write to shared memory region

- OS establishes shared channel between the processes

1. physical pages mapped into each virtual address space
2. VA_P1 and VA_P2 map to the same physical addresses
3. VA_P1 != VA_P2
4. physical memory doesn't have to be contiguous

Shared Memory IPC

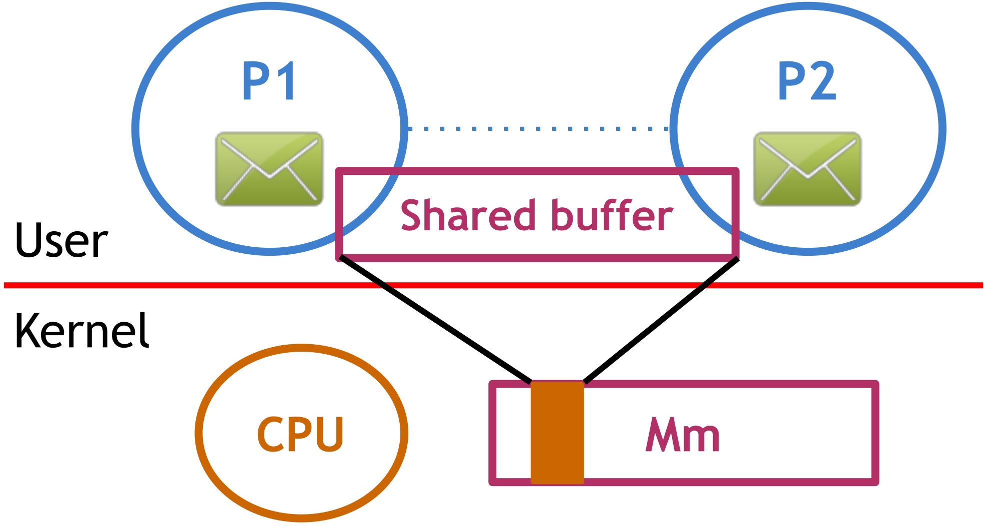


read and write to shared memory region

- OS establishes shared channel between the processes

system calls only for setup
data copies potentially reduced (but not eliminated)

Shared Memory IPC



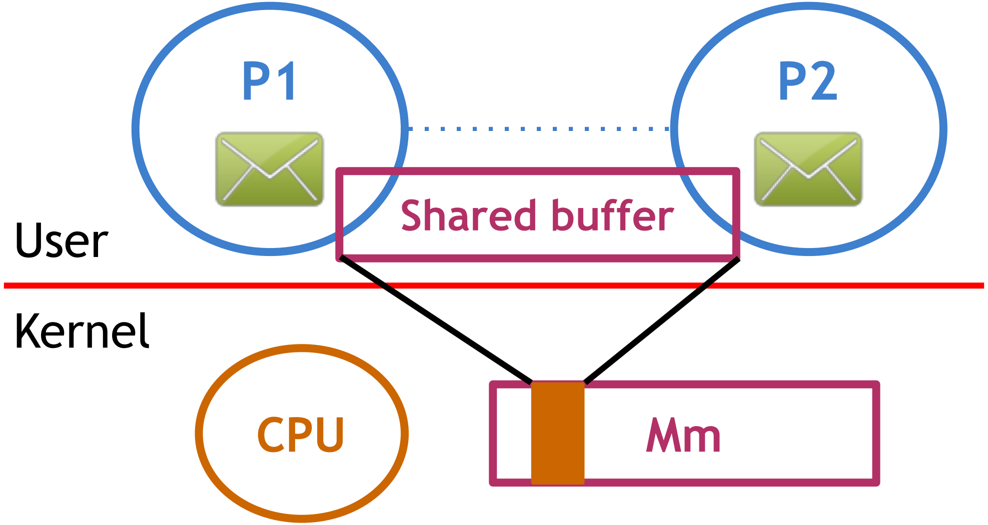
read and write to shared memory region

- OS establishes shared channel between the processes

\+ system calls only for setup
data copies potentially reduced
(but not eliminated)

− explicit synchronization
communication protocol
shared buffer management
programmer responsibility

Shared Memory IPC

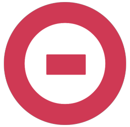


API's: System V API, POSIX API, memory mapped files, Android ashmem

read and write to shared memory region

- OS establishes shared channel between the processes

system calls only for setup
data copies potentially reduced (but not eliminated)



explicit synchronization
communication protocol
shared buffer management
programmer responsibility

IPC COMPARISON QUIZ

IPC Comparison Quiz

Consider using IPC to communicate between processes. You can use either a message-passing or memory-based API. Which one do you think will perform better?

- ○ the message passing
- ○ the shared memory
- ○ neither; it depends

IPC Comparison Quiz

Consider using IPC to communicate between processes. You can use either a message-passing or memory-based API. Which one do you think will perform better?

- the message passing => must send multiple copies!
- the shared memory => must establish shared channel and map it to both address spaces!
- neither; it depends

COPY VS. MAP

Copy (Messages) vs. Map (Shared Mem.)

Goal: transfer data from one into target address space

Copy

Map

- CPU cycles to copy data to/from port

Copy (Messages) vs. Map (Shared Mem.)

Goal: transfer data from one into target address space

Copy

- CPU cycles to copy data to/from port

>

Map

- CPU copy data to channel
- CPU cycles to map memory into address space

=> set up once, use many times => good payoff!

Copy (Messages) vs. Map (Shared Mem.)

Goal: transfer data from one into target address space

Copy

- CPU cycles to copy data to/from port

>

Map

- CPU copy data to channel
- CPU cycles to map memory into address space

=> set up once, use many times => good payoff!
=> can perform well for 1-time use

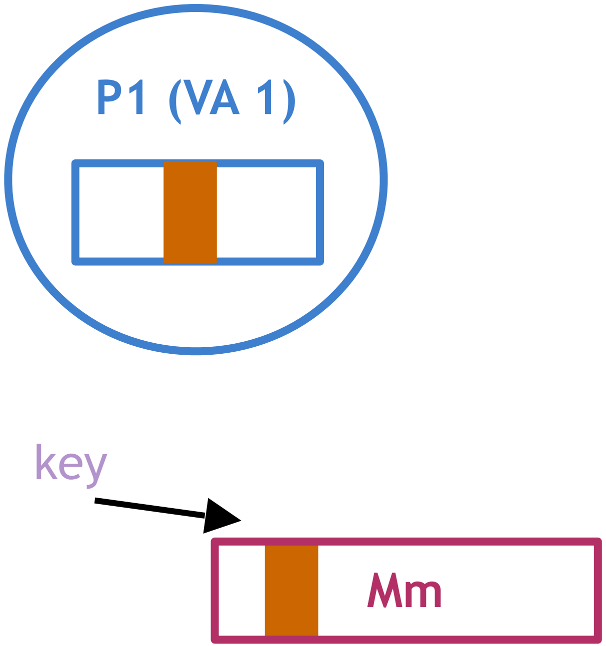
e.g., tradeoff exercised in Windows “Local” Procedure Calls (LPC)

SYSTEM V SHARED MEMORY

System V Shared Memory Overview

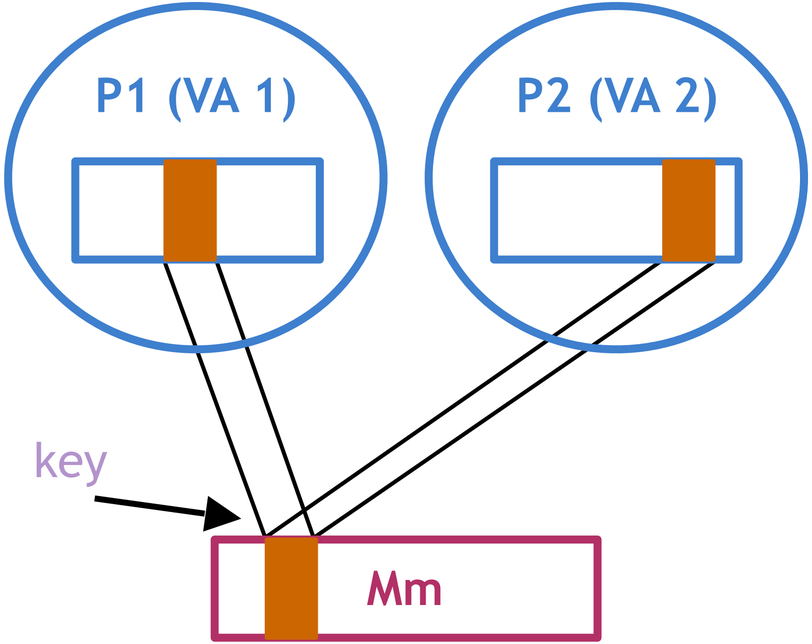
- “segments” of shared memory => not necessarily contiguous physical memory
- shared memory is system wide => system limits on number of segments and size

System V Shared Memory Overview



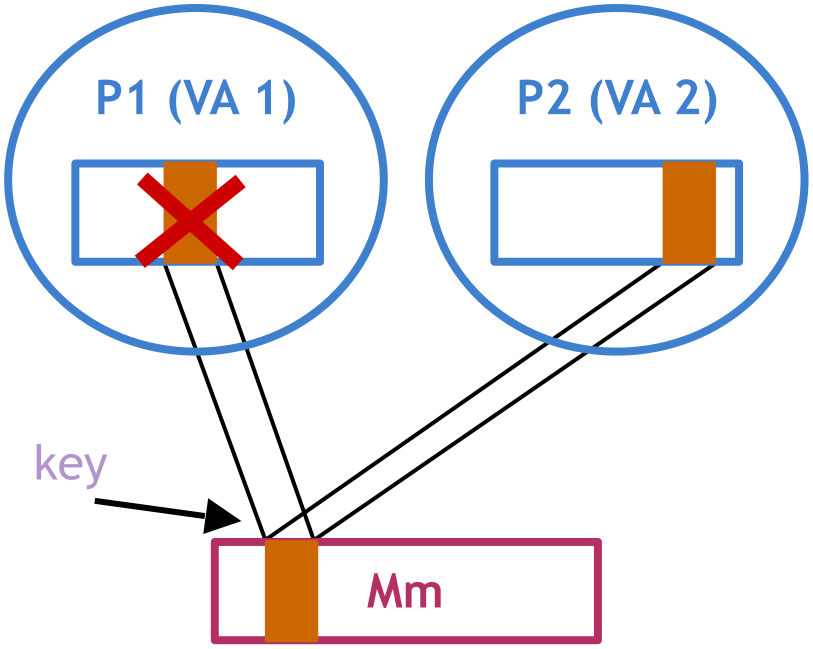
1. Create
   - OS assigns unique key

System V Shared Memory Overview



1. Create
   - OS assigns unique key
2. Attach
   - map virtual => physical addr.

System V Shared Memory Overview



1. Create
   - OS assigns unique key
2. Attach
   - map virtual => physical addr.
3. Detach
   - invalidate address mappings
4. Destroy
   - only remove when explicitly deleted or system reboot

SYSTEM V SHARED MEMORY API

System V Shared Memory API

1. shmget(shmid, size, flags)
   - create or open

   ftok(pathname, proj_id)
   - same args => same key
2. shmat(shmid, addr, flags)
   - addr=NULL => arbitrary
   - cast addr. to arbitrary type
3. shmdt(addr)
4. shmctl(shmid, cmd, buf)
   - destroy with IPC_RMID

1. Create
   - OS assigns unique key
2. Attach
   - map virtual => physical addr.
3. Detach
   - invalidate address mappings
4. Destroy
   - only remove when explicitly deleted or system reboot

POSIX SHARED MEMORY API

POSIX Shared Memory API

segment == file
key == file descriptor

1. shm_open()
   - returns file descriptor
   - in “tmpfs”
2. mmap() and unmap()
   - mapping virtual => physical addr.
3. shm_close()
4. shm_unlink()

1. Create
   - OS assigns unique key
2. Attach
   - map virtual => physical addr.
3. Detach
   - invalidate address mappings
4. Destroy
   - only remove when explicitly deleted or system reboot

SHARED MEMORY AND SYNC

Shared Memory and Sync

"like threads accessing shared state in a single address space…
but for processes"

Synchronization methods...

1. mechanisms supported by process threading library (pthread mutexes and condition variables)
2. OS-supported IPC for synchronization

Either method must coordinate...

- number of concurrent accesses to shared segment
- when data is available and ready for consumption

PTHREAD SYNC FOR IPC

PThreads Sync for IPC

pthread_mutexattr_t
pthread_condattr_t

PTHREAD_PROCESS_SHARED

Synchronization data structure must be shared!

PThreads Sync for IPC

```
// ...make shm data struct
typedef struct {
    pthread_mutex_t mutex;
    char *data;
} shm_data_struct, *shm_data_struct_t;

// ...create shm segment
seg = shmget(ftok(arg[0], 120), 1024, IPC_CREATE|IPC_EXCL));
shm_address = shmat(seg, (void *) 0, 0);
shm_ptr = (shm_data_struct_t_)shm_address;

// ...create and init mutex
pthread_mutexattr_t(&m_attr);
pthread_mutexattr_set_pshared(&m_attr,PTHREAD_PROCESS_SHARED);
pthread_mutex_init(&shm_prt.mutex, &m_attr);
```

SYNC FOR OTHER IPC

Sync for Other IPC

Message Queues

=> implement "mutual exclusion" protocol via send/recv

Example Protocol

- P1 writes to shared memory, sends message to queue
- P2 receives message from queue, then reads from shared memory

Sync for Other IPC

Message Queues

=> implement “mutual exclusion” protocol via send/recv

Example Protocol

- P1 writes to shared memory, sends message to queue
- P2 receives message from queue, then reads from shared memory

Semaphores

=> binary semaphore ⇔ mutex

- if value == 0 => stop/blocked
- if value == 1 => decrement (lock) and go/proceed

MESSAGE QUEUE QUIZ

Message Queue Quiz

Using message queues relies on Linux system calls. Which system call is used to…

- send messages to a message queue?
- receive messages from a message queue?
- perform a message control operation?
- get a message queue identifier?

Note: Use only single word answers such as “reboot” or “recv”. And, please feel free to use the Internet as a resource.

Message Queue Quiz

Using message queues relies on Linux system calls. Which system call is used to…

- send messages to a message queue?
- receive messages from a message queue?
- perform a message control operation?
- get a message queue identifier?

msgsnd

msgrcv

msgctl

msgget

Note: Use only single word answers such as "reboot" or "recv". And, please feel free to use the Internet as a resource.

IPC Command Line Tools

IPC Command Line Tools

ipcs == list IPC facilities created

-m display information about active shared memory segments

ipcrm == delete IPC facilities

-m [shmid] delete shm segment with given id

SHARED MEM DESIGN CONSIDERATIONS

Shared Mem Design Considerations

=> different APIs/mechanisms to sync...

=> OS provides shared memory, and then is out of the way...

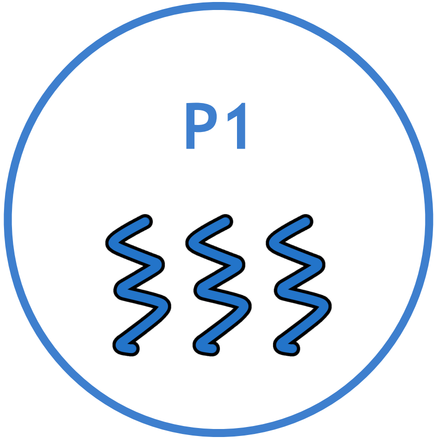
=> data passing/sync protocols are up to the programmer...

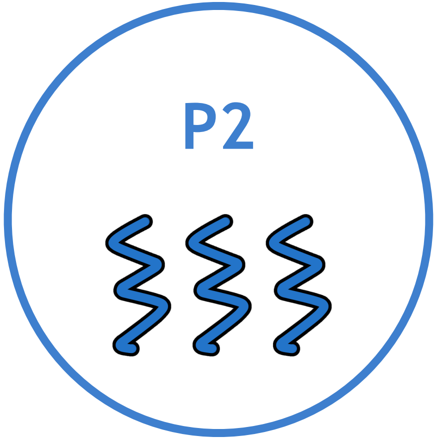
elf dressed as spiderman

"With great power comes great responsibility"

HOW MANY SEGMENTS?

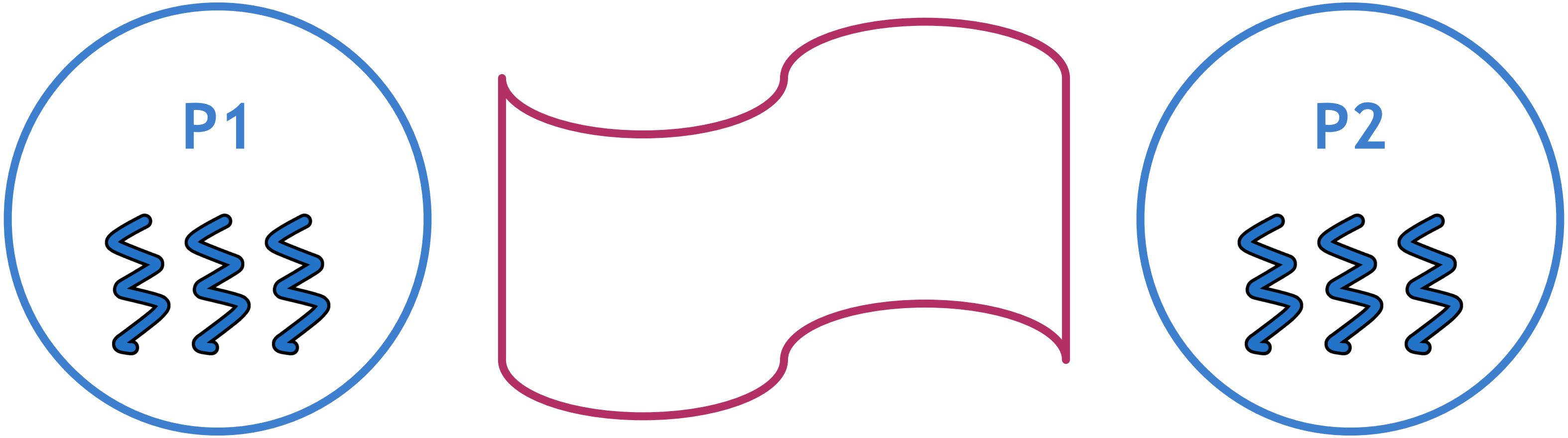
Shared Mem Design Considerations





Shared Mem Design Considerations

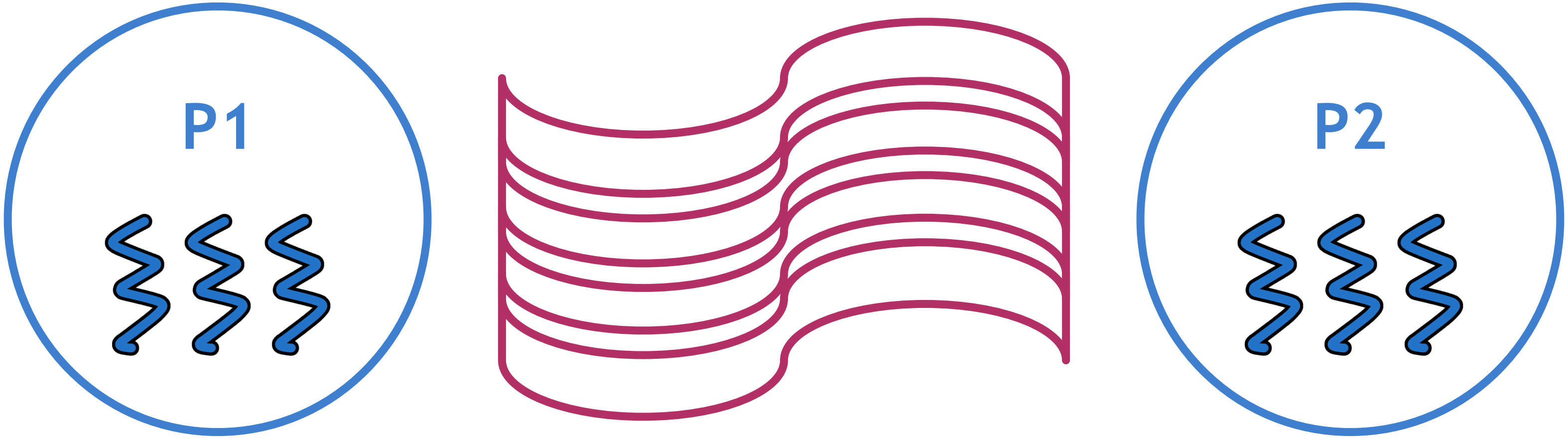
How many segments?



1 large segment => manager for allocating/freeing mem. from shared segment

Shared Mem Design Considerations

How many segments?



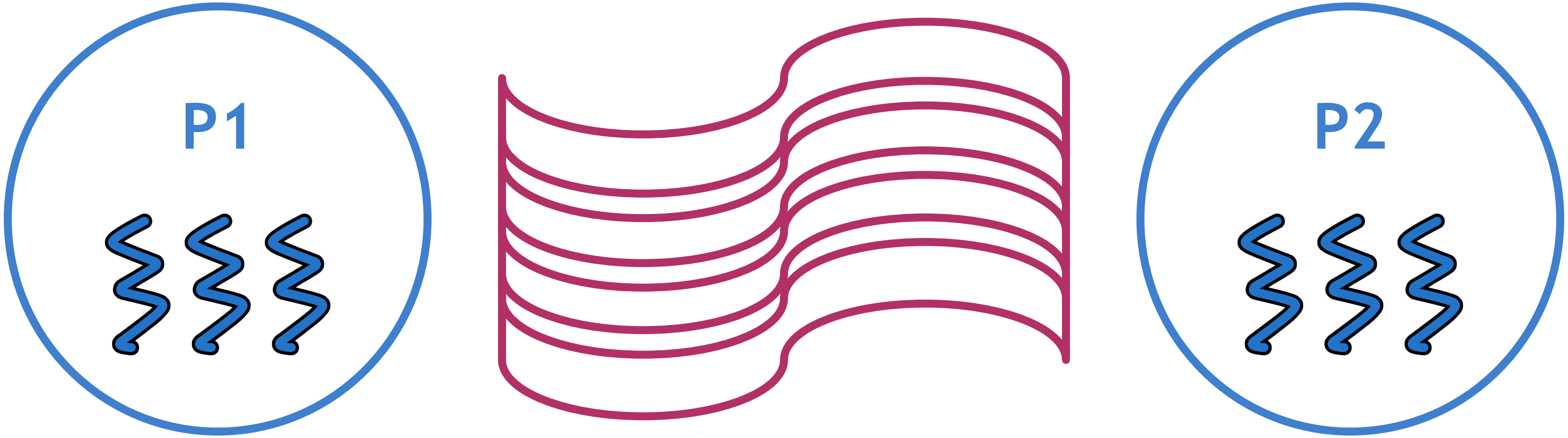
1 large segment => manager for allocating/freeing mem. from shared segment

many small segments => use pool of segments => queue of segment IDs
=> communicate segment IDs between processes

WHAT SIZE SEGMENTS?

Shared Mem Design Considerations

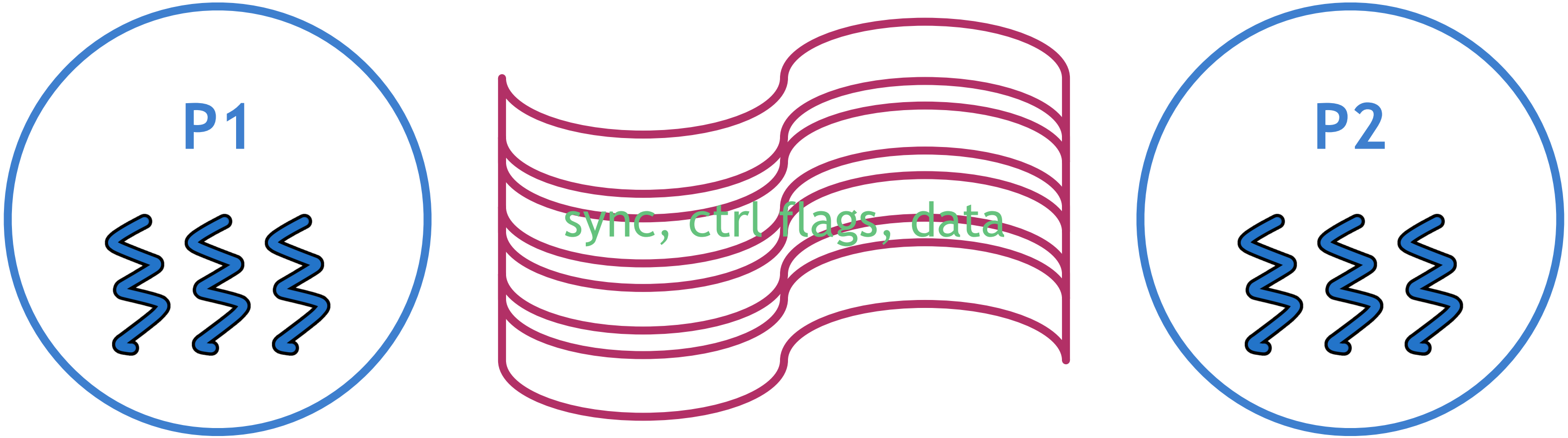
What size segments? What if data does not fit?



segment size == data size => works for well-known static msg sizes; limit on total size of data transfer

Shared Mem Design Considerations

What size segments? What if data does not fit?



segment size == data size => works for well-known static msg sizes; limit on total size of data transfer

segment size < message size => transfer data in rounds; include protocol for tracking