Deadlock wrap-up

Interprocess Communication

CS 241

March 31, 2014

University of Illinois

Cheating vs. collaborating

Cheating

- Copying code, pseudo-code, flow charts
- Writing someone else's code line by line

Not cheating

- Discussing high-level approaches
- Discussing MP requirements, C language, tools
- Helping each other with debugging

Does this mean I can't help my friend?

- No, but don't solve their problems for them

Deadlock Detection & Recovery

Deadlock Detection

Goal: Check to see if a deadlock has occurred

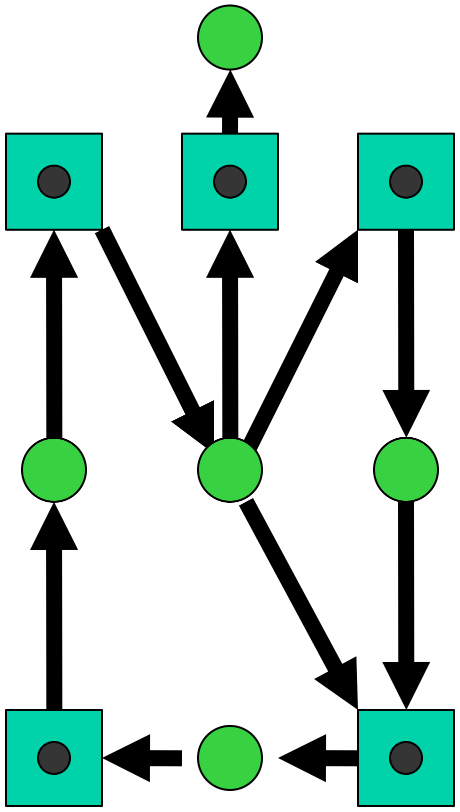
Special case: Single resource per type

- E.g., mutex locks (value is zero or one)
- Check for cycles in the resource allocation graph

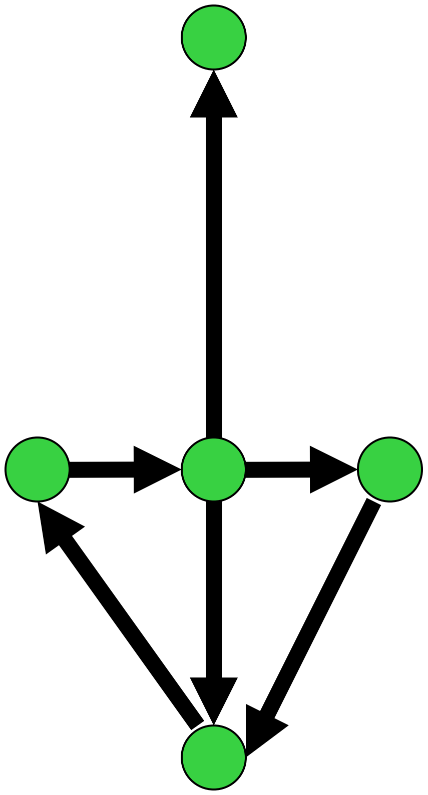
General case

- E.g., semaphores, memory pages, ...
- See book, p. 355 – 358

Dependencies between processes



Resource allocation graph



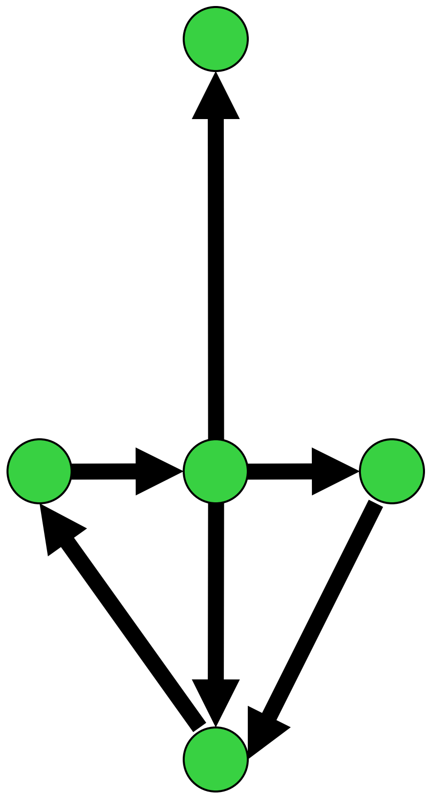
Corresponding process dependency graph

Deadlock Recovery

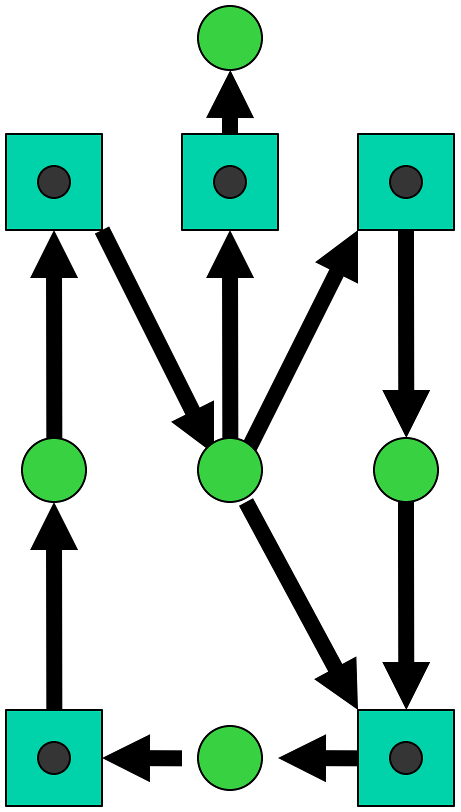
Recovery idea: get rid of the cycles in the process dependency graph

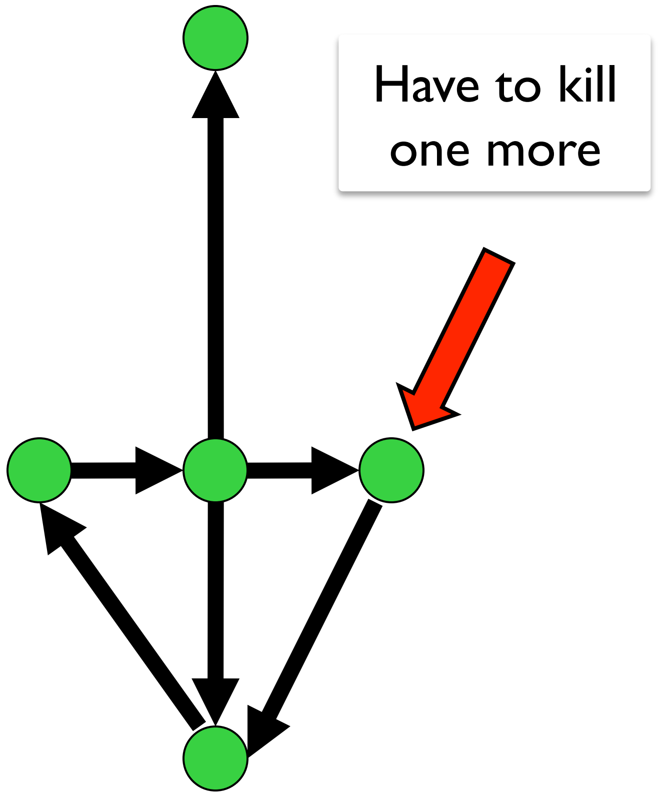
Options:

- Kill all deadlocked processes
- Kill one deadlocked process at a time and release its resources
- Steal one resource at a time
- Roll back all or one of the processes to a checkpoint that occurred before they requested any resources, then continue
  - Difficult to prevent indefinite postponement



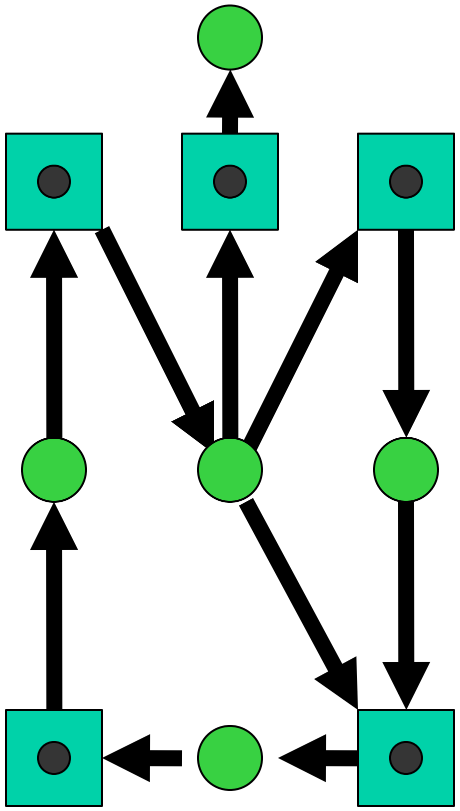
Deadlock Recovery


Resource allocation graph

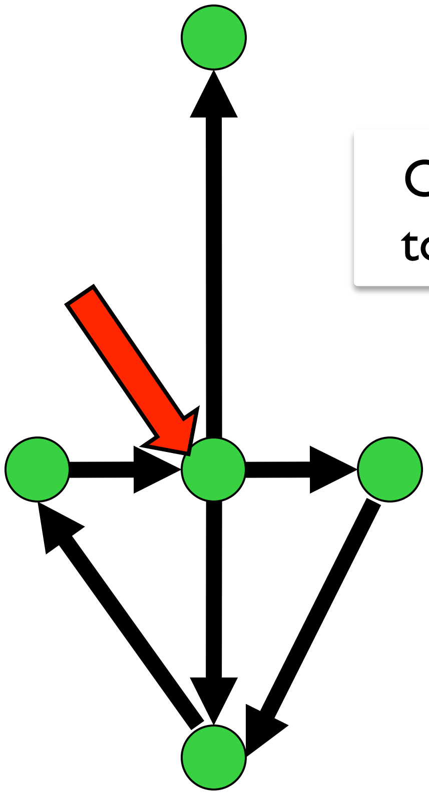


Corresponding process dependency graph

Deadlock Recovery



Resource allocation graph



Only have to kill one

Corresponding process dependency graph

Deadlock Recovery

How should we pick a process to kill?

We might consider...

- process priority
- current computation time and time to completion
- amount of resources used by the process
- amount of resources needed by the process to complete
- the minimal set of processes we need to eliminate to break deadlock
- is process interactive or batch?

Rollback instead of killing processes

Selecting a victim

- Minimize cost of rollback (e.g., size of process's memory)

Rollback

- Return to some safe state
- Restart process for that state
- Note: Large, long computations are sometimes checkpointed for other reasons (reliability) anyway

Challenge: Starvation

- Same process may always be picked as victim
- Fix: Include number of rollbacks in cost factor

Deadlock Summary

Deadlock: cycle of processes/threads each waiting for the next

- Nasty timing-dependent bugs!

Detection & Recovery

- Typically very expensive to kill / checkpoint processes

Avoidance: steer around deadlock

- Requires knowledge of everything an application will request
- Expensive to perform on each scheduling event

Prevention (ordered resources)

- Imposes conservative rules on application that preclude deadlock
- Application can do it; no special OS support

Deadlock Summary

Typical solution:

- OS (Unix/Windows) do nothing (Ostrich Algorithm)
- Application uses general-purpose deadlock prevention

Transaction systems (e.g., credit card processing) may use detection/recovery/avoidance

Where we are in 241

C basics

Memory

Processes

Threads

Scheduling

Synchronization

Interprocess communication

Networking

Filesystems

Interprocess Communication

Interprocess Communciation

What is IPC?

- Mechanisms to transfer data between processes

Why is it needed?

- Not all important procedures can be easily built in a single process

Interprocess Communication

Cooperating processes

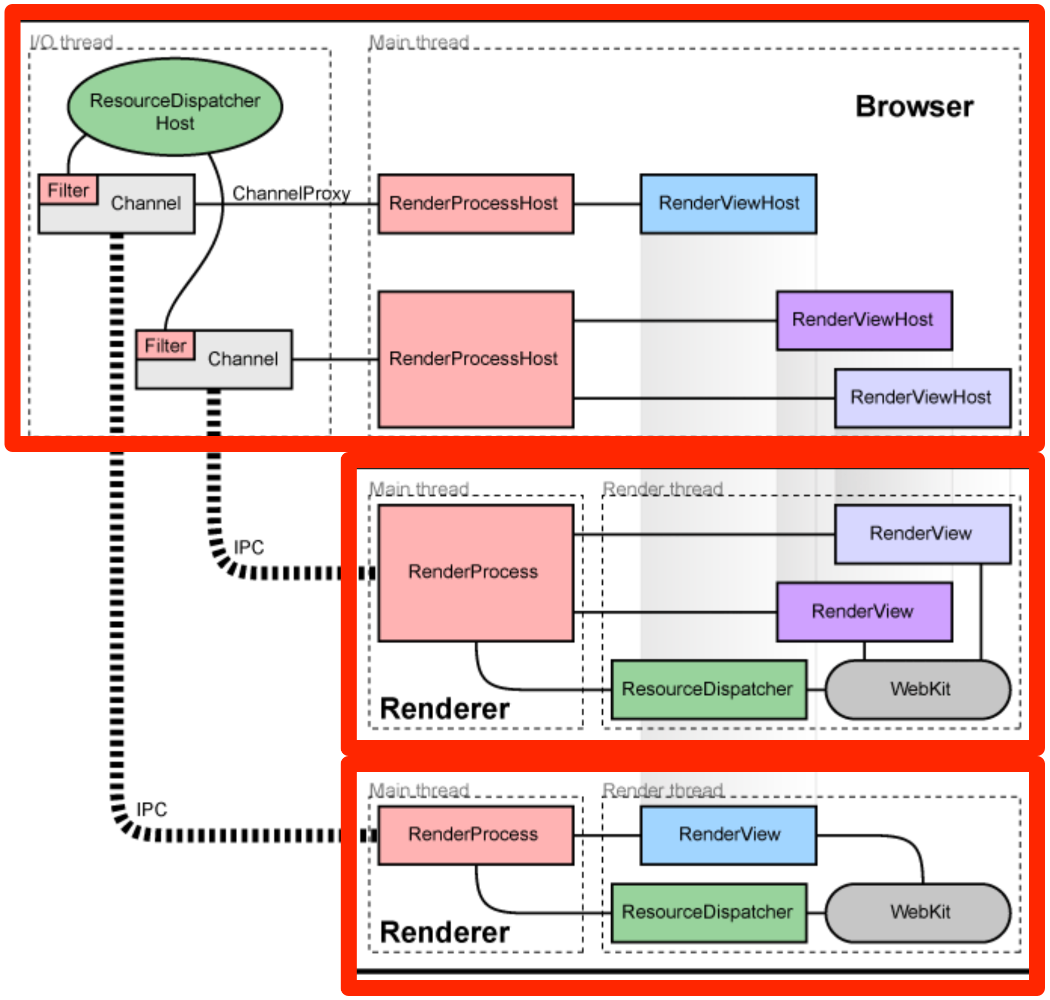
- Can affect or be affected by other processes, including sharing data
  - Just like cooperating threads!
- Benefits
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience

Interprocess Communication

Can you think of a common use of IPC?

Can you think of any large applications that use IPC?

Google Chrome architecture (figure borrowed from Google)

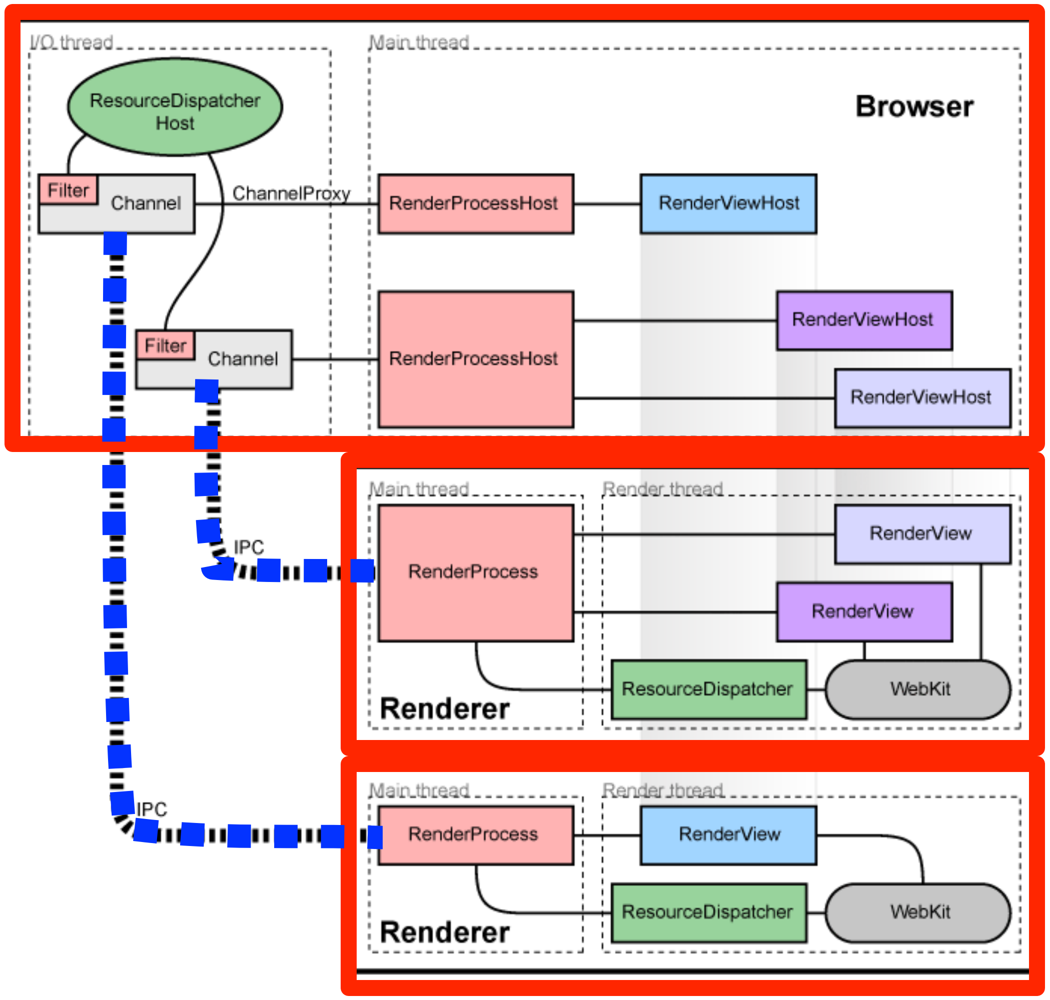


Separate processes for browser tabs to protect the overall application from bugs and glitches in the rendering engine

Restricted access from each rendering engine process to others and to the rest of the system

https://sites.google.com/a/chromium.org/dev/developers/design-documents/multi-process-architecture

Google Chrome architecture (figure borrowed from Google)

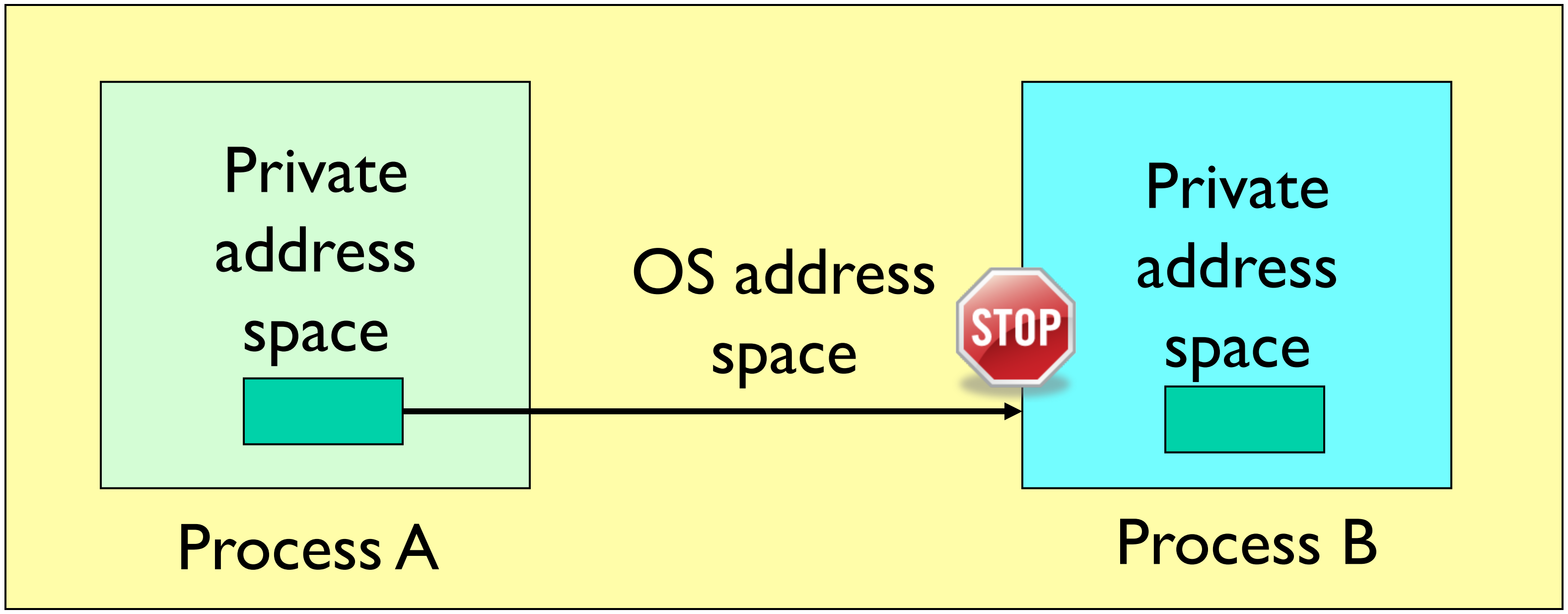


A named pipe is allocated for each renderer process for communication with the browser process

Pipes are used in asynchronous mode to ensure that neither end is blocked waiting for the other

https://sites.google.com/a/chromium.org/dev/developers/design-documents/multi-process-architecture

IPC Communications Model



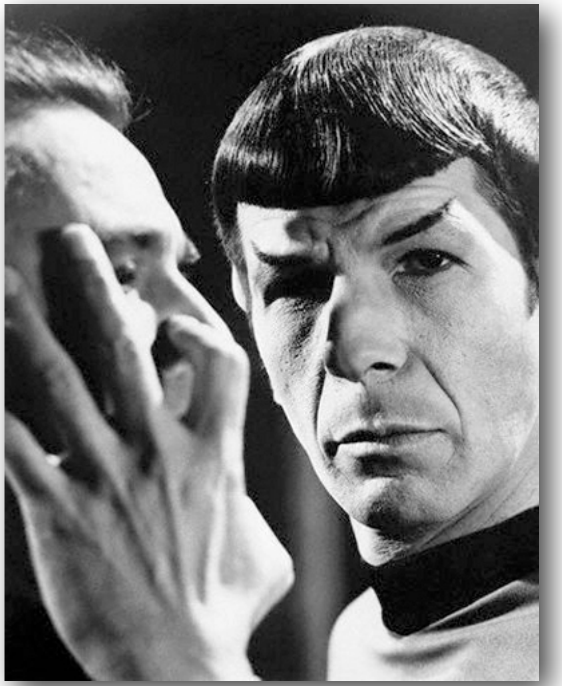
Each process has a private address space

No process can write to another process's space

How can we get data from process A to process B?

Two kinds of IPC

“Mind meld”

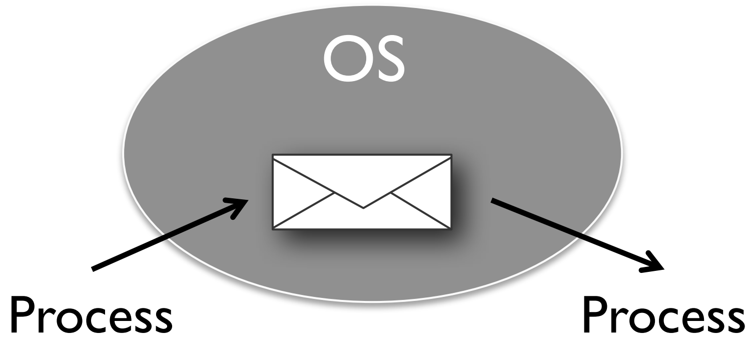


Shared address space

- Shared memory
- Memory mapped files

Today

“Intermediary”



Message transported by OS from one address space to another

- Files
- Pipes
- FIFOs

Shared memory

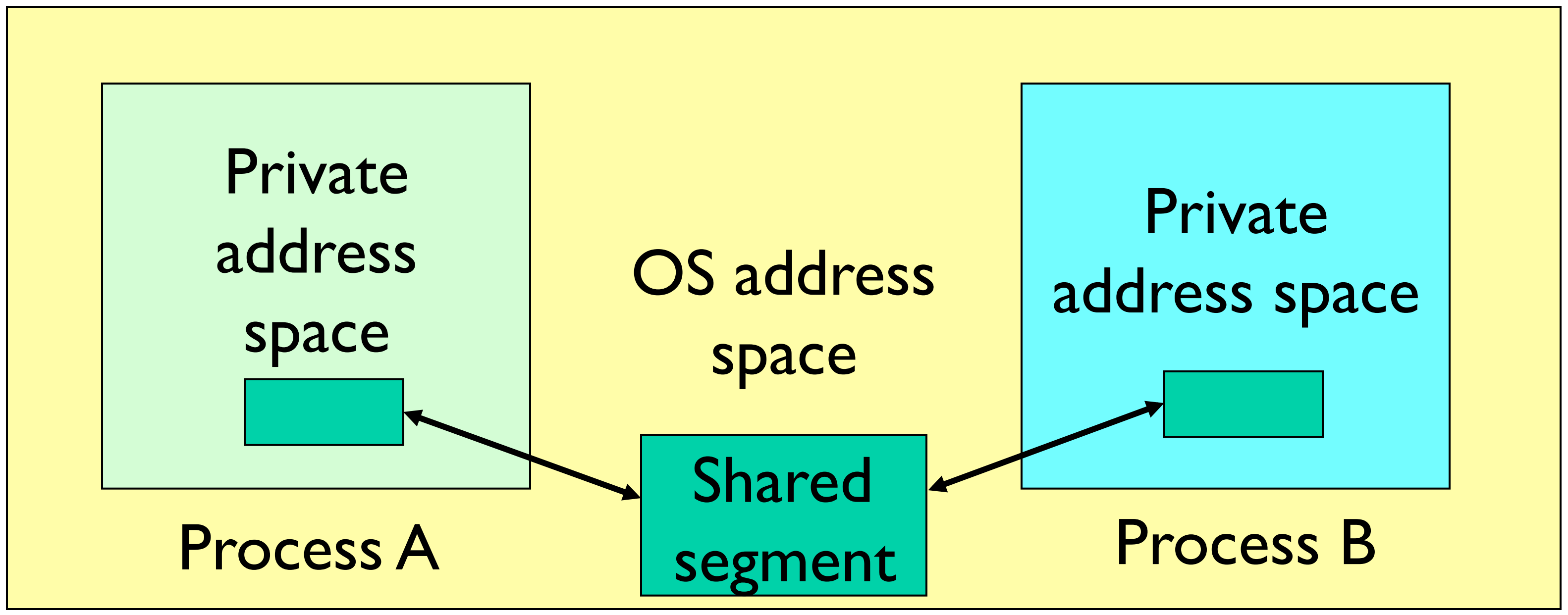
Processes share the same segment of memory directly

- Memory is mapped into the address space of each sharing process
- Memory is persistent beyond the lifetime of the creating or modifying processes (until deleted)

Does not provide mutual exclusion

- Processes using the shared memory must do this on their own

Shared Memory

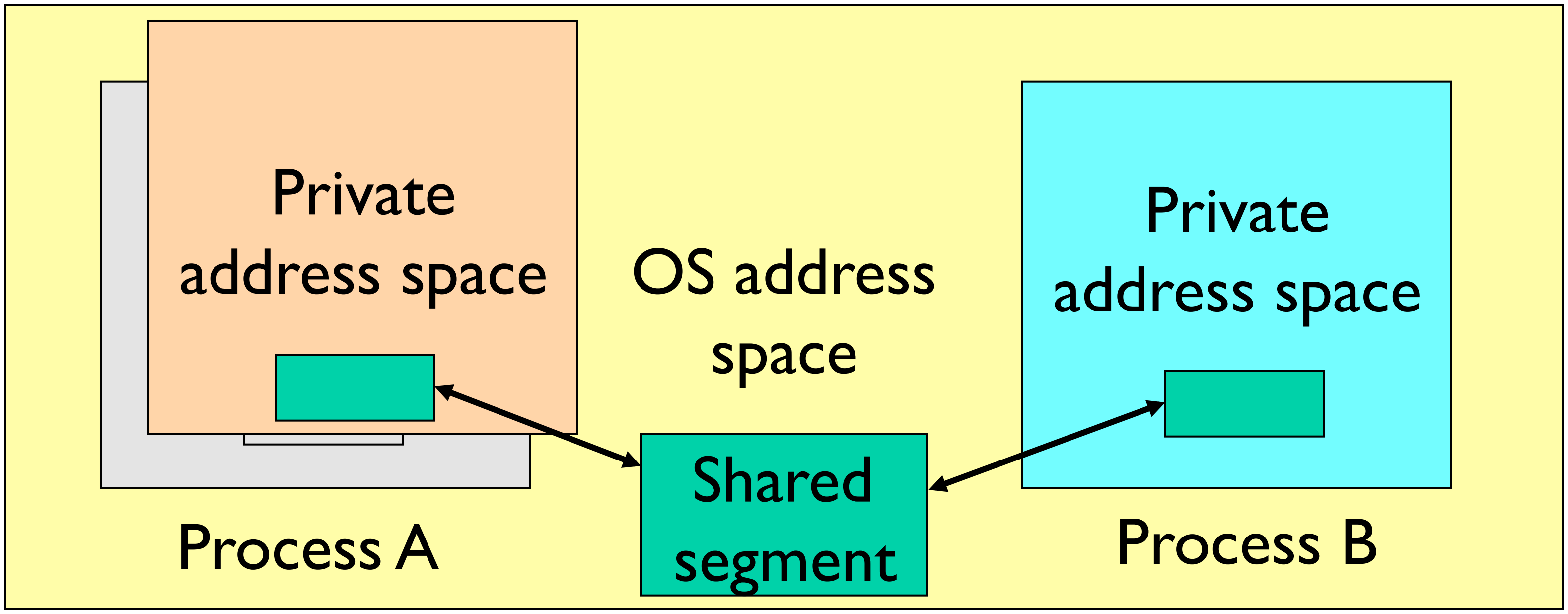


Processes request the segment

OS maintains the segment

Processes can attach/detach the segment

Shared Memory



Can mark segment for deletion on last detach

POSIX Shared Memory

```
#include <sys/types.h>

#include <sys/shm.h>
```

Create identifier (“key”) for a shared memory segment

```
key_t ftok(const char *pathname, int proj_id);
k = ftok(“/my/file”, 0xaa);
```

Create shared memory segment

```
int shmget(key_t key, size_t size, int shmflg);
id = shmget(key, size, 0644 | IPC_CREAT);
```

Access to shared memory requires an attach

```
void *shmat(int shmid, const void *shmaddr, int shmflg);

shared_memory = (char *) shmat(id, (void *) 0, 0);
```

POSIX Shared Memory

Write to the shared memory using normal system calls

```
sprintf(shared_memory, "Writing to shared
  memory");
```

Detach the shared memory from its address space

```
int shmdt(const void *shmaddr);
shmdt(shared_memory);
```

Shared Memory example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define SHM_SIZE 1024  /* a 1K shared memory segment */

int main(int argc, char *argv[]) {
   key_t key;
   int shmid;
   char *data;
   int mode;
```

Shared Memory example

```
/* make the key: */
if ((key = ftok(”shmdemo.c", 'R')) == -1) {
    perror("ftok");
    exit(1);
}
/* connect to (and possibly create) the segment: */
if ((shmid = shmget(key, SHM_SIZE, 0644 | IPC_CREAT)) == -1) {
    perror("shmget");
    exit(1);
}
/* attach to the segment to get a pointer to it: */
data = shmat(shmid, (void *)0, 0);
if (data == (char *)(-1)) {
    perror("shmat");
    exit(1);
}
```

Shared Memory example

```
    /* read or modify the segment, based on the command line: */
    if (argc == 2) {
        printf("writing to segment: \"%s\"\n", argv[1]);
        strncpy(data, argv[1], SHM_SIZE);
    } else
        printf("segment contains: \"%s\"\n", data);

    /* detach from the segment: */
    if (shmdt(data) == -1) {
        perror("shmdt");
        exit(1);
    }

    return 0;
}
```

Run demo