

---

# Sistemas Operativos I

*“Operating systems are like underwear—nobody really wants to look at them”*

—BILL JOY

C

2022

Rafael Ignacio Zurita <[rafa@fi.uncoma.edu.ar](mailto:rafa@fi.uncoma.edu.ar)>

---

**Advertencia:** Estos slides traen ejemplos.

**No copiar (ctrl+c) y pegar en un shell o terminal los comandos aquí presentes.**

Algunos no funcionarán, porque al copiar y pegar también van caracteres “ocultos” (no visibles pero que están en el pdf) que luego interfieren en el shell.

Sucedio en vivo :)

Conviene “escribirlos” manualmente al trabajar.

# Sistemas Operativos I - Comunicación entre procesos

---

## Contenido:

- pipes
- memoria compartida
- sockets
- señales
- pasaje de mensajes
- archivos

# Sistemas Operativos I - Comunicación entre procesos

---

Motivación:

Procesos son independientes, y están aislados por el sistema operativo y el hardware.

# Sistemas Operativos I - Comunicación entre procesos

---

Motivación:

Procesos son independientes, y están aislados por el sistema operativo y el hardware.

Peero, muchas aplicaciones requieren comunicarse para trabajar cooperativamente.

Sean INDEPENDIENTES o NO (EN UN SENTIDO LÓGICO)

# Sistemas Operativos I - Comunicación entre procesos

---

Motivación: Ejemplo de aplicaciones **INDEPENDIENTES**

El usuario desea conocer :

- todos los archivos
- que implementen el system call open
- ordenados por tamaño
- y que en la salida no haya líneas duplicadas.

# Sistemas Operativos I - Comunicación entre procesos

---

Motivación: Ejemplo de aplicaciones **INDEPENDIENTES**

El usuario desea conocer :

- todos los archivos
- que implementen el system call open
- ordenados por tamaño
- y que en la salida no haya líneas duplicadas.

```
# ls device/*/* | grep open | xargs du -sk | sort -n | uniq
```

# Sistemas Operativos I - Comunicación entre procesos

## Motivación: Ejemplo de aplicaciones COOPERATIVAS NATIVAMENTE

chromium.org/developers/design-documents/multi-process-architecture/



Actualizar

restricted.

### Architectural overview

We use separate processes for browser tabs to protect the overall application from bugs and glitches in the rendering engine. We also restrict access from each rendering engine process to others and to the rest of the system. In some ways, this brings to web browsing the benefits that memory protection and access control brought to operating systems.

We refer to the main process that runs the UI and manages tab and plugin processes as the "browser process" or "browser." Likewise, the tab-specific processes are called "render processes" or "renderers." The renderers use the [Blink](#) open-source layout engine for interpreting and laying out HTML.

Chrome has a [multi-process architecture](#) and each process is heavily multi-threaded. In this document we will go over the basic threading system shared by each process. The main goal is to keep the main thread (a.k.a. "UI" thread in the browser process) and IO thread (each process's thread for receiving [IPC](#)) responsive. This means offloading any blocking I/O or other expensive operations to other threads. Our approach is to use message passing as the way of communicating between threads. We discourage locking and thread-safe objects. Instead, objects live on only one (often virtual - we'll get to that later!) thread and we pass messages between those threads for communication. Absent external requirements about latency or workload, Chrome attempts to be a [highly concurrent, but not necessarily parallel](#), system.



# Sistemas Operativos I - Comunicación entre procesos

---

## Motivación: Ejemplo de aplicaciones **COOPERATIVAS NATIVAMENTE**

chromium.org/developers/design-documents/multi-process-architecture/



Actualizar

restricted.

### Architectural overview

We use separate processes for browser tabs to protect the overall application from bugs and glitches in the rendering engine. We also restrict access from each rendering engine process to others and to the rest of the system. In some ways, this brings to web browsing the benefits that memory protection and access control brought to operating systems.

We refer to the main process that runs the UI and manages tab and plugin processes as the "browser process" or "browser." Likewise, the tab-specific processes are called "render processes" or "renderers." The renderers use the [Blink](#) open-source layout engine for interpreting and laying out HTML.

Chrome has a [multi-process architecture](#) and each process is heavily multi-threaded. In this document we will go over the basic threading system shared by each process. The main goal is to keep the main thread (a.k.a. "UI" thread in the browser process) and IO thread (each process's thread for receiving IPC) responsive. This means offloading any blocking I/O or other expensive operations to other threads. Our approach is to use message passing as the way of communicating between threads. We discourage locking and thread-safe objects. Instead, objects live on only one (often virtual - we'll get to that later!) thread and we pass messages between those threads for communication. Absent external requirements about latency or workload, Chrome attempts to be a [highly concurrent, but not necessarily parallel](#), system.

**Chrome utiliza pasaje de mensajes como mecanismo de comunicación entre sus procesos**

# Sistemas Operativos I - Comunicación entre procesos

Principios: existen dos mecanismos

- memoria compartida
- pasaje de mensajes

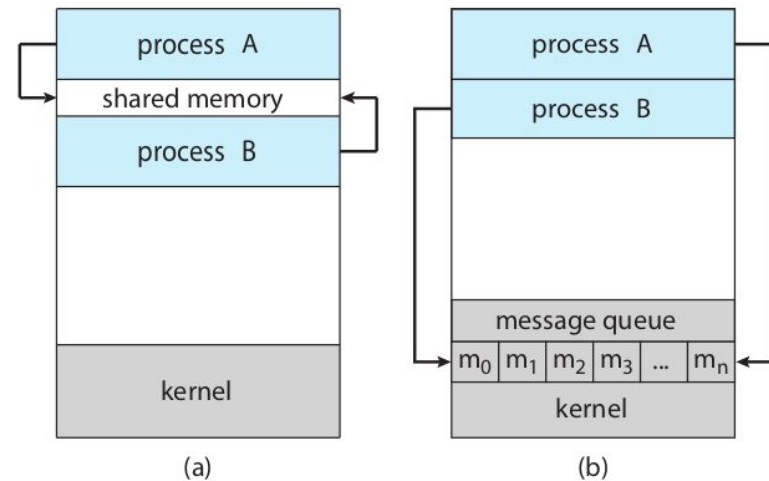


Figure 3.11 Communications models. (a) Shared memory. (b) Message passing.

# Sistemas Operativos I - Comunicación entre procesos

---

## Ejemplo de **memoria compartida** : posix shared memory (UNIX)

*POSIX shared memory is organized using memory-mapped files, which associate the region of shared memory with a file.*

NAME  
shm\_open, shm\_unlink - create/open or unlink POSIX shared memory objects

SYNOPSIS  
#include <sys/mman.h>  
#include <sys/stat.h> /\* For mode constants \*/  
#include <fcntl.h> /\* For O\_\* constants \*/  
  
int shm\_open(const char \*name, int oflag, mode\_t mode);  
  
int shm\_unlink(const char \*name);  
  
Link with -lrt.

DESCRIPTION  
shm\_open() creates and opens a new, or opens an existing, POSIX shared memory object. A POSIX shared memory object is in effect a handle which can be used by unrelated processes to mmap(2) the same region of shared memory. The shm\_unlink() function performs the converse operation, removing an object previously created by shm\_open().

# Sistemas Operativos I - Comunicación entre procesos

## Ejemplo de **memoria compartida** : posix shared memory (UNIX)

*POSIX shared memory is organized using memory-mapped files, which associate the region of shared memory with a file.*

```
/* productor */

int main()
{
    const int SIZE = 4096;
    const char *name = "OS";
    const char *mess0= "Studying ";
    const char *mess1= "Operating Systems ";

    int shm_fd;
    void *ptr;

    /* create the shared memory segment */
    shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);

    /* configure the size of the shared memory segment */
    ftruncate(shm_fd,SIZE);

    /* now map the shared memory segment in the address space of the process */
    ptr = mmap(0,SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (ptr == MAP_FAILED) {
        printf("Map failed\n");
        return -1;
    }

    /* Now write to the shared memory region. */
    sprintf(ptr,"%s",mess0);
    ptr += strlen(mess0);
    sprintf(ptr,"%s",mess1);

    return 0;
}
```

```
/* consumidor */

int main()
{
    const char *name = "OS";
    const int SIZE = 4096;

    int shm_fd;
    void *ptr;
    int i;

    /* open the shared memory segment */
    shm_fd = shm_open(name, O_RDONLY, 0666);
    if (shm_fd == -1) {
        printf("shared memory failed\n");
        exit(-1);
    }

    /* now map the shared memory segment in the address space of the process */
    ptr = mmap(0,SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);
    if (ptr == MAP_FAILED) {
        printf("Map failed\n");
        exit(-1);
    }

    /* now read from the shared memory region */
    printf("%s",(char *)ptr);

    /* remove the shared memory segment */
    if (shm_unlink(name) == -1) {
        printf("Error removing %s\n",name);
        exit(-1);
    }

    return 0;
}
```

# Sistemas Operativos I - Comunicación entre procesos

## Principios de pasaje de mensajes

## Decisiones de diseño a nivel del OS

- ¿Son los mensaje de tamaño fijo o variable?
- ¿Cuántos mensajes pueden estar pendientes?
- ¿Dónde se almacenan los menajes?
- ¿Cómo se especifica el receptor?
- ¿Conoce el receptor la identidad del emisor?
- ¿Será una comunicación bidireccional?
- ¿Es la interfaz síncrona? ¿o asíncrona?

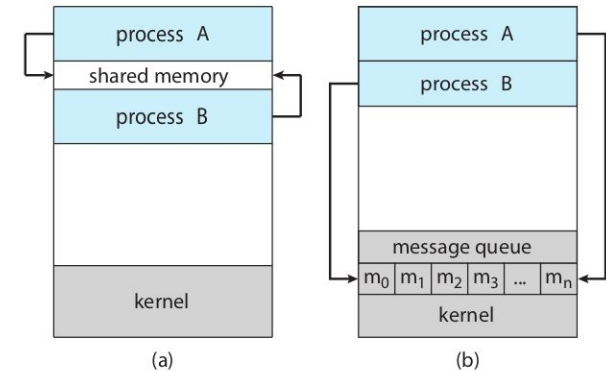


Figure 3.11 Communications models. (a) Shared memory. (b) Message passing.

# Sistemas Operativos I - Comunicación entre procesos

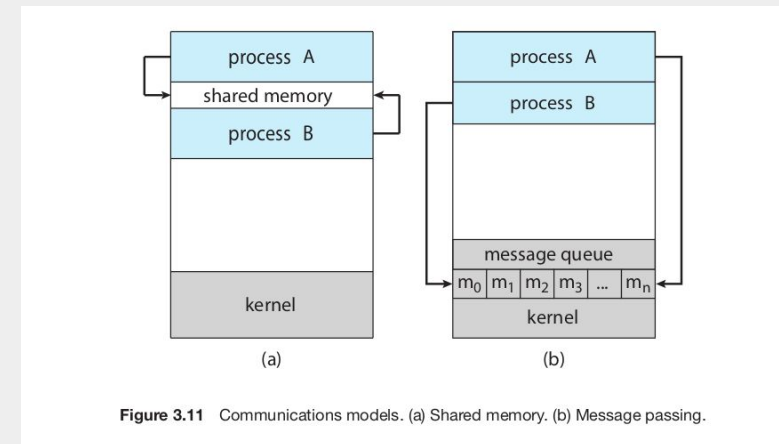
## Principios de pasaje de mensajes- Decisiones de diseño a nivel del OS

### Interfaz síncrona

- La llamada al sistema bloquea hasta que finalice la operación
- Fácil de entender y programar
- Procesos intermediarios podrían ser utilizados para implementar en espacio de usuario una interfaz asíncrona.

### Interfaz asíncrona

- Se inicia una operación de envío o recepción
- El OS permite continuar la ejecución
- El OS debe implementar algún mecanismo de notificación



# Sistemas Operativos I - Comunicación entre procesos

Ejemplo de **pasaje de mensajes** : Xinu

Mecanismo de bajo nivel sencillo

Comunicación directa (un proceso a otro)

Los mensajes son de una PALABRA

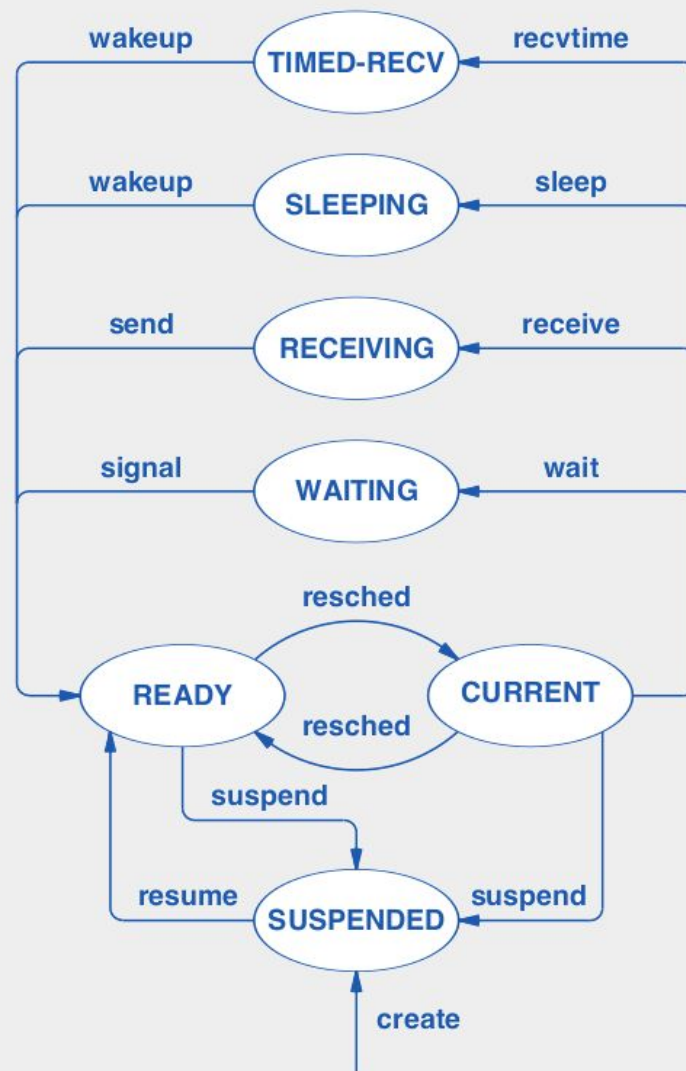
El buffer de implementación es de un mensaje

Síncrono para recepción

Asíncrono para transmisión y reset

```
/* API */
```

```
send(msg, pid);  
msg = receive(); /* bloqueante */  
msg = recvclr();
```



# Sistemas Operativos I - Comunicación entre procesos

---

Implementaciones a nivel de sistemas operativos tradicionales:

## Ejemplo UNIX

- **IPC System V :**
  - **cola de mensajes,**
  - **memoria compartida,**
  - **arreglo de semáforos**
- **BSD sockets: unix domain e internet domain** (Windows también)
- **pipes y archivos FIFOs** (Windows también)
- **Señales UNIX**
- **Memoria compartida (POSIX)** (Windows también)



# Sistemas Operativos I - Comunicación entre procesos

---

Implementaciones a nivel de sistemas operativos tradicionales:

## Ejemplo UNIX

- Señales UNIX : Ejecutamos un proceso: `find /`

```
rafa@gabideb ~ $ ps -ef | grep find
rafa          204763  204647 17 10:41 pts/12    00:00:00 find /
```

```
rafa@gabideb ~ $ kill -SIGSTOP 204763
# proceso DETENIDO
```

```
rafa@gabideb ~ $ kill -SIGCONT 204763
# proceso ahora CONTINUA
```

```
rafa@gabideb ~ $ kill -SIGKILL 204763
rafa@gabideb ~ $
```