Sistemas Operativos I

"Operating systems are like underwear—nobody really wants to look at them"
—BILL JOY

Rafael Ignacio Zurita <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 tambien van caracteres "ocultos" (no visibles pero que están en el pdf) que luego interfieren en el shell.

Sucedió 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

Los procesos son independientes.
Están aislados por el sistema operativo y el hardware.

Sistemas Operativos I - Comunicación entre procesos

Motivación

Los procesos son independientes.
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)

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

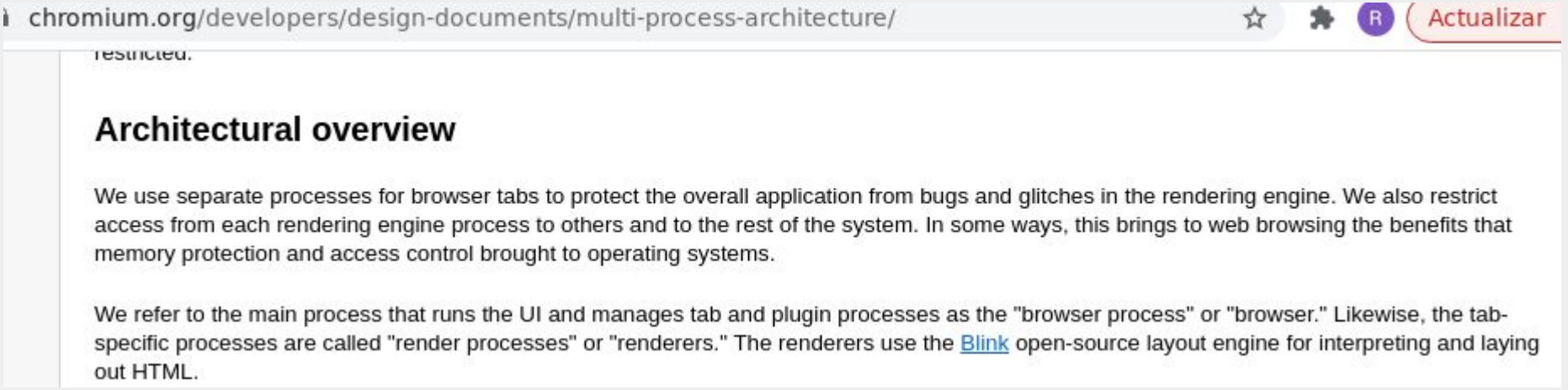
El usuario desea conocer :

- 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

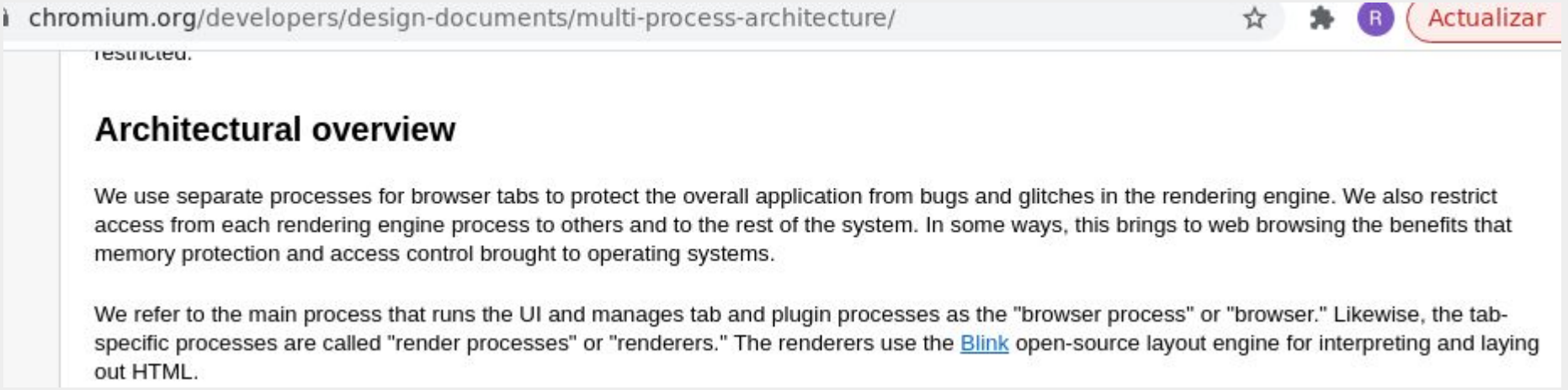
Motivación: Ejemplo de aplicaciones COOPERATIVAS NATIVAMENTE



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



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

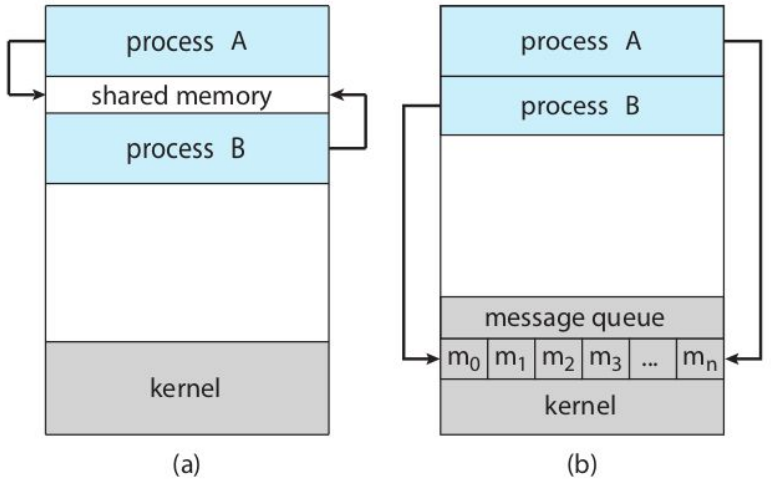


Figure 3.11 Communications models. (a) Shared memory. (b) Message passing.

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?

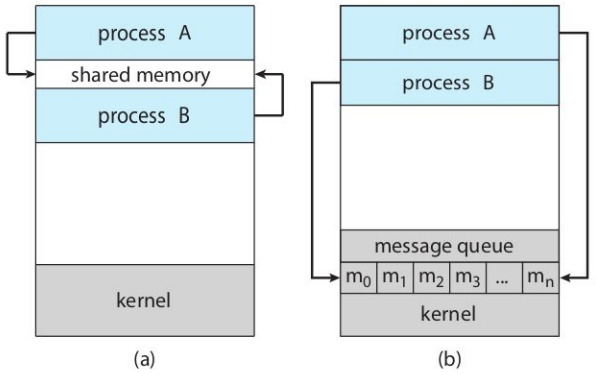


Figure 3.11 Communications models. (a) Shared memory. (b) Message passing.

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

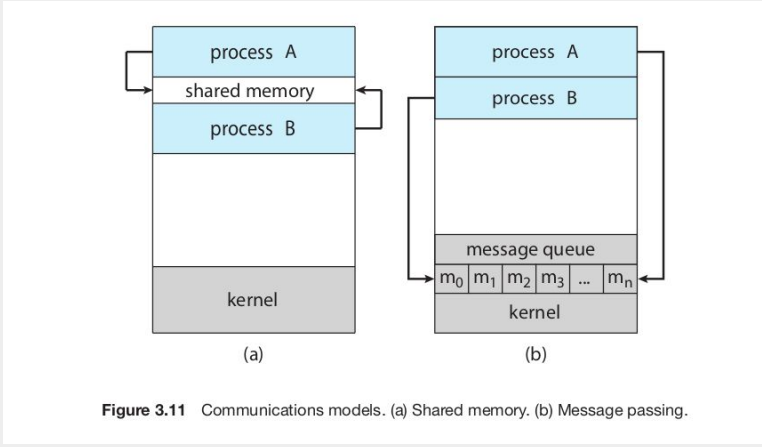


Figure 3.11 Communications models. (a) Shared memory. (b) Message passing.

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

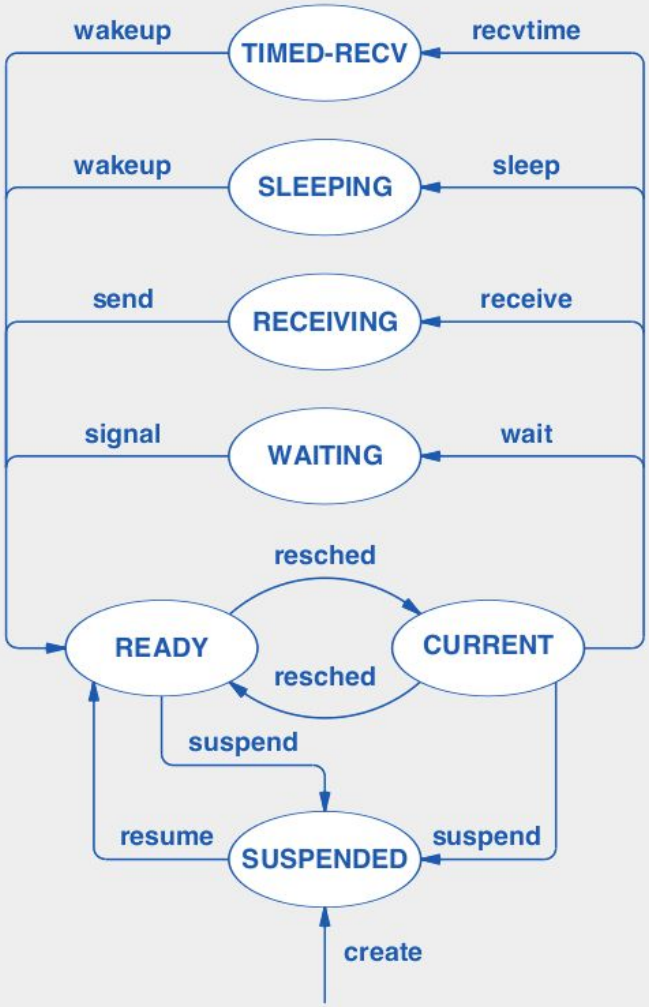
Mecanismo de bajo nivel sencillo
Comunicación directa (un proceso a otro)

Los mensajes son de una PALABRA

El buffer de implementación es de un mensaje

Síncrono para recepción

Asíncrono para transmición y reset




```
/* API */

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

Sistemas Operativos I - Comunicación entre procesos

Implementaciones a nivel de sistemas operativos tradicionales:

Ejemplos en 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
# el proceso recibe la señal de DETENERSE

rafa@gabideb ~ $ kill -SIGCONT 204763
# el proceso recibe la señal de CONTINUAR


rafa@gabideb ~ $ kill -SIGKILL 204763
# el proceso recibe la señal de que debe FINALIZAR
```

Diferentes clases de Sistemas Operativos

- Sistemas Operativos Distribuidos
- Sistemas Operativos Embebidos
- Sistemas Operativos de Tiempo Real

Sistemas Operativos Distribuidos

Un Sistema Operativo Distribuido
administra los recursos de N sistemas de cómputo,
aunque presenta una interfaz al usuario transparente
como si fuese un sistema uniprocesador.

Sistemas Operativos Distribuidos

Un Sistema Operativo Distribuido
administra los recursos de N sistemas de cómputo,
aunque presenta una interfaz al usuario transparente
como si fuese un sistema uniprocesador.

- **transparencia**
- **tolerante a fallas**
- **mismo kernel en todos los sistemas**
- **llamadas al sistema**

Sistemas Operativos Embebidos

Un Sistema Operativo Embebido que administra y presenta un conjunto mínimo de recursos que necesita la aplicación embebida.

-

Sistemas Operativos Embebidos

Un Sistema Operativo Embebido que administra y presenta un conjunto mínimo de recursos que necesita la aplicación embebida.

- **conjunto básico de drivers**
- **conjunto mínimo de procesos de sistema**
- **conjunto mínimo de recursos**

Sistemas Operativos Embebidos

Un Sistema Operativo Embebido que administra y presenta un conjunto mínimo de recursos que necesita la aplicación embebida.

- Android??? iOS??
- QNX
- VxWorks
- uCos
- FreeRTOS
- NetBSD
- Windows Embebido
- Linux Embebido
    - Openwrt/Armbian/Raspbian
    - OpenEmbedded (yocto)
    - BuildRoot (busybox)

Sistemas Operativos Real-Time

Un Sistema ~~Operativo~~ con gestor de tareas y herramientas para la sincronización y comunicación entre tareas. Es predecible y determinista.

Su modelo permite definir un sistema con requerimientos de tiempos de respuesta límites.

Sistemas Operativos Real-Time

Un Sistema ~~Operativo~~ con gestor de tareas y herramientas para la sincronización y comunicación entre tareas. Es predecible y determinista.

Su modelo permite definir un sistema con requerimientos de tiempos de respuesta límites.

- **VxWorks**
- **uCos**
- **FreeRTOS**