Sistemas Operativos I

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

—BILL JOY

Clase: threads - comunicación inter procesos

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.

Contenido

- Threads
- Comunicación inter procesos
- Ejemplos
  - Linux
  - XINU

Sistemas Operativos I - Repaso de desalojo de CPU

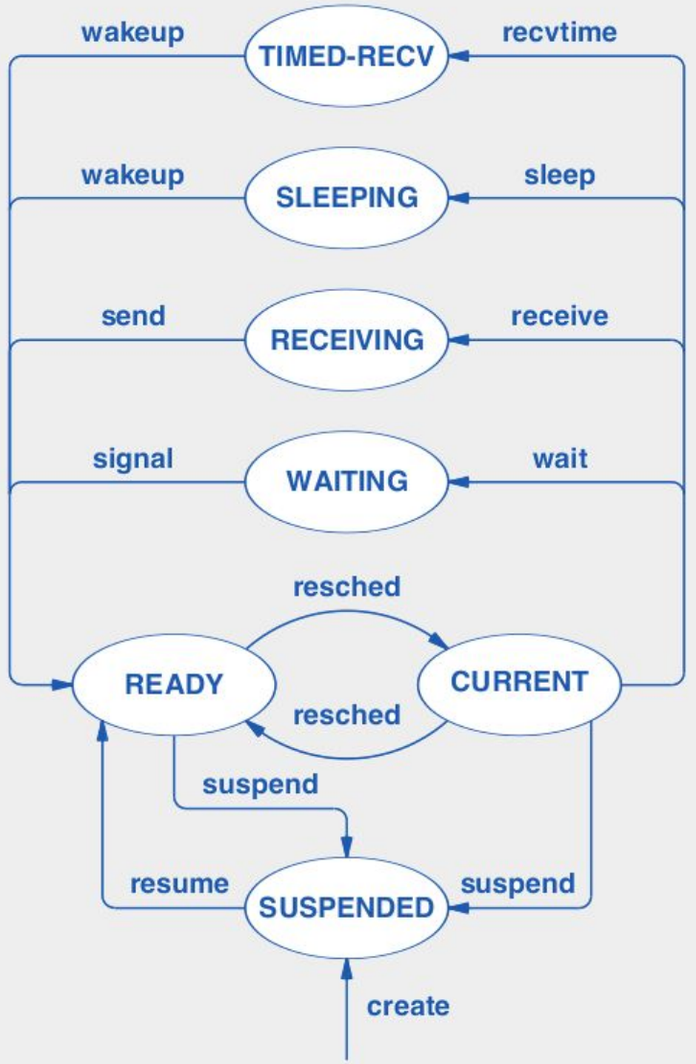
```
/* Ejemplo de programa en XINU */
#include <xinu.h>

char saludo[] = “Hola mundo”;

void    main(void)
{
    int pid;

    pid = create(sndA, 128, 20, "process 1", 0) );
    resume(pid);
    resume(create(sndA, 128, 20, "process 2", 0) ));
    sleep(10);
    kill(pid);
}

void    sndA(void)
{
    char t[80];
    memcpy(t, saludo, strlen(saludo)+1);
    while( 1 )
                printf(“%s! \n”, t);
}
```

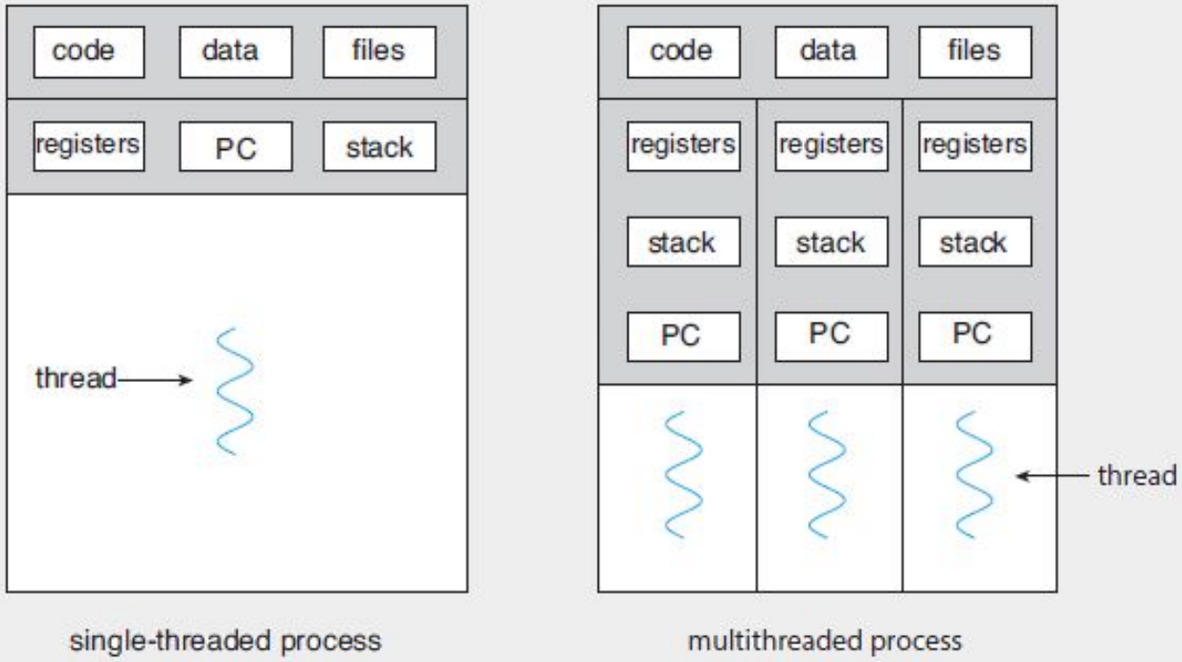


¿Cuántos procesos hay en ejecución? - ¿Y cuando termina main, finaliza process 2?

¿Cuáles son llamadas al sistema y cuales funciones de la biblioteca de C?

Threads

- Unidad básica de utilización de la CPU
- Tiene su propio ID, PC, registros, y stack
- Comparte la sección de código, datos
  y otros recursos (como archivos abiertos).

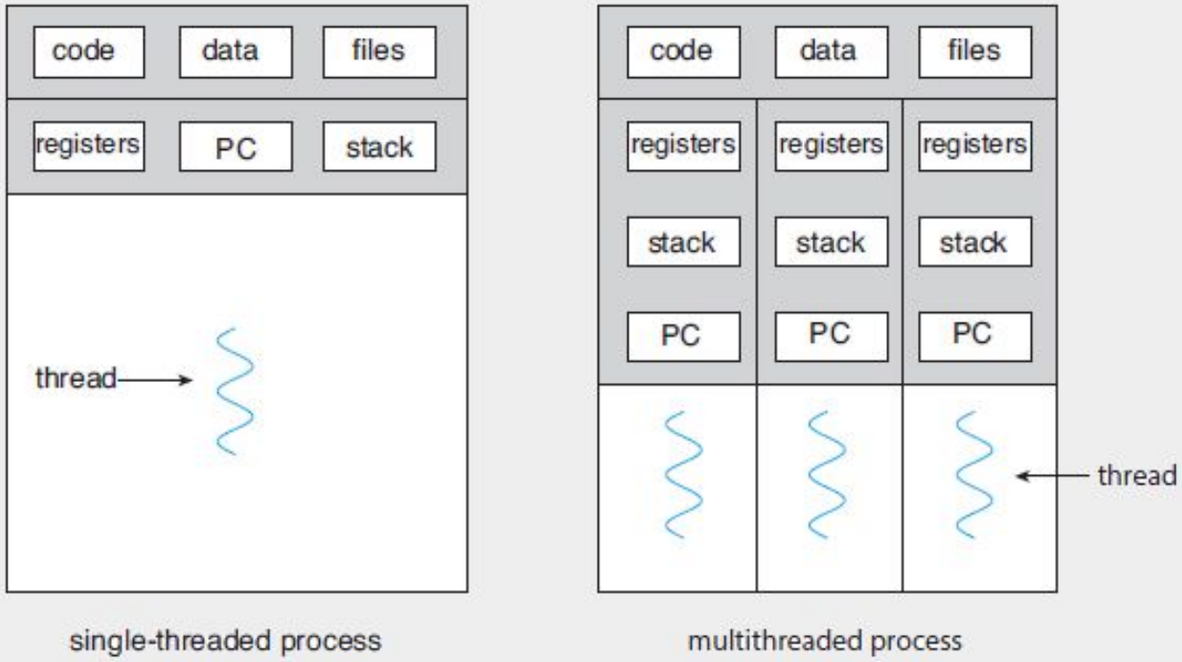


Threads

- **Unidad básica de utilización de la CPU**
- **Tiene su propio ID, PC, registros, y stack**
- **Comparte la sección de código, datos**
  y otros recursos (como archivos abiertos).

Otras denominaciones:

- Hilo de control
- Proceso de peso liviano (LWP)
- Thread de control



single-threaded process

multithreaded process

Un thread NO ES UN PROCESO HIJO

Threads

Si...

- n procesos en un sistema con m CPU son concurrentes/paralelos
  (comparten recursos del SO -dispositivos lógicos, semáforos, etc-)

Si...

- n threads en un proceso con m CPU son concurrentes/paralelos
  (comparten recursos del proceso)

Threads

Si...

- n procesos en un sistema con m CPU son concurrentes/paralelos
  (comparten recursos del SO -dispositivos lógicos, semáforos, etc-)

Si...

- n threads en un proceso con m CPU son concurrentes/paralelos
  (comparten recursos del proceso)

Entonces...¿tiene algún sentido tener THREADS?????

Threads

Si...

- n pro

(

Si...

- n thr

(

Enton

-)

????

Threads

Algunos posibles beneficios:

- **Capacidad de respuesta**
- **Compartir recursos**

  Menor overhead en:

  la creación y finalización de threads

  el cambio de contexto.
- **Escalabilidad**

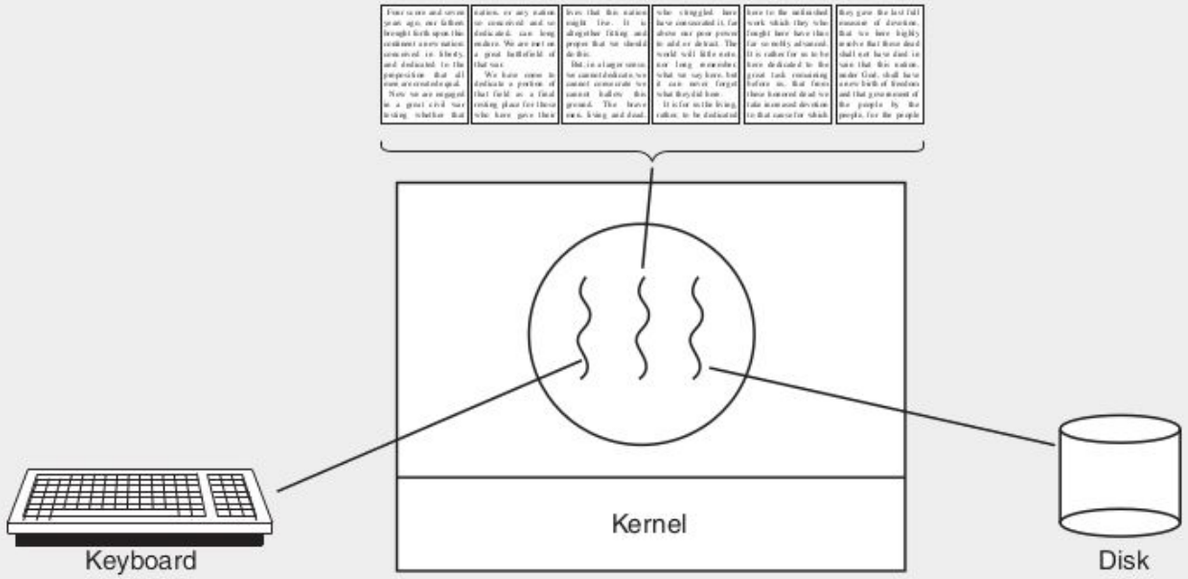


Figure 2-7. A word processor with three threads.

Threads

Algunos posibles beneficios:

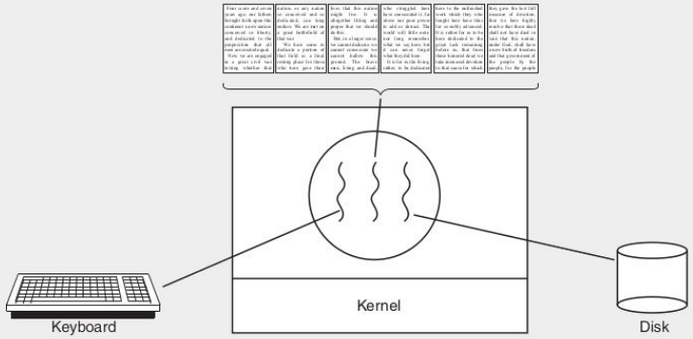


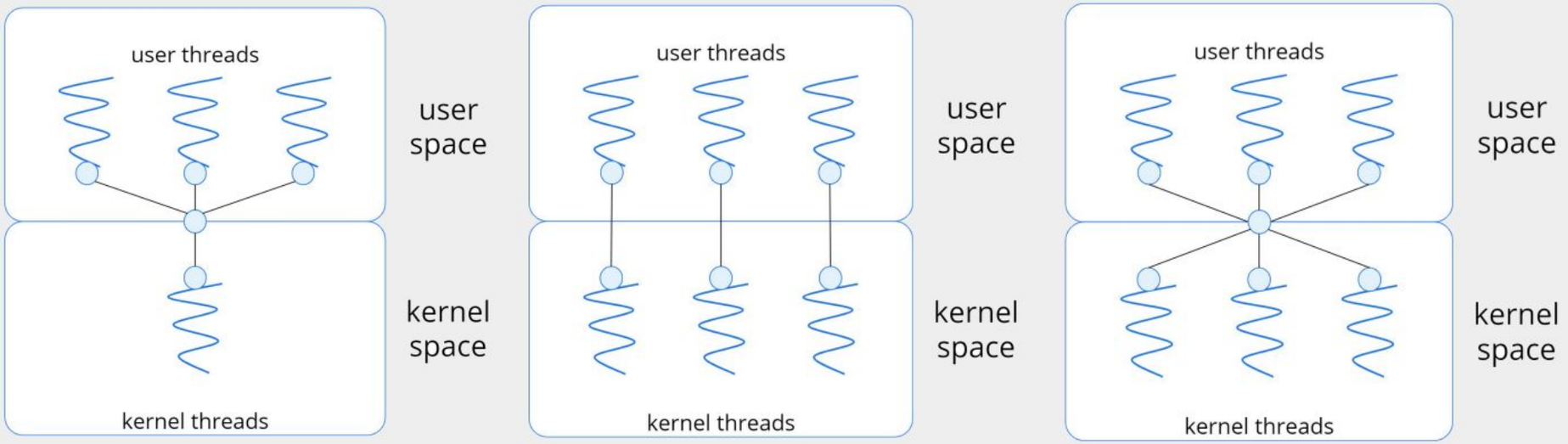
Figure 2-7. A word processor with three threads.

- Capacidad de respuesta: un thread espera I/O, otro utiliza CPU, etc.
- Compartir recursos: compartir codigo y datos, archivos abiertos, etc.
  - Menor overhead en:
    - la creación y finalización de threads
    - el cambio de contexto.
- Escalabilidad: aprovechar multiples CPU y tener paralelismo real.

Threads - Modelos multithreading

(para un único proceso)

- Muchos a Uno
- Uno a Uno
- Muchos a Muchos



Threads - Implementación de threads

- **En espacio de usuario.**

  En el ambiente de tiempo de ejecución (run-time system)

- **En espacio del kernel**

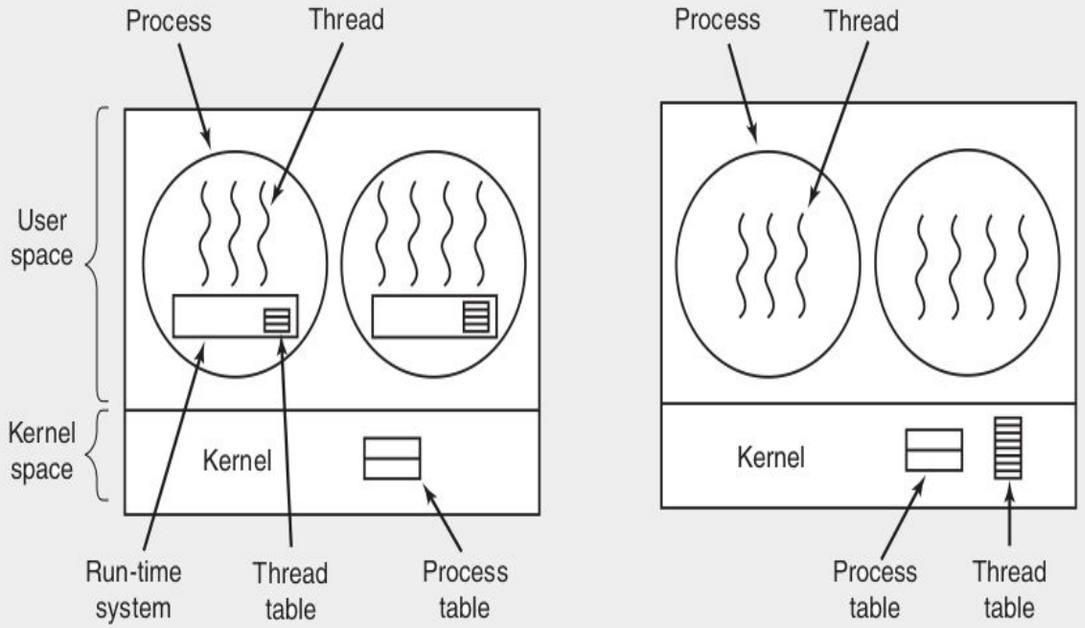


Figure 2-16. (a) A user-level threads package. (b) A threads package managed by the kernel.

Threads - Implementación de threads - Implementación en espacio de usuario

Ventajas:

- No es necesario pasar a modo kernel ni ejecutar código del SO. Mejor rendimiento que kernel threads.
- Aún mejor que planificar diferentes procesos (el planificador de threads tiene mejor rendimiento)
- Un algoritmo de planificación diferente para los threads (por proceso)

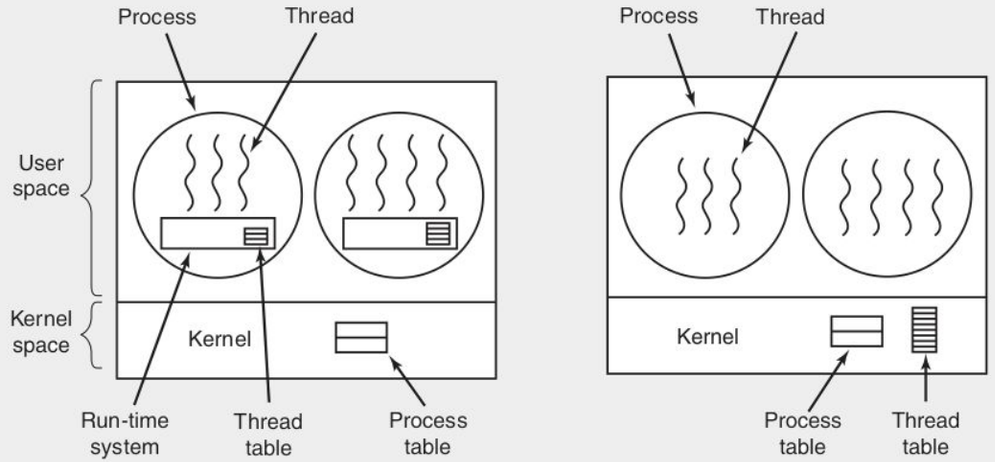


Figure 2-16. (a) A user-level threads package. (b) A threads package managed

Desventajas:

- ¿Cómo gestionar las llamadas al sistema bloqueantes? Posibilidad : cambiar la API de read, o usar select en UNIX.
- Si existe un page fault el proceso es bloqueado por el SO.
- El thread podría tomar la CPU y no liberarla. Posibilidad: implementar una señal de clock periódica.
- Las aplicaciones más destinadas a ser resueltas con threads realizan muchas llamadas al sistema bloqueantes.

Threads - Implementación de threads - Implementación en espacio del kernel

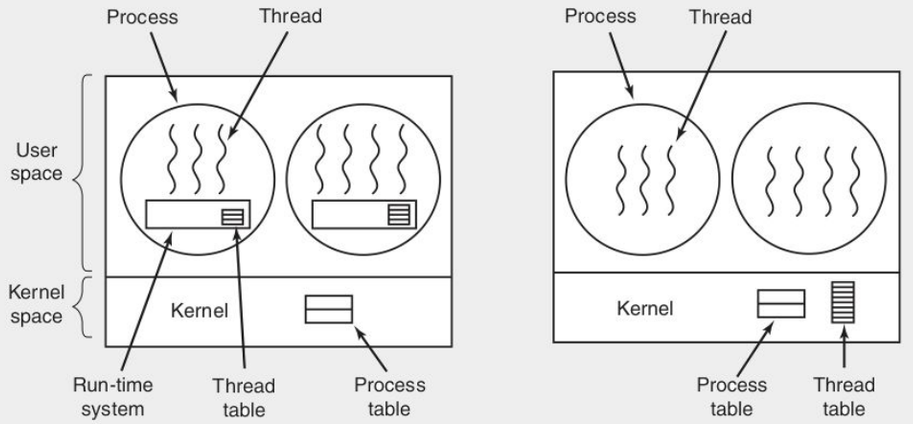


Figure 2-16. (a) A user-level threads package. (b) A threads package managed by the kernel.

Ventajas:

- Soluciona casi todos los problemas en la implementación de threads en espacio de usuario.

Desventajas:

- La creación y finalización de threads es "mas costosa".
  Posibilidad : al finalizar un thread no liberar su gestión y reusarla luego.
- Ante un system call desde un thread el SO podría ejecutar otro thread (bien), pero también podría switchear a otro proceso (ouch).
- Un system call es mas compleja que una llamada a una función.
- ¿Cómo tratar las señales recibidas por el proceso?
- ¿Cómo tratar a un fork()?

Sistemas Operativos I - Procesos y Threads

Threads - Implementación de threads

Implementación en Sistemas Operativos

- Linux, Windows, Apple OS y otros sistemas operativos tradicionales de PC y servidores contienen implementación en espacio del kernel.

- Xinu y otros RTOS: el modelo de procesos es el de threads.
  Desventaja:
  Los procesos no tienen memoria protegida e independiente, todos comparten el segmento de código y de datos.
  Las pilas son independientes, pero podrían llegar a solaparse y corromper el sistema.

```c
/* Ejemplo Linux. Compilar con: gcc -o p p.c -lpthread */

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void *p_msg( void *ptr );

main()
{
     pthread_t thread1, thread2;
     char *msg1 = "Thread 1";
     char *msg2 = "Thread 2";
     int  r1, r2;

    /* Create independent threads each of which will execute function */
     r1 = pthread_create( &thread1, NULL, p_msg, (void*) msg1);
     r2 = pthread_create( &thread2, NULL, p_msg, (void*) msg2);

     /* Wait till threads are complete before main continues.  */
     pthread_join( thread1, NULL);
     pthread_join( thread2, NULL);

     printf("Thread 1 returns: %d\n",r1);
     printf("Thread 2 returns: %d\n",r2);
     exit(0);
}

void *p_msg( void *ptr )
{
     char *m;
     m = (char *) ptr;
     printf("%s \n", m);
}
```

Documentación de system calls y de la biblioteca de C en Linux

man 2 intro
man 2 syscalls
man 2 ipc
man 7 libc

man 5 proc # ver /proc/pid/status
man 5 sysfs

man 7 standards

Documentación de system calls y de la biblioteca de C en Linux

man 2 intro
man 2 syscalls
man 2 ipc
man 7 libc

man 5 proc # ver /proc/pid/status
man 5 sysfs

man 7 standards

COMUNICACIÓN INTER PROCESOS

Motivación

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

COMUNICACIÓN INTER PROCESOS

Motivación

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

Peero, muchas aplicaciones requieren comunicarse
para *trabajar cooperativamente*.

Motivación: Ejemplo de aplicaciones INDEPENDIENTES

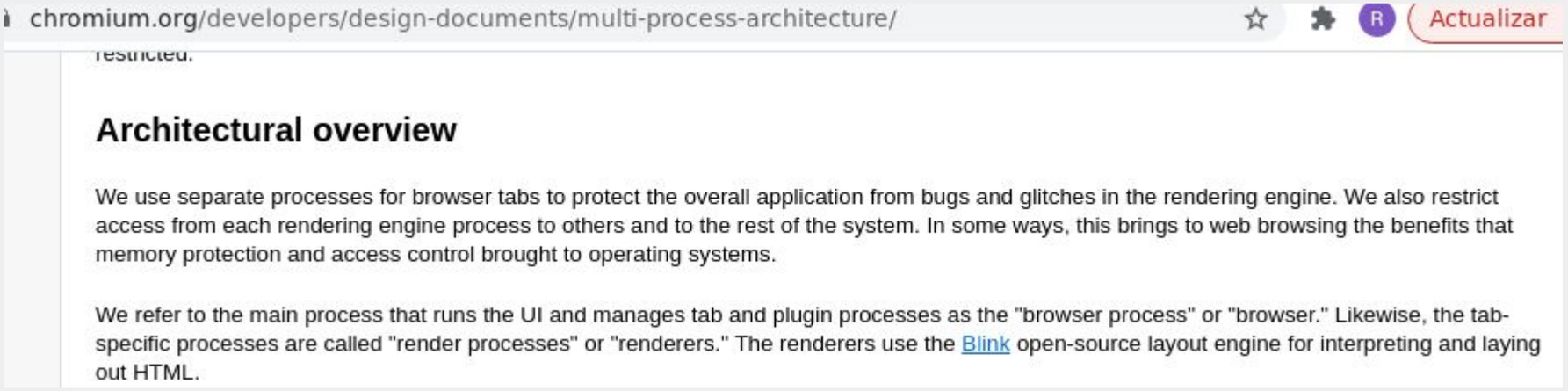
El usuario desea conocer :

- todos los archivos fuentes de XINU
- 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

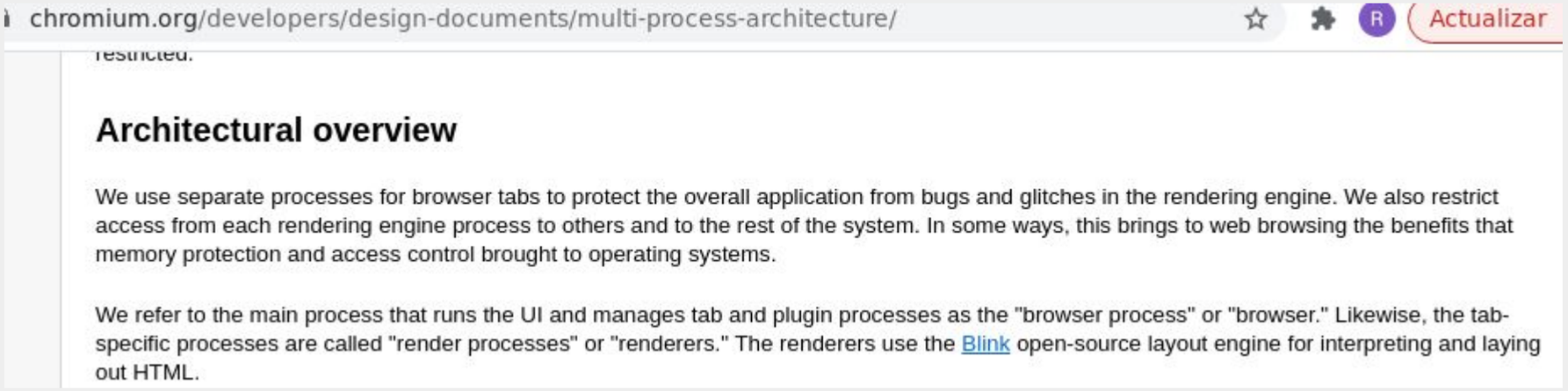
Motivación: Ejemplo de aplicaciones COOPERATIVAS NATIVAMENTE



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

Principio: existen dos mecanismos

- memoria compartida
- pasaje de mensajes

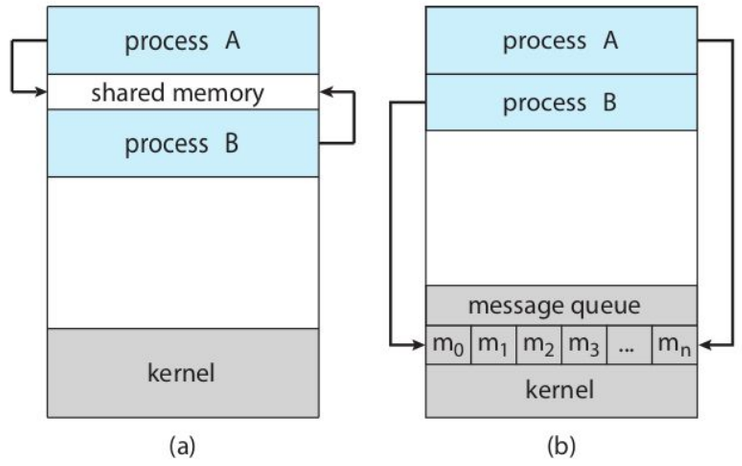


Figure 3.11 Communications models. (a) Shared memory. (b) Message passing.

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().
```

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.

```c
/* 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;
}
```

```c
/* 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;
}
```

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 mensajes?
¿Cómo se especifica el receptor?
¿Conoce el receptor la identidad del emisor?
¿Será una comunicación bidireccional?
¿Es la interfaz síncrona? ¿o asíncrona?

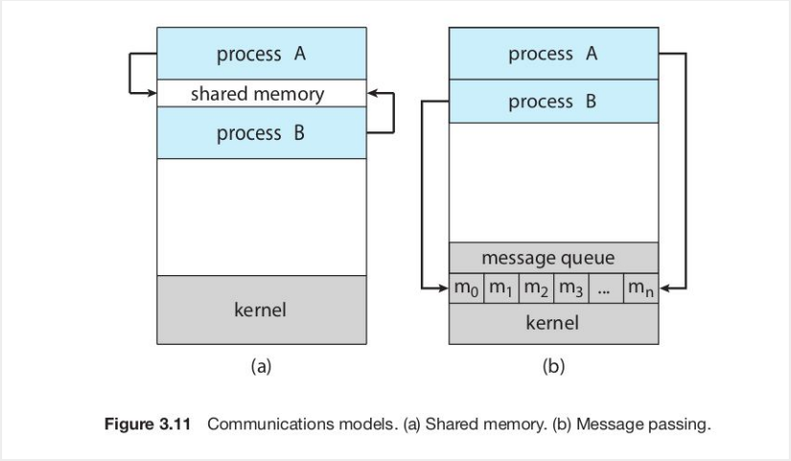


Figure 3.11 Communications models. (a) Shared memory. (b) Message passing.

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

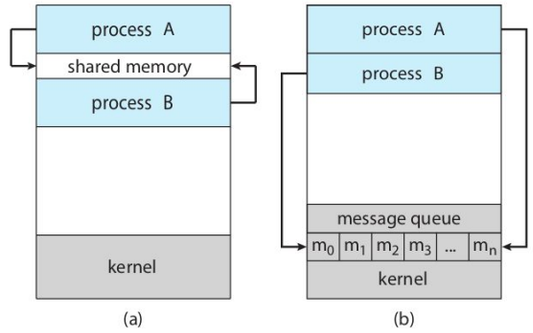


Figure 3.11 Communications models. (a) Shared memory. (b) Message passing.

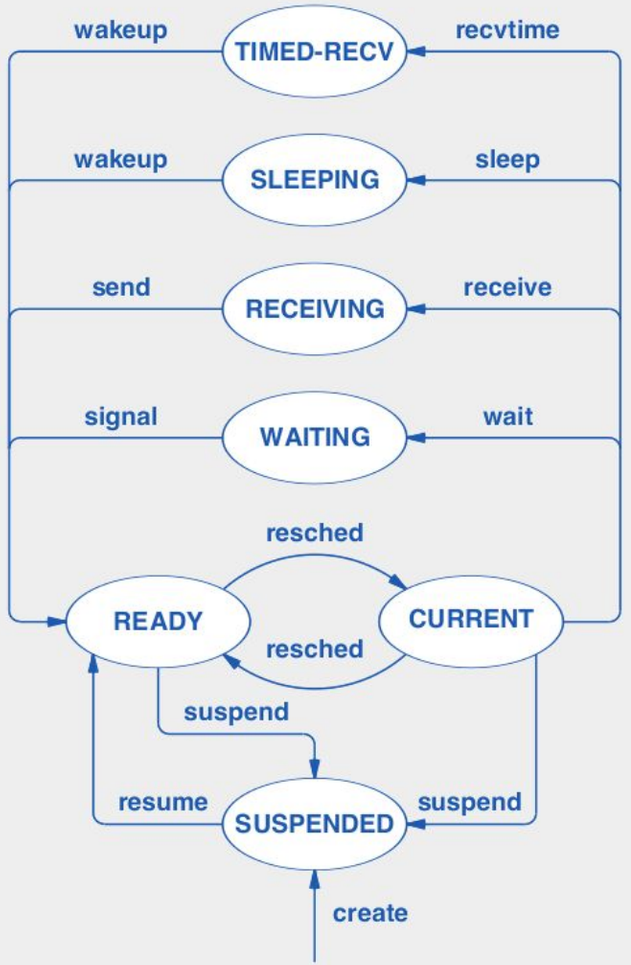
Interfaz asíncrona

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

Ejemplo de pasaje de mensajes : XINU

- 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(pid, msg);
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**
- **pipes y archivos FIFOS**
- **Señales UNIX**
- **Memoria compartida (POSIX)**
  man shm_overview

Ejemplos en Windows

The following IPC mechanisms are supported by Windows:

- Clipboard
- COM
- Data Copy
- DDE
- File Mapping
- Mailslots
- Pipes
- RPC
- Windows Sockets

Sistemas Operativos I - Comunicación entre procesos

Implementaciones a nivel de sistemas operativos tradicionales:

Ejemplo de señales en 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
```