

Departamento de Lenguajes y Ciencias de la
Computación Universidad de Málaga

Sistemas Operativos

Tema 4: Programación Multihebra

Curso 2001/2002



E.T.I. Informática Gestión

Profesor:
Francisco López Valverde

SISTEMAS OPERATIVOS

Tema 4: Programación Multihebra

Introducción

Procesos y hebras

**Principios de
programación
concurrente**

**Programación con
Pthreads**

Problemas clásicos

**Fuentes de
información**

Índice de contenidos

- [Introducción a la programación concurrente](#)
- [Procesos y hebras](#)
- [Principios de programación concurrente](#)
- [Programación con Pthreads](#)
- [Problemas clásicos de programación concurrente](#)
- [Fuentes de información](#)



Introducción

- Programación concurrente
- Arquitecturas hardware
- Estilos de programación
- Programación concurrente en sistemas operativos

Procesos y hebras**Principios de programación concurrente****Programación con Pthreads****Problemas clásicos****Fuentes de información**

- Programación concurrente
- Arquitecturas hardware
- Estilos de programación
- Programación concurrente en sistemas operativos

**Programación concurrente****Introducción**

- Programación concurrente
- Arquitecturas hardware
- Estilos de programación
- Programación concurrente en sistemas operativos

Procesos y hebras**Principios de programación concurrente****Programación con Pthreads****Problemas clásicos****Fuentes de información**

- Un programa concurrente es
 - Aquél que contiene uno o más procesos que trabajan de forma conjunta para realizar una determinada tarea
- Dentro del programa concurrente
 - Cada proceso es un programa secuencial
- Diferencia entre un programa secuencial y uno concurrente
 - Un programa secuencial tiene un único flujo de ejecución
 - Un programa concurrente tiene múltiples flujos de control



Programación concurrente

Introducción

- Programación concurrente
- Arquitecturas hardware
- Estilos de programación
- Programación concurrente en sistemas operativos

Procesos y hebras

Principios de programación concurrente

Programación con Pthreads

Problemas clásicos

Fuentes de información

- Los procesos de un programa concurrente interactúan entre sí de dos formas:
 - Comunicación
 - Sincronización
- La comunicación se lleva a cabo de dos formas
 - Compartiendo memoria
 - Enviando información por medio de mensajes
- Hay dos tipos básicos de sincronización
 - Exclusión mutua
 - Sincronización mediante condiciones



Comunicación mediante memoria compartida

Introducción

- Programación concurrente
- Arquitecturas hardware
- Estilos de programación
- Programación concurrente en sistemas operativos

Procesos y hebras

Principios de programación concurrente

Programación con Pthreads

Problemas clásicos

Fuentes de información

- En este esquema
 - Los procesos de un programa concurrente se comunican compartiendo un conjunto de variables que son visibles a todos ellos
 - Estas variables son accedidas de la misma forma que cualquier otra variable
- Analogía:
 - Los asistentes a una clase y el contenido de la pizarra

```
/* Variable compartida */
bool impresoraOcupada = false ;
```

```
/* Proceso 1 */
if (!impresoraOcupada)
    impresoraOcupada = true ;
// imprimir
impresoraOcupada = false ;
```

```
/* Proceso 2 */
if (!impresoraOcupada)
    impresoraOcupada = true ;
// imprimir
impresoraOcupada = false ;
```



Comunicación mediante paso de mensajes

Introducción

- Programación concurrente
- Arquitecturas hardware
- Estilos de programación
- Programación concurrente en sistemas operativos

Procesos y hebras

Principios de programación concurrente

Programación con Pthreads

Problemas clásicos

Fuentes de información

- En este esquema
 - Los procesos de un programa concurrente se comunican enviándose información mediante mensajes
 - Es el mecanismo que se emplea cuando los procesos no comparten memoria
- Analogía:
 - Comunicación entre personas mediante correo electrónico
- Los programas concurrentes basados en paso de mensajes
 - Son más complejos, habitualmente, que los basados en memoria compartida



Sincronización mediante exclusión mutua

Introducción

- Programación concurrente
- Arquitecturas hardware
- Estilos de programación
- Programación concurrente en sistemas operativos

Procesos y hebras

Principios de programación concurrente

Programación con Pthreads

Problemas clásicos

Fuentes de información

- En un programa concurrente existen regiones o secciones críticas
 - Son trozos de código que sólo los puede ejecutar un único proceso en un instante dado
- El problema de la exclusión mutua
 - Es el problema de garantizar que las secciones críticas se ejecuten de forma mutuamente exclusiva

```
/* Variable compartida */
bool impresoraOcupada = false ;
```

```
/* Proceso 1 */

if (!impresoraOcupada)
    impresoraOcupada = true ;
// imprimir
impresoraOcupada = false ;
```

```
/* Proceso 2 */

if (!impresoraOcupada)
    impresoraOcupada = true ;
// imprimir
impresoraOcupada = false ;
```



Introducción

- Programación concurrente
- Arquitecturas hardware
- Estilos de programación
- Programación concurrente en sistemas operativos

Procesos y hebras**Principios de programación concurrente****Programación con Pthreads****Problemas clásicos****Fuentes de información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Sincronización mediante condiciones

- La sincronización mediante condiciones
 - Es el problema de retrasar la ejecución de un proceso hasta que se cumpla una determinada condición
- Habitualmente,
 - La sincronización mediante condiciones se combina con la sincronización mediante exclusión mutua

```
/* Variable compartida */
bool impresoraOcupada = false ;
```

```
/* Proceso 1 */
```

```
wait if (impresoraOcupada)
    impresoraOcupada = true ;
// imprimir
impresoraOcupada = false ;
```

```
/* Proceso 2 */
```

```
wait if (impresoraOcupada)
    impresoraOcupada = true ;
// imprimir
impresoraOcupada = false ;
```

Introducción

- Programación concurrente
- Arquitecturas hardware
- Estilos de programación
- Programación concurrente en sistemas operativos

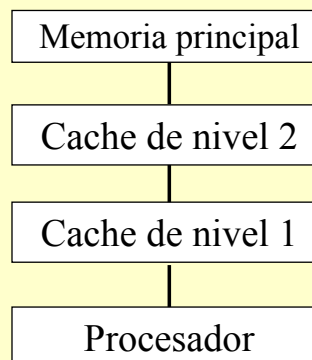
Procesos y hebras**Principios de programación concurrente****Programación con Pthreads****Problemas clásicos****Fuentes de información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Arquitecturas hardware

- Tres tipos básicos de arquitecturas
 - Sistemas monoprocesador
 - Sistemas multiprocesadores
 - Sistemas distribuidos



Arquitectura de un sistema monoprocesador

Introducción

- Programación concurrente
- Arquitecturas hardware
- Estilos de programación
- Programación concurrente en sistemas operativos

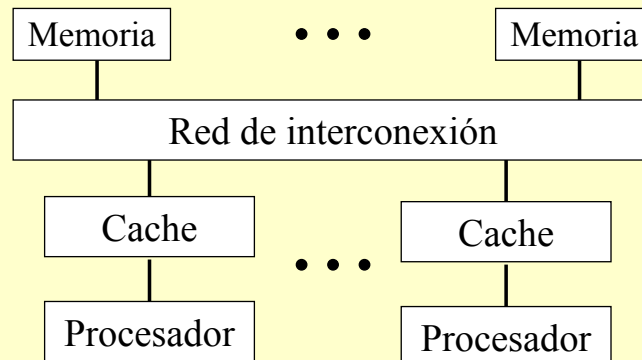
Procesos y hebras**Principios de programación concurrente****Programación con Pthreads****Problemas clásicos****Fuentes de información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Arquitecturas hardware

- Los sistemas multiprocesadores
 - Están compuestos por un conjunto de procesadores que acceden a los módulos de memoria por medio de una red de interconexión
 - Esta red suele ser un bus o un conmutador de barras cruzadas (*crossbar*)



Arquitectura de un sistema multiprocesador

Introducción

- Programación concurrente
- Arquitecturas hardware
- Estilos de programación
- Programación concurrente en sistemas operativos

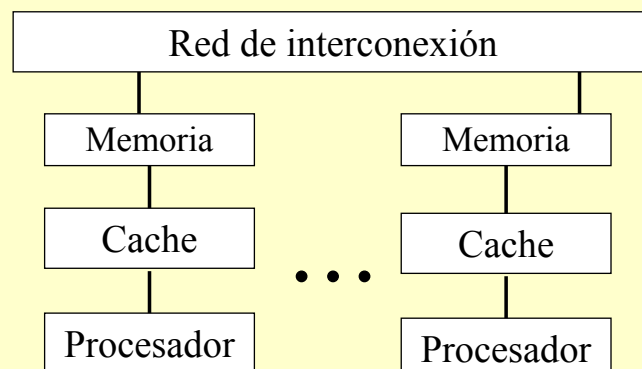
Procesos y hebras**Principios de programación concurrente****Programación con Pthreads****Problemas clásicos****Fuentes de información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Arquitecturas hardware

- Los sistemas distribuidos
 - Difieren de los multiprocesadores en que cada procesador tiene su propia memoria local
 - Se suelen denominar *redes* cuando la red de interconexión es una red de área local (LAN) o de área extensa (WAN)



Arquitectura de un sistema distribuido

Introducción

- Programación concurrente
- Arquitecturas hardware
- Estilos de programación
- Programación concurrente en sistemas operativos

Procesos y hebras**Principios de programación concurrente****Programación con Pthreads****Problemas clásicos****Fuentes de información**

Estilos de programación

- Se pueden distinguir tres estilos de programación, que permiten desarrollar tres tipos de aplicaciones diferentes
 - Programación concurrente
 - Programación distribuida
 - Programación paralela
- Estas disciplinas no son excluyentes

Introducción

- Programación concurrente
- Arquitecturas hardware
- Estilos de programación
- Programación concurrente en sistemas operativos

Procesos y hebras**Principios de programación concurrente****Programación con Pthreads****Problemas clásicos****Fuentes de información**

Estilos de programación

- Programación concurrente
 - Es la disciplina que estudia la programación de procesos concurrentes
 - Aunque se suele aplicar al contexto en el que los procesos se comunican mediante memoria compartida
- Ejemplos de aplicaciones:
 - Sistemas de ventanas
 - Sistemas operativos

Introducción

- Programación concurrente
- Arquitecturas hardware
- Estilos de programación
- Programación concurrente en sistemas operativos

Procesos y hebras**Principios de programación concurrente****Programación con Pthreads****Problemas clásicos****Fuentes de información**

Estilos de programación

- Programación distribuida
 - Es la rama de la programación concurrente que se centra en el estudio de la programación sobre sistemas distribuidos
 - Los procesos se comunican mediante paso de mensajes
- Ejemplos de aplicaciones:
 - Sistemas de ficheros en red
 - Servidores Web
 - Bases de datos para reserva de billetes
- Es habitual que los componentes de un sistema distribuido sean a su vez programas concurrentes

Introducción

- Programación concurrente
- Arquitecturas hardware
- Estilos de programación
- Programación concurrente en sistemas operativos

Procesos y hebras**Principios de programación concurrente****Programación con Pthreads****Problemas clásicos****Fuentes de información**

Estilos de programación

- Programación paralela
 - A veces se considera un sinónimo de programación concurrente
 - Se utiliza el término *computación paralela* cuando se utiliza un computador paralelo con el fin de
 - resolver un problema más rápidamente,
 - resolver un problema mayor en un mismo tiempo, o
 - resolver un problema de mayor tamaño
- Ejemplos de aplicaciones
 - Computaciones científicas que modelan y simulan el clima, el movimiento de las mareas, etc.
 - Procesamiento de imágenes
 - Problemas de optimización combinatoria

Introducción

- Programación concurrente
- Arquitecturas hardware
- Estilos de programación
- Programación concurrente en sistemas operativos

Procesos y hebras**Principios de programación concurrente****Programación con Pthreads****Problemas clásicos****Fuentes de información**

- La mayoría de las aplicaciones concurrentes utilizan cinco tipos de soluciones, o *paradigmas*
 - Paralelismo iterativo
 - Paralelismo recursivo
 - Productores y consumidores
 - Clientes y servidores
 - Paralelismo entre iguales (*peer parallelism*)

**Introducción**

- Programación concurrente
- Arquitecturas hardware
- Estilos de programación
- Programación concurrente en sistemas operativos

Procesos y hebras**Principios de programación concurrente****Programación con Pthreads****Problemas clásicos****Fuentes de información**

- Paralelismo iterativo
 - Aplicación habitual: computación científica paralela (ejemplo: producto de matrices)
- Paralelismo recursivo
 - Aplicación habitual: ordenación, optimización combinatoria, juegos (ejemplo: ajedrez)
- Productores y consumidores
 - Aplicación habitual: procesos que generan datos que han de usar otros procesos (ejemplo: tuberías en UNIX)
- Clientes y servidores
 - Aplicación habitual: sistemas distribuidos (ejemplo: navegadores y servidores Web)



Introducción

- Programación concurrente
- Arquitecturas hardware
- Estilos de programación
- Programación concurrente en sistemas operativos

Procesos y hebras**Principios de programación concurrente****Programación con Pthreads****Problemas clásicos****Fuentes de información**

- Paralelismo entre iguales
 - Aplicación habitual: programas distribuidos paralelos (ejemplo: producto de matrices distribuidos)

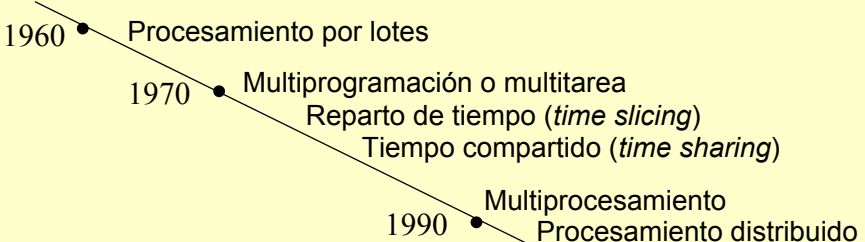


Programación concurrente en sistemas operativos

Introducción

- Programación concurrente
- Arquitecturas hardware
- Estilos de programación
- Programación concurrente en sistemas operativos

Procesos y hebras**Principios de programación concurrente****Programación con Pthreads****Problemas clásicos****Fuentes de información**

- La programación concurrente tiene su origen en los sistemas operativos
- Surge como solución a un problema:
 - Maximizar el uso de la capacidad de procesamiento de los sistemas informáticos
- Evolución
 



Procesos y hebras

Introducción

Procesos y hebras

- Por qué procesos multihebra
- Ventajas que aportan las hebras
- Sistemas operativos multihebra
- Tipos de hebras
- Implementación de hebras
- Bibliotecas de hebras

Principios de programación concurrente

Programación con Pthreads

Problemas clásicos

Fuentes de información



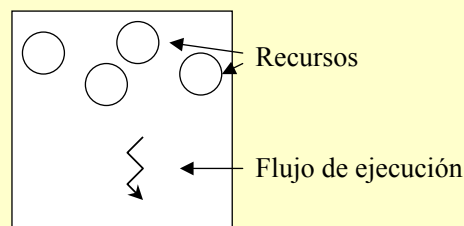
Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

- Hasta mediados de los años 80, para los sistemas operativos los procesos son entidades que:
 - Presentan una actividad dentro del sistema operativo (son las entidades activas)
 - Su flujo de ejecución es secuencial
 - Su ejecución la planifica el sistema operativo
 - Se ejecutan en un espacio de direcciones
 - Solicitan y usan los recursos del sistema operativo

Proceso tradicional:

Un flujo de ejecución secuencial



Por qué procesos multihebra

Introducción

Procesos y hebras

- Por qué procesos multihebra
- Ventajas que aportan las hebras
- Sistemas operativos multihebra
- Tipos de hebras
- Implementación de hebras
- Bibliotecas de hebras

Principios de programación concurrente

Programación con Pthreads

Problemas clásicos

Fuentes de información



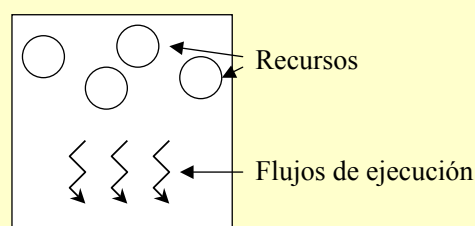
Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

- Inconvenientes de este modelo de proceso
 - Son entidades costosas de crear y manipular
 - Compartir recursos entre procesos es complejo y costoso
- Solución adoptada:
 - Generalizar el concepto de proceso, de forma que a cada proceso se le pueda asociar más de una actividad de procesamiento

Proceso actual:

Varios flujos de ejecución concurrentes



Introducción**Procesos y hebras**

- Por qué procesos multihebra
- Ventajas que aportan las hebras
- Sistemas operativos multihebra
- Tipos de hebras
- Implementación de hebras
- Bibliotecas de hebras

Principios de programación concurrente**Programación con Pthreads****Problemas clásicos****Fuentes de información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Por qué procesos multihebra

- Separación de conceptos
 - Un *proceso* es un entorno de ejecución sobre el que se ejecutan una o varias hebras (*threads*)
 - Una *hebra* es una entidad que presenta una actividad
- A un proceso se asocia
 - Espacio de direcciones
 - Variables globales
 - Ficheros abiertos
- A una hebra se asocia
 - Registro contador de programa
 - Pila
 - Estado (listo, ejecución, bloqueado)

Introducción**Procesos y hebras**

- Por qué procesos multihebra
- Ventajas que aportan las hebras
- Sistemas operativos multihebra
- Tipos de hebras
- Implementación de hebras
- Bibliotecas de hebras

Principios de programación concurrente**Programación con Pthreads****Problemas clásicos****Fuentes de información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Por qué procesos multihebra

- A este tipo de procesos se le denomina *multihebra* (*multithreaded process*)
- En un proceso multihebra
 - Todas las hebras comparten todos los recursos del proceso
 - Las hebras de un proceso se comunican mediante memoria compartida
- Al contrario de lo que ocurre entre procesos distintos
 - El sistema operativo no tiene que preocuparse de que las hebras de un proceso interfieran entre sí

Ventajas que aportan las hebras

Introducción

Procesos y hebras

- Por qué procesos multihebra
- Ventajas que aportan las hebras
- Sistemas operativos multihebra
- Tipos de hebras
- Implementación de hebras
- Bibliotecas de hebras

Principios de programación concurrente

Programación con Pthreads

Problemas clásicos

Fuentes de información



Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

- Los procesos multihebra aportan las siguientes ventajas:
 - Solapamiento de operaciones de entrada/salida y computación dentro de un mismo proceso
 - Multiprocesamiento
 - Una hebra consume menos recursos y requiere un coste de manipulación menor que un proceso
 - Existen aplicaciones concurrentes cuya estructura se amolda de forma natural a un esquema multihebra

Sistemas operativos multihebra

Introducción

Procesos y hebras

- Por qué procesos multihebra
- Ventajas que aportan las hebras
- Sistemas operativos multihebra
- Tipos de hebras
- Implementación de hebras
- Bibliotecas de hebras

Principios de programación concurrente

Programación con Pthreads

Problemas clásicos

Fuentes de información



Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

- La mayoría de los sistemas operativos actuales son multihebra
 - Solaris
 - Linux
 - Windows NT/2000
 - MacOS X

Tipos de hebras

Introducción

Procesos y hebras

- Por qué procesos multihebra
- Ventajas que aportan las hebras
- Sistemas operativos multihebra

- Tipos de hebras
- Implementación de hebras
- Bibliotecas de hebras

Principios de programación concurrente

Programación con Pthreads

Problemas clásicos

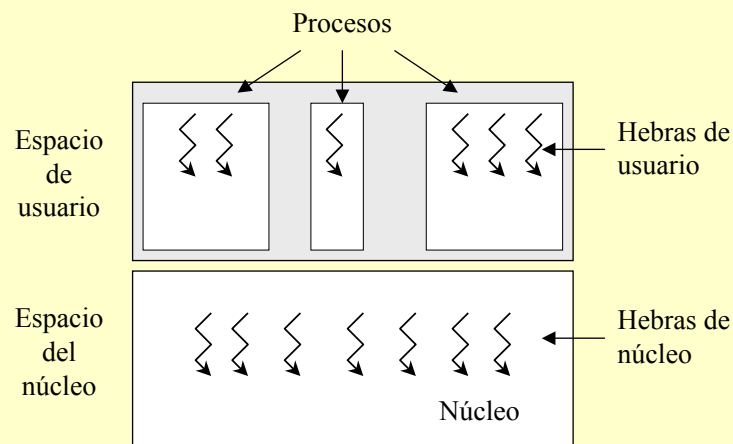
Fuentes de información



Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

- Hay dos tipos de hebras
 - Las hebras que se ejecutan en el espacio de usuario
 - Las hebras que se ejecutan en el espacio del núcleo



Implementación de hebras

Introducción

Procesos y hebras

- Por qué procesos multihebra
- Ventajas que aportan las hebras
- Sistemas operativos multihebra
- Tipos de hebras
- Implementación de hebras
- Bibliotecas de hebras

Principios de programación concurrente

Programación con Pthreads

Problemas clásicos

Fuentes de información



Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

- Existen varias formas de hacer corresponder las hebras de usuario con las hebras de núcleo
 - Muchas-a-Una
 - Una-a-Una
 - Muchas-a-Muchas
- Cada una estas formas da lugar, respectivamente, a un esquema de implementación diferente
 - Implementación a nivel de usuario
 - Implementación a nivel de núcleo
 - Implementación híbrida

Implementación a nivel de usuario

Introducción

Procesos y hebras

- Por qué procesos multihebra
- Ventajas que aportan las hebras
- Sistemas operativos multihebra
- Tipos de hebras
- Implementación de hebras
- Bibliotecas de hebras

Principios de programación concurrente

Programación con Pthreads

Problemas clásicos

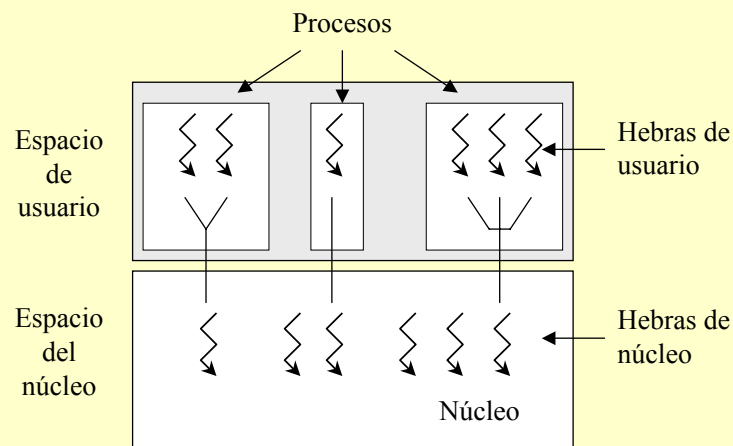
Fuentes de información



Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

- Características
 - Correspondencia Muchas-a-Una
 - Todas las hebras de usuario de un proceso se vinculan a una única hebra del núcleo



Implementación a nivel de usuario

Introducción

Procesos y hebras

- Por qué procesos multihebra
- Ventajas que aportan las hebras
- Sistemas operativos multihebra
- Tipos de hebras
- Implementación de hebras
- Bibliotecas de hebras

Principios de programación concurrente

Programación con Pthreads

Problemas clásicos

Fuentes de información



Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

- Ventajas
 - Gestión de hebras eficiente
 - Esquema empleado en sistemas operativos no multihebra
- Inconvenientes
 - Si una hebra se bloquea, se bloquea todo el proceso
 - No permite multiprocesamiento
- Ejemplos
 - La biblioteca de procesos de peso ligero (LWP o *light weighted processes*) del sistema operativo SunOS
 - Los sistemas operativos actuales no admiten este esquema (ver [implementaciones híbridas](#))

Introducción**Procesos y hebras**

- Por qué procesos multihebra
- Ventajas que aportan las hebras
- Sistemas operativos multihebra
- Tipos de hebras
- Implementación de hebras
- Bibliotecas de hebras

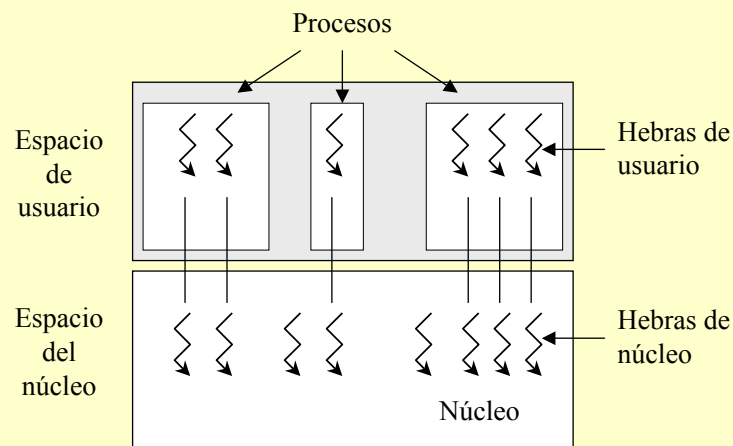
Principios de programación concurrente**Programación con Pthreads****Problemas clásicos****Fuentes de información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Implementación a nivel de núcleo

- Características
 - Correspondencia Una-a-Una
 - Cada hebra de usuario se vincula a una única hebra del núcleo

**Introducción****Procesos y hebras**

- Por qué procesos multihebra
- Ventajas que aportan las hebras
- Sistemas operativos multihebra
- Tipos de hebras
- Implementación de hebras
- Bibliotecas de hebras

Principios de programación concurrente**Programación con Pthreads****Problemas clásicos****Fuentes de información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Implementación a nivel de núcleo

- Ventajas
 - Permite obtener todos los beneficios de aportan las hebras
- Inconvenientes
 - Gestión de hebras más costosa que en el caso de la implementación a nivel de usuario
 - Número máximo de hebras teóricamente inferior al de una implementación a nivel de usuario
- Ejemplos de sistemas
 - Windows NT/2000
 - Linux

Introducción**Procesos y hebras**

- Por qué procesos multihebra
- Ventajas que aportan las hebras
- Sistemas operativos multihebra
- Tipos de hebras
- Implementación de hebras
- Bibliotecas de hebras

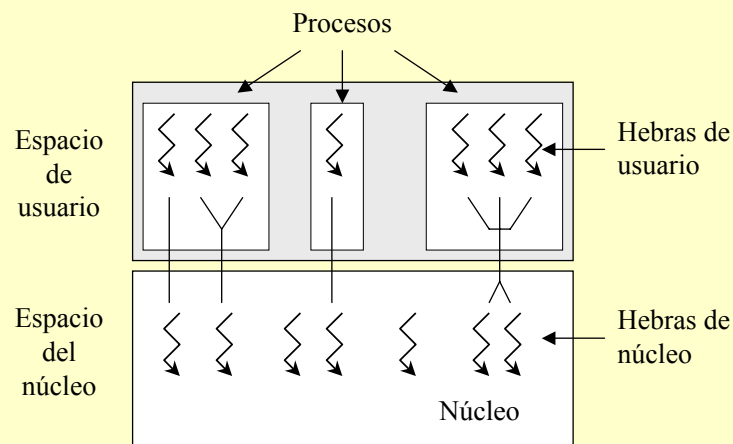
Principios de programación concurrente**Programación con Pthreads****Problemas clásicos****Fuentes de información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Implementación híbrida

- Características
 - Correspondencia Muchas-a-Muchas
 - Existe varias formas de vincular las hebras de usuario con las hebras del núcleo

**Introducción****Procesos y hebras**

- Por qué procesos multihebra
- Ventajas que aportan las hebras
- Sistemas operativos multihebra
- Tipos de hebras
- Implementación de hebras
- Bibliotecas de hebras

Principios de programación concurrente**Programación con Pthreads****Problemas clásicos****Fuentes de información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Implementación híbrida

- Ventajas
 - Combina las ventajas de la implementación a nivel de usuario (bajo coste de manipulación) con las de la implementación a nivel de usuario (no limita las posibilidades que permiten las hebras)
- Inconvenientes
 - Modelo de programación complejo cuando la vinculación entre hebras de usuario y hebras de núcleo es responsabilidad del programador
 - Ejemplo: Solaris
- En otros sistemas
 - La vinculación es automática
 - Ejemplo: Tru64 (antiguo Digital UNIX)

Bibliotecas de hebras

Introducción

Procesos y hebras

- Por qué procesos multihebra
- Ventajas que aportan las hebras
- Sistemas operativos multihebra
- Tipos de hebras
- Implementación de hebras
- Bibliotecas de hebras

Principios de programación concurrente

Programación con Pthreads

Problemas clásicos

Fuentes de información



Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

- Los lenguajes más usados para programar sobre sistemas operativos (C/C++) no incorporan hebras
- Pero puede hacer uso de las hebras del sistema operativo a través de una biblioteca de funciones
- En la actualidad existen dos bibliotecas de hebras de uso mayoritario
 - Hebras Win32, usadas en los sistemas operativos de 32 bits de Microsoft
 - Hebras POSIX (Pthreads), usadas en sistemas UNIX

Bibliotecas de hebras

Introducción

Procesos y hebras

- Por qué procesos multihebra
- Ventajas que aportan las hebras
- Sistemas operativos multihebra
- Tipos de hebras
- Implementación de hebras
- Bibliotecas de hebras

Principios de programación concurrente

Programación con Pthreads

Problemas clásicos

Fuentes de información



Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

- Comparación entre hebras Win32 y Pthreads
 - Windows NT/2000 fue diseñado desde el principio para que fuese multihebra
 - En UNIX se ha producido una adaptación de procesos tradicionales a procesos multihebra
- Consecuencia
 - En Windows NT/2000 las hebras están integradas de forma natural dentro del sistema
 - En UNIX existen conflictos cuando se usan características que están pensadas para procesos tradicionales (ejemplos: bloqueo de ficheros, señales)

Introducción

Procesos y hebras

Principios de
programación
concurrente

- Conceptos
- Exclusión mutua
- Sincronización mediante condiciones

Programación con
Pthreads

Problemas clásicos

Fuentes de
información

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Principios de programación concurrente

- Definición: estado de un programa
 - El **estado** de un programa concurrente viene dado por el valor de las variables del programa en un instante concreto de tiempo
 - Todo programa comienza en un estado inicial
 - Cada proceso del programa se ejecuta de forma independiente
- Definición: acción atómica
 - Los procesos ejecutan secuencias de instrucciones
 - Cada instrucción se compone de secuencias de una o varias **acciones atómicas**
 - Son acciones que, de forma indivisible, examinan o alteran el estado del programa

Introducción

Procesos y hebras

Principios de
programación
concurrente

- Conceptos
- Exclusión mutua
- Sincronización mediante condiciones

Programación con
Pthreads

Problemas clásicos

Fuentes de
información

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Principios de programación concurrente

- Definición: intercalación
 - La ejecución de un programa concurrente da lugar a la **ejecución intercalada** (*interleaving*) de las secuencias de acciones atómicas ejecutadas por cada proceso
- Definición: historia
 - Una ejecución concreta de un programa concurrente da lugar a una **historia**, que es una secuencia de acciones atómicas

$$S_0 \rightarrow S_1 \rightarrow \dots \rightarrow S_n$$
 - Una ejecución paralela puede modelarse como una historial secuencial, porque el efecto de ejecutar acciones atómicas en paralelo es equivalente a ejecutarlas en algún orden secuencial

Introducción**Procesos y hebras****Principios de
programación
concurrente**

- Conceptos
- Exclusión mutua
- Sincronización mediante condiciones

**Programación con
Pthreads****Problemas clásicos****Fuentes de
información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Principios de programación concurrente

- Definición: propiedad
 - Cada ejecución de un programa concurrente produce una historia
 - El número de historias posibles para un programa es muy elevado
 - El motivo es que existen muchas formas de intercalar las acciones atómicas de un programa, aunque se parta siempre del mismo estado inicial
 - Una **propiedad** es un atributo del programa que se cumple en cualquier posible historia del mismo
 - Existen dos tipos de propiedades: de seguridad y de viveza

Introducción**Procesos y hebras****Principios de
programación
concurrente**

- Conceptos
- Exclusión mutua
- Sincronización mediante condiciones

**Programación con
Pthreads****Problemas clásicos****Fuentes de
información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Principios de programación concurrente

- Definición: propiedad de seguridad
 - Una **propiedad de seguridad** (*safety property*) es aquella que garantiza que el programa nunca entra en un estado no válido
 - Un estado no es válido cuando algunas variables de estado tienen un valor incorrecto
 - Ejemplos de propiedades: exclusión mutua, ausencia de interbloqueo (*deadlock*)
- Definición: propiedad de viveza
 - Una **propiedad de viveza** (*liveness property*) es aquella que garantiza que el programa entrará eventualmente en un estado válido
 - Ejemplo de propiedades: terminación de un programa, entrada en una sección crítica

Introducción**Procesos y hebras****Principios de
programación
concurrente**

- Conceptos
- Exclusión mutua
- Sincronización mediante condiciones

**Programación con
Pthreads****Problemas clásicos****Fuentes de
información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

El problema de la exclusión mutua

- Características del problema
 - Existe un conjunto de procesos que repetidamente intentan ejecutar una sección crítica y después una sección no crítica
 - La sección crítica ha de ir precedida de un protocolo de entrada y seguida por un protocolo de salida

```
/* La estructura de este código es la misma para */
/* todos los procesos del programa concurrente */

while (true) {
    /* protocolo de entrada */
    /* sección crítica */
    /* protocolo de salida */
    /* sección no crítica */
} /* while */
```

Introducción**Procesos y hebras****Principios de
programación
concurrente**

- Conceptos
- Exclusión mutua
- Sincronización mediante condiciones

**Programación con
Pthreads****Problemas clásicos****Fuentes de
información**

El problema de la exclusión mutua

- Una solución satisfactoria al problema debe el protocolo de entrada y de salida cumplan las siguientes propiedades
 - Exclusión mutua:
 - Sólo un proceso como máximo puede estar a la vez en su sección crítica
 - Ausencia de interbloqueo (*livelock*):
 - Si dos o más procesos quieren entrar a la vez en sus secciones críticas, alguno lo conseguirá
 - Ausencia de retrasos innecesarios:
 - En el caso de que únicamente un proceso quiera entrar en la sección crítica
 - Entrada eventual:
 - Si un proceso intenta entrar en su sección crítica, eventualmente lo conseguirá



Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Introducción**Procesos y hebras****Principios de programación concurrente**

- Conceptos
- Exclusión mutua
- Sincronización mediante condiciones

Programación con Pthreads**Problemas clásicos****Fuentes de información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Exclusión mutua con Pthreads

- Para delimitar el acceso a secciones críticas, Pthreads proporciona un objeto de sincronización llamado *mutex*

```
/* Ejemplo de sección crítica (SC) usando mutexes de */
/* la biblioteca Pthreads */

#include <pthread.h>
bool impresoraOcupada = false ;
...
pthread_mutex_t mutex ;
pthread_mutex_init(&mutex, NULL) ;
...

pthread_mutex_lock(&mutex) ;    /* entrada en la SC */
/* sección crítica */
if (!impresoraOcupada)
    impresoraOcupada = true ;
pthread_mutex_unlock(&mutex) ; /* salida de la SC */

/* sección no crítica */
impresoraOcupada = false ;
```

Introducción**Procesos y hebras****Principios de programación concurrente**

- Conceptos
- Exclusión mutua
- Sincronización mediante condiciones

Programación con Pthreads**Problemas clásicos****Fuentes de información**

Sincronización mediante condiciones

- La sincronización de procesos mediante exclusión mutua permite el acceso a secciones críticas de código
 - Sin embargo, este mecanismo es insuficiente para resolver otro problema de sincronización, como es el bloqueo de un proceso a la espera de que se cumpla una determinada condición
- Ejemplo tipo:
 - El problema de los productores/consumidores, que se comunican a través de un buffer circular
 - Un productor no puede insertar elementos si el buffer está lleno
 - El consumidor no puede extraer elementos si el buffer está vacío



Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Introducción**Procesos y hebras****Principios de
programación
concurrente**

- Conceptos
- Exclusión mutua
- Sincronización mediante condiciones

**Programación con
Pthreads****Problemas clásicos****Fuentes de
información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Sincronización mediante condiciones

- Contexto del problema
 - Un proceso dentro de una sección crítica comprueba si una condición se cumple
 - Si se cumple, el proceso continua su ejecución dentro de la sección crítica
 - Si no se cumple, el proceso debe esperar hasta que la condición se cumpla, pero para ello debe abandonar temporalmente la sección crítica

```
/* Variable compartida */
bool impresoraOcupada = false ;
```

```
/* Proceso 1 */
```

```
wait if (impresoraOcupada)
    impresoraOcupada = true ;
// imprimir
impresoraOcupada = false ;
```

```
/* Proceso 2 */
```

```
wait if (impresoraOcupada)
    impresoraOcupada = true ;
// imprimir
impresoraOcupada = false ;
```

Introducción**Procesos y hebras****Principios de
programación
concurrente**

- Conceptos
- Exclusión mutua
- Sincronización mediante condiciones

**Programación con
Pthreads****Problemas clásicos****Fuentes de
información**

Variables de condición en Pthreads

- El problema de la sincronización mediante condiciones se resuelve en Pthreads mediante el uso de *variables de condición*
- Una variable de condición es un objeto de sincronización sobre el que se definen dos operaciones básicas
 - `pthread_cond_wait(&cond, &mutex)`
 - `pthread_cond_signal(&cond)`
- La operación de espera provoca la suspensión de la hebra y la liberación del mutex
- La operación de señalización reanuda la ejecución de la hebra previamente suspendida



Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Introducción**Procesos y hebras****Principios de
programación
concurrente**

- Conceptos
- Exclusión mutua
- Sincronización mediante condiciones

**Programación con
Pthreads****Problemas clásicos****Fuentes de
información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Variables de condición en Pthreads

```

/* Ejemplo de utilización de variables de condición */
/* con Pthreads */

#include <pthread.h>
bool impresoraOcupada = false ;
...
pthread_mutex_t mutex ;
pthread_mutex_init(&mutex, NULL) ;
...
pthread_cond_t sePuedeImprimir ;
pthread_cond_init(&sePuedeImprimir) ;
...
pthread_mutex_lock(&mutex) ;    /* entrada en la SC */
/* sección crítica */
while (!impresoraOcupada)
    pthread_cond_wait(&sePuedeImprimir, &mutex) ;
impresoraOcupada = true ;
pthread_mutex_unlock(&mutex) ; /* salida de la SC */
...
/* sección no crítica */
...
pthread_mutex_lock(&mutex) ;    /* entrada en la SC */
impresoraOcupada = false ;
pthread_cond_signal(&sePuedeImprimir) ;
pthread_mutex_unlock(&mutex) ; /* salida de la SC */

```

Introducción**Procesos y hebras****Principios de
programación
concurrente****Programación con
Pthreads**

- Funciones
- Ciclo de vida de una hebra
- Atributos

Problemas clásicos**Fuentes de
información**

Programación con Pthreads

- Funciones
- Ciclo de vida de una hebra
- Atributos



Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Introducción**Procesos y hebras****Principios de
programación
concurrente****Programación con
Pthreads**

- Funciones
- Ciclo de vida de una hebra
- Atributos

Problemas clásicos**Fuentes de
información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Grupos de funciones

- Las funciones de la biblioteca Pthreads se dividen en dos grupos
 - Funciones para gestión de hebras
 - Funciones para sincronización de hebras
- Funciones básicas de gestión de hebras
 - Creación `pthread_create()`
 - Terminación `pthread_exit()`
 - Espera de terminación `pthread_join()`
- Objetos de sincronización
 - Mutexes
 - Variables de condición
 - Semáforos
 - Cerrojos de lectura/escritura (*rwlocks*)

Introducción**Procesos y hebras****Principios de
programación
concurrente****Programación con
Pthreads**

- Funciones
- Ciclo de vida de una hebra
- Atributos

Problemas clásicos**Fuentes de
información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Ciclo de vida de una hebra

- En su forma más básica, la utilización de una hebra requiere los siguientes pasos
 - Incluir el fichero de cabecera `pthread.h`

```
#include <pthread.h>
```

- Declarar un descriptor

```
pthread_t idHebra ;
```

- Crear la hebra

```
pthread_create(&idHebra, NULL, funcion, argumentos) ;
```

- Terminar la ejecución de la hebra

```
pthread_exit(valor)
```

- Esperar que la hebra termine

```
pthread_join(idHebra, NULL)
```

Introducción**Procesos y hebras****Principios de
programación
concurrente****Programación con
Pthreads**

- Funciones
- Ciclo de vida de una hebra
- Atributos

Problemas clásicos**Fuentes de
información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Ciclo de vida de una hebra

- Ejemplo: programa simple.c

```

/*
 * Programa simple.c: ejemplo de ciclo de vida de una hebra
 */
#include <pthread.h>
#include <stdio.h>

void * codigoHebra(void *parametros) {
    printf("Soy la hebra\n") ;
} /* codigoHebra */

int main(int argc, char ** argv) {
    pthread_t idHebra ;
    int resultado ;

    resultado = pthread_create(&idHebra, NULL, codigoHebra, NULL) ;
    if (resultado != 0) { /* 0 indica éxito */
        printf("main: error al crear la hebra\n") ;
        return -1 ;
    } /* if */

    printf("Soy la hebra principal\n") ;

    pthread_join(idHebra, NULL) ;

    printf("Fin de la hebra principal\n") ;

} /* main */

```

Introducción**Procesos y hebras****Principios de
programación
concurrente****Programación con
Pthreads**

- Funciones
- Ciclo de vida de una hebra
- Atributos

Problemas clásicos**Fuentes de
información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Ciclo de vida de una hebra

- Ejemplo: programa simple.c
- Compilación del programa
 - En Solaris:


```
% cc simple.c -o simple -mt
```
 - En Linux:


```
% cc simple.c -o simple -D_REENTRANT -lpthread
```
- Ejecución

```

S mbolo del sistema - telnet 192.168.198.3
zeus% cc simple.c -o simple -mt -lpthread
zeus% simple
Soy la hebra principal
Soy la hebra
Fin de la hebra principal
zeus%

```

Introducción**Procesos y hebras****Principios de
programación
concurrente****Programación con
Pthreads**

- Funciones
- Ciclo de vida de una hebra
- Atributos

Problemas clásicos**Fuentes de
información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Atributos de una hebra

- Una hebra POSIX puede tener atributos
 - Cuando se crea una hebra, por defecto sus atributos tienen asignados un valor inicial
- Los atributos afectan, entre otros, a:
 - Tamaño de la pila
`pthread_attr_setstacksize()`
 - Dirección de la pila
`pthread_attr_setstackaddr()`
 - Ámbito
`pthread_attr_setscope()`
 - Planificación
`pthread_attr_setschedpolicy()`

Introducción**Procesos y hebras****Principios de
programación
concurrente****Programación con
Pthreads**

- Funciones
- Ciclo de vida de una hebra
- Atributos

Problemas clásicos**Fuentes de
información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Atributos de una hebra

- Ejemplo de uso de atributos
 - En Solaris, cuando una hebra se crea, por defecto es una hebra de usuario. Esto es así porque el atributo que afecta por defecto al ámbito de planificación de una hebra es el proceso
 - Para crear una hebra de núcleo, hay que crear la hebra con ámbito de sistema

```
#include <pthread.h>
...
pthread_attr_t atributos ;      // atributos de la hebra
pthread_attr_init(&atributos) ; // inicializacion

// Se establece el atributo de ambito
pthread_attr_setscope(&atributos, PTHREAD_SCOPE_SYSTEM) ;

resultado = pthread_create(&idHebra,
                           &atributos,
                           codigoHebra,
                           NULL) ;

...
```

Introducción**Procesos y hebras****Principios de
programación
concurrente****Programación con
Pthreads****Problemas clásicos**

- Productores/consumidores

**Fuentes de
información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Problemas clásicos de programación concurrente

- Existen varios problemas que, ya sea por ser instructivos o por su utilidad en la práctica, se han convertido en problemas clásicos de programación concurrente
- Entre estos problemas están:
 - Exclusión mutua
 - El problema de los filósofos
 - Lectores/Escritores
 - Productores/consumidores

Introducción**Procesos y hebras****Principios de
programación
concurrente****Programación con
Pthreads****Problemas clásicos**

- Productores/consumidores

**Fuentes de
información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Productores/consumidores

- Definiciones
 - Sea un buffer circular con una capacidad de MAX_BUF elementos
 - Un productor es un proceso o hebra que quiere insertar elementos en el buffer
 - Un consumidor es un proceso o hebra que quiere extraer elementos del buffer
- Enunciado: dado un conjunto de productores y de consumidores que se comunican a través de un buffer común, se deben cumplir dos condiciones
 - Un productor se ha de bloquear si intenta insertar un elemento cuando el buffer está lleno
 - Un consumidor se ha de bloquear si intenta extraer un elemento cuando el buffer está vacío

Introducción**Procesos y hebras****Principios de
programación
concurrente****Programación con
Pthreads****Problemas clásicos**

- Productores/consumidores

**Fuentes de
información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Implementación con Pthreads

- Requisitos

- Un buffer será una instancia de un tipo abstracto de datos `mt_buffer_t`. Los elementos del buffer serán números enteros
- Las operaciones del buffer serán seguras desde el punto de vista de su utilización en programas multihebras (*thread safe*)
- La especificación las operaciones es la siguiente:

```
...
/* funciones para crear y destruir un buffer */
int mt_buffer_init(mt_buffer_t *) ;
int mt_buffer_destroy(mt_buffer_t *) ;

/* funciones para acceder al buffer */
int mt_buffer_insertar(mt_buffer_t *, int) ;
int mt_buffer_extraer (mt_buffer_t *, int *) ;
int mt_buffer_vacio(mt_buffer_t *, bool *) ;
int mt_buffer_lleno(mt_buffer_t *, bool *) ;

...
```

Introducción**Procesos y hebras****Principios de
programación
concurrente****Programación con
Pthreads****Problemas clásicos**

- Productores/consumidores

**Fuentes de
información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Productores/consumidores

- Implementación del tipo `mt_buffer_t`

```
...
typedef struct {
    int buffer[MAX_BUF] ;
    int elemento          ; /* Primer elemento del buffer      */
    int hueco              ; /* Primer hueco del buffer */
    int num_elementos      ; /* Numero de elementos en el buffer */

    pthread_mutex_t mutex ;
    pthread_cond_t  esta_vacio ;
    pthread_cond_t  esta_lleno ;

} mt_buffer_t ;

...
```


Introducción**Procesos y hebras****Principios de
programación
concurrente****Programación con
Pthreads****Problemas clásicos**

- Productores/consumidores

**Fuentes de
información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Productores/consumidores

- Implementación de la operación
mt_buffer_extraer

```
int mt_buffer_extraer (mt_buffer_t * buf, int * elemento) {
    int error ; /* Variable usada en el control de errores */

    /*
     * Paso 1: se bloquea el mutex
     */
    error = pthread_mutex_lock(&(buf->mutex)) ;
    if (error != 0) {
        /* Error al bloquear el mutex */
        return error ;
    } /* if */

    /*
     * Paso 2: se espera hasta que haya algun elemento en el buffer
     */
    while (buf->num_elementos == 0) {
        error = pthread_cond_wait(&(buf->esta_vacio), &(buf->mutex)) ;
        if (error != 0) {
            /* Error al espera en la variable de condicion esta_vacio */
            return error ;
        } /* if */
    } /* while */

    /* -----> sigue */
```

Introducción**Procesos y hebras****Principios de
programación
concurrente****Programación con
Pthreads****Problemas clásicos**

- Productores/consumidores

**Fuentes de
información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Productores/consumidores

- Implementación de la operación
mt_buffer_extraer

```
/* <----- Viene de atrás */
/*
 * Paso 3: se extrae el elemento
 */
*elemento = buf->buffer[buf->elemento] ;
buf->elemento = (buf->elemento + 1) % MAX_BUF ;
buf->num_elementos -- ;

/*
 * Paso 4: se comprueba si el buffer estaba lleno, y se despierta
 * a un hipotetico productor que estuviese bloqueado en
 * esta_lleno
 */
if (buf->num_elementos == (MAX_BUF - 1))
    pthread_cond_signal(&(buf->esta_lleno)) ;
/*
 * Paso 5: se libera el mutex
 */
error = pthread_mutex_unlock(&(buf->mutex)) ;
if (error != 0) {
    /* Error al desbloquear el mutex */
    return error ;
} /* if */
return 0 ;
} /* mt_buffer_extraer */
```

Introducción**Procesos y hebras****Principios de
programación
concurrente****Programación con
Pthreads****Problemas clásicos**

- Productores/consumidores

**Fuentes de
información**

Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Productores/consumidores

- Ficheros:
 - `mt_buffer.h` contiene la especificación completa del tipo
 - `mt_buffer.c` contiene la implementación
 - `prodcons.c` contiene un ejemplo de uso de un buffer `mt_buffer_t` por parte de un productor y de un consumidor
 - `linux.mak` y `solaris.mak` son ficheros para compilar los programas en Linux y Solaris, respectivamente

Introducción**Procesos y hebras****Principios de
programación
concurrente****Programación con
Pthreads****Problemas clásicos****Fuentes de
información**

- Bibliografía
- Direcciones en la Web



Francisco López Valverde

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga

Fuentes de información

- Bibliografía
- Direcciones en la Web

Introducción**Procesos y hebras****Principios de
programación
concurrente****Programación con
Pthreads****Problemas clásicos****Fuentes de
información**

- Bibliografía
- Direcciones en la Web

- UNIX Programación Práctica. Guía para la concurrencia, la comunicación y los Multihilos
 - Kay A. Robbins, Steven Robbins
 - Prentice Hall, 1997
- Foundations of Multithreaded, Parallel and Distributed Programming
 - Gregory R. Andrews
 - Addison-Wesley, 2000

**Introducción****Procesos y hebras****Principios de
programación
concurrente****Programación con
Pthreads****Problemas clásicos****Fuentes de
información**

- Bibliografía
- Direcciones en la Web

- Referencia sobre Pthreads, tomada de la Web del OpenGroup
 - <http://www.unix-systems.org/version2/whatsnew/threadsref.html>
- Grupo de noticias de programación multihebra
 - <news://comp.programming.threads>
- Páginas que contienen enlaces relacionados con Pthreads
 - <http://www.cs.iastate.edu/~mayuresh/pthreads.shtml>
 - <http://www.humanfactor.com/pthreads/pthreadlinks.html>

