

**DEPARTAMENTO DE SISTEMAS DE INFORMACIÓN**

# Sistemas Operativos

Laboratorio 6

Race Conditions y Sincronización de procesos

# INTRODUCCIÓN

Las Race Conditions surgen en entornos multihilo cuando varios hilos acceden o modifican estructuras de datos compartidas simultáneamente. Las Race Conditions no solo llevaran a resultados incorrectos, sino también a comportamientos indefinidos. Por lo tanto, para prevenir las Race Conditions, se deben utilizar mecanismos de sincronización adecuados.

# OBJETIVOS

El objetivo de este laboratorio es que el estudiante sea capaz de presentar los problemas de acceso a las secciones críticas, cuyas soluciones pueden utilizarse para asegurar la coherencia de los datos compartidos. Aprenderá sobre las condiciones de carrera, uno de los errores más sutiles de la programación concurrente, y aprenderá como coordinar diferentes hilos para evitar condiciones de carrera con primitivas de sincronización que incluyen mutex, spinlocks y variables condicionales.

wget --no-check-certificate 'https://docs.google.com/uc?export=download&id=1\_QAfT8qQ36mgZa1rGrlDnPLr5HNbLeNi' -O Lab06.zip

# ANTECEDENTES: RACE CONDITIONS

Una Race Condition ocurre cuando dos o más hilos o procesos acceden a datos compartidos simultáneamente, y al menos uno de los hilos modifica la estructura de datos.

Los riesgos de una Race Condition permitirían que algún atacante pueda tomar ventaja del intervalo de tiempo que se da entre el inicio de un servicio y el momento en que un control de seguridad surte efecto.

**Ejercicio 1:** Compila y ejecuta el código race\_condition.c y responde las siguientes preguntas **(20pts)**:

1. **¿Cuál es su salida?**

La salida es variable y nos entrega estos valores:  
Texto

Descripción generada automáticamente

1. **¿Cuál es la salida esperada?**

La salida esperada debería ser 0 ya que suma elementos hasta la mitad de la cantidad de threads y luego resta para la otra mitad.

1. **¿Cómo y por qué se genera esta salida?**

Se genera por el conflicto entre los hilos, ya que los datos se sobreponen. En palabras técnicas se producen condiciones de carrera.

**PROTEGIENDO DATOS CON MUTEX**

Para prevenir las Race Conditions, debemos utilizar mecanismos de sincronización. Los mecanismos de sincronización comunes incluyen Mutex, Semáforos, Spinlocks y variables condicionales, entre otros. Mutex es un mecanismo de sincronización básico en la programación concurrente, que impone el acceso exclusivo a los recursos.

La librería pthread proporciona principalmente las siguientes funciones clave para implementar operaciones con mutex:

#include <pthread.h>

Int pthread\_mutex\_init(pthread\_mutex\_t \*mutex, const pthread\_mutexttr\_t \*attr);

Int pthread\_mutex\_lock(pthread\_mutex\_t \*mutex);

Int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);

Int pthread\_mutex\_trylock(pthread\_mutex\_t \*mutex);

Int pthread\_mutex\_destroy(pthread\_mutex\_t \*mutex);

Entre ellas, la función pthread\_mutex\_init se utiliza para inicializar el mutex. Puedes optar por pasar atributos específicos o usar los atributos predeterminados. Cuando un hilo desea acceder a un recurso o segmento de código protegido por un mutex, debe llamar a pthread\_mutex-lock para intentar adquirir el mutex. Si el mutex ya está en posesión de otro hilo, el hilo se bloqueará hasta que el mutex sea liberado.

**Ejercicio 2:** Completa el código en mutex.c para proteger la variable compartida con un mutex. Compila y ejecuta el código, responde la siguiente pregunta **(20pts)**:

1. **¿Cuál es la salida?**

La salida obtenida es la esperada, 0:

Texto

Descripción generada automáticamente

**PROTEGIENDO DATOS CON SPINLOCKS**

SpinLock es otro mecanismo de sincronización que garantiza la ejecución exclusiva. A diferencia de Mutex, cuando un hilo intenta adquirir un spin lock ya ocupado, girará en un bucle comprobando la disponibilidad de bloqueo. Los spinlocks pueden ofrecer una mayor eficiencia cuando la contención no es grave, debido a que evitan la sobrecarga del cambio de contexto de los hilos. Pero en escenarios de alta contención, también pueden llevar a un desperdicio de recursos de CPU.

En la librería pthread, la operación de spinlock implica principalmente las siguientes funciones clave:

#include <pthread.h>

int pthread\_spin\_init(pthread\_spinlock\_t \*lock, int pshared);

int pthread\_spin\_lock(pthread\_spinlock\_t \*lock);

int pthread\_spin\_unlock (pthread\_spinlock\_t \*lock);

int pthread\_spin\_trylock (pthread\_spinlock\_t \*lock);

int pthread\_spin\_destroy(pthread\_spinlock\_t \*lock);

Entre ellas, la funcion pthread\_spin\_init se utiliza para inicializar un spinlock, aceptando un puntero de bloqueo y un flag para determinar si el bloqueo debe ser compartido entre procesos. Para adquirir un spin lock, un hilo utiliza la función pthread\_spin\_lock. Si el bloqueo ya está en posesión de otro hilo, seguirá intentando adquirirlo hasta que lo consiga.

**Ejercicio 3:** Completa el código en spinlock.c para proteger la variable compartida con spinlocks. Compila y ejecuta el código, responde la siguiente pregunta **(20pts)**:

1. **¿Cuál es la salida?**

La salida para este ejercicio es la misma que se obtuvo al usar mutex, la correcta:

Texto

Descripción generada automáticamente

**Ejercicio 4:** Al proteger estructuras de datos compartidas, ¿Cuándo deberíamos usar un mutex en lugar de un spinlock? Puede que no haya una respuesta simple a esta pregunta. Aquí, intentaremos responder esta pregunta según un criterio simple: el tiempo de ejecución. Por favor, intenta medir el tiempo de ejecución para los programas en los ejercicios 2 y 3 anteriores, respectivamente. Responde la siguiente pregunta **(20pts)**:

1. **¿Cuál se ejecuta de manera más eficiente? (no necesitas escribir ningún código nuevo, investiga algún comando que te sirva para medirlo)**

Después de utilizar el comando *time* al momento de ejecutar los códigos se evidencia que el **spinlock** tiene mejores valores en todos los tiempos a evaluar, por lo que seria el mas optimo, por lo menos para esta clase de ejercicios.

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

**VARIABLES CONDICIONALES**

Una variable condicional es un mecanismo de sincronización utilizado para la comunicación entre hilos, el cual evita la espera activa.

En la librería pthread, la operación de las variables condicionales implica principalmente las siguientes funciones clave:

#include <pthread.h>

int pthread\_cond\_init(pthread\_cond\_t \*cond, const pthread\_condattr\_t \*attr);

int pthread\_cond\_wait(pthread\_cond\_t \*cond, pthread\_mutex\_t \*mutex);

int pthread\_cond\_signal(pthread\_cond\_t \*cond);

int pthread\_cond\_broadcast(pthread\_cond\_t \*cond);

int pthread\_cond\_destroy(pthread\_cond\_t \*cond);

Entre ellas, la función pthread\_cond\_init inicializa la variable condicional cond basada en el atributo attr proporcionado (o en el atributo predeterminado). Cuando un hilo necesita esperar a que cierta condición cond se cumpla, llamará a la función pthread\_cond\_wait y se dormirá sobre la variable condicional cond.

Usando variables condicionales junto con mutex, podemos resolver el problema de múltiples productores-consumidores de forma más elegante, donde múltiples productores colocan elementos de datos en un búfer compartido, mientras que múltiples consumidores obtienen elementos de datos de ese búfer simultáneamente.

**Ejercicio 5:** Completa el código en produce-consumer.c, para proteger el búfer compartido con variables condicionales junto con mutex **(20pts)**.

**Resultados obtenidos:**

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente con confianza media

**Código Utilizado:**

Texto

Descripción generada automáticamente

# Entrega

Documente (screenshots) cada una de las acciones antes señaladas. Es de exclusiva responsabilidad del estudiante respetar el formato de entrega de informe de esta guía **(debajo de cada enunciado su screenshot en donde aparezca de forma clara las sentencias utilizadas)**. El formato de entrega debe ser en PDF, y el nombre del archivo debe contener su nombre y apellido (**Laboratorio\_6\_Nombre\_Apellido**). Todas las actividades deben ser entregadas (subidas) a la plataforma digital en las fechas establecidas, por cada hora de atraso, se descontará 1 pto.