Práctica de programación concurrente y consistencia de memoria

J. Daniel García Sánchez (coordinador)
David Expósito Singh
Javier García Blas
Óscar Pérez Alonso
J. Manuel Pérez Lobato

Arquitectura de Computadores
Grupo ARCOS
Departamento de Informática
Universidad Carlos III de Madrid

1. Objetivo

Esta práctica tiene como objetivo fundamental familiarizar al estudiante con la programación libre de cerrojos y compender el impacto que puede tener en el rendimiento de una aplicación. En concreto, se evaluarán las ventajas e inconvenientes de uso de tipos de datos atómicos frente a las técnicas básicas basadas en cerrojos.

2. Descripción

En esta práctica se evaluarán las distintas alternativas para implementar un buffer acotado circular (*circular bounded buffer*). Para evaluar la seguridad frente a hilos de ambas estructuras de datos se usarán dos técnicas: programación basada en cerrojos y programación libre de cerrojos. En ambos casos se hará uso del estándar ISO/IEC 14882:2011 (C++11).

2.1. Material suministrado

Para la realización de la práctica se suministra la implemntación de un búfer acotado circular en el archivo **seq_buffer.h**.

2.2. Evaluación del rendimiento

Para realizar evaluación del rendimiento, se pueden usar algunos de los siguientes métodos:

- Medición de tiempos usando la biblioteca estándar de C++ (espacio de nombres **chrono**).
- Acceso al módulo del núcleo de Linux `perf`.

3. Tareas

3.1. Estudio inicial

Estudie el código fuente suministrado y analice su funcionamiento.
Incluya en un apartado de su memoria una explicación detallada de la estructura de datos.

- Indique bajo qué condiciones se elevan excepciones.
- ¿Qué utilidad tienen los datos miembro **next_read_** y **next_write_**.
- ¿Cuál es el máximo número de elementos que pueden almacenarse?

Analice también el programa principal y explique detalladamente su funcionamiento.

3.2. Ejecución concurrente con cerrojos

Escriba una versión de la estructura de datos que sea *thread-safe* para el caso de un único productor y un único consumidor. Utilice para la implementación **mutex** y variables condición. Esta versión debe sustituir las excepciones por esperas bloqueantes.

Escriba también una versión del programa principal en la que un hilo actúa como productor y otro hilo actúa como consumidor.

3.3. Ejecución concurrente libre de cerrojos

Escriba una versión de la estructura de datos que sea *thread-safe* y libre de cerrojos usando tipos atómicos. Esta versión debe sustituir las excepciones por esperas activas usando un modelo de consistencia de memoria. Considere el uso de consistencia secuencial frente a modelos más relajados.

Escriba también una versión del programa principal en la que un hilo actúa como productor y otro hilo actúa como consumidor.

3.4. Evaluación del rendimiento

Evalúe las tres soluciones en un computador con al menos dos núcleos. Realice las evaluaciones que considere más adecuadas para determinar las ventajas e inconvenientes de las distintas aproximaciones.