DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2

Pthreads

8 de Junio de 2016 — Sistemas Operativos

Integrante	LU	Correo electrónico
Costa, Manuel José Joaquin	035/14	manucos94@gmail.com
Coy, Camila Paula	033/14	camicoy94@gmail.com
Ginsberg, Mario Ezequiel	145/14	ezequielginsberg@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

0.1. Read-Write Lock

Para implementar el `RWLock` utilizamos los siguientes atributos privados:

- `mutex`
- `cv_write`: Una variable de condición para la variable `write`.
- `cv_read`: Una variable de condición para la variable `cant_reads`.
- `cant_reads`: Tiene la cantidad de lecturas que se estan realizando.
- `write`: Un booleano que cuando llega un write se pone en true. Lo usamos para evitar que los threads que realizan reads le generen inanición a los threads que quieren hacer write.

Además usamos los siguientes métodos públicos:

- `RWLock()`: Inicializa el mutex, las variables de condición, `cant_reads` y `write`.
- `void rlock()`: Genera el lock para los reads. Si llega un write deja esperando los reads para que no pueda haber inanición, luego aumenta `cant_reads`, ya que hay alguien leyendo.
- `void runlock()`: Decrementa `cant_reads` indicando que dejo de leer. Si después de eso `cant_reads` = 0 entonces envía un broadcast a todos los threads que están esperando por la variable de condición `cv_write`.
- `void wlock()`: Genera el lock para los writes. Si llegan reads o writes, quedan bloqueados hasta que se haga el `void wlock()`. Notar que lockea el `mutex` y no lo libera.
- `void wunlock()`:

0.2. Servidor

Para el servidor nos basamos en `backend-mono`, haciendole los cambios necesarios para que pudo soportar más de un jugador. Cada vez que acepta a un jugador crea un thread para atenderlo mediante la función `atenderdor_de_jugador` y guardamos la información del thread en un vector. Además creamos dos vectores de vectores de RWLock para proteger ambos tableros y todos los lugares donde en `backend-mono` se escribío o leyo algun tablero agregamos las protecciones de este. Para poder saber cuando todos los jugadores mandaron el mensaje de LISTO y poder pasar a la fase de batalla, creamos la variable `cant_clientes` y la protegimos para que no la puedan modificar dos threads a la vez. Entonces cada vez que entra un cliente entra aumentamos en uno esta variable, y cada vez que un thread muere o un jugador dice LISTO esta decrementa.