DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico II

"SOScrabel"

Sistemas Operativos
Primer Cuatrimestre 2015

Integrante	LU	Correo electrónico
Aldasoro Agustina	86/13	agusaldasoro@gmail.com
More Ángel	931/12	angel_21_fer@hotmail.com
Zimenspitz Ezequiel	155/13	ezeqzim@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

1. Read-Write Lock

Implementamos los algoritmos de ***Read-Write Lock*** libre de inanición utilizando unicamente *mutex* y respetando la interfaz provista en los archivos `backend-multi/RWLock.h` y `backend-multi/RWLock.cpp`.

Nuestro objetivo es restringir el acceso a la variable `tablero`, de modo que puedan leer los datos en él simultaneamente diversa cantidad de *lectores* pero sólo puedan *escribir* de a uno por vez y cuando nadie este leyendo.

La exclusión mutua que se lleva a cabo se comporta de modo que la ejecución de un thread en la sección crítica no excluye a otros threads que ingresen a la misma. Sin embargo, si determinado tipo de thread es quien se encuentra en la sección crítica prohibe a las otras categorías de threads ingresar a ella.

Armamos la clase ***Lightswitch***, la cual contiene un contador (`count`) el que llevará la cantidad de gente que se encuentra leyendo el tablero y un mutex (`mut`) para acceder al mismo.

Las funciones pertinentes a la clase serán: *lock* y *unlock*. Serán invocadas solamente por los *readers* para leer y dejar de leer el tablero respectivamente. Su comportamiento es simple: se toma el mutex para actualizar la variable `count` ($+1$ si es lock, -1 unlock) y luego sólo modificarán al mutex `m` pasado por parámetro si es el primero en leer o si es el último en dejar de leer.

Si ingresa un nuevo lector y no había nadie leyendo el tablero ($count == 1$), hará un wait del mutex `m`. Análogamente si es el lector que al retirarse, dejará al tablero vacío ($count == 0$) hará un signal liberando al mutex `m`.

Las variables con las que trabajarán las funciones Read-Write Lock serán tres: un Lightswitch llamado **readSwitch** y dos mutex **turnstile** y **roomEmpty**.

El `readSwitch` será el Lightswitch que se comportará acorde a lo explicado anteriormente unicamente para el uso del *reader*. De modo que permitirá a múltiples lectores acceder al tablero de manera simultánea.

Por otro lado, vamos a tener al mutex `roomEmpty`, el cual se va a habilitar unicamente cuando nadie este leyendo el tablero (sea leyendo o escribiendo). Es preciso aclarar que `roomEmpty` es el mutex que será pasado por parámetro a la hora de invocar a las funciones del `readSwitch`.

Por último, el mutex `turnstile` será el encargado de impedir la inanición. Los *escritores* son quienes podrían correr riesgo de *inanición* si sucede que llega una escritura, pero también lecturas y por no ejecutarse en orden siempre se bloquea el acceso al tablero por los *lectores*.

Tanto los *lectores*, como los *escritores* deberán pasar por el mutex `turnstile`, haciéndole wait. Pero los *lectores* le darán signal en la instrucción siguiente, mientras que los *escritores* lo harán después de esperar al mutex roomEmpty.

De esta manera, si un escritor no puede avanzar en el mutex `turnstile` obliga a los lectores que lleguen a "ponerse en espera" del `turnstile`. Y cuando el último lector de los que estaban en el tablero, lo abandone, permitirá al escritor tener acceso al tablero antes de cualquiera de los lectores que se ejecutaron después de él y estaban esperando.

2. Servidor Backend

El informe debera ser de caracter breve e incluir el pseudocodigo de los algoritmos que se ejecutan en el servidor de backend frente a cada peticion de un cliente, poniendo enfasis en las primitivas de sincronizacion al estilo del primer parcial de la materia. Si fuera necesario, puede ser buena idea incluir una explicacion del funcionamiento del servidor en lenguaje natural. Cualquier decision de diseno que hayan tomado debera ser incluida aquı. La implementacion que realicen del servidor de backend debe estar libre de condiciones de carrera y presentar la funcionalidad descripta arriba a cada uno de los clientes. A su vez, debe:

Permitir que multiples clientes se conecten al backend de forma simultanea.

Permitir que todos los jugadores coloquen letras en casilleros distintos de forma simultanea.

Permitir que varios clientes consulten el estado del tablero de forma simultanea.