DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico I

22 / 04 / 2015 Sistemas Operativos

Integrante	LU	Correo electrónico
Abdala, Leila	950/12	abdalaleila@gmail.com
Enrique, Natalia	459/12	natu_2714@hotmail.com
Salinas, Pablo	456/10	salinas.pablom@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

1. Introducción

En este informe presentaremos la implementación de diversos schedulers y una breve experimentación comparativa entre estos. La idea de este TP es conocer las distintas formas de administrar el scheduler, entendiendo asi que ventajas presenta cada modo y en que contexto. El desarrollo del informe se basa en el enunciado, por lo que cada item del mismo tiene en

2. Ejercicios

2.1. Ejercicio 1: Read-Write Lock

Para este ejercicio nos basamos en el libro *"The Little Book of Semaphores"(Seccion 4.2)*. En la misma se analiza y resuelve el problema en el cual varios threads escriben o leen los mismos datos. Pueden leer varios al mismo tiempo, pero solo uno podra escribir. Al igual que en nuestro Scrabble, queremos que esto suceda para evitar inanicion. Incluimos como variables un pthread_mutex_t llamado *roomEmpty* el cual se activa cuando no hay threads (leyendo o escribiendo) en la seccion critica, y un int *cantidadDeSolicitudes* que son la cantidad de threads leyendo.

Para la lectura modificamos las funciones de la siguiente forma:
En el caso del lock

```
void RWLock::rlock(){
IF (cantidadDeSolicitudes == 0){ //No habia nadie en la seccion critica
    pthread_mutex_lock(&roomEmpty); //Bloqueo roomEmpty porque ahora ingresara alguien y dejara de estarlo
}
cantidadDeSolicitudes++; //Aumento en uno cantidadDeSolicitudes por el nuevo ingresante;
}
```

Para el unlock

```
void RWLock::runlock(){
cantidadDeSolicitudes--;          //Disminuyo cantidadDeSolicitudes porque uno dejara de leer
IF (!cantidadDeSolicitudes){               //Si no queda nadie en la seccion critica
    pthread_mutex_unlock(&roomEmpty);        //activo la sen\~al de roomEmpty para el que quiera escribir
}
}
```

En el caso de la escritura no debemos permitir que haya mas de un thread en la seccion critica. En el caso del lock:

```
void RWLock::wlock(){
pthread_mutex_lock(&roomEmpty); //Bloqueamos la variable roomEmpty para que nadie pueda acceder
}
```

Para el unlock:

```
void RWLock::wunlock(){
pthread_mutex_unlock(&roomEmpty); //Liberamos roomEmpty para que puedan acceder otros
}
```

2.2. Ejercicio 2: Backend Multithreaded