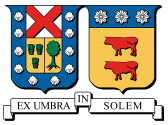

Universidad Técnica Federico Santa María
Campus Santiago San Joaquín
Primer Semestre 2016

Tarea 3 Sistemas Operativos

Rodrigo Elicer
201473539-1

Diciembre 2016

1 Problema de Escritura y Lectura confluente

La solución al problema fue realizado en el lenguaje C. Se utilizaron las liberías stdio.h, pthread.h y semaphore.h y un archivo 'entrada.txt' el cual iba a tomar la función del recurso el cual solo podía ser usado por un solo thread.

En el código fuente se pueden ver 4 funciones. **main()**, **proceso()**, **write()** y **read()**.

main() lo que hace es pedir por consola la cantidad de lectores y escritores que van a interactuar en el problema. Este no puede ser mayor a 100, número impuesto arbitrariamente para poder acotar el problema. Finalmente ingresa estos valores a la función **proceso()**.

La función **proceso()** inicia los semáforos mutex, queue, escritor y lector en 1. Luego se crean los lectores y luego los escritores.(Para ver que efectivamente se toma prioridad apenas llegue un escritor) Finalmente pthread_join lo que hace es esperar a que los procesos anteriores terminen.

Como última instancia están las funciones **read()** y **write()** cuyas funcionalidades son las esenciales para que se cumpla lo planteado por el problema. Estos se explicarán a continuación.

Es justo mencionar que para que los threads esperen algo de tiempo y se empiecen a "amontonar" en la lista de espera se hizo que tanto el lector como el escritor escribieran en el archivo 100 veces "LECTOR" y 10 veces "ESCRITOR" respectivamente, con el fin de poder ver mejor la preferencia de lectores frente a escritores.

1.1 Solución al problema a través de semáforos

Para solucionar el problema se hizo uso de 4 semáforos, lector, escritor, queue y mutex, todos inicializados en 1.

En la función **read()** se hace un wait al semáforo *queue*, cuya funcionalidad es dejar entrar solo a 1 lector, teniendo que pasar después por el semáforo *lector*. Este semáforo *queue* lo que hace es encolar a los lectores para darle preferencia a los escritores. Luego se hace un wait al semáforo *lector* para poder dejar que solo 1 lector llame al recurso, haciendo un wait al semáforo *mutex*. Luego se

libera el semáforo *lector*, permitiendo que la función **write()** pueda bloquear a un lector mientras espera a que el recurso quede liberado. (Se explicará más adelante). Cuando el lector deje de leer liberará el recurso y se saldrá de la cola.

La función **write()** hace uso de 3 semáforos, *mutex*, *escritor* y *lector*. Este consiste de 2 secciones wait-post (wait-signal) utilizando el semáforo *escritor*. La primera sección lo que hace es sumar 1 a un contador, procurando que cuando este sea 1 (o sea que haya sido el primero o que haya solo un escritor pidiendo el recurso), se haga un wait al semáforo *lector*, impidiendo que estos puedan pedir el archivo. Luego se utiliza el semáforo *mutex* para la zona crítica.

La segunda sección wait-post del semáforo *escritor* consiste básicamente en ir restando 1 al contador. Cuando este llegue a 0 (indicando que no hay ningún escritor usando el recurso), recién ahí se hace un post al semáforo *lector*, significando que tanto un lector como un escritor pueden abrir el archivo dependiendo de cuál llegue primero, o bien le de preferencia a un lector.

La situación descrita anteriormente se fija que si ya hay un escritor utilizando el recurso, ningún lector puede entrar a pedirlo, sin embargo esto si lo puede hacer un escritor, quedando en espera gracias al contador. Cuando estos vayan saliendo el contador irá restando en 1, permitiendo abrir el semáforo *lector* solo cuando este llegue a 0.