Barreras en memoria compartida

daryl.butron@ucsp.edu.pe

Abril 2021

1. Introducción

Hay un problema en la programación con memoria compartida, el problema es sobre sincronizar los subprocesos para asegurarnos que todos estén en un punto sobre algun programa dado. Esta sincronización es llamada barrera, ya que ningún thread puede seguir avanzando mas allá de esta barrera dada. Esto tiene varias aplicaciones como por ejemplo revisar si una vez terminada la carrera de los threads verificar cual fue mas lento al finalizar una tarea dada a este thread.

2. Busy-Wait y Mutex

Para implementar una barrera usando Busy-Wait y un mutex se requiere hacer lo siguiente:

1. Usar un contador compartido que este protegido por el mutex.
2. Verificar con el contador si todos los subprocesos ingresaron a sección crítica
3. Si se cumple el paso anterior, entonces dejar un bucle busy-wait.

Esta estrategia desperdiciara ciclos en CPU cuando los threads esten bucle busy-wait, además si ejecutamos un programa en el cual le damos mas threads que nucleos que pueda soportar entonces se producira degradación de perfomance al código.

3. Semaforo

Aqui también se utiliza un contador para saber cuantos threads han ingresado a la barrera además de otro contador que sera para contar los

semaforos. Aqui no importa si el thread esta ejecutando el bucle de llamadas o si estan bloqueadas, ya que eventualmente estas terminaran y llegaran al mismo punto. Esta estrategia de usar semaforos es mejor que la anterior, ya que los threads no consumen ciclos en CPU cuando estan bloqueadas en un semaforo de espera.

4. Variable Condicional

Este utliza un objeto llamado variable de condición que permite a los threads se encuentren en modo de suspensión de ejecución hasta que suceda algo que active la variable de condición y por esto despierte. Una variable de condición siempre esta asociado con un mutex. Para trabajar con esta estrategia se requiere el tipo de dato pthread$_{c}ond_{t}parahacerdeseñalesobroadcast.Esesencialquesellamecorr$

5. Barrera de Pthreads

Para este caso se requiere del uso de 3 metodos:

1. pthread$_{b}arrier_{i}nitparainicializarlafuncionylosthreadsrequeridosausarpthread_{b}arrier_{w}aitpara$

2. pthread$_{b}arrier_{d}estroyparadestruiryunirlosthreads$

6. Observaciones y Conclusiones

1. La estrategia de usar Busy-Wait junto con el mutex es la peor solución posible en caso de usar clocks en CPU

2. Las variables de condición trabajan como un join en threads

3. Propiamente en pthreads se ha desarrollado 3 funcionalidades que permiten inicializar, esperar y cerrar o destruir los threads una vez estos terminen sus procesos en carrera.