Universidad Nacional Autónoma De México
Facultad De Ciencias

Computación Concurrente

Práctica 3

Fecha:

18 de Abril del 2019

Nombre:

Muñiz Patiño, Andrea Fernanda.
No. de cuenta 311175318

1. Problema a resolver

La práctica 3 consistirá en resolver el problema **Productores - Consumidores**, se usarán semáforos que se encuentran en la biblioteca *semaphore* de **C**, pues mediante ellos se resolverá este problema.

1.1. Conceptos previos

- **Exclusión mutua:** Para resolver el problema de la exclusión mutua se debe cumplir con las siguientes 3 condiciones:
  - Mutex: En cualquier momento hay como máximo un proceso en la sección crítica
  - Garantía de entrada: cada hilo/proceso que tenga que entrar a la sección crítica, lo logrará eventualmente.
  - Ausencia de Deadlocks: No habrá ningún bloqueo, es decir aunque varios procesos intenten entrar a la sección crítica un proceso lo lograra.

  Esto es importante, puesto que el algoritmo que propondremos deberá cumplir esas características.
- **Semáforo:** Estructura de datos concurrente que consta de un contador, una cola de procesos y tiene las operaciones *wait* y *signal*
- **Consumidor:** Aquel proceso que extrae datos del buffer
- **Productor:** Aquel proceso que coloca datos en el buffer

1.2. Definición del problema: Productores Consumidores

Este problema también es conocido como el problema del buffer limitado. Consiste en darle acceso concurrente a un recurso compartido, llamado buffer, a los productores y consumidores. El *buffer* tiene un tamaño limitado, digamos que tiene un tamaño fijo n.

Los productores introducen elementos en el *buffer*, los consumidores extraen elementos del *buffer*. Esto debe ser consistente, para todos en todo momento.

1. Problema que radica en que si el buffer esta lleno y el productor intenta introducir un elemento esto ocasionara un error.
2. Otro problema sería cuando un consumidor quiere extraer del buffer un elemento y este esta vació. Dicho de otra manera, los consumidores solo pueden acceder al buffer cuando haya elementos en el.

La pila/buffer del problema debe ser mutex, pues no deben acceder al mismo tiempo a este elemento ya que podría estarse modificando los valores del buffer por dos procesos (o más) al mismo tiempo, lo cual arrojaría información corrompida.

Este problema surge cuando un proceso A, genera información que debe utilizar un proceso B, es decir el proceso A pondrá en memoria elementos que debe esperar el proceso B para poder continuar con la ejecución.

2. Como se resolverá

Para resolver el problema 1, simplemente dormiremos al productor cuando el buffer este lleno, esperando a que un consumidor tome un elemento y libere un espacio.

El problema 2, se resuelve con un semáforo que nos indique cuantos elementos tiene el buffer, si el contador es igual a 0 entonces dormimos al consumidor hasta que un productor coloque un elemento en el buffer.

Para controlar el acceso a la región critica, usaremos un semáforo para que a la vez solo haya un proceso modificando el buffer y no tener un traslape de información y/o generar datos erróneos.

El **Buffer** será modelado con una variable entera, a la cual si un consumidor quiere generar un elemento, simplemente se incrementara esta variable y un consumidor, simplemente la decrementara.

3. Algunas observaciones

Inicialmente el tamaño del buffer es 100, el número de iteraciones es 2.
Esto se puede cambiar con las directivas/macros en el código.

Los semáforos deben ser globales por que se llamarán las funciones *wait* y *signal* desde distintos procesos, en la biblioteca *semaphore*, son las funciones: *wait* y *post*.

Inicialmente el buffer estará vacío para esta implementación.

Para determinar que hilo será productor o consumidor se define un intervalo, los hilos que estén en el intervalo $0 \leq id\_Hilo < n$ serán consumidores, donde n es el número de consumidores solicitados.

4. Sobre compilación

Para compilar el programa basta con hacer

$ gcc Concurrente_Practica3.c -fopenmp -o salida.out

Sea p el número de productores en el programa y c el número de consumidores.

Para que funcione la implementación, al haber pedido las condiciones de iterar un número de terminado de veces los hilos, los parámetros tienen que cumplir las siguientes condiciones:

$$c \leq p$$

Es decir dónde los consumidores son menores que los productores, pues planteemos el caso en que cada hilo correspondiente a ser un productor terminó de ejecutar sus n iteraciones indicadas, al terminar de producir, si se ejecuta un número mayor de consumidores habrá un *deadlock* puesto que algún consumidor α se quedará esperando a que algún hilo productor genere un elemento en el buffer. Y esto no sucedera.

Para ejecutar el programa:

$ ./salida.out «**p**» «**c**»

Referencias

[1] Andrew S. Tanenbaum, *Sistemas operativos modernos*, tercera edicion, Prentice Hall, Mexico, DF, 2009.