Universidad de Guadalajara

Centro Universitario De Ciencias Exactas E Ingenieritas CUCEI

Ingeniería Informática

Actividad 11 – Productor Consumidor

Juan Antonio Ramírez Aguilar

Código: 212482507

Mtra. Becerra Velázquez Violeta del Rocío

Seminario de Solución de Problemas de Uso, Adaptación, Explotación de Sistemas Operativos

Indicé

I. El problema Productor Consumidor

El problema productor-consumidor en C es uno de los problemas más famosos asociados con los sistemas operativos.

En el problema productor-consumidor en C, hay un productor que produce productos (datos) y hay un consumidor que consume los productos producidos por el productor. Ahora, en el problema productor-consumidor en C, hay un buffer. Un buffer es un área de almacenamiento temporal en la memoria que almacena datos. El tampón en el problema productor-consumidor en C contiene los artículos producidos por el productor. El consumidor consume los productos del mismo amortiguador. En términos más sencillos, podemos decir que el buffer es un espacio compartido entre el productor y el consumidor.

Antes de entrar en el problema productor-consumidor en C, vamos a discutir brevemente el trabajo del productor y el consumidor.

- **Productor**: El papel del productor es producir o generar los datos y ponerlos en el buffer, y comenzar de nuevo con el mismo.
- **Consumidor**: El papel del consumidor es consumir los datos producidos del mismo buffer compartido. Cuando el consumidor consume los datos, los datos se eliminan de la memoria intermedia.

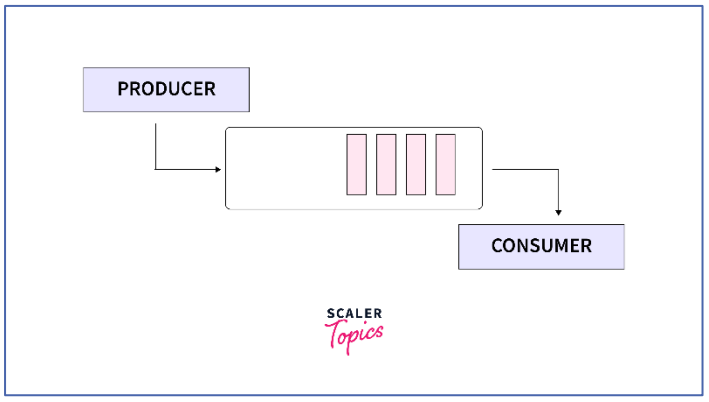

Imagen 1. Producto y Consumidor.

En el problema productor-consumidor en C, se nos ha proporcionado un tampón de tamaño fijo. El problema establece que puede surgir una situación en la que la memoria intermedia está llena y el productor todavía está produciendo el producto de manera similar, el consumidor intenta consumir el (los) producto (s) cuando no hay ningún producto que consumir de la memoria intermedia, por lo que ambos paralelos no podrán realizar sus trabajos. Por lo tanto,

tenemos que solucionar este problema para que el productor no intente producir los datos en el búfer cuando el búfer está lleno y, por otro lado, el consumidor no intenta consumir los datos producidos cuando el búfer está vacío. Debemos saber que el problema productor-consumidor en C es un ejemplo del problema de la sincronización multiproceso.

II. ¿Cuál es la solución para el problema productor-consumo?

Dado que el problema es que el productor sigue produciendo los datos incluso si la memoria intermedia está llena y el consumidor intenta consumir los datos incluso si la memoria intermedia está vacía. Podemos poner al productor a dormir o descartar los datos producidos por el productor si el buffer está lleno. Por lo tanto, siempre que el consumidor intente consumir los datos del buffer, notificará al productor que los datos se están consumiendo. Esto hará que el productor vuelva a producir los datos.

Del mismo modo, podemos poner al consumidor a dormir cuando no hay datos en el buffer (caso de buffer vacío). Por lo tanto, siempre que el productor produzca los datos y los ponga en el buffer, notificará al consumidor que los datos se están produciendo. Esto hará que el consumidor vuelva a consumir los datos.

También debemos notar que, si se propone una solución inadecuada, entonces puede surgir una situación en la que tanto el productor como el consumidor están a la espera (para ser despertados).

III. Propuesta de Solución

Para implementar una solución robusta y eficiente al problema del productor-consumidor, se propone el uso de semáforos. Esta es una técnica clásica de sincronización que permite gestionar el acceso a recursos compartidos de manera ordenada y segura, evitando las condiciones de carrera y los interbloqueos mencionados anteriormente.

La solución se basa en la implementación de tres semáforos distintos que controlarán las tres restricciones críticas del problema:

1. **Acceso Exclusivo al Buffer:** Solo un proceso (ya sea el productor o el consumidor) puede modificar el buffer en un momento dado.
2. **Control del Buffer Lleno:** El productor debe detenerse y esperar si el buffer está lleno.
3. **Control del Buffer Vacío:** El consumidor debe detenerse y esperar si el buffer está vacío.

Para gestionar estas condiciones, se utilizarán los siguientes semáforos:

- **mutex (Exclusión Mutua):** Un semáforo binario (con valor inicial de 1) que actúa como un cerrojo. Antes de acceder al buffer, cualquier proceso debe adquirir este semáforo. Al terminar, debe liberarlo. Esto garantiza que solo un proceso a la vez se encuentre en su **sección crítica** (el bloque de código que manipula el buffer).
- **full (Lleno):** Un semáforo de conteo que registra el número de espacios **ocupados** en el buffer. Su valor inicial es 0. El consumidor esperará en este semáforo si su valor es cero (buffer vacío).
- **empty (Vacío):** Un semáforo de conteo que registra el número de espacios **disponibles** en el buffer. Su valor inicial es N (la capacidad total del buffer, en nuestro caso, 15). El productor esperará en este semáforo si su valor es cero (buffer lleno).

Algoritmo del Productor

El proceso del productor seguirá un ciclo continuo con los siguientes pasos:

1. **Producir un elemento:** Genera un nuevo dato o producto.

2. **Esperar por un espacio vacío:** Ejecuta una operación wait() sobre el semáforo empty. Si empty es mayor que cero, decrementa su valor y continúa. Si es cero (buffer lleno), el productor se bloquea (se "duerme") hasta que el consumidor libere un espacio.
3. **Adquirir el cerrojo:** Ejecuta una operación wait() sobre el semáforo mutex para obtener acceso exclusivo al buffer.
4. **Añadir el elemento:** Coloca el producto en la siguiente ranura disponible del buffer.
5. **Liberar el cerrojo:** Ejecuta una operación signal() sobre mutex, permitiendo que otro proceso pueda acceder al buffer.
6. **Notificar que hay un nuevo producto:** Ejecuta una operación signal() sobre el semáforo full, incrementando su valor y notificando (o despertando, si está dormido) al consumidor de que hay un nuevo elemento disponible.
7. **Repetir:** Vuelve al paso 1.

Algoritmo del Consumidor

De manera simétrica, el proceso del consumidor seguirá este ciclo:

1. **Esperar por un producto:** Ejecuta una operación wait() sobre el semáforo full. Si full es mayor que cero, lo decrementa y continúa. Si es cero (buffer vacío), el consumidor se bloquea hasta que el productor añada un nuevo elemento.
2. **Adquirir el cerrojo:** Ejecuta wait() sobre mutex para obtener acceso exclusivo.
3. **Retirar el elemento:** Extrae un producto de la siguiente ranura ocupada del buffer.
4. **Liberar el cerrojo:** Ejecuta signal() sobre mutex.
5. **Notificar que hay un nuevo espacio vacío:** Ejecuta signal() sobre el semáforo empty, incrementando su valor y notificando al productor que ahora hay un espacio disponible.
6. **Consumir el elemento:** Procesa el dato que fue retirado.
7. **Repetir:** Vuelve al paso 1.

Esta arquitectura de tres semáforos resuelve de manera elegante y segura todos los desafíos del problema, asegurando una cooperación eficiente entre el productor y el consumidor sin pérdida de datos y sin riesgo de bloqueo mutuo.

IV. Links de la actividad

Código del programa

https://github.com/IngJuanRamirez/Producto-Consumidor

Video de prueba

https://drive.google.com/file/d/1-bU7eBPK68LviAy24Q5xDCGubFbkkRD5/view?usp=sharing

V. Conclusión

En esta actividad aprendí que el problema del productor consumidor es otra manera de explicar como los procesos piden recursos al procesador, y este tiene que dárselos, bloquearlos, pausarlos para que la administración de los recursos fuera más eficiente.

Para esto tenemos los semáforos, que actúan como banderas o condicionales para que dos o mas procesos canibalicen recursos del sistema. Esto hace mas sencillo el manejo de uno o varios procesos, pero donde brilla más, es en la programación concurrente, donde hay varios hilos y procesos consumiendo recursos.

VI. Referencias

- Gaurav, S. (2022, November 15). *What is producer consumer problem in C?* Scaler Topics. https://www.scaler.com/topics/producer-consumer-problem-in-c/