UNIVERSIDAD DEL VALLE DE GUATEMALA

Programación de microprocesadores – SECCÍON 10

Catedrática: Kimberly Marisol Barrera Alvarez

Proyecto 2 Enlace de vídeo

231221– Sánchez Sánchez, José Roberto

23661 – Mérida Jiménez, Ángel de Jesús

23574 – López Pivaral, André Emilio

Nueva Guatemala de la Asunción, 25 de febrero de 2026

Preguntas:

1. **identifique claramente la sección crítica en su código.**

R// La sección crítica (SC) es el bloque donde se accede y modifica el buffer compartido y sus índices. Está delimitada por el pthread_mutex_lock(&mutex) y pthread_mutex_unlock(&mutex) en ambos hilos: En el cajero (productor y en En el empacador (consumidor):

2. **¿Cómo se implementa la exclusión mutua?**

R// Se implementa con un mutex binario (pthread_mutex_t mutex):

- Inicialización: pthread_mutex_init(&mutex, NULL);
- Adquisición: pthread_mutex_lock(&mutex);
- Liberación: pthread_mutex_unlock(&mutex);

3. **¿Qué variable compartida requiere protección?**

R// Las variables compartidas que requieren protección son:

- area_empaque[BUFFER_SIZE] el buffer circular
- indice_in
- indice_out
- total_producidos
- total_consumidos

4. **¿Dónde podría ocurrir una condición de carrera?**

R// Sin el mutex, ocurriría condición de carrera en:

- Dos cajeros escribiendo al mismo tiempo en area_empaque[indice_in] y modificando indice_in → se perderían productos o se sobrescribirían.
- Un cajero y un empacador modificando indice_in e indice_out simultáneamente → el buffer se corrompería (índices erróneos).
- Actualización simultánea de total_producidos y total_consumidos → conteos incorrectos.

Además, sin los semáforos sem_empty y sem_full, podría haber buffer overflow (más de 5 productos) o underflow (leer producto inexistente).

5. **¿Existe posibilidad de deadlock? Explique.**

R// No existe posibilidad de deadlock en esta implementación.

Razones:

- El mutex se adquiere siempre después de hacer sem_wait (nunca al revés).
- El mutex se libera antes de hacer sem_signal del otro semáforo.
- Nunca se mantiene el mutex mientras se espera en un semáforo.
- El orden de adquisición es siempre el mismo en todos los hilos.

6. ¿Existe posibilidad de starvation? Justifique.

Teóricamente sí es posible, pero muy poco probable.Los semáforos POSIX (y tu implementación manual) no garantizan equidad (fairness). Un hilo podría despertar siempre antes que otro. Con 3 cajeros y 2 empacadores, los empacadores podrían "robarse" los productos constantemente, dejando a algún cajero esperando mucho tiempo en sem_empty. Sin embargo, en la práctica con Linux y pocos hilos (5 en total), el scheduler es suficientemente justo y el buffer pequeño (5) hace que el starvation sea prácticamente imposible de observar.

7. ¿Qué ocurriría si se duplican los hilos/procesos?

Si se duplican los hilos (ej. 6 cajeros y 4 empacadores):

- El sistema seguiría funcionando correctamente (la solución es escalable).
- Aumentaría mucho el throughput (más productos por segundo).
- Habría más contención en el mutex → más tiempo de espera en pthread_mutex_lock.
- El buffer de tamaño 5 se llenaría y vaciaría más rápido, aumentando el número de bloqueos en sem_empty y sem_full.
- La simulación terminaría igual en 60 segundos, pero con muchos más productos procesados.

8. ¿Cómo se comporta el sistema bajo alta carga?

Bajo alta carga (muchos cajeros o buffer pequeño)

- El mutex se convierte en bottleneck y todos los hilos compiten por él.
- Los semáforos gestionan muy bien el flujo (no se produce overflow ni underflow).
- El sistema se ralentiza por contención, pero nunca falla (no hay corrupción ni crash).

9. Diferencie concurrencia y paralelismo en su implementación.

- Concurrencia: Siempre presente. Los 5 hilos (3 cajeros + 2 empacadores + timer) están vivos al mismo tiempo y se interleavan. Aunque la máquina tenga un solo núcleo, el sistema es concurrente gracias a los hilos y la sincronización.
- Paralelismo: Solo ocurre si tu máquina tiene múltiples núcleos (la mayoría actual). En ese caso, varios cajeros y empacadores pueden estar ejecutando realmente al mismo tiempo (en diferentes cores).

10. ¿Qué métricas demuestran que su solución es correcta?

Las métricas que demuestran corrección son:

1. total_producidos - total_consumidos está entre 0 y BUFFER_SIZE (5) al final.
2. Nunca se observa buffer con menos de 0 ni más de 5 productos (gracias a los semáforos).
3. Los contadores total_producidos y total_consumidos son coherentes.
4. Todos los hilos terminan limpiamente después de 60 segundos (no quedan hilos zombies).
5. No hay corrupción en los nombres de productos ni en los índices.
6. En los logs se ve que el buffer siempre está entre 0/5 y 5/5.
7. El programa nunca hace segmentation fault ni deadlock.