TALLER 03 SINCRONIZACIÓN POSIX .

SISTEMAS OPERATIVOS .

INTEGRANTES:

MAURICIO BELTRÁN HUERTAS

ALEJANDRO BELTRAN HUERTAS

ANDRES DIAZ

2025

Introducción:

En el presente informe se describe el desarrollo y análisis del Taller 03: Sincronización POSIX, en el cual se busca aplicar los conceptos de concurrencia, exclusión mutua y sincronización entre procesos e hilos por medio del uso de las interfaces POSIX en lenguaje C.

Este taller se divide en dos actividades:

1. Implementación del problema Productor–Consumidor con semáforos POSIX y memoria compartida.

2. Implementación del problema Productor–Consumidor con hilos POSIX (pthreads), mutex y variables de condición.

Con estas actividades se busca evidenciar la mejor coordinación entre tareas concurrentes, evitando condiciones de carrera y garantizando el acceso ordenado a los recursos compartidos.

Actividad 1: Sincronización con Semáforos POSIX

Descripción

En la primera actividad se implementó un sistema Productor–Consumidor en el que dos programas (producer.c y consumer.c) se comunican mediante memoria compartida y se sincronizan utilizando semáforos POSIX con nombre.

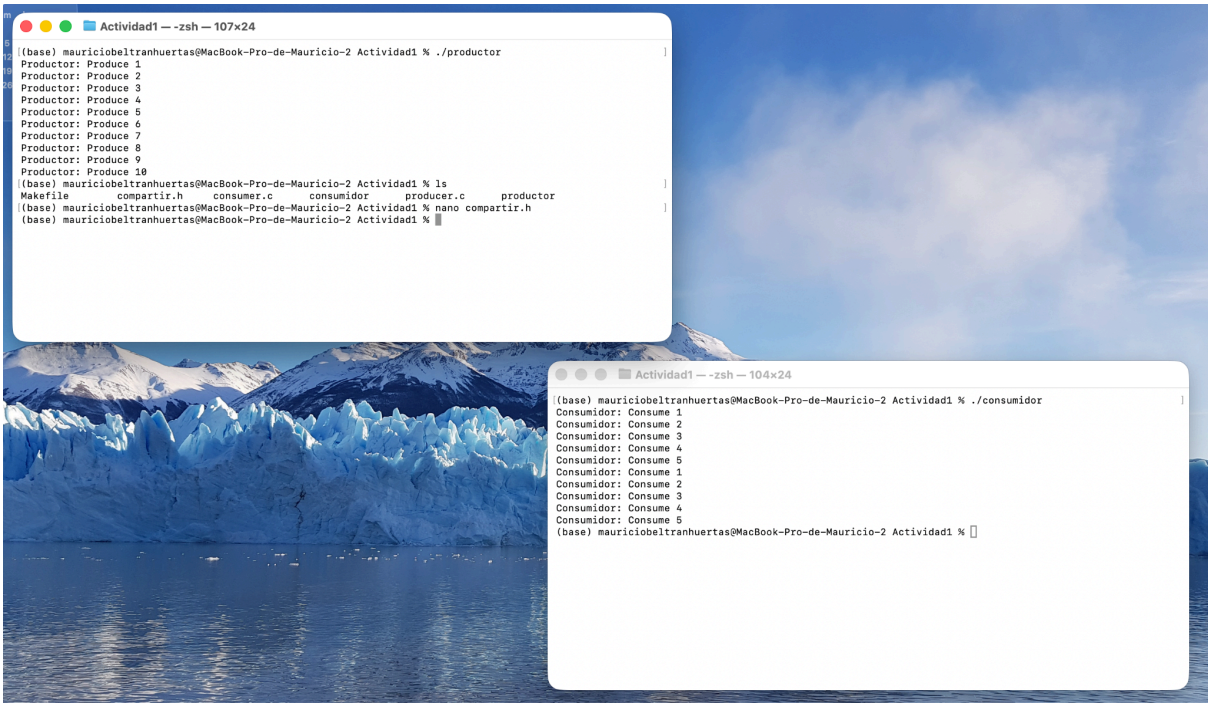
El productor genera elementos en un buffer compartido, mientras que el consumidor los retira secuencialmente.

Los semáforos /vacio y /lleno controlan la cantidad de espacios disponibles y ocupados, evitando tanto la sobreproducción como el consumo sin datos.

Ejecución

El productor se ejecuta primero, seguido del consumidor.

Al correr ambos procesos en terminales diferentes, se observa cómo el consumidor procesa cada elemento justo después de su producción, sin bloqueos ni pérdidas de información.



Importancia

Todo lo abordado en esta actividad es de suma importancia. Sin embargo. El punto más importante es la creación y manejo correcto de semáforos con nombre, junto con la gestión ordenada del buffer en memoria compartida, demostrando control concurrente entre dos procesos distintos.

Actividad 2: Sincronización con Hilos POSIX (Pthreads)

Descripción

La segunda actividad amplía el concepto de sincronización, pero ahora dentro de un solo proceso multihilo.

El programa posixSincro.c crea 10 hilos productores y un hilo consumidor.
Cada productor genera líneas de texto que se almacenan en un buffer circular, y el spooler las imprime en pantalla de manera sincronizada.

Ejecución

```
(base) mauriciobeltranhuertas@MacBook-Pro-de-Mauricio-2 ~ % cd TallerPosix
(base) mauriciobeltranhuertas@MacBook-Pro-de-Mauricio-2 TallerPosix % cd Actividad2
(base) mauriciobeltranhuertas@MacBook-Pro-de-Mauricio-2 Actividad2 % ./posixSincro
Thread 4: 1
Thread 6: 1
Thread 5: 1
Thread 2: 1
Thread 8: 1
Thread 9: 1
Thread 0: 1
Thread 7: 1
Thread 3: 1
Thread 1: 1
Thread 6: 2
Thread 0: 2
Thread 1: 2
Thread 5: 2
Thread 2: 2
Thread 4: 2
Thread 9: 2
Thread 7: 2
Thread 8: 2
Thread 3: 2
Thread 7: 3
Thread 6: 3
Thread 0: 3
Thread 3: 3
Thread 2: 3
Thread 4: 3
Thread 8: 3
Thread 1: 3
Thread 9: 3
Thread 5: 3
Thread 8: 4
Thread 7: 4
Thread 3: 4
Thread 6: 4
Thread 0: 4
Thread 9: 4
Thread 2: 4
Thread 4: 4
Thread 5: 4
Thread 1: 4
Thread 8: 5
Thread 2: 5
Thread 0: 5
Thread 7: 5
Thread 6: 5
Thread 3: 5
Thread 9: 5
Thread 1: 5
Thread 4: 5
Thread 5: 5
Thread 8: 6
Thread 4: 6
Thread 9: 6
Thread 7: 6
Thread 5: 6
Thread 6: 6
Thread 1: 6
Thread 2: 6
Thread 0: 6
Thread 3: 6
Thread 4: 7
Thread 8: 7
Thread 7: 7
```

Los hilos trabajan concurrentemente, pero las líneas se imprimen completas y ordenadas, sin interferencias parciales.

El correcto uso del mutex y las condiciones asegura que nunca dos hilos accedan al mismo espacio del buffer simultáneamente.

Importancia

En esta actividad todo fue también de suma importancia. Sin embargo, la sincronización lograda con pthread_mutex y pthread_cond_t es el elemento más relevante de esta actividad, ya que representa la aplicación directa de los principios de concurrencia vistos en clase.
El código demuestra un control eficiente del acceso al recurso compartido, respetando el flujo productor–consumidor con total exclusión mutua.

Conclusiones

- El taller permitió comprender y aplicar de manera práctica los conceptos de sincronización y exclusión mutua en entornos concurrentes.

- En la Actividad 1, se consolidó el uso de semáforos POSIX y memoria compartida para coordinar procesos independientes.

- En la Actividad 2, se comprobó cómo los mismos principios pueden aplicarse dentro de un solo proceso mediante hilos POSIX (pthreads), mutex y variables de condición.

- Las implementaciones desarrolladas garantizan la integridad de los datos y la coherencia en la ejecución, evidenciando la importancia de los mecanismos de sincronización en sistemas operativos multitarea.

- Finalmente, este trabajo fortaleció las habilidades de programación concurrente en C, aplicando estructuras POSIX de bajo nivel, fundamentales para el desarrollo de software eficiente y seguro en entornos reales.