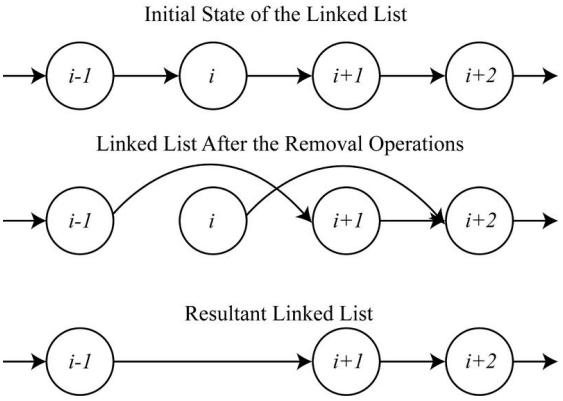
Sincronización de procesos

La ejecución de múltiples procesos trae consigo la aparición de colisiones en cuanto al uso de los recursos, sea por acceder a estos al mismo tiempo o porque ya se encuentran en uso cuando se solicita el acceso a los mismos, dichos problemas se presentan en tipos de procesos como:

- **Procesos no relacionados:** aquellos que no tienen una tarea en común pero compiten por un recurso
- **Procesos relacionados indirectamente:** aquellos que a pesar de no ser emparentados comparten un espacio en memoria para la realización de sus tareas, también denominado *cooperación mediante compartición.*
- **Procesos relacionados directamente:** aquellos que se comunican directamente para la relación de una tarea específica, es decir, presenta *cooperación*.

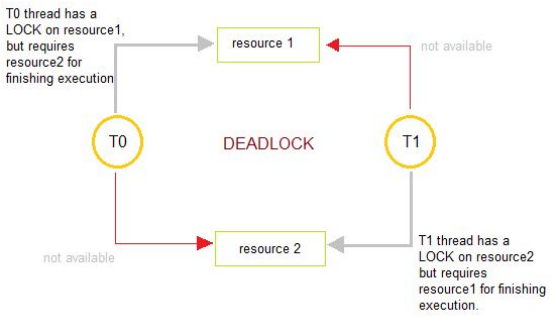
A pesar de ser naturaleza distinta los procesos mencionados anteriormente, se tiene que los problemas presentes en ellos corresponden a:

Exclusión Mutua:
El problema en la exclusión mutua corresponde aquel en el que dos procesos intentan a acceder a un mismo recurso al tiempo, vulnerando la integridad de los datos. Un ejemplo de ello se puede evidenciar en la modificación de una lista enlazada, para la cual se solicita remover el elemento i y el i+1 al mismo tiempo. Debido a la naturaleza de funcionamiento de las listas enlazadas, al remover i el elemento i-1 apuntará a i+1 y del mismo modo al remover i+1, i apuntará a i+2 dejando como resultado una lista en la que i+1 no fue verdaderamente removido como se evidencia en el siguiente dibujo:



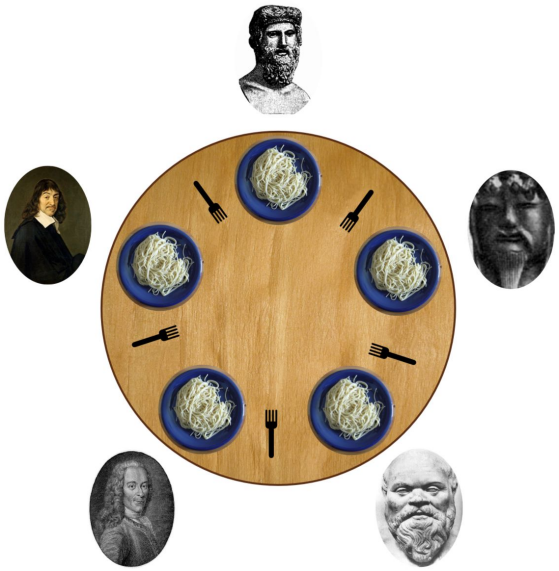
Interbloqueos:

A diferencia del problema en exclusión mutua para el cual ambos procesos acceden al tiempo a un mismo recurso, en interbloqueo cuando se tiene acceso por parte de un proceso este lo bloquea inhibiendo a los demás procesos acceder al mismo recurso. Lo anterior conlleva a que cuando dos procesos, P1 y P2 requieren dos recursos R1 y R2 para la realización de sus tareas, pero el sistema asignó un recurso a cada uno, por ejemplo, R1 a P1 y R2 a P2, ambos procesos quedan a la espera de que el otro desocupe el recurso faltante.



Inanición:

Corresponde a un caso derivado de interbloqueos, en el cual existen tres o más procesos para los cuales uno de ellos necesita un recurso que es ocupado por los otros dos, quedándose siempre a la espera o inactivo. Un ejemplo se puede evidenciar generalmente cuando se presentan ciertas reglas de sincronización que inhabilitan la obtención de los recursos, en particular para el problema de la cena de filósofos, si a los diez minutos de estar esperando un tenedor se tuviera que esperar otros diez minutos para volver a intentarlo, luego se presenta la inanición.



Dado lo anterior, se hace pertinente el desarrollo de mecanismos de sincronización para otorgar orden al acceso de los recursos, uno de esto es el método de los semáforos el cual se aborda más adelante.

¿Cómo el hardware soporta el Mutex (Exclusión Mutua)?

Entrando un poco más en el funcionamiento de un Mutex, este corresponde a un número en memoria para el cual su valor nos indica si se encuentra bloqueado o no, esto es, el valor inicial del mutex es 0, lo que significa que está desbloqueado, si se desea bloquear el mutex, es necesario verificar si es cero y luego asignar uno.

Su funcionamiento a nivel de hardware se basa en las operaciones de “test-and-set”, instrucción que escribe en una posición de memoria y retorna su antiguo valor de manera atómica. Para su correcto funcionamiento se hace uso de la DPRAM o Dual Ported RAM, la cual al al recibir una instrucción “test-and-set” del CPU 1 este guarda el valor actual de la posición de memoria en un registro especial y setea con un flag a la posición requerida por CPU 1, en caso de recibir otro test-and-set, el DPRAM envía una interrupción para evitar la colisión y continua con el procesamiento de la petición de CPU 1, al finalizar el test en caso de dar bien escribe el valor dado por CPU 1 y en caso de no vuelve a escribir el valor antiguo trayéndolo del registro especial, dando paso a la siguiente petición “test-and-set”.

Semáforos

Los semáforos tienen una implementación similar al de los Mutex, estos son variables en las que se almacena un entero sobre el que se puede inicializar, incrementar o decrementar su valor, permitiendo la sincronización de procesos mediante la revisión del valor encontrado en la variable, esto es: si el valor encontrado es negativo luego los recursos están siendo utilizados por otro proceso bloqueando así la acciones sobre estos, de lo contrario el proceso que solicita acceso puede continuar normalmente.

La diferencia entre el mutex y los semáforos radica estrictamente hablando en que los semáforos son un método de señalización mientras que los mutex un método de bloqueo, por lo tanto su aplicación es diferente, esto se evidencia en el problema del productor-consumidor.

Usando Mutex, siempre y cuando el búfer este siendo llenado por el productor, el consumidor debe esperar y viceversa, esto es, solo un hilo puede trabajar con todo el búfer.

Usando Semáforos, el buffer puede ser dividido en cuatro partes y a esto se les puede asociar un semáforo, por lo que el consumidor y el productor pueden trabajar en diferentes buffers al mismo tiempo.

Por último, se presenta una breve explicación del código observado en el enunciado del taller. En este se tienen dos hilos, los cuales imprimirán *p+1* veces el valor asignado a *data.* para evitar colisiones en la impresión, esto es, que un hilo imprima cuando otro aún no haya terminado, se hace uso de un semáforo que garantiza el orden en la impresión, así los valores impresos por un hilo no serán interrumpidos hasta cuando este haya culminado su proceso.

Bibliografía

https://github.com/capedrazab/os-20191/blob/master/taller-concurrencia.pdf

https://www.geeksforgeeks.org/mutex-vs-semaphore/

https://mortoray.com/2019/02/20/how-does-a-mutex-work-what-does-it-cost/

https://en.wikipedia.org/wiki/Test-and-set#Hardware_implementation_of_test-and-set

https://en.wikipedia.org/wiki/Semaphore_(programming)

https://en.wikipedia.org/wiki/Mutual_exclusion

https://en.wikipedia.org/wiki/Deadlock