**TAREA 3**

**Lectura 4:**

3.4 Comunicación interprocesos:

* Los procesos pueden ser independientes o cooperativos.
* Razones para permitir la cooperación entre procesos:
  + compartir información
  + acelerar cálculos
  + modularidad
  + conveniencia.
* Mecanismos de comunicación interprocesos: memoria compartida y paso de mensajes.

3.4.1 Sistemas de memoria compartida:

* Requiere establecer una región de memoria compartida.
* Los procesos deben acordar eliminar la restricción de acceso a la memoria de otros procesos.
* Ejemplo del problema del productor-consumidor.
* Implementación de búferes limitados y no limitados.
* Variables de control como in y out para un búfer circular.

3.4.2 Sistemas de paso de mensajes:

* Alternativa a la memoria compartida donde los procesos no comparten el mismo espacio de direcciones.
* Se basa en enviar y recibir mensajes.
* Puede ser útil en entornos distribuidos.
* Operaciones básicas: envío y recepción de mensajes.

3.4.2.1 Nombrado:

* Comunicación directa o indirecta entre procesos.
* Esquemas de comunicación directa simétrica o asimétrica.
* Comunicación indirecta a través de buzones de correo.

3.4.2.2 Sincronización:

* Envío y recepción con bloqueo o sin bloqueo.
* Posibles combinaciones de operaciones send() y receive().
* Conceptos de síncrono y asíncrono en la comunicación interprocesos.
* Solución al problema del productor-consumidor utilizando send() y receive() con bloqueo.

3.4.2.3 Almacenamiento en búfer:

* Implementación de colas temporales para los mensajes.
* Tres tipos de colas: capacidad cero, limitada e ilimitada.
* Casos de sistemas con y sin almacenamiento en búfer.

**3.5 Ejemplos de sistemas IPC**

3.5.1 Memoria compartida en POSIX

* La API de POSIX ofrece mecanismos IPC como memoria compartida y paso de mensajes.
* Para utilizar memoria compartida en POSIX, se crea un segmento con shmget(), especificando tamaño y modo de acceso.
* El proceso asociado a un segmento de memoria compartida lo conecta a su espacio de direcciones con shmat().
* Una vez conectado, el proceso accede a la memoria compartida normalmente como a cualquier otra región de memoria.
* La desconexión de la memoria compartida se hace con shmdt(), y su eliminación con shmctl().

3.5.2 Mach

* Mach, un sistema operativo desarrollado en la Universidad Carnegie Mellon, utiliza mensajes para la comunicación.
* Las comunicaciones se realizan mediante buzones de correo (puertos).
* Mach utiliza llamadas al sistema como msg\_send() y msg\_receive() para enviar y recibir mensajes.
* Mach también soporta llamadas a procedimientos remotos (RPC) mediante msg\_rpc().
* Los mensajes en Mach constan de una cabecera fija seguida de datos variables.
* Mach emplea técnicas para evitar la doble copia de mensajes y garantiza la colocación FIFO en la cola de mensajes.

3.5.3 Windows XP

* Windows XP emplea modularidad para incrementar funcionalidad y reducir tiempos de implementación.
* Utiliza la facilidad de paso de mensajes llamada LPC (local procedure call) para la comunicación entre procesos.
* LPC establece conexiones mediante objetos puerto, que pueden ser de conexión o de comunicación.
* Para mensajes pequeños, se utiliza una cola de mensajes del puerto; para mensajes grandes, se utiliza una región de memoria compartida.
* Windows XP permite el uso de mecanismos de retrollamada para tratamiento asincrónico de mensajes.
* La facilidad LPC no es parte de la API de Win32 y no es visible directamente para el programador de aplicaciones.

**Lectura 5:**

**Transferencia de mensajes**

* Utiliza primitivas como **SEND** y **RECEIVE**, que son llamadas al sistema.
* **SEND** envía un mensaje a un destino dado, mientras que **RECEIVE** recibe un mensaje de un origen dado.
* El receptor puede bloquearse hasta que llegue un mensaje o regresar inmediatamente con un código de error si no hay mensajes disponibles.

**Aspectos de diseño de los sistemas de transferencia de mensajes**

* En sistemas distribuidos, pueden perderse mensajes en la red.
* Para protegerse contra la pérdida de mensajes, se pueden implementar mecanismos de acuse de recibo o confirmación.
* Se utilizan números de secuencia para distinguir mensajes nuevos de retransmisiones.
* Es importante resolver la cuestión del nombre de los procesos para evitar ambigüedades.
* La verificación de autenticidad es crucial para asegurar que la comunicación se realice con los procesos correctos.

**El problema de productor-consumidor con transferencia de mensajes**

* Se puede resolver utilizando transferencia de mensajes sin compartir memoria.
* Se emplean N mensajes, análogos a las N ranuras de un buffer en memoria compartida.
* El productor envía mensajes vacíos al consumidor, y cada vez que tiene un elemento para entregar, intercambia un mensaje vacío por uno lleno.
* El número total de mensajes en el sistema permanece constante y se pueden almacenar en una cantidad de memoria conocida.

**Variantes de la transferencia de mensajes**

* Los mensajes pueden dirigirse a procesos específicos o a buzones.
* Los buzones pueden ser lugares temporales de almacenamiento para mensajes.
* Algunos sistemas de mensajes utilizan un enfoque de "cita" o rendezvous, donde el emisor y el receptor operan estrictamente sincronizados.
* En UNIX, la comunicación entre procesos de usuario se realiza a través de conductos, que efectivamente son buzones.

**Video:**

**Unidad 2: Comunicación entre Procesos (1ra parte)**

Concurrencia

* Multiprogramación: Gestión de varios procesos en un sistema monoprocesador.
* Multiproceso: Gestión de varios procesos en un sistema multiprocesador.
* Proceso distribuido: Gestión de procesos en sistemas de computadoras múltiples y remotas.
* La concurrencia es crucial para el diseño de sistemas operativos.
* Incluye temas como la comunicación entre procesos, la sincronización y la competencia por recursos.

Comunicación entre procesos

* Los procesos necesitan comunicarse para compartir información.
* Ejemplos incluyen tuberías donde la salida de un proceso se convierte en la entrada del siguiente.
* Es esencial una comunicación estructurada para evitar interrupciones.

Condiciones de competencia (Race Conditions)

* En sistemas operativos, los procesos pueden competir por recursos, creando condiciones de carrera.
* Se deben evitar para garantizar la consistencia y la integridad de los datos.

Secciones y Regiones Críticas

* Se establecen condiciones para garantizar la exclusión mutua y evitar condiciones de carrera.
* Ningún proceso debe bloquear indefinidamente a otros dentro de su región crítica.

Semáforos

* Los semáforos son una solución para problemas de concurrencia como el Productor-Consumidor.
* Permiten a los procesos sincronizarse y coordinar su actividad.

Monitores

* Los monitores proporcionan una abstracción más estructurada para la exclusión mutua.
* Ejecutan solo un procedimiento monitor a la vez y controlan el acceso a recursos compartidos.

The producer-consumer problem with N messages

* Se presenta el problema del Productor-Consumidor con N mensajes.
* Se busca una solución para coordinar la producción y el consumo de mensajes entre procesos.

**TAREA 4:**

**Lectura 6:  
6.1 Fundamentos**

**Modelo de Sistema:**

* Desarrollo de un modelo de sistema con procesos o hebras secuenciales cooperativas.
* Ejecución asíncrona y posible compartición de datos entre procesos.
* Ejemplo del problema del productor-consumidor para ilustrar el modelo.

**Uso de Búfer Limitado:**

* Descripción de cómo utilizar un búfer limitado para permitir a los procesos compartir memoria.
* Propuesta de modificación del algoritmo para remediar deficiencias.
* Demostración de problemas potenciales cuando se ejecutan procesos de manera concurrente.
* Introducción al concepto de condición de carrera y la necesidad de sincronización.

**6.2 El Problema de la Sección Crítica**

**Definición del Problema:**

* Sección crítica como un segmento de código en el que un proceso puede modificar variables comunes.
* Importancia de garantizar que ningún otro proceso ejecute su sección crítica simultáneamente.

**Requisitos del Problema:**

1. Exclusión mutua: Un proceso ejecutando su sección crítica impide que otros lo hagan.
2. Progreso: Si ningún proceso está en su sección crítica, uno debe poder entrar.
3. Espera limitada: Un proceso que solicita entrar eventualmente lo hace.

**Implementación y Métodos:**

* Descripción de soluciones al problema de la sección crítica: software (Solución de Peterson) y hardware (instrucciones específicas).

**6.3 Solución de Peterson**

**Descripción del Algoritmo:**

* Solución basada en software que proporciona una descripción algorítmica de la resolución del problema.
* Detalles sobre la exclusión mutua, progreso y espera limitada proporcionados por la solución.

**6.4 Hardware de Sincronización**

**Uso de Cerrojos:**

* Importancia de los cerrojos para proteger regiones críticas.
* Descripción de soluciones adicionales utilizando soporte hardware para garantizar exclusión mutua.

**6.5 Semáforos**

**Definición y operaciones básicas**

* Un semáforo es una variable entera que solo se accede mediante dos operaciones atómicas: **wait()** y **signal()**.
* **wait()** reduce el valor del semáforo, mientras que **signal()** lo incrementa.
* Estas operaciones deben ejecutarse de forma indivisible para evitar condiciones de carrera.

**Utilización**

* Hay semáforos contadores y binarios.
* Los binarios son utilizados para la exclusión mutua.
* Los contadores controlan el acceso a recursos finitos.
* Se pueden usar para sincronizar procesos concurrentes.

**Implementación**

* La definición clásica de semáforo tiene espera activa, lo que puede desperdiciar ciclos de CPU.
* Una alternativa es la implementación con bloqueo y desbloqueo de procesos.
* Los semáforos se implementan como estructuras con un valor entero y una lista de procesos en espera.
* Operaciones como **wait()** y **signal()** deben ejecutarse atómicamente.

**6.5.3 Interbloqueos e inanición**

**Interbloqueos**

* Pueden ocurrir cuando varios procesos están esperando a que otro realice una operación.
* Esto puede llevar a una situación de espera indefinida.

**Bloqueo indefinido o muerte por inanición**

* Ocurre cuando algunos procesos esperan indefinidamente en un semáforo, especialmente si se usa una política LIFO para la gestión de la lista de procesos.