PROYECTO
SISTEMAS OPERATIVOS

GIULIA GIACOMODONATO – SANTIAGO CASTRO
COMISIÓN 43

06/11/2024

Contenido

1. Experimentación de Procesos y Threads con los Sistemas Operativos.

1.1 Procesos, threads y Comunicación

1. PUMPER NIC.

a)

Implementación del sistema de PUMPER NIC utilizando pipes para la comunicación entre procesos.

Entidades:

- Los empleados y el despachador son procesos separados que se quedan en un bucle infinito, leyendo pedidos de su pipe correspondiente.
- Cada cliente es un proceso separado, realiza su pedido y cuando lo obtiene se retira.

Los pipes permiten que los procesos envíen y reciban datos. Utilizamos los pipes para facilitar la comunicación entre los clientes, el despachador y los empleados de cocina. Los pipes funcionan en un modelo de FIFO (First In, First Out), lo que significa que los datos se leen en el mismo orden en que se escribieron, de esta forma se mantiene la secuencia de los pedidos.

- Por simplificación, cada cliente puede pedir sólo un tipo de combo. Los pedidos son modelados mediante una estructura Pedido, que contiene el tipo, si el cliente es vip o no y el número del cliente que lo pidió.
- Para manejar la prioridad de los clientes vip, el despachador siempre lee primero de manera no bloqueante del pipe de los pedidos vip, si hay pedidos, atiende todos primero y luego atiende a los pedidos no vip de a uno.
  Como limitación, al despachador esperar de manera bloqueante a los clientes vip, una vez que atendió a todos los vips y está esperando a un cliente regular, si el cliente regular nunca llega se quedará bloqueado para siempre y los clientes vip sufrirán inanición. Esta forma de manejar la prioridad la utilizamos por simplicidad asumiendo que los clientes regulares siempre van a llegar.
- Cada cliente tiene una política de decisión sobre si esperar en la cola o irse, la decisión es aleatoria, 1 de cada 7 clientes se van y no vuelven.
- Una vez que el despachador atendió a los clientes, envía los pedidos a través del pipe correspondiente, según el tipo de pedido al empleado que debe cocinar.
- Los empleados leen los pedidos y los preparan, al finalizar, les envían a los clientes el pedido terminado a través del pipe de pedidos listos según el tipo de pedido.
- Los clientes esperan a que el despachador los atienda y luego a que su combo esté listo, toman el primero que haya en el pipe de pedidos listos y se retiran.
  Como limitación, puede ocurrir que a veces los clientes se lleven los pedidos de los demás si estaban esperando por el mismo tipo de pedido debido a la simplificación del problema y las consideraciones mencionadas.

Debe tenerse en cuenta que como los procesos ciclan para siempre buscando pedidos a pesar de que no haya más clientes, nunca se va a terminar la ejecución.

Para compilar y ejecutar ir a la carpeta PumperNicPipes, escribir en la consola el comando ‘make’ y luego ejecutar el comando ‘make run’.

b)

Implementación del sistema de PUMPER NIC utilizando colas de mensajes para la comunicación entre procesos.

Entidades:

- Los empleados y el despachador son procesos separados que se quedan en un bucle infinito, leyendo pedidos de la cola de mensajes.
- Cada cliente es un proceso separado, realiza su pedido y cuando lo obtiene se retira.

Manejamos cada proceso en un archivo separado y los cargamos todos mediante su respectiva imagen ejecutable usando 'execv' en el proceso padre.

Utilizamos una cola de mensajes para manejar la comunicación entre diferentes tipos de procesos (clientes, despachador, empleados). Utilizamos los diferentes tipos de mensajes como señales para que cada entidad se pueda comunicar con las demás.

La cola de mensajes es creada por el proceso padre y todos comparten la estructura que la contiene.

- Por simplificación, cada cliente puede pedir sólo un tipo de combo. Los pedidos son modelados mediante una estructura mensaje_pedido, que contiene el tipo del mensaje, el tipo de combo, si el cliente es vip o no y el número del cliente que lo pidió.
- Para manejar la prioridad de los clientes vip, utilizamos la prioridad ya implementada de las colas de mensajes buscando clientes vip primero si hay, atiende a todos y luego atiende a los clientes regulares.
- Cada vez que llega un cliente toma un mensaje de tipo fila, cuando no hay más quiere decir que hay mucha gente, si en este momento llega un cliente nuevo el mismo decide aleatoriamente si quedarse o no a pesar de la multitud. Si el cliente decide irse no vuelve.
  Cuando el despachador atiende al cliente y envía su pedido al respectivo empleado, devuelve un lugar en la fila para que nuevos clientes puedan realizar sus pedidos.
- Una vez que el cliente realiza el pedido espera a que los empleados que lo preparan envíen el mensaje de que está listo. El cliente toma el primer pedido del combo que pidió y se retira.

Ventajas de esta implementación:

- Eficiencia: Al utilizar cola de mensajes podemos solucionar el problema con una única cola, en cambio con pipes necesitábamos uno por cada tipo de mensaje que queríamos enviar. De esta forma utilizamos menos memoria.
- Código más simple: Utilizar cola de mensajes permite simplificar y modularizar el código permitiendo extensibilidad.
- La prioridad es manejada por la cola de mensajes.

Debe tenerse en cuenta que como los procesos ciclan para siempre buscando pedidos, a pesar de que no haya más clientes no se va a terminar la ejecución por sí sola.

Para compilar y ejecutar ir a la carpeta PumperNicColasMSG, escribir en la consola el comando 'make' y luego ejecutar el comando 'make run'.

2. MINI SHELL

Comandos implementados:

a) Mostrar una ayuda con los comandos posibles. Implementamos el comando help para mostrar la ayuda de la Shell.
   - No toma argumentos.
   - Imprime una lista de los comandos disponibles en la shell.

b) Crear un directorio. Implementamos el comando mkdir para crear un directorio.
   - Toma un argumento: el nombre del directorio a crear.

c) Eliminar un directorio. Implementamos el comando rmdir para eliminar un directorio.
   - Toma un argumento: el nombre del directorio a eliminar.

d) Crear un archivo. Implementamos el comando touch para crear un archivo vacío.
   - Toma un argumento: el nombre del archivo a crear.

e) Listar el contenido de un directorio. Implementamos el comando ls para listar el contenido de un directorio.
   - Puede tomar un argumento opcional: el directorio a listar (si no se proporciona usa el directorio actual).

f) Mostrar el contenido de un archivo. Implementamos el comando cat para mostrar el contenido de un archivo.
   - Toma un argumento: el nombre del archivo a mostrar.
   - Lee el archivo carácter por carácter y lo imprime en la consola.

g) Modificar los permisos de un archivo. Los permisos son de lectura, escritura y ejecución. Implementamos el comando chmod para cambiar permisos de archivos.
   - Toma dos argumentos: el nombre del archivo y los permisos en octal.

h) Exit

Para la implementación de la Mini Shell, tuvimos en cuenta la extensibilidad y el funcionamiento del sistema de línea de comandos de Linux. En el diseño de esta Mini Shell, optamos por la creación de un proceso independiente para cada comando, lo que emula el comportamiento del shell de Linux. De este modo, cada comando se implementa en un archivo separado, a excepción del comando "exit", que gestiona la terminación del Shell.

La arquitectura de esta Mini Shell permite que, al momento de ser solicitados, los comandos sean cargados mediante la llamada al sistema execv(), la cual recibe como parámetros el nombre del archivo correspondiente y los argumentos necesarios para su ejecución. Esta decisión de diseño simplifica la estructura del código, y también proporciona una ventaja en términos de mantenimiento y extensión del sistema.

Para compilar y ejecutar ir a la carpeta Shell, escribir en la consola el comando ‘make’ y luego ejecutar el comando ‘make run’.

1.2 Sincronización

1. TALLER DE MOTOS

Pensamos las actividades del taller como una secuencia para luego poder implementarla mediante semáforos.

Entidades: Cada operario se implementa en un hilo independiente

Operarios:

- Operario de ruedas: Espera a que el semáforo sem_nuevaMoto esté disponible, indicando el inicio de una nueva moto. Este proceso se repite para ambas ruedas.
- Operario de cuadro: Espera a que sem_ruedas haya sido incrementado dos veces (una por cada rueda).
- Operario de motor: Espera a que el cuadro esté listo (sem_cuadro).
- Pintores: Ambos pintores esperan a que el motor esté listo (sem_motor). El pintor que obtiene el semáforo primero pinta la moto de su color.
- Operario de equipamiento: Espera a que la moto esté pintada (sem_pintor). Después de que una moto ha sido pintada, incrementa sem_nuevaMoto dos veces para reiniciar el ciclo de construcción de la siguiente moto.
  Para mantener la simplicidad, legibilidad del código y la mínima utilización de semáforos, forzamos un poco de secuencialidad, el operario de equipamiento extra espera dos veces en sem_pintor y cada dos motos agrega el equipamiento extra.

Debe tenerse en cuenta que los hilos ciclan para siempre armando motos, por lo cual nunca se va a terminar la ejecución por sí sola.

Para compilar y ejecutar ir a la carpeta TallerMotos, escribir en la consola el comando ‘make’ y luego ejecutar el comando ‘make run’.

MODIFICACIONES REENTREGA

La secuencia es RRCM(PR o PV)RRCM(PR o PV)ERRCM(PR o PV)RRCM(PR o PV)E.....

- R: Operario Ruedas.
- C: Operario Cuadro.
- M: Operario Motor.
- PR: Operario Pintor Rojo.
- PV: Operario Pintor Verde.
- E: Operario Equipamiento Extra.

Agregando un semáforo extra ‘sem_extra’ podemos evitar la secuencialidad forzada. El operario de ruedas al comienzo de un ciclo de dos motos, espera por el ‘sem_extra’ que comienza en 1. Los pintores siempre permiten al operario de ruedas comenzar a fabricar sus dos ruedas. Una moto esta lista luego de que es pintada, y en una de cada dos motos listas el operario de equipamiento extra agrega modificaciones.

Una vez que se fabricaron 2 motos, el operario de equipamiento extra envía ‘sem_extra’ para permitir al operario de ruedas comenzar un nuevo ciclo de armado.

2. SANTA CLAUS

Encontramos similitudes con los problemas clásicos Barbero Dormilón y Baño Unisex. Tanto Santa Claus como el barbero deben activarse bajo ciertas condiciones; lo mismo sucede en el baño unisex, donde los usuarios de un género solo pueden entrar cuando el baño está desocupado u ocupado por personas de su mismo género. Por otro lado, el número de recursos (sillas, espacio en el baño, atención de Santa) es limitado y requiere mecanismos de sincronización.

Santa solo es despertado cuando hay trabajo acumulado, ya sea porque se reunió un grupo completo de renos o un grupo de elfos necesita ayuda.

Entidades:

- Un hilo para Santa Claus.
- Múltiples hilos para renos y elfos.

Uso de semáforos:

- Los semáforos binarios santaSem y santaDuerme se utilizan para despertar a Santa cuando es necesario y para indicar cuándo Santa está durmiendo.
- Los semáforos de conteo renosAtendidos y elfosAtendidos se utilizan para llevar un registro de cuántos renos o elfos están listos para ser atendidos. Santa solo puede atender de a 3 elfos y 9 renos.
- Los semáforos binarios renos_problemas y elfos_problemas se utilizan para indicar a Santa qué grupo necesita atención.
- Los semáforos binarios listoReno y listoElfo se utilizan para coordinar las acciones específicas de Santa con los renos y los elfos, indicándoles cuándo han sido atendidos.

Cuando Santa se despierta por uno de los grupos, prioriza a los renos. Primero verifica si los renos requieren ayuda (esto ocurre cuando 9 renos se encuentran listos), y si no hay renos, entonces verifica si hay elfos con problemas (cuando hay 3 elfos esperando ayuda).

Uso de mutexs: Utilizamos dos mutexs (emutex y rmutex) para proteger las secciones críticas que manipulan los semáforos tanto en los renos como en los elfos.

Debe tenerse en cuenta que los hilos ciclan para siempre, por lo cual nunca se va a terminar la ejecución por sí sola.

Para compilar y ejecutar ir a la carpeta SantaClaus, escribir en la consola el comando ‘make’ y luego ejecutar el comando ‘make run’.

MODIFICACIONES REENTREGA

Ahora para que Santa no se despierte y se vuelva a dormir si los renos y los elfos llegan al mismo tiempo, en lugar de usar un semáforo extra, cuando atendemos a los renos, preguntamos si los elfos están en problemas y consumimos el token que habrá de más en 'santaSem'. Por otro lado, primero avisamos a los renos/elfos que los atendí y luego permitimos el ingreso de nuevos para que no se amontonen antes de tiempo.

Agregamos un semáforo extra 'nuevosElfos' para asegurarse de que solo puede haber 3 elfos esperando a santa en la puerta de la tienda en el mismo momento y utilizamos 'elfosAtendidos' para contar cuántos hay y que el tercero pueda ir a despertar a santa cuando es necesario.

2. Problemas

2.1. Lectura - Seguridad: Fallo de disponibilidad (Comisión 36 a 47)

El fallo de disponibilidad es un problema crítico en el ámbito de la seguridad informática, ya que afecta directamente la capacidad de los usuarios para acceder a sistemas, datos o servicios cuando los necesitan. La disponibilidad garantiza que los sistemas estén operativos y accesibles en todo momento, lo que es vital para cualquier organización que dependa de la tecnología para sus operaciones diarias. Cuando se produce un fallo en la disponibilidad, los sistemas pueden volverse inaccesibles, lo que provoca interrupciones en las actividades comerciales, la prestación de servicios y, en casos graves, pone en riesgo la seguridad de datos críticos.

La disponibilidad se refiere a la capacidad de los sistemas y servicios para estar en funcionamiento continuo y accesibles cuando son requeridos. Forma parte del modelo de seguridad conocido como la CIA (Confidencialidad, Integridad y Disponibilidad). Mientras que la confidencialidad asegura que la información solo es accesible por personas autorizadas, y la integridad garantiza que los datos no son modificados de manera no autorizada, la disponibilidad asegura que los recursos están disponibles en todo momento para aquellos que los necesitan.

Un fallo de disponibilidad ocurre cuando una interrupción, ya sea planeada o no, impide que un sistema, aplicación o servicio esté operativo. En un entorno empresarial o industrial, la falta de disponibilidad puede causar desde inconvenientes menores hasta pérdidas financieras catastróficas, daños a la reputación o incluso riesgos para la seguridad de las personas.

Causas comunes de fallos de disponibilidad:

1. Actualizaciones Defectuosas: Los parches o actualizaciones de software que no fueron adecuadamente probados en entornos de prueba pueden causar conflictos o errores graves, como ocurrió en el caso de CrowdStrike.
2. Ataques de denegación de servicio: Un ataque de denegación de servicio intenta hacer que un sistema, red o servidor no esté disponible al sobrecargarlo con un gran volumen de solicitudes.

3. Fallas de hardware o infraestructura: Equipos obsoletos o fallos físicos en los sistemas de almacenamiento, redes o servidores pueden provocar interrupciones. Cortes de energía o desastres naturales también pueden afectar la disponibilidad.
4. Errores humanos: Las malas configuraciones, decisiones erróneas o falta de capacitación en el uso de sistemas pueden causar fallos en la disponibilidad.
5. Sobrecarga del Sistema: Una carga excesiva de usuarios o procesos puede superar los recursos disponibles del sistema, generando un fallo de disponibilidad.

Consecuencias de un fallo de disponibilidad:

1. Pérdida de productividad: Si el sistema afectado es una herramienta de trabajo, su inactividad puede llevar a interrupciones en los procesos productivos.

2. Impacto financiero: La inactividad puede provocar pérdidas financieras significativas, especialmente en empresas de comercio electrónico, servicios financieros y otras industrias en las que la disponibilidad de los sistemas es crítica para la generación de ingresos.

3. Daño a la reputación: Los usuarios suelen perder confianza en un servicio que falla frecuentemente.

4. Problemas de cumplimiento: En ciertos sectores regulados, como el financiero o el de la salud, la disponibilidad es un requisito normativo. Si no se cumple, la empresa puede enfrentar sanciones legales o multas.

5. Pérdida de datos y problemas de integridad: Algunos fallos de disponibilidad pueden generar inconsistencias en los datos o incluso pérdida de información, especialmente si el sistema falla durante una operación de actualización o transacción.

6. Incremento en costos de recuperación: Volver a poner en funcionamiento un sistema después de un fallo de disponibilidad puede requerir horas de trabajo, inversiones en infraestructura y, en algunos casos, la implementación de soluciones de respaldo o redundancia.

Prevención de Fallos de Disponibilidad:

Este caso de CrowdStrike resalta la importancia de tomar medidas preventivas para evitar fallos de disponibilidad. Algunas de las estrategias clave para prevenir estos fallos incluyen:

1. Pruebas exhaustivas antes de implementar actualizaciones: Es crucial que cualquier actualización de software pase por pruebas rigurosas en entornos controlados que simulen diversos escenarios de uso.
2. Monitoreo y detección de fallos en tiempo real: Las herramientas de monitoreo continuo permiten detectar problemas en tiempo real, lo que facilita la intervención temprana antes de que los errores se propaguen masivamente.
3. Planes de contingencia y recuperación ante desastres: Las empresas deben contar con un plan detallado para restaurar sistemas rápidamente en caso de fallos de disponibilidad. Esto puede incluir sistemas de respaldo y redundancia para garantizar que, si un sistema falla, otro pueda asumir sus funciones sin interrupciones significativas.

4. Redundancia de sistemas: Tener infraestructura duplicada en diferentes ubicaciones geográficas (alta disponibilidad) garantiza que, si un centro de datos experimenta problemas, otro puede asumir el control sin interrupción.

Los fallos de disponibilidad son un riesgo importante en el ámbito de la seguridad informática y pueden tener consecuencias graves si no se gestionan adecuadamente. La implementación de estrategias debe ser prioritaria en toda organización que dependa de sus sistemas para la continuidad de sus operaciones y para la seguridad de sus datos.

El análisis del fallo de disponibilidad en CrowdStrike enseña la importancia crítica de mantener sistemas accesibles y operativos en todo momento, especialmente en un contexto donde las organizaciones dependen intensamente de la tecnología para la continuidad de sus operaciones.

Bibliografía utilizada:

- https://www.lanacion.com.ar/el-mundo/que-es-crowdstrike-y-que-se-sabe-hasta-ahora-de-la-falla-que-causo-caos-internacional-nid19072024/
- https://www.linkedin.com/pulse/2024-crowdstrike-incident-simply-explained-patrick-mccormack-jdxqc
- https://www.infobae.com/america/agencias/2024/07/19/crowdstrike-confirma-que-el-error-de-actualizacion-de-su-plataforma-no-es-un-incidente-de-seguridad-ni-un-ciberataque/
- https://www.ibm.com/docs/es/powerha-aix/7.2?topic=aix-high-availability-versus-fault-tolerance
- https://historiadelaempresa.com/sistemas-de-alta-disponibilidad

FALLO DE DISPONIBILIDAD EN CROWDSTRIKE: LA MAYOR INTERRUPCIÓN DE IT DE LA HISTORIA

EL 19 DE JULIO DE 2024, UNA ACTUALIZACIÓN DEFECTUOSA DEL SOFTWARE FALCON SENSOR

UN FALLO DE DISPONIBILIDAD OCURRE CUANDO UN SISTEMA O SERVICIO CRÍTICO NO ESTÁ OPERATIVO O ACCESIBLE, LO QUE IMPIDE QUE LOS USUARIOS PUEDAN INTERACTUAR CON ÉL DE MANERA NORMAL. ESTOS FALLOS PUEDEN SER PROVOCADOS POR:

- ERRORES HUMANOS.
- ATAQUES CIBERNÉTICOS.
- PROBLEMAS EN LA INFRAESTRUCTURA TECNOLÓGICA.
- ERRORES EN EL SOFTWARE (CROWDSTRIKE).

IMPACTO

- IMPACTO EN LA SEGURIDAD INFORMÁTICA Y DISPONIBILIDAD DE SISTEMAS CRÍTICOS.
- PERDIDAS ECONÓMICAS Y REPUTACIONALES SIGNIFICATIVAS PARA LAS EMPRESAS.

ESTRATEGIAS DE SEGURIDAD Y RESILIENCIA

- PRUEBAS EXHAUSTIVAS DE ACTUALIZACIONES.
- REDUNDANCIA DE SISTEMAS.
- PLANES DE RECUPERACIÓN ANTE DESASTRES.
- MONITOREO CONTINUO.
- COPIAS DE SEGURIDAD.

2.2. Problemas Conceptuales

1. a)

Traducción:

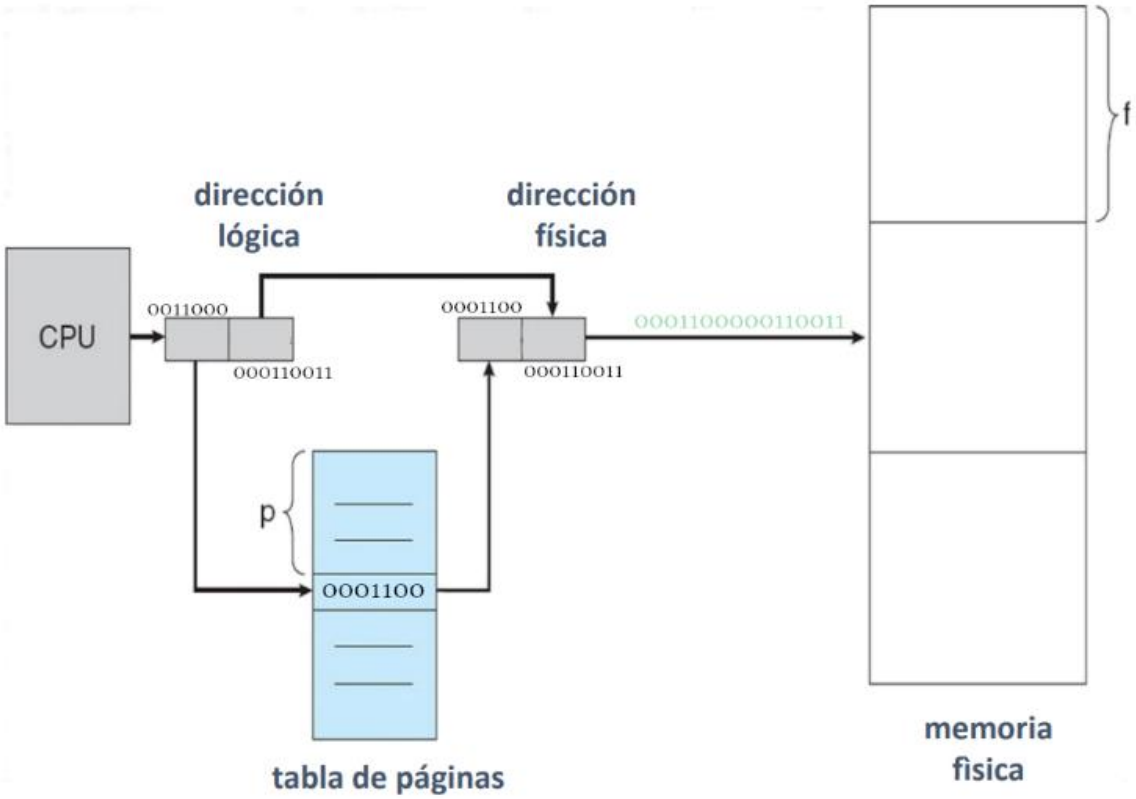
- Dirección lógica: 0011000000110011 (16 bits)
- Tamaño de página: 512 bytes.
- Desplazamiento dentro de la página $log_2(512) = 9$: Los últimos 9 bits representan el desplazamiento.
- Número de página: 16 - 9 = 7. Los primeros 7 bits representan el número de página 0011000 (en binario), que es 24 en decimal.
- Número de marco: El número de marco está dado por $M = \frac{P}{2}$. P = 24 entonces el número de marco será: $\frac{24}{2} = 12$.

Dirección física: La dirección física está formada por el número de marco seguido del desplazamiento dentro de la página.

- Número de marco en binario: 0001100 (7 bits).
- Desplazamiento en binario: 000110011 (9 bits).

Dirección física completa: 0001100000110011

0001100000110011
Número de páginaDesplazamiento

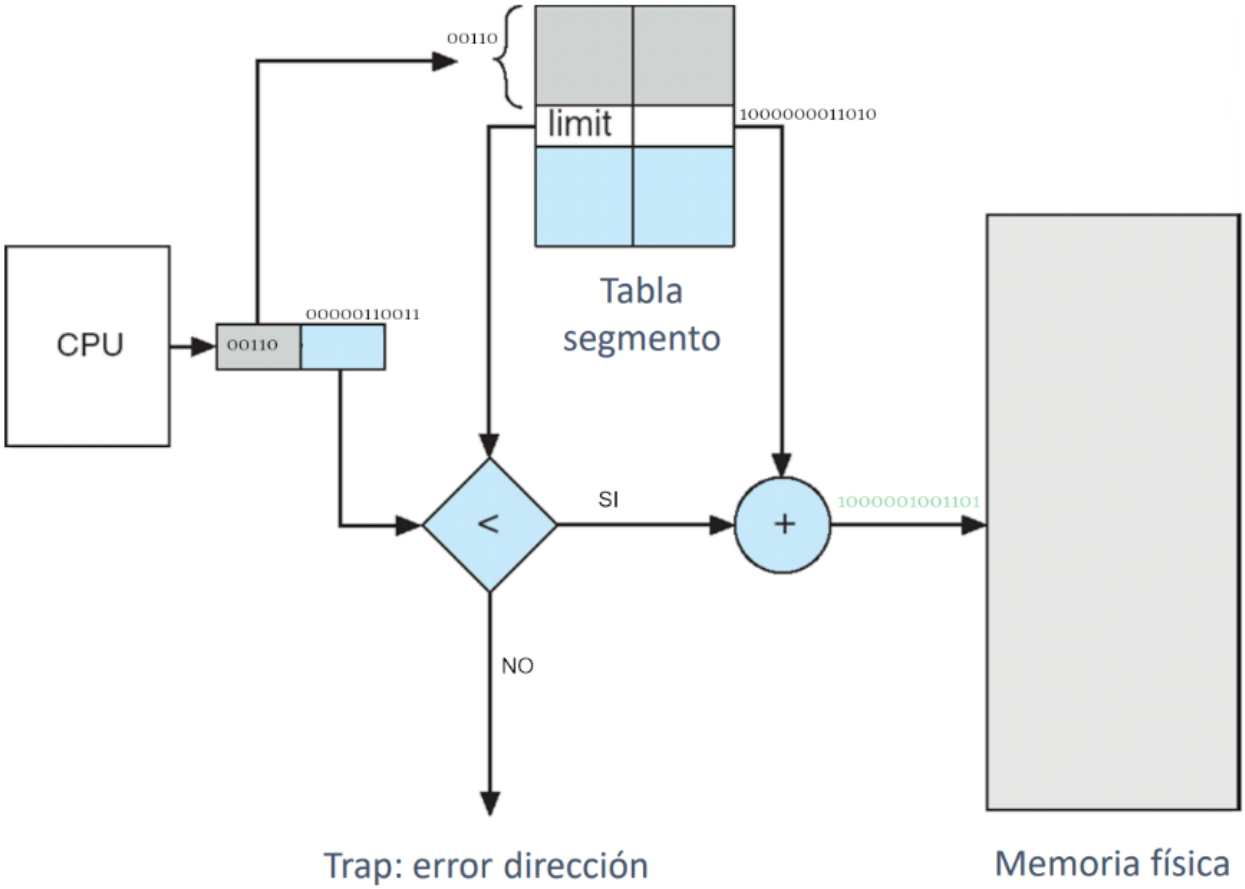


b)

Traducción:
El tamaño máximo de un segmento es de 2K direcciones, por lo que necesitamos 11 bits para el desplazamiento y el resto para el número de segmento.

- Dirección lógica: 0011000000110011
- Desplazamiento dentro del segmento (11 bits): Los últimos 11 bits representan el desplazamiento: 00000110011.
- Número de segmento 16 bits – 11 = 5 bits. Los primeros 5 bits representan el número de segmento: 00110 (en binario), que es 6 en decimal.
- Base = 20 + 4096 +6 = 4122 (1000000011010 en binario)
- Dirección física: Base + Desplazamiento = 4122+51 = 4173 (en decimal)

Dirección física completa = 1000001001101 (en binario)



2. a)

Dirección Lógica: 0x621C
En binario: 0110 0010 0001 1100
Número de Página: 0110 (página 6 en decimal)
Desplazamiento: 0010 0001 1100 (0x21C en hexadecimal)

Traducción:

- Página 6 está en el marco 8 (1000 en binario).
- Dirección física = 1000 0010 0001 1100. **0x821C en hexadecimal.**
- Bit de referencia para la página 6 se establece en 1.

Dirección Lógica: 0xF0A3
En binario: 1111 0000 1010 0011
Número de Página: 1111 (página 15 en decimal)
Desplazamiento: 0000 1010 0011 (0x0A3 en hexadecimal)

Traducción:

- Página 15 está en el marco 2 (0010 en binario).
- Dirección física = 0010 0000 1010 0011. **0x20A3 en hexadecimal.**
- Bit de referencia para la página 15 se establece en 1.

Dirección Lógica: 0xBC1A
En binario: 1011 1100 0001 1010
Número de Página: 1011 (página 11 en decimal)
Desplazamiento: 1100 0001 1010 (0x01A en hexadecimal)

Traducción:

- Página 11 está en el marco 4 (0100 en binario).
- Dirección física = 0100 1100 0001 1010. **0x4C1A en hexadecimal.**
- Bit de referencia para la página 11 se establece en 1.

Dirección Lógica: 0x5BAA
En binario: 0101 1011 1010 1010
Número de Página: 0101 (página 5 en decimal)
Desplazamiento: 1011 1010 1010 (0xBAA en hexadecimal)

Traducción:

- Página 5 está en el marco 13 (1101 en binario).
- Dirección física = 1101 1011 1010 1010. **0xDBAA en hexadecimal.**
- Bit de referencia para la página 5 se establece en 1.

Dirección Lógica: 0x0BA1

En binario: 0000 1011 1010 0001

Número de Página: 0000 (página 0 en decimal)

Desplazamiento: 1011 1010 0001 (0xBA1 en hexadecimal)

Traducción:

- Página 0 está en el marco 9 (1001 en binario).
- Dirección física = 1001 1011 1010 0001. **0x9BA1 en hexadecimal.**
- Bit de referencia para la página 0 se establece en 1.

Pagina	Marco	Bit Referencia
0	9	1
1	-	0
2	10	0
3	15	0
4	6	0
5	13	1
6	8	1
7	12	0
8	7	0
9	-	0
10	5	0
11	4	1
12	1	0
13	0	0
14	-	0
15	2	1

b)

Para que ocurra un fallo de página, necesitamos elegir una dirección virtual cuya página no esté cargada en memoria.

Ejemplo de dirección con fallo de página: 0x1000

- Número de página: 1
- Esta dirección virtual fallará porque la página 1 no está en memoria.

c)

Si ocurre un fallo de página y se requiere reemplazo, el conjunto de marcos de página entre los cuales el algoritmo LRU elegiría sería: 0, 1, 5, 6, 7, A, C, F (todos los marcos son bit de referencia 0).