Sistemas Operativos y Distribuidos

Iren Lorenzo Fonseca
iren.fonseca@.ua.es

TEMA 1. Sistemas Operativos.

Ejercicios

Pregunta 1

Procesos

Referente a los procesos de un sistema operativo contesta:

a) ¿Qué son los procesos?

b) Compara y contrasta los procesos independientes, cooperativos y competitivos, proporciona ejemplos para cada tipo y discute las ventajas y desafíos de cada uno.

c) ¿Los tipos de procesos son excluyentes?

Respuesta Pregunta 1

a

Un proceso:

- Es un programa que se está ejecutando, es la instancia activa de un programa que se ha cargado en la memoria y está en ejecución.
- Cada proceso tiene su propio espacio de direcciones en la memoria, lo que significa que posee su propia memoria para datos, código y pila. También tiene asignados recursos del sistema, como descriptores de archivo y acceso a dispositivos.
- El sistema operativo gestiona todos los aspectos de la ejecución del proceso, incluyendo la creación, planificación, cambio de contexto, y terminación del proceso.
- Para que un proceso se ejecute, debe ser cargado en la memoria principal (RAM), donde el CPU puede acceder a él.

Pregunta 1

Procesos

Referente a los procesos de un sistema operativo contesta:

a) ¿Qué son los procesos?

b) Compara y contrasta los procesos independientes, cooperativos y competitivos, proporciona ejemplos para cada tipo y discute las ventajas y desafíos de cada uno.

c) ¿Los tipos de procesos son excluyentes?

Respuesta Pregunta 1

b

Procesos Independientes
Los procesos independientes son aquellos que no requieren información de otros procesos y no afectan el comportamiento de otros procesos. Son totalmente autónomos y pueden ejecutarse en cualquier orden sin afectar el resultado.

Ejemplo: Un programa que calcula el promedio de una lista de números y otro que genera un gráfico de barras. Ambos programas pueden ejecutarse simultáneamente sin interferirse entre sí.

•**Ventajas**:

- **Simplicidad**: Al no depender de otros procesos, su diseño es más simple.
- **Robustez**: Un fallo en un proceso no afecta a otros procesos. No introduce problemas de interferencia o dependencia entre procesos.

Respuesta Pregunta 1

b

Procesos Cooperativos
Los procesos cooperativos son aquellos que necesitan colaborar entre sí, compartiendo datos y recursos. Estos procesos se comunican y coordinan para lograr un objetivo común.

Ejemplo: Un sistema de procesamiento de pedidos en línea, donde un proceso maneja las solicitudes de los clientes y otro gestiona el inventario. Ambos procesos deben trabajar juntos para completar un pedido.

Ventajas:

- **Eficiencia**: Pueden mejorar la eficiencia mediante la colaboración y el intercambio de información.
- **Optimización de recursos**: Los recursos pueden ser utilizados de manera más efectiva.

•**Desafíos**:

- **Complejidad en la gestión**: Se requiere un control cuidadoso de la comunicación y la sincronización entre procesos.
- **Riesgo de interbloqueo (deadlock)**: Si no se manejan adecuadamente, pueden producirse interbloqueos, donde los procesos quedan esperando indefinidamente.

Respuesta Pregunta 1

b

Procesos Competitivos
Los procesos competitivos son aquellos que compiten por los mismos recursos del sistema, lo que puede causar conflictos de acceso.

Ejemplo: Dos procesos que intentan acceder a un archivo en el disco duro al mismo tiempo.

•**Desafíos**:

- **Condiciones de carrera**: Puede haber situaciones donde los resultados dependan del orden de ejecución, lo que puede llevar a problemas de condición de carrera
- **Necesidad de mecanismos de sincronización**: Requieren un manejo cuidadoso de la sincronización y la exclusión mutua para prevenir conflictos.
- En sistemas mal diseñados puede haber problemas de interbloqueo e inanición.

Respuesta Pregunta 1

b

Procesos Competitivos
Los procesos competitivos son aquellos que compiten por los mismos recursos del sistema, lo que puede causar conflictos de acceso.

Ejemplo: Dos procesos que intentan acceder a un archivo en el disco duro al mismo tiempo.

•**Desafíos**:

- **Condiciones de carrera**: Puede haber situaciones donde los resultados dependan del orden de ejecución, lo que puede llevar a problemas de condición de carrera
- **Necesidad de mecanismos de sincronización**: Requieren un manejo cuidadoso de la sincronización y la exclusión mutua para prevenir conflictos.
- En sistemas mal diseñados puede haber problemas de interbloqueo e inanición.

Pregunta 1

Procesos

Referente a los procesos de un sistema operativo contesta:

a) ¿Qué son los procesos?

b) Compara y contrasta los procesos independientes, cooperativos y competitivos, proporciona ejemplos para cada tipo y discute las ventajas y desafíos de cada uno.

c) ¿Los tipos de procesos son excluyentes?

Respuesta Pregunta 1

C

- Los procesos independientes y cooperativos son excluyentes entre sí.
- Los procesos competitivos pueden ser independientes o cooperativos, pero la característica clave es la competencia por recursos limitados

Pregunta 2

Procesos

2. Sobre los estados de procesos y transiciones responda:

a) Describe los diferentes estados por los que puede pasar un proceso durante su ciclo de vida en un sistema operativo (no es necesario incluir el estado suspendido).

b) Proporciona un diagrama que represente estos estados y las posibles transiciones entre ellos

Respuesta Pregunta 2

a

Los estados típicos de un proceso incluyen:

- **Nuevo**: El proceso se ha creado.
- **Listo** : El proceso está listo para ejecutarse, pero esperando asignación de la CPU.
- **En ejecución**: El proceso está siendo ejecutado por la CPU.
- **Bloqueado** : El proceso no puede continuar hasta que se cumpla alguna condición externa.
- **Terminado**: El proceso ha finalizado su ejecución.

Pregunta 2

Procesos

2. Sobre los estados de procesos y transiciones responda:

a) Describe los diferentes estados por los que puede pasar un proceso durante su ciclo de vida en un sistema operativo (no es necesario incluir el estado suspendido).

b) Proporciona un diagrama que represente estos estados y las posibles transiciones entre ellos

Respuesta Pregunta 2

b

Posibles transiciones:

- [Nuevo] -> [Listo]
- [Listo] -> [En ejecución]
- [En ejecución] -> [Terminado]
- [En ejecución] -> [Listo]
- [En ejecución] -> [Bloqueado]
- [Bloqueado] -> [Listo]

Pregunta 3

Procesos

El proceso A crea un nuevo proceso B utilizando la llamada al sistema fork().

a) Describe detalladamente los pasos que realiza el sistema operativo para crear el nuevo proceso B desde el proceso A. Incluye la asignación de PID, la creación del PCB, y cómo se inicializa el proceso B.

b) Especifica dónde almacena el SO las siguientes estructuras de datos del proceso A dentro de la memoria principal (espacio de memoria del proceso o memoria del kernel): segmento de código del proceso, colas de procesos, tabla de descriptores de archivo del proceso, tabla de archivos abiertos, tabla de procesos.

c) Que llamadas se podrían hacer desde el proceso A para que el proceso quedar en estado bloqueado y para que se utilizan esas llamadas

Respuesta Pregunta 3

a

1. Asignación de un PID único para el nuevo proceso.
2. Creación del PCB que almacena información del proceso: Ej. PID, PPID, estado del proceso, contador del programa, espacio de direcciones, prioridad, cola de planificación, tiempo de uso de la CPU, tiempo de espera de la CPU, etc.
3. Asignación de espacio de memoria para el proceso.
4. Copia del contexto del proceso padre.
5. Inicialización del proceso configurando el contador de programa y la pila.
6. Colocación en la cola de planificación para ser programado.
7. Transición al estado listo para comenzar su ejecución.

Pregunta 3

Procesos

El proceso A crea un nuevo proceso B utilizando la llamada al sistema fork().

a) Describe detalladamente los pasos que realiza el sistema operativo para crear el nuevo proceso B desde el proceso A. Incluye la asignación de PID, la creación del PCB, y cómo se inicializa el proceso B.

b) Especifica dónde almacena el SO las siguientes estructuras de datos del proceso A dentro de la memoria principal (espacio de memoria del proceso o memoria del kernel): segmento de código del proceso, colas de procesos, tabla de descriptores de archivo del proceso, tabla de archivos abiertos, tabla de procesos.

c) Que llamadas se podrían hacer desde el proceso A para que el proceso quedar en estado bloqueado y para que se utilizan esas llamadas

Respuesta Pregunta 3

b

Segmento de Código del Proceso

Almacenamiento: Espacio de memoria del proceso.

Este segmento contiene el código ejecutable del proceso A. Es la parte de la memoria donde se almacena el código de la aplicación en ejecución y se encuentra en el espacio de direcciones del proceso.

Colas de Procesos:

Almacenamiento: Memoria del kernel.

Las colas de procesos (como la cola de listos, bloqueados, etc.) se almacenan en la memoria del kernel. Estas estructuras son utilizadas por el sistema operativo para gestionar el estado de los procesos y su programación.

Tabla de Descriptores de Archivo del Proceso:

Almacenamiento: Espacio de memoria del proceso.

Esta tabla se encuentra en el espacio de memoria del proceso A. Contiene información sobre los descriptores de archivo abiertos por el proceso, incluyendo punteros a la tabla de archivos abiertos del sistema.

Respuesta Pregunta 3

b

Tabla de Archivos Abiertos

Almacenamiento: Memoria del kernel.

Esta tabla es global y se almacena en la memoria del kernel. Contiene información sobre todos los archivos abiertos en el sistema, como el estado del archivo, la posición de lectura/escritura, y el acceso permitido. Cada proceso hace referencia a esta tabla mediante los descriptores de archivo en su tabla de descriptores de archivo.

Tabla de Procesos

Almacenamiento: Memoria del kernel.

La tabla de procesos, que contiene un PCB (Bloque de Control del Proceso) para cada proceso en el sistema, se almacena en la memoria del kernel. Esta tabla es fundamental para la gestión de procesos, ya que permite al sistema operativo acceder y controlar el estado y la información de cada proceso.

Respuesta Pregunta 3

b

Estructura de Datos	Almacenamiento
Segmento de Código del Proceso	Espacio de memoria del proceso
Colas de Procesos	Memoria del kernel
Tabla de Descriptores de Archivo del Proceso	Espacio de memoria del proceso
Tabla de Archivos Abiertos	Memoria del kernel
Tabla de Procesos	Memoria del kernel

Pregunta 3

Procesos

El proceso A crea un nuevo proceso B utilizando la llamada al sistema fork().

a) Describe detalladamente los pasos que realiza el sistema operativo para crear el nuevo proceso B desde el proceso A. Incluye la asignación de PID, la creación del PCB, y cómo se inicializa el proceso B.
b) Especifica dónde almacena el SO las siguientes estructuras de datos del proceso A dentro de la memoria principal (espacio de memoria del proceso o memoria del kernel): segmento de código del proceso, colas de procesos, tabla de descriptores de archivo del proceso, tabla de archivos abiertos, tabla de procesos.
c) Que llamadas se podrían hacer desde el proceso A para que el proceso quedar en estado bloqueado y para que se utilizan esas llamadas

Respuesta Pregunta 3

C

El proceso A puede quedar en estado bloqueado mediante varias llamadas al sistema, las cuales son útiles para sincronización y control del flujo de ejecución. Algunas de estas llamadas son:

wait()
El proceso A puede llamar a wait() para esperar la terminación de su proceso hijo (B). Cuando se llama a wait(), el proceso A se bloquea hasta que el proceso B finaliza su ejecución. Esto es útil para que el proceso padre no continúe ejecutándose hasta que el hijo haya completado su tarea.

pause()
Al invocar pause(), el proceso A se bloquea hasta que recibe una señal. Esta llamada es útil para esperar la ocurrencia de un evento o la llegada de una señal que indique que se debe continuar.

Respuesta Pregunta 3

C

sleep(seconds)
La llamada sleep() puede hacer que el proceso A se bloquee durante un período específico. Durante este tiempo, el proceso no consume CPU y se vuelve inactivo.

Lectura de archivos o tuberías (read())
Si el proceso A intenta leer desde un archivo o una tubería y no hay datos disponibles, el proceso se bloqueará hasta que haya datos para leer. Esto asegura que el proceso no siga ejecutándose con datos incompletos.

Operaciones de semáforos (wait())
En un entorno de concurrencia, el proceso A puede intentar realizar una operación de espera en un semáforo (por ejemplo, wait()). Si el semáforo está bloqueado (valor 0), el proceso A se quedará en estado bloqueado hasta que el semáforo sea liberado.

Pregunta 4

Hilos

Tienes que entrenar un modelo de inteligencia artificial que requiere procesamiento intensivo en CPU.

a) ¿Optarías por hilos a nivel de usuario o a nivel de kernel?

b) Los hilos gestionados por el SO en qué espacio de memoria se encuentran

c) ¿Optarías por usar multiprocessing o multithreading en Python? Justifica tu respuesta basándote en la capacidad de ambos enfoques para manejar tareas intensivas en CPU y el impacto del GIL.

Respuesta Pregunta 4

a

Optaría por hilos a nivel de kernel.

Optar por hilos a nivel de kernel para tareas de procesamiento intensivo en CPU tiene varias ventajas en comparación con los hilos a nivel de usuario, especialmente debido a las desventajas inherentes de la gestión de hilos a nivel de usuario.

Ventajas de los Hilos a Nivel de Kernel

Planificación y Gestión Eficiente:
Los hilos a nivel de kernel son gestionados directamente por el SO, lo que permite que el scheduler del SO optimice la asignación de CPU entre hilos en función de su prioridad y necesidades. Esto es especialmente útil en sistemas multiprocesador, donde los hilos pueden ejecutarse en núcleos diferentes, mejorando el rendimiento general.

Interrupciones de Bloqueo:
Si un hilo a nivel de kernel se bloquea (por ejemplo, esperando I/O), el sistema operativo puede desprogramar ese hilo y asignar la CPU a otros hilos dentro del mismo proceso o incluso a otros procesos. Esto mejora la utilización de la CPU.

Respuesta Pregunta 4

a

Acceso a Recursos del Sistema:
Los hilos a nivel de kernel tienen acceso directo a las funciones del sistema operativo y a los recursos del sistema, lo que les permite gestionar mejor la memoria y otros recursos del sistema.

Multihilo Verdadero:
En un entorno con múltiples núcleos, los hilos a nivel de kernel pueden ejecutarse simultáneamente en diferentes núcleos, maximizando el rendimiento de las aplicaciones intensivas en CPU.

Respuesta Pregunta 4

a

Desventajas de los Hilos a Nivel de Usuario

Planificación Menos Eficiente:
La gestión de hilos a nivel de usuario es menos eficiente porque el sistema operativo no es consciente de ellos. Todo el trabajo de programación y planificación se realiza dentro del proceso, lo que puede conducir a un uso ineficiente de la CPU, especialmente en aplicaciones que requieren un alto rendimiento.

Bloqueo del Proceso:
Si un hilo a nivel de usuario se bloquea (por ejemplo, esperando I/O), el proceso completo se bloquea, lo que impide que otros hilos se ejecuten. Esto es un gran inconveniente para la concurrencia, ya que limita la capacidad de aprovechar la CPU cuando uno de los hilos está inactivo.

Respuesta Pregunta 4

a

- Si bien los hilos a nivel de usuario ofrecen ventajas como una menor sobrecarga en la creación y destrucción de hilos y la facilidad de implementación, las desventajas superan estas ventajas en escenarios de procesamiento intensivo en CPU.
- Las limitaciones en la planificación y la gestión de bloqueos hacen que los hilos a nivel de kernel sean la opción preferida, ya que proporcionan una gestión más eficiente, una mejor utilización de los recursos del sistema y una capacidad superior para escalar en entornos multiprocesador.

Pregunta 4

Hilos

Tienes que entrenar un modelo de inteligencia artificial que requiere procesamiento intensivo en CPU.

a) ¿Optarías por hilos a nivel de usuario o a nivel de kernel?

b) Los hilos gestionados por el SO en qué espacio de memoria se encuentran

c) ¿Optarías por usar multiprocessing o multithreading en Python? Justifica tu respuesta basándote en la capacidad de ambos enfoques para manejar tareas intensivas en CPU y el impacto del GIL.

Respuesta Pregunta 4

b

- **Hilos (a nivel de usuario y a nivel de kernel):** Residen en el **espacio de direcciones del proceso**. Cada hilo tiene su propia pila, pero comparten el código y datos del proceso.

Pregunta 4

Hilos

Tienes que entrenar un modelo de inteligencia artificial que requiere procesamiento intensivo en CPU.

a) ¿Optarías por hilos a nivel de usuario o a nivel de kernel?

b) Los hilos gestionados por el SO en qué espacio de memoria se encuentran

c) ¿Optarías por usar multiprocessing o multithreading en Python? Justifica tu respuesta basándote en la capacidad de ambos enfoques para manejar tareas intensivas en CPU y el impacto del GIL.

Respuesta Pregunta 4

Procesos

Opción: Multiprocessing

Justificación:

- Las tareas intensivas en CPU, como el entrenamiento de modelos de inteligencia artificial, se benefician más del multiprocesamiento.
- Cada proceso en Python tiene su propio espacio de memoria y puede ejecutarse en un núcleo diferente, evitando el Global Interpreter Lock (GIL) de Python, que limita la ejecución de hilos a uno a la vez en un único proceso.
- El GIL es una restricción en CPython (la implementación más común de Python) que permite que solo un hilo ejecute código de Python a la vez, lo que puede ser un obstáculo para el rendimiento en aplicaciones que requieren mucha CPU. Al usar multiprocessing, cada proceso tiene su propia instancia de Python y, por lo tanto, el GIL no se convierte en un problema.
- Aunque multithreading es útil para operaciones de I/O, el multiprocessing permite un mejor rendimiento en cálculos intensivos, facilitando el uso de bibliotecas que pueden ejecutar operaciones en paralelo.

Pregunta 5

Planificación de Procesos

Dados los siguientes procesos con sus tiempos de llegada y tiempos de CPU:

a) Aplica el algoritmo Round Robin con un quantum de 2 unidades de tiempo. Determina el orden de ejecución, el tiempo de respuesta y el tiempo de retorno para cada proceso.

b) Determina el tiempo promedio de retorno y de respuesta para estos procesos

c) ¿Qué significa el tiempo de retorno y de respuesta obtenido ?

Proceso	Tiempo de llegada	Tiempo de CPU
P1	0	5
P2	1	3
P3	2	8
P4	3	6

Respuesta Pregunta 5

a

Proceso	llegada	CPU
P1	0	5
P2	1	3
P3	2	8
P4	3	6

Tiempo de llegada: Momento en que el proceso entra en la cola de procesos listos.
Tiempo de CPU: Tiempo total que el proceso necesita para completar su ejecución.
Quantum: Tiempo máximo que un proceso puede ejecutar antes de que se le desaloje y se dé oportunidad a otro proceso (en este caso, 2 unidades de tiempo).

P1: Tiempo de llegada = 0, Tiempo de CPU = 5
P2: Tiempo de llegada = 1, Tiempo de CPU = 3
P3: Tiempo de llegada = 2, Tiempo de CPU = 8
P4: Tiempo de llegada = 3, Tiempo de CPU = 6

Respuesta Pregunta 5

a

P1: Tiempo de llegada = 0, Tiempo de CPU = 5
P2: Tiempo de llegada = 1, Tiempo de CPU = 3
P3: Tiempo de llegada = 2, Tiempo de CPU = 8
P4: Tiempo de llegada = 3, Tiempo de CPU = 6

Tiempo	Proceso ejecutando	CPU restante	Cola de procesos final	Observaciones
0-2	P1	3	P2(3), P3(8), P4(6), P1(3)	P1 ejecuta 2 unidades, quedan 3
2-4	P2	1	P3(8), P4(6), P1(3), P2(1)	P2 ejecuta 2 unidades, quedan 1
4-6	P3	6	P4(6), P1(3), P2(1), P3(6)	P3 ejecuta 2 unidades, quedan 6
6-8	P4	4	P1(3), P2(1), P3(6), P4(4)	P4 ejecuta 2 unidades, quedan 4
8-10	P1	1	P2(1), P3(6), P4(4), P1(1)	P1 ejecuta 2 unidades, quedan 1
10-11	P2	0	P3(6), P4(4), P1(1)	P2 ejecuta 1 unidad, termina
11-13	P3	4	P4(4), P1(1), P3(4)	P3 ejecuta 2 unidades, quedan 4
13-15	P4	2	P1(1), P3(4), P4(2)	P4 ejecuta 2 unidades, quedan 2
15-16	P1	0	P3(4), P4(2)	P1 ejecuta 1 unidad, termina
16-18	P3	2	P4(2), P3(2)	P3 ejecuta 2 unidades, quedan 2
18-20	P4	0	P3(2)	P4 ejecuta 2 unidades, termina
20-21	P3	0	-	P3 ejecuta 2 unidades, termina

Pregunta 5

Planificación de Procesos

Dados los siguientes procesos con sus tiempos de llegada y tiempos de CPU:

a) Aplica el algoritmo Round Robin con un quantum de 2 unidades de tiempo. Determina el orden de ejecución, el tiempo de respuesta y el tiempo de retorno para cada proceso.

b) Determina el tiempo promedio de retorno y de respuesta para estos procesos

c) ¿Qué significa el tiempo de retorno y de respuesta obtenido ?

Proceso	Tiempo de llegada	Tiempo de CPU
P1	0	5
P2	1	3
P3	2	8
P4	3	6

Respuesta Pregunta 5

b

Tiempos de respuesta:

- **P1**: 0 (llega y empieza a ejecutar)
- **P2**: 1 (llega a 1, empieza a 2)
- **P3**: 2 (llega a 2, empieza a 4)
- **P4**: 3 (llega a 3, empieza a 6)

Tiempos de retorno:

- **P1**: 16 (termina a 16, llegó a 0, entonces 16 - 0 = 16)
- **P2**: 10 (termina a 11, llegó a 1, entonces 11 - 1 = 10)
- **P3**: 19 (termina a 21, llegó a 2, entonces 21 - 2 = 19)
- **P4**: 17 (termina a 20, llegó a 3, entonces 20 - 3 = 17)

Promedios:

- **Tiempo de respuesta promedio**: (0 + 1 + 2 + 3) / 4 = 1.5
- **Tiempo de retorno promedio**: (16 + 10 + 19 + 17) / 4 = 15.5

Pregunta 5

Planificación de Procesos

Dados los siguientes procesos con sus tiempos de llegada y tiempos de CPU:

a) Aplica el algoritmo Round Robin con un quantum de 2 unidades de tiempo. Determina el orden de ejecución, el tiempo de respuesta y el tiempo de retorno para cada proceso.

b) Determina el tiempo promedio de retorno y de respuesta para estos procesos

c) ¿Qué significa el tiempo de retorno y de respuesta obtenido ?

Proceso	Tiempo de llegada	Tiempo de CPU
P1	0	5
P2	1	3
P3	2	8
P4	3	6

Respuesta Pregunta 5

C

El tiempo de retorno y el tiempo de respuesta promedio son métricas clave para evaluar el rendimiento de un algoritmo de planificación de CPU.

Tiempo de Retorno Promedio

El **tiempo de retorno** es el tiempo total que un proceso tarda desde que llega al sistema hasta que finaliza. Incluye el tiempo de espera en la cola, el tiempo de ejecución en la CPU y cualquier tiempo de bloqueo. Un menor tiempo de retorno promedio indica que los procesos están completando su ejecución más rápidamente, lo que es deseable en muchos sistemas.

Respuesta Pregunta 5

C

•**Tiempo de retorno promedio obtenido**: 15.5 unidades de tiempo.

- Este valor relativamente alto sugiere que, aunque los procesos reciben atención periódicamente debido a la naturaleza cíclica de RR, el constante cambio de contexto y la sobrecarga asociada pueden aumentar el tiempo total que los procesos pasan en el sistema.
- En escenarios donde hay muchos procesos o procesos con tiempos de CPU largos, RR puede resultar en mayores tiempos de retorno comparado con otros algoritmos más simples como FIFO.

Respuesta Pregunta 5

C

Tiempo de Respuesta Promedio

El **tiempo de respuesta** es el tiempo desde que un proceso llega al sistema hasta que empieza su primera ejecución en la CPU. Un menor tiempo de respuesta promedio es crítico en sistemas donde la interactividad es importante, ya que reduce la latencia inicial percibida por los usuarios.

Respuesta Pregunta 5

C

•**Tiempo de respuesta promedio obtenido**: 1.5 unidades de tiempo

- Este valor bajo es un punto fuerte de RR, ya que muestra que los procesos empiezan a ejecutarse rápidamente después de llegar al sistema.
- RR es especialmente beneficioso en sistemas donde es crucial que todos los procesos obtengan acceso a la CPU en un corto periodo de tiempo, como en sistemas interactivos.

Pregunta 6

Sistema de archivos

El SO utiliza para la memoria secundaria Asignación Contigua. Supón que tienes un disco dividido en bloques de 4 KB cada uno. Los bloques pueden estar llenos (asignados a un archivo) o vacíos (disponibles para asignación). La distribución inicial de los bloques y su estado se representa en la siguiente tabla:

Bloque	Estado	Tamaño (KB)
1	Vacío	4
2	Lleno	4
3	Vacío	4
4	Vacío	4
5	Vacío	4
6	Lleno	4
7	Vacío	4
8	Vacío	4
9	Lleno	4

Pregunta 6

Sistema de archivos

Ahora, llegan tres archivos para ser almacenados en el disco con los siguientes tamaños y tiempos de llegada:

Archivo	Tamaño (KB)	Tiempo de Llegada
A	6	0
B	4	1
C	8	2

a) Utiliza el algoritmo del primer hueco para asignar los archivos A, B y C a los bloques disponibles del disco. Debes mostrar cómo se realiza la asignación y qué bloques se ocupan.

b) ¿Cuáles son las ventajas y desventajas de la asignación contigua?

c) ¿En el ejemplo hay fragmentación interna? En caso afirmativo explica dónde y qué significa la fragmentación interna

Respuesta Pregunta 6

a

Archivo	Tamaño (KB)	Tiempo de Llegada
A	6	0
B	4	1
C	8	2

El archivo A necesita 2 bloques (4 KB cada uno). En total, necesitará 8 KB, pero solo requerirá 6 KB. Primer Hueco: Se revisan los bloques desde el principio y encuentra contiguos los bloques 3 y 4

El archivo B necesita 1 bloque (4 KB). Primer Hueco: Se revisan los bloques desde el principio y encuentra el bloque 1

El archivo C necesita 2 bloques (4 KB cada uno). Primer Hueco: Se revisan los bloques desde el principio y encuentra contiguos los bloques 7 y 8

Pregunta 6

Sistema de archivos

Ahora, llegan tres archivos para ser almacenados en el disco con los siguientes tamaños y tiempos de llegada:

Archivo	Tamaño (KB)	Tiempo de Llegada
A	6	0
B	4	1
C	8	2

a) Utiliza el algoritmo del primer hueco para asignar los archivos A, B y C a los bloques disponibles del disco. Debes mostrar cómo se realiza la asignación y qué bloques se ocupan.

b) ¿Cuáles son las ventajas y desventajas de la asignación contigua?

c) ¿En el ejemplo hay fragmentación interna? En caso afirmativo explica dónde y qué significa la fragmentación interna

Respuesta Pregunta 6

b

Ventajas

1.Simplicidad:

La gestión y la implementación son más simples porque los bloques de un archivo están almacenados de manera consecutiva. Esto facilita la lectura y escritura de datos.

2.Rendimiento:

La asignación contigua permite un acceso secuencial rápido, lo que mejora el rendimiento de las operaciones de entrada/salida (I/O). Al leer datos contiguos, el disco puede transferirlos de manera más eficiente, minimizando el tiempo de búsqueda.

3.Facilidad de acceso:

La localización de los bloques de un archivo es sencilla, lo que facilita la búsqueda y la gestión de archivos, ya que se puede calcular fácilmente la dirección del bloque inicial y los siguientes.

Respuesta Pregunta 6

b

Desventajas

1.Fragmentación Interna:

Puede ocurrir fragmentación interna si los archivos no utilizan completamente los bloques asignados. Por ejemplo, si un archivo de 6 KB se asigna a dos bloques de 4 KB, quedará un espacio de 2 KB sin usar.

2.Dificultades para el Crecimiento Dinámico:

Si un archivo necesita más espacio (por ejemplo, al crecer), puede ser difícil encontrar suficientes bloques contiguos libres, lo que limita la capacidad de crecimiento de los archivos.

Pregunta 6

Sistema de archivos

Ahora, llegan tres archivos para ser almacenados en el disco con los siguientes tamaños y tiempos de llegada:

Archivo	Tamaño (KB)	Tiempo de Llegada
A	6	0
B	4	1
C	8	2

a) Utiliza el algoritmo del primer hueco para asignar los archivos A, B y C a los bloques disponibles del disco. Debes mostrar cómo se realiza la asignación y qué bloques se ocupan.

b) ¿Cuáles son las ventajas y desventajas de la asignación contigua?

c) ¿En el ejemplo hay fragmentación interna? En caso afirmativo explica dónde y qué significa la fragmentación interna

Respuesta Pregunta 6

C

La **fragmentación interna** se refiere al espacio desperdiciado dentro de un bloque asignado a un archivo. Esto ocurre cuando un archivo no utiliza completamente el espacio asignado. Es decir, se asigna más espacio del que realmente necesita para almacenar su contenido.

En el Ejemplo
En el caso del **Archivo A**, que ocupa 6 KB, se asignaron 8 KB (2 bloques de 4 KB cada uno). Como resultado, hay un **desperdicio de 2 KB** en el bloque 4 porque el archivo no utiliza la totalidad del espacio asignado en ese bloque.

Pregunta 7

Concurrencia

El **problema del barbero durmiente** es un clásico problema de sincronización en programación concurrente que ilustra la interacción entre un barbero y sus clientes en una barbería. La barbería tiene un número limitado de sillas para los clientes que esperan. Aquí está la descripción del escenario:

- **Barbero**: El barbero corta el cabello a los clientes. Si no hay clientes, el barbero se duerme.
- **Clientes**: Los clientes llegan a la barbería. Si encuentran al barbero durmiendo, lo despiertan. Si no hay sillas disponibles, se van.
- **Sillas**: Hay un número finito de sillas en la barbería donde los clientes pueden esperar su turno.

El objetivo es manejar la sincronización entre el barbero y los clientes de manera que se eviten situaciones de carrera y se garantice que cada cliente sea atendido correctamente.

Pregunta 7

Concurrencia

```
int num_clientes = 0;

void cliente() {
    while (true) {
        num_clientes++; // Un nuevo cliente entra
              // Obtener corte de cabello
        esperar(); // Simular tiempo de corte
        num_clientes--; // Un cliente se va
    }
}
```

```
void barbero() {
    while (true) {

        // Cortar el cabello
        trabajar(); // Simular tiempo de corte
    }
}
```

```
void main() {
        cobegin
        barbero(); // Iniciar el proceso del barbero
        // Levantar múltiples clientes
        for (int i = 0; i < 10; i++) {
            cliente();
        }
      coend;
}
```

En la programación en seudocódigo siguiente:

a) Identifique la sección crítica
b) Utilice semáforos para:
    a) Exclusión mutua de la sección critica
    b) Limitar el número de clientes a 5, siendo el barbero quien habilita espacios cuando termine con un corte
    c) Sincronizar procesos para cuando llegue el primer cliente despierte al barbero y si no hay vuelva a dormir

Respuesta Pregunta 7

a

```
int num_clientes = 0;
```

```
void cliente() {
    while (true) {
        num_clientes++; // Un nuevo cliente entra
            // Obtener corte de cabello
        esperar(); // Simular tiempo de corte
        num_clientes--; // Un cliente se va
    }
}
```

```
void barbero() {
    while (true) {

        // Cortar el cabello
        trabajar(); // Simular tiempo de corte
    }
}
```

```
void main() {
        cobegin
        barbero(); // Iniciar el proceso del barbero
        // Levantar múltiples clientes
        for (int i = 0; i < 10; i++) {
            cliente();
        }
    coend;
}
```

La sección crítica se refiere a las partes del código donde se accede a recursos compartidos que deben ser protegidos para evitar condiciones de carrera. En este caso, los recursos compartidos son la variable num_clientes, que lleva el conteo de los clientes en la barbería.
Existen varios clientes concurrente por lo que este aumento y decremento hay que protegerlo para que no haya condición de carrera.

Respuesta Pregunta 7

b-a

```
TSemáforo mutex;
int num_clientes = 0;

void cliente() {
    while (true) {
        wait(mutex);
        num_clientes++; // Un nuevo cliente entra
        signal(mutex);
       // Obtener corte de cabello
        esperar(); // Simular tiempo de corte
        wait(mutex);
        num_clientes--; // Un cliente se va
       signal(mutex);
    }
}
void main() {
     inicializar(mutex, 1);
        cobegin
        barbero(); // Iniciar el proceso del barbero
        // Levantar múltiples clientes
        for (int i = 0; i < 10; i++) {
            cliente();
        }
    coend;
}
```

```
void barbero() {
    while (true) {

        // Cortar el cabello
        trabajar(); // Simular tiempo de corte
    }
}
```

En la programación en seudocódigo siguiente:

a) Identifique la sección crítica

b) Utilice semáforos para:

   a) Exclusión mutua de la sección critica

   b) Limitar el número de clientes a 5, siendo el barbero quien habilita espacios cuando termine con un corte

   c) Sincronizar procesos para cuando llegue el primer cliente despierte al barbero y si no hay vuelva a dormir

Respuesta Pregunta 7

b-b

```
TSemáforo mutex, s_cliente;

int num_clientes = 0;
const int MAX_CLIENTES = 5

void cliente() {
    while (true) {
        wait(s_cliente);
        wait(mutex);
        num_clientes++; // Un nuevo cliente entra
        signal(mutex);
       // Obtener corte de cabello
        esperar(); // Simular tiempo de corte
        wait(mutex);
        num_clientes--; // Un cliente se va
       signal(mutex);
    }
}
void main() {
     inicializar(mutex, 1);
     inicializar(s_cliente, MAX_CLIENTES);
      cobegin
      barbero(); // Iniciar el proceso del barbero
      // Levantar múltiples clientes
      for (int i = 0; i < 10; i++) {
          cliente();
      }
   coend;
}
```

```
void barbero() {
   while (true) {

      // Cortar el cabello
      trabajar(); // Simular tiempo de corte
       signal(s_cliente); // un cliente ha sido atendido
   }
}
```

En la programación en seudocódigo siguiente:

a) Identifique la sección crítica

b) Utilice semáforos para:

- **a) Exclusión mutua de la sección critica**
- **b) Limitar el número de clientes a 5, siendo el barbero quien habilita espacios cuando termine con un corte**

c) Sincronizar procesos para cuando llegue el primer cliente despierte al barbero y si no hay vuelva a dormir

b-c

```
TSemáforo mutex, s_cliente, s_barbero;
int num_clientes = 0;
const int MAX_CLIENTES = 5

void cliente() {
   while (true) {
      wait(s_cliente);
      wait(mutex);
      num_clientes++; // Un nuevo cliente entra
      if (num_clientes == 1) {
         signal(s_barbero); // Despertar barbero si primer cliente
      }
      signal(mutex);
     // Obtener corte de cabello
      esperar(); // Simular tiempo de corte
      wait(mutex);
      num_clientes--; // Un cliente se va
     signal(mutex);
   }
}
```

```
void barbero() {
   while (true) {
        wait(s_barbero); // Dormir hasta
                         que haya un cliente
      // Cortar el cabello
      trabajar(); // Simular tiempo de corte
       signal(s_cliente); // un cliente ha sido atendido
   }
}
```

```
void main() {
     inicializar(mutex, 1);
     inicializar(s_cliente, MAX_CLIENTES);
     inicializar(s_barbero, 0);
      cobegin
      barbero(); // Iniciar el proceso del
barbero
      // Levantar múltiples clientes
      for (int i = 0; i < 10; i++) {
         cliente();
      }
   coend;
}
```

En la programación en seudocódigo siguiente:

a) Identifique la sección crítica

b) Utilice semáforos para:

- **a) Exclusión mutua de la sección critica**
- **b) Limitar el número de clientes a 5, siendo el barbero quien habilita espacios cuando termine con un corte**
- **c) Sincronizar procesos para cuando llegue el primer cliente despierte al barbero y si no hay vuelva a dormir**

Sistemas Operativos y Distribuidos

Iren Lorenzo Fonseca
iren.fonseca@.ua.es

TEMA 1. Sistemas Operativos.

Ejercicios