UNIVERSIDAD DE BUENOS AIRES - FACULTAD DE INGENIERÍA
75.08 / 95.03 - SISTEMAS OPERATIVOS - CÁTEDRA MÉNDEZ
1er RECUPERATORIO

Nombre:	MEGRossi GIANLUCA				Nota:
Padron:	103 184	Cuatrimestre:	1/2024	Fecha: 19/06/2024	

GRILLA DE RESPUESTAS A PREGUNTAS

Responda las preguntas multiple choice utilizando la grilla de respuestas marcando con una X la respuesta correcta.
Las preguntas tienen una única respuesta correcta

Resolver los ejercicios en una hoja aparte.

Cada pregunta y ejercicio vale 1 punto. Para aprobar se necesitan por lo menos 2 bien de los ejercicios 8, 9 y 10

	1	2	3	4	5	6	7	8	9	10
A	X				X					
B			X			X	X			
C		X		X						
D										

---

1 - Pregunta: En el CFS, ¿cómo se determina el "vruntime" (tiempo de ejecución virtual) de una tarea y cuál es su importancia?

a) El "vruntime" se incrementa en función del tiempo de ejecución real de la tarea y de la carga del sistema; es importante porque ayuda a mantener la equidad en la asignación de CPU.
b) El "vruntime" se incrementa en función de la prioridad de la tarea; es importante porque determina la prioridad de la tarea en el sistema.
c) El "vruntime" se incrementa solo cuando la tarea está esperando; es importante porque reduce la latencia de las tareas en tiempo real.
d) El "vruntime" se mantiene constante para todas las tareas; es importante porque simplifica la programación.

2 - Pregunta: ¿Cuál es el propósito principal del Virtual File System (VFS) en Linux?

a) Proporcionar una interfaz para la comunicación entre dispositivos.
b) Permitir a los usuarios ejecutar múltiples aplicaciones en paralelo.
c) Abstraer las operaciones del sistema de archivos y permitir la coexistencia de múltiples sistemas de archivos.
d) Es una estructura lógica que organiza, gestiona y almacena archivos en dispositivos de almacenamiento.

3 - Pregunta: En el contexto de la gestión de memoria en sistemas operativos, ¿por qué los programadores generalmente utilizan malloc en lugar de brk directamente para la asignación de memoria dinámica?

a) brk es una función de bajo nivel que solo puede asignar memoria en bloques de tamaño fijo, mientras que malloc puede asignar bloques de cualquier tamaño.
b) malloc proporciona una abstracción más alta que incluye gestión de memoria, reutilización de bloques libres, y manejo de fragmentación, lo cual no es directamente manejado por brk.
c) brk solo puede ser utilizado por el kernel del sistema operativo y no está disponible para las aplicaciones de usuario.
d) malloc es más rápido que brk porque malloc no requiere llamadas al sistema.

4 - Pregunta: En un sistema concurrente, dos hilos están intentando actualizar un contador compartido sin utilizar ninguna forma de sincronización. ¿Cuál de los siguientes problemas describe mejor el fenómeno que puede ocurrir y cómo puede ser mitigado?

a) Interbloqueo (deadlock); puede ser mitigado utilizando un algoritmo de detección de interbloqueo.
b) Inanición (starvation); puede ser mitigado utilizando un planificador de hilos justo.
c) Condición de carrera (race condition); puede ser mitigado utilizando mecanismos de sincronización como mutexes o locks.
d) Fragmentación de memoria; puede ser mitigado utilizando memoria dinámica mediante malloc (en lugar de brk)

5 - Pregunta: En el capítulo 1 de "Operating Systems: Principles and Practice" de Anderson y Dahlin, los autores describen tres roles principales de los sistemas operativos: "referee", "ilusionista" y "pegamento". ¿Cuál de las siguientes opciones define correctamente estos roles?

a) Referee: Gestiona el acceso a recursos compartidos para evitar conflictos; Ilusionista: Crea la ilusión de recursos dedicados y abundantes; Pegamento: Facilita la comunicación y coordinación entre aplicaciones.
b) Referee: Asigna recursos de manera justa; Ilusionista: Mejora el rendimiento del sistema mediante optimizaciones; Pegamento: Gestiona la memoria virtual.
c) Referee: Proporciona mecanismos de seguridad y aislamiento; Ilusionista: Simplifica la interfaz de usuario; Pegamento: Administra la entrada/salida de datos.
d) Referee: Asegura que todas las aplicaciones se ejecuten en tiempo real; Ilusionista: Proporciona acceso directo al hardware; Pegamento: Coordina los procesos en tiempo compartido.

6 - Pregunta: En el contexto de la llamada al sistema fork en Unix/Linux, ¿cuál de las siguientes afirmaciones describe mejor lo que sucede cuando fork es ejecutado por un proceso?

a) fork crea un nuevo hilo dentro del proceso existente, compartiendo el mismo espacio de direcciones.
b) fork duplica el proceso llamante, creando un nuevo proceso hijo con un espacio de direcciones independiente pero idéntico al del proceso padre en el momento de la llamada.
c) fork crea un nuevo proceso hijo que comparte el mismo espacio de direcciones con el proceso padre, pero tiene su propio contador de programa y puntero de pila.
d) fork inicia un nuevo programa especificado por el proceso llamante, cargando un nuevo ejecutable en el espacio de direcciones del proceso hijo.

7 - Pregunta: El concepto de "Copy-On-Write" (COW) es utilizado en sistemas operativos para optimizar el uso de memoria. ¿Cuál de las siguientes afirmaciones describe mejor cómo funciona Copy-On-Write y sus beneficios?

a) COW crea inmediatamente copias duplicadas de los datos en memoria cuando se realiza un fork(), mejorando la velocidad de acceso.
b) COW posterga la creación de copias duplicadas de los datos en memoria hasta que uno de los procesos intenta modificar los datos, optimizando el uso de memoria.
c) COW permite a los procesos compartir los descriptores de archivos, reduciendo la sobrecarga de apertura y cierre de archivos.
d) COW desactiva la paginación de memoria para los procesos hijo, permitiendo un acceso más rápido a la memoria.

8 - **Ejercicio:** Partiendo de un filesystem vacío se ejecutan una serie de comandos en la shell en el orden indicado. Para los comandos en **negrita**, indicar la cantidad de accesos a **inodos** y accesos a **bloques de datos** que el sistema operativo debe realizar. **Desarrolle**.
Notas:

- Asuma que no existe ningún mecanismo de caché u optimización, y el sistema operativo siempre tiene que acceder a todos los inodos y bloques de datos necesarios en cada comando.
- Justifique los pasos que el sistema operativo necesita hacer para completar cada comando en negrita, y que resulten en la cantidad de accesos que previamente ha indicado.

`# mkdir /dir  /dir/s`
`# echo 'hola' > /dir/x`
`# echo 'mundo' > /dir/s/y`	

`# ls /dir/x` **`# cat /dir/s/y`**	**Inodos:** 3 **Blq. datos:** 3 **Inodos:** 4 **Blq. datos:** 4

`# ln /dir/x /dir/h`
`# ln -s /dir/s/y /dir/y`
`# rm /dir/x`	

`# cat /dir/h` **`# cat /dir/y`**	**Inodos:** 3 **Blq. datos:** 3 **Inodos:** 3 **Blq. datos:** 3

9 - **Ejercicio:** Considere un sistema x86 de memoria virtual paginada de dos niveles con un espacio de direcciones de 32 bits, donde cada página tiene un tamaño de 4096 bytes. Un entero ocupa 4 bytes y se tiene un array de 50,000 enteros que comienza en la dirección virtual **0x01FBD000**
El arreglo se recorre completamente, accediendo a cada elemento una vez. En este proceso, ¿a cuántas **páginas distintas** (no la cantidad total de accesos) necesita acceder el sistema operativo para conseguir esto? Recuerde contar las tablas de páginas intermedias, no solo las páginas que contienen los elementos del array. **Desarrolle.**

Cantidad de páginas distintas: 49

10 - **Ejercicio:** Sea un disco que posee 2049 bloques de 4kb y un sistema operativo cuyos i-nodos son de 256 bytes. Definir un sistema de archivos FFS. Explique las decisiones tomadas. **Desarrolle.**

RECUPERATORIO 19/06/2024

GRILLA DE RESPUESTAS A PREGUNTAS

	1	2	3	4	5	6	7	8	9	10
A	X				X					
B			X			X	X			
C		X		X						
D										

Ejercicio 8

`ls /dir/x`
Inodos: 2 accesos (3 también se toma como bien)
Datos: 2 accesos

`cat /dir/s/y`
Inodos: 4 accesos
Datos: 4 accesos

`cat /dir/h`
Inodos: 3 accesos
Datos: 3 accesos

`cat /dir/y`
Inodos: 7 accesos
Datos: 7 accesos

Ejercicio 9

Cantidad de paginas: 51

Ejercicio 10

1 superbloque
1 bitmap de inodos
1 bitmap de datos
129 bloques de inodos
1917 bloques de datos (2049 - 129 - 1 - 1 - 1)

Desarrollo

Ejercicio 8

`ls /dir/x`

1. Acceso a inodo '/
2. Leer el bloque de datos con el contenido de /, encuentra la entrada para ‘dir’
3. Acceso al inodo dir
4. Leer el bloque de datos con el contenido de dir, encuentra la entrada para ‘x’

No accede a los datos del archivo.

Inodos: 2 accesos (1, 3)
Datos: 2 accesos (2, 4)

Nota: Sin parámetros ls no muestra metadata del archivo. Como esto no estaba especificado en el enunciado, consideramos bien 3 accesos a inodos en este ejercicio si el alumno asume que se lee la metadata

`cat /dir/s/y`

1. Acceso a inodo /
2. Leer el bloque de datos con el contenido de /, encuentra la entrada para ‘dir’
3. Acceso al inodo ‘dir’
4. Leer el bloque de datos con el contenido de ‘dir’, encuentra la entrada para ‘s’
5. Acceso al inodo ‘s’
6. Leer el bloque de datos con el contenido de ‘s’, encuentra la entrada para ‘y’
7. Acceso al inodo ‘y’
8. Leer el contenido de ‘y’ y mostrarlo en pantalla

Inodos: 4 accesos (1, 3, 5, 7)
Datos: 4 accesos (2, 4, 6, 8)

`cat /dir/h`

/dir/h se creo como un hard link. Esto significa que si el archivo original se elimina, el hard link sigue funcionando normalmente.

1. Acceso a inodo /
2. Leer el bloque de datos con el contenido de /, encuentra la entrada para ‘dir’
3. Acceso al inodo ‘dir’
4. Leer el bloque de datos con el contenido de ‘dir’, encuentra la entrada para ‘h’
5. Acceso al inodo ‘h’

6. Leer el contenido de ‘h’ y mostrarlo en pantalla

Inodos: 3 accesos (1, 3, 5)
Datos: 3 accesos (2, 4, 6)

```
cat /dir/y
```

/dir/y es un soft link. Esto significa que es simplemente un archivo que contiene el path del archivo asociado. Esto implica que el sistema operativo tiene que leer el path del archivo, y el path contenido en el soft link, uno después del otro.

1. Acceso a inodo /
2. Leer el bloque de datos con el contenido de /, encuentra la entrada para ‘dir’
3. Acceso al inodo ‘dir’
4. Leer el bloque de datos con el contenido de ‘dir’, encuentra la entrada para ‘y’
5. Acceso al inodo ‘y’
6. Leer el contenido de ‘y’. Obtiene el path original: /dir/s/y
7. Acceso a inodo /
8. Leer el bloque de datos con el contenido de /, encuentra la entrada para ‘dir’
9. Acceso al inodo ‘dir’
10. Leer el bloque de datos con el contenido de ‘dir’, encuentra la entrada para ‘s’
11. Acceso al inodo ‘s’
12. Leer el bloque de datos con el contenido de ‘s’, encuentra la entrada para ‘y’
13. Acceso al inodo ‘y’
14. Leer el contenido de ‘y’ y mostrarlo en pantalla

Inodos: 7 accesos (1, 3, 5, 7, 9, 11, 13)
Datos: 7 accesos (2, 4, 6, 8, 10, 12, 14)

Ejercicio 9

Cantidad total de bytes que ocupa el array: 50,000 x 4 = 200,000

Posición del primer byte: 0x01FBD000 (definido en el enunciado)

Posición del último byte: 0x01FBD000 + (50000 x 4 - 1) = 0x01FEDD3F

Componentes de la dirección virtual del primer y último byte utilizados (separadas en grupos de 10, 10 y 12 según la paginación de x86):

- `0x01FBD000 = 0000000111 1110111101 000000000000`
- `0x01FEDD3F = 0000000111 1111101101 110100111111`

Los primeros dos grupos de 10 bits indican índices dentro de las tablas de páginas. Estos índices en decimal son:

Indice del page directory:

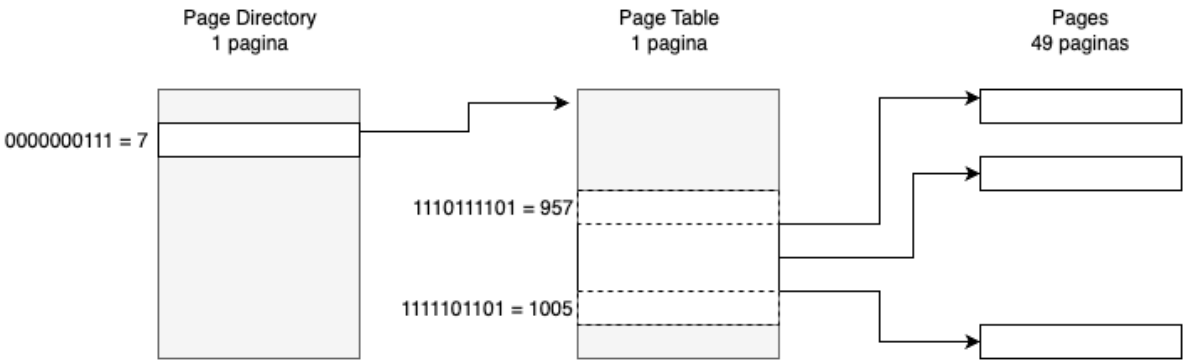
- `0000000111 = 7`

En el primer nivel (el directorio de páginas), se usa un solo registro, en el índice 7. Este contiene la dirección (y metadata) de la **única tabla de páginas** que será necesaria en el segundo nivel. Si se requirieran más tablas de páginas de segundo nivel, este índice cambiaría para algunas direcciones involucradas en el rango. Esto no ocurre para el rango propuesto en este ejercicio.

Índices de las page table

- `1110111101 = 957`
- `1111101101 = 1005`

En el segundo nivel, se utilizan varios registros, cada uno contiene la dirección de las páginas donde están los datos del array. Concretamente, se usan los registros desde el 957 hasta el 1005, es decir, 49 registros, lo que implica 49 páginas de datos.



En el tercer nivel, cada página contiene 4096 bytes de datos. Si queremos guardar 200,000 bytes entonces necesitamos 48.8 páginas, es decir, necesitamos utilizar 49 páginas. Esto es consistente con el cálculo que hicimos antes a partir de las direcciones de los extremos del rango de bytes del array.

Resultado: 1 page directory (por definición de la arquitectura x86) + 1 page table (justificado anteriormente) + 49 páginas de datos (justificado anteriormente) = **51 páginas en total** que el sistema operativo necesita utilizar para guardar y acceder a este array.