Departamento de Arquitectura

Examen de Sistemas Operativos

22 de diciembre de 2023

Lea detenidamente las siguientes instrucciones. No cumplir los requerimientos puede implicar la pérdida del examen.

- **Indique su nombre completo y número de cédula en cada hoja** (no se corregirán las hojas sin nombre). Numere todas las hojas e indique la cantidad total de hojas en la primera.
- **Escriba las hojas de un solo lado y empiece cada problema en una hoja nueva**.
- Si se entregan varias versiones de un problema, solo se corregirá la primera versión.
- Sólo se contestarán dudas de letra y no se aceptarán dudas en los últimos 30 minutos del examen.
- El examen es **SIN material**. En su banco solo puede tener las hojas del examen, lápiz, goma y lapicera. Todas sus demás pertenencias debe colocarlas en el piso debajo de su asiento.
- Para aprobar el examen es necesario un mínimo de **60 puntos**.
- El examen dura **3 horas**. Al momento de finalizar el examen **no se podrá escribir absolutamente nada en las hojas**, el estudiante debe dirigirse a la fila de entrega. Identificar cada una de las hojas con nombre, cédula y numeración debe realizarse dentro de la duración del examen.

Problema 1 (28 pts)

(a) Sobre de los requerimientos para la virtualización:

I. (2 pts) Según las hipótesis de Popek y Goldberg, ¿cuáles son las instrucciones sensibles?

> **Solución:** Instrucciones sensibles (Popek, Goldberg):
>
> 1. Operaciones E/S
> 2. Configuración de MMU
> 3. Administración de interrupciones

II. (3 pts) ¿Cuáles son los requerimientos adicionales para la virtualización (además de las instrucciones sensibles)?

> **Solución:**
>
> 1. Una máquina virtualizada debe comportarse igual que una máquina real
> 2. Debe tener dos modos: kernel y usuario
> 3. Las instrucciones sensibles deben ejecutarse en modo kernel
> 4. Instrucciones privilegiadas: disparan una trap si son ejecutadas en modo usuario
> 5. Para que un sistema pueda soportar virtualización las instrucciones sensibles deben ser un subconjunto de las instrucciones privilegiadas

(b) Sobre deadlocks:

I. (4 pts) Mencione y explique brevemente las condiciones necesarias para la existencia de deadlock.

II. (3 pts) Considere un sistema con doce unidades de un recurso y el siguiente estado:

	Máximo	Actual	Diferencia
P_0	10	5	5
P_1	4	2	2
P_2	9	3	6

Hay dos unidades libres del recurso, explique si el sistema se encuentra en un estado seguro y si es posible determinar si un deadlock ocurrirá o no.

Solución: i. Las condiciones necesarias para la existencia de deadlock (condiciones de Coffman) son:

1. Exclusión mutua: al menos un recurso debe ser de uso exclusivo (no compartible).
2. Retención y espera: un proceso debe mantener un recurso mientras espera para obtener otro.
3. No expropiable: los recursos no deben ser expropiables (se liberan voluntariamente).
4. Espera circular: dados los procesos $\{P_0, P_1, ..., P_n\}$, P_0 debe esperar por un recurso obtenido por P_1, P_1 por uno de P_2, ..., P_n por uno de P_0.

Las cuatro condiciones son necesarias individualmente y suficientes cuando se tiene una sola instancia de cada recurso

ii. El sistema descrito se encuentra en un estado inseguro. El proceso P1 podría completarse, liberando así un total de cuatro recursos, pero no es posible garantizar que los procesos P0 y P2 puedan completarse. Sin embargo, es posible que un proceso pueda liberar recursos antes de solicitar más recursos. Por ejemplo, el proceso P2 podría liberar un recurso, aumentando así el número total de recursos a cinco. Esto permite que el proceso P0 complete su ejecución, lo que liberaría un total de nueve recursos, permitiendo así que el proceso P2 también se complete. Por lo tanto, no es posible determinar si un deadlock ocurrirá o no (depende del orden de ejecución de los procesos).

(c) Sobre el manejo de memoria:

I. (3 pts) Explique el soporte a nivel de hardware que proveen los sistemas operativos para almacenar en memoria caché la tabla de páginas y explique cómo se mejora el rendimiento.

II. (5 pts) Defina la métrica utilizada para evaluar el tiempo de acceso en un sistema que usa TLB y calcule la ganancia promedio para un hit ratio del 70 %, si el tiempo de acceso a memoria es de 150 ns. el acceso a la tabla de páginas demanda un único acceso a memoria y la búsqueda en la TLB demanda 10 ns.

Solución: El soporte a nivel de hardware consiste en utilizar una caché asociativa y de rápido acceso (Translation Look-aside Buffer, TLB). Cada entrada en la TLB tiene una clave (tag) y un valor (el número de frame). La búsqueda de una clave *número de página) en la TLB es simultánea en todos los tags. Si se encuentra la clave (TLB hit), inmediatamente se genera la dirección buscada a partir del valor asociado. En caso contrario (TLB miss), se debe acceder a memoria para obtener el número de frame. Posteriormente, se guarda el valor obtenido en la TLB para posteriores accesos. El tiempo efectivo de acceso se define por EAT = tiempo de búsqueda en TLB + (hit ratio $\times$ tiempo de acceso a memoria) + (1 - hitratio) $\times$ (2 $\times$ tiempo de acceso a memoria).
El hit ratio indica el porcentaje de veces que el número de página se encuentra en la TLB. El tiempo de acceso a memoria de 150 ns implica que se necesitan 150 ns cuando el número de página está en la TLB (hit) y 300 ns (acceder a la memoria para obtener el número de frame y acceder a la memoria para obtener el dato) cuando el número de página no está en la TLB

(miss) EAT = 10 + 0.70 × 150 + 0.30 × 300 = 205 ns. La ganancia promedio es 1 - (205 ns / 300 ns) = 0.31666.

(d) Sobre los métodos de asignación en file systems:

I. (3 pts) Explique cuál es el principal problema del método de asignación en forma de lista en un file system y cómo lo resuelve el método de asignación indexada.

II. (5 pts) Considere un archivo que tiene 150 bloques apuntados por el mismo bloque de índice. El bloque de índice ya está en memoria y no debe escribirse a disco. ¿Cuántas operaciones de E/S (al disco) se requieren para escribir un nuevo bloque en el archivo al utilizar una estrategia de asignación indexada mononivel (asuma que hay espacio en el bloque de índice para el puntero al nuevo bloque)? ¿Y para borrar un bloque?

Solución: La asignación indexada resuelve el problema que tiene la asignación en forma de lista de que no soporta un acceso directo eficiente, dado que los punteros a los bloques están dispersos con los propios bloques por todo el disco y deben recuperarse en orden. La asignación indexada resuelve este problema al reunir todos los punteros en una ubicación: el bloque de índice.

En la estrategia indexada solo debe actualizarse el índice en memoria y escribir en nuevo bloque. Se debe buscar un lugar en la lista de libres (un read y un write). Para escribir se requiere una operación de E/S (write). Total: tres operaciones de E/S al disco.

Para borrar un bloque no se necesitan accesos al disco, simplemente se elimina la dirección del bloque de la lista indexada en el bloque de índice (en memoria). Se debe buscar un lugar en la lista de libres (un read y un write). Total: dos operaciones de E/S al disco.

Problema 2 (32 pts)

Considere un sistema operativo multiprogramado operando un único procesador, que aplica un modelo de hilos M×1. El planificador es expropiativo y trabaja con una planificación por niveles con retroalimentación. En todas las colas se utiliza una planificación Round Robin con un cuanto de 2 unidades de tiempo. Para desempatar se toma el proceso con menor índice.

Cuando un proceso ingresa en la cola de listos, es asignado a la cola de nivel 0 (mayor prioridad). Las reglas para el cambio de prioridad son: i) si un proceso en el nivel 0 ejecuta durante 2 unidades de tiempo seguidas, baja a la cola de nivel 1 (su prioridad baja a la del nivel 1); ii) si un proceso en el nivel 1 pasa 3 unidades de tiempo sin utilizar el recurso procesador, sube a la cola de nivel 0.

En ningún momento puede ejecutar un proceso de prioridad 1 si existe un proceso en la cola de nivel 0. A nivel de hilos se utiliza una planificación SJF. Para desempatar se considera primero los hilos A y luego los hilos B.

Considere dos procesos P1 y P2, cada uno con dos hilos (P1A, P1B, P2A y P2B), los cuales ejecutan copias idénticas de la siguiente descripción: i) P1: ejecuta 4u, se bloquea 1u, ejecuta 2u; ii) P2: ejecuta 1u, se bloquea 2u, ejecuta 3u. Se asume que la operación de bloqueo lleva un tiempo despreciable (se interpreta como que la última instrucción del bloque de ejecución se encarga de bloquear).

Inicialmente, el sistema no tiene ningún proceso de usuario ejecutando. En t=0 se lanza la ejecución de P1 y en t=3 se lanza la ejecución de P2.

Para la situación descrita:

(a) (27 pts) Realice los diagramas de planificación (tiempo vs. procesos y tiempo vs. hilos), comenzando en el tiempo t=0, indicando el estado de cada uno de los procesos e hilos y su posición en cada una de las colas. Utilice la notación $\text{Estado}_{\text{prioridad}}$ (un proceso en la cola de listos con prioridad 1 L_1, un proceso ejecutando con prioridad 0 E_0, etc.)

(b) (2 pts) Defina tiempo de espera y tiempo de retorno.

(c) (3 pts) Calcule el tiempo de espera y el tiempo de retorno de cada proceso.

Solución:

(a) Los diagramas de planificación (tiempo vs. procesos y tiempo vs. hilos) son:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P1	E0	E0	E1	L1	E1	B	E0	E0	L1	E1	E1	L1	L1	E1	E1	B	E0	E0	T	T	T
P2	-	-	-	E0	B	B	L0	L0	E0	B	B	E0	E0	L1	L1	E1	L1	L1	E1	E1	E1
P1A	E	E	E		E	B	E	E	T	T	T	T	T	T	T	T	T	T	T	T	T
P1B	L	L	L		L	B	L	L		E	E			E	E	B	E	E	T	T	T
P2A	-	-	-	E	B	B			L	B	B	E	E			E	T	T	T	T	T
P2B	-	-	-	L	B	B			E	B	B	L	L			L			E	E	E

En $t = 0$ inicia el proceso P1, con la ejecución de su thread P1A (por desempate)
En $t = 3$ inicia el proceso P2. P1 pasa a estado Listo porque su prioridad es menor que la de P2. P2 toma la CPU para ejecutar su thread P2A (por desempate)
En $t = 6$ el proceso P1 reingresa a la cola de procesos listos y se le asigna la prioridad 0 (como indica la letra).
En $t = 8$ finaliza el thread P1A.
En $t = 8$ el thread P2B ejecuta porque la política de planificación de hilos en SJF y el thread P2B tiene CPU burst 1 u, menor que el CPU burst de P2A, que es 3 u.
En $t = 11$ ejecuta el thread P2A por desempate (ambos threads P2A y P2B tienen CPU burst de 3 u).
En $t = 13$ se termina el cuanto del proceso P2, desciende os a la cola de prioridad 1 y cede la CPU al proceso P1, de acuerdo con la política de Round Robin.
En $t = 16$ finaliza el thread P2A.
En $t = 6$ el proceso P1 reingresa a la cola de procesos listos y se le asigna la prioridad 0.
En $t = 18$ finaliza el thread P1B y el proceso P1.
En $t = 20$ finaliza el thread P2B y el proceso P2.

(b) El tiempo de espera se define como la suma de los intervalos de tiempo que un proceso estuvo en la cola de procesos listos.

El tiempo de retorno se define como el intervalo de tiempo desde que un proceso es cargado hasta que finaliza su ejecución.

(c) De acuerdo con el diagrama de ejecución presentado, el tiempo de espera para el proceso P1 es de 4 u. El tiempo de espera para el proceso P2 es de 6 u.

De acuerdo con el diagrama de ejecución presentado, el tiempo de retorno para el proceso P1 es de (18−0) 18 u. El tiempo de retorno para el proceso P2 es de (21−3) 18 u.

Problema 3 (40 pts)

Se desea modelar una tienda de venta de comida y bebida en un estadio. En la tienda se venden bebidas y hamburguesas. Las bebidas se encuentran en una heladera y las hamburguesas, ya prontas, se encuentran en un horno para que no se enfríen. En la tienda trabajan cuatro vendedores. El acceso a la heladera y al horno solo puede hacerse por un vendedor a la vez.

Para poder comprar, los clientes forman una cola frente a la única caja de la tienda. Por razones de espacio no puede haber más de 20 clientes en la cola. Si no hay lugar, los clientes se retiran sin esperar.

Los vendedores toman el pedido y lo cobran en la caja. Luego retiran el pedido y se lo entregan al cliente. Se supone que cada cliente ordena solo uno de los dos productos. Los vendedores no pueden dejar la caja sola, por lo que para poder ir a retirar el pedido debe haber otro vendedor en el área de la caja. Por razones de espacio no puede haber más de dos vendedores en el área de caja a la vez. Los vendedores solo toman pedidos cuando están en la caja.

Modele la realidad planteada usando mailboxes a los clientes y los vendedores. No se permite usar procesos auxiliares. Se dispone de los siguientes procedimientos auxiliares:

que_quiero(): [bebida, hamburguesa]
Ejecutada por los clientes para saber que producto desean consumir

obtener_producto()
Ejecutada por los vendedores para obtener el producto pedido por el cliente (luego de obtener el acceso exclusivo al recurso correspondiente)

entregar_producto()
Ejecutada por los vendedores para entregar el producto al cliente correspondiente

cobrar()
Ejecutada por los vendedores para cobrar al cliente

Solución:

```
var cola: mailbox of integer
    largo_cola: mailbox of integer
    espera_producto: array [1..20] of mailbox of nil
    caja: mailbox of (integer, integer)
    acceso_caja: mailbox of nil
    espera_salir: mailbox of boolean
    en_caja: mailbox of integer
    salir_caja: mailbox of nil
    heladera: mailbox of nil
    horno: mailbox of nil

procedure cliente
var p: producto
    id, esperando: integer
begin
    esperando := receive(largo_cola);
    if esperando < 20 then
        send(largo_cola, esperando + 1);

        id = receive(cola);

        esperando := receive(largo_cola);
        send(largo_cola, esperando - 1);

        p := que_quiero();
        send(caja, (p, id));
        receive(espera_producto[id], nil);
        send(cola, id);
    else
        send(largo_cola, esperando);
    end
end cliente

procedure empleado
var id, adentro: integer
    pedido: producto
    salir: boolean
begin
    while true do
        receive(acceso_caja);

        adentro := receive(en_caja);
```

```
            send(en_caja, adentro + 1);

            salir := receive(espera_salir);
            if salir then
                send(salir_caja);
            end
            send(espera_salir, false);

            (pedido, id) := receive(caja);
            cobrar();

            adentro := receive(en_caja);
            if adentro > 1 then { me puedo ir }
                send(acceso_caja);
                send(en_caja, adentro - 1);
            else { tengo que esperar }
                salir := receive(espera_salir);
                send(espera_salir, true);
                send(en_caja, adentro);

                receive(salir_caja);

                adentro := receive(en_caja);
                send(en_caja, adentro - 1);
                send(acceso_caja);
            end

            if pedido = bebida then
                receive(heladera);
                obtener_producto();
                send(heladera, nil);
            else
                receive(horno);
                obtener_producto();
                send(horno, nil);
            end

            entregar_producto();
            send(espera_producto[id], nil);
        end while
    end empleado

    begin
        for i := 1 to 20 do
            send(cola, i);
        end
        for i := 1 to 2 do
            send(acceso_caja, nil);
        end
        send(heladera, nil);
        send(horno, nil);
        send(largo_cola, 0);
        send(espera_salir, false);
        send(en_caja, 0);

        cobegin
```

```
            empleado();
            empleado();
            empleado();
            empleado();
            cliente();
            ...
            cliente();
        coend
    end
```