Departamento de Arquitectura

Super parcial de Sistemas Operativos

2 de Diciembre de 2021

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

Formato:

- 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 de ellas.

Dudas:

- Sólo se contestarán dudas de letra.
- No se aceptarán dudas en los últimos 15 minutos del parcial.

Material:

- El parcial es **SIN** material (no puede utilizarse ningún apunte, libro ni calculadora). Sólo puede tenerse las hojas del parcial, lápiz, goma y lapicera en su banco. Todas sus demás pertenencias debe colocarlas en el piso debajo de su asiento.

Finalización:

- El parcial dura **3 horas**.
- Al momento de finalizar el parcial no se podrá escribir absolutamente nada en las hojas, debiéndose parar e ir a la fila de entrega. Identificar cada una de las hojas con nombre, cédula y numeración forma parte de la duración del parcial.

Importante: Debe justificar todas las respuestas. Siempre.

Problema 1 (10 pts)

(a) (2 pts) Describa brevemente de qué forma se puede garantizar que todos los accesos a los dispositivos, por parte de un proceso de usuario, se deban hacer a través del sistema operativo.

(b) (2 pts) Explique cómo funciona el método de asignación indexada en un sistema de archivos.

(c) (2 pts) Describa los campos principales de la estructura de datos para representar un archivo en una FAT.

(d) (2 pts) Explique brevemente cómo se puede implementar memoria compartida en un sistema que utiliza paginación pura.

(e) (2 pts) ¿Por qué en el PCB (Process Control Block) se reserva lugar para los registros de la CPU?

Problema 2 (22 pts) Se tiene un sistema con dos procesadores sobre el que ejecuta un sistema operativo con un planificador SJF expropiativo y un modelo de hilos 1×1. Inicialmente en el sistema no se tiene ningún proceso de usuario ejecutando y en tiempo $t = 0$ se lanza la ejecución del siguiente programa de usuario:

```
begin
  fork()
  pid = fork()
  if(pid == 0){
    Ejecuta 15ms
    Bloquea 5ms
    Ejecuta 5ms
  } else {
    create_thread(proc1)
  }
end
```

```
procedure proc1()
begin
  Ejecuta 5ms
  Bloquea 5ms
  Ejecuta 5ms
end
```

Se sabe que la operación fork() es estilo Unix y al igual que en Unix, luego de un fork() el proceso hijo siempre tiene un único hilo de ejecución. La operación create_thread() crea un nuevo hilo de ejecución y comienza su ejecución en el procedimiento recibido como argumento. Salvo que se indique explícitamente, el tiempo de ejecución de todas las funciones es de $5ms$ (fork, create_thread, condición del if, asignación, etc). La planificación definida para los hilos a nivel de usuario es FCFS.

Se pide:

(a) (3 pts) Explique el modelo de hilos Mx1.

(b) (16 pts) Realice un diagrama de planificación (tiempo vs procesos/hilos), comenzando en el tiempo $t = 0$, indicando el estado de cada uno de los hilos y su posición en cada una de las colas.

(c) (3 pts) ¿Cuál es el proceso con menor tiempo de espera? ¿Cuál es el proceso con mayor tiempo de retorno? Justifique su respuesta.

Solución:

(a) Ver teórico (Procesos diapositiva 35)

(b) Diagrama:

	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75
P1H1	E25	E20	E15	E10	E5	T									
P1H2	-	-	-	-	-	E5	B	E5	T						
P2H1	-	E20	E15	E10	E5	T									
P2H2	-	-	-	-	-	E5	B	E5	T						
P3	-	-	L25	L25	L25	L25	E25	L20	E20	E15	E10	E5	B	E5	T
P4	-	-	L25	L25	L25	L25	E25	L20	E20	E15	E10	E5	B	E5	T

(c) Los procesos con menor tiempo de espera son $P1$ y $P2$ cuyo tiempo de espera es 0. Los procesos con mayor tiempo de retorno son $P3$ y $P4$.

Problema 3 (20 pts) Se tiene un sistema de memoria con paginación y una TLB (Transaltion Look-aside Buffer). La tabla de páginas cuenta con 16 entradas. La TLB cuenta con 6 entradas y utiliza un algoritmo de reemplazo FIFO. El tiempo de acceso a la TLB es de $0{,}001$ segundos y el tiempo de acceso a la memoria principal es de $0{,}1$ segundos. Al momento de ejecutar un proceso, su tabla de páginas y la TLB estan cargadas como se muestra en la Figura 1.

Suponga que las entradas de la TLB en la imagen estan ordenadas por antigüedad. Es decir, la entrada para la página 4 es la mas antigua y la entrada para la página 6 es la mas reciente.

Se pide:

(a) (3 pts) Mencione cuales son las ventajas de utilizar la TLB.

Tabla de Paginas

	Frame	Valido/ Invalido
0	505	V
1	506	V
2	507	V
3	403	V
4	404	V
5	128	V
6	423	V
7	424	V
8	425	V
9	426	V
10	522	V
11	523	V
12	0	I
13	0	I
14	0	I
15	0	I

TLB

Pagina	Frame
4	404
9	426
0	505
7	424
5	128
6	423

Figura 1: Contenido de la tabla de páginas y la TLB

(b) (4 pts) Se realiza la siguiente secuencia de acceso a páginas: 7, 8, 9, 0, 1. De un estimativo del tiempo de acceso total para dicha secuencia y el estado resultante de la TLB.

(c) (5 pts) ¿Que ocurre si el proceso intenta acceder a la página 14? Describa la secuencia de pasos que debe realizar el sistema.

(d) (4 pts) Se observa que en la tabla de páginas, algunas secuencias contiguas de páginas se corresponden con secuencias contiguas de frames en memoria. A su vez, se sabe que los accesos regularmente se dan entre secuencias contiguas de páginas (por ejemplo: 3, 4, 10, 11). Explique como modificaría el algoritmo de reemplazo de la TLB para mejorar el hit-rate en la realidad planteada.

(e) (4 pts) Volviendo al algoritmo de reemplazo FIFO original. Suponga que los procesos de su sistema realizan una enorme cantidad de accesos a las páginas que almacenan el código y unos pocos accesos a las páginas que almacenan el stack y la memoria dinámica. Tambien suponga que el código usualmente se almacena entre las páginas 0 y 4. Explique como modificaría el algoritmo de reemplazo de la TLB para mejorar el hit-rate en la realidad planteada.

Solución:

(a) El cache TLB es asociativa y de rápido acceso. La búsqueda de un clave en la cache TLB es simultánea entre todas las tags. Si la clave es encontrada (TLB hit), inmediatamente se genera la dirección buscada a partir del valor asociado. Soluciona el problema del acceso a memoria duplicado debido a la necesidad primero acceder a la tabla para obtener el número de frame y, posteriormente, al lugar de memoria solicitado.

(b) El tiempo total se calcula como

$$T = T_{cache} * cant_{accesos} + 1 * T_{memoria} * cant_{hits} + 2 * T_{memoria} * cant_{miss} \quad (1)$$

En nuestro caso particular es:

$$T = 0{,}001 * 5 + 1 * 0{,}1 * 3 + 2 * 0{,}1 * 2 = 0{,}705 segundos \quad (2)$$

La TLB queda con el siguiente orden (de más antigua a más reciente): 0-7-5-6-8-1.

(c) Como el bit de validez está apagado se genera el trap de fallo de página. Rutina de atención del fallo de página:

1- Se busca un frame libre en memoria principal, sino hay ejecuta un algoritmo de reemplazo.

2- Se realiza un operación de lectura sobre el disco que guarda la página. La página se carga en el frame obtenido en el paso anterior.

3- Se actualiza la tabla de página e información en las estructuras del PCB del proceso para marcar que la página está disponible en memoria principal.

4- Finalmente, se retoma el control a la instrucción que fue interrumpida debido al fallo de página.

(d) Dado que cuando se accede a una página es probable que posteriormente se quiera acceder a la/s siguiente/s es posible optimizar el algoritmo haciendo que cuando se guarda una traducción en la TLB también se guarde la de la página siguiente.

(e) Dado que la cantidad de páginas accedidas son menores que la capacidad de la TLB se pueden dejar fijas en esta caché las traducciones de las cinco primeras páginas y únicamente modificar la entrada restante.

Otra alternativa es utilizar un algoritmo de reemplazo LRU.

Problema 4 (33 pts) Se desea modelar un sistema de ordeño automático. El mismo tiene capacidad para ordeñar 6 vacas en forma simultánea. Los animales van accediendo al sistema de a uno por orden de llegada. También se cuenta con dos empleados de mantenimiento que deben limpiar cada puesto luego de ser usado y antes de que ingrese otra vaca. Si ocurre que un puesto de ordeño queda libre por más de 5 minutos (porque no llegan animales o porque los empleados de mantenimiento no llegaron a limpiarlo) se debe activar una alarma.

Se pide:

(a) (33 pts) Implementar en ADA los procesos puesto, vaca y empleado. Se pueden usar tareas auxiliares.

Se dispone de los siguientes funciones auxiliares:

- `ordeñar(puesto:integer)` → Ejecutada por el puesto cuando la vaca llega para comenzar el ordeñe.
- `limpiar(puesto:integer)` → Ejecutada por el empleado para limpiar el puesto.
- `alarma(puesto:integer)` → Ejecutado por el puesto para indicar que ha estado 5 minutos sin usarse.

Solución:

```
task type vaca is
end vaca;

task body vaca is
var puesto: Integer;
begin
    admin.fila(puesto);
    puestos[puesto].entrar_vaca;
end vaca;


task type empleado:

task body empleado is
var puesto: Integer;
begin
    loop
        admin.dar_puesto_empleado(puesto);
        puestos[puesto].entrar_empleado;
        limpiar(puesto);
        puestos[puesto].salir_empleado;
        admin.puesto_limpio(puesto);
    end loop;
end empleado;


task type puesto is
    entry init(i: IN Integer);
    entry entrar_vaca;
    entry entrar_empleado;
    entry salir_empleado;
end puesto;

task body puesto is
var id: Integer;
begin
    accept init(i) do
        id := i;
    end init;
    loop
        select
            accept entrar_vaca
                ordeñar(id);
            end accept
            admin.requiere_limpieza(id);
        or
            accept entrar_empleado;
            accept salir_empleado;
        or
            delay (5*60)
```

```
            alarma(id);
        end select
    end loop
end puesto;


task admin is
    entry fila(p: OUT Integer);
    entry requiere_limpieza(p: IN Integer);
    entry dar_puesto_empleado(p: OUT Integer);
    entry puesto_limpio(p: IN Integer);
end admin;

task body admin is
var
    puesto_libre: Integer;
    puestos_libres: Array (1..6) of Boolean;
    puesto_sucio: Integer;
    puestos_sucios: Array (1..6) of Boolean;

begin
    puesto_libre := -1;
    puesto_sucio := -1;
    for i := 1 to 6 do
        puestos[i].init(i);
        puestos_libres[i] := True;
        puestos_sucios[i] := False;
    end for;

    loop
        if puesto_libre == -1 then
            for i := 1  to 6 do
                if puestos_libres[i] then
                    puesto_libre := i;
                    break;
                end if
            end for;
        end if

        if puesto_sucio == -1 then
            for i := 1  to 6 do
                if puestos_sucios[i] then
                    puesto_sucio := i;
                    break;
                end if
            end for;
        end if

        select
            when puesto_libre > 0 =>
```

```
                accept fila(p: Integer)
                    p := puesto_libre;
                end fila
                puestos_libres[puesto_libre] := False;
                puesto_libre := -1;
            or when puesto_sucio > 0 =>
                accept dar_puesto_empleado(p: Integer)
                    p := puesto_sucio;
                end dar_puesto_empleado;
                puestos_sucios[puesto_sucio] := False;
                puesto_sucio := -1;
            or
                accept requiere_limpieza(p: Integer)
                    puestos_sucios[p] := True;
                end requiere_limpieza;
            or
                accept puesto_limpio(p: Integer)
                    puestos_libres[p] := True;
                end cont_libres;
        end select;
    end loop;
end admin;

puestos: Array (1..6) of puesto;
empleados: Array (1..2) of empleado;
```