Departamento de Arquitectura

Segundo parcial de Sistemas Operativos

28 de junio de 2019

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

Formato:

- Indique su nombre completo y su 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 todas las hojas.
- Escriba las hojas de un solo lado y empiece cada problema en una hoja nueva.

Dudas:

- Sólo se contestarán dudas de letra.
- No se contestarán 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 15 minutos**.
- Al momento de finalizar el parcial no se podrá escribir absolutamente nada en las hojas. El estudiante debe ponerse de pie e ir a la fila de entrega. Se debe identificar cada una de las hojas con nombre, cédula y numeración antes de la hora de finalización del parcial.

Importante: Debe justificar todas las respuestas, siempre.

Problema 1 (10 puntos)

(a) (2 puntos) Mencione cuáles son las tres estrategias que utilizan los SO modernos para enfrentarse a un deadlock.

(b) (3 puntos) Sobre la disposición de archivos en un dispositivo de almacenamiento secundario:

i. Describa brevemente los tres métodos de asignación vistos en el curso.

ii. Muestre las estructuras de datos vistas en el curso para su implementación.

(c) (2 puntos) ¿Qué es el sistema de archivos virtual? Describa brevemente sus dos funcionalidades más importantes.

(d) (2 puntos) Describa brevemente la estructura RAID 5 e indique los pasos requeridos para realizar la escritura de 1 bloque en RAID 5.

(e) (1 punto) Enumere 3 tipos de ataques a la seguridad a nivel de aplicación en el contexto de un SO y explique brevemente mecanismos de mitigación.

Problema 2 (25 puntos)

Se tiene un sistema operativo que utiliza un gestor de memoria implementada con memoria virtual utilizando un modelo de paginación bajo demanda con las siguientes características:

- Estructura jerárquica de dos niveles de paginación.
- Tablas de páginas, por proceso, de 256 bytes.
- Cada tabla de página ocupa una página completa.
- Entradas de las tablas de páginas de 16 bytes.

(a) (4 puntos) Determine cuántos bits tienen las direcciones virtuales y especifique cómo se destribuyen estos bits en la dirección virtual, indicando para que se utilizan.

Solución: Como las tablas tienen 256 bytes, se precisan 8 bits para direccionarlas, y por tanto el desplazamiento de las direcciones virtuales es de 8 bits. Como las tablas ocupan 1 página (256 bytes), y cada entrada es de 16 bytes, entonces hay $256/16 = 16$ entradas por tabla. Se precisan 4 bits para direccionar cada nivel. Por lo tanto, las direcciones son de 16 bits y se interpretan como: | 4 bits - 1er nivel | 4 bits - 2ndo nivel | 8 bits - offset |

(b) (3 puntos) Muestre un diagrama explicando como se realiza la traducción de una dirección y presente un ejemplo concreto.

(c) (2 puntos) Indique el tamaño máximo de memoria virtual direccionable por cada proceso.

Solución: Cada proceso puede direccionar 2^{16} bytes = 64KB de memoria virtual.

(d) (6 puntos) Se tiene un proceso que requiere 1050 bytes para almacenar su código, mientras que para su pila requiere 258 bytes de espacio de memoria. Asumiendo espacio contiguo de almacenamiento para ambos espacios, determine la cantidad de tablas requeridas por el proceso.

Solución: Las páginas son de 256 bytes, entones el proceso requiere 5 páginas para almacenar su código, y 2 páginas para almacenar su stack. Como cada página de segundo nivel direcciona 16 páginas, bastaría con una entrada de segundo nivel si las secciones estuvieran ubicados uno a continuación del otro desde la dirección 0. Típicamente el stack se ubica a partir de las direcciones más altas de memoria virtual, en este caso, 0xFFFF, y el código desde las más bajas, es decir, desde la dirección 0. Por esa razón, el código utilizará la primer entrada de la tabla de páginas de primer nivel, mientras que el stack utilizará la última. Esto implica que las dos secciones utilizan tablas de página de segundo nivel diferentes. Por esta razón, el proceso utiliza 3 tablas de páginas: 1 de primer nivel y 2 de segundo nivel.

(e) (3 puntos) Si este sistema posee solamente 128 KB de memoria física y considerando la respuesta de la parte c, ¿puede este sistema ejecutar 4 procesos simultáneamente?

Solución: Si cada proceso direcciona hasta 64KB y la memoria física disponible es de 128KB, entonces los 4 procesos direccionarán más memoria que la memoria física disponible. Para que el Sistema Operativo pueda direccionar más memoria que la memoria física, se deberá contar con un área de swapping en memoria secundaria de al menos 128KB. En caso de que esta área exista, la respuesta es que sí, el sistema podrá soportar los 4 procesos.

(f) (4 puntos) Describa las técnicas de reemplazo NRU (Not Recently Used) y Óptima, especifique las fortalezas y la utilidad de cada una de ellas.

(g) (3 puntos) Compare las técnicas de reemplazo globales y las locales.

Problema 3 (27 puntos)

Se desea modelar usando ADA una fábrica de automóviles. La misma cuenta con una línea de producción con 25 estaciones de trabajo. En cada estación pueden trabajar hasta 3 empleados simultáneamente.

Los empleados trabajan en turnos de 60 minutos, luego descansan 10 minutos y vuelven a trabajar. Pueden trabajar en cualquier estación pero deberán darle prioridad a las estaciones con menos empleados cuando terminan su descanso. Si alguna de las estaciones está mas de 5 minutos sin al menos una persona se debe activar una alarma.

Hay un supervisor que verifica el estado de la diferentes estaciones de trabajo. Debe tener acceso exclusivo sobre la estación para poder supervisar y tiene prioridad sobre los empleados. El supervisor cuenta para no hacer sonar la alarma.

Se desean modelar en ADA las estaciones de trabajo, el supervisor y los empleados. Se pueden usar tareas auxiliares.

Se dispone de las siguientes funciones:

1. `trabajar()` ejecutada por el empleado para trabajar 60 minutos.
2. `descanso()` ejecutada por el empleado para descansar 10 minutos.
3. `alarma(estación: integer)` ejecutada por la estación de trabajo cuando no hay empleados por más de 5 minutos.
4. `elegir_estacion():integer` ejecutada por el supervisor para elegir la estación a supervisar.
5. `supervisar()` ejecutada por el supervisor para hacer su trabajo.

Solución:

```
procedure PROBLEMA_3 is
  CANT_ESTACIONES := 25;

  task type EMPLEADO is
  end EMPLEADO

  task body EMPLEADO is
  begin
    while true loop
      int estacionTrabajar := estacion_manager.solicitarAccesoEmpleado();
      estaciones[estacionTrabajar].solicitarAccesoTrabajo();
      trabajar();
      estaciones[estacionTrabajar].solicitarFinTrabajo();
      estacion_manager.finalizarAccesoEmpleado(estacionTrabajar);
      descansar();
    end loop
  end EMPLEADO

  -- SECCION CRITICA DE ESTACIONES
  task ESTACION_MANAGER is
  begin
    entry solicitarAccesoSupervisor(idEstacion: integer);
    entry finalizarAccesoSupervisor(idEstacion: integer);
    entry solicitarEstacionEmpleado(out idEstacion: integer);
    entry finalizarAccesoEmpleado(idEstacion: integer);
  end ESTACION_MANAGER

  task body ESTACION_MANAGER is
    cantEmpleados: array(1..CANT_ESTACIONES) of integer;
    haySupervisor: array(1..CANT_ESTACIONES) of boolean;
    id: integer;
  begin
    function getEstacionLibre()
    begin
      int minEmpleados := 3;
      int estacionLibre := -1;
```

```
      -- busco la estacion sin supervision con
      -- espacio libre y menor numero de empleados
      for i in 1..CANT_ESTACIONES loop
        if cantEmpleados[i] < minEmpleados and not haySupervisor[i] then
        begin
          minEmpleados := cantEmpleados[i];
          estacionLibre := i;
        end
      end loop
      return estacionLibre;
    end

    for i in 1..CANT_ESTACIONES loop
      cantEmpleados[i] := 0;
      haySupervisor[i] := false;
      estaciones[i].getId(i);
    end loop

    while true loop
      est_libre := getEstacionLibre();
      select
        accept solicitarAccesoSupervisor(idEstacion) do
            id := idEstacion;
        end
        haySupervisor[id] := true;
      or
        accept finalizarAccesoSupervisor(idEstacion) do
            id := idEstacion;
        end
        haySupervisor[id] := false;
      or when est_libre <> -1 =>
        accept solicitarEstacionEmpleado(out idEstacion) do
          idEstacion := est_libre;
        end
        cantEmpleados[est_libre]++;
      or
        accept finalizarAccesoEmpleado(idEstacion) do
            id := idEstacion;
        end
        cantEmpleados[id]--;
      end select
    end loop
  end ESTACION_MANAGER

  task type ESTACION is
  begin
    entry solicitarAccesoSupervisor();
    entry finalizarSupervision();
    entry solicitarAccesoTrabajo();
    entry solicitarFinTrabajo();
```

```
  entry getId(i:integer);
end ESTACION

task body ESTACION is
  id: integer
  empleados: integer
  haySupervisor: boolean
begin
  empleados := 0;
  haySupervisor := false;

  accept getId(i) do
    id := i;
  end

  while true loop
    if not haySupervisor and empleados = 0 then
      select
        accept solicitarAccesoSupervisor;
        haySupervisor := true;
      or
        -- acepto empleados cuando no hay supervisor esperando
        when solicitarAccesoSupervisor'count = 0 =>
          accept solicitarAccesoTrabajo();
          empleados++;
      or
        delay 300
          alarma(id);
      end SELECT
    else
      select
        when haySupervisor =>
          accept finalizarSupervision();
          haySupervisor := false;
      or
        -- acepto empleados cuando no hay supervisor trabajando
        -- o esperando y si hay capacidad
        when solicitarAccesoSupervisor'count = 0 and
            not haySupervisor and empleados < 3 =>
          accept solicitarAccesoTrabajo();
          empleados++;
      or
        when empleados > 0 =>
          accept solicitarFinTrabajo();
          empleados--;
      end SELECT
    end if
  end loop
end ESTACION
```

```
  task SUPERVISOR is
  begin

  end SUPERVISOR
  task body SUPERVISOR is
  begin
    while true loop
      int idEstacion := elegir_estacion();
      estacion_manager.solicitarAccesoSupervisor(idEstacion);
      estaciones[idEstacion].solicitarAccesoSupervision();
      supervisar();
      estaciones[idEstacion].finalizarSupervision();
      estacion_manager.finalizarAccesoSupervisor(idEstacion);
    end loop
  end SUPERVISOR

  estaciones: array(1..CANT_ESTACIONES) of ESTACION;

end PROBLEMA_3
```