Departamento de Arquitectura

Examen de Sistemas Operativos

21 de febrero de 2024

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 (31 pts)

(a) Describa:

I. (3 pts) Qué tareas implica la llamada a una System Call

II. (3 pts) Las maneras de enviar/recibir parámetros en una System Call

Solución: La llamada a un system call incluye las siguientes tareas:

- Cargar los parámetros en el lugar adecuado (stack o registros).
- Cargar el número de system call en algún registro específico (Ej: eax en Intel).
- Invocar a la interrupción por software (trap) adecuada (system call handler).
- El hardware cambia el bit de modo a monitor e invoca al manejador de la interrupción que controla que el número de system call pasado en el registro sea menor que el mayor del sistema y, finalmente, invoca al system call correspondiente.
- El valor retornado por el system call es puesto en un registro específico (Ej.: eax en Intel).
- Se vuelve el sistema a modo usuario y se retorna el control al proceso que invocó.

Existen 3 formas de pasar los parámetros al sistema operativo:

- A través de los registros: Se utilizan un conjunto de registros para pasar los parámetros. Tiene el problema de la cantidad de parámetros es fija y que restringe el tamaño del valor. En Intel se utilizan 5 registros: ebx, ecx, edx, esi, y edi.
- Un bloque de memoria apuntado a través de un registro.
- En el stack del proceso que realiza el llamado. El proceso guarda los parámetros con operaciones push sobre el stack y el sistema operativo los saca con la operación pop.

De la misma forma se pueden recibir los datos de respuesta

(b) En el contexto de los sistemas de archivos:

I. (4 pts) ¿Qué es un soft link y que es un hard link? ¿Cómo se representan en las estructuras del sistema de archivos?

II. (3 pts) Sea un archivo llamado `source.file` y que se crea un hardlink llamado `hardlink.file` que apunta a `source.file`. Una vez creado el archivo `hardlink.file` se elimina el archivo `source.file`. ¿Qué sucede al acceder a `hardlink.file`? Justifique su respuesta.

Solución: Un directorio es un archivo cuyo contenido es una tabla. Cada entrada de esa tabla (las entradas de directorios) relaciona un nombre de archivo con un número de inodo. Al querer abrir un archivo (digamos, miarchivo.txt), el FS busca en esa tabla la entrada con ese nombre, y mira cuál es el número de inodo que le corresponde (Ej: 721). Luego realiza las operaciones con el número de inodo.

Un hardlink es una entrada de directorio (hl_miarchivo.txt) que apunta a un inodo asociado con un archivo ya existente (721). Las dos entradas de directorio referencian al mismo archivo. Es importante tener siempre presente que, a partir de que se crea el hard link (es decir, que se crea la segunda entrada apuntando al mismo inodo), ya no hay diferencia entre la referencia original y la nueva. Por este motivo el SO utiliza un conteo de referencias para saber cuántas referencias hay a un cierto inodo. Esta herramienta es muy útil pero es válida solamente en un mismo Filesystem.

Un softlink es un archivo nuevo que referencia una ruta. Este archivo tinene su propio inodo. En el bloque de datos se guarda la ruta del archivo al cual referencia. En el inodo del archivo apuntado el conteo de referencia no aumenta al utilizar esta técnica. Esta herramienta se puede utilizar para referenciar archivos o directorios entre varios Filesystems

Al crear hardlink.file el conteo de referncias en el inodo asociado a source.file aumenta en 1. Al eliminar el archivo source.file el conteo de refrencias en el inodo asociado disminuye en 1. Como todavía existe la referencia al inodo con nombre hardlink.file, se accede al archivo sin problemas.

(c) En un momento se debe quitar la CPU al proceso P1 y asignarla al proceso P2 (context switch).

I. (2 pts) ¿Cuál es el nombre del componente del sistema operativo que realiza dicha tarea?

II. (3 pts) Enumere las tareas que debe realizar dicho componente.

Solución: El componente se llama Despachador

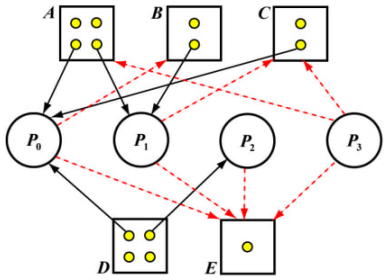
Es el módulo del SO que da el control de la CPU al proceso seleccionado por el planificador de corto plazo. Las tareas que realiza son:

- Cambio de contexto: Salvar registros del procesador en PCB del proceso saliente. Cargar los registros con los datos del PCB del proceso entrante.
- Cambiar el bit de modo a usuario.
- Saltar a la instrucción adecuada que había quedado el proceso que se asignó a la CPU (registro program counter).
- La latencia del despachador debe ser la menor posible
- El planificador es el responsable de seleccionar el próximo proceso a ejecutarse.

(d) En el contexto de la programación concurrente:

i. (3 pts) Explique si la existencia de un ciclo en el grafo de asignación-recursos implica que un sistema entrará en estado de deadlock.

ii. (4 pts) Determine si el sistema dado en la figura genera un deadlock o no. Las flechas negras indican asignaciones de recursos y las flechas rojas punteadas indican requerimientos de recursos. Justifique su respuesta.



Solución: i. La existencia de un ciclo en el grafo de asignación—recursos no implica necesariamente que que un sistema entrará en estado de deadlock (es una condición necesaria pero no suficiente). El deadlock puede producirse o no, dependiendo del orden de ejecución de los procesos y del acceso a los recursos. ii. Pese a tener un ciclo en el grafo, el sistema no necesariamente genera un deadlock, si se sigue el siguiente orden de ejecución:

1. Como los recursos requeridos por P0 pueden satisfacerse con los recursos disponibles, se puede ejecutar P0 y luego de que finalice disponer de los recursos liberados (disponibles para a ser [1,0,1,1,0] + [2,1,1,2,1] = [3,1,2,3,1]
2. A continuación, los recursos requeridos por P1 pueden satisfacerse con los recursos disponibles, se puede ejecutar P1 y luego de que finalice disponer de los recursos liberados (disponibles para a ser [3,1,2,3,1]+ [1,1,0,0,0] = [4,2,2,3,1].
3. Luego se ejecuta P2 y disponibles pasa a ser [0,0,0,0,1] + [4,2,2,3,1]. = [4,2,2,4,1].
4. Por último, se ejecuta P3, sin generarse un deadlock

(e) En el contexto de la comunicación entre procesos basada en pasaje de mensajes:

i. (3 pts) Describa las primitivas provistas por el sistema operativo para soportar comunicación directa.

ii. (3 pts) Describa las primitivas provistas por el sistema operativo para soportar comunicación indirecta.

Solución:

- En la comunicación directa, el receptor se debe mencionar de forma explícita (típicamente con su PID). Las primitivas provistas por el SO tienen la forma:

  enviar(P, mensaje) — envía un mensaje al proceso P.
  recibir(Q, mensaje) — recibe del proceso Q.

- En la comunicación indirecta, se utilizan estructuras independientes llamadas mailboxes. Las primitivas provistas por el SO tienen la forma:

  enviar(M, mensaje) — envía mensaje al mailbox M.
  recibir(M, mensaje) — recibe mensaje del mailbox M.

Problema 2 (32 pts)

Se tiene un sistema operativo con soporte para memoria virtual con las siguientes características:

- Se utiliza un modelo de paginación bajo demanda .
- El hardware y el sistema operativo utilizan direcciones de memoria de 24 bits.
- El tamaño de cada página es de 256 bytes.

- La MMU es configurable y puede trabajar con tablas de páginas con estructura jerárquica de dos o cuatro niveles.
- En cada configuración, todos lo niveles indexan la misma cantidad de entradas por tabla.
- Cada entrada de la tabla de páginas ocupa 24 bits.

(a) (3 pts) Describa qué es la MMU y para qué se utiliza.

Solución: La Unidad de Administracion de Memoria (MMU) es un componente de hardware encargado de realizar la traducción de las direcciones logicas a direcciones físicas en tiempo de ejecucion. Los procesos solo manipulan direcciones logicas y no visualizan las físicas, que solamente son vistas por la MMU.

(b) (3 pts) Determine cómo se distribuyen los bits de la dirección virtual cuando se utiliza una estructura jerárquica de dos y de cuatro niveles. Indique para qué se utiliza cada parte de la dirección virtual en cada caso.

Solución: Por letra, las direcciones virtuales son de 24 bits. Para el offset se precisan 8 bits ya que cada página contiene 256 bytes.

Dos niveles: | 8 bits 1er nivel | 8 bits 2do nivel | 8 bits offset |

Cuatro niveles: | 4 bits 1er nivel | 4 bits 2do nivel | 4 bits 3er nivel | 4 bits 4to nivel | 8 bits offset |

(c) (3 pts) ¿Cuál es la cantidad máxima de memoria virtual que puede ser asignada a un proceso en cada uno de los niveles jerárquicos soportados?

Solución: En ambos es igual: 2^{24} bytes

(d) (6 pts) Suponga un proceso con la máxima cantidad de memoria virtual asignada y otro con tan solo una página asignada. Indique cuántos bytes de memoria utilizaría el sistema operativo para sus tablas de páginas dependiendo de la cantidad de niveles jerárquicos utilizados.

Solución: Dos niveles:

Proceso máxima cantidad de memoria asignada: Se precisaría 1 tabla de 1er nivel y 2^8 tablas de segundo nivel. Cada tabla ocupa $2^8 \times \frac{24}{8}$ bytes. Entonces el sistema operativo utilizará $(1 + 2^8) \times (2^8 \times 3) = 197{,}376$ bytes.

Proceso 1 página asignada: Se precisaría 1 tabla de 1er nivel y 1 tabla de segundo nivel. Cada tabla ocupa $2^8 \times \frac{24}{8}$ bytes. Entonces el sistema operativo utilizará $2 \times (2^8 \times 3) = 2^9 \times 3 = 1{,}536$ bytes.

Cuatro niveles:

Proceso máxima cantidad de memoria asignada: Se precisaría 1 tabla de 1er nivel, 2^4 tablas de segundo nivel, $2^4 \times 2^4 = 2^8$ tablas de tercer nivel y $2^8 \times 2^4 = 2^{12}$ tablas de cuarto nivel. Cada tabla ocupa $2^4 \times \frac{24}{8}$ bytes. Entonces el sistema operativo utilizará $(1 + 2^4 + 2^8 + 2^{12}) \times (2^4 \times 3) = 209{,}712$ bytes.

Proceso 1 página asignada: Se precisaría 1 tabla de 1er nivel, una de 2do, otra de 3ero y una de 4to nivel. Cada tabla ocupa $2^4 \times \frac{24}{8}$ bytes. Entonces el sistema operativo utilizará $4 \times (2^4 \times 3) = 2^6 \times 3 = 192$ bytes.

(e) (3 pts) Suponga que la TLB se encuentra vacía, ¿cuántos accesos a memoria principal requiere un proceso para leer un byte de memoria principal a partir de una dirección virtual en cada uno de los niveles jerárquicos soportados?

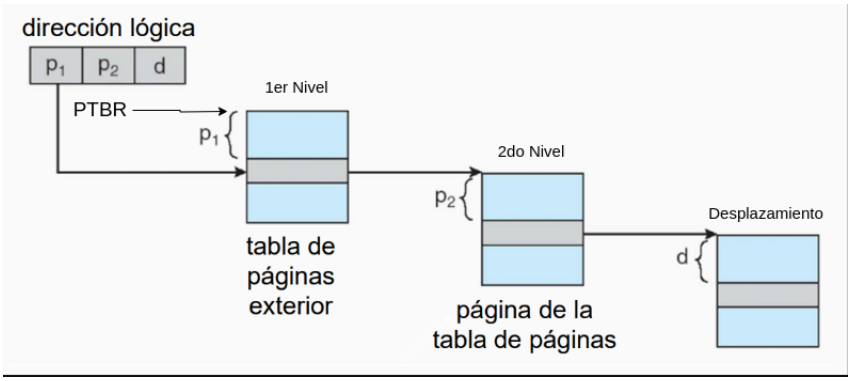
Solución: Dos niveles: 1 acceso para la tabla de 1er nivel, 1 acceso para la tabla de 2do nivel y 1 acceso para acceder al offset dentro de la página buscada = total 3 accesos.
Cuatro niveles: 1 acceso para la tabla de 1er nivel, 1 acceso para la tabla de 2do nivel, 1 acceso para la tabla de 3er nivel, 1 acceso para la tabla de 4to nivel y 1 acceso para acceder al offset dentro de la página buscada = total 5 accesos.

(f) (5 pts) Discuta y justifique en qué casos utilizaría una estructura jerárquica de dos niveles para los procesos de un sistema, y en qué casos utilizaría una de cuatro niveles.

Solución: La estructura jerárquica de 2 niveles es preferible en escenarios en los que la velocidad de acceso a memoria es importante. Además, es más eficiente en el uso de memoria que el esquema de 4 niveles en sistemas con procesos con alto consumo de memoria.
En cambio, una estructura jerárquica de 4 niveles sacrifica velocidad de acceso a memoria a cambio de una reducción en el consumo de memoria en escenarios con grandes cantidades de procesos con bajo consumo de memoria. Esta característica puede resultar fundamental en sistemas con reducida cantidad de memoria.

(g) (3 pts) Muestre un diagrama explicando cómo se realiza la traducción de una dirección de memoria virtual a memoria física para alguno de los dos niveles soportados y presente un ejemplo concreto.

Solución:



(h) (6 pts) Suponga ahora que la MMU brinda un tercer modo de funcionamiento que consiste en una estructura jerárquica de dos niveles con las mismas características que antes pero con un tamaño de página de 64 KiB (2^{16} bits). Determine cómo se destribuyen los bits de la dirección virtual en este caso, y discuta en qué caso sería conveniente utilizar esta estructura en lugar de las dos anteriores y en qué casos no lo sería.

Solución: | 4 bits 1er nivel | 4 bits 2do nivel | 16 bits offset |
Este esquema presenta las mejores características de los dos esquemas anteriores. Es muy eficiente en velocidad de traducción y a la vez es eficiente en la memoria utilizada por el sistema operativo para almacenar las tablas de páginas. En cambio, tiene la desventaja de que presenta un mayor nivel de fragmentación interna en las páginas. Esto puede redundar en una mayor cantidad de memoria principal desaprovechada por lo que no es un esquema adecuado para sistemas con grandes cantidades de procesos.

Problema 3 (37 pts)

Considere una clínica donde realizan exámenes médicos para la expedición de la libreta de conducir. El examen consta de dos partes, una con un médico general y otra con un oftalmólogo. Los pacientes llegan a la clínica, donde son atendidos en orden, primero por el médico general y luego por el oftalmólogo. En la clínica hay solamente un médico general y un oftalmólogo.

Ocasionalmente, un enfermero puede precisar ayuda del médico para atender una emergencia. En caso de que el enfermero precise asistencia, tendrá prioridad sobre los clientes para solicitarla.

Si el médico general está atendiendo una emergencia, los pacientes ingresan únicamente a la consulta con el oftalmólgo, quien realiza las tareas del médico general adicionalmente a las suyas. Aunque en este caso, los pacientes que ya hayan sido atendidos por el médico general y les reste realizarse el examen oftalmológico tienen prioridad sobre los pacientes que van a ser únicamente atendidos por el oftalmólogo.

Se cuenta con las siguientes funciones auxiliares:

- examenMedicinaGeneral(): Ejecutada por el médico general o por el oftalmólogo para realizar la parte del examen correspondiente a medicina general.
- examenOftalmologico(): Ejecutada por el oftalmólogo.
- atenderEmergencia(): Ejecutada por el médico general **y por** el enfermero para atender la emergencia.
- descansar(): Ejecutada por médico y el oftalmólogo luego de atender un paciente o una emergencia.
- otrasTareas(): Ejecutada por el enfermero cuando no precisa ayuda del médico.
- obtenerCertificado(): Ejecutada por el paciente luego de que finalizó ambos exámenes.

Se pide:

Escribir los procesos medicoGeneral, oftalmólogo, enfermero y paciente, y modelar la realidad planteada usando Monitores.

Solución:

```
Se utilizan monitores de MESA.

#define ESPERANDO 1
#define ATENDIENDO 2
#define EMERGENCIA 3

Monitor salaDeEspera {

    bool estadoMedico = ESPERANDO;
    bool estadoOftalmologo = ESPERANDO;
    bool oftalmologoHaceExamenGeneral = false;
    bool hayEnfermero = false;

    int pacientesEsperandoMedico = 0;
    int pacientesEsperandoOftalmologo = 0;

    condition medico;
    condition enfermero;
    condition oftalmologo;
    condition esperoMedico;
```

```
condiiton esperoOftalmologo;
condition esperoExamenMedico;
condition esperoExamenOfalmologico;

void llegaPaciente() {
    if (estadoMedico == ESPERANDO){
        medico.signal();
    } else if (estadoMedico == EMERGENCIA &&
                estadoOftalmologo == ESPERANDO) {
        // atendido por oftalmologo
        oftalmologoHaceExamenGeneral = true;
        oftalmologo.signal();
    } else {
        pacientesEsperandoMedico++;
        esperoMedico.wait();
    }

    esperoExamenMedico.wait();

    if (!oftalmologoHaceExamenGeneral){
        if (estadoOftalmologo == ESPERANDO){
            oftalmologo.signal();
        } else {
            pacientesEsperandoOftalmologo++;
            esperoOftalmologo.wait();
        }
    }

    esperoExamenOfalmologico.wait();
    oftalmologoHaceExamenGeneral = false;
}

void medicoListo() {
    if (hayEnfermero){
        enfermero.signal();
        estadoMedico = EMERGENCIA;
    } else if (pacientesEsperandoMedico > 0){
        pacientesEsperandoMedico--;
        esperaMedico.signal();
    } else {
        estadoMedico = ESPERANDO;
        medico.wait();

        if (hayEnfermero){
            estadoMedico = EMERGENCIA;
            return true;
        } else {
            estadoMedico = ATENDIENDO;
            return false;
        }
    }
}

void medicoTerminaExamen() {
    esperoExamenMedico.signal();
}
```

```
        void medicoTerminaEmergencia(){
            hayEnfermero = false;
        }

        void oftalmologoListo(){
            if  (pacientesEsperandoOftalmologo > 0){
                pacientesEsperandoOftalmologo--;
                esperaOftalmologo.signal();
            } else if (estadoMedico == EMERGENCIA &&
                       pacientesEsperandoMedico > 0) {
                pacientesEsperandoMedico--;
                esperoMedico.wait();
                estadoOftalmologo = ATENDIENDO;
                oftalmologoHaceExamenGeneral = true;
                return true; //
            } else {
                estadoOftalmologo = ESPERANDO;
                oftalmologo.wait();

                estadoOftalmologo = ATENDIENDO;
                if (oftalmologoHaceExamenGeneral){
                    return true;
                } else{
                    return false;
                }
            }
        }

        void oftalmologoTermina{
            if (oftalmologoHaceExamenGeneral){
                oftalmologoHaceExamenGeneral = false;
                esperoExamenMedico.signal();
            }
            esperoExamenOfalmologico.signal();
        }

        void llegaEnfermero() {
            hayEnfermero = true;
            if (estadoMedico == ESPERANDO){
                medico.signal();
            } else {
                enfermero.wait();
            }
        }

    }


    void medicoGeneral(){
        while (true){
            bool esEmergencia = salaDeEspera.medicoListo();
            if (esEmergencia){
                atenderEmergencia();
                salaDeEspera.medicoTerminaEmergencia();
            } else {
```

```
            examenMedicinaGeneral();
            salaDeEspera.medicoTerminaExamen();
        }
        descansar();
    }
}


void oftalmologo(){
    while (true){
        bool tuvoExamenGeneral = salaDeEspera.oftalmologoListo();
        if (!tuvoExamenGeneral)
            examenMedicinaGeneral();
        examenOftalmologico();
        salaDeEspera.oftalmologoTermina();
        descansar();
    }
}

void paciente() {
    salaDeEspera.llegaPaciente()
    obtenerCertificado();
}

void enfermero(){
    while (true){
        otrasTareas();
        salaDeEspera.llegaEnfermero();
        atenderEmergencia();
    }
}

cobegin
    oftalmologo();
    medicoGeneral();
    enfermero();
    paciente();
    paciente();
    ...
    ...
    paciente();
coend
```