Departamento de Arquitectura

Segundo parcial de Sistemas Operativos

30 de noviembre de 2019

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 (20 pts)

Se tiene un sistema con un único procesador en el cual inicialmente no se tiene ningún proceso de usuario ejecutando y en el tiempo t=0 se lanza la ejecución del siguiente programa de usuario:

Programa 1

```
begin
   A = 10
   Bloquea 5ms
   A = A + 90
   if(A == 100)
     pid = fork()
     if(pid == 0){
         A = 0
         Bloquea 15ms
         A = A+1
      }else{
         A =  A + pid
         Bloquea 5ms
         A = A+1
   print A
      }
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. Salvo que se indique explícitamente, el tiempo de ejecución de todas las funciones es de 5 ms (fork, create_thread, condición del if, asignación, etc) y debe tener

en cuenta la composición de las mismas. La planificación definida para los hilos a nivel de usuario es FCFS.

Se pide:

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

Solución: **Tiempo de retorno:** Suma de los intervalos de tiempo en los que el proceso estuvo en la cola de procesos listos.
Tiempo de retorno: intervalo de tiempo desde que el proceso es cargado en memoria hasta que el mismo finaliza su ejecucion.

(b) (14 pts) Suponiendo ejecuta un sistema operativo con un planificador SJF expropiativo y un modelo de hilos Mx1, realice un diagrama de planificación (tiempo vs hilos), comenzando en el tiempo t=0, indicando el estado de cada uno de los hilos.

Solución:

	5	10	15	20	25	30	35	40	45	50
A	E	B	E	E	E	E	L	L	L	E
A'							E	E	E	B

	55	60	65	70	75	80	85	90	95
A	E	E	E	B	L	E	E	E	T
A'	B	B	L	E	E	T	T	T	T

(c) (4 pts) Calcule el tiempo de espera y tiempo de retorno de todos los procesos.

Solución: Tiempo de espera para A: (45 ms – 30 ms) + (75ms – 70ms) = 15 ms + 5 ms = 20 ms
Tiempo de retorno de A: 90 ms – 0 ms = 90 ms
Tiempo de espera para A': 65 ms – 60 ms = 5ms
Tiempo de retorno de A': 75 ms – 30 ms = 45 ms

Problema 2 (20 pts)

(a) (3 pts) Explique el concepto de ensamblaje dinámico de bibliotecas.

Solución: El ensamblaje dinámico consiste en que, en lugar de incorporar las bibliotecas al ejecutable, estas sean cargadas en tiempo de ejecución.

(b) (3 pts) Explique como funciona la asociación de direcciones en tiempo de carga y en tiempo de ejecución. Describa una ventaja de cada tipo se asociación.

Solución: La asociación en tiempo de carga se basa en que al compilar se mantienen direcciones de memoria relativa y recién cuando se carga en memoria estas direcciones se convierten en absolutas. Es mucho más fácil de implementar pues no requiere ningún hardware externo para hacerlo. La asociación en tiempo de ejecución por el contrario no fija las direcciones permitiendo que durante la propia ejecución las direcciones físicas pueden variar.

(c) (3 pts) ¿Con qué mecanismos de hardware es necesario contar para implementar de forma eficiente la asociación de direcciones en tiempo de ejecución?

Solución: Para la implementación eficiente es necesario contar con una MMU que realice la traducción de direcciones y una TLB para que sea eficiente la traducción.

(d) Se desea implementar un sistema operativo con soporte para memoria virtual. Se cuenta con un hardware con un procesador Intel 80386 con una MMU que soporta las siguientes características:

- Direcciones virtuales de 32 bit.
- Estructura jerárquica de dos niveles de paginación.
- Todas las tablas de páginas (primer y segundo nivel) tienen el mismo tamaño y distribución.
- Cada tabla de página ocupa una página completa.
- El tamaño de una página es de 4 KiB.

Se pide:

I. (2 pts) Determine que tamaño tiene una entrada de la tabla de páginas y especifique para qué se utiliza cada bit de una dirección virtual.
II. (2 pts) Muestre un diagrama explicando como se realiza la traducción de una dirección virtual a una física. Presente un ejemplo concreto.
III. (2 pts) Indique el tamaño máximo de memoria física utilizable por un proceso y el tamaño máximo utilizable por todo el sistema.
IV. (1 pt) ¿Es posible implementar ensamblaje dinámico de bibliotecas en este sistema?
V. (2 pts) Suponga que dos procesos diferentes escriben simultáneamente en la misma dirección de memoria virtual. Discuta cuales son los posibles resultados de este evento.
VI. (2 pts) Suponga que el sistema no cuenta con TLB. Calcule cuántos accesos a memoria principal son necesarios para leer a 1 byte de la memoria principal.

Solución:

(i) Como las páginas son de 4KiB (2^{12}B) entonces son necesarios 12bits para el offset por tanto quedan 20 para las tablas de páginas, al ser dos niveles se tienen 10 bits para la de primer nivel y otros 10 para la segunda. Entonces las direcciones se interpretan como: 10 bits – 1er nivel — 10 bits – segundo nivel — 12 bits - offset
Al tener 10 bits para direccionar dentro de cada tabla tenemos entonces 2^{10} entradas. Entonces tenemos 2^{12}B/2^{10}=2^2B = 4 bytes como tamaño de entrada

(ii)

(iii) El tamaño máximo tanto para el proceso como para el sistema es 2^{32} Bytes
Nota: *También es válido si se multiplica 2^{32} por un múltiplo de 2 justificando adecuadamente.*

(iv) La paginación permite tener determinadas páginas compartidas por más de un proceso, esto puede ser usado para implementar el ensamblaje dinámico.

(v) Las direcciones virtuales de un proceso son (en principio) disjuntas. Como las direcciones virtuales deben ser transformadas en físicas antes de realmente saber si habrían o no problemas. El problema surgiría si ambos procesos tienen en su tabla de páginas para esa dirección virtual la misma dirección física.

(vi) Como tenemos dos niveles de tabla de páginas se necesitará primero acceder a la tabla de primer nivel, con el resultado que se consiga en esta a la de segundo y finalmente con lo obtenido en esta última se accederá a memoria principal a leer el byte. En total se necesitarán tres accesos a memoria

Problema 3 (20 pts)

Se desea implementar un sistema de archivos siguiendo un modelo de asignación en forma de lista. El sistema debe soportar un estructura jerárquica de directorios en forma de árbol. Suponiendo que el tamaño de cada bloque de disco es de 1 KiB, que se tiene una cantidad `CANT_BLOQUES` de bloques disponibles y que se cuenta con los siguientes tipos definidos:

```
type sector = array [0..1023] of byte;
type disk = array [0..(CANT_BLOQUES-1)] of sector;
type fat = array [0..(CANT_BLOQUES-1)] of integer;

type entrada_dir = Record
  ...
end; // 32 bytes
```

(a) (3 pts) Complete la definición de `entrada_dir` sabiendo que su estructura ocupa 32 bytes de memoria y defina todo lo necesario para trabajar con el sistema planteado.

Solución: Para poder trabajar en el sistema, falta definir la estructura entrada_dir (32 bytes):

```
type entrada_dir = record
    usado: boolean                  \\ 1 bit
    tipo: (file, dir)               \\ 1 bit
    inicioFAT: integer              \\ 2 bytes
    nombre: array [0..26] of char   \\27 bytes
    tamaño: unsigned integer        \\ 2 bytes
    reservado: array [0..5] of bit \\ 6 bits
end;  \\32 bytes en total
```

Asumimos que un valor de -1 en la tabla FAT indica fin del archivo/directorio y un valor de -2 indica que el bloque esta libre. También asumimos que el primer bloque de disco (el bloque 0) esta asignado al directorio raíz.

Dadas las estructuras definidas en la parte (a) se pide responder:

(b) (2 pts) ¿Cuál es el tamaño máximo de un archivo?

Solución: El primer bloque del disco está asignado al directorio raíz por lo que un archivo puede tener tantos bloques asignados como hay disponibles en el sistema menos uno, es decir $(\mathrm{CANT\_BLOQUES} - 1)$ KiB. El tamaño de un archivo también esta acotado por el valor del campo `tamaño` en su respectivo `entrada_dir`. Al ser un entero sin signo de 16 bits tiene un valor máximo de 2^{16} bytes = 64 KiB (64 bloques en disco). Por último los bloques del disco en la FAT se referencian con un valor entero con signo, es decir que no podemos referenciar más de 2^{15} bloques = 32768 bloques, es decir 32 MiB. En definitiva, el tamaño máximo de un archivo está dado por $\min\{(\mathrm{CANT\_BLOQUES} - 1), 64\}$ KiB.

(c) (4 pts) ¿Cuál es la cantidad máxima de archivos soportados?

Solución: Para que el sistema tenga la mayor cantidad de archivos todas las entradas del directorio raíz deben ser archivos. Como en las `entrada_dir` de tipo directorio no se utiliza el campo tamaño, el directorio raíz puede ocupar todos los bloques del sistema (con un máximo referenciable de 32768 por parte anterior). Cada uno de estos bloques puede almacenar $2^{10}/2^5 = 2^5 = 32$ `entrada_dir` de tipo archivo. Por lo tanto, en total se pueden tener $\min\{\mathrm{CANT\_BLOQUES}, 32768\} \times 32$ archivos en el sistema.

(d) (2 pts) Mencione una ventaja y una desventaja de aumentar el tamaño del bloque de disco (suponiendo que la capacidad total del disco se mantiene constante).

Solución:
Ventaja: Dado que la unidad de acceso al disco es un bloque, si se aumenta el tamaño de bloque se aumentará el desempeño de los accesos al disco porque se reducirán los movi-

mientos del cabezal y se leerán más datos contiguos.
Desventaja: Ocurrirá más fragmentación interna ya que un archivo podría no necesariamente utilizar todo el espacio disponible de un bloque que tenga asignado. Esto lleva a que parte de la memoria quede inutilizada.

(e) (9 pts) Defina e implemente una función que elimine un archivo dada su ruta absoluta. Suponga que para su implementación cuenta con tres funciones auxiliares: una para leer un bloque desde el disco, una similar para escribir un bloque y una para partir la ruta absoluta del archivo. Debe definir estas funciones auxiliares pero no es necesario que las implemente.

Solución:

```
1  funcion eliminarArchivo(ruta[]: char): boolean {
2    var camino[][0..26]: char;
3    var cant: int;
4    var entradas[0..31]: entrada_dir;
5    var punteroFAT, punteroAUX: int;
6    var bloquePadre: int;
7    var encontrado: boolean;
8    var iterEntradas: int;
9    var iterCamino: int;
10
11   cant = partirRuta(ruta,camino);
12
13   iterCamino = 0;
14   /* Comenzamos desde la raiz en el bloque de datos 0 */
15   punteroFAT = 0;
16
17   while (iterCamino < cant){
18     encontrado = false;
19     while(punteroFAT != -1 && !encontrado){
20       if (!leerBloque(punteroFAT,entradas)) return false;
21
22       iterEntradas = 0;
23       while(iterEntradas < 32 && !encontrado){
24         if(entradas[iterEntradas].usado &&
25         entradas[iterEntradas].nombre == camino[iterCamino]){
26
27           // Si es el ultimo del camino debe ser archivo
28           if(iterCamino==cant-1 &&
29             entradas[iterEntradas].tipo==dir)
30               return false;
31
32           // Si no es el ultimo del camino debe ser dir
33           if(iterCamino<cant-1 &&
34             entradas[iterEntradas].tipo==file)
35               return false;
36
37           encontrado = true;
38           bloquePadre = punteroFAT;
39         } else {
```

```
40            iterEntradas++;
41          }
42        }
43
44        if (!encontrado)
45          // Sigo buscando en el siguiente bloque del directorio
46          punteroFAT = fat[punteroFAT];
47        else
48          punteroFAT = entradas[iterEntradas].inicioFAT;
49      }
50
51      if (encontrado){
52        iterCamino++;
53      } else {
54        return false;
55      }
56    }
57
58    // Libero los bloques de la FAT
59    while (punteroFAT != -1){
60      punteroAUX = fat[punteroFAT];
61      fat[punteroFAT] = -2;
62      punteroFAT = punteroAUX;
63    }
64
65    // Borro la entrada de directorio
66    entradas[iterEntradas].usado = false;
67    if (!escribirBloque(bloquePadre,entradas)) return false;
68
69    return true;
70  }
71
72  /*Funciones auxiliares:*/
73
74
75  /* Lee el bloque apuntado por numBloque en el disco
76  y lo carga en buffer y retorna si fue OK */
77  bool leerBloque(numBloque: int; var buffer[1024]: byte);
78
79  /* Escribe los datos almacenados en buffer en disco en el
80  bloque apuntado por numBloque y retorna si fue OK */
81  bool escribirBloque(numBloque: int; buffer[1024]: byte);
82
83
84  /* Dada una ruta absoluta, devuelve las partes del camino,
85  junto con la cantidad de partes:
86   - Para "/home/sistoper/a.txt" se tiene que partes es
87     ["\home", "\sistoper", "\a.txt"] y el valor de
88     retorno de la funcion es 3.
89   - Para "/" se tiene que partes es [ ] y el valor de
90     retorno de la funcion es 0 */
```

```
91 int partirRuta(ruta[]: char; var camino[][0..26]: char);
```