Departamento de Arquitectura

Primer parcial de Sistemas Operativos

28 de setiembre 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.

Problema 1 (6 pts)

(a) (2 pts) Justifique por que no es posible representar el siguiente grafo utilizando Cobegin-Coend. ¿Qué herramienta utilizaría para representarlo?

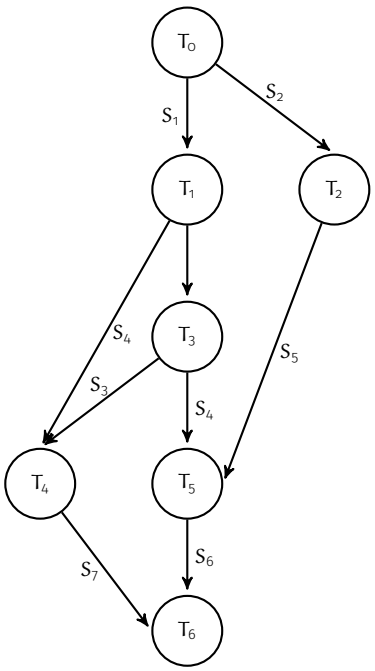

Figura 1: Grafo de precedencias.

(b) (2 pts) Defina *system call* y mencione tres ejemplos de system calls asociados a la gestión de archivos. Enumere las formas tradicionales de pasar parámetros al invocar una system call.

(c) (2 pts) Describa brevemente los métodos para efectuar una operación de entrada/salida.

Problema 2 (9 pts)

Implemente con monitores la solución al problema del productor-consumidor con buffer finito de tamaño N pero considerando que el buffer, cuando no tiene productos, devuelve un producto vacío. Además se desea saber la cantidad de ocasiones que se consume un producto vacío, para ello debe implementar también una función `cuantosVacios()` que devuelve la cantidad de productos vacíos consumidos al retorno de la función.

Se dispone de los siguientes procedimientos auxiliares:

`producir():Producto` Función, produce y devuelve un producto.

`consumir(Producto p)` Procedimiento, consume el producto `p`.

`insertar(Producto p)` Procedimiento, inserta el producto `p` en el buffer.

`obtener():Producto` Función, devuelve el primer producto del buffer. En caso que el buffer esté vacío devuelve un producto vacío (que debe ser consumido.

Solución:

```
1  monitor buffer {
2    Producto contenido[N];
3    int tamaño,vacios;
4    condition lleno;
5
6    void m_insertar(Producto p) {
7      if (tamaño==N)
8        lleno.wait();
9      tamaño++;
10     insertar(p);
11   }
12
13   Producto m_obtener() {
14     Producto p;
15     if (tamaño>0) tamaño--;
16     else vacios++;
17     p=obtener();
18     lleno.signal();
19     return p;
20   }
21
22   int m_cuantosVacios() {
23     return vacios;
24   }
25
26   {
27     tamaño=0;
28     vacios=0;
29   }
30 }
```

```
31
32  void productor() {
33    Producto p;
34    while(1) {
35      p=producir();
36      buffer.m_insertar(p);
37    }
38  }
39
40  void consumidor() {
41    Producto p;
42    while(1) {
43      p=buffer.m_obtener();
44      consumir(p);
45    }
46  }
47
48  int cuantosVacios() {
49    return buffer.m_vacios();
50  }
51
52  void main() {
53    cobegin {
54      productor();...productor();
55      consumidor();...productor();
56    } //coend
57  }
```

Problema 3 (25 pts)

Se desea modelar un club deportivo usando mailboxes. El club dispone un gimnasio con 10 caminadores, 15 bicicletas fijas y un área de musculación que puede ser usada a lo sumo por 5 personas a la vez. La capacidad máxima del club es de 40 personas y, si está completo, los socios esperarán afuera. Además hay 10 profesores que asisten a los socios del club sobre qué tipo de ejercicios deben realizar.

Al entrar al club los socios deberán elegir con cuál profesor consultar y este les indicará la actividad a realizar (caminar, bicicleta o musculación). No se admiten colados entre los socios que consultan al mismo profesor. Luego los socios pasarán al gimnasio donde realizarán la actividad indicada por el profesor. El acceso al gimnasio es por orden de llegada al **club**.

Se dispone de los siguientes procedimientos auxiliares:

`que_profesor():[1 .. 10]` Ejecutada por los socios para decidir que profesor consultar.

`seleccionar_actividad():{caminar=0, bicicleta=1, musculación=2}` Ejecutada por el profesor para seleccionar la actividad a realizar.

`realizar_actividad()` Ejecutada por los socios para efectuar el ejercicio seleccionado.

Se pide implementar las tareas **Socio** y **Profesor**. Se pueden usar tareas auxiliares. Es necesario indicar la semántica de los mailboxes utilizados.

Solución:

```
/* Semantica:
   mailbox infinitos, send no bloqueante y recieve bloqueante. */

mailbox mtx_puestoentrada<nil>,puesto<int>,
  entrada_gym<nil>[40],siguiente_gym<int>,entre_gym<nil>,
  mtx_cola_prof<nil>[10],consultar_prof<nil>[10],asign_prof<int>[10],
  caminadoras<nil>,bicis<nil>,musculacion<nil>;

void socio() {
  receive(mtx_puestoentrada);
  //Entra al club
  int lugar=recive(puesto);
  //Inmediatamente obtiene su lugar de ingreso al gimnasio
  send(siguiente_gym,lugar);
  send(mtx_puestoentrada,nil);
  //mtx_puestoentrada asegura que no hayan "colados"

  int profe = que_profesor();

  //Ordena la consulta al profesor y
  //garantiza que no se mezclen mensajes entre profesor y socio
  recive(mtx_cola_prof[profe]);
  //Despierta profesor
  send(consultar_prof[profe],nil);
  //Recibe la actividad asignada
  int actividad=recive(asign_prof[profe]);
  send(mtx_cola_prof[profe],nil);

  //Espera a que le toque el turno para entrar al gimnasio
  recive(entrada_gym[lugar]);

  if(actividad == caminar){
    recive(caminadoras);
    //Notifico que el socio entró al gimnasio recién cuando
    //puede realizar la actividad
    send(entre_gym,nil);
    realizar_actividad();
    send(caminadoras,nil);
  }else if(actividad == bicicleta){
    recive(bicis);
    send(entre_gym,nil);
    realizar_actividad();
    send(bicis,nil);
  }else{
    recive(musculacion);
    send(entre_gym,nil);
    realizar_actividad();
    send(musculacion,nil);
  }

  send(puesto,lugar);
```

```
52  }
53
54  void gimnasio() {
55    int sig;
56    while(true){
57      //Obtiene el siguiente a ingresar
58      sig=recive(siguiente_gym);
59      //Permite el ingreso del siguiente
60      send(entrada_gym[sig],nil);
61      //Espera a que efectivamente ingrese
62      recive(entre_gym);
63    }
64  }
65
66  void profesor(int i){
67    int actividad;
68    while(true){
69      //Espera que soliciten asesoramiento
70      recive(consultar_prof[i]);
71      actividad=seleccionar_actividad();
72      //Envía el asesoramiento
73      send(asign_prof[i],actividad);
74    }
75  }
76
77  void main() {
78    for(int i=0;i<40;i++) send(puesto,i);
79    for(int i=0;i<10;i++) send(caminadoras,nil);
80    for(int i=0;i<5;i++) send(musculacion,nil);
81    for(int i=0;i<15;i++) send(bicis,nil);
82    for(int i=0;i<10;i++) send(mtx_cola_prof[i],nil);
83    send(mtx_puestoentrada,nil);
84
85    cobegin {
86      socio();...;socio();
87      profesor(0);profesor(1);profesor(2);profesor(3);
88      profesor(4);profesor(5);profesor(6);profesor(7);
89      profesor(8);profesor(9);
90      gimnasio();
91    } //coend
92  }
```

Nota: En la anterior solución los socios esperan fuera del gimnasio hasta tener lugar para realizar la actividad indicada. Sin embargo, durante el parcial se aclaró que también es válido que los socios esperen dentro del gimnasio siempre y cuando entren en el orden correcto.