Universidad de Las Palmas de Gran Canaria
Escuela Universitaria de Informática
Facultad de Informática

Calificación
1
2
3
4
5

Sistemas Operativos
Examen parcial, 11 de mayo de 2002

SOLUCIONES

---

1 (2'5 puntos) Responda con brevedad y precisión a las siguientes preguntas:

- ¿Qué ventajas tiene poder declarar hilos dentro de un proceso pesado?

➢ La ventaja principal es poder dotar de concurrencia interna a una aplicación, con un coste reducido. Varios hilos dentro de un proceso pesado comparten el mismo espacio de memoria (código y datos), así como los recursos que la aplicación tenga asignados. Si nuestro sistema operativo no nos diera la posibilidad de declarar hilos en un proceso, las actividades concurrentes de una aplicación tendrían que definirse mediante procesos pesados, que no comparten memoria ni recursos, y por tanto se consumirían más recursos.

- El intérprete de órdenes (*command interpreter*) puede ser una aplicación independiente (ej. UNIX), o puede estar implementado dentro del núcleo del sistema operativo (ej. CP/M). ¿Qué ventajas o inconvenientes observa usted en cada alternativa?

➢ **Ventajas del intérprete en el núcleo:** Dado que es una aplicación de uso muy frecuente, si se tiene el código en el núcleo, la ejecución de las aplicaciones puede ser más rápida (no hay que cargarla del disco).

➢ **Inconvenientes del intérprete en el núcleo:** El núcleo crece de tamaño. Cuando no se está utilizando el intérprete de órdenes, se está ocupando memoria con código que no se emplea. El sistema es menos flexible, ya que si quisiéramos cambiar la versión del intérprete de órdenes, tendríamos que reinstalar el núcleo del s.o.

- ¿Por qué es necesario establecer dos modos de ejecución (modo supervisor y modo usuario)?

➢ Principalmente, porque en un sistema operativo multiprogramado o multiusuario existen restricciones de acceso a los recursos: protección de zonas de memoria, restricción del uso de archivos y dispositivos de E/S, etc. En segundo lugar, y muy relacionado con lo anterior, para poder preservar la integridad del sistema operativo, que no debe ser dañado por el uso indebido de los programas de usuario.

- ¿Qué diferencia hay entre las llamadas al sistema y los programas del sistema?

➢ Las llamadas al sistema son los mecanismos que utilizan las aplicaciones para solicitar servicios al núcleo del sistema operativo, mientras que los programas del sistema son aplicaciones independientes que se distribuyen junto al sistema operativo y que resuelven necesidades básicas de operación o administración (ej. editores, compiladores, intérpretes de órdenes, etc.) Las llamadas al sistema son la interfaz del s.o. para las aplicaciones, mientras que los programas del sistema son la interfaz del s.o. para los usuarios.

- Si un sistema no es multiusuario, ¿tampoco es multitarea?

➢ No necesariamente. Un sistema puede ser multitarea sin ser multiusuario. Aunque sólo exista un usuario en el sistema, éste puede encontrar útil lanzar varias aplicaciones concurrentes, como puede suceder en un entorno de ventanas.

---

2 (1'25 puntos) Se propone una nueva política de planificación de procesos, que consiste en ceder siempre la CPU al proceso que menos memoria necesita.

a) ¿Esta política ofrece algún beneficio?

➢ Se está dando preferencia a los programas de pequeño tamaño, con lo cual se estimula a los programadores a que escriban aplicaciones lo más pequeñas que puedan (lo cual es deseable).

➢ (ampliación) Si esta idea se extendiera al Planificador de Largo Plazo, estaríamos hablando de mantener en la cola de preparados a los procesos con menos demanda de memoria: en ese caso, la política tendría el efecto beneficioso de mantener en ejecución la mayor cantidad de procesos posible.

b) ¿Y algún efecto negativo?

➢ El efecto negativo más inmediato es el perjuicio causado a los procesos que demanden más memoria para su ejecución, con el riesgo de inanición. Otro efecto negativo es que la política no tiene en cuenta características importantes de los procesos, como la duración de sus ráfagas de CPU, sus requisitos de interactividad, etc., que sabemos que influyen mucho en el rendimiento de la planificación de procesos. Un proceso de pequeño tamaño no implica un proceso de menos duración. Por tanto, el rendimiento de esta política sería imprevisible y, desde luego, alejado del óptimo.

c) ¿Es implementable?

➢ Por supuesto que sí, ya que se pueden conocer por anticipado las necesidades de memoria de un programa (código y datos), o al menos una estimación fiable.

---

3 (2'5 puntos) Tenemos un sistema concurrente con una capacidad de memoria de N bytes. Cada proceso del sistema, para poder ejecutarse, debe reservar previamente un número de bytes. Si no hay cantidad suficiente de memoria, debe quedarse bloqueado hasta que otros procesos vayan liberando memoria suficiente. La ejecución del proceso debe estar regulada mediante un monitor, como se muestra a continuación:

```
monitor.reserva (nbytes);
   … se ejecuta el proceso …
monitor.devuelve (nbytes);
```

Implemente las operaciones de este monitor. Para escribir su solución, puede usar un monitor clásico o bien cerrojos y variables condición.

Nombre

Con un monitor (lenguaje estilo Ada):

```
Monitor ReservaMemoria is
begin

  procedure entry reserva (nbytes: natural) is
  begin
     while espacio_libre < nbytes loop
       HayEspacio.Wait;
       HayEspacio.Signal;
     end loop;
     espacio_libre := espacio_libre - nbytes;
  end reserva;

  procedure entry devuelve (nbytes: natural) is
  begin
     espacio_libre := espacio_libre + nbytes;
     HayEspacio.Signal;
  end devuelve;

  HayEspacio: condition;
  N: constant natural := …;
  espacio_libre: natural range 0..N := N;

end ReservaMemoria;
```

NOTA: El *signal* posterior al *wait* es imprescindible para despertar a **todos** los procesos en espera cuando se devuelve memoria al sistema. P.ej. si hay dos procesos en espera que necesitan 10 y 30 bytes, y un proceso que finaliza devuelve 50 bytes, hay que desbloquear a los dos procesos. Considérese también el ejemplo en el que estén esperando tres procesos que necesitan 60, 10 y 30 bytes; y que un proceso devuelve 50 bytes. Si se desbloquea el proceso de 60, tiene que volver a bloquearse, pero hay que darle una oportunidad a los procesos de 10 y de 30 bytes.

El *signal* de devuelve() se puede sustituir por un *broadcast*, en cuyo caso no hace falta hacer el *signal* posterior al *wait* del método reserva().

Con cerrojos y variables condición (estilo Nachos, escrito en C++):

```
const int N = …;

class ReservaMemoria
{
  private:
     int espacio_libre;
     Lock cerrojo;
     Condition HayEspacio;

  public:

     ReservaMemoria ()
     : HayEspacio(cerrojo)
     { espacio_libre = N; }

     void reserva (int nbytes)
     {
       cerrojo.Acquire();
         while ( espacio_libre < nbytes )
         {
            HayEspacio.Wait();
            HayEspacio.Signal();
         }
         espacio_libre -= nbytes;
       cerrojo.Release();
     }

     void devuelve (int nbytes)
     {
       cerrojo.Acquire();
         espacio_libre += nbytes;
         HayEspacio.Signal();
       cerrojo.Release();
     }
};
```

NOTA: la misma que para la solución de Ada.

---

4 (1'25 puntos) Dada la siguiente carga de trabajo, obtener el diagrama de Gantt, el tiempo medio de retorno y el tiempo medio de espera al aplicar las siguientes políticas de planificación:

- Primero el más corto (SJF) con expropiación.
- Round-Robin con cuanto de 3 u.t.

Proceso	Tiempo de llegada	Duración
P0	0	9
P1	2	5
P2	3	2
P3	5	4

Nombre

Primero el más corto (SJF) con expropiación: admite dos posibilidades

Solución A

P0	P0	P1	P2	P2	P3	P3	P3	P3	P1	P1	P1	P1	P0	P0	P0	P0	P0	P0	P0

Proceso	Tiempo de retorno	Tiempo de espera
P0	20	11
P1	11	6
P2	2	0
P3	4	0
Media	**9,25**	**4,25**

Solución B

P0	P0	P1	P2	P2	P1	P1	P1	P1	P3	P3	P3	P3	P0	P0	P0	P0	P0	P0	P0

Proceso	Tiempo de retorno	Tiempo de espera
P0	20	11
P1	7	2
P2	2	0
P3	8	4
Media	**9,25**	**4,25**

Round-Robin (Q = 3 u.t.): admite dos posibilidades

Solución A:

P0	P0	P0	P1	P1	P1	P2	P2	P0	P0	P0	P3	P3	P3	P1	P1	P0	P0	P0	P3

Proceso	Tiempo de retorno	Tiempo de espera
P0	19	10
P1	14	9
P2	5	3
P3	15	11
Media	**13,2**	**8,2**

Solución B:

P0	P0	P0	P1	P1	P1	P0	P0	P0	P2	P2	P3	P3	P3	P1	P1	P0	P0	P0	P3

Proceso	Tiempo de retorno	Tiempo de espera
P0	19	10
P1	14	9
P2	8	6
P3	15	11
Media	**14**	**9**

---

5 (2'5 puntos) A continuación se muestra la implementación de los semáforos en Nachos:

```
void Semaphore::P()
{
    IntStatus oldLevel = interrupt->SetLevel(IntOff);
    while (value == 0) {
     queue->Append((void *)currentThread);
     currentThread->Sleep();
    }
    value--;
    (void) interrupt->SetLevel(oldLevel);
}

void Semaphore::V()
{
    Thread *thread;
    IntStatus oldLevel = interrupt->SetLevel(IntOff);

    thread = (Thread *)queue->Remove();
    if (thread != NULL)
       scheduler->ReadyToRun(thread);
    value++;
    (void) interrupt->SetLevel(oldLevel);
}
```

a) ¿Qué significa el **`ReadyToRun()`** de la operación V()?

- Sirve para reincorporar en la cola de preparados a un hilo que estaba bloqueado (apuntado por la variable **`thread`**).

b) ¿Qué tipo de cola es el atributo **`queue`**? ¿FIFO? ¿LIFO? ¿con prioridad?

- Se está usando como una cola FIFO.

c) ¿La implementación de los semáforos en Nachos utiliza espera activa?

- No. Cuando un hilo que hace una P() descubre que el semáforo vale cero, se bloquea con Sleep() y no reevalúa el semáforo hasta que alguien le despierta. Por tanto, mientras está bloqueado está fuera de la cola de preparados.

d) ¿Por qué hace falta un **`while`** en la implementación de la operación P()? ¿No puede sustituirse por un **`if`**?

Nombre

- El **`while`** es necesario, porque cuando el hilo que hizo la P() se recupera del Sleep(), ingresa en la cola de preparados por el final. Delante de él, en la cola de preparados, puede haber otros procesos que van a hacer operaciones P() sobre el mismo semáforo, con lo cual, cuando le llegue el turno de CPU al proceso despertado, puede ocurrir que el semáforo vuelva a estar a cero. Por eso tiene que volver a evaluar el valor del semáforo y bloquearse si de nuevo lo encuentra a cero.
- Para sustituir el **`while`** por un **`if`**, habría que hacer modificaciones en los algoritmos aquí expuestos.