SISTEMAS OPERATIVOS (Código: 71902048) **2026**

Material permitido: **Solo calculadora no programable** Tiempo: **120 minutos** **RE**	**Aviso 1**: Todas las respuestas deben estar debidamente razonadas. **Aviso 2**: Escriba con buena letra y evite los tachones. **Aviso 3**: Fecha de revisión en https://www.dia.uned.es/71902048/

1. Conteste **razonadamente** a las siguientes preguntas:

a) *(1 p)* Describir el algoritmo de planificación basado en *múltiples colas de prioridad y realimentación*.

b) *(1 p)* Enumerar y explicar las *cuatro condiciones* necesarias y suficientes para la existencia de *interbloqueo*.

c) *(1 p)* ¿Qué es el *spooling* y cómo se implementa?

d) *(1 p)* Enumerar y explicar brevemente las posibles formas en que se puede estructurar la información contenida en un archivo.

2. *(2 p)* Enumerar las ventajas e inconvenientes de los *hilos a nivel de núcleo*.

3. *(2 p)* Considérese un sistema con memoria virtual en el que se utiliza la técnica de paginación por demanda. En este sistema se han asignado a un cierto proceso X tres marcos de página para su ejecución. La cadena de referencias de página que produce la ejecución del proceso X es:

```
5 3 4 3 8 9 10 7 4 5 4 11 5 5 8 18
```

Determinar el número de fallos de página que se producen si se utiliza el algoritmo de reemplazamiento de páginas LRU.

4. *(2 p)* Dos procesos A y B se ejecutan concurrentemente en un determinado sistema. El proceso A ejecuta unas tareas (`tareas_1()`) y alcanza un punto de encuentro. Posteriormente realiza otras tareas (`tareas_2()`) y finaliza. Por su parte el proceso B ejecuta unas tareas (`tareas_3()`) y llega al punto de encuentro. Posteriormente realiza otras tareas (`tareas_4()`) y finaliza. El primer proceso que llega al punto de encuentro no puede continuar su ejecución hasta que no llegue el otro proceso. No se sabe qué proceso comienza a ejecutarse primero o cuál es el primero que termina. Escribir el pseudocódigo de un programa basado en C que coordine la actividad de los procesos A y B usando **paso de mensajes**, suponer que la comunicación es indirecta a través de buzones y que se dispone de la operación `send` sin bloqueo y de la operación `receive` con bloqueo.

SISTEMAS OPERATIVOS (Cód. 71902048)
Solución Examen Febrero 2026 - Reserva Especial

Solución Ejercicio 1

a) En el algoritmo de planificación basada en prioridades se suponía la existencia de una única cola de procesos preparados. Otra alternativa es implementar varias colas de procesos preparados o *colas de prioridad*, cada una de las cuales solo pueda contener procesos con una determinada prioridad o dentro de un rango de prioridades. En este caso se dice que se utiliza un *algoritmo de planificación basada en múltiples colas de prioridad*.

Con este algoritmo siempre se ejecuta un proceso perteneciente a la cola de preparados con mayor prioridad. Sólo cuando dicha cola está vacía se puede planificar un proceso perteneciente a una cola de preparados de menor prioridad.

En el algoritmo de planificación basada en múltiples colas de prioridad un proceso solo puede pertenecer a una determinada cola de preparados o cola de prioridad durante su tiempo de vida. En consecuencia, posee una prioridad estática. Esta restricción simplifica la implementación del algoritmo pero también lo hace poco flexible. Si se levanta esta restricción y se permite que un proceso pueda ir cambiando de cola de prioridad durante su existencia, es decir, que su prioridad pueda ser modificada dinámicamente, se dice que el algoritmo posee *realimentación*.

La implementación de la realimentación requiere definir un mecanismo que regule cómo se modifica dinámicamente la prioridad de los procesos durante su tiempo de existencia. O visto de otra forma, cuándo se produce la transición de un proceso de una cola de prioridad a otra.

b) Para que se produzca un interbloqueo se deben cumplir necesariamente las siguientes cuatro condiciones:

1. *Exclusión mutua*. Cada instancia de un recurso solo puede ser asignada a un proceso como máximo.
2. *Retención y espera*. Cada proceso retiene los recursos que le han sido asignado mientras espera por la adquisición de los otros recursos que necesita.
3. *No existencia de expropiación*. Si un proceso posee un recurso, éste no se le puede expropiar.
4. *Espera circular*. Existe una cadena circular de dos o más procesos, de tal forma que cada proceso de la cadena se encuentra esperando por un recurso retenido por el siguiente proceso de la cadena.

c) En sistemas multiprogramados el sistema operativo se debe encargar de asignar y controlar el uso de los dispositivos de E/S. En el caso de los dispositivos de E/S dedicados, como es el caso de una impresora, una técnica de control bastante utilizada es el *spooling*. Esta técnica se implementa con un proceso demonio (ver sección 2.2.3) y con un directorio especial, denominado *directorio de spooling* o *spool*. El proceso demonio es el único autorizado para escribir en el dispositivo. Si un proceso de usuario quiere escribir en el dispositivo, debe enviar los archivos que desea escribir al directorio de spooling. El proceso demonio irá mandando al dispositivo de salida uno a uno los archivos que encuentre en este directorio, ya sean de un proceso u otro.

   Usando spooling se previene que un proceso obtenga el uso de un dispositivo de salida y no lo ceda durante un tiempo excesivo, impidiendo que lo puedan usar otros procesos. El spooling no solo se usa para el control de la impresora, también se puede usar para la transferencia de paquetes de datos por una red.

d) La información contenida en un archivo se puede estructurar principalmente de tres posibles formas:

   – *Secuencia de bytes*. El archivo se estructura como un conjunto de bytes contiguos. En consecuencia las operaciones de lectura y escritura se realizan a nivel de byte. Algunos sistemas operativos, como UNIX o Windows, estructuran el contenido de los archivos de esta forma debido a su alta flexibilidad. El sistema operativo no tiene que interpretar la información contenida en cada byte del archivo. Dicha responsabilidad recae sobre los programas de aplicación que se ejecutan a nivel de usuario. Ejemplos de programas de aplicación que acceden a los archivos como una secuencia de bytes son los procesadores de texto y los compiladores.

   – *Secuencia de registros*. El archivo se estructura como un conjunto de registros contiguos de igual longitud. Cada registro posee a su vez una cierta estructura interna. Cuando se realiza una operación de lectura o de escritura se realiza a nivel de registro.

   – *Registros indexados*. El archivo se estructura como un conjunto de registros de longitud variable. Cada registro contiene un campo clave o índice que permite identificarlo. El archivo se organiza en función de la clave de los registros que lo componen. Para realizar una operación de lectura o de escritura se debe especificar la clave del registro que se desea leer o escribir. Si se añade un nuevo registro al archivo, se debe especificar su clave. Es el sistema operativo el que se encarga de ubicarlo dentro del conjunto de registros. Esta forma de estructurar la información de un archivo es utilizada por algunos sistemas operativos de macrocomputadores usados para gestión de datos de transacciones comerciales.

Solución Ejercicio 2

Los *hilos a nivel de núcleo* o más abreviadamente *hilos del núcleo*, son implementados y gestionados directamente por el núcleo del sistema operativo. No requieren de la existencia de una biblioteca de hilos. El sistema operativo mantiene una única tabla de hilos que contienen los bloques de control de todos los hilos del núcleo existentes. Esta tabla es implementada en el espacio de direcciones del núcleo.

La principal ventaja que tienen los hilos del núcleo, es que si uno se bloquea se puede planificar otro del mismo proceso o de otro proceso distinto. Además, en sistemas multiprocesador es posible ejecutar varios hilos del núcleo simultáneamente, cada uno de ellos en un procesador distinto.

Por otra parte, el principal inconveniente de los hilos del núcleo radica en que su gestión contribuye a la sobrecarga del sistema. Para resolver este problema, muchos sistemas limitan el número de hilos del núcleo que se pueden crear, y además reciclan los hilos del núcleo ya existentes. Cuando un hilo del núcleo es destruido, se marca en su bloque de control como no planificable, pero sus estructuras de datos no son eliminadas. Posteriormente, cuando se requiere crear un nuevo hilo del núcleo, lo que se hace es reactivar un hilo del núcleo marcado como no planificable con el objetivo de ahorrarse la sobrecarga de asignar estructuras de datos al nuevo hilo.

Solución Ejercicio 3

Se va a suponer que para implementar el algoritmo LRU se utiliza una lista enlazada gestionada como una pila con tres posiciones disponibles. En la Tabla 1 se muestra el contenido de la pila antes y después de cada referencia de la secuencia que se produce de acuerdo con la información dada en el enunciado, se indica también si dicha referencia produce un fallo (F) o un acierto (A). Se observa que se producen un total de **12 fallos de página**.

Página	**Pila LRU**	**Fallo/Acierto**
5		F
	5	
3		F
	5 → 3	
4		F
	5 → 3 → 4	
3		A
	5 → 4 → 3	
8		F
	4 → 3 → 8	
9		F
	3 → 8 → 9	
10		F
	8 → 9 → 10	
7		F
	9 → 10 → 7	
4		F
	10 → 7 → 4	
5		F
	7 → 4 → 5	
4		A
	7 → 5 → 4	
11		F
	5 → 4 → 11	
5		A
	4 → 11 → 5	
5		A
	4 → 11 → 5	
8		F
	11 → 5 → 8	
18		F
	5 → 8 → 18	

Tabla 1 – Evolución detallada de la pila LRU con tres marcos (12 fallos de página)

Solución Ejercicio 4

La solución que se muestra en la Figura 3 para modelar el punto de encuentro entre los procesos A y B mediante paso de mensajes utiliza dos buzones (`buzón1` y `buzón2`) que inicialmente están vacíos.

```
void proceso_A() /*Proceso A*/
{
        mensaje X;
        tareas_1();
        send(buzón1,X);
        receive(buzón2,X);
        tareas_2();
}

void proceso_B() /*Proceso B*/
{
        mensaje X;
        tareas_3();
        send(buzón2,X);
        receive(buzón1,X);
        tareas_4();
}

main()/*Creación de buzones y ejecución concurrente*/
{
        crear_buzón(buzón1);
        crear_buzón(buzón2);
        ejecución_concurrente(proceso_A,proceso_B);
}
```

Figura 1