PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2333 — Sistemas Operativos y Redes — 1/2018

Interrogación 2

Martes 8-Mayo-2018

Duración: 2 horas
SIN CALCULADORA

1. **[15p]** Sincronización

   1.1) **[3p]** Se ha mencionado que una manera de proveer exclusión mutua para un *thread* es deshabilitar las interrupciones al momento de acceder a una sección crítica, de manera que el *thread* no pueda ser interrumpido, y luego rehabilitarlas al salir de la sección crítica. ¿Qué problema presenta esta implementación en un computador moderno del punto de vista de la exclusión mutua?

   1.2) **[12p]** Se desea simular un proceso de construcción de bicicletas. Una bicicleta está formada por, exactamente, un marco y dos ruedas, y se construye llamando a la función existente `construir()`.

   En el sistema hay múltiples *threads* constructores. Algunos construyen ruedas y otros construyen marcos. Cada vez que un *thread* ha construido una rueda, invoca la función `rueda_lista()`. Cada vez que un *thread* ha construido un marco, invoca la función `marco_listo()`. Cualquier *thread* puede llamar a la función, ya definida, `construir()` una vez que haya un marco y dos ruedas listas. Después que la función `construir()` ha terminado, exactamente una invocación a `marco_listo()` y dos invocaciones a `rueda_lista()` deben retornar.

   Escriba un código (C-like) para la función `rueda_lista()` y un código para la función `marco_listo()`, que permitan implementar el comportamiento descrito. Puede agregar las variables compartidas que considere necesarias. Puede utilizar *locks*, semáforos, y/o variables de condición, usando la API descrita.

   Su solución no debe permitir bloqueos, y debe ser libre de inanición (*starvation*).

2. **[15p]** Considere un sistema con páginas de 16KB, direcciones virtuales de 48 bit, direcciones físicas de 30 bit.

   2.1) **[2p]** Indique la cantidad máxima, en Byte, de memoria virtual que puede utilizar cada proceso, y la cantidad máxima, en Byte, de memoria física que puede direccionar el sistema.

   2.2) **[4p]** Calcule el tamaño, en Byte, de una tabla de páginas de 1 nivel para este sistema. Considere que se utilizan 4 bit para metadata: *valid*, *reference*, *dirty*, y *RW*. ¿Cabe la tabla de páginas en la memoria física?

   2.3) **[6p]** Proponga un esquema de paginación multinivel con la mínima cantidad de niveles necesarios, en el cual para cada nivel, una tabla de ese nivel cabe completamente en una página de memoria. Para justificar su solución debe indicar el tamaño de una tabla (en Byte) de cada nivel y dibujar el esquema de direccionamiento con las dimensiones de cada tabla. Puede suponer que la metadata se encuentra únicamente en la última tabla.

   2.4) **[3p]** Calcule el tamaño, en Byte, de una tabla de páginas invertida de 1 nivel para este sistema. Suponga que los procesos utilizan 8 bit para almacenar su identificador.

3. **[15p]** Memoria y algoritmos de reemplazo.

   3.1) **[3p]** ¿Qué aspecto del problema de direccionamiento de memoria se intenta resolver con el multinivel?

   3.2) **[3p]** ¿Qué relación hay entre el concepto de *working set* y el fenómeno de *thrashing*?

   3.3) **[3p]** ¿Qué ventaja presenta la segmentación respecto a paginación? ¿Qué método se usa en los sistemas modernos?

3.4) **[3p]** Se ha mencionado que LRU es un algoritmo que aproxima bastante bien la decisión óptima de qué *frame* conviene reemplazar. ¿Qué dificultades prácticas tiene la implementación de LRU? Mencione otro algoritmo que se implemente en lugar de LRU y por qué su implementación es mejor (si bien no entregue siempre un mejor resultado que LRU).

3.5) **[3p]** Describa el significado y el uso de *valid bit*, *dirty bit* y *reference bit*. Si corresponde, incluya para qué algoritmo se usan.

4. **[15p]** Discos

4.1) **[3p]** ¿Cuál es la ventaja de utilizar un *buffer* a nivel del sistema operativo para comunicarse con un sistema de I/O (respecto a no tener *buffer*, o tenerlo solo a nivel de usuario)? Explique brevemente su respuesta.

4.2) **[6p]** Para un conjunto de N discos, cada uno con capacidad C. Compare los arquitecturas RAID-0, RAID-1, RAID-5 y RAID-6, en cuanto a:

a) Capacidad máxima de almacenamiento

b) Cantidad máxima de discos que pueden fallar sin interrumpir el funcionamiento del sistema

c) Velocidad para escritura

4.3) **[6p]** Considere la siguiente secuencia ordenada de solicitudes a cilindros del disco: 20, 44, 40, 4, 118, 12, 60. El disco posee 120 cilindros (numerados de 0 a 119), la cabeza lectora se encuentra posicionada en el cilindro 40, y antes de ello se había leído una posición en el cilindro 42. Para los siguientes algoritmos de scheduling de disco, determine la secuencia de lecturas y el desplazamiento total de la cabeza lectora

a) SSTF (*Shortest Seek Time First*)

b) SCAN

c) C-LOOK

API para sincronización

`lock.acquire()`, `lock.release()`. Toma y libera un *lock*.

`semaphore.init(N)` inicializa un semáforo a `N`

`semaphore.P()`, ó `semaphore.down()` decrementa el contador del semáforo

`semaphore.V()`, ó `semaphore.up()` incrementa el contador del semáforo

`condition.wait(l)`, espera incondicionalmente en el *lock* `l`

`condition.signal()`, desbloquea a un *thread* que esté esperando en `condition`, si lo hay

Tantos los semáforos como las variables de condición implementan una cola FIFO para mantener a los *threads* bloqueados.

i	2^i B
6	64 B
7	128 B
8	256 B
9	512 B
10	1024 B

i	2^i B
10	1 KB
20	1 MB
30	1 GB
40	1 TB