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

IIC2333 — Sistemas Operativos y Redes — 1/2019
Midterm (Sistemas Operativos)

Martes 7-Mayo-2019

Duración: 2 horas

1. **[10p]** Para las siguientes afirmaciones, indique si son verdaderas o falsas. Si son falsas **DEBE agregar una justificación** para obtener puntaje. En ambos casos (verdaderas o falsas), puede agregar alguna precisión si lo estima conveniente siempre que no cambie el sentido original de la frase.

   Verdaderas: 0 si es incorrecta; 1 si es correcta.

   Falsas: 0 si es incorrecta, no hay justificación, o la justificación es incorecta; 1 si la justificación es correcta.

   1.1) En un sistema operativo con multiprogramación, pero sin *multitasking*, un proceso que ejecuta `while(1);` podría bloquear al sistema.

   Verdadero. Si no hay *multitasking*, el sistema debe esperar hasta que el proceso en ejecución libere la CPU. Si no hay interrupciones, el sistema operativo no puede intervenir y terminar el proceso.

   1.2) Los sistemas operativos móviles se construyen comunmente en base a *microkernels*.

   Falso. Los sistemas basados en Android son monolíticos, mientras que los basados en iOS son híbridos.

   1.3) La instalación de módulos como extensiones del sistema operativo en el espacio del *kernel* es un potencial riesgo para el sistema operativo.

   Verdadero. Al ejecutar en modo *kernel*, un módulo puede acceder a todos las funcionalidades del *hardware*. La clave es donde dice "espacio del *kernel*", ya que si fuera en el espacio del usuario (como podría ocurrir en un *microkernel*) no sería tan grave.

   1.4) Una función retorna `void` y ejecuta `int* largo = (int*)malloc(1000*sizeof(int))`. Cuando la función termina, el espacio de 1000 `int`s se recupera del *stack* ya que la variable largo se destruye y no hay valor de retorno.

   Falso. Los 1000 `int`s se crean en el *heap*.

   1.5) Uno de los mayores costos de efectuar un *context switch* es el costo de copiar la tabla de páginas del proceso saliente a memoria, y después copiar la tabla de páginas del proceso entrante.

   Falso. Las tablas de páginas no se copian. Se referencian a partir de un registro.

   1.6) Después de ejecutar `fork()`, el proceso nuevo puede modificar el valor de las variables que ha obtenido de su creador sin afectar las variables del creador.

   Verdadero. El nuevo proceso funciona en un espacio de direcciones distinto al espacio del proceso que lo creó.

   1.7) Considere una versión *multithreaded* de un proceso *I/O bound*, usando *kernel threads*. La ejecución sobre un sistema *multi-core* o *single-core* no siempre hace una diferencia significativa en tiempo de ejecución.

   Verdadero. Si el proceso es *I/O bound*, la mayor parte del tiempo los *threads* estarán esperando por operaciones de entrada y salida. En estas condiciones el hecho que puedan ejecutar entre uno o más núcleos no hace una gran diferencia.

   1.8) Si un *thread* quiere entrar a su sección crítica, pero solo puede hacerlo cuando **no** hay *threads* con mayor prioridad que quieran entrar, entonces **no** se cumple **espera acotada**.

   Verdadero. El *thread* no puede saber en cuántos turnos le tocará entrar a su sección crítica, ya que siempre pueden llegar *threads* que entren antes que él debido a su prioridad.

1.9) El esquema de **relocalizar variables** de un programa al momento de cargarlo en memoria, de acuerdo al registro `base` definido por el sistema operativo, permite resolver el problema de protección de regiones de memoria entre procesos.

Falso. El uso de relocalización no garantiza que se resuelvan los problemas de acceso. Un programa podría intencionalmente referenciar memoria más allá de su espacio y, al aplicar la relocalización, terminaría accediendo a memoria de otro proceso.

Podría considerarse **verdadero** si argumentan que el registro `limit` permite completar la protección para no acceder a memoria más allá del espacio asignado.

1.10) Cuando ocurre un evento de `pagefault`, el sistema operativo elimina de la tabla de procesos al proceso que lo generó.

Falso. Cuando ocurre un `pagefault`, se busca la página en disco y se trae a memoria, para luego continuar desde la misma instrucción.

2. **[22p]** Responda de manera breve y lo más precisa posible las siguiente preguntas.

2.1) **[4p]** Un sistema operativo posee un mecanismo de *scheduling* interactivo del tipo *Round-Robin*. Se plantea que este mismo *scheduler* se podría utilizar para planificar procesos de tiempo real (*realtime*) ajustando el tamaño del *quántum* cada vez que haya un proceso de tiempo real presente en el sistema. De esta manera se logra que los procesos de tiempo real avancen rápidamente en la cola *ready*. ¿Es ésta una buena decisión? Explique si este mecanismo serviría para planificar procesos de tiempo real y justifique su respuesta.

R. Un proceso de tiempo real requiere que la ejecución se haya completado antes de un *deadline* determinado. En un sistema RR tradicional, con quántum fijo, esta condición no se puede garantizar ya que cada proceso que llega a la cola *ready* desde esperar, en el peor caso, que transcurra el *quantum* completo de cada proceso anterior que esté en la cola. Si se reduce el tamaño del *quantum*, el tiempo de espera se reduce, pero necesariamente los procesos que están antes del procesos de tiempo real, deben ejecutar. Adicionalmente, si se reduce demasiado el tamaño del quántum, se mantiene un *overhead* por el tiempo de los cambios de contexto.

4p. Mencionar argumentos: requisitos del proceso de tiempo real, efecto de reducir el tamaño del *quantum*, imposibilidad de reducir lo suficiente el tiempo antes que el proceso de tiempo real pueda ejecutar.

2p. Falta algún elemento que permite llegar a la conclusión que no se puede garantizar el cumplimiento del *deadline* debido a condiciones de la cola al momento de llegar el proceso de tiempo real, o bien algunos de los argumentos es incorrecto.

0p. Argumentos mayoritariamente incorrectos.

2.2) **[4p]** Se tiene una biblioteca con funciones compiladas que puede ser usada por múltiples procesos. ¿Qué solución se utiliza para que todos los procesos puedan acceder a estas funciones, sin replicarla múltiples veces en memoria? ¿Cómo se protege el acceso para que ningún proceso la modifique "por error"?

R. Se carga la librería una única vez en memoria, y se hace que los procesos referencien al mismo espacio de direcciones para encontrar las funciones. Éste es el funcionamiento de una biblioteca dinámica. La protección se consigue marcando las páginas en que se carga la biblioteca en modo *read-only*.

4p. Argumentos principales: se carga una única vez en memoria; procesos apuntan al mismo espacio de direcciones; protección usando páginas con modo *read-only*.

2p. Falta algún argumento, o algo no está respondido. Indicar que se carga en memoria del *kernel* es incorrecto.

0p. Argumentos mayoritariamente incorrectos.

2.3) **[4p]** En los sistemas de paginación, se suele incluir un sistema de caché llamado TLB (*Translation Look-aside Buffer*), el cual almacena algunas de las entradas de la tabla de páginas.

a) ¿Cuántos bit necesita, como mínimo, cada entrada de este TLB? Suponga que el sistema de paginación utilizada p bit para el número de página, f bit para el número de *frame*, y d bit para el *offset*.

b) Los TLB, aún teniendo pocas entradas, suelen tener tasas de acierto (*hit rate*) bastante altas. ¿A qué se debe esto?

R.

a) Una entrada de TLB necesita almacenar el número de página, y el número de *frame*. La respuesta es $p + f$.
2p. Respuesta correcta.
0p. Respuesta incorrecta.

b) Se debe a la localidad de referencia. El TLB se beneficia tanto de localidad temporal como de localidad espacial. Si se accede a una dirección en una página, con alta probabilidad se accedera a continuación a otra dirección dentro de la misma página, por lo tanto vale la pena almacenar la traducción en *cache*. El aspecto de localidad temporal ayuda a que no se necesitan muchas entradas de *cache* para obtener un buen *hit rate*, ya que los acceso se repiten con alta probabilidad en cortos periodos de tiempo.
2p. Mencionar el concepto de localidad de referencia (o simplemente localidad). Puede especificar localidad espacial o temporal. En cualquier caso debe haber una justificación de por qué la localidad de referencia beneficia al *caché*.
1p. Se menciona el concepto de localidad de referencia (espacial o temporal), pero no se justifica.
0p. Conceptos incorrectos o ausentes.

2.4) **[3p]** En términos de funcionalidad, ¿qué diferencia hay entre el uso de *locks* y el uso de semáforos en un programa que sincroniza *threads*?

R. El *lock* permite únicamente un proceso como máximo dentro de una sección protegida. Los semáforos permiten una cantidad arbitraria, pero fija de procesos. Un *lock* puede ser liberado sólo una vez, en cambio el semáforo puede ser incrementado (`V()`) múltiples veces para permitir el acceso de más procesos.

3p. Mencionar al menos una diferencia: cantidad de procesos, cantidad de veces que puede ser liberado/incrementado.

1.5p. Diferencia incompleta o parcialmente incorrecta.

0p. Diferencia presentada es incorrecta, o no corresponde a una funcionalidad sino que es algo más bien de forma o interfaz.

2.5) **[3p]** Luego de ejecutar `exit()`, un proceso ya no es escogible para ejecutar. Sin embargo bajo algunas condiciones el sistema podría desear dejar la entrada en el PCB durante cierto tiempo. ¿Por qué el sistema operativo podría querer hacer esto?

R. Para comunicar el estado de término al proceso *parent*. La entrada de PCB queda disponible hasta que el proceso *parent* (sea el que lo creó, o bien el proceso `init` haga `wait()` por él.

3p. Mencionar que otro proceso espera por su resultado de salida. No es necesario explicar las condiciones (no se pedía), pero si las menciona deben ser correctas. Es el proceso *parent* el que debe hacer `wait()`, y no el proceso que hizo `exit()`.

1.5p. Conceptos parcialmente incorrectos.

0p. Conceptos mayoritariamente incorrectos.

2.6) **[4p]** Respecto de los algoritmos de reemplazo de páginas:

a) ¿Cuáles son los criterios que se desea optimizar al momento de diseñar un algoritmo de reemplazo de páginas, y por qué?

b) ¿Cuál es el mejor algoritmo de reemplazo de páginas? ¿Qué dificultad tiene su implementación?

c) Describa brevemente un algoritmo de reemplazo de páginas que se aproxime a LRU, e indique por qué es una buena aproximación.

R.

a) Minimizar cantidad de *pagefaults*, aumentar *hit rate*, minimizar reemplazos de páginas, minimizar transferencias desde/hacia el disco. Todas estas operaciones son costosas en tiempo comparadas con un acceso regular a memoria física.
1p. Mencionar alguna de estas alternativas, e indicar que hay un problema de costo de la operación.
0.5p. Menciona una alternativa incorrecta, o no hace referencia al problema de costo.
0p. No menciona una alternativa correcta ni referencia el problema de costo.

b) El mejor es OPT ó MIN. La dificultad es que su implemetación requiere conocimiento completo de los accesos futuros.
1p. Mencionar OPT ó MIN, o bien describirlo. Menciona el hecho de tener que conocer la secuencia de accesos futuros.
0.5p. Mencionar LRU, y justifica que es costoso actualizar referencias en cada acceso. O bien, menciona OPT/MIN, pero la justificación es incorrecta.
0p. No menciona ninguna de los anteriores, o las justificaciones son incorrectas.

c) Pueden mencionar el de segunda oportunidad, el reloj, los reference *bytes*, la combinación de *dirty-bit* con *reference-bit*, o alguno que conozcan. Son buenas aproximaciones porque aproximan, a intervalos discretos, las decisiones de LRU. Y LRU a su vez intenta aproximar a OPT/MIN.
2p. Describe la idea del algoritmo, y justifica por qué es buena aproximación.
1p. Descripción incompleta del algoritmo, o falta justificación.
0p. No hay descripción, o es incorrecta, o bien hay una descripción incompleta sin justificación o una justificación incorrecta.

3. **[10p]** El siguiente código sincroniza `N` *threads* para escribir en un archivo compartido `FILE *sharedFile`. Este código utiliza 2 variables compartidas: `bool lock;`, y el arreglo `bool waiting[N];`. La variable `i` indica el identificador del *thread* que ejecuta este código.

```
void run() {
  bool key;
  while(1) {
    // muchas líneas de código
    waiting[i] = true;
    key = true;
    while (waiting[i] && key)
      key = test_and_set(&lock);
    waiting[i] = false;
    // ... ...
    write(sharedFile);  // secc critica
    // ... ...
    // ... ...
    j = (i+1)%N;
    while (j != i) && !waiting[j])
      j = ++j%N;
    if (j==i)
      lock = false;
    else
      waiting[j] = false;
    // fuera de la sección crítica
    // ... ...
  }
}
```

3.1) **[4p]** ¿Con qué valores debe iniciar las variables compartidas para que el código funcione correctamente como una solución al problema de la sección crítica?

R.

2p. `bool lock = false;`

2p. `bool waiting[N]= {false};` todas en `false`

Cualquier otro valor: 0p

3.2) **[4p]** Explique si este código cumple la propiedad de espera acotada. Si no la cumple, indique por qué, o bien qué modificación podría hacerse para que la cumpliera.

R.

Sí cumple espera acotada. Si un *thread* desea entrar a la sección crítica hay dos posibilidades: no hay nadie adentro, o ya hay alguien dentro. Si no hay nadie dentro, el `lock` está en *false*, y por lo tanto la ejecución de `test_and_set(lock)` retornará *false*, y el *thread* podrá entrar de inmediato. Si había algún *thread* dentro, el nuevo *thread* se quedará bloqueado. La pregunta es si se puede acotar la cantidad de turnos que tendrá que esperar. El *thread* que sale es el que decide quién entra a continuación, y lo hace buscando desde su posición en el arreglo `waiting` en adelante. Eso garantiza que si el *thread* `i` está adentro, y el *thread* `k` desea entrar, deberá esperar que pasen todos los *threads* en el rango entre `i` y `k` (el arreglo se recorre de manera circular). Esta distancia es máximo `N` por lo tanto sí hay espera acotada.

Los siguientes 4 argumentos pueden estar mezclados en la respuesta, pero deben estar presentes.

1p. Indicar que sí cumple espera acotada.

1p. Mostrar que si es el único *thread*, entonces no espera nada.

2p. Mostrar que si hay más un *thread* dentro, no esperará más de `N` turnos. El `N` puede no estar explícito pero debe expresar la idea que la cantidad de turnos es fija y/o que no puede entrar una cantidad arbitraria de *threads* entre el que está adentro y el que espera.

Argumentar que no cumple espera acotada es incorrecto, pero puede haber hasta 2p si hay algún buen razonamiento entre medio (inevitablemente tendrá que haber un error ahí).

3.3) **[2p]** ¿Cómo se maneja en este código el traspaso del `lock` de un proceso a otro?

R.

El `lock` se entrega al próximo *thread* siguiendo el orden de los identificadores de forma circular, que desea ingresar a la sección crítica. Si no hay ninguno, `lock` vuelve a *false*, y queda libre para el próximo que llegue. Si había alguien esperando, se busca el siguiente en el arreglo `waiting` y se libera ése *thread* específicamente, el cual queda de "responsable" de liberar al `lock` una vez que termine su turno.

1p. Idea de que no hay traspaso explícito del *lock* cuando hay *threads* esperando (no se regresa a *false*, sino que se libera al siguiente que está esperando.

1p. Idea de que el traspaso se hace a un *thread* definido: el próximo que está esperando, o se libera (se regresa a *false*) si no hay *threads* en espera.

4. **[18p]** Considere que direcciona hasta 32 GB de memoria virtual, y hasta 4 GB de memoria física.

4.1) **[4p]** ¿Cuántos bit se necesitan tanto para la dirección virtual como para la dirección física?. Cada dirección de memoria referencia a un Byte.

R. Para resolver este ejercicio basta con expresar cada tamaño en una potencia de 2 y aplicar el logaritmo en base 2 para obtener el número de bits (o bien, quedarse con el número indicado por el exponente).

2p. Bits de dirección virtual: $32\text{GB} = 2^5\text{GB} = 2^5 \times 2^{30}\text{B} = 2^{35}\text{B} \rightarrow$ 35 bits de dirección virtual.

2p. Bits de dirección física: $4\text{GB} = 2^2\text{GB} = 2^2 \times 2^{30}\text{B} = 2^{32}\text{B} \rightarrow$ 32 bits de dirección virtual.

$\rightarrow$ Se da **1p.** en cada cálculo si expresa la fórmula de forma correcta, pero comete errores de cálculo/arrastre.

4.2) **[4p]** ¿Cómo se dividen tanto la dirección virtual como la dirección física si se usan páginas de 4 KB?

asdasdasd Para resolver este ejercicio deben obtener el número de bits de *offset*, dado que esto permite obtener el número de bits de número de página para las direcciones virtuales y el número de bits de número de marco físico, permitiendo descomponer ambas direcciones.

2p. Bits de *offset*: $4\text{KB} = 2^2\text{KB} = 2^2 \times 2^{10}\text{B} = 2^{12}\text{B} \rightarrow$ 12 bits de *offset*.

$\rightarrow$ Se da **1p.** en este cálculo si expresa la fórmula de forma correcta, pero comete errores de cálculo/arrastre.

1p. Dirección virtual: $(35 - 12 = 23)$ bits de número de página + 12 bits de *offset*.

1p. Dirección física: $(32 - 12 = 20)$ bits de número de marco físico + 12 bits de *offset*.

$\rightarrow$ Se da el puntaje completo en cada dirección si expresan bien la separación, independiente de si el *offset* es correcto o no. Se da **0.5p.** si lo expresan bien, pero hacen mal la resta (independiente de si el *offset* es correcto o no).

4.3) **[5p]** Si el tamaño de cada entrada en la tabla de páginas (PTE) está alineado a una cantidad de bits que sea múltiplo de 1 Byte, ¿de qué tamaño, en Byte, es la tabla de páginas? ¿cuántos bit quedan disponibles para *metadata* (incluye aquí los que haya agregado como alineamiento) ?

R. En esta pregunta, es importante notar que alinear la cantidad de bits de la PTE a un múltiplo de 1 Byte es equivalente a alinear a un múltiplo de 8 bits.

1p. Se determina que el tamaño de cada PTE es, al menos, el número de bits del marco físico (ya que su almacenamiento permite efectuar la traducción de dirección virtual a física).

$\rightarrow$ Se da el puntaje completo si determinan al menos eso, independiente de si usan o no el valor correcto de bits de marco físico obtenido en la pregunta anterior. No hay puntaje intermedio en este aspecto.

1p. Al tener 20 bits de marco físico, para tener una cantidad alineada al múltiplo de 1 Byte es necesario añadir 4 bits adicionales, lo que implica un tamaño de PTE de 24 bits = 4 Byte.

$\rightarrow$ Se da el puntaje completo si el aumento de bits es consistente con el número de marco físico obtenido, sin importar si este último es correcto o no de la pregunta anterior. Se da **0.5p.** si lo expresan bien, pero asignan más bits de los necesarios (por ejemplo, añaden 12 bits para lograr un tamaño de 4B cuando podían haber añadido menos).

1p. Se menciona que los bits de *metadata* es igual al número de bits añadido para el alineamiento de la PTE a múltiplos de 1 Byte. No se da puntaje si se da otro tipo de respuesta (más o menos bits de los añadidos).

2p. Tamaño de tabla: $2^{23} \times 3\text{B} = 2^3 \times 3 \times 2^{20}\text{B} = 24\text{MB}$

$\rightarrow$ Se da el puntaje completo en cada dirección si expresan bien la fórmula, independiente de si las dimensiones (bits de página, tamaño PTE) son correctas o no. Se da **1p.** si lo expresan bien, pero hacen mal el cálculo independiente del valor de las dimensiones utilizadas.

4.4) **[5p]** Sin agregar más bits a la dirección, reserve los primeros 4 para un nivel adicional de paginación. Suponga que cada puntero a una dirección de memoria es de 32 bit (4 Byte). ¿Cuánta memoria se necesita para traducir una dirección virtual a una física? Recuerde que los bits de *metadata* sólo se usan en el último nivel.

R.

0.5p. La nueva composición de direcciones queda como sigue: 4 bits de paginación de primer nivel + $(23 - 4 = 19)$ bits de paginación de segundo nivel + 12 bits de *offset*.

$\rightarrow$ Se da el puntaje completo siempre que se sigan estos cálculos, independiente si los bits de número de página y/u *offset* son correctos o no. No hay puntaje si hacen otro cálculo (4 bits menos significativos como primer nivel, por ejemplo).

Ahora, para obtener los tamaños correctos, es importante notar que las tablas de primer nivel tendrán una PTE de 4 Byte, dado que contienen punteros de memoria que apuntan a las tablas correspondientes de segundo nivel.

2p. Tamaño de tabla de páginas de primer nivel: $2^4 \times 4\text{B} = 2^6\text{B} = 64\text{B}$.

$\rightarrow$ Se da el puntaje completo independiente de si el número de bits de primer nivel es correcto o no. Se da **1p.** si expresan bien la fórmula, pero con una cantidad de Byte distinta (igual a 3, por ejemplo), o bien hacen el cálculo mal a pesar de las consideraciones.

2p. Tamaño de tabla de páginas de segundo nivel: $2^{19} \times 3\text{B} = 2^{19} \times (2+1)\text{B} = (2^{20} + 2^{19})\text{B} = 1\text{MB} + 512\text{KB} = 1{,}5\text{MB}$.

$\rightarrow$ Se da el puntaje completo independiente de si el número de bits de segundo nivel es correcto o no. Se da **1p.** si expresan bien la fórmula, pero con una cantidad de Byte distinta (igual a 4, por ejemplo), o bien hacen el cálculo mal a pesar de las consideraciones.

0.5p. Tamaño total utilizado: Tamaño tabla primer nivel + tamaño tabla segundo nivel $\rightarrow 64\text{B} + 1{,}5\text{MB}$.

$\rightarrow$ Se da el puntaje completo independiente de si el tamaño obtenido para cada tabla es correcto o no. No se da puntaje si suman cualquiera de las tablas más de una vez (por ejemplo, (Tamaño tabla segundo nivel)$\times 2^4$), dado que se explicitó durante la prueba que bastaba con sumar el tamaño de cada tabla una sola vez.