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

IIC 2333 — Sistemas Operativos y Redes
Soluciones Interrogación 2

Miércoles 1-Octubre-2014

Duración: 2 horas

1. **[14p]** Responda brevemente las siguientes preguntas:

   Esta pauta debe ser tomada como una guía de respuestas y no como una planilla absoluta de soluciones. Las respuestas en esta pauta pueden ser intencionalmente más largas de lo solicitado a los estudiantes, para efectos de dar explicaciones adicionales y respuestas alternativas.

   1.1) **[3p]** En un sistema multicore, explique por qué los métodos de *Processor Affinity* y *Load Balancing* podrían verse como objetivos conflictivos.

   R. *Processor Affinity* tiende a asignar el mismo proceso al mismo *core* con el objetivo de aprovechar las entradas que pudieran estar almacenadas en el caché. *Load Balancing* trata de distribuir equitativamente los procesos a los *core*. Es posible que por *processor afinity* un grupo de procesos sea asignado siempre al mismo *core*, dejando a los otros *core*s sin trabajo, lo que contraviene la estrategia de *load balancing*.

   1.2) **[3p]** Considere el método de estimación del próximo uso de CPU utlizando para implementar SJF. Describa el comportamiento de la estimación utilizando (a) $\alpha = 0$, $\tau_0 = 100$ms, (b) $\alpha = 0{,}99$, $\tau_0 = 10$ms

   R. La estimación se hace con la fórmula: $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$

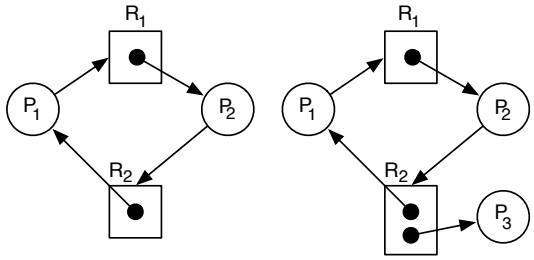
   a) Con $\alpha = 0$, $\tau_0 = 100$ms, la estimación no varía. La fórmula se transforma en $\tau_{n+1} = \tau_n$, con lo que la estimación será siempre $\tau_i = 100$ms.

   b) Con $\alpha = 0{,}99$, $\tau_0 = 10$ms, la fórmula queda $\tau_{n+1} = 0{,}99t_n + 0{,}01\tau_n$. En sucesivas iteraciones se genera una estimación más similar al último tiempo observado, y el término $0{,}01\tau_i$ tiende a desaparecer. La estimación será muy sensible a cambios repentinos de las ráfagas observadas.

   1.3) **[3p]** En un grafo de asignación de recursos, la ausencia de un ciclo indica la ausencia de *deadlock*, pero la presencia de un ciclo no garantiza la existencia de un *deadlock*. (a) ¿Bajo qué condiciones la existencia de un ciclo garantiza *deadlock*? (b) Muestre un ejemplo de cada caso (un caso en que *ciclo* $\Rightarrow$*deadlock*, y un caso en que *ciclo* $\not\Rightarrow$ *deadlock*)

   a) Cuando sólo hay una instancia de cada tipo de recurso disponible, la existencia de un ciclo implica necesariamente la presencia de un *deadlock*. Cuando hay más de una instancia, es posible que aún queden instancias disponibles, por lo cual el ciclo no implicará necesariamente que se ha producido un *deadlock*

   b) Para el primer caso, hay solamente una instancia de cada recurso. Al producirse un ciclo, se produce también un *deadlock*. El segundo caso, en que hay dos instancias del recurso R_2, se presenta un ciclo, pero no *deadlock* ya que el proceso P_3 puede terminar y liberar una instancia de R_2.



1.4) **[3p]** Respecto al sistema de reemplazo de páginas de segunda oportunidad, (a) explique por qué una página que es frecuentemente utilizada no será elegida aún cuando el algoritmo modifica periódicamente su *reference bit* a 0 (y, por lo tanto la próxima que sea examinada, ésta debería ser elegida). (b) ¿Qué ocurre cuando todos los bit están en 1?

R.

a) Una página frecuentemente utilizada tendrá su *reference bit* frecuentemente actualizado a 1. Cada vez que el algoritmo de segunda oportunidad cambia el *reference bit* de esta página a 0, el siguiente acceso a la página volverá a modificarlo a 1. Si estos accesos ocurren frecuentemente, es muy poco probable que el algoritmo recorra todas las páginas y vuelva ver el *reference bit* de esta página en 0 (el algoritmo le dará todas las veces una segunda oportunidad a la página).

b) Si todos los bit están en 1, entonces el algoritmo pasará por todos ellos y le dará una segunda oportunidad a cada uno (modificándolos a 0) y luego elegirá el primero de la lista (el más antiguo). El algoritmo será similar a uno FIFO.

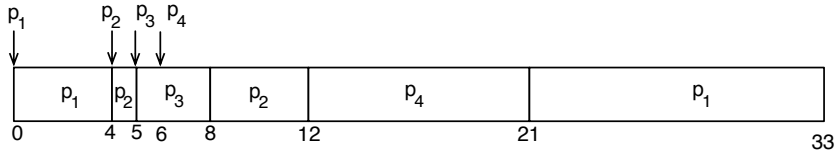
1.5) **[2p]** ¿Para qué se utilizan los *working set* en los sistemas de memoria virtual?

R.. El *working set* permite determinar la **cantidad** de *frames* que están siendo frecuentemente accedidos por el proceso. Es una manera de estimar la cantidad de *frames* que este proceso necesita para poder ejecutar con una frecuencia baja de ocurrencia de faltas de página.

2. **[24p]** Respecto a los temas de sincronización, planificación (*scheduling*) de procesos, y *deadlock*s responda las siguientes preguntas

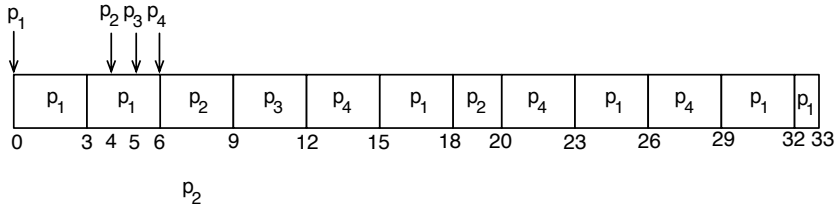
2.1) **[9p]** Considere un conjunto de procesos $P = \{p_1, p_2, p_3, p_4\}$ con ráfagas (*burst-time*) $T = \{T_1 = 16, T_2 = 5, T_3 = 3, T_4 = 9\}$, tiempos de llegada $A = \{A_1 = 0, A_2 = 4, A_3 = 5, A_4 = 6\}$ Calcule el *turnaround time* y el *waiting time* para cada proceso de acuerdo a los siguientes algoritmos:

a) **[3p]** SJF con expropiación (*preemptive*)
R. El siguiente gráfico muestra el orden y tiempo de ejecución.



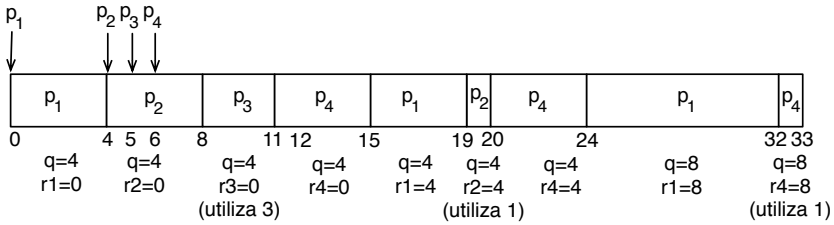
Turnaround time: $t(p_1) = 33-0 = 33$, $t(p_2) = 12-4 = 8$, $t(p_3) = 8-5 = 3$, $t(p_4) = 21-6 = 15$.
Waiting time: $w(p_1) = 21-4 = 17$, $w(p_2) = 8-5 = 3$, $w(p_3) = 0$, $w(p_4) = 12-6 = 6$

b) **[3p]** Round Robin, con $q = 3$
R. El siguiente gráfico muestra el orden y tiempo de ejecución.



Turnaround time: $t(p_1) = 33-0 = 33$, $t(p_2) = 20-4 = 16$, $t(p_3) = 12-5 = 7$, $t(p_4) = 29-6 = 23$.
Waiting time: $w(p_1) = 15-6+23-18+29-26 = 17$, $w(p_2) = 6-4+18-9 = 11$, $w(p_3) = 9-5 = 4$, $w(p_4) = 12-6+20-15+26-23 = 14$

c) **[3p]** Round Robin en que $q = \max\{r_i, 4\}$, donde r_i es el tiempo de ejecución que ha acumulado el proceso i al observarlo cuando está en la cola *Ready*.
R. El siguiente gráfico muestra el orden y tiempo de ejecución.



Turnaround time: $t(p_1) = 32-0 = 33, t(p_2) = 20-4 = 16, t(p_3) = 11-5 = 6, t(p_4) = 33-6 = 27$.
Waiting time: $w(p_1) = 15 - 4 + 24 - 19 = 16$, $w(p_2) = 19 - 8 = 11$, $w(p_3) = 8 - 5 = 3$, $w(p_4) = 11 - 6 + 20 - 15 + 32 - 24 = 18$

2.2) **[6p]** Considere la siguiente propuesta de solución al problema de los filósofos comensales:

```
1  Semaphore chopstick[5];
2
3  Filosofos() {
4    for (int i=0; i<4; i++) chopstick[i] = new Semaphore(1);
5    for (int i=0; i<4; i++) startThread(Cenar, i)
6  }
7
8  void Cenar(int p) {
9    while(1) {
10     Pensar();
11     chopstick[p].wait();
12     chopstick[(p+1)%5].wait();
13     Comer();
14     chopstick[(p+1)%5].signal();
15     chopstick[p].signal();
16     Dormir();
17   }
18 }
```

a) **[3p]** Argumente por qué este código puede producir *deadlock*, haciendo referencias a las propiedades de un *deadlock*.
R. Respecto a las propiedades de un *deadlock*, hay un requerimiento de **exclusión mutua** en que cada *chopstick* no puede ser utilizado por más de un proceso simultáneamente (debido a que usan un semáforo inicializado en 1). Se produce una situación de **hold and wait** cuando un proceso ha ejecutado un *wait* de un *chopstick* y debe ejecutar un *wait* del segundo *chopstick*, que puede estar tomado por otro proceso. Hay **ausencia de expropiación** ya que la primitiva de semáforo no permite quitar un *chopstick* a un proceso que lo ha tomado. Finalmente se puede producir **espera circular** en caso que cada proceso ha ejecutado solamente hasta la línea 11 (cada uno ha ejecutado *wait* por un *chopstick*, porque los semáforos estaban en 1). Cualquier proceso que intente ejecutar la línea 12 quedará esperando ya que hará *wait* por un semáforo que está en 0.

b) **[3p]** Reimplemente `Cenar(int p)` *utilizando semáforos* para que no se produzcan *deadlock*s.
R. Una solución sencilla consiste en hacer que uno de los procesos (cualquiera), tome los *chopsticks* en orden distinto al resto. De esta manera se evita el caso que produce el *deadlock* en que todos han podido tomar el primer *chopstick* y no el segundo. El orden de los *signal* no es, en este caso, relevante ya que la operación no es bloqueante.
Con esta solución, al menos un proceso no podrá tomar su primer *chopstick* y no intentará tomar el segundo, rompiendo la condición de espera circular.

```
1  void Cenar(int p) {
2    while(1) {
```

```
3       Pensar();
4       if(p==2) {
5         chopstick[(p+1)%5].wait();
6         chopstick[p].wait();
7       }
8       else {
9         chopstick[p].wait();
10        chopstick[(p+1)%5].wait();
11      }
12      Comer();
13      chopstick[(p+1)%5].signal();
14      chopstick[p].signal();
15      Dormir();
16    }
17  }
```

2.3) **[9p]** Considere la siguiente situación en un sistema con cinco procesos $\{p_1, p_2, p_3, p_4, p_5\}$, y cuatro tipos de recursos $\{r_1, r_2, r_3, r_4\}$. La cantidad disponible de recursos de cada tipo es Available $= (2, 1, 2, 0)$. Las asignaciones y necesidades máximas son:

Allocated	r_1	r_2	r_3	r_4	Max	r_1	r_2	r_3	r_4
p_1	0	0	1	2	p_1	0	0	3	2
p_2	2	0	0	0	p_2	2	7	5	0
p_3	0	0	3	4	p_3	6	6	5	6
p_4	2	3	5	4	p_4	4	3	5	6
p_5	0	3	3	2	p_5	0	6	5	2

a) **[3p]** Indique si este sistema se encuentra en un estado seguro, inseguro, o *deadlocked* y por qué
R. La matriz *need* se puede construir como Max − Allocated

Need	r_1	r_2	r_3	r_4
p_1	0	0	2	0
p_2	0	7	5	0
p_3	6	6	2	2
p_4	2	0	0	2
p_5	0	3	2	0

Con esto se puede verificar si los procesos pueden satisfacer sus demandas máximas.
p_1 puede satisfacer la solicitud $(0, 0, 2, 0) \Rightarrow$ Available $= (2, 1, 2, 0) + (0, 0, 1, 2) = (2, 1, 3, 2)$
p_4 puede satisfacer la solicitud $(2, 0, 0, 2) \Rightarrow$ Available $= (2, 1, 3, 2) + (2, 3, 5, 4) = (4, 4, 8, 6)$
p_5 puede satisfacer la solicitud $(0, 3, 2, 0) \Rightarrow$ Available $= (4, 4, 8, 6) + (0, 3, 3, 2) = (4, 7, 11, 8)$
p_2 puede satisfacer la solicitud $(0, 7, 5, 0) \Rightarrow$ Available $= (4, 7, 11, 8) + (2, 0, 0, 0) = (6, 7, 11, 8)$
p_3 puede satisfacer la solicitud $(6, 6, 2, 2) \Rightarrow$ Available $= (6, 7, 11, 8) + (0, 0, 3, 4) = (6, 7, 14, 12)$
Dado que todos pueden terminar, el sistema se encuentra en un estado *seguro*, y nunca entrará en *deadlock*.

b) **[3p]** A partir del estado planteado, el proceso p_1 efectúa una solicitud de recursos $(0, 4, 2, 0)$. ¿Cuál debe ser la respuesta del sistema?
R. La solicitud $(0, 4, 2, 0)$ excede la cantidad máxima declarada por p_1. La aserción Request$_1 \leq$ Need$_1$ no se cumple y el proceso p_1 falla.

c) **[3p]** A partir del estado planteado inicialmente, el proceso p_2 efectúa una solicitud de recursos $(0, 1, 2, 0)$. Indique si el sistema queda en un estado seguro, inseguro o *deadlocked* y por qué.
R. Si se asigna la solicitud Request$_2 = (0, 1, 2, 0)$, el nuevo arreglo de recursos disponibles es Available $= (2, 0, 0, 0)$, además de Allocated$_2 = (2, 1, 2, 0)$, y Need$_2 = (0, 6, 3, 0)$.
Con esta situación, ningún proceso podría satisfacer su demanda máxima, en caso que la solicitara. El sistema **NO** se encuentra en *deadlock*, pero sí en un estado **inseguro** y podría quedar en *deadlock*.

3. **[16p]** Respecto a los métodos de direccionamiento de memoria,

3.1) **[9p]** Considere un sistema con 8GB de memoria física, y páginas de tamaño 8KB. Cada entrada en la tabla de páginas utiliza 32-bit. Se requiere mapear un espacio de direcciones virtuales de 46-bit.

a) **[2p]** ¿Cuánto espacio ocupa una tabla de páginas para estas condiciones?
R. Hay 46 bit disponibles para la dirección virtual. Para almacenar un desplazamiento dentro de una página de 8KB $= 8 \times 2^{10}$B $= 2^{13}$B se necesitan 13 bit. Quedan $46 - 13 = 33bit$ para direccionar una página. La tabla de páginas de cada proceso podría tener $2^{33} = 8 \times 2^{30}$ entradas. Si cada entrada de la tabla de páginas utiliza 32bit $=$ 4byte, entonces la tabla de páginas completa utiliza $4 \times 8 \times 2^{30}$byte $=$ 32GB.

b) **[3p]** Si se implementa un sistema de paginación multinivel, ¿qué profundidad se necesita para que cada tabla de página quepa completamente en 8KB?
R. Si una tabla de página puede ocupar, a lo más 8×2^{10}B, y cada entrada utiliza 4B, entonces la tabla de página puede almacenar $2^{13}/2^2 = 2^{11}$ entradas.
Con un nivel de paginación, se pueden direccionar $2^{11} \times 2^{13}$B $= 2^{24}$B $=$ 4MB
Con dos niveles de paginación, se direccionan $2^{11} \times 2^{11} \times 2^{13}$B $= 2^{35} =$ 32GB
Con tres niveles de paginación, se direccionan $2^{11} \times 2^{11} \times 2^{11} \times 2^{13}$B $= 2^{46} =$ 64TB
Se necesitan 3 niveles de profundidad en la paginación multinivel para alcanzar el espacio de 46-bit.

c) **[2p]** Considere un proceso con 3 segmentos: 12 KB de código ejecutable, 256KB de datos, y 4KB de *stack*, y que es cargado completamente en memoria. ¿Cuántas páginas utiliza este proceso de acuerdo al esquema original?
R. El esquema original considera páginas de 8KB. El segmento de código necesita 2 páginas, el segmento de datos necesita 32 páginas, el segmento de *stack* necesita 1 página. Con esto, el proceso necesita al menos de 35 páginas.
También se necesita cargar la tabla de páginas completa. Si la tabla de páginas requiere 32GB, entonces se necesitan, además, $2^{35}/2^{13} = 2^{22}$ páginas.
En total, se necesitan $2^{22} + 35$ páginas.

d) **[2p]** ¿Cuántas páginas utiliza este proceso de acuerdo al esquema de la pregunta (b)? (cada tabla de páginas en una página)
Tal como en la pregunta anterior, se necesitan al menos 35 páginas para los segmentos del proceso. Además para la paginación se necesita una página para la tabla del primer nivel, una página para la primera tabla del segundo nivel, y una página para la primera tabla del tercer nivel. Esta última tabla será la que contenga las entradas de las 35 páginas del proceso.
En total se necesitan 38 páginas, sumando $38 \times$ 8KB $=$ 304KB

3.2) **[4p]** Considere un sistema de memoria con paginación que utiliza un TLB. El tiempo de acceso al TLB es 10 ns (ns$= 10^{-9}$), y el tiempo de acceso a la memoria es 100 ns.
Si se desea que el tiempo efectivo de acceso a la memoria no sea mayor que un 10 % del tiempo de acceso al TLB, ¿cuál debe ser la tasa de aciertos H (*hit rate*)?
R. El tiempo efectivo de acceso a memoria, considerando una tasa de aciertos H se puede estimar por:

$$t_{ef} = H \times t_{\text{TLB}} + (1 - H) \times (t_{\text{TLB}} + t_{\text{mem}})$$

Con $t_{ef} = 1{,}1 \times 10\text{ns} = 11\text{ns}$, $t_{\text{TLB}} = 10\text{ns}$, y $t_{\text{mem}} = 100\text{ns}$, se tiene

$$\begin{aligned} t_{ef} &\geq H \times t_{\text{TLB}} + (1 - H) \times (t_{\text{TLB}} + t_{\text{mem}}) \\ 11\text{ns} &\geq 10H\text{ns} + (1 - H) \times 110\text{ns} \\ 11 &\geq 10H + 110 \times (1 - H) \\ H &\geq 99/100 = 0,99 \end{aligned}$$

3.3) **[9p]** Considere un sistema con 3 frames de memoria. Suponga la siguiente secuencia de acceso a páginas (*page reference string*):

5,4,8,2,4,5,4,1,5,2,5,4,5,2,7

Calcule la cantidad de *page faults* que se producen, y el conjunto final de páginas que quedan cargadas en memoria para los siguientes algoritmos de reemplazo:

a) **[3p]** Reemplazo LRU

R. En este caso se producen 9 *page faults*

Access	5	4	8	2	4	5	4	1	5	2	5	4	5	2	7
Frame 0	5	5	5	2	2	2	2	1	1	1	1	4	4	4	7
Frame 1	-	4	4	4	4	4	4	4	4	2	2	2	2	2	2
Frame 2	-	-	8	8	8	5	5	5	5	5	5	5	5	5	5
PFs	1	2	3	4	4	5	5	6	6	7	7	8	8	8	9

b) **[3p]** Reemplazo FIFO

R. En este caso se producen 11 *page faults*

Access	5	4	8	2	4	5	4	1	5	2	5	4	5	2	7
Frame 0	5	5	5	2	2	2	2	1	1	1	1	4	4	4	4
Frame 1	-	4	4	4	4	5	5	5	5	2	2	2	2	2	7
Frame 2	-	-	8	8	8	8	4	4	4	4	5	5	5	5	5
PFs	1	2	3	4	4	5	6	7	7	8	9	10	10	10	11

c) **[3p]** Reemplazo óptimo

R. En este caso se producen 7 *page faults*. En el caso del último *page fault* de esta secuencia, cuando no es posible determinar qué *frame* va a ser menos utilizado en el futuro, se puede reemplazar cualquiera. En este caso se usó el criterio FIFO.

Access	5	4	8	2	4	5	4	1	5	2	5	4	5	2	7
Frame 0	5	5	5	5	5	5	5	5	5	5	5	5	5	5	7
Frame 1	-	4	4	4	4	4	4	1	1	1	1	4	4	4	4
Frame 2	-	-	8	2	2	2	2	2	2	2	2	2	2	2	2
PFs	1	2	3	4	4	4	4	5	5	5	5	6	6	6	7

Considere que la memoria física se encuentra inicialmente vacía, por lo tanto los primeros 3 acceso necesariamente generarán un *page fault*