Administració de memòria

1.- A la taula de pàgines hi ha un camp anomenat protecció que conté una combinació de bits per permetre lectura (r), escriptura (w) y execució (x) que ens indica qué es pot fer amb la pàgina. Serveix aquest camp per garantir la protecció de memòria entre processos? Si és que si raoneu com ho fa i si és que no expliqueu per què no serveix per això.

2.- Doneu alguns exemples de per què pot tenir sentit que hi hagi àrees de swap diferents per a diferents pàgines o porcions de memòria.

3.- Raoneu per què les pàgines que no es poden modificar també es guarden a l'àrea de swap.

4.- En quines situacions es pot donar el cas de que una crida d'alliberar/reservar memòria no toqui la taula de pàgines?

5.- Com pot ser que un procés sigui desbancat de la CPU en les següents condicions:
La política és Round Robin pur
El procés no ha acabat el seu quantum
El procés no ha fet cap crida al sistema

6.- Per què no podem tenir memòria compartida entre dos processos amb les estructures de dades explicades a classe? Què creus que s'hauria d'afegir (sense gaire detall i sense pensar en la eficiència) per que es suportés la memòria compartida?

7.- Examen Final Q2 02-03. En un sistema con memoria virtual paginada uno de los parámetros del sistema es el tamaño de página. Enuncia alguna ventaja y alguna desventaja de tener un tamaño de página grande (p.ej. tamaño de 1Mb frente a 4 kb).

8.- Examen Final Q1 00-01. Existeix un cas on un procés pot accedir a una adreça de memòria invàlida sense ser error, quin? Com hauria d'actuar llavors el sistema operatiu?

9.- Examen Final Q1 01-02. Com ho faríem a Onion per saber a quina adreça de memòria havia accedit l'usuari quan es produeix una excepció de memòria?

10.- Examen Final Q2 01-02. ¿Por qué el bit de referencia y modificado (dirty bit) deben ser actualizados por el hardware y no por el sistema operativo?

11.- Examen Final Q2 01-02. ¿Qué se necesita hacer con la tabla de páginas de un proceso al hacer un cambio de contexto?

12.- Suposeu que tenim una màquina amb 4 pàgines de memòria física (inicialment buida) i tenim la següent seqüència d'accessos a memòria:
W(1) R(2) W(3) R(4) W(4) R(5) R(6) R(1) W(3) R(4)
Si apliquem una política NRU (FIFO per desempatar), quantes fallades a pàgina tindrem?
I si netejem el bit d'accedit cada 4 accessos a memòria?
I si afegim el bit de *dirty*?

13.- Supongamos que a Onion le incorporamos memoria virtual paginada. el espacio lógico de un proceso se divide en: Espacio de sistema (páginas de la 00h a 7Fh) y espacio de

usuario (páginas de la 80h a la FFh). todas las rutinas del sistema operativo están siempre cargadas en memoria física.

Cada ciclo de reloj se realiza una referencia a memoria. Cuando se produce un fallo de página (en el mismo ciclo) se activa la rutina **MMU que dura 2 ciclos** (página lógica 11h). Durante el primer ciclo, esta rutina aplica **NRU** como algoritmo de reemplazo. Cada trama dispone de **un bit de referencia r** y de **un bit de modificado m**. En caso de conflicto, aplicar FIFO. Además se programa la transferencia de disco (aplica **prefetch**: lectura de la página que produjo el fallo y de la siguiente página lógica) y se elije un nuevo proceso para RUN.

Cada 4 ciclos la rutina multiplexar actualiza (reset) el bit r de todas las tramas. Suponemos un tiempo inapreciable para la ejecución de esta rutina.

Pasados 3 ciclos desde la finalización de la rutina MMU se produce la interrupción de disco que activa la rutina **DISK** (página lógica 12h). Esta rutina, que tarda **1 ciclo** en ejecutarse, gestiona el fin de la transferencia de disco (primero se trae la página de prefetch y luego la que produjo el fallo; ambas tendrán el bit de referencia a 1) y aplica preempción inmediata si el proceso que se desbloquea es más prioritario que el que se ejecuta. Se supone que puede servirse más de una transferencia con disco en paralelo sin aumentar el tiempo de transferencia.

La interrupción más prioritaria es la de reloj, seguida de la de fallo de página y por último la de disco. La rutina **MMU se ejecuta con las interrupciones desinhibidas**.

El algoritmo de scheduling de Onion es **Round-Robin por prioridades** asignando un quantum de 2 a todos los procesos. el proceso P0 o nulo que siempre referencia a su página lógica 80h, y que siempre se encuentra cargada en memoria, tiene prioridad 0. Los procesos P1 y P2 tienen ambos prioridad 2. Todas las rutinas de sistema son más prioritarias que los procesos de usuario.

Las referencias a páginas lógicas que realizan a partir del instante 0 son las siguientes (donde R(n) indica una lectura de la página n, y W(n) una escritura en la página n):

P1: ..., R(80h), W(81h), R(90h), W(91h), R(92h).

P2: ..., R(80h), W(90h), R(A0h), R(A1h).

En el instante 0, justo después de ejecutarse multiplexar tenemos estas páginas en memoria (first indica más tiempo en memoria). En este instante, multiplexar coloca en RUN a P1.

Página	*Proceso*	*r*	*m*	
81	1	0	0	*first*
80	1	0	0	
91	1	0	0	
90	1	0	1	
81	2	0	0	
80	2	0	0	*last*

Disponemos de sólo **6 tramas** en memoria para asignar de forma **global** a los procesos P1 y P2.

a) Dibujar el diagrama de Gantt indicando el proceso que se ejecuta y la página a la que referencia, así como los instantes en que se producen fallos de página con la notación FP(proceso, página). Incluir también los ciclos en que se ejecuta código de SO.

b) Especificar: página, proceso, bits r y m de las páginas que se encuentran en memoria en los instantes: 6, 10 y 17.