EXAMEN DE JUNIO de 2007 (PROCESOS)

Un sistema operativo dado tiene planificación expulsiva y basada en prioridades pero con un núcleo no expulsivo (las llamadas al sistema no se ven interrumpidas).

En todo sistema basado en prioridades se puede dar una situación anómala cuando un proceso de alta prioridad se encuentre esperando a otro proceso de prioridad inferior del cual depende de alguna manera, dicho problema se denomina *inversión de prioridad*. Un proceso puede estar esperando a otro, por ejemplo, debido al bloqueo en un semáforo.

a. Proponga un escenario en el cual se dé la situación antes citada. Indique qué procesos habría en ejecución, con qué prioridades y la traza de ejecución que daría lugar a esta situación. (1 punto)

Una estrategia eficiente en la resolución de la *inversión de prioridades* es la llamada *herencia de prioridad*. Esta técnica consiste en que cuando un proceso de alta prioridad espera por otro de prioridad menor, el segundo gana la prioridad del primero, para así finalizar cuanto antes la espera del más prioritario.

b. Si nos centramos únicamente en el bloqueo de procesos entre sí por medio de semáforos binarios: (4 puntos)

i. A la hora de subir un semáforo, en un proceso de baja prioridad, aumentada en base a esta técnica ¿Sería suficiente mantener dos indicadores de prioridad (original y efectiva) en el BCP del proceso y al subir el semáforo asignar como prioridad efectiva la original (devolverle a su prioridad)? ¿Habría que hacer algo más?

ii. Cuando se sube un semáforo, ¿se debería despertar siempre al que lleva más tiempo esperando en dicho semáforo? Indique la forma más eficiente de ordenar u organizar la cola de procesos bloqueados en un semáforo.

iii. Indique el algoritmo que seguiría la llamada *bajar_semáforo*.

iv. Indique el algoritmo que seguiría la llamada *subir_semáforo*.

Supongamos que en el diseño del sistema se plantea extender el uso de las prioridades a la gestión de memoria. El máximo número de marcos de página que puede usar un proceso dependerá de su prioridad. Dicha estrategia pretende que los procesos más prioritarios causen un menor número de fallos de página. Asuma que para el control de parte de la gestión de memoria el sistema dispone de un demonio de paginación.

c. Si inicialmente no consideramos el caso de la *herencia de prioridad* : (4 puntos)

i. Indique las modificaciones puntuales que habría que hacer a:

La rutina de tratamiento de fallo de página.

Algoritmo de reemplazo.

ii. Se plantea que este proceso puede ser especialmente delicado para los marcos de páginas compartidos. Para ello comience indicando qué marcos de página pueden tener compartidos dos procesos, cuándo se comparte y cuándo se dejan de compartir.

iii. ¿El tratamiento propuesto para los marcos de página debería considerar de forma diferente las que están compartidos? Justifique la respuesta y plantee un esquema que valga para todos los casos correspondientes.

d. Indique, considerando ahora la *herencia de prioridad*, si sería mejor otorgar marcos de página a un proceso en base a su prioridad original o a la modificada por medio de este mecanismo (1 punto).

SOLUCIÓN

a) Supongamos que hay tres porcesos: A (prio:0) B (prio:10) C(prio:20)

Inicialmente sólo está C (de baja prioridad) y baja una semáforo para acceder a un recurso. En ese momento, como resultado de la finalización de una operación de E/S, se desbloquean los procesos A y B. En ese momento, se llama al planificador que concede el procesador al proceso de más alta prioridad, A, éste ejecuta pero intenta bajar el mismo semáforo. A se bloquea en el semáforo que posee C. El siguiente proceso a ejecutar (B y C están listos) sería B como proceso activo de más prioridad. Si B no realiza llamadas bloqueantes, en un esquema de prioridades como el de este sistema seguiría ejecutándose, en una situación en la cual hay un proceso más prioritario (A) que no puede ejecutar porque un proceso de muy baja prioridad (C) no llega a ejecutar para liberar el semáforo.

Hay que tener en cuenta que la inversión de prioridad se produce siempre fuera del código protegido de una llamada al sistema. Que un proceso de poca prioridad esté realizando una llamada al sistema y que otro proceso de mayor prioridad espere en la cola de listos, no es una inversión de prioridad, propiamente dicha, se debe al funcionamiento normal de un núcleo no expulsivo.

b)

i. Cada proceso debe tener los dos contadores, pero no basta con cambiar el efectivo al subir un semáforo debido a que puede tener más de un semáforo o producirse esperas en cadena. Lo más razonable sería recorrer las colas de espera de todos los semáforos que ha bajado y tomar la prioridad efectiva del proceso más prioritario de todas esas colas.

ii. No, la herencia de prioridad está pensada para intentar liberar al proceso que espera que tenga mayor prioridad si ahora se despierta sólo al que lleva más tiempo esperando éste puede ser de cualquier prioridad y se repetiría el problema anterior. O se despiertan todos y todos compiten en el planificador o sólo se despierta al que tiene más prioridad. Esta segunda opción es mucho más eficiente. Para agilizar este proceso, una alternativa interesante sería ordenar las colas de procesos bloqueados en un semáforo por prioridades. Este esquema puede causar inanición, pero ese efecto es propio de cualquier sistema con prioridades (o se quiere tener prioridades estrictas o se quiere repartir equitativamente el

procesador).

iii. Se deberían seguir los siguientes pasos:

1. Verifico si el semáforo no está subido, si no lo está entonces se baja y registro en la estructura del semáforo que ese proceso es el que lo tiene.
2. Si ya está bajado, este proceso hay que bloquearlo, para ello:
    1. Se coloca ordenado en la cola de espera del semáforo según su prioridad efectiva.
    2. Se modifica la prioridad efectiva del proceso que posee el semáforo (sólo si ésta es menor), asignándole la prioridad efectiva del que se ha bloqueado.
    3. Si el proceso que dispone del semáforo está a su vez bloqueado por otro proceso en otro semáforo, la prioridad efectiva modificada se propaga hacia el otro proceso.
    4. Se llama al planificador

iv. Se deberían seguir los siguientes pasos:

1. Se modifica la prioridad efectiva del proceso, devolviéndole su prioridad original, para empezar.
2. Recorro la lista de todos los semáforos que aún dispone el proceso (salvo el que se ha subido). Si el primer proceso de la cola (el más prioritario) tiene como prioridad efectiva una mayor que la de este proceso, entonces le asigno dicha prioridad efectiva.
3. El primer proceso de la lista de bloqueo del semáforo que se ha subido se coloca en la lista de listos.
4. Si la prioridad efectiva del proceso ha variado (quiere decir que se ha desbloqueado a un proceso que me había cedido su prioridad), entonces llama al planificador

c)

i. La rutina de tratamiento de un fallo de página. Puede provocarse por varios aspectos, pero los que afectarían al algoritmo son aquellos que implican la necesidad de ocupar un nuevo marco de página (rellenándolo desde disco o desde memoria [ceros o copias]). En este caso se debe comprobar si el número de marcos de página imputables a este proceso no excederían el límite debido a su prioridad.

1. Si lo excede, entonces se debe aplicar un algoritmo de reemplazo local entre los marcos de página de este proceso.
2. Si no lo excede, entonces
    1. Se verifica si hay marcos libres, si es así se toma uno.
    2. En caso contrario, se aplicará el algoritmo de reemplazo con ámbito global.

Por contra, el algoritmo de reemplazo, una vez determinado si es local o global sería el mismo.

Esta política no implica que el algoritmo de reemplazo tome páginas de procesos poco prioritarios y se las dé a los de mayor prioridad. El reemplazo es siempre de tipo LRU, pero aplicando el ámbito local/global apropiado.

ii. Dos procesos pueden compartir marcos de páginas cuando:

    1. Ambos ejecutan el mismo código o biblioteca dinámica: Se comparten al hacer un *fork* o al hacer un *exec* que afecta a un código que ya está en memoria. Se dejan de compartir cuando el penúltimo de los procesos que los utilizan finaliza.

    2. Páginas de datos con COW: Se comparten al hacer un *fork* y se dejan de compartir al escribir en ellas o al morir uno de los procesos.

    3. Ficheros proyectados compartidos o regiones declaradas explícitamente: Se comparten, por ejemplo, al invocar la llamada *mmap*. Se dejan de compartir por muerte del proceso o de forma explícita (al hacer un *mumap*).

iii. Los marcos de página compartidos, no se deberían contabilizar en el cómputo del límite del proceso. La expulsión de estos marcos afecta al conjunto residente de otro proceso, que puede ser de prioridad mayor. Además, esos marcos de página aunque se dejen de referencial desde el mapa de memoria del proceso seguirían estando residentes (no implica liberación alguna de recursos). Se podría dar la circunstancia de que un proceso muy prioritario y otro con muy poca prioridad ejecuten el mismo código. El proceso de alta prioridad no ha recuperado su cupo de marcos de página, pero el otro sí. Si el proceso de poca prioridad causa un fallo de página puede elegir expulsar una de las páginas de código que el otro proceso está también usando (sería la más antigua de ellas, pero el otro proceso la puede seguir usando con cierta frecuencia). Si se expulsa esa página se daría el ridículo caso de que si el proceso de alta prioridad no compartiese nada con nadie dispondría de todos esos marcos de página, pero que si los comparte con procesos de baja prioridad puede verse privado de esos marcos.

    Sin embargo existe un problema derivado de no contabilizarlos cuando está compartidos. Este problema se plantea cuando el marco deja de estar en ese estado de compartido. En ese momento la contabilidad de marcos asignados al proceso se incrementa en todos los marcos antes compartidas, eso implica que un proceso puede ver incrementado el contado de marcos de página privados que usa, en el momento que el contador de referencias de uno esos marcos compartidos llegue a 1. Esto puede deberse, por ejemplo, a que un proceso que compartía una página de datos con COW, escriba en ella. En ese caso podría requerir activar el algoritmo de reemplazo de forma local sobre este proceso, para hacerle cumplir el límite de marcos de página. Lo mismo ocurriría, aunque no de forma tan traumática, cuando un marco privado se comparte. En este caso bastaría con reducir el contador de marcos privados, sin requerir acción alguna. Una solución de compromiso es aplicar de forma estricta el límite para las operaciones que se deben al propio proceso (sus llamadas al sistema o el tratamiento de un fallo de página provocado por él) y relajar esa condición para los eventos que provocan otros procesos. El cumplimiento del límite se puede entonces delegar al demonio de paginación, para alimentar la caché de páginas, y haciéndose en *background*.

    Otro efecto singular de esta política es el desalojo de páginas compartidas. Al no estar en la contabilidad propia de cada proceso, no se ven afectadas por los algoritmos de reemplazo locales. Esto hace que una página compartida que ocupa un marco lo siga haciendo mucho

después de ser utilizada. Este problema sólo se da si no se ejecuta nunca un reemplazo global y la memoria está completamente ocupada. Sin embargo, esta situación sólo se da si no se crean nuevos procesos y los procesos no cambian de prioridad. Por contra, el sistema está dotado de un demonio de paginación, este demonio puede ejecutarse con ámbito local o global. En este último caso, sería el encargado de ir liberando estas páginas antiguas y de ir cargando la cache y el buffering de páginas.

d) La herencia de prioridad está pensada, principalmente, para que el proceso más prioritario se pueda desbloquear cuanto antes. Si el proceso de baja prioridad cuando ha heredado la prioridad de otro proceso requiere más memoria (según los casos visto antes) y tiene que aplicar un reemplazo local, se pueden producir mayores demoras (escritura de páginas a expulsar). Sin embargo, si se le permite incrementar su consumo de memoria en esos momentos, entonces dispondría de las páginas de la *caché/buffering* de páginas, con la posibilidad de un mejor rendimiento.

La pega que se debe al hecho de modificar varias veces la prioridad del proceso esto implicaría sólo un problema cuando la prioridad se reduce, reduciendo en ese momento su cuota de marcos de página. Sin embargo, esa situación se dará en el momento en el que un proceso de mayor prioridad se ha desbloqueado. Y en cualquier caso el demonio de paginación se encargaría de volver hacer cuadrar las cuentas.