DISEÑO DE SISTEMAS OPERATIVOS 4º. Junio de 2005

Ejercicio 1

Responda razonadamente a las siguientes preguntas sobre los cambios de contexto.

a) El cambio de contexto, ¿es sólo cosa de dos? Para analizar esta cuestión, se han incluido en la función que encapsula el cambio de contexto algunas sentencias que imprimen información de los procesos involucrados:

```
BCP *proceso_anterior, *proceso_posterior;
cambio (BCP *previo, BCP *siguiente) {
        printk("previo %d siguiente %d \n", previo->pid, siguiente->pid);
        proceso_anterior = previo; proceso_posterior = siguiente;
        cambio_contexto(&previo->contexto, &siguiente->contexto);
        printk("previo %d siguiente %d proceso_anterior %d proceso_posterior %d\n",
                        previo->pid, siguiente->pid, proceso_anterior->pid, proceso_posterior->pid);
}
```

Suponiendo que se ejecutan continuamente y de forma cíclica, mediante *round-robin*, tres procesos P1 (*pid*=1), P2 (*pid*=2), P3 (*pid*=3), en ese orden, ¿qué información se imprimiría durante el intervalo de tiempo que va desde que P1 vuelve a ejecutar, después de que P3 le ceda el control, hasta el momento en que nuevamente P3 le va a ceder el control (es decir, un ciclo completo de ejecución)?

b) ¿Qué acciones sobre la TLB, la memoria cache y la tabla de páginas se realizan durante un cambio de contexto? Suponiendo que se pretende que estas acciones se realicen en el momento en que se reanuda la ejecución del proceso (es decir, que las realice el propio proceso activado en el cambio de contexto), ¿en qué punto de la rutina *cambio* se incluirían y a qué proceso se aplicarían? Para ello, especifique el nombre de la variable (*proceso_anterior*, *proceso_posterior*, *previo* o *siguiente*) que haga referencia al proceso involucrado.

c) Suponiendo que se trata de un sistema con *threads*, ¿cómo afectaría a las acciones descritas en el apartado anterior la circunstancia de que los dos *threads* involucrados en un cambio pertenezcan al mismo proceso? Incluya en la rutina *cambio*, en el punto especificado en el apartado anterior, el código que hace la comprobación de esta circunstancia mostrando el nombre de las variables involucradas en esta comprobación.

d) ¿Qué tipo de núcleo puede generar más cambios de contexto: uno expulsivo o uno que no lo es? Distinga entre cambios voluntarios e involuntarios y explíquelo con un ejemplo. Nótese que no se trata de analizar en qué momento se lleva a cabo el cambio de contexto en cada tipo de núcleo, sino de calcular cuántos se producen en total, es decir, plantear situaciones en las cuales el número total de cambios de contexto que se dan con un tipo de núcleo es mayor que con el otro.

e) De los dos mecanismos usados en los núcleos expulsivos para evitar los problemas de sincronización entre llamadas al sistema concurrentes, ¿cuál puede generar más cambios de contexto? Distinga entre cambios voluntarios e involuntarios y explíquelo con un ejemplo.

Solución

a) Para responder a esta cuestión hay que tener en cuenta la manera en que se lleva a cabo un cambio de contexto. Cuando un proceso quiere ceder el procesador y ejecuta la rutina cambio, su ejecución queda detenida en la operación de cambio de contexto, guardándose su estado de manera que, cuando en su momento reanude su ejecución, lo haga justo a partir de ese punto, completando la segunda parte de la rutina cambio.

Otro aspecto que hay que tener en cuenta es cómo afecta el cambio de contexto a los datos que gestiona el sistema operativo. Concretamente, dado que la ejecución de la rutina cambio no se hace de manera continua en el tiempo, sino que puede transcurrir un intervalo de tiempo impredecible hasta que se reanude la ejecución del proceso y se complete la rutina, ¿qué habrá ocurrido con los valores de las variables locales de la rutina, así como con las variables globales usadas en la misma?, ¿seguirán manteniendo sus valores cuando se reanude la ejecución del proceso en el ámbito de la segunda parte de la rutina cambio? Para responder a estas preguntas, hay que observar que las variables locales de la rutina se almacenan en la pila de sistema del proceso que la ejecuta, mientras que las variables globales del sistema operativo se almacenan en la región de datos del mismo. Por tanto, dado que como parte de un cambio de contexto se deja de usar la pila de sistema del proceso saliente para comenzar a utilizar la del proceso entrante, el valor de las variables locales al reanudarse la ejecución de la rutina será el mismo que tenían antes del cambio de contexto. No ocurre así con las variables globales cuyo valor corresponderá con su última actualización, que, evidentemente, podrá haber sido posterior al cambio de contexto. Teniendo en cuenta estas consideraciones, la salida generada por la traza de ejecución planteada en el enunciado sería la siguiente:

1. P1 reanuda su ejecución: previo 1 siguiente 2 proceso_anterior 3 proceso_posterior 1
2. P1 invoca cambio(P1, P2): previo 1 siguiente 2
3. P2 reanuda su ejecución: previo 2 siguiente 3 proceso_anterior 1 proceso_posterior 2

4. P2 invoca cambio(P2, P3): previo 2 siguiente 3
5. P3 reanuda su ejecución: previo 3 siguiente 1 proceso_anterior 2 proceso_posterior 3
6. P2 invoca cambio(P3, P1): previo 3 siguiente 1

Nótese que a continuación se repetiría el paso 1 y así sucesivamente. Si se observa una ejecución completa de la rutina cambio por parte de un proceso (por ejemplo, el proceso P1), se puede apreciar que realmente están involucrados tres procesos (el cambio de contexto es cosa de tres): el proceso que invoca a la rutina para ceder el procesador (identificado por previo, siendo P1 en el ejemplo), el proceso al que se le cede el control (identificado por siguiente, siendo P2 en el ejemplo), y el proceso que posteriormente le devolverá el control al proceso que invocó la rutina cambio para que complete su ejecución (identificado por proceso_anterior, siendo P3 en el ejemplo). En el apartado *c* se incidirá más en este aspecto.

b) Distingamos entre los tres elementos de gestión de memoria planteados en el enunciado:

- TLB. Si se trata de un procesador en el que las entradas de la TLB no almacenan información del proceso al que corresponde dicha entrada, será necesario invalidar toda la TLB puesto que la información de traducción que contiene ya no será válida para el próximo proceso. En caso de que cada entrada contenga información del proceso al que corresponde esa traducción, no será necesaria esta operación.
- Memoria cache. En el caso más habitual de que sea una memoria cache que usa direcciones físicas, no habría que realizar ninguna operación específica sobre la cache durante el cambio de proceso, ya que la información contenida en la misma sigue siendo válida después del cambio. Si se tratase de una memoria cache que usa direcciones virtuales, habría que invalidarla ya que esas direcciones dejan de tener sentido para el proceso que entra a ejecutar.
- Tabla de páginas. Cada proceso usa su propia tabla de páginas. Por tanto, en un cambio de contexto hay que informar a la MMU de que tiene que usar otra tabla de páginas: la del proceso al que se le cede el procesador. En algunos procesadores, esta operación consiste simplemente en escribir la dirección de dicha tabla en un registro de la MMU.

Nótese que estas operaciones se podrían realizar tanto antes como después de la invocación de la rutina cambio_contexto. En caso de hacerlo antes, se estará ejecutando en el contexto del proceso que cede el procesador (previo). Por tanto, la tabla de páginas que hay que instalar será la correspondiente al proceso al que hace referencia la variable siguiente. Si, por el contrario, se opta por realizar las operaciones después de la llamada, corresponderá con hacerlo en el contexto del proceso que reanuda su ejecución, que es lo que se plantea en el enunciado. En este caso, la variable correspondiente a este proceso es previo (o proceso_posterior), con lo que la función cambio quedaría:

```
BCP *proceso_anterior, *proceso_posterior;
cambio (BCP *previo, BCP *siguiente) {
        printk("previo %d siguiente %d \n", previo->pid, siguiente->pid);
        proceso_anterior = previo; proceso_posterior = siguiente;
        cambio_contexto(&previo->contexto, &siguiente->contexto);
        invalidar_TLB();
        instalar_tabla_paginas(previo);
        printk("previo %d siguiente %d proceso_anterior %d proceso_posterior %d\n",
                        previo->pid, siguiente->pid, proceso_anterior->pid, proceso_posterior->pid);
}
```

c) En el caso de un sistema con *threads*, cuando se produce un cambio entre *threads* del mismo proceso, no es necesario realizar ninguna operación de coherencia en ninguno de los tres elementos de gestión de memoria planteados en el enunciado puesto que los *threads* ejecutan en el ámbito del mismo mapa de memoria: se usa la misma tabla de páginas, las traducciones en la TLB siguen siendo válidas, y, evidentemente, el contenido de la memoria cache continúa siendo correcto. Precisamente, el ahorrarse estas operaciones es lo que hace principalmente que los cambios entre *threads* del mismo proceso sean más ligeros.
Para incluir en la rutina cambio la comprobación de si se trata de *threads* del mismo proceso en el mismo punto que en el apartado anterior, tal como plantea el enunciado, habría que hacer referencia a las variables que representan al proceso que reanuda su ejecución (previo o proceso_posterior) y al que deja el procesador (proceso_anterior), que generalmente no será el mismo al que se le cedió el procesador en la ejecución de la primera parte de la rutina (al que hace referencia la variable siguiente). Esta situación ilustra claramente que en el cambio de contexto, visto de forma global, intervienen tres procesos. La rutina cambio quedaría de la siguiente forma:

```
BCT *proceso_anterior, *proceso_posterior;
cambio (BCT *previo, BCT *siguiente) {
```

```
        printk("previo %d siguiente %d \n", previo->pid, siguiente->pid);
        proceso_anterior = previo; proceso_posterior = siguiente;
        cambio_contexto(&previo->contexto, &siguiente->contexto);
        if (proceso_posterior->proceso != proceso_anterior->proceso) {
                invalidar_TLB();
                instalar_tabla_paginas(previo);
        }
        printk("previo %d siguiente %d proceso_anterior %d proceso_posterior %d\n",
                        previo->pid, siguiente->pid, proceso_anterior->pid, proceso_posterior->pid);
}
```

d) A primera vista, podría parecer que en ambos tipos de núcleos se producen el mismo número de cambios de contexto. La única diferencia aparente es que en el núcleo no expulsivo el cambio de contexto involuntario tiene que diferirse hasta que termine la llamada al sistema en curso, en caso de que la hubiera, pero finalmente acabará realizándose, con lo que se producirá el mismo número de cambios de contexto. Sin embargo, en un núcleo no expulsivo, durante el intervalo de tiempo que transcurre desde que se produce el evento que marca la necesidad de un cambio de contexto involuntario hasta que se produce el cambio de contexto, pueden suceder eventos adicionales que modifiquen de alguna manera el cambio de contexto pendiente. A continuación, se analiza este aspecto mediante ejemplos.

Supóngase que, mientras un proceso de baja prioridad está realizando una llamada al sistema de duración apreciable, se produce una interrupción que desbloquea a un proceso de prioridad media, y, poco después, llega una segunda interrupción que desbloquea a un proceso de prioridad alta. A continuación, se analiza qué ocurriría en cada tipo de núcleo.

En un núcleo no expulsivo, el tratamiento de la primera interrupción activaría una interrupción software ya que se requiere un cambio de contexto involuntario al proceso de prioridad media. Sin embargo, al acabar la rutina de interrupción no se dispararía la interrupción software ya que tiene menor prioridad que la llamada al sistema. Mientras prosigue la llamada al sistema del proceso de prioridad baja se produce una segunda interrupción que también implica un cambio de contexto involuntario, ya que el proceso desbloqueado tiene prioridad alta. Nuevamente, no se se dispararía la interrupción software y continuaría la llamada. Al final de la misma, se dispararía la interrupción software que cedería el procesador al proceso de prioridad alta (cambio de contexto involuntario). Cuando terminase la ejecución del proceso de prioridad alta le cedería el procesador al de prioridad media (cambio de contexto voluntario), que, a su vez, al terminar, se lo cedería al proceso de prioridad baja (cambio de contexto voluntario). En total, se producen tres cambios de contexto: 2 voluntarios y 1 involuntario.

En un núcleo expulsivo, el tratamiento de la primera interrupción también activará una interrupción software, pero en este caso, en cuanto termine el tratamiento de la interrupción, se disparará la interrupción software dejando sin completar la llamada al proceso de baja prioridad y cediendo el procesador al proceso de prioridad media (cambio de contexto involuntario). A continuación, llegará la segunda interrupción, que, de manera similar, provocará un cambio de contexto involuntario al proceso de alta prioridad. El resto de la traza de ejecución es similar a la que se produce con un núcleo no expulsivo: el proceso de prioridad alta termina su ejecución haciendo un cambio de contexto voluntario al proceso de prioridad media que, al terminar, realiza un cambio voluntario al proceso de prioridad baja (que tendrá, en este caso, que completar la llamada). Por tanto, se producen 4 cambios de contexto: 2 voluntarios y 2 involuntarios.

En conclusión, se han producido más cambios de contexto en el núcleo expulsivo, concretamente, de tipo involuntario.

Se podrían plantear otros ejemplos donde se aprecia esta misma situación. Por ejemplo, si se produce una interrupción que desbloquea a un proceso más prioritario mientras otro está terminando (es decir, ejecutando la llamada al sistema que causa su terminación), con un núcleo expulsivo habrá un cambio de contexto involuntario que no se produciría en un núcleo no expulsivo que permite que se complete la llamada y se ceda el procesador de forma voluntaria (nótese que este último cambio de contexto voluntario también se produciría con el núcleo no expulsivo cuando se reanudase la ejecución de la llamada que termina el proceso).

Resumiendo, en un núcleo expulsivo se pueden producir más cambios de contexto que en uno no expulsivo. Estos cambios de contexto adicionales serán de carácter involuntario, ya que los cambios voluntarios son intrínsecos a la ejecución del programa y no dependen del tipo de núcleo, aunque en el siguiente apartado se hará algún comentario adicional sobre este aspecto.

Por último, téngase en cuenta que la aparición de cambios de contexto involuntarios adicionales es parte del precio que hay que pagar por la mejor latencia que ofrecen los núcleos expulsivos.

e) Se usan dos mecanismos en los núcleos expulsivos para evitar los problemas de sincronización entre llamadas al sistema concurrentes:

- Elevar el nivel de interrupción de manera que no se permita que se dispare la interrupcion software (aunque sí el resto de las interrupciones) mientras se ejecuta la sección crítica dentro de la llamada al sistema. Esta solución es la recomendable en el caso de secciones críticas muy cortas, ya que en caso de que éstas tengan una duración apreciable, el núcleo tiende a perder su carácter expulsivo.

- Utilizar algún mecanismo de sincronización, como semáforos o mutex, para controlar el acceso a la sección crítica. Es la estrategia más apropiada excepto si la sección crítica es muy corta, ya que en ese caso puede causar una sobrecarga inadmisible.

Dado que con la primera solución el núcleo se convierte en no expulsivo durante la sección crítica, mientras que con la basada en semáforos no pierde su carácter expulsivo, se pueden aplicar los razonamientos vertidos en la sección anterior y concluir que con la segunda estrategia se puede producir un número mayor de cambios de contexto involuntarios.
Por otra parte, la presencia de los semáforos provoca un mayor número de cambios de contexto voluntarios que con la solución basada en elevar el nivel de interrupción, como se puede apreciar en el siguiente ejemplo.
Supóngase que un proceso de baja prioridad está ejecutando una sección crítica y en ese instante se produce una interrupción que desbloquea a un proceso de mayor prioridad que intentará acceder a la sección crítica. A continuación, se analiza qué ocurriría en cada solución.
En la solución basada en elevar el nivel de interrupción durante la sección crítica, el tratamiento de la interrupción activaría una interrupción software ya que se requiere un cambio de contexto involuntario al proceso de mayor prioridad. Sin embargo, al acabar la rutina de interrupción no se dispararía la interrupción software ya que está inhibida dentro de la sección crítica. Cuando termina la sección crítica, al permitirse de nuevo la interrupción software, ésta se dispara realizándose el cambio de contexto involuntario al proceso de mayor prioridad que podrá acceder sin problemas a la sección crítica.
En la solución basada en semáforos, el tratamiento de la interrupción activaría también una interrupción software, pero en este caso, en cuanto termine el tratamiento de la interrupción, se disparará la interrupción software cediendo el procesador al proceso de mayor prioridad media (cambio de contexto involuntario). El proceso de mayor prioridad intentará acceder a la sección crítica, quedándose bloqueado en el semáforo (cambio de contexto voluntario) cediendo el procesador al proceso de baja prioridad. Éste completará la sección crítica liberando el semáforo, lo que desbloquea al proceso de mayor prioridad, causando nuevamente un cambio de contexto involuntario, el cual podrá acceder sin problemas a la sección crítica.
En resumen, con la solución basada en elevar el nivel de interrupción sólo se produce un único cambio de contexto que es de tipo involuntario, mientras que con la basada en semáforos se producen 3: 2 involuntarios y 1 voluntario. Nótese que no sólo ha aumentado el número de cambios involuntarios, sino también los de tipo voluntario.
Recapitulando y retomando el comentario final del apartado anterior, toda esta sobrecarga de cambios de contexto es el coste de tener un núcleo con un comportamiento expulsivo al máximo grado posible.