Problema 2.11 (febrero de 2010)

El ejecutable P con montaje dinámico utiliza las bibliotecas B1 y B2. Las características de estos ficheros son las siguientes:

	P	***B1***	***B2***
Sección de código	*2,7 KB*	*3,5 KB*	*4,7 KB*
Datos con valor inicial	*724 B*	*1,7 KB*	*123 B*
Datos sin valor inicial	*1004 B*	*2,3 KB*	*1,2 KB*
Símbolos exportados	□	*12*	*32*

*Supóngase que se trabaja con un sistema **biprocesador** con las siguientes características:*

- *Sistema paginado con páginas de 1 KB y mapa de memoria individual por proceso.*
- *Tabla de páginas de 3 niveles.*
- *Direcciones de 32 bits, divididas de la siguiente forma: 1er nivel 6 bits, 2º nivel 8 bits y 3er nivel 8 bits.*
- *Montaje dinámico con carga de bibliotecas al crear la imagen del proceso y con resolución de símbolos en carga.*
- *Núcleo expulsivo con un esquema de planificación basado en prioridades, con una única cola de procesos listos y sin usar afinidad al procesador.*
- *Dispone de una IPI (InterProcessor Interrupt).*
- *Implementación de threads basada en procesos de peso variable (similar a la de Linux).*

Responda a las siguientes preguntas:

a) Dibuje las regiones que tiene la imagen de memoria inicialmente, especificando: dirección de comienzo, dirección final y características de cada una de ellas. Expresar las direcciones en hexadecimal.

b) Determine las estructuras necesarias para el acceso a las bibliotecas dinámicas, así como su ubicación en la imagen de memoria.

c) Suponiendo que la bolsa de marcos de páginas libres contiene los marcos 0000A1, 0000A2, 0000A3, 0000A4, 0000A5… y que se van asignando según este orden, dibuje la parte de la tabla de páginas relativa al código del programa P y a sus datos (no es necesario incluir las bibliotecas ni la pila) en el momento del arranque, indicando el contenido de sus campos.

d) Considerando que el programa comienza ejecutando el siguiente fragmento, que convierte a mayúsculas los caracteres de un fichero de 4KB (t=4096), dibuje los elementos de la tabla de páginas que han sido modificados o creados al finalizar el mismo:

```
char *p = mmap(NULL, t, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
for (i=0; i<t; i++, p++)
   if (islower(*p))  *p = toupper(*p);
```

e) Suponga que, justo a continuación, el proceso ejecuta la siguiente secuencia de operaciones: crea un hijo (`fork`) y tanto el proceso padre (PP) como el hijo (PH) crean un thread (TP y TH, respectivamente). Explique qué estructuras de gestión de memoria se crean al ejecutar esa secuencia y cómo se construyen esas estructuras a partir de las que tenía el proceso original antes de comenzar dicha secuencia de operaciones. Dibuje los bloques de control de los procesos descritos, incluyendo las referencias a esas estructuras de gestión de memoria. ¿Por qué motivo algunos sistemas implementan los threads usando procesos de peso variable como ocurre en el ejemplo planteado en este ejercicio?

f) Especifique la traza de ejecución de la secuencia planteada a continuación, identificando las activaciones del sistema operativo, así como los cambios de contexto, distinguiendo entre voluntarios e involuntarios.

- *La prioridad de los procesos es, de mayor a menor: TP, TH, PP, PH.*
- *En el punto inicial considerado en la traza, TP y TH están ejecutando en modo usuario en los procesadores P1 y P2, respectivamente; PP y PH están listos para ejecutar después de haber sido expulsados previamente: PP fue expulsado cuando estaba en modo usuario, mientras que PH estaba en medio de una llamada getpid.*
- *TP realiza una llamada para leer de una tubería vacía. Cuando, más adelante, haya datos y se complete la llamada, realizará cálculos en modo usuario durante un cierto tiempo para, finalmente, terminar (pthread_exit).*
- *TH realiza cálculos en modo usuario durante un cierto tiempo y luego termina (pthread_exit), un poco después de que lo haya hecho TP. Al principio de esos cálculos, pero un cierto tiempo después de que TP se haya bloqueado, produce un fallo de página al acceder a una página de código. La interrupción del disco asociada al servicio de ese fallo de página llegará al procesador P1 algún tiempo después de la escritura en la tubería (véase PP).*
- *Cuando PP reanuda su ejecución, realiza una primera etapa de cálculos de usuario, para, a continuación, un cierto tiempo después de que TH se haya bloqueado, escribir en la tubería. Después, ejecuta indefinidamente llamadas getpid (asúmase que cualquier evento asíncrono que afecte a PP lo encontrará siempre en modo sistema).*
- *Cuando PH reanuda su ejecución, está ejecutando indefinidamente llamadas getpid (asúmase que cualquier evento asíncrono que afecte a PH lo encontrará siempre en modo sistema).*

¿En qué podría cambiar la traza si el esquema de planificación tuviera en cuenta la afinidad al procesador?

Solución

a) La figura 2.3 muestra una posible solución para la imagen de memoria. Observe los siguientes aspectos:

- Dado que es un sistema paginado, cada región ha de tener una entrada en la tabla de primer nivel. Esto supone que las direcciones de comienzo de las regiones son 0, 2^{26}, $2 \cdot 2^{26}$, $3 \cdot 2^{26}$, $4 \cdot 2^{26}$..., o, en hexadecimal, 00000000, 04000000, 08000000, 0C000000, 10000000, …
- Nos dicen que el montaje dinámico es con carga de bibliotecas al crear la imagen del proceso, por lo que la imagen inicial incluye las bibliotecas.

- dado que tienen la misma protección y que son todos ellos privados, se ha considerado una única región que incluye los datos con valor inicial, sin valor inicial y heap. Igualmente, para las bibliotecas se ha incluido una única región de datos. Sin embargo, los datos con valor inicial tienen como soporte el ejecutable, mientras que los datos sin valor inicial no tienen soporte. Por tanto, no se pueden mezclar en una misma página datos con y sin valor inicial. Esto hace que se necesiten las siguientes páginas:

	Programa P	**Biblioteca 1**	**Biblioteca 2**
Datos c.v.i	1 página	2 páginas	1 página
Datos s.v.i	1 página	3 páginas	2 páginas
Heap	p.e. 3 páginas		

- Tanto la región de texto del programa P como su región de datos tienen su ubicación fija, definida en el ejecutable. Se ha supuesto que estas ubicaciones son la 00000000 y la 04000000.

- Se ha puesto la pila al final del mapa de memoria y se ha considerado que crece según direcciones decrecientes.

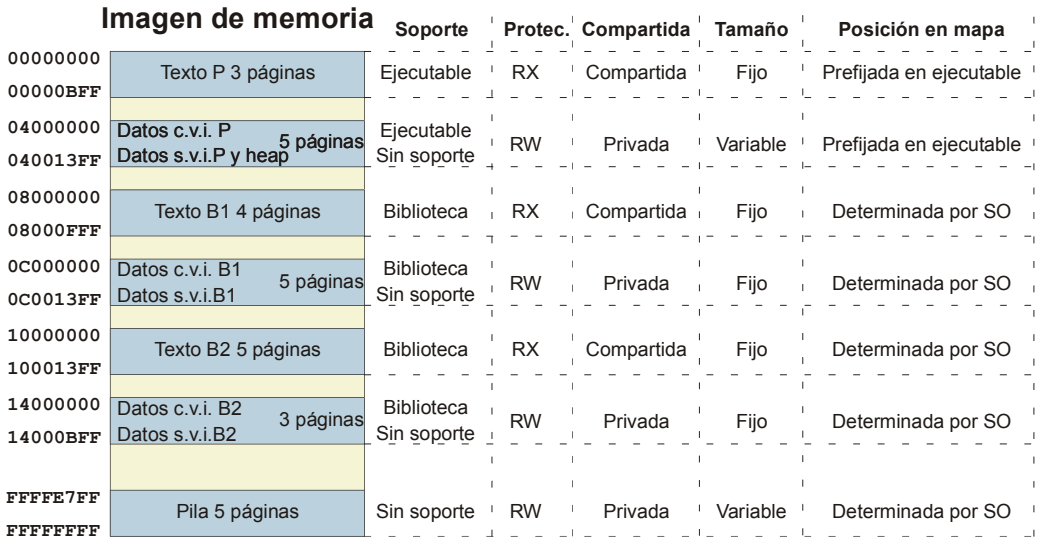

Figura 2.3:

b) Dado que el montaje dinámico es con resolución de símbolos en carga, se necesita incluir los saltos a los símbolos en la región de texto. En la figura 2.4 se pueden observar dichos saltos, que se han incluido al final del texto. Además, es necesario incluir las indirecciones de dichos saltos. Como dichas indirecciones dependen de la imagen de memoria del proceso, han de estar en su zona de datos. En la figura 2.4 se observa que se han incluido al comienzo de los datos sin valor inicial. Si se pusiesen al principio o final de los datos con valor inicial habría que llevar a memora la correspondiente página del ejecutable, dado que dichos datos tienen a éste como soporte.

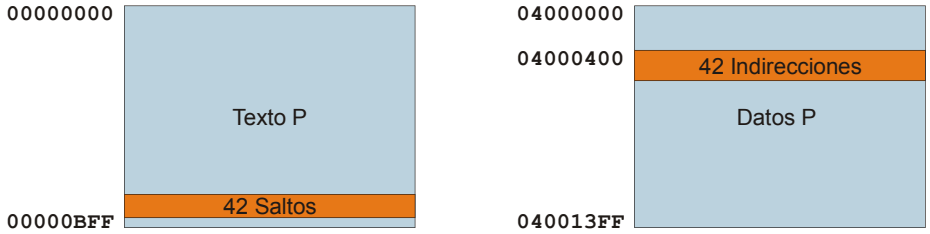

Figura 2.4:

c) Existen dos posibilidades, creación perezosa de la tabla de páginas o no perezosa. La figura 2.5 muestra la tabla completa al principio de la ejecución para creación no perezosa. Observe que los únicos marcos asignados inicialmente corresponden a:

- La pila, puesto que el SO incluye en ella el entorno al crear la imagen de memoria del proceso.
- Las indirecciones de las bibliotecas dinámicas.

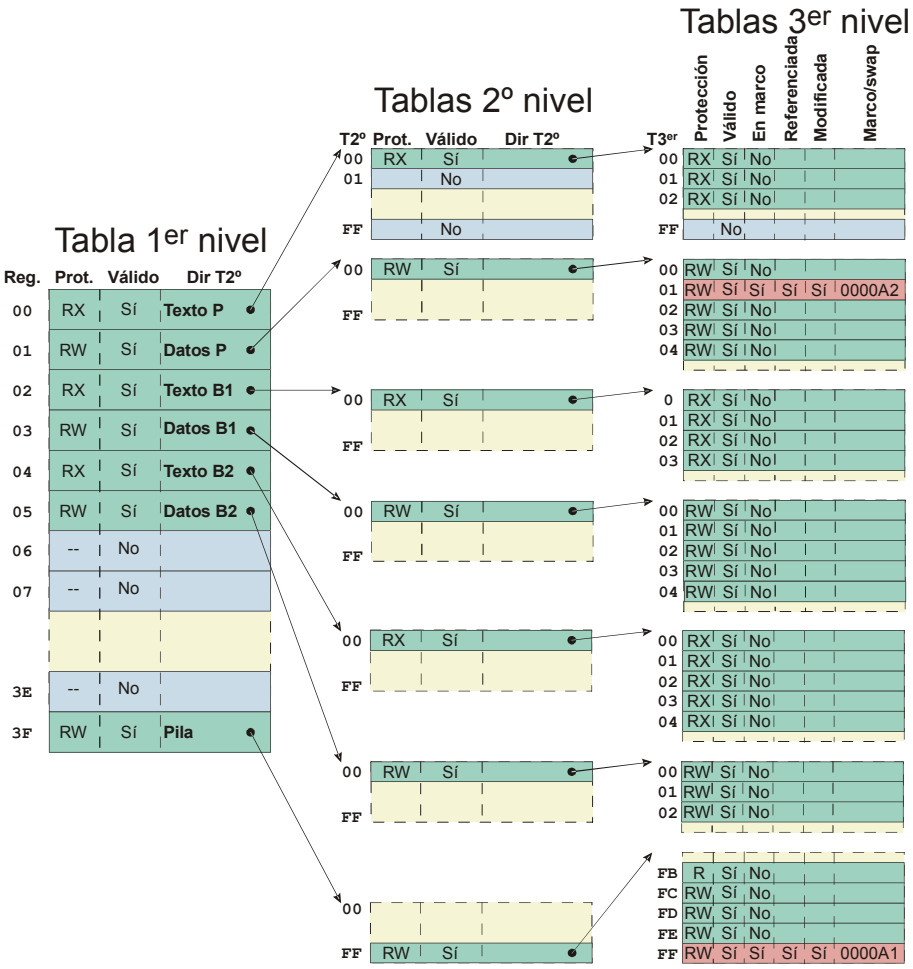

Figura 2.5:

La figura 2.6 muestra la tala de páginas para el caso perezoso. Se observará que solamente se construyen los elementos necesarios, esto es:

- Tabla de primer nivel. En esta tabla están marcadas como inválidas todas las entradas menos la de datos de P y la de la pila. Al producirse una violación de memora, el SO comprueba si se trata de una acceso permitido, pero cuya tabla de páginas no ha sido construida. En caso afirmativo la construye y, ademas, asigna un marco de página para la página referenciada.
- Tabla de 2º y 3er nivel de los datos de P. Estas tablas son necesarias, puesto que las indirecciones se calculan en el momento de la carga.
- Tabla de 2º y 3er nivel de la pila. Estas tablas son necesarias, puesto que la pila tiene uno o varios marcos al comienzo de la ejecución.
- La página FB de la pila tiene solamente permisos de lectura, de forma que si, al crecer la pila, se llega a escribir en ella, se produce un fallo de escritura, de forma que el SO sabe que la pila ya está casi llena, por lo que procederá a incrementarla.

d) Cuando comienza la ejecución del proceso se producen fallos de página y se irán asignado los correspondientes marcos. La figura 2.7 muestra la tabla en el momento solicitado. Hay que notar los siguientes aspectos:

- Se ha considerado que solamente se ha utilizado la primera página del programa P y que no se ha utilizado ninguna de las dos bibliotecas.
- Se ha considerado que la variable t tiene valor inicial.
- En caso de política perezosa no existirían todavía las cuatro tablas de 2º y 3er nivel de las bibliotecas dinámicas.

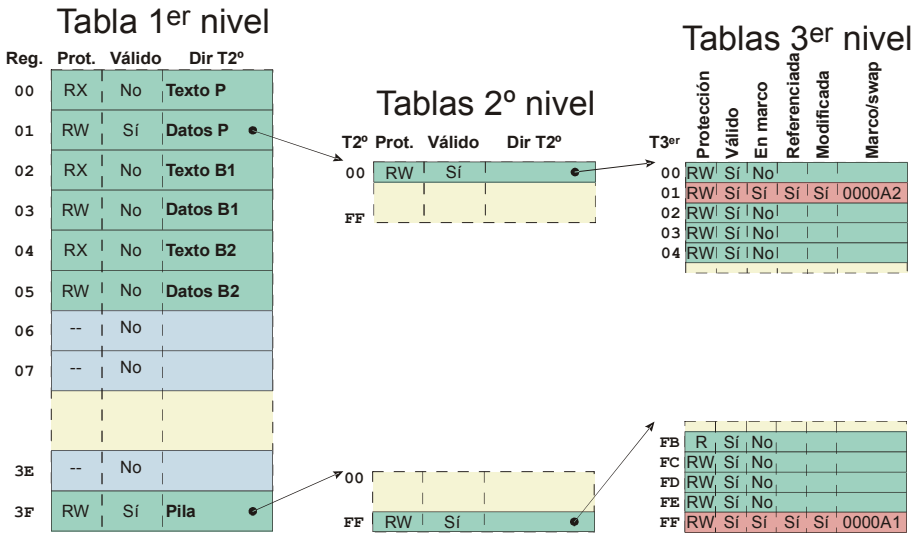

Figura 2.6:

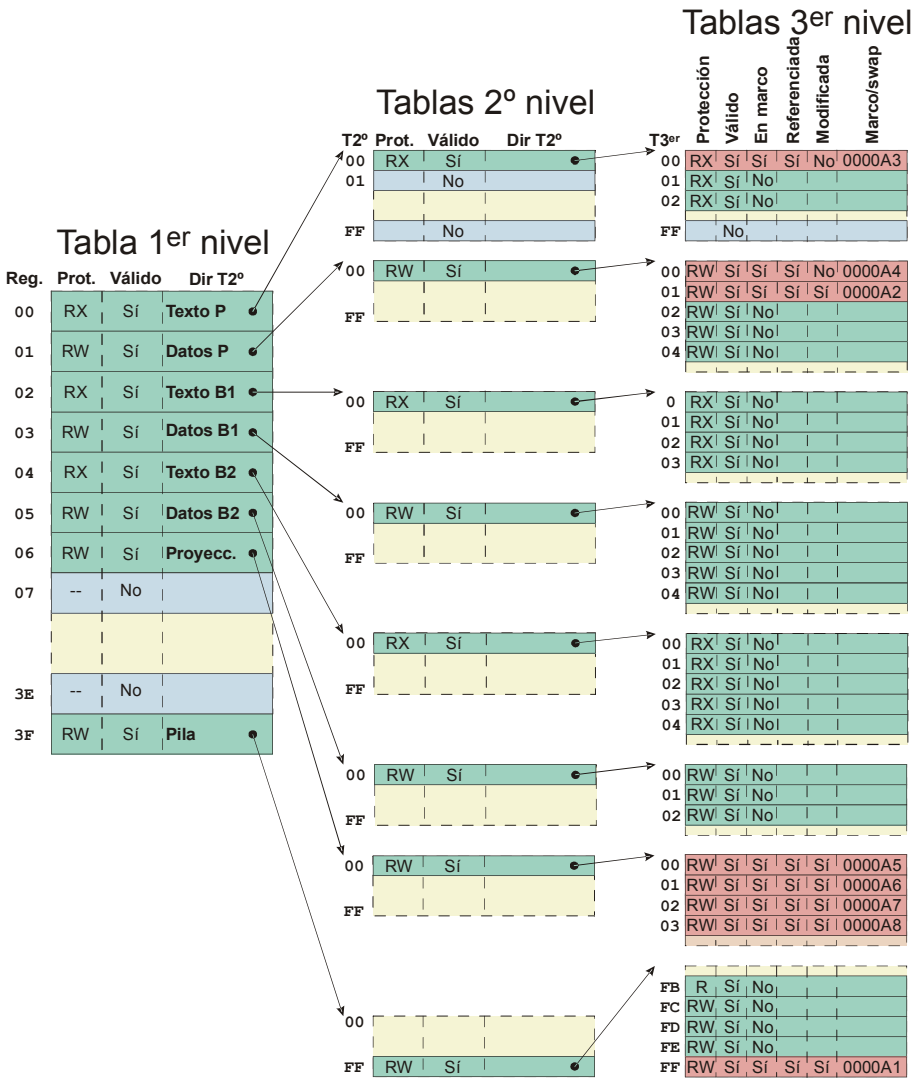

Figura 2.7:

e) Comencemos por la pregunta final de esta cuestión: ¿por qué motivo algunos sistemas operativos, como Linux, implementan los *threads* usando procesos de peso variable en vez de hacerlo de forma nativa (es decir, incluyendo explícitamente esta nueva abstracción en el sistema operativo)?

Básicamente, se trata de una cuestión de reducir el impacto que genera la inclusión de una nueva abstracción en un sistema ya existente. La incorporación del concepto de *threads* en el diseño e implementación de un sistema que fue concebido sin tener en cuenta esta nueva abstracción requiere el rediseño de una gran parte del sistema, dado el profundo impacto que tiene en las diversas partes del sistema operativo. Por ello, algunos sistemas han optado por un enfoque alternativo. En vez de implementar dos abstracciones (el proceso como contenedor de un contexto de ejecución y el *thread* como expresión de un flujo de ejecución), se mantiene sólo la abstracción de proceso, pero extendiendo su comportamiento, de manera que se pueda controlar el grado de compartimiento de recursos entre los procesos (el proceso hijo puede usar un duplicado de un recurso del padre, que representa la opción convencional, pero también puede compartirlo, gracias a esta extensión). Este esquema, que tiene mucho menos impacto sobre el diseño original del sistema, permite implementar una funcionalidad de *threads* sin necesidad de incorporar explícitamente esa abstracción. Además de dar soporte a *threads*, esta solución basada en procesos de peso variable permite crear esquemas de concurrencia novedosos, donde los procesos pueden compartir ciertos recursos pero no otros (por ejemplo, los ficheros abiertos, pero no el directorio de trabajo actual).

Centrándonos en la secuencia planteada en el enunciado, a continuación se explica cómo evolucionan las estructuras de datos de gestión de memoria de los procesos involucrados en la misma:

- *PP* realiza una llamada `fork`: Se crea una copia tanto de la tabla de regiones como de la de páginas de *PP*. Se reserva un BCP para el nuevo proceso (*PH*) y se inicia de manera que haga referencia a las copias de las tablas. Además, se eliminan los permisos de escritura de las páginas residentes que pertenezcan a regiones privadas tanto en la tabla de páginas de *PP* como en la *PH*, posibilitando de esta forma la técnica del COW.
- *PP* crea un *thread*: Se reserva un BCP para el proceso que representará a ese nuevo *thread*, *TP*, y se inicia de manera que haga referencia a la tabla de regiones y de páginas de *PP*.
- *PH* crea un *thread*: Se reserva un BCP para el proceso que representará a ese nuevo *thread*, *TH*, y se inicia de manera que haga referencia a la tabla de regiones y de páginas de *PH*.

La figura 2.8 muestra las relaciones entre estas estructuras de datos cuando se completa la secuencia planteada en el enunciado.

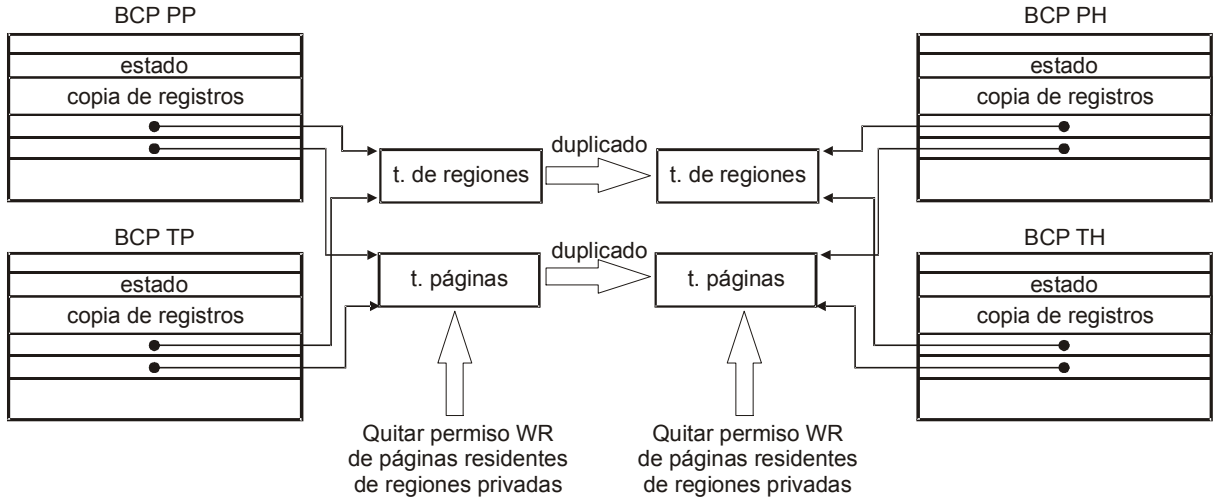

Figura 2.8

f) La figura 2.9 muestra la traza de ejecución planteada en el enunciado.

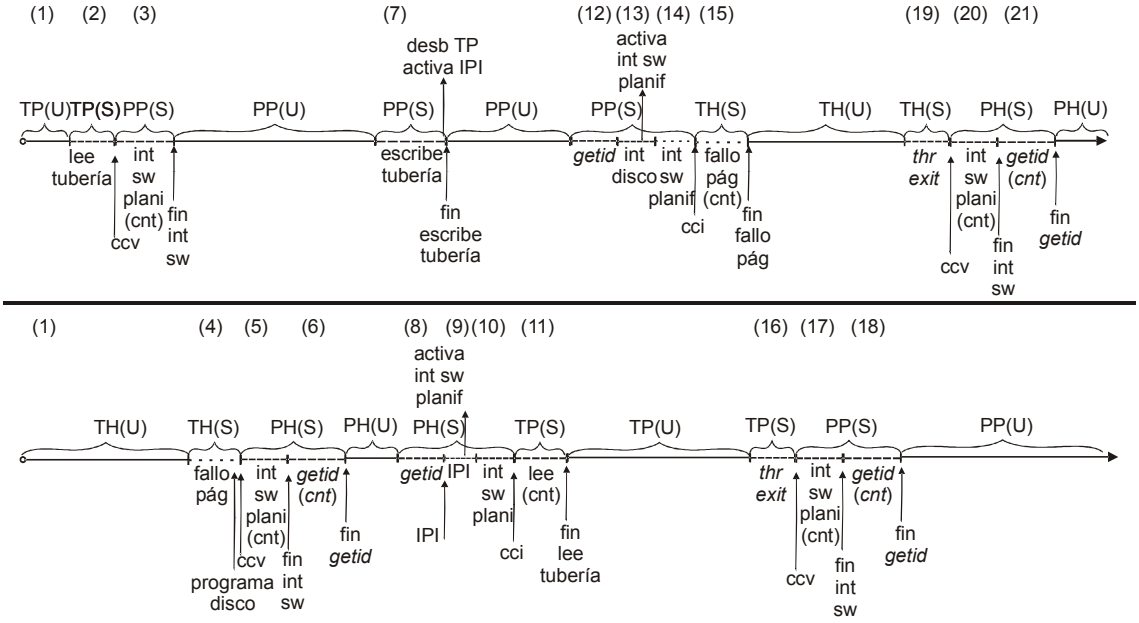

Figura 2.9

A continuación, se detallan cada uno de los eventos que ocurren durante la traza:

1. En este punto inicial *TP* está ejecutando en modo usuario en el procesador *P1* y *TH* está ejecutando en el mismo modo en el procesador *P2*.
2. *TP* realiza una llamada al sistema solicitando leer de una tubería. Dado que la tubería está vacía, se bloquea (cambio de contexto voluntario), cediendo el procesador al proceso *PP*, que es el más prioritario entre los procesos listos para ejecutar.
3. *PP* reanuda su ejecución en el contexto de una rutina de tratamiento de una interrupción software de planificación donde quedó detenida su ejecución. Al completarla, retorna a modo usuario.
4. Poco tiempo después, en el procesador *P2*, *TH* provoca un fallo al acceder a una página de código. En el tratamiento de ese fallo, se programa el disco para leer la página y se bloquea al proceso (cambio de contexto voluntario), cediendo el procesador al proceso *PH*.
5. *PH* reanuda su ejecución en el contexto de una rutina de tratamiento de una interrupción software de planificación donde quedó detenida su ejecución. Al completarla, reanuda la ejecución de la llamada getpid justo en el punto donde quedo interrumpida.
6. Cuando *PH* completa la llamada `getpid` retorna a modo usuario.
7. Un cierto tiempo después, en el procesador *P1*, *PP* realiza una llamada al sistema solicitando escribir en una tubería. Como parte de esta llamada, se produce el desbloqueo del proceso *TP*, que estaba esperando datos en esa tubería. Al comprobar la prioridad del proceso desbloqueado, se detecta que es mayor que la que tienen los dos procesos en ejecución, por lo que se procede a la expulsión del menos prioritario, *PH*, enviándole una interrupción software de planificación. Al estar ejecutando en otro procesador, es necesario enviar previamente una IPI a ese procesador (*P2*) para forzar la expulsión del proceso. Terminadas estas operaciones, se completa la llamada que escribe en la tubería y se retorna a modo usuario.
8. Justo después del inicio de la llamada para escribir en una tubería en el procesador *P1*, en el procesador *P2* el proceso *PH* realiza la llamada `getpid`. Durante el transcurso de esta llamada, llega la IPI desde el procesador *P1*.
9. Al completarse el tratamiento de la IPI, toma control la rutina de interrupción software de planificación.
10. La rutina de interrupción software de planificación lleva a cabo el cambio de contexto involuntario requerido, con lo que *TP* continúa la ejecución de la llamada al sistema justo en el punto donde quedó detenida. Nótese que al desbloquearse, como consecuencia de una estricta aplicación de las prioridades de los procesos, *TP* ha cambiado de procesador (del *P1* al *P2*), perdiéndose la posible afinidad con el primero.
11. *TP* completa la llamada de lectura de una tubería retornando a modo usuario.
12. Algo más tarde, en el procesador *P1*, el proceso *PP* realiza la llamada `getpid`. Durante el transcurso de esta llamada, llega la interrupción del disco.
13. Durante el tratamiento de la interrupción del disco, se produce el desbloqueo del proceso *TH*, que estaba esperando por esa operación. Al comprobar la prioridad del proceso desbloqueado, se detecta

que es menor que la del proceso ejecutando en *P2* pero mayor que la del que ejecuta en ese mismo procesador, *PP*, por lo que se procede a su expulsión enviándole una interrupción software de planificación. Obsérvese que en este caso no se requiere una IPI puesto que el proceso a expulsar está ejecutando en el mismo procesador en el que se produce el evento que causa la expulsión.

14. La rutina de interrupción software de planificación lleva a cabo el cambio de contexto involuntario requerido, con lo que *PP* continúa la ejecución del tratamiento del fallo de página justo en el punto donde quedó detenida. Nuevamente, observe que al desbloquearse, como consecuencia de una estricta aplicación de las prioridades de los procesos, *TH* ha cambiado de procesador (del *P2* al *P1*), perdiéndose la posible afinidad con el primero.
15. *TH* completa el tratamiento del fallo de página retornando a modo usuario.
16. *TP* realiza una llamada al sistema para completar su ejecución. Se produce un cambio de contexto voluntario, cediendo el procesador al proceso *PP*, que es el más prioritario entre los procesos listos para ejecutar. Nótese que *PP* también ha cambiado de procesador durante su ejecución.
17. *PP* reanuda su ejecución en el contexto de una rutina de tratamiento de una interrupción software de planificación donde quedó detenida su ejecución. Al completarla, reanuda la ejecución de la llamada getpid justo en el punto donde quedo interrumpida.
18. Cuando *PP* completa la llamada `getpid` retorna a modo usuario.
19. *TH* realiza una llamada al sistema para completar su ejecución. Se produce un cambio de contexto voluntario, cediendo el procesador al proceso *PH*, que es el único proceso listo para ejecutar. Obsérvese que *PH* también ha cambiado de procesador durante su ejecución.
20. *PH* reanuda su ejecución en el contexto de una rutina de tratamiento de una interrupción software de planificación donde quedó detenida su ejecución. Al completarla, reanuda la ejecución de la llamada `getpid` justo en el punto donde quedo interrumpida.
21. Cuando PH completa la llamada `getpid` retorna a modo usuario.

Esta traza de ejecución corresponde a un esquema de planificación que no tenga en cuenta la afinidad al procesador por parte de los procesos. En este caso, es únicamente la prioridad la que determina qué procesos se están ejecutando en cada momento, no intentándose aprovechar la información de los procesos que pueda haber en el sistema de memorias caché correspondiente a sus turnos de ejecución previos.

En un esquema de planificación que sí incorporará el concepto de afinidad, se puede decidir que ejecute un cierto proceso en un determinado procesador mientras que otro de mayor prioridad no lo haga, siempre que dicho proceso tenga afinidad a ese procesador y la diferencia de prioridades esté por debajo de un determinado umbral (la afinidad puede alterar ligeramente la aplicación de las prioridades, pero no desvirtuarla).

La incorporación del concepto de afinidad al esquema de planificación afecta a los puntos de ejecución del mismo: los bloqueos (cambios de contexto voluntarios) y las expulsiones (cambios de contexto involuntarios). En estas situaciones, la afinidad al procesador puede causar la selección de un proceso que no sea el más prioritario.

Por tanto, en el caso de la traza planteada, podría afectar a cualquier punto en donde se produjera un bloqueo o un desbloqueo de un proceso. A continuación, sólo como ejemplo, se identifican dos puntos independientes en la traza que podrían verse afectados por la afinidad (el primero corresponde a un cambio de contexto involuntario y el primero a uno voluntario):

- En el punto 7, al desbloquearse *TP*, si las prioridades de *PH* y *PP* son suficientemente próximas, para satisfacer la afinidad de *TP*, cuyo último turno de ejecución se produjo en el procesador *P1*, podría expulsarse a *PP* en vez de a *PH*, si las prioridades de estos dos procesos son suficientemente próximas. Nótese que en ese caso no habría que usar una IPI.
- En el punto 16, *TP* podría ceder el procesador a PH en vez de a *PP*, si las prioridades de estos dos procesos son suficientemente próximas, puesto que en el turno de ejecución previo de estos procesos, *PH* lo hizo en el procesador *P2* mientras que *PP* usó *P1*.