Ejercicio 1. Septiembre de 2007

Se pretende desarrollar el manejador de un dispositivo de entrada que opera en modo carácter y usa interrupciones. A continuación, se muestran 4 versiones del manejador. Se trata de esquemas simplificados que ilustran alternativas de diseño, obviando diversos aspectos, como el tratamiento de los problemas de sincronización presentes en este tipo de módulos.

Versión 1

```
char *dir_buf;
tipo_cola_procesos cola_disp;
int lectura(char *dir, int tam) {
     dir_buf = dir;
     while (tam--) {
          out(R_CONTROL, LECTURA);
          Bloquear(&cola_disp);
     }
}
void interrupcion() {
     *(dir_buf++) = in(R_DATOS);
     Desbloquear(&cola_disp);
}
```

Versión 2

```
char buf;
tipo_cola_procesos cola_disp;
int lectura(char *dir, int tam) {
     while (tam--) {
          out(R_CONTROL, LECTURA);
          Bloquear(&cola_disp);
          *(dir++) = buf;
     }
}
void interrupcion() {
     buf = in(R_DATOS);
     Desbloquear(&cola_disp);
}
```

Versión 3

```
int tam_pet;
tipo_buffer_car buf;
tipo_cola_procesos cola_disp;
int lectura(char *dir, int tam) {
     tam_pet = tam;
     out(R_CONTROL, LECTURA);
     Bloquear(&cola_disp);
     while (tam--)
          *(dir++) = extraer_car(&buf);
}
void interrupcion() {
     insertar_car(&buf, in(R_DATOS));
     if (--tam_pet>0)
          out(R_CONTROL, LECTURA);
     else
          Desbloquear(&cola_disp);
}
```

Versión 4

```
int tam_pet;
tipo_buffer_car buf;
tipo_cola_procesos cola_disp;
int lectura(char *dir, int tam) {
     tam_pet = tam;
     out(R_CONTROL, LECTURA);
     while (tam--) {
          if (vacio(&buf))
            Bloquear(&cola_disp);
          while (!vacio(&buf))
            *(dir++) = extraer_car(&buf);
     }
}
void interrupcion() {
     insertar_car(&buf, in(R_DATOS));
     if ((--tam_pet==0)||(num_ele(&buf)==UMBRAL))
          Desbloquear(&cola_disp);
     if (tam_pet>0)
          out(R_CONTROL, LECTURA);
}
```

a) La primera versión es errónea. ¿Qué fallo de diseño hay en la misma?

Para comparar las otras versiones vamos a utilizar el siguiente ejemplo:

- En el sistema hay sólo dos procesos: P1, que alterna fases de cálculo con llamadas al sistema no bloqueantes; y P2, de mayor prioridad, que realiza una llamada de lectura para este dispositivo solicitando leer 4 bytes.
- El *buffer* especificado en la llamada de lectura de P2 corresponde a una variable `V` de tipo entero con valor inicial incluida en una página que no ha sido accedida por el proceso previamente (`read(d, &V, 4);`).
- El tiempo medio de una operación sobre este dispositivo es de 10 ms, mientras que el del disco es de 5 ms.
- Se trata de un núcleo no expulsivo, con un algoritmo de planificación expulsivo basado en prioridad. Además, se supone que hay suficientes marcos libres en el sistema para la ejecución de los procesos.

b) Responda a las siguientes cuestiones analizando la ejecución del ejemplo usando la 2ª versión:

b1) Especifique la traza de ejecución de los procesos hasta que se completa la llamada de lectura, mostrando qué operaciones se llevan a cabo en cada activación del sistema operativo y resaltando los cambios de contexto, distinguiendo entre voluntarios e involuntarios.

b2) Para estimar la eficiencia, calcule cuánto tarda en completarse la operación de lectura de P2 teniendo en cuenta sólo el tiempo de operación de los dispositivos. Además, determine cuántos cambios de contexto se producen.

c) Repita el apartado anterior para la 3ª versión, comparando los resultados en cuanto a eficiencia.

d) Repita el apartado anterior para la 4ª versión suponiendo que `UMBRAL` es igual a 1 (es decir, que no hay umbral). Compare los resultados con las dos versiones previas.

e) Repita el apartado anterior suponiendo que `UMBRAL` es igual a 2. Analizando los resultados, ¿para qué considera que sirve el umbral?

Solución

a) La primera versión no usa un *buffer* intermedio para realizar la lectura, sino que los caracteres leídos desde el dispositivo se copian directamente a la zona de memoria especificada en la llamada. Para ello, en la función de lectura se guarda en una variable global (`dir_buf`) la dirección de comienzo de la zona de memoria donde se deben copiar los datos, de manera que desde la rutina de interrupción se puedan copiar sucesivamente los datos leídos del dispositivo en dicha zona.

```
*(dir_buf++) = in(R_DATOS);
```

Esta solución no es válida ya que desde una rutina de interrupción no se debe acceder al mapa de usuario del proceso que está en ejecución en ese instante, puesto que, dado el carácter asíncrono de la interrupción, es impredecible conocer a priori qué proceso estará ejecutando en el momento que se produzca. Al hacer referencia a la dirección almacenada en `dir_buf`, el sistema de gestión de memoria la interpreta dentro del contexto del proceso actual, que, evidentemente, no es el que realizó la llamada, puesto que está bloqueado. Como resultado, se estarán copiando los datos leídos en una zona impredecible del mapa del proceso que estuviera ejecutando en ese momento. Además de este error inadmisible, el acceso desde una rutina de interrupción al mapa de usuario tampoco es tolerable porque, si la página accedida no está residente, causará un fallo de página con el consiguiente bloqueo, que es inaceptable dentro de una rutina de interrupción.
La solución habitual es usar un *buffer* intermedio que permita copiar los datos desde la función de lectura, que sí ejecuta de una manera síncrona en el contexto del proceso que solicitó la operación. Esta es la alternativa utilizada en las restantes versiones planteadas en este ejercicio. En el caso de que, dadas las características del dispositivo, no se pueda admitir la sobrecarga asociada a la copia adicional requerida por el uso de un *buffer* intermedio y sea preciso realizar la copia directamente desde la rutina de interrupción, se puede usar una solución que retenga en memoria las páginas que incluyen la zona de memoria especificada en la operación de lectura, para evitar que se produzcan fallos de página al copiar los datos, y que proyecte la zona de memoria afectada dentro del mapa de memoria del sistema operativo, de manera que el sistema operativo pueda acceder a la misma desde la rutina de interrupción utilizando direcciones del sistema, que son válidas sea cual sea el proceso que esté ejecutando.

b1) A continuación, se detalla la traza de ejecución para la segunda versión que conllevaría los siguientes pasos:
1. Comienza ejecutando P2, ya que tiene mayor prioridad, y realiza la llamada de lectura del dispositivo, pasando a modo sistema. Se programa el dispositivo y se bloquea el proceso, realizando un **cambio de contexto voluntario** a P1.
2. P1 continúa ejecutando sus fases de cálculo y de llamadas al sistema no bloqueantes hasta que se produce una interrupción del dispositivo a los 10 ms. desde el comienzo de la traza, aproximadamente. La rutina de interrupción copia el carácter en el *buffer* y desbloquea el proceso P2. Al detectar que P2 es más prioritario que el proceso en ejecución, se activa una interrupción software para forzar un cambio de contexto. El uso de la interrupción software asegura que si P1 estaba ejecutando una llamada al sistema en el momento de producirse la interrupción, ésta concluye antes de realizarse un cambio de contexto. En el tratamiento de la interrupción software se lleva a cabo el cambio de **contexto involuntario** de P1 a P2.
3. P2 continúa la ejecución de la llamada copiando el carácter desde el *buffer* interno a la zona de memoria especificada en la llamada (que corresponde a la variable `v`), produciéndose un fallo de página puesto que la página donde se incluye dicha variable no está residente en memoria. Dado que la página está almacenada en el fichero ejecutable, al tratarse de una variable con valor inicial, la rutina de tratamiento del fallo de página programa una lectura del disco y bloquea el proceso realizando un cambio de **contexto voluntario** de P2 a P1.
4. P1 continúa su ejecución hasta que se produce la interrupción del disco (habrán pasado aproximadamente 15 ms. desde el comienzo de la traza). La rutina de tratamiento de dicha interrupción desbloquea a P2 y activa una interrupción software, en cuyo tratamiento se realiza el cambio de **contexto involuntario** a P2.
5. P2 ejecuta la siguiente iteración del bucle programando el dispositivo y realizando un **cambio de contexto voluntario** a P1.
6. Mientras ejecuta P1, llega la interrupción que corresponde al siguiente carácter (aproximadamente, en el instante 25 ms.), en cuya rutina de tratamiento se copia el nuevo carácter en el *buffer* interno y se desbloquea el proceso P2, produciéndose un **cambio de contexto involuntario** a P2 dentro del tratamiento de la interrupción software activada en el desbloqueo.
7. P2 copia el carácter en el siguiente byte de la variable `v`, pero en este caso no hay fallo de página puesto que la página ya está residente. A continuación, programa el dispositivo para leer el siguiente carácter y se bloquea realizando un **cambio de contexto voluntario** a P1.
8. Se repite el paso 6 (instante 35 ms.) para el tercer carácter (**cambio de contexto involuntario** a P2).
9. Se repite el paso 7 para el tercer carácter (**cambio de contexto voluntario** a P1).
10. Se repite el paso 6 (instante 45 ms.) para el cuarto carácter (**cambio de contexto involuntario** a P2).
11. P2 continúa su ejecución completando la llamada y retornando a modo usuario.

b2) En la traza se puede apreciar que el tiempo total es de 45 ms., habiendo 10 cambios de contexto, 5 voluntarios y 5 involuntarios.

c1) A continuación, se detalla la traza de ejecución para la tercera versión que conllevaría los siguientes pasos:
1. Comienza ejecutando P2, ya que tiene mayor prioridad, y realiza la llamada de lectura del dispositivo, pasando a modo sistema. Se programa el dispositivo y se bloquea el proceso, realizando un **cambio de contexto voluntario** a P1.
2. P1 continúa ejecutando sus fases de cálculo y de llamadas al sistema no bloqueantes hasta que se produce una interrupción del dispositivo a los 10 ms. desde el comienzo de la traza, aproximadamente. La rutina de interrupción copia el carácter en el *buffer* y termina, retornando P1 a modo usuario.
3. Se repite el paso anterior para el segundo carácter (instante 20 ms.).
4. Se repite el paso anterior para el tercer carácter (instante 30 ms.).
5. Se produce la cuarta interrupción del dispositivo (instante 40 ms.). En este caso, al completarse los datos solicitados, se desbloquea el proceso P2, activándose una interrupción software para forzar un cambio de contexto. En el tratamiento de la interrupción software se lleva a cabo el cambio de **contexto involuntario** de P1 a P2.
6. P2 continúa la ejecución de la llamada copiando el primer carácter desde el *buffer* interno a la zona de memoria especificada en la llamada, produciéndose un fallo de página puesto que la página donde se incluye dicha variable no está residente. La rutina de tratamiento del fallo de página programa una lectura del disco y bloquea el proceso realizando un cambio de **contexto voluntario** de P2 a P1.

7. P1 continúa su ejecución hasta que se produce la interrupción del disco (habrán pasado aproximadamente 45 ms. desde el comienzo de la traza). La rutina de tratamiento de dicha interrupción desbloquea el proceso P2 y activa una interrupción software, en cuyo tratamiento se realiza el cambio de **contexto involuntario** a P2.
8. P2 completa la copia de los otros tres caracteres, terminando la llamada y retornando a modo usuario.

c2) En la traza se puede apreciar que el tiempo total es de 45 ms., como en la versión anterior. Sin embargo, hay sólo 4 cambios de contexto, 2 voluntarios y 2 involuntarios. La clave de esta importante reducción en el número de cambios de contexto es trasladar la programación del dispositivo desde la rutina de lectura, como ocurre en la segunda versión, a la rutina de interrupción, exceptuando la primera vez. Con esta técnica, no es necesario desbloquear el proceso que solicitó la operación para reprogramar el dispositivo, llevándose a cabo en el contexto del proceso que esté ejecutando actualmente. Es interesante resaltar que realizar la reprogramación del dispositivo dentro del contexto de la rutina de interrupción tiene otras consecuencias adicionales en el modo de operación del dispositivo. Supóngase que un proceso de baja prioridad solicita una lectura del dispositivo. Con un esquema en el que la reprogramación se realiza en la función de lectura (como en la versión 2), dado que esta labor la ejecuta directamente el proceso involucrado, la operación de lectura sólo progresará cuando pueda ejecutar dicho proceso, es decir, cuando no haya procesos más prioritarios en el sistema. Sin embargo, si la reprogramación se realiza en la rutina de interrupción (como en las versiones 3 y 4), ésta se llevará a cabo automáticamente en cada interrupción, con independencia de la prioridad del proceso que ha realizado la petición. Se podría decir que con este segundo esquema se le da más prioridad a la gestión del dispositivo que a la ejecución de cualquier proceso, lo cual puede ser conveniente si se trata de un dispositivo usado con cierta frecuencia por los procesos que conviene que sea gestionado eficientemente para aprovechar sus prestaciones.
Además de eficiencia en cuanto al tiempo que conlleva la operación, es importante analizar los requisitos en lo que se refiere al uso de memoria. Este esquema requiere reservar una zona de memoria del sistema tan grande como el tamaño que tenga la petición, lo cual puede ser un gasto excesivo.

d1) A continuación, se detalla la traza de ejecución para la cuarta versión con UMBRAL igual a 1, que conllevaría los siguientes pasos:
1. Comienza ejecutando P2, ya que tiene mayor prioridad, y realiza la llamada de lectura del dispositivo, pasando a modo sistema. Se programa el dispositivo y se bloquea el proceso, realizando un **cambio de contexto voluntario** a P1.
2. P1 continúa ejecutando sus fases de cálculo y de llamadas al sistema no bloqueantes hasta que se produce una interrupción del dispositivo a los 10 ms. desde el comienzo de la traza, aproximadamente. La rutina de interrupción copia el carácter en el *buffer*, desbloquea el proceso P2, ya que el número de caracteres en el *buffer* es igual a 1 (valor del umbral), y programa el dispositivo para leer el siguiente carácter. Al detectar que P2 es más prioritario que el proceso en ejecución, se activa una interrupción software para forzar un cambio de contexto. En el tratamiento de la interrupción software se lleva a cabo el cambio de **contexto involuntario** de P1 a P2.
3. P2 continúa la ejecución de la llamada copiando el carácter desde el *buffer* interno a la zona de memoria especificada en la llamada (que corresponde a la variable `v`), produciéndose un fallo de página puesto que la página donde se incluye dicha variable no está residente. La rutina de tratamiento del fallo de página programa una lectura del disco y bloquea el proceso realizando un cambio de **contexto voluntario** de P2 a P1.
4. P1 continúa su ejecución hasta que se produce la interrupción del disco (habrán pasado aproximadamente 15 ms. desde el comienzo de la traza). La rutina de tratamiento de dicha interrupción desbloquea a P2 y activa una interrupción software, en cuyo tratamiento se realiza el cambio de **contexto involuntario** a P2.
5. P2 ejecuta la siguiente iteración del bucle, realizando un **cambio de contexto voluntario** a P1, ya que el *buffer* está vacío.
6. Mientras ejecuta P1, llega la interrupción que corresponde al siguiente carácter (aproximadamente, en el instante 20 ms.), en cuya rutina de tratamiento se copia el nuevo carácter en el *buffer* interno, se programa el dispositivo y se desbloquea el proceso P2, ya que el número de caracteres en el *buffer* es igual a 1, produciéndose un **cambio de contexto involuntario** a P2 dentro del tratamiento de la interrupción software activada en el desbloqueo.
7. P2 copia el carácter en el siguiente byte de la variable `v`, pero en este caso no hay fallo de página puesto que la página ya está residente. A continuación, al encontrarse el *buffer* vacío, se bloquea realizando un **cambio de contexto voluntario** a P1.
8. Se repite el paso 6 (instante 30 ms.) para el tercer carácter (**cambio de contexto involuntario** a P2).
9. Se repite el paso 7 para el tercer carácter (**cambio de contexto voluntario** a P1).
10. Mientras ejecuta P1, llega la interrupción que corresponde al último carácter (aproximadamente, en el instante 40 ms.), en cuya rutina de tratamiento se copia el nuevo carácter en el *buffer* interno y se desbloquea el proceso P2, produciéndose un **cambio de contexto involuntario** a P2 dentro del tratamiento de la interrupción software activada en el desbloqueo.
11. P2 continúa su ejecución completando la llamada y retornando a modo usuario.

d2) En la traza se puede apreciar que el tiempo total es de 40 ms., habiendo 10 cambios de contexto, 5 voluntarios y 5 involuntarios. Esta versión consigue una importante reducción en el tiempo gracias a que consigue que se realice en paralelo el acceso al dispositivo y la copia de los caracteres a la zona de memoria especificada en la llamada, que puede implicar, como ocurre en este caso, el tratamiento de fallos de página que conllevan accesos al disco. Nótese que la estructura del manejador se corresponde directamente con el modelo productor-consumidor, donde la rutina de interrupción asume el papel de productor, mientras que la función de lectura tiene el rol de consumidor. En el supuesto planteado en el enunciado, el *buffer* interno nunca llega a mantener más de un elemento. Sin embargo, si el tiempo que tarda el acceso al disco fuera superior al del dispositivo, llegarían a almacenarse varios caracteres en el *buffer*, que serían consumidos por la función de lectura al terminar de servirse el fallo de página. En ese caso, habría un menor número de cambios de contexto durante la operación.
Con respecto al gasto de memoria, lo más razonable sería usar un *buffer* interno de un tamaño razonable, que podría ser menor que el tamaño de la petición. Se trataría de implementar el esquema del productor-consumidor con un *buffer* acotado, que, para simplificar, no se ha incluido en la versión mostrada en el enunciado.

e1) A continuación, se detalla la traza de ejecución para la cuarta versión con UMBRAL igual a 2, que conllevaría los siguientes pasos:

1. Comienza ejecutando P2, ya que tiene mayor prioridad, y realiza la llamada de lectura del dispositivo, pasando a modo sistema. Se programa el dispositivo y se bloquea el proceso, realizando un **cambio de contexto voluntario** a P1.
2. P1 continúa ejecutando sus fases de cálculo y de llamadas al sistema no bloqueantes hasta que se produce una interrupción del dispositivo a los 10 ms. desde el comienzo de la traza, aproximadamente. La rutina de interrupción copia el carácter en el *buffer*, y programa el dispositivo para leer el siguiente carácter. Observe que no se desbloquea el proceso P2 ya que el número de caracteres en el buffer es menor que el umbral (1 < 2). La rutina de interrupción termina retornando P1 a modo usuario.
3. P1 continúa ejecutando sus fases de cálculo y de llamadas al sistema no bloqueantes hasta que se produce otra interrupción del dispositivo a los 20 ms. desde el comienzo de la traza, aproximadamente. La rutina de interrupción copia el carácter en el *buffer*, desbloquea el proceso P2, ya que el número de caracteres en el *buffer* es igual al umbral, y programa el dispositivo para leer el siguiente carácter. Al detectar que P2 es más prioritario que el proceso en ejecución, se activa una interrupción software para forzar un cambio de contexto. En el tratamiento de la interrupción software se lleva a cabo el cambio de **contexto involuntario** de P1 a P2.
4. P2 continúa la ejecución de la llamada copiando el carácter desde el *buffer* interno a la zona de memoria especificada en la llamada (que corresponde a la variable v), produciéndose un fallo de página puesto que la página donde se incluye dicha variable no está residente. La rutina de tratamiento del fallo de página programa una lectura del disco y bloquea el proceso realizando un cambio de **contexto voluntario** de P2 a P1.
5. P1 continúa su ejecución hasta que se produce la interrupción del disco (habrán pasado aproximadamente 25 ms. desde el comienzo de la traza). La rutina de tratamiento de dicha interrupción desbloquea a P2 y activa una interrupción software, en cuyo tratamiento se realiza el cambio de **contexto involuntario** a P2.
6. P2 copia los dos caracteres disponibles en el buffer interno y realiza un **cambio de contexto voluntario** a P1.
7. Mientras ejecuta P1, llega la interrupción que corresponde al siguiente carácter (aproximadamente, en el instante 30 ms.), en cuya rutina de tratamiento se copia el nuevo carácter en el *buffer* interno, completándose la rutina de interrupción.
8. P1 continúa su ejecución hasta que llega la interrupción que corresponde al siguiente carácter (aproximadamente, en el instante 40 ms.), en cuya rutina de tratamiento se copia el nuevo carácter en el *buffer* interno, se programa el dispositivo y se desbloquea el proceso P2, ya que el número de caracteres en el *buffer* ha alcanzado el umbral (además, la petición se ha completado), produciéndose un **cambio de contexto involuntario** a P2 dentro del tratamiento de la interrupción software activada en el desbloqueo.
9. P2 copia los dos caracteres completando la llamada y retornando a modo usuario.

e2) En la traza se puede apreciar que el tiempo total es de 40 ms. y se producen 6 cambios de contexto, 3 voluntarios y 3 involuntarios. En esta versión se consigue una importante reducción en el número de cambios de contexto gracias al uso del umbral, que permite retardar el desbloqueo de un proceso hasta que haya un número suficiente de caracteres en el *buffer*.
La determinación del valor óptimo del umbral para un determinado dispositivo depende de factores tales como el tiempo de acceso al dispositivo y el tiempo de acceso al disco. Para un funcionamiento eficiente del dispositivo, así como del sistema en su totalidad, hay que intentar maximizar el paralelismo entre el acceso al dispositivo y la copia de los datos al buffer de usuario. Asimismo, es importante asegurar que cada vez que ejecuta un proceso después de desbloquearse tenga una racha de ejecución de un tamaño considerable (significativamente mayor que la duración de un cambio de contexto). En caso contrario, el porcentaje de uso útil del procesador se verá afectado negativamente.