Diseño de Sistemas Operativos. Junio de 2015.
Ejercicio de procesos y memoria

Se pretende desarrollar el manejador de un dispositivo de entrada de tipo teclado que opera en modo carácter y usa interrupciones. Nótese que un teclado es un dispositivo asíncrono en el sentido de que cada vez que se pulsa una tecla se produce una interrupción, sin necesidad de programar cada operación de lectura. El manejador no permitirá el tecleado anticipado: no se guardarán los caracteres que lleguen no habiendo ningún proceso esperando por ellos. 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 *dirb;
tipo_cola_procesos cola_term;
int lectura(char *dir, int tam) {
  dirb = dir;
  while (tam-- > 0)
       Bloquear(&cola_term);
}
void interrupcion() {
  if (cola_term.vacia()) return;//nadie lo espera
  *dirb++ = in(R_DATOS_TERM);
  Desbloquear(&cola_term);
}
```

Versión 2

```
char b;
tipo_cola_procesos cola_term;
int lectura(char *dir, int tam) {
  while (tam-- > 0) {
       Bloquear(&cola_term);
       *dir++ = b; }
}
void interrupcion() {
  if (cola_term.vacia()) return;//nadie lo espera
  b = in(R_DATOS_TERM);
  Desbloquear(&cola_term);
}
```

Versión 3

```
char *dirb; int tam_pet;
tipo_cola_procesos cola_term;
int lectura(char *dir, int tam) {
  dirb = kmalloc(tam); tam_pet = tam;
  Bloquear(&cola_term);
  copy_from_sys_to_user(dirb, dir, tam);
  kfree(dirb);
}
void interrupcion() {
  if (cola_term.vacia()) return;//nadie lo espera
  *dirb++ = in(R_DATOS_TERM);
  if (--tam_pet == 0)
    Desbloquear(&cola_term);
}
```

Versión 4

```
char *dirb, *diro; int tam_pet, tam_b;
tipo_cola_procesos cola_term;
int lectura(char *dir, int tam) {
  dirb = kmalloc(tam); tam_pet = tam;
  diro = dir; tam_b = tam;
  Bloquear(&cola_term);
  kfree(dirb);
}
void interrupcion() {
  if (cola_term.vacia()) return;//nadie lo espera
  *dirb++ = in(R_DATOS_TERM);
  if (--tam_pet == 0) {
    copy_from_sys_to_user(dirb, diro, tam_b);
    Desbloquear(&cola_term); }
}
```

a) Una de las dos primeras versiones es errónea. Explique qué fallo de diseño hay en la misma identificando en qué sentencia del código se produce este problema.

b) Una de las dos últimas versiones es errónea. Explique qué fallo de diseño hay en la misma identificando en qué sentencia del código se produce este problema.

Para comparar las dos versiones correctas vamos a utilizar el siguiente ejemplo:

- **P** (prioridad alta): solicita una lectura de 2 bytes especificando como *buffer* una variable global sin valor inicial que ocupa dos bytes almacenados al principio de una página que nunca se había accedido previamente. Completada la llamada de lectura, **P** termina de forma normal.
- **Q** (prioridad baja): Recién creado, alternando fases de cálculo en modo usuario con llamadas `getpid`. Suponga que las sucesivas interrupciones que se produzcan durante la traza encuentran alternativamente a este proceso en modo usuario y en sistema.

c) Detalle la traza de ejecución planteada usando la primera versión correcta, identificando las activaciones del S.O. y los cambios de contexto suponiendo un núcleo **expulsivo** y que hay suficientes marcos libres en el sistema.

d) Repita el apartado anterior para la segunda versión correcta.

e) Calcule, para cada una de las dos versiones correctas, cuántos cambios de contexto voluntarios y cuántos involuntarios se producirán en el ejemplo anterior durante la operación de lectura si el proceso **P** solicita *N* bytes, en vez de 2, suponiendo que no se produjeran fallos de página durante la misma. Repita el cálculo para un núcleo **no expulsivo**.

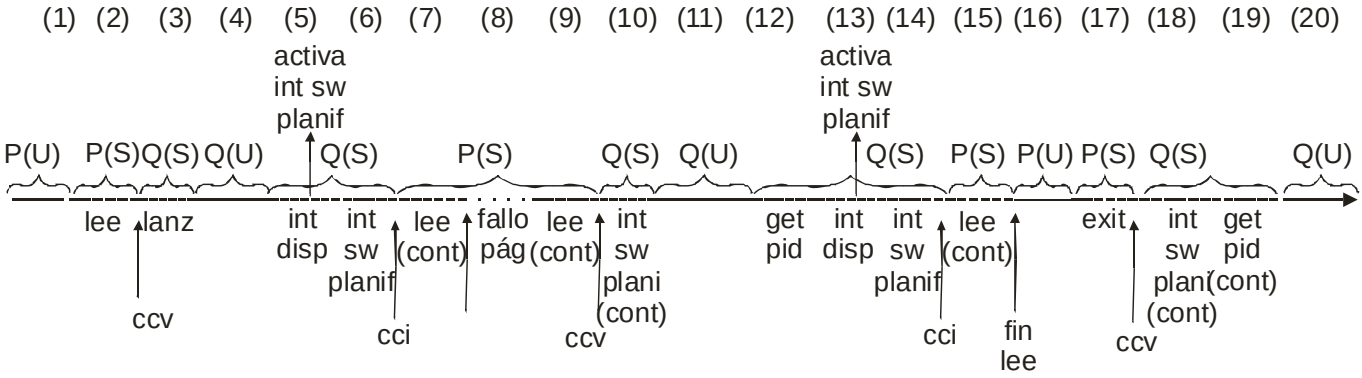
f) Plantee, al menos, dos ejemplos, ya sean basados en el escenario descrito en este ejercicio o en uno independiente del mismo, donde se produzcan más cambios de contexto con un esquema expulsivo que con uno no expulsivo.

Solución

a) La primera versión es incorrecta puesto que en la misma se está accediendo al mapa de usuario del proceso durante la ejecución de la rutina de tratamiento de una interrupción: `*dirb++ = in(R_DATOS)`. Nótese que en esta versión el puntero `dir_b` apunta al *buffer* de usuario. Este acceso es erróneo puesto que desde el tratamiento de un evento asíncrono no se puede acceder al mapa de usuario del proceso puesto que es impredecible qué proceso está ejecutando en ese instante. Por su parte, la segunda versión también accede al *buffer* de usuario (`*dir++ = b`) pero lo hace desde el tratamiento de un evento síncrono, una llamada al sistema, por lo que el mapa de usuario activo corresponde al proceso que realizó la llamada.

b) La cuarta versión es errónea por la misma razón que en la pregunta anterior: se está accediendo al mapa de usuario del proceso durante la ejecución de la rutina de tratamiento de una interrupción en la sentencia `copy_from_sys_to_user(diro, dir, tam)`. La tercera versión también ejecuta esa sentencia pero en el contexto de un evento síncrono.

c) La siguiente gráfica muestra la traza de ejecución del ejemplo planteado para la segunda versión del manejador:

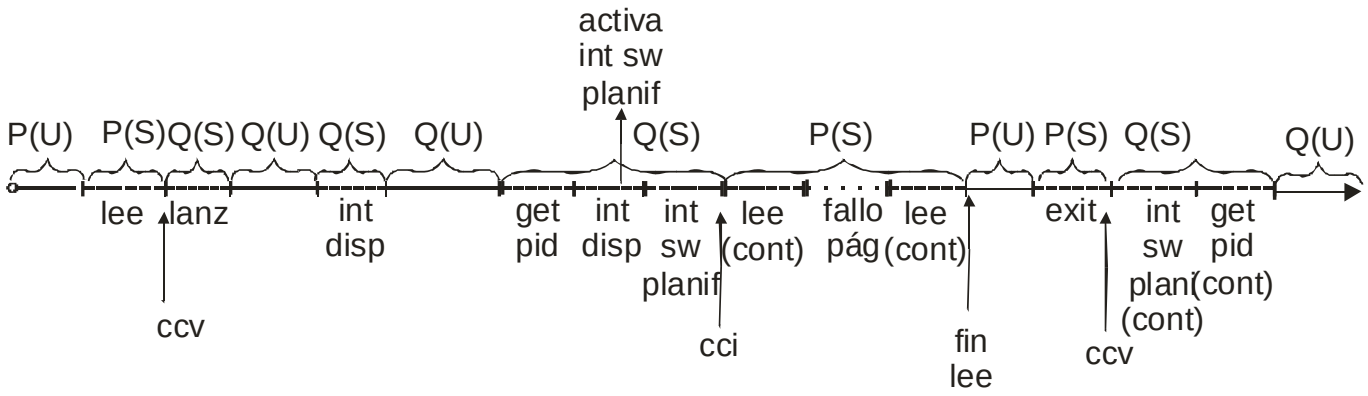


A continuación, se detallan cada uno de los eventos que ocurren durante la traza:

1. **P** en modo usuario realiza una llamada al sistema solicitando leer dos bytes del dispositivo, especificando como *buffer* una variable global sin valor inicial que ocupa dos bytes y está almacenada en una única página no accedida previamente.
2. **P** en modo sistema, dentro de la llamada de lectura, se bloquea esperando que lleguen caracteres, produciéndose un cambio de contexto voluntario a **Q**. Nótese que no se permite el tecleado anticipado por lo que no puede haber caracteres almacenados previamente.
3. **Q** comienza ejecutando en modo sistema una función de lanzadera que rápidamente se completa pasando a ejecutar en modo usuario la rutina inicial del programa.
4. Mientras **Q** ejecuta en modo usuario, se produce una interrupción del teclado indicando que ya se leído la página correspondiente.
5. La rutina de interrupción del teclado desbloquea al proceso **P**. Dado que el proceso desbloqueado es más prioritario que el actual, se activa una interrupción software expulsiva de planificación para llevar a cabo el cambio de contexto involuntario de **Q** a **P**.
6. La rutina de interrupción software de planificación lleva a cabo el cambio de contexto involuntario requerido, con lo que **P** continúa la llamada de lectura justo en el punto donde se bloqueó.
7. **P** en modo sistema intenta escribir en el primer byte de la variable especificada por el programa en la llamada al sistema, pero, dado que la página que contiene esa variable no está residente, se produce un fallo de página que se anida con la llamada al sistema.
8. Al tratarse de una variable global sin valor inicial, estará incluida en una región anónima (sin soporte en el fichero ejecutable), por lo que no requerirá acceder a disco para traerla. El tratamiento del fallo de página buscará un marco libre, que por el enunciado sabemos que habrá, y lo rellenará con ceros por motivos de seguridad, no produciéndose ningún cambio de contexto.
9. **P** en modo sistema continúa con la llamada de lectura y, dado que todavía quedan caracteres por leer, se bloquea esperando que lleguen, produciéndose un cambio de contexto voluntario a **Q**.
10. **Q** 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.
11. **Q** ejecuta en modo usuario hasta que realiza la llamada al sistema `getpid`.
12. Mientras **Q** ejecuta en modo sistema la llamada `getpid`, se produce una interrupción del dispositivo.
13. La rutina de interrupción del dispositivo desbloquea al proceso **P**. Dado que el proceso desbloqueado es más prioritario que el actual, se activa una interrupción software expulsiva de planificación para llevar a cabo el cambio de contexto involuntario de **Q** a **P**. Al tratarse de un núcleo expulsivo, el tratamiento de la interrupción software de planificación es más prioritario que la llamada al sistema quedándose está sin completar.

14. La rutina de interrupción software de planificación lleva a cabo el cambio de contexto involuntario requerido, con lo que **P** continúa la ejecución de la llamada al sistema justo en el punto donde quedó detenida.
15. **P** copia este segundo carácter al buffer, no produciéndose fallo de página puesto que se acaba de traer a memoria, y detecta que no quedan más datos por leer, completándose la llamada y retornando a modo usuario.
16. **P** ejecuta en modo usuario hasta que realiza la llamada al sistema para completar su ejecución.
17. La llamada de terminación completa la ejecución de **P**, produciéndose un cambio de contexto voluntario a **Q**.
18. **Q** 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 al punto de la llamada `getpid` donde quedó interrumpido.
19. **Q** continúa y completa la llamada `getpid`.
20. **Q** prosigue su ejecución en modo usuario.

d) La siguiente gráfica muestra la traza de ejecución del ejemplo planteado para la tercera versión del manejador:



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

1. **P** en modo usuario realiza una llamada al sistema solicitando leer dos bytes del dispositivo, especificando como *buffer* una variable global sin valor inicial que ocupa dos bytes y está almacenada en una única página no accedida previamente.
2. **P** en modo sistema, dentro de la llamada de lectura, se bloquea esperando que lleguen caracteres, produciéndose un cambio de contexto voluntario a **Q**. Nótese que no se permite el tecleado anticipado por lo que no puede haber caracteres almacenados previamente.
3. **Q** comienza ejecutando en modo sistema una función de lanzadera que rápidamente se completa pasando a ejecutar en modo usuario la rutina inicial del programa.
4. Mientras **Q** ejecuta en modo usuario, se produce una interrupción del teclado indicando que ya se leído la página correspondiente.
5. La rutina de interrupción del teclado copia en el buffer del sistema reservado previamente el carácter leído y, como detecta que quedan todavía datos por leer, se completa la rutina de interrupción sin desbloquear a ningún proceso.
6. **Q** prosigue ejecutando en modo usuario hasta que realiza la llamada al sistema `getpid`.
7. Mientras **Q** ejecuta en modo sistema la llamada `getpid`, se produce una interrupción del dispositivo.
8. La rutina de interrupción del dispositivo copia en el buffer del sistema el segundo carácter leído y, como detecta que no quedan más datos por leer, desbloquea al proceso **P**. Dado que el proceso desbloqueado es más prioritario que el actual, se activa una interrupción software expulsiva de planificación para llevar a cabo el cambio de contexto involuntario de **Q** a **P**. Al tratarse de un núcleo expulsivo, el tratamiento de la interrupción software de planificación es más prioritario que la llamada al sistema quedándose está sin completar.
9. La rutina de interrupción software de planificación lleva a cabo el cambio de contexto involuntario requerido, con lo que **P** continúa la ejecución de la llamada al sistema justo en el punto donde quedó detenida.
10. **P** en modo sistema intenta copiar el primer byte del buffer interno a la variable especificada por el programa en la llamada al sistema, pero, dado que la página que contiene esa variable no está residente, se produce un fallo de página que se anida con la llamada al sistema.
11. Al tratarse de una variable global sin valor inicial, estará incluida en una región anónima (sin soporte en el fichero ejecutable), por lo que no requerirá acceder a disco para traerla. El tratamiento del fallo de página buscará un marco libre, que por el enunciado sabemos que habrá, y lo rellenará con ceros por motivos de seguridad, no produciéndose ningún cambio de contexto.
12. **P** en modo sistema continúa con la llamada de lectura copiando el segundo carácter, no produciéndose fallo de página puesto que se acaba de traer a memoria, completa la llamada y retorna a modo usuario.
13. **P** ejecuta en modo usuario hasta que realiza la llamada al sistema para completar su ejecución.
14. La llamada de terminación completa la ejecución de **P**, produciéndose un cambio de contexto voluntario a **Q**.

15. **Q** 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 al punto de la llamada `getpid` donde quedó interrumpido.
16. **Q** continúa y completa la llamada `getpid`.
17. **Q** prosigue su ejecución en modo usuario.

e) Para la segunda versión, se produce un cambio de contexto voluntario (una llamada a *Bloquear*) por cada carácter leído y un cambio de contexto involuntario (una llamada a *Desbloquear* que despierta a un proceso más prioritario) también por cada carácter (por cada interrupción del dispositivo). Por tanto, se producirán ***N*** cambios de contexto voluntarios y N involuntarios durante la operación de lectura.
En cuanto a la cuarta versión, sólo se produce **1** cambio de contexto voluntario al principio de la operación y **1** involuntario al final de la misma, cuando llega la última interrupción, con independencia del número de bytes que se pretendan leer.
Estos resultados son independiente de si el núcleo es expulsivo o no lo es puesto que eso sólo influye en si la expulsión de un proceso de difiere hasta que termine la llamada en curso o no.

f) El número de cambios de contexto voluntarios es el mismo para ambos tipos de núcleos puesto que este tipo de cambios están asociados a los bloqueos de los procesos a la espera de un evento.
En cuanto al número de cambios de contexto involuntarios, en principio, debería ser similar en ambos tipos de núcleo, puesto que, al fin y al cabo, el núcleo no expulsivo puede retrasar la expulsión de un proceso pero ésta acaba haciéndose.
Sin embargo, hay situaciones en la que este retardo en realizar el cambio de contexto involuntario puede llevar a reducir el número de cambios involuntarios requeridos. A continuación, se plantean dos ejemplos en los que se reduce el número de cambios de contexto involuntarios si se usa un núcleo expulsivo:

- Un proceso **P** de baja prioridad está ejecutando una llamada al sistema que puede ser bloqueante (por ejemplo, leer de una tubería) y durante la misma se produce una interrupción que desbloquea a un proceso de mayor prioridad **Q**. Con un núcleo no expulsivo cuando se complete la interrupción, seguirá la llamada del proceso P no produciéndose su expulsión hasta que ésta se complete. Pero si en vez de completarse la llamada, **P** se bloquea (en el ejemplo, porque la tubería está vacía), se producirá el cambio de contexto voluntario de **P** a **Q** y no el involuntario (explicado de manera informal, antes de echar al proceso, éste se ha ido). Nótese que con un núcleo expulsivo sí se habría producido el cambio involuntario y, en el futuro, cuando **P** reanudase su ejecución, se llevaría a cabo el cambio voluntario de **P** a otro proceso al encontrarse la tubería vacía.
- En un núcleo no expulsivo, está ejecutando un proceso **P** de baja prioridad una llamada al sistema y durante la misma se van produciendo interrupciones que desbloquean a procesos de prioridad creciente. En esta situación, sólo se producirá al final de la llamada un cambio de contexto involuntario al proceso de más prioridad desbloqueado. Con un núcleo expulsivo, se generaría una cadena de cambios de contexto involuntarios según van llegando las interrupciones que desbloquean a procesos de prioridad creciente.