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

Se pretende desarrollar el manejador de un dispositivo de salida 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

```
tipo_cola_procesos cola_disp;
int escritura(char *dir, int tam) {
     while (tam-- > 0) {
          out(R_DATOS, *dir++);
          out(R_CONTROL, ESCR);
          Bloquear(&cola_disp);
     }
}
void interrupcion() {
     Desbloquear(&cola_disp);
}
```

Versión 2

```
char *dir_b; int tam_pet;
tipo_cola_procesos cola_disp;
int escritura(char *dir, int tam) {
     dir_b = dir; tam_pet = tam;
     out(R_DATOS, *dir_b++); out(R_CONTROL, ESCR);
     Bloquear(&cola_disp);
}
void interrupcion() {
  if (--tam_pet > 0){
    out(R_DATOS,*dir_b++);out(R_CONTROL, ESCR);}
  else Desbloquear(&cola_disp);
}
```

Versión 3

```
char *dir_b; int tam_pet;
tipo_cola_procesos cola_disp;
void programar_dispositivo() {
  out(R_DATOS, *dir_b++); out(R_CONTROL, ESCR);
}
int escritura(char *dir, int tam) {
  dir_b = kmalloc(tam); tam_pet = tam;
  copy_from_user_to_sys(dir, dir_b, tam);
  programar_dispositivo();
  Bloquear(&cola_disp);
  kfree(dir_b);
}
void interrupcion() {
  if (--tam_pet > 0) programar_dispositivo();
  else Desbloquear(&cola_disp);
}
```

Versión 4

```
char *dir_b; int tam_pet;
tipo_cola_procesos cola_disp;
void programar_dispositivo() {
  out(R_DATOS, *dir_b++); out(R_CONTROL, ESCR);
  Bloquear(&cola_disp);
}
int escritura(char *dir, int tam) {
  dir_b = kmalloc(tam); tam_pet = tam;
  copy_from_user_to_sys(dir, dir_b, tam);
  programar_dispositivo();
  kfree(dir_b);
}
void interrupcion() {
  if (--tam_pet > 0) programar_dispositivo();
  else Desbloquear(&cola_disp);
}
```

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 escritura de 2 bytes especificando como *buffer* una variable global con valor inicial que ocupa dos bytes almacenados al principio de una página que nunca se había accedido previamente. Completada la llamada de escritura, **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 no expulsivo y que hay suficientes marcos libres en el sistema.
d) Repita el apartado anterior para la segunda versión correcta.

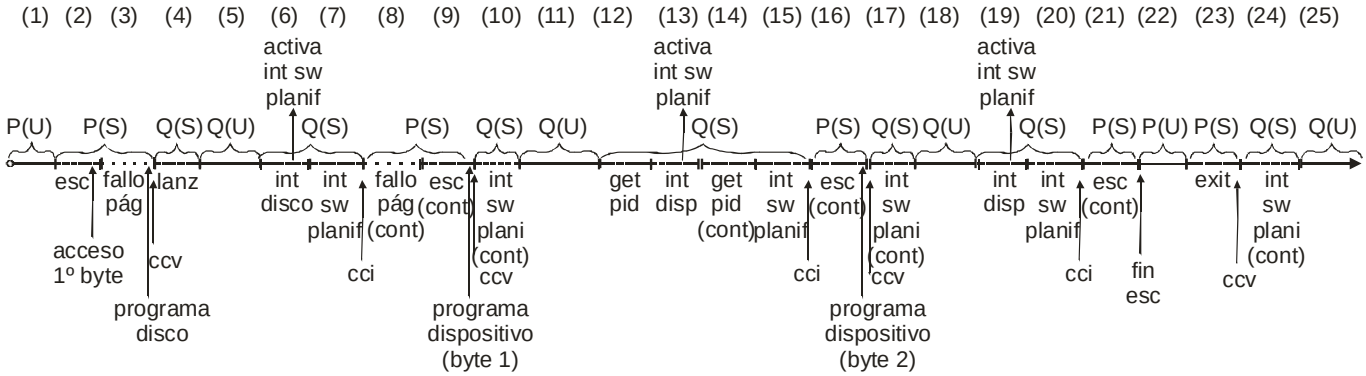
e) Compare la eficiencia en la ejecución de la traza de las versiones correctas en cuanto a: (i) nº de fallos de página producidos; (ii) nº de interrupciones del dispositivo. (iii) nº de cambios de contexto voluntarios requeridos.
f) Para realizar una última comparación, suponga que cuando se está ejecutando el proceso **Q** después de que se haya programado la escritura del primer byte en el dispositivo pero antes de que se produzca la primera interrupción del mismo, se desbloquea un proceso **R** de máxima prioridad que expulsa a **Q**. ¿Cómo afectaría la ejecución del proceso **R** al progreso de la operación de escritura en cada versión? ¿Para qué versión se podría decir que se les da más prioridad a los dispositivos que a los procesos?

Solución

a) La segunda 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: `out(R_DATOS,*dir_b++)`. 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 primera versión también accede al *buffer* de usuario (`out(R_DATOS, *dir++)`) 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 ya que en la misma se está bloqueando la ejecución del proceso actual en el contexto de la rutina de tratamiento de una interrupción al invocar la función auxiliar `programar_dispositivo`. Es erróneo bloquear a un proceso dentro de una rutina de interrupción puesto que esto causaría que no se completase dicha rutina así como las posibles rutinas anidadas a la misma.

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

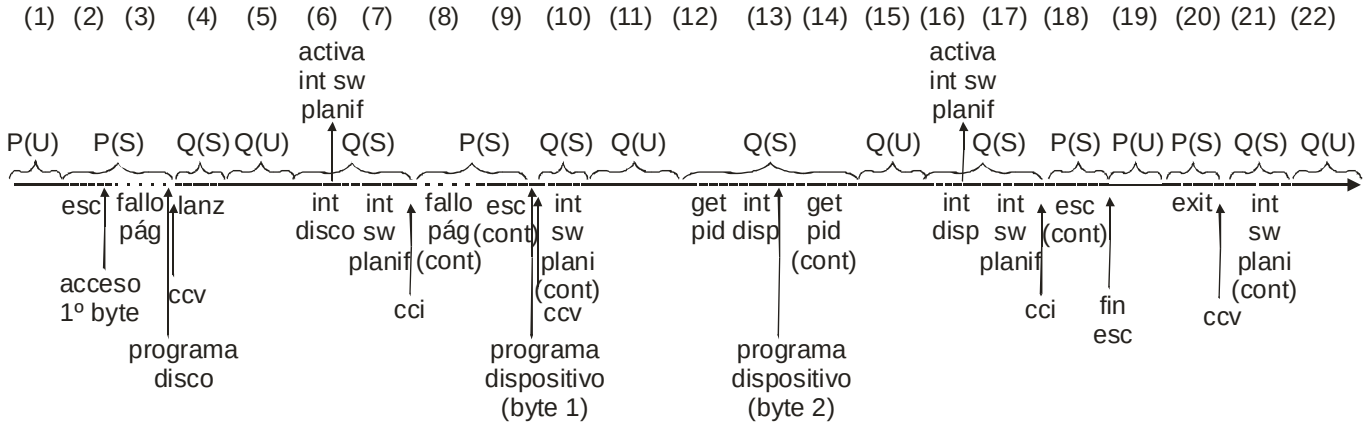


A continuación, se detallan cada uno de los eventos que ocurren durante la traza:

1. **P** en modo usuario realiza una llamada al sistema solicitando escribir dos bytes del dispositivo, especificando como *buffer* una variable global con valor inicial que ocupa dos bytes y está almacenada en una única página no accedida previamente.
2. **P** en modo sistema accede al 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.
3. Al tratarse de una variable global con valor inicial, estará incluida en una región con soporte en el fichero ejecutable, por lo que requerirá acceder a disco para traerla. El tratamiento del fallo de página buscará un marco libre, que por el enunciado sabemos que habrá, y programará la operación en el disco, produciéndose un cambio de contexto voluntario a **Q**.
4. **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.
5. Mientras **Q** ejecuta en modo usuario, se produce una interrupción del disco indicando que ya se leído la página correspondiente.
6. La rutina de interrupción del disco desbloquea al proceso **P**. Dado que el proceso desbloqueado es más prioritario que el actual, se activa una interrupción software no expulsiva de planificación para llevar a cabo el cambio de contexto involuntario de **Q** a **P**.
7. La rutina de interrupción software de planificación lleva a cabo el cambio de contexto involuntario requerido, con lo que **P** continúa el tratamiento del fallo de página justo en el punto donde quedó detenido.
8. **P** en modo sistema completa la rutina de fallo de página que estaba anidada con la llamada al sistema.
9. **P** en modo sistema, dentro de la llamada de escritura, programa el dispositivo para que escriba el primer byte y se bloquea, 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 no expulsiva de planificación para llevar a cabo el cambio de contexto involuntario de **Q** a **P**.
14. **Q** continúa y completa la llamada `getpid`, Nótese que, al ser un núcleo no expulsivo, Q completa la llamada antes de hacer el cambio de contexto involuntario.

15. 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.
16. **P** detecta que quedan datos por escribir, programa el dispositivo para que escriba el segundo byte y se bloquea, produciéndose un cambio de contexto voluntario a **Q**.
17. **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.
18. Durante la ejecución en modo usuario de **Q**, se produce una nueva interrupción del dispositivo.
19. Al detectar que se han escrito todos los datos, 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 no expulsiva de planificación para llevar a cabo el cambio de contexto involuntario de **Q** a **P**.
20. 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.
21. **P** prosigue la llamada, completando la ejecución de la misma al no haber más datos que escribir y retornando a modo usuario.
22. **P** ejecuta en modo usuario hasta que realiza la llamada al sistema para completar su ejecución.
23. La llamada de terminación completa la ejecución de **P**, produciéndose un cambio de contexto voluntario a **Q**.
24. **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.
25. **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 escribir dos bytes del dispositivo, especificando como *buffer* una variable global con valor inicial que ocupa dos bytes y está almacenada en una única página no accedida previamente.
2. **P** en modo sistema accede al 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.
3. Al tratarse de una variable global con valor inicial, estará incluida en una región con soporte en el fichero ejecutable, por lo que requerirá acceder a disco para traerla. El tratamiento del fallo de página buscará un marco libre, que por el enunciado sabemos que habrá, y programará la operación en el disco, produciéndose un cambio de contexto voluntario a **Q**.
4. **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.
5. Mientras **Q** ejecuta en modo usuario, se produce una interrupción del disco indicando que ya se leído la página correspondiente.
6. La rutina de interrupción del disco desbloquea al proceso **P**. Dado que el proceso desbloqueado es más prioritario que el actual, se activa una interrupción software no expulsiva de planificación para llevar a cabo el cambio de contexto involuntario de **Q** a **P**.
7. La rutina de interrupción software de planificación lleva a cabo el cambio de contexto involuntario requerido, con lo que **P** continúa el tratamiento del fallo de página justo en el punto donde quedó detenido.
8. **P** en modo sistema completa la rutina de fallo de página que estaba anidada con la llamada al sistema.
9. **P** en modo sistema, dentro de la llamada de escritura, programa el dispositivo para que escriba el primer byte y se bloquea, 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 detecta que quedan todavía datos por escribir y programa la escritura del segundo byte en el dispositivo, completándose la rutina de interrupción.
14. **Q** continúa y completa la llamada `getpid`, Nótese que al no desbloquearse el proceso **P**, no influye en la traza si el núcleo es expulsivo o no.
15. Durante la ejecución en modo usuario de **Q**, se produce una nueva interrupción del dispositivo.
16. Al detectar que se han escrito todos los datos, 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 no expulsiva de planificación para llevar a cabo el cambio de contexto involuntario de **Q** a **P**.
17. 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.
18. **P** prosigue la llamada, completando la ejecución de la misma al no haber más datos que escribir y retornando a modo usuario.
19. **P** ejecuta en modo usuario hasta que realiza la llamada al sistema para completar su ejecución.
20. La llamada de terminación completa la ejecución de **P**, produciéndose un cambio de contexto voluntario a **Q**.
21. **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.
22. **Q** prosigue su ejecución en modo usuario.

e) Revisemos cada una de las opciones planteadas:

i. Fallos de página: en ambas versiones se produce el mismo número de fallos de página: un único fallo que se genera al acceder al primer byte del *buffer* de usuario.
ii. Interrupciones de los dispositivos: se mantendrá el mismo número (una interrupción por cada byte) con independencia de qué versión se use.
iii. Cambios de contexto voluntarios: en la primera versión se produce un cambio de contexto voluntario por cada byte escrito, mientras que en la segunda se produce un único cambio de contexto voluntario durante toda la operación. Nótese que se ha obviado el cambio de contexto voluntario asociado al fallo de página puesto que está presente en ambas versiones.

f) En la primera versión, cuando se produce la primera interrupción del dispositivo, se desbloqueará el proceso **P**, pero al tener éste menos prioridad que **R**, no lo expulsará y, por tanto, la programación de la escritura del segundo byte no se llevará a cabo hasta que el proceso **R** se bloquee o termine.

En la segunda versión, la programación de la escritura del segundo byte (y posteriores si se tratara de una escritura de mayor tamaño) se lleva a cabo en el contexto de la rutina de interrupción del dispositivo, por lo que la operación progresará mientras sigue la ejecución de **R**. Por tanto, se podría decir que con este segundo diseño se está priorizando a los dispositivos sobre los procesos.