Diseño de Sistemas Operativos. Febrero de 2014.
Ejercicio de procesos y memoria

El S.O. proporciona paralelismo entre las operaciones de E/S de un proceso y la ejecución del mismo usando técnicas como la escritura diferida (*delayed-write*). Para analizar esta técnica, se va a usar un hipotético dispositivo en el que las operaciones de escritura se hacen byte a byte y tal que cuando se completa la escritura de un byte, operación relativamente lenta que tarda unos 10 ms,, se genera una interrupción para notificarlo, quedando el dispositivo listo para volver a ser usado.
Tomamos como punto de partida un fragmento de pseudo-código que corresponde a una primera versión del manejador del dispositivo, **sin** escritura diferida, en un sistema **monoprocesador** con un núcleo de tipo **no expulsivo**. En la misma se usa un esquema donde la programación del dispositivo se realiza en el contexto de la interrupción para minimizar los cambios de contexto, usando un *buffer* de tamaño fijo (*TAM*) para almacenar los datos a escribir. Se trata de una versión simplificada que considera que el dispositivo lo usa de forma exclusiva un proceso y que obvia problemas de sincronización y de errores.

```
struct {...} buf; // tamaño = TAM (p. ej. 128)
cola_procesos cola_disp; int tam_datos, a_esc;
int escritura(char *dir, int tam) {
  tam_datos = tam;
  while (tam_datos>0) {
    a_esc = (min(tam_datos, TAM));
    for (i=0; i<a_esc; i++)
       insertar(&buf, *(dir++)); //buf.nelem++
    programar(); Bloquear(&cola_disp);
    tam_datos -=  a_esc;
  }
}
```

```
void interrupcion() {
  if (CONDICION)
    Desbloquear(&cola_disp);
  else
    programar();
}
void programar(){
  char c =  extraer(&buf); // buf.nelem--
  out(R_DATOS, c);
  out(R_CONTROL, ESCRITURA); }
```

a) En primer lugar, se debe determinar cuál es la condición ***CONDICION***. Para ello, estudie los siguientes casos:
a1) Analice, y explique brevemente, qué ocurre cuando la petición de escritura especifica un tamaño inferior al del *buffer*.
a2) Analice, y explique brevemente, qué sucede cuando la petición de escritura especifica un tamaño superior al del *buffer*.
a3) Teniendo en cuenta los análisis previos, exprese la condición pedida.
Para comparar esta primera versión con la que usa escritura diferida, se usará el siguiente ejemplo de ejecución de procesos:

- **P** (prioridad alta): solicita una primera escritura de 2 bytes del dispositivo 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. Justo después, realiza cálculos durante 15 ms. y, a continuación, solicita una segunda escritura de un byte usando una variable almacenada en esa misma página. Finalmente, al retornar de la llamada al sistema, **P** termina de forma normal.
- **Q** (prioridad baja): Proceso recién creado, que alterna fases de cálculo en modo usuario con llamadas al sistema **`getpid`**.
- Sólo la primera interrupción del dispositivo se producirá estando el proceso en ejecución en modo sistema.

b) Detalle la traza de ejecución planteada identificando las activaciones del S.O. y los cambios de contexto.
Esta segunda versión incluye la técnica de escritura diferida, de manera que al retornar la operación de escritura, se siguen escribiendo de forma diferida los datos al dispositivo.

```
struct {...} buf; // tamaño = TAM (p. ej. 128)
cola_procesos cola_disp; int tam_datos, a_esc;
int disp_activo = 0;
int escritura(char *dir, int tam) {
  tam_datos = tam;
  while (tam_datos>0) {
    a_esc = (min(tam_datos, TAM));
    if (a_esc > TAM - buf.nelem)
      Bloquear(&cola_disp);
    for (i=0; i<a_esc; i++)
       insertar(&buf, *(dir++)); //buf.nelem++
    programar();
    tam_datos -=  a_esc;
  }
}
```

```
void interrupcion() {
  if (COND1)
    programar();
  else
    dispo_activo = 0;
  if (cola_disp.primero != NULL) //proc en espera
    if (COND2) Desbloquear(&cola_disp);
}
void programar() {
  if (!dispo_activo) {
    char c =  extraer(&buf); // buf.nelem--
    out(R_DATOS, c); out(R_CONTROL, ESCRITURA);
    dispo_activo=1;}
}
```

c) En primer lugar, se debe determinar cuáles son las condiciones ***COND1*** y ***COND2***, a través de los siguientes casos:
c1) Explique qué ocurre cuando una petición de escritura pide un tamaño *TAM/4* y se encuentra en el *buffer* con *TAM/2* bytes.
c2) Explique qué sucede cuando una petición de escritura solicita un tamaño *3*TAM/4* y se encuentra en el *buffer* con *TAM/2*.
c3) Explique qué ocurre cuando una petición de escritura solicita un tamaño *TAM* y se encuentra con el *buffer* lleno (*TAM*).
c4) Explique qué sucede cuando una petición de escritura pide un tamaño *3*TAM/2* y se encuentra con el *buffer* lleno (*TAM*).
c5) Teniendo en cuenta los análisis previos, exprese las condiciones pedidas.

d) Repita la traza planteada para la versión con escritura diferida.

Solución

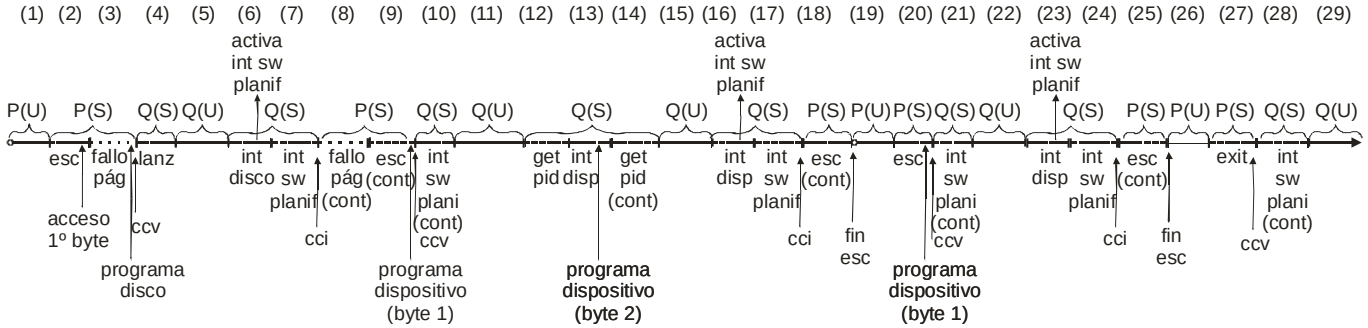
a1) Cuando el tamaño de escritura solicitado es menor o igual que el tamaño del *buffer* del manejador (`TAM`), la función de escritura copia todos los a datos a escribir en el *buffer* interno y programa el dispositivo para que realice la escritura del primer byte y se bloquea hasta que se complete toda la transferencia. Es la propia interrupción la que programará las restantes operaciones de escritura byte a byte del dispositivo, desbloqueando al proceso sólo cuando se haya completado la operación, es decir, en el momento que el *buffer* interno quede vacío.

a2) Si el tamaño solicitado es mayor que el del *buffer* interno, la operación no puede llevarse a cabo con un único bloqueo (en una sola "tanda"). La función de escritura llena el buffer interno y programa el dispositivo para que realice la escritura del primer byte. La interrupción del dispositivo irá programando la escritura de los sucesivos bytes y sólo desbloqueará al proceso cuando el *buffer* se quede vacío. Al ejecutar después del desbloqueo, la función de escritura copiará los siguientes datos al buffer interno, llenándolo en caso de que queden todavía datos para ello, y programará la escritura del primer byte de esta segunda tanda, bloqueándose de nuevo hasta que se vacíe el *buffer*. Este proceso se repetirá hasta que se hayan escrito todos los datos. Nótese que, por tanto, la operación de escritura conllevaría tantas tandas (tantos bloqueos) como la división entera redondeada por exceso entre el tamaño solicitado y el del *buffer* interno, tal que en cada tanda el proceso se desbloquea cuando el *buffer* interno queda vacío.

a3) Del análisis previo se desprende que la condición para desbloquear al proceso es que el *buffer* interno se haya quedado vacío:

```
if (buf.nelem == 0)
  Desbloquear(&cola_disp);
else
  programar();
```

b) La siguiente gráfica muestra la traza de ejecución del ejemplo planteado:



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 procede a copiar el primer byte desde la variable especificada por el programa en la llamada al sistema al *buffer* interno, 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 suponemos 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, copia el segundo byte al *buffer* interno (no pudiendo haber fallo de página al estar ya la página residente), 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 (habrán transcurrido 10 ms. desde la programación 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 (nuevamente, 10 ms. después de la programación).
16. Al detectar que el *buffer* interno está vacío, 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 realizando cálculos durante 15 ms. (evidentemente, la traza no está dibujada a escala) hasta que realiza una segunda llamada al sistema para escribir un byte.
20. **P** en modo sistema procede a copiar el byte almacenado en la variable especificada por el programa al *buffer* interno, no produciéndose un fallo ya que la página ya está residente. A continuación, programa el dispositivo para que escriba un byte y se bloquea, 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** ejecuta en modo usuario hasta que se produce una interrupción del dispositivo (10 ms. después de la programación).
23. Al detectar que el *buffer* interno está vacío, 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**.
24. 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.
25. **P** prosigue la llamada, completando la ejecución de la misma al no haber más datos que escribir y retornando a modo usuario.
26. **P** ejecuta en modo usuario hasta que realiza la llamada al sistema para completar su ejecución.
27. La llamada de terminación completa la ejecución de **P**, produciéndose un cambio de contexto voluntario a **Q**.
28. **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.
29. **Q** prosigue su ejecución en modo usuario.

c1) Cuando una escritura se encuentra que en el *buffer* interno hay espacio suficiente para almacenar los datos a escribir (`TAM/4 < TAM/2`), la operación no requiere realizar ningún bloqueo: simplemente se procede con la copia de los datos del *buffer* del usuario al *buffer* interno (esa copia sí podría producir bloqueos en caso de que se produzcan durante la misma fallos de página que requieran acceso al disco). Nótese que la llamada final a `programar` no hará nada puesto que la escritura diferida sigue estando activa al haber todavía datos en el *buffer* cuando se desbloqueo al proceso.

c2) Cuando el tamaño solicitado por la lectura es mayor que el espacio libre en el *buffer* pero es menor que el tamaño del *buffer* (`3*TAM/4 > TAM/2`), la operación de escritura se bloquea hasta que haya sitio suficiente para copiar los datos (es decir, un hueco de `3*TAM/4`). Será, por tanto, la rutina de interrupción la que realice ese desbloqueo cuando detecte esa condición. Cuando se desbloquee, se copiarán los datos y la llamada se completará. Nótese que la llamada final a `programar` no hará nada puesto que la escritura diferida sigue estando activa al haber todavía datos en el *buffer* cuando se desbloqueo al proceso.

c3) En caso de que coincidan el tamaño de los datos con el del *buffer* y éste esté lleno, como en el caso previo, el proceso se tendrá que bloquear hasta que haya el hueco suficiente que, en este caso, corresponde a cuando el buffer se queda vacío. Cuando se desbloquee, se copiarán los datos y la llamada se completará. Nótese que en este caso sí sería necesario volver a reactivar la escritura diferida ya que ésta se habrá detenido al estar el *buffer* vacío. Por tanto, la llamada final a `programar` sí tendrá efecto en este caso, dejando el dispositivo "vivo" al completar la llamada.

c4) Si el tamaño de los datos solicitados es mayor que el del *buffer* interno, la operación implicará varias tandas, con sus consiguientes bloqueos. En el caso planteado, que se encuentra con el *buffer* interno lleno, en la primera iteración será igual que en el caso previo: se queda bloqueado hasta que el buffer se quede vacío. Cuando se desbloquea, copia `TAM` bytes, programa el dispositivo, que se había quedado "parado", y realiza una segunda iteración en la que se bloquea puesto que quedan por copiar datos (`TAM/2` bytes) y el *buffer* interno está lleno. Como ocurría en el segundo caso, será la rutina de interrupción la que realice ese desbloqueo cuando detecte que ya hay sitio en el *buffer*. Cuando se desbloquee, se copiarán los datos (`TAM/2` bytes) y la llamada se completará. Nótese que en esta segunda iteración la llamada final a `programar` no hará nada puesto que la escritura diferida sigue estando activa al haber todavía datos en el *buffer* cuando se desbloqueo al proceso.

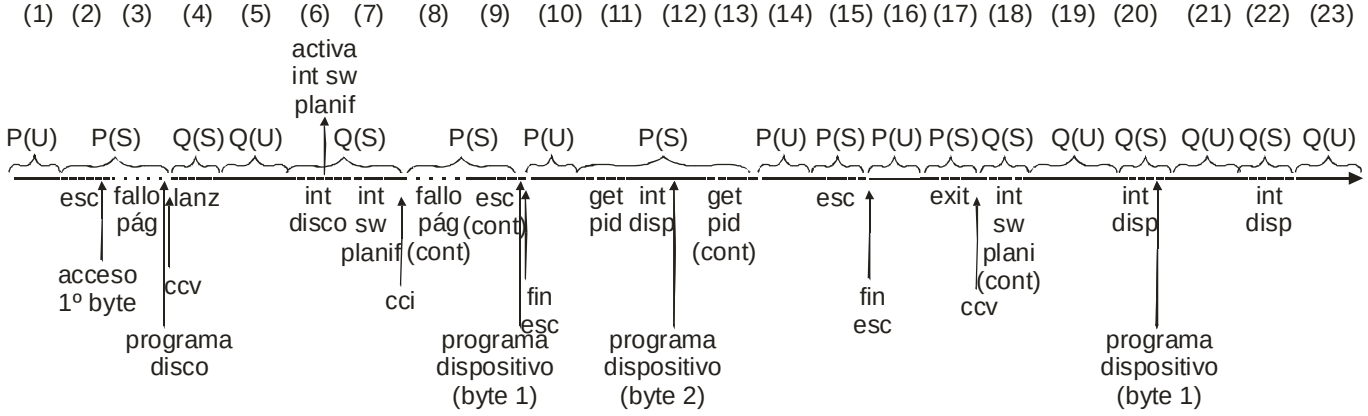
c5) Del análisis previo se desprende que al incluir el uso de escritura diferida, la única condición presente en la primera versión hay que desdoblarla en dos:

- Control de la actividad del dispositivo (*`COND1`*): después de una interrupción sólo se volverá a programar el dispositivo si el *buffer* no está vacío. Esta condición es independiente de si hay alguna petición de escitura pendiente en este momento.
- Control del estado del proceso solicitante (*`COND2`*): esta condición decide en qué momento se desbloquea el proceso, siempre que haya alguno. El proceso se desbloqueará cuando haya espacio suficiente en el *buffer* interno para copiar los

datos correspondientes a la iteración en curso. Para expresar esta condición, se puede usar la variable global `a_esc`. Nótese que, lógicamente, es justo la condición inversa de la que hizo que el proceso se bloqueara (`if (a_esc > TAM - buf.nelem)`).

```
if (buf.nelem > 0)
  programar();
else
  dispo_activo = 0;
if (cola_disp.primero != NULL) //proc en espera
  if (a_esc <= TAM - buf.nelem) Desbloquear(&cola_disp);
```

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



A continuación, se detallan los eventos que ocurren durante la traza. Dado que la traza es igual hasta el noveno paso, para evitar la reiteración, comenzaremos en ese punto.

9. **P** en modo sistema, dentro de la llamada de escritura, copia el segundo byte al *buffer* interno (no pudiendo haber fallo de página al estar ya la página residente), programa el dispositivo para que escriba el primer byte y termina **la llamada al sistema**.
10. **P** ejecuta en modo usuario realizando cálculos y, dado que el enunciado plantea que la primera interrupción del dispositivo se encuentra al proceso actual en modo sistema, vamos a suponer que realiza la llamada al sistema `getpid` (cualquier otro supuesto razonable sería válido).
11. Mientras **P** ejecuta en modo sistema la llamada `getpid`, se produce una interrupción del dispositivo (habrán transcurrido 10 ms. desde la programación del dispositivo).
12. 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.
13. **P** continúa y completa la llamada `getpid`.
14. **P** ejecuta en modo usuario completando los cálculos que le ocupan 15 ms. (evidentemente, la traza no está dibujada a escala) y realiza una segunda llamada al sistema para escribir un byte.
15. **P** en modo sistema procede a copiar el byte almacenado en la variable especificada por el programa al *buffer* interno, no produciéndose un fallo ya que la página ya está residente. A continuación, programa el dispositivo para que escriba un byte y termina **la llamada al sistema**.
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 a modo usuario.
19. Durante la ejecución en modo usuario de **Q**, se produce una nueva interrupción del dispositivo (nuevamente, 10 ms. después de la programación).
20. La rutina de interrupción del dispositivo detecta que quedan todavía datos por escribir y programa la escritura del byte de la segunda escritura en el dispositivo, completándose la rutina de interrupción.
21. **Q** ejecuta en modo usuario hasta que se produce una interrupción del dispositivo (10 ms. después de la programación).
22. Al detectar que el *buffer* interno está vacío, la rutina de interrupción no programa el dispositivo.
23. **Q** prosigue su ejecución en modo usuario.