Diseño de Sistemas Operativos. septiembre de 2012.
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 lectura anticipada (*read-ahead* o *prefetching*). Para analizar esta técnica, se va a usar un hipotético dispositivo en el que las operaciones de lectura se hacen byte a byte y tal que cuando se completa la lectura de un byte, operación relativamente lenta que tarda unos 10 ms,, se genera una interrupción para notificarlo, quedando en el registro de datos el valor leído.
Tomamos como punto de partida un fragmento de pseudo-código que corresponde a una primera versión del manejador del dispositivo, **sin** lectura anticipada, 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 leídos. 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_leer;
int lectura(char *dir, int tam) {
  tam_datos = tam;
  while (tam_datos>0) {
    a_leer = (min(tam_datos, TAM));
    programar(); Bloquear(&cola_disp);
    tam_datos -=  a_leer;
    for (i=0; i<a_leer; i++)
      *(dir++) = extraer(&buf); //buf.nelem--
} }
```

```
void interrupcion() {
  char c =  in(R_DATOS);
  insertar(&buf,c); // buf.nelem++
  if (CONDICION)
    Desbloquear(&cola_disp);
  else
    programar();
}
void programar(){out(R_CONTROL, LECTURA); }
```

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 lectura especifica un tamaño inferior al del *buffer*.
a2) Analice, y explique brevemente, qué sucede cuando la petición de lectura 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 lectura anticipada, se usará el siguiente ejemplo de ejecución de procesos:

- **P** (prioridad alta): solicita una primera lectura de 2 bytes del dispositivo 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. Justo después, realiza cálculos durante 15 ms. y, a continuación, solicita una segunda lectura 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 lectura anticipada, de manera que al completarse una operación, se siguen leyendo por anticipado más datos del dispositivo.

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

```
void interrupcion() {
  char c =  in(R_DATOS);
  insertar(&buf,c); // buf.nelem++
  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) {
    out(R_CONTROL, LECTURA); 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 lectura solicita un tamaño *TAM/4* y se encuentra en el *buffer* con *TAM/2* bytes.
c2) Explique qué sucede cuando una petición de lectura solicita un tamaño *3*TAM/4* y se encuentra en el *buffer* con *TAM/2*.
c3) Explique qué ocurre cuando una petición de lectura solicita un tamaño *TAM* y se encuentra con el *buffer* lleno (*TAM*).
c4) Explique qué sucede cuando una petición de lectura solicita 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 lectura anticipada.

Solución

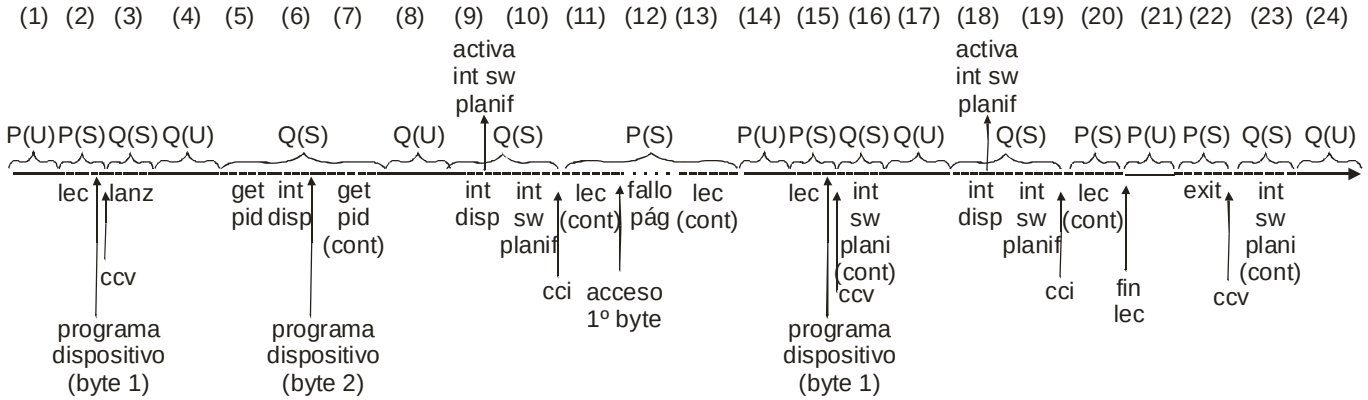
a1) Cuando el tamaño de lectura solicitado es menor o igual que el tamaño del *buffer* del manejador (`TAM`), la función de lectura programa el dispositivo para que realice la lectura del primer byte y se bloquea hasta que se complete toda la transferencia. Es la propia interrupción la que programará las restantes operaciones de lectura byte a byte del dispositivo, desbloqueando al proceso sólo cuando se haya completado la operación. En ese momento, el proceso copiará los datos del buffer interno al especificado por el programa de usuario en la llamada al sistema.

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 interrupción del dispositivo desbloqueará al proceso cuando el *buffer* esté lleno para que éste lo copie a la zona de usuario. Una vez completada la copia, la función de lectura volverá a programar la lectura del siguiente byte del dispositivo, volviéndose a bloquear hasta que se llene el *buffer* o se complete la operación. Nótese que, por tanto, la operación de lectura 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 esté lleno, excepto en la última que lo hará cuando se complete el tamaño pedido.

a3) Del análisis previo se desprende que la condición para desbloquear al proceso es que o bien se haya alcanzado la cantidad de datos solicitada por el proceso, o bien se haya llenado el *buffer* interno. Dado que la variable global `a_leer` almacena justo el mínimo entre esos dos valores, la condición puede expresarse directamente comparando el valor de esa variable y la cantidad de datos almacenada en el *buffer* interno:

```
if (a_leer==buf.nelem)
  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 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 programa el dispositivo para que lea el primer byte y se bloquea, produciéndose un cambio de contexto voluntario a **Q**.
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. **Q** ejecuta en modo usuario hasta que realiza la llamada al sistema `getpid`.
5. Mientras **Q** ejecuta en modo sistema la llamada `getpid`, se produce una interrupción del dispositivo (habrán transcurrido 10 ms. desde la programación del dispositivo).
6. La rutina de interrupción del dispositivo detecta que quedan todavía datos por leer y programa la lectura del segundo byte en el dispositivo, completándose la rutina de interrupción.
7. **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.
8. Durante la ejecución en modo usuario de **Q**, se produce una nueva interrupción del dispositivo (nuevamente, 10 ms. después de la programación).
9. Al detectar que la cantidad de datos almacenada en el *buffer* interno satisface los datos solicitados, 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**.
10. 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.
11. **P** en modo sistema procede a copiar el primer byte almacenado en el *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.

12. Al tratarse de una variable global sin valor inicial, estará incluida en una región sin soporte, por lo que no requerirá acceder a disco para traerla. El tratamiento del fallo de página buscará un marco libre, que suponemos que habrá, y simplemente lo rellenará con ceros, por motivos de seguridad, no requiriéndose ningún cambio de contexto.
13. **P** prosigue la llamada al sistema copiando el segundo byte, lo que no provocará fallo al estar la página ya residente, completando así la ejecución de la misma y retornando **P** a modo usuario.
14. **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 leer un byte.
15. **P** en modo sistema programa el dispositivo para que lea un byte y se bloquea, produciéndose un cambio de contexto voluntario a **Q**.
16. **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.
17. **Q** ejecuta en modo usuario hasta que se produce una interrupción del dispositivo (10 ms. después de la programación).
18. Al detectar que la cantidad de datos almacenada en el *buffer* interno satisface los datos solicitados (sólo 1 byte), 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**.
19. 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.
20. **P** en modo sistema procede a copiar el byte almacenado en el *buffer* interno a la variable especificada por el programa en la llamada al sistema, no produciéndose un fallo ya que la página está residente, completándose así la ejecución de la llamada y retornando **P** a modo usuario.
21. **P** ejecuta en modo usuario hasta que realiza la llamada al sistema para completar su ejecución.
22. La llamada de terminación completa la ejecución de **P**, produciéndose un cambio de contexto voluntario a **Q**.
23. **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.
24. **Q** prosigue su ejecución en modo usuario.

c1) Cuando una lectura se encuentra que en el *buffer* interno se han leído de forma anticipada una cantidad de datos mayor o igual que las solicitadas (`TAM/4 < TAM/2`), la operación no requiere programar el dispositivo ni realizar ningún bloqueo: simplemente se procede con la copia del *buffer* interno al *buffer* del usuario (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 lectura anticipada sigue estando activa (`dispo_activo`).

c2) Cuando el tamaño solicitado por la lectura es mayor que la cantidad de datos almacenados en el *buffer* leídos de forma anticipada (`3*TAM/4 > TAM/2`), la operación de lectura se bloquea hasta que estén disponibles todos los datos. Será, por tanto, la rutina de interrupción la que realice ese desbloqueo cuando detecte esa condición. Nótese que la llamada a `programar` que se hace antes del bloqueo no hará nada puesto que la lectura anticipada sigue estando activa (`dispo_activo`).

c3) En caso de que coincidan el tamaño de lo solicitado y el de lo leído de forma anticipada, la situación es igual a la considerada en el primer escenario: se copian los datos sin ningún tipo de bloqueo. Sin embargo, el caso concreto planteado en este punto especifica, además, que ese tamaño sea igual a `TAM`, lo que implica una diferencia con respecto a ese primer caso: será necesario reactivar la lectura por anticipado ya que ésta se habrá detenido (`!dispo_activo`) al estar el *buffer* lleno. 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 bloques. En el caso planteado, que se encuentra con el *buffer* interno lleno, al solicitar más cantidad que el tamaño de dicho *buffer* (`3*TAM/2 > TAM`), en una primera iteración del bucle se copiarán todos los datos del *buffer* interno al de usuario sin ningún bloqueo. En la segunda iteración, se llamará a `programar`, que sí tendrá efecto en este caso ya que el dispositivo está parado (`!dispo_activo`) al estar inicialmente el *buffer* lleno, y el proceso se bloqueará. Cuando la rutina de interrupción detecte que están disponibles los datos solicitados desbloqueará al proceso.

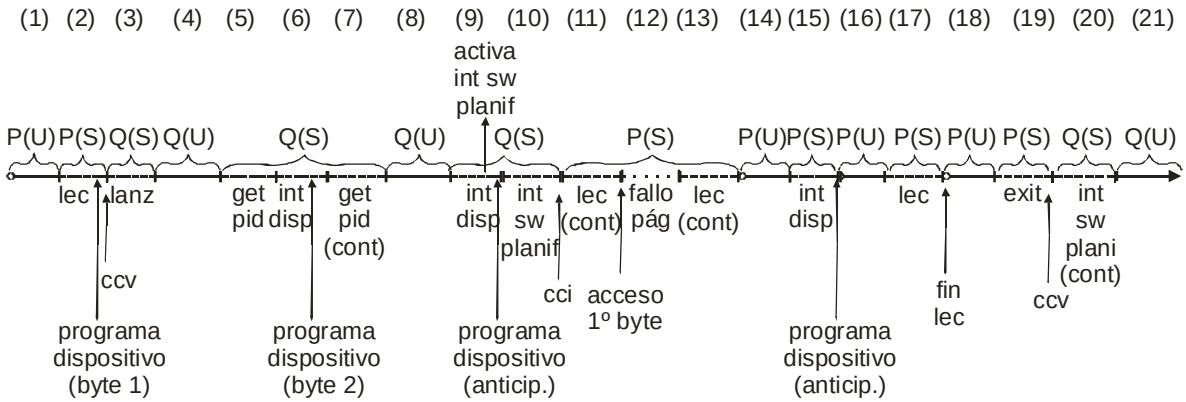
c5) Del análisis previo se desprende que al incluir el uso de lectura anticipada, la única condición presente en la primera versión hay que desdoblarla en dos:

- Control de la actividad del dispositivo (*`COND1`*): después de una interrupción sólo se volverá a programar el dispositivo si el *buffer* no está lleno. Esta condición es independiente de si hay alguna petición de lectura 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. Será igual que en la primera versión: el proceso se desbloqueará cuando estén disponibles todos los datos que necesita, o bien cuando el *buffer* se haya completado. Para expresar esta condición, se puede usar la variable global `a_leer`, igual que en la primera versión.

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

```
if (a_leer==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 decimocuarto paso, para evitar la reiteración, comenzaremos en ese punto. Sin embargo, hay que hacer una matización en el noveno paso debido al uso de la lectura anticipada.

9. La rutina de tratamiento de la segunda interrupción, además de desbloquear al proceso **P**, como el *buffer* no está lleno, programa una nueva operación de lectura en el dispositivo, implementando así la técnica de lectura anticipada.

La traza empezará a divergir a partir de este punto:

14. **P** ejecuta en modo usuario realizando cálculos durante 15 ms., pero, en este caso, durante ese intervalo le llega la interrupción del dispositivo correspondiente a la lectura anticipada.
15. La rutina de interrupción copia el byte en el *buffer* interno y vuelve a programar el dispositivo al no estar lleno. No habrá operaciones de desbloqueo ya que no hay ninguna operación activa.
16. **P** sigue en modo usuario completando sus 15 ms. de cálculos. A continuación, realiza una segunda llamada al sistema para leer un byte.
17. **P** en modo sistema se encuentra con un dato en el *buffer* leído de forma anticipada y procede a copiarlo en la variable especificada por el programa en la llamada al sistema, no produciéndose un fallo ya que la página está residente, completándose así la ejecución de la llamada y retornando **P** a modo usuario.
18. **P** ejecuta en modo usuario hasta que realiza la llamada al sistema para completar su ejecución.
19. La llamada de terminación completa la ejecución de **P**, produciéndose un cambio de contexto voluntario a **Q**. Como parte de la finalización del proceso, se cerrarían los descriptores de fichero, lo que desactivaría la lectura anticipada.
20. **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.
21. **Q** prosigue su ejecución en modo usuario.

Comparando las dos trazas, se puede observar que gracias al uso de la lectura anticipada se ha podido solapar la operación de entrada/salida de un proceso con su propia computación.