Práctica 6: Buffering

En esta práctica vamos a suponer que el dispositivo en cuestión produce interrupciones a una tasa muy elevada, por ejemplo un modem de alta velocidad. Una tasa de interrupciones demasiado elevada provoca que enviar un mensaje a la tarea en cada ejecución de la rutina de interrupción sea un trabajo excesivo y puedan perderse interrupciones. Para evitarlo se emplea la técnica del buffering.

Supondremos que cada ejecución de la rutina de interrupción extrae un dato de un dispositivo. El dato lo almacena en un **buffer acotado**. Sólo cuando el buffer esté próximo a llenarse, la interrupción enviará el mensaje `HARD_INT` a la tarea. El servicio a `HARD_INT` debe sacar todo el contenido del buffer acotado de la rutina de interrupción y ponerlo en un buffer acotado propio. Llamaremos a estos buffers, buffer de bajo nivel (BBN) y de alto nivel (BAN) respectivamente, según ilustra la figura 1. El servicio a `DEV_READ` extrae los datos que se van almacenando en el BAN y los pasa al proceso de usuario.

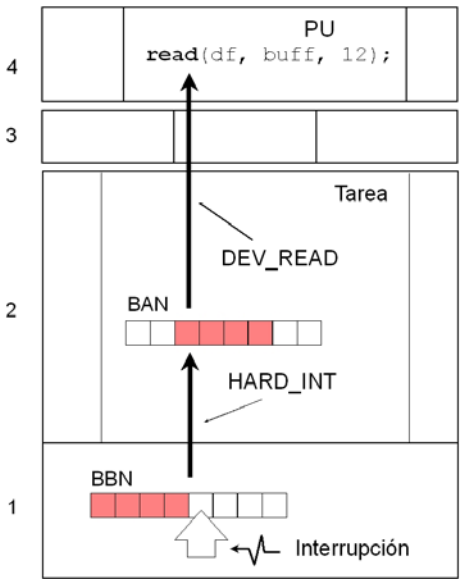

Figura 1. Los buffers de alto nivel y bajo nivel

1. Gestión del buffer de bajo nivel

Vamos a seguir trabajando con la rutina de interrupción del reloj. Esta va a mantener un entero que es incrementado por cada ejecución e introducido en el buffer de bajo nivel (BBN) de la tarea del ratón. La figura 2 muestra este buffer.

Modificación 1.
Se trata de implementar el buffer `bbn` y de que la rutina de interrupción del reloj, `clock_handler`, ponga un ítem en cada ejecución invocando a la función `bbn_write`. Cuando

el buffer esté próximo a llenarse (al 75%), `clock_handler`, debe enviar el mensaje HARD_INT a la tarea del ratón. El servicio a HARD_INT debe sacar el contenido de `bbn` mediante la función `bbn_read` y mostrarlo en pantalla, ítem a ítem. `bbn_read` devolverá 1 cuando saque un ítem y cero cuando el BBN esté vacío. A continuación se proporciona un esbozo de la función `bbn_write`, mientras que la implementación de `bbn_read` se deja como tarea.

```
#define BUF_MAX      256
#define HIGHWATER  ((BUF_MAX * 3) / 4)

struct bbn {
  int buf[BUF_MAX];
  int count;
  int in;
  int out;
};
typedef struct bbn bbn *bbn_t;

void bbn_init(bbn_t bbn)
{
   bbn->count = 0;
   bbn->in    = 0;
   bbn->out   = 0;
}

void bbn_write(bbn_t bbn, int item)
{
  if(bbn->count == BUF_MAX)
    return; /* buffer lleno, descarta el item */
  bbn->buf[in] = item;
  if(++bbn->in == BUF_MAX)
    bbn->in = 0;
  if(++bbn->count == HIGHWATER)
    interrupt(MOUSE);
 }

int bbn_read(bbn_t bbn, int *item)
{
  ...
}
```

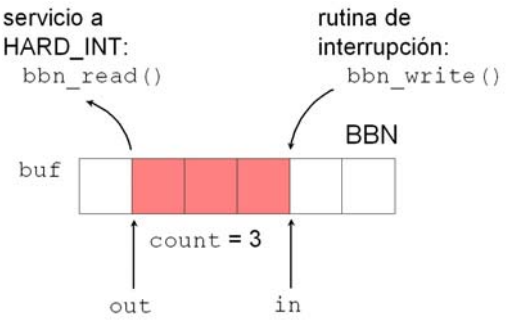

Figura 2. El buffer de bajo nivel

2. Gestión del buffer de alto nivel

En esta ocasión, la tarea del ratón no debe imprimir nada, sino poner en el BAN lo que saca del buffer de bajo nivel. El servicio a DEV_READ quita del BAN los octetos solicitados por el proceso de usuario. Cuando el BAN está vacío se suspende a este según estudiamos en la práctica 5. El proceso de usuario de esta práctica 6 debe escribir en pantalla lo que lee de `/dev/mouse`.

Modificación 2.
Escribir el BAN y sus métodos y utilizarlo según se ha descrito en el párrafo anterior.