Apellidos: **Nombre:**

Sistemas Operativos

Ingeniería Informática de Sistemas

Examen Septiembre 2008

1. **Dispositivo de bloques con buffer** [2.5 puntos]

   Implementar las funciones `buffer_block_read()` y `buffer_block_write()`. Las funciones se diferencian de `block_read/write()` en que guardan un buffer indexado por la función `int hash(int block_number)` de tamaño `NUMBUFFERS` con las últimos bloques usados. Existe un campo `dirty`, para indicar si el contenido es distinto del buffer en disco. La estructura `buffer_dev` esta protegida por el mutex `lock`, cada uno de los buffers individuales esta protegido por `lock_buffer[i]`. Hay que minimizar el tiempo usado con cada uno de los locks adquiridos. La lecturas/escrituras (tanto de disco como a memoria) hay que realizarlas con el `lock_buffer` correspondiente adquirido.

```
#define BLOCKSIZE ...
#define NUMBUFFERS ...
struct buffer {
        char block[BLOCKSIZE];
};
struct buffer_dev {
        mutex lock;
        struct device *dev;
        int block_num[NUMBUFFERS];
        int dirty[NUMBUFFERS];
        mutex lock_buffer[NUMBUFFERS];
        struct buffer buf[NUMBUFFERS];
};
int block_read(struct device *dev, void *buffer, int block_number)}
int block_write(struct device *dev, void *buffer, int block_number)}
```

2. **Algoritmo de segunda oportunidad** [2.5 puntos]

   Implemente en C el algoritmo de segunda oportunidad: `int second_chance(struct frames *frames, int page, int write)` teniendo en cuenta:

   - Si el mutex está ocupado, pasar a la siguiente página.
   - El mutex se bloquea antes de comenzar una escritura de una página. La función `page_write` devuelve el control despues de iniciar la escritura. Cuando la escritura termina, el controlador desbloquea el lock.
   - La tabla de modificaciones de los bits es:

A viejo	M viejo	A nuevo	M nuevo	aclaración
0	0	1	w	se usa esta entrada para la nueva página.
0	1	0	0	Se inicia la escritura de la página.
1	0	0	0	Se marca la página como no accedido.
1	1	0	1	Se marca la página como no accedida.

```
#define NUMFRAMES
struct frame {
        int num_page;    /* Num of page stored here */
        bool access;     /* Acces bit */
        bool modify;     /* Modify bit */
        mutex busy;      /* La pagina se esta ocupada */
};

struct frames {
        int next;
        struct frame[NUMFRAMES];
};

int page_read(int page, int frame);
int page_write(int page, int frame);
void mutex_lock(mutex *m);
void mutex_unlock(mutex *m);
int mutex_trylock(mutex *m);
```

3. **Llamada `file_write()` con extents** [2.5 puntos]

   Defina la función `int file_write(struct file_system *fs, struct disk_inode *ino, void *buffer, int block_num))` que dado un inodo y un bloque dentro del fichero, escribe el bloque correspondiente del sistema de ficheros. Si es necesario asignará espacio. La función `get_next_free()` devuelve el siguiente bloque libre a partir del pasado como argumento.

```
struct file_system {
        int blocksize;
...
};
struct extent {
        int start;
        int size;
};
#define EXTENTS ...
struct disk_inode {
        int num_blocks;
        struct extent e[EXTENTS];
        ...
};
int data_read(struct file_system *fs, void *buffer,
              int block_num);
int data_write(struct file_system *fs, void *buffer,
               int block_num);
int get_next_free(int block_num);
```

4. **Llamada `file_read()` con sistema de ficheros tipo unix** [2.5 puntos]

   Defina la función `int file_read(struct file_system *fs, struct disk_inode *ino, void *buffer, int block_num))` que dado un inodo y un bloque dentro del fichero, lee el bloque correspondiente del sistema de ficheros. Si el número de bloque en el sistema de ficheros es 0, eso indica que debe de leer todo ceros.

```
struct file_system {
        int blocksize;
...
};
#define DIRECTS ...
struct disk_inode {
        int num_blocks;
        int directos[DIRECTS];
        int indirect;
        int indirect2;
        ...
};
int data_read(struct file_system *fs, void *buffer,
              int block_num);
void *memset(void *s, int c, size_t n);
```