OS3: Assessed Coding Exercise 2009-2010

Sheet 2 (of 2)

(hand-in due by 10:00am on Tuesday, 9 March 2010)

This is an assessed exercise, which will count for 12% of the marks in this module.

Attached is a rough design sketch that you can use, along with your own preliminary design, to guide your work on the exercise. The test harness consists of two main components: a simulated disk device and a collection of fake application threads. There are a variety of other useful pieces available, such as a `BoundedBuffer` and the `FreeSectorDescriptorStore`, which you may also use. You are to provide an implementation for the disk driver, matching the spec in `diskdriver.h`.

The code for this exercise can be found in:

`/users/level3/software/public/OS3/assessed-exercise`

Background information (taken from the `.h` files)

The test harness application threads are distinguished by their Pid:

```
/*
 * a Pid is used to distinguish between the different fake applications
 * it is implemented as an unsigned long, but is guaranteed not to exceed
 * MAX_PID
 */

typedef unsigned long Pid;
#define MAX_PID 10
```

The actual block number to read or write on the disk is indicated by its Block:

```
/*
 * Define the data type for a block number on the disk
 */

typedef unsigned long Block;
```

A SectorDescripter contains the Pid of the requesting application, the Block of the sector of interest, and the buffer containing the data to write or a buffer into which the sector data is read.

The disk device supports the following calls – your code may invoke these:

```
/*
 * write sector to disk, returns 1 if successful, 0 if unsuccessful
 * this may take a substantial amount of time to return
 * if unsuccessful, you should return an indication of this lack of
 * success to the application
 */
int write_sector(DiskDevice dd, SectorDescriptor *sd);

/*
 * read sector from disk, returns 1 if successful, 0 if unsuccessful
 * this may take a substantial amount of time to return
 * if unsuccessful, you should return an indication of this lack of
 * success to the application
 */
int read_sector(DiskDevice dd, SectorDescriptor *sd);
```

The fake application threads may make the following calls to your code – you must implement them.

```
/*
 * the following calls are used to write a sector to the disk
 * the nonblocking call must return promptly, returning 1 if successful at
 * queueing up the write, 0 if not (in case internal buffers are full)
 * the blocking call will usually return promptly, but there may be
 * a delay while it waits for space in your buffers.
 * neither call should delay until the sector is actually written to the disk
 * for successful nonblocking call and for the blocking call, a voucher is
 * returned that is required to determine the success/failure of the write
```

```
 */
void blocking_write_sector(SectorDescriptor sd, Voucher *v);
int nonblocking_write_sector(SectorDescriptor sd, Voucher *v);

/*
 * the following calls are used to initiate the read of a sector from the disk
 * the nonblocking call must return promptly, returning 1 if successful at
 * queueing up the read, 0 if not (in case internal buffers are full)
 * the blocking call will usually return promptly, but there may be
 * a delay while it waits for space in your buffers.
 * neither call should delay until the sector is actually read from the disk
 * for successful nonblocking call and for the blocking call, a voucher is
 * returned that is required to collect the sector after the read completes.
 */
void blocking_read_sector(SectorDescriptor sd, Voucher *v);
int nonblocking_read_sector(SectorDescriptor sd, Voucher *v);

/*
 * the following call is used to retrieve the status of the read or write
 * the return value is 1 if successful, 0 if not
 * the calling application is blocked until the read/write has completed
 * if a successful read, the associated SectorDescriptor is returned in sd
 */
int redeem_voucher(Pid p, Voucher v, SectorDescriptor *sd);
```

You must also implement this initialization call:

```
/*
 * called before any other methods to allow you to initialize data
 * structures and to start any internal threads.
 *
 * Arguments:
 *   dd: the DiskDevice that you must drive
 *   mem_start, mem_length: some memory for SectorDescriptors
 *   fsds_ptr: you hand back a FreeSectorDescriptorStore constructed
 *             from the memory provided in the two previous arguments
 */
void init_disk_driver(DiskDevice dd, void *mem_start, unsigned long mem_length,
                      FreeSectorDescriptorStore *fsds_ptr);
```

You may use this initialization call:

```
/* resets the sector descriptor to empty - used by fake applications */
void init_sector_descriptor(SectorDescriptor *sd);
```

You will need to use the `FreeSectorDescriptorStore` facilities. You can create a `FreeSectorDescriptorStore` with:

```
FreeSectorDescriptorStore create_fsds(void);
```

populate it with `SectorDescriptors` created from a memory range using:

```
void create_free_sector_descriptors(FreeSectorDescriptorStore fsds,
                                    void *mem_start, unsigned long mem_length);
```

and then acquire and release `SectorDescriptors` using:

```
/*
 * as usual, the blocking versions only return when they succeed
 * the nonblocking versions return 1 if successful, 0 otherwise
 * the _get_ functions set their final argument if they succeed
 */

void blocking_get_sd(FreeSectorDescriptorStore fsds, SectorDescriptor *sd);
int nonblocking_get_sd(FreeSectorDescriptorStore fsds, SectorDescriptor *sd);

void blocking_put_sd(FreeSectorDescriptorStore fsds, SectorDescriptor sd);
int nonblocking_put_sd(FreeSectorDescriptorStore fsds, SectorDescriptor sd);
```

Finally, given a `SectorDescriptor` you can access the embedded Pid and Block fields using:

```
Pid sector_descriptor_get_pid(SectorDescriptor *sd);
Block sector_descriptor_get_block(SectorDescriptor *sd);
```

What to Hand-in

You must hand-in the following:

A status report (max 1 paragraph), indicating how well your solution matches the spec and how complete it is (partly coded?, compiled?, debugged? etc). Also indicate whether the buffers etc were your own code; if not you must state where you acquired them.

A design diagram (UML diagrams and/or other pictorial representations), showing the dataflows and interactions in your system; hand-drawn diagrams are preferred (to save you time) but they must be legible. This may be similar to, or different from, your initial design. **WARNING:** at most 2/12 marks for this exercise will be awarded for these diagrams, do not spend very much time making them pretty, there are no positive marks for presentation (although marks may be withheld for illegible material).

A brief (max 2 sides) design note, explaining any design features or decisions that you feel are noteworthy. This must also explain precisely when threads can or cannot block, why you made those decisions, and how you have ensured you achieved the indicated behaviour.

A brief (max 1 side) note explaining precisely how your implementation behaves if there is a shortage of `SectorDescriptors`. Justify your design.

An a2ps code listing of your `diskdriver.c` file should be included. You also need to send me a copy of your `diskdriver.c` file as an attachment to an email to joe@dcs.gla.ac.uk. I will be compiling your code to see how well it works, as well as comparing it with the submissions of others to look for potential collusion (this is to be your own work); if you have not used the other modules supplied to you in the TestHarness directory, such as using your own linked list or bounded buffer implementation, you must include those as well, together with a Makefile showing how the program should be built; if I only receive `diskdriver.c`, I will presume that you have built it against the .o files in TestHarness. The emphasis in the mark scheme is on the quality of the code: how well it is engineered and presented.

There will be a hand-in box next to the teaching office into which you may turn in your hardcopy.

All documents must be clearly labeled with your matric number, and stapled together into a single bundle. Please avoid putting your name in the material that is submitted, in particular beware of "author" comments in your code and a2ps putting your name on the printout. Your code should contain the usual boilerplate indicating that it is your own work, consistent with the ethics statement that you have signed at the beginning of the year.