Operating Systems, Program 3
D-shell Shared Memory

Daniel Nix

April 26, 2015

Program Description The D-shell is a simple shell to provide the user with information about processes running on the system. The purpose of the program 3 expansion is to implement shared memory between D-shells, specifically:

- Create a block of shared memory mailboxes with `mboxinit`
- Write to a specific mailbox with `mboxwrite`
- Copy one mailbox to another with `mboxcopy`
- Free the block of shared memory with `mboxdel`

1 Algorithm

1.1 Event Loop

The event loop is handled in main and has not changed from program 2, aside from accepting new shared memory commands. As long as "exit" has not been executed, the program continues to accept commands from the user.

1.2 Parsing

To implement the new functions, the "parse_arg" function was added to split a string of commands into a 2 dimensional character array.

1.3 Shared Memory Implementation

To implement shared memory the c library sys/shm.h was used. It allowed easy creation of a shareable block of memory using a key managed by the OS.

1.3.1 Mailbox Creation

Creating shared mailboxes requires two features to be addressed:

1. Allocate memory to be shared
2. Split shared memory block into individual mailboxes

To allocate memory a call to shmget was made with a request for the appropriate amount of memory and flags IPC_CREAT and IPC_EXCL set. This indicates if shared memory with the given key is already in existence. If shared memory exists we know that reinitializing shared mailboxes unnecessary so the D-shell notifies the user that shared memory has already been set up before returning to the D-shell command line.

If memory has not been set up then the newly allocated memory must be split into individual mailboxes. The shared memory block begins with a small mailbox to hold two integers, the first integer to hold the number of subsequent mailboxes and the second integer to indicate the size (in KB) of each mailbox.

To ensure the read, write, and copy functions behave correctly the first character of all mailboxes is initialized to null.

Synchronization while using mailboxes is discussed in section 1.4.

1.3.2 Read from Mailbox

Mailbox reading requires the memory location of the mailbox. The helper function `box_num_to_addr` is used to convert the box number to base address. Using this address the mailbox is read until a null terminator is found. The write function guarantees that the final character in a mailbox string is a null terminator so the size of a mailbox is not required.

Reading also requires a guarantee that no other D-shells attempt to write to that mailbox at the same time. This means that a semaphore on the mailbox must be set before the read and released upon read completion. Semaphore use is further discussed in section 1.4.1.

1.3.3 Write to Mailbox

Mailbox writing requires the memory location of the mailbox determined by the function `box_num_to_addr`, and the size of a mailbox held in the first mailbox in the shared memory block.

Before writing, the message provided by the user is checked for length. If it is larger than the size of a mailbox minus 1 (to allow for the null terminator) the message length is shortened to fit in a mailbox without overflow. The message is then written to the desired mailbox with a null terminator appended to indicate the end of used memory in the mailbox.

The D-shell must also guarantee that no other D-shells are accessing the mailbox while a write is performed. This is further discussed in section 1.4.

1.3.4 Mailbox Deletion

Before deleting a mailbox block the D-shell must confirm that this instance of the shell created the memory. This is done with a flag set when memory is created set to true if this shell created the memory and false otherwise.

If this shell created the memory then a lock on all memory locations must be made to ensure no other D-shells are accessing the shared memory when it is deleted. This is further discussed in section 1.4.

Once it is confirmed this shell created the shared memory and no other shells are accessing it, mailbox deletion is done with a call to `shmctl` with the IPC_RMID flag set.

1.4 Synchronization

Any time shared memory is implemented the standard synchronization problems including data races, starvation, and deadlock arise. To address these problems the c sys/sem.h library was used.

1.4.1 Data Races

When using shared memory the D-shell must guarantee that memory stays consistent between all shells. For ease of implementation the D-shell allows one shell to read or write to a memory block at a time. This approach was selected for ease of implementation and rate of memory access. If the shared mailboxes were to be used in a high performance computing application then this solution would likely lead to starvation but for a simple message passing system between shells it is effective.

1.4.2 Starvation

As with any shared memory implementation, starvation of processes must be addressed. However, it was not a major concern in D-shell shared memory implementation because of the low rate of memory accesses. Shared memory is only accessed when a user manually requests a read, write, or copy with the respective mbox command. Each access takes only a short amount of time, orders of magnitude less than the fastest rate a user could manually send them, so there is essentially no risk of starvation.

1.4.3 Deadlock

To avoid deadlock the D-shell is only allowed to read or write to one mailbox (except in the case of copying in which case a read followed by a write is done). This guarantees no deadlocks may occur because a read or write may only last a finite amount of time and all processes will finish.

The only concern of deadlock arises when two shells attempt to copy at the same time. For example, shell 1 attempts a copy from mailbox 1 to mailbox 2 while shell 2 attempts a copy from mailbox 2 to mailbox 1. This does not threaten deadlock because, even if both shells began their reads at the same time, the reads would finish, the semaphores would be released, and the writes would be carried out.

2 D-shell Testing

Unit testing for the D-shell shared memory implementation was done in an iterative process. Each time code was added to the shell it was tested to confirm the new code performed the desired function and did not break existing shared memory implementation.

After completing shared memory implementation with unit testing, D-shell regression testing and system was done. All commands previously allowed were tested to confirm that previously written functions were not affected by the newly implemented shared memory code. New functions were also tested to confirm the system worked as a whole.

3 Submission Description

3.1 Compiling Instructions

To compile the source code, unzip the prog3.tar file. From the prog3 directory, type “make" and the source code will be compiled. The dash executable will be placed in the root prog3 directory. Run the D-Shell with the command "./dash". All output from the dash shell is directed to the terminal.

3.2 External Functions

For portability, functions were separated into dash1_funcs.cpp, dash2_funcs.cpp, and dash3_funcs.cpp. The header file dash.h was also created to hold constants.