CSC456 Program 3 - Dash 3.0

Daniel Andrus, Joseph Mowry, & Austin Rotert

Description

DiAgnostic Shell 3.0 contained within. Has the same functionality as the previous version (though signal catching was disabled and left so as it was causing a great number of issues (such as catching and ignoring segmentation faults) during the development and testing of the new features,) with the addition of a middleware mailbox manager that uses shared memory resources to allow communications between different processes. The mailbox manager is presented in a command line style like dash. It is entered by typing `mail` at the dash prompt, with the following functions available to users: `mboxinit`, `mboxdel`, `mboxwrite`, `mboxread`, and `mboxcopy`. Boxes are created by the user with `mboxinit`, and multiple processes can access the boxes at the same time via `mboxwrite`, `mboxread`, and `mboxcopy`, as they are protected from race conditions by POSIX named semaphores. Boxes are also automatically cleaned up upon the exit of the last attached process, but persistent unless deleted through `mboxdel` until that point.

Includes

Holdovers from Dash 2.0 include `iostream` and `fstream` for in / output, `string` and `cstring` for in / output manipulation, `cctype` for formatting, `cstdlib` and `dirent.h` for directory navigation, `signal.h` for signal in / output, `fcntl` for redirection, `sys/time`, `resource`, and `types` for access to certain structures, `sys/wait` to wait for children, `errno.h` to handle errors from external functions, and `unistd.h` for external execution calls.

New additions for 3.0 include: `sys/ipc.h` and `sys/shm.h` for access to shared memory functionality, `semaphore.h` and `pthread.h` for creation of critical sections during creation, deletion, read, write, and copy.

Structure

Within a do-while, takes user input, parses it, and decides how to respond: unknown commands produce error messages, usage messages are printed on incorrect function calls to actual functions, and correct calls initiate one of the below functions. The do-while breaks on `exit`.

Testing

Attempts were made to do a mixture of read, write, delete, and create at the same time: my measures at creating a critical section via a POSIX named semaphore seemed to prevent issues from arising. Additionally, safety measures within the program make it impossible to read or write to nonexistent boxes, as well as prevention of writing a message larger than the boxes' sizes.

Testing `psim` consisted of running the included file `processes.in` through each of the available process scheduling algorithms and cross-referencing the results with that of the theoretical results.

Testing `msim` consisted of running the included file `refstring.in` through each of the available memory paging algorithms and cross-referencing the results with that of the theoretical results.

Compilation

The easiest method for compiling this program is using the included makefile to run make.

```
make dash
```

However, should this fail, you can also manually compile it using the following command:

```
g++ -std=c++11 -g -Wall -o dash dash.cpp mailbox.cpp psim.cpp
msim.cpp mmu.cpp
```

Available Commands

In addition to the commands available in previous versions of Dash, version 3.0 includes the following commands:

- `mail`
  Enter mailbox manipulation mode. From here, you will have access to all mailbox-related commands. Typing “exit” once in mailbox manipulation mode will return you to Dash.
- `psim <file> <sjf | p | rr <quantum>>`
  Runs a process scheduling simulator. As the first argument, supply an existing text file to use for input. The data in the text file needs to follow a very specific format: a list of incoming processes, each separated by a new line, and each entry consisting of three integers: the first is a positive integer declaring the time at which the process is added to the scheduling queue; the second is a positive integer indicating the number of time units the process will take to complete; the third is an integer representing the process' priority, with lower-value numbers representing higher priorities. The command will then output a log of how the specified scheduling algorithm handles and schedules the incoming processes.
  - Argument 1, required: input file
  - Argument 2, required: scheduling algorithm
    Can be one of three options: `sjf` (shortest-job-first), `p` (priority), or `rr` (round robin).
  - Argument 3, required only if argument 2 is `rr`: quantum
    The number of time units the round robin quantum lasts between cycling out a process.

    Must be a positive integer.

- `msim <file> <frames> <fifo | opt | lru | lfu | mfu | sc | c>`
  Runs a memory paging simulator. This simulator illustrates how the selected algorithm performs given a specific reference string and number of pages. Keeps track of page faults and displays the total number of page faults and at what points they occurred.
  - Argument 1, required: text input file of a specific format
    Multiple lines, each containing only a single integer.
  - Argument 2, required: number of frames available to the simulator
    Must be a positive integer
  - Argument 3, required: paging algorithm
    Can be one of seven options: `fifo` (first-in-first-out), `opt` (optimal), `lru` (least recently used), `lfu` (least frequently used), `mfu` (most frequently used), `sc` (second chance), and `c` (clock).
- `mmu <frame size> <virtual pages> <physical pages>`
  Launches the Memory Management Unit simulator. Like the mailbox command, the mmu command enters a special sum-dash mode that reads, parses, and executes commands that are unique to this mode.
  - Argument 1, required: the size in bytes of all simulator frames
    Must be a positive integer that is a power of 2
  - Argument 2, required: number of pages that the virtual address space can support
    Must be a positive integer
  - Argument 3, required: number of pages that the physical memory can contain
    Must be a positive integer

Mailbox Functions:

-
  ```
  /*---------------------------------------------------------------------------
   * Function:        MailboxAttach
   * Purpose:         Attempts to attach to a baseBlock
   * Input args:      N/A
   * Output args:     N/A
   * Return:          Returns true if block was found to attach to, false otherwise
   */
  ```

-
  ```
  /*---------------------------------------------------------------------------
   * Function:        MailboxBeginSem
   * Purpose:         Attempts to attach and checkout / create a semaphore
   * Input args:      N/A
   * Output args:     N/A
  ```

```
 * Return:          Returns false if no block was found to attach, otherwise true
 */
```

-
  ```
  /*--------------------------------------------------------------------------
   * Function:        MailboxEndSem
   * Purpose:         Posts and destroys the previously used semaphore
   * Input args:      N/A
   * Output args:     N/A
   * Return:          N/A
   */
  ```

-
  ```
  /*--------------------------------------------------------------------------
   * Function:        MailboxClientInit
   * Purpose:         Prepares client for use by attaching to an existing block
   *                  or printing a warning
   * Input args:      N/A
   * Output args:     N/A
   * Return:          N/A
   */
  ```

-
  ```
  /*--------------------------------------------------------------------------
   * Function:        MailboxClientDel
   * Purpose:         Destroys client and - if this was the last client attached -
   *                  destroys any remaining mailboxes
   * Input args:      N/A
   * Output args:     N/A
   * Return:          N/A
   */
  ```

-
  ```
  /*--------------------------------------------------------------------------
   * Function:        MailboxClient
   * Purpose:         REPL for the mailbox client
   * Input args:      N/A
   * Output args:     N/A
   * Return:          N/A
   */
  ```

-
  ```
  /*--------------------------------------------------------------------------
   * Function:        MailboxDel
   * Purpose:         Destroys any created mailboxes
   * Input args:      N/A
   * Output args:     N/A
   * Return:          N/A
   */
  ```

-
  ```
  /*--------------------------------------------------------------------------
   * Function:        MailboxInit
   * Purpose:         Creates mailboxes as specified by the user
  ```

```
 * Input args:          num, size
 * Output args:         N/A
 * Return:              Errno on failure (typically EEXIST), 0 otherwise
 */
```

-
  ```
  /*--------------------------------------------------------------------------
   * Function:            MailboxWriter
   * Purpose:             Takes user-entered text and saves it to the specified box
   * Input args:          box
   * Output args:         N/A
   * Return:              N/A
   */
  ```

-
  ```
  /*--------------------------------------------------------------------------
   * Function:            MailboxReader
   * Purpose:             Retreives the contents of a box and prints it
   * Input args:          box
   * Output args:         N/A
   * Return:              N/A
   */
  ```

-
  ```
  /*--------------------------------------------------------------------------
   * Function:            MailboxCopier
   * Purpose:             Copies the contents of one box into the other
   * Input args:          srcNum, destNum
   * Output args:         N/A
   * Return:              N/A
   */
  ```

-
  ```
  /*--------------------------------------------------------------------------
   * Function:            MailboxClientParser
   * Purpose:             Decides how to respond to user input (usage, unknown command,
   *                      etc.)
   * Input args:          cmd
   * Output args:                 N/A
   * Return:              false if the command was exit, otherwise true
   */
  ```
- (Usage functions excluded for brevity)

Available MMU Commands

While in MMU mode, you have access to specific commands that all relate to the memory management unit simulation.

Known Issues

There are a few known issues with Dash 3.0.

1. Piping and/or file redirection does not always behave as intended.
2. The command `msim` only supports reference strings, not distance strings. We were unable to implement this feature due poor project planning on our part and the fact that we didn't know what a distance string was.
3. The command `mmu` is not fully implemented, and thus will display a message when called.