Paul Prince's MPX

R1

Generated by Doxygen 1.7.3

Sun Feb 27 2011 01:28:15

Contents

Chapter 1

Introduction

1.1 Repository

Version-control information is managed by Git, and hosted by GitHub: https://github.com/pprince/cs450

1.2 Documentation

Documentation for developers is generated by Doxygen; for detailed information about the files, functions, data structures, etc. that make up MPX and how they relate to each other, refer to:

- "MPX Programmer's Manual"

which can be found in the doc/ directory. Also, in the same directory, you can find the current version of:

- "MPX User's Manual"

Todo

Generally, documentation is incomplete.

Todo

Generally, we need to make lines break cleanly at 80-columns; Doxygen forces such line-breaks on us in the LaTeX output, but our source code frequently uses longer lines (making the PDF version of the developer manual very ugly!

Chapter 2

Todo List

page Introduction Generally, documentation is incomplete.

Generally, we need to make lines break cleanly at 80-columns; Doxygen forces such line-breaks on us in the LaTeX output, but our source code frequently uses longer lines (making the PDF version of the developer manual very ugly!

File mpx_cmds.c We should typedef structs (particularly struct mpx_command).

Chapter 3

Bug List

Global add_command(char ∗name, void(∗function)(int argc, char ∗argv[])) This function doesn't check for failure to allocate memory for the new command struct.

Global mpx_shell(void) A command should be able to depend on argv[argc] == NULL, but we do not currently implement this feature.

Chapter 4

Data Structure Documentation

4.1 date_rec Struct Reference

Data Fields

- int **month**
- int **day**
- int **year**

The documentation for this struct was generated from the following file:

- mpx/mpx_supt.h

4.2 mpx_command Struct Reference

```
#include <mpx_cmds.h>
```

Data Fields

- char * **name**
- void(* **function**)(int argc, char *argv[ ])
- struct mpx_command * **next**

4.2.1 Detailed Description

Node type for a singly-linked list of MPX commands.

The documentation for this struct was generated from the following file:

- mpx/mpx_cmds.h

4.3 params Struct Reference

Data Fields

- int **op_code**
- int **device_id**
- char ∗ **buf_p**
- int ∗ **count_p**

The documentation for this struct was generated from the following file:

- mpx/mpx_supt.c

Chapter 5

File Documentation

5.1 mpx/mpx.c File Reference

MPX main() function.

```
#include "mpx_supt.h"
#include "mpx_util.h"
#include "mpx_sh.h"
#include "mpx_cmds.h"
```

Functions

- void main (int argc, char ∗argv[ ])

5.1.1 Detailed Description

MPX main() function.

Author

Paul Prince <paul@littlebluetech.com>

Date

2011

This file contains the start-of-execution, i.e. function main(), for MPX, and also the top-level Doxygen documentation that becomes the introductory sections of the developer's manual.

5.1.2 Function Documentation

5.1.2.1 void main (int *argc*, char ∗ *argv[]*)

This is the start-of-execution for the MPX executable.

```
{
        sys_init( MODULE_R1 );  /* System-specific initialization.       */
        init_commands();        /* Initialization for MPX user commands. */
        mpx_shell();            /* Execute the command-handler loop.     */

        /* mpx_shell() should never return, so if we get here, then
         * we should exit with error status (but don't actually...). */
        printf("FATAL ERROR: mpx_shell() returned! That shouldn't happen...\n");
        sys_exit();     /* Terminate, after doing MPX-specific cleanup. */
}
```

5.2 mpx/mpx_cmds.c File Reference

MPX shell commands (help, ls, exit, etc.)

`#include "mpx_cmds.h"`

`#include "mpx_supt.h"`

`#include "mpx_util.h"`

`#include <string.h>`

Functions

- void add_command (char ∗name, void(∗function)(int argc, char ∗argv[ ]))

  Adds a command to the MPX shell.

- void dispatch_command (char ∗name, int argc, char ∗argv[ ])

  Runs the shell command specified by the user, if it is valid.

- void **mpxcmd_commands** (int argc, char ∗argv[ ])
- void mpxcmd_date (int argc, char ∗argv[ ])
- void **mpxcmd_exit** (int argc, char ∗argv[ ])
- void **mpxcmd_help** (int argc, char ∗argv[ ])
- void **mpxcmd_version** (int argc, char ∗argv[ ])
- void **mpxcmd_ls** (int argc, char ∗argv[ ])
- void **init_commands** (void)

Variables

- static struct mpx_command ∗ list_head = NULL

  A linked-list of MPX shell commands.

5.2.1 Detailed Description

MPX shell commands (help, ls, exit, etc.)

Author

Paul Prince <paul@littlebluetech.com>

Date

2011

This file implements each of the user commands for MPX.

Todo

We should typedef structs (particularly struct mpx_command).

5.2.2 Function Documentation

5.2.2.1 void add_command (char ∗ *name,* void(∗)(int argc, char ∗argv[ ]) *function*)

Adds a command to the MPX shell.

Bug

This function doesn't check for failure to allocate memory for the new command struct.

Parameters

in	*name*	The command name that will be made available in the shell.
in	*function*	The C function which will implement the shell command.

```
{
        /* Temporary variable for iterating through the list of commands. */
        struct mpx_command *this_command;

        /* Allocate space for the new command structure. */
        struct mpx_command *new_command =
                (struct mpx_command *)sys_alloc_mem(sizeof(struct mpx_command));
        new_command->name = (char *)sys_alloc_mem(MAX_ARG_LEN+1);
        /* Initialize the structure. */
        strcpy( new_command->name, name );
        new_command->function = function;
        new_command->next = NULL;

        /* Insert the new command into the linked-list of commands. */
        this_command = list_head;
        if ( this_command == NULL ) {
                list_head = new_command;
        } else {
                while ( this_command->next != NULL ){
                        this_command = this_command->next;
                }
                this_command->next = new_command;
        }
}
```

5.2.2.2 void dispatch_command (char * *name*, int *argc*, char * *argv[]*)

Runs the shell command specified by the user, if it is valid.

This function checks to see if the shell command given unabiguously matches a valid MPX shell command, and if so, runs that command (passing the provided argc and argv through).

This dispatcher allows abbreviated commands; if the requested command matches multiple (or zero) valid MPX shell commands, the user is alerted.

Attention

Produces output (via printf)!

```
                                                            {

        /* Temporary variable for iterating through the list of commands. */
        struct mpx_command *this_command = list_head;

        /* Temporary variables to keep track of matching command names. */
        int num_matches = 0;
        struct mpx_command *first_match;

        /* Iterate through the linked list of commands, */
        while( this_command != NULL ) {

                /* Check to see if the given command is a valid abbrev. for the c
      urrent command from the list */
                if( strncmp( this_command->name, name, strlen(name) ) == 0 ) {
                        /* If so, keep track of how many matches thus far, */
                        num_matches++;
                        if (num_matches == 1) {
                                /* This is the first match in the list for the gi
      ven command. */
```

```
                                first_match = this_command;
                        } else if (num_matches == 2) {
                                /* This is the first duplicate match in the list;

                                 * Print out the 'ambiguous command' header,
                                 * plus the first AND current ambiguous commands.
 */
                                printf("Ambiguous command: %s\n", name);
                                printf("    Matches:\n");
                                printf("        %s\n", first_match->name);
                                printf("        %s\n", this_command->name);
                        } else {
                                /* This is a subsequent duplicate match;
                                 * by this time, the header etc. has already been
 printed,
                                 * so we only need to print out the current comma
nd name. */
                                printf("        %s\n", this_command->name);
                        }
                }

                this_command = this_command->next;
        }

        /* If we got a command name that matches unambiguously, run that command.
 */
        if ( num_matches == 1 ){
                first_match->function(argc, argv);
        }

        /* Otherwise, if we got no matches at all, say so. */
        if ( num_matches == 0 ){
                printf("ERROR: Invalid command name.\n");
                printf("Type \"commands\" to see a list of valid commands.\n");
        }
}
```

5.2.2.3 void mpxcmd_date (int *argc*, char ∗ *argv[]*)

< Temp. storage for the return value of sys_ functions.

< Structure to hold a date (day, month, and year). Will be used for both getting and setting the MPX system date.

```
                                                {
        int retval;
        date_rec date;

        if ( argc == 1 ){
                sys_get_date(&date);
                printf("Current MPX system date (yyyy-mm-dd): %04d-%02d-%02d\n",
date.year, date.month, date.day);
                return;
        }
        if ( argc == 4 ){

                date.year  = atoi(argv[1]);
                date.month = atoi(argv[2]);
                date.day   = atoi(argv[3]);

                if ( ! mpx_validate_date(date.year, date.month, date.day) ) {
                        printf("ERROR: Invalid date specified; MPX system date is
 unchanged.\n");
                        printf("       Valid dates are between 1900-01-01 and 299
9-12-31, inclusive.\n");
                        return;
                }

                retval = sys_set_date(&date);
                if ( retval != 0 ) {
                        printf("ERROR: sys_set_date() returned an error.\n");
                        return;
                }

                printf("The MPX system date has been changed.\n");
                return;
        }

        printf("ERROR: Wrong number of arguments to 'date'.\n");
        printf("       Type 'help date' for usage information.\n");
}
```

5.2.3 Variable Documentation

5.2.3.1 struct mpx_command∗ list_head = NULL [static]

A linked-list of MPX shell commands.

5.3 mpx/mpx_sh.c File Reference

MPX Shell, aka Command Handler.

```
#include "mpx_sh.h"
#include "mpx_supt.h"
#include "mpx_util.h"
#include "mpx_cmds.h"
#include <string.h>
```

Functions

- void mpx_setprompt (char ∗new_prompt)

  Sets the current prompt to whatever string is given.

- void mpx_shell (void)

Variables

- static char ∗ mpx_prompt_string = NULL

  The current prompt string.

5.3.1 Detailed Description

MPX Shell, aka Command Handler. This file implements the user interface for MPX.

5.3.2 Function Documentation

5.3.2.1 void mpx_setprompt (char ∗ *new_prompt*)

Sets the current prompt to whatever string is given.

If new_prompt is NULL, this is a no-op.

```
                                      {
        if (new_prompt == NULL) return;
        if (mpx_prompt_string != NULL) {
                sys_free_mem(mpx_prompt_string);
        }
        mpx_prompt_string = (char *)sys_alloc_mem(strlen(new_prompt)+1);
        strcpy(mpx_prompt_string, new_prompt);
}
```

5.3.2.2 void mpx_shell (void)

This function implements the MPX shell (command-line user interface).

mpx_shell() never returns!

Bug

A command should be able to depend on argv[argc] == NULL, but we do not currently implement this feature.

```
                {

  /* A buffer to hold the command line input by the user.
   * We include space for the \r, \n, and \0 characters, if any. */
  char cmdline[ MAX_CMDLINE_LEN+2 ];

  /* Buffer size argument for passing to sys_req(). */
  int line_buf_size = MAX_CMDLINE_LEN;

  /* Used to capture the return value of sys_req(). */
  int err;

  /* argc to be passed to MPX command; works just like the one passed to ma
in(). */
  int argc;
  /* argv array to be passed to MPX command; works almost just like the one
 passed to main().
   *
   * But there is one caveat: argv[argc] is undefined in my implementation,
 not garanteed to be NULL. */
  char **argv;

  /* Temporary pointer for use in string tokenization. */
  char *token;

  /* Delimiters that separate arguments in the MPX shell command-line envir
onment. */
  char *delims = "\t \n";

  /* An index for use in for(;;) loops. */
  int i;
  /* An index for use in nested for(;;) loops. */
  int j;

  /* We must initialize the prompt string. */
  mpx_setprompt(MPX_DEFAULT_PROMPT);

  /* Loop Forever; this is the REPL. */
  /* This loop terminates only via the MPX 'exit' command. */
  for(;;) {
          /* Output the current MPX prompt string. */
          printf("%s", mpx_prompt_string);

          /* Read in a line of input from the user. */
          sys_req( READ, TERMINAL, cmdline, &line_buf_size );

          /* Remove trailing newline. */
          mpx_chomp(cmdline);

          /* Allocate space for the argv argument that is to be sent to an
MPX command. */
          argv = (char **)sys_alloc_mem( sizeof(char**) * (MAX_ARGS+1) ); /
* +1 for argv[0] */
          for( i=0; i < MAX_ARGS+1; i++ ){                               /
* +1 for argv[0] */
                  argv[i] = sys_alloc_mem(MAX_ARG_LEN+1);                /
* +1 for \0 */
```

```
                }


                /* Tokenize the command line entered by the user, and set argc. *
/
                /* 0 is a special value here for argc; a value > 0 after the for
loop indicates
                 * that tokenizing was successful and that argc and argv contain
valid data.
                 *
                 *****  NOTE:  argc includes argv[0], but MAX_ARGS does not!  ***
**/

                argc = 0; token = NULL;

                for( i=0; i < MAX_ARGS+1; i++ ){

                        if (i==0) {
                                token = strtok( cmdline, delims );
                        } else {
                                token = strtok( NULL, delims );
                        }

                        if (token == NULL) {
                                /* No more arguments. */
                                break;
                        }

                        if (strlen(token) > MAX_ARG_LEN) {
                                /* This argument is too long. */
                                printf("ERROR: Argument too long. MAX_ARG_LEN is
%d.\n", MAX_ARG_LEN);
                                argc = 0;
                                break;
                        }

                        argc++;
                        strcpy( argv[i], token );
                }

                if ( strtok( NULL, delims ) != NULL ){
                        /* Too many arguments. */
                        printf("ERROR: Too many arguments. MAX_ARGS is %d.\n", MA
X_ARGS);
                        continue;
                }

                if ( argc <= 0 ) {
                        /* Blank command; just re-print the prompt. */
                        continue;
                }

                /* Run the command, or print an error if it is invalid. */
                dispatch_command( argv[0], argc, argv );

                /* Free the memory for the dynamically-allocated *argv[] */
                for( i=0; i < MAX_ARGS+1; i++ ){
                        sys_free_mem( argv[i] );
                }
                sys_free_mem( argv );
        }
}
```

5.4 mpx/mpx_util.c File Reference

Various utility functions used by all of MPX.

`#include "mpx_util.h"`

`#include "mpx_supt.h"`

`#include <string.h>`

`#include <stdio.h>`

Functions

- int mpx_chomp (char ∗str)
- int **mpx_validate_date** (int year, int month, int day)
- int **mpx_cat** (char ∗file_name)

5.4.1 Detailed Description

Various utility functions used by all of MPX. This file contains the functions etc. to implement the user interface for MPX.

5.4.2 Function Documentation

5.4.2.1 int mpx_chomp (char ∗ *str*)

Removes trailing newline, if any.

This function checks to see if the last character in a string is a newline, and, if so, removes it. Otherwise, the string is left unchanged.

The input must be a valid (allocated and null-terminated) C string, otherwise the results are undefined (but will most likley result in a segmentation fault / protection fault).

Returns the number of characters removed from the string.

Parameters

str	The string to chomp.

```
                              {
        if( strlen(str) > 0 ){
                if( str[ strlen(str)-1 ] == '\n' ){
                        str[ strlen(str)-1 ] = '\0';
                        return 1;
                }
        }
        return 0;
}
```

Index