PMOS Programmers Manual

0.2

Generated by Doxygen 1.7.1

Tue Oct 12 2010 19:58:59

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 mem Struct Reference

`#include <mpx_r2.h>`

Data Fields

- int size

  Number of words in memory.

- unsigned char * loadADDR

  Address to load data to.

- unsigned char * execADDR

  Address of first INSTRUCTION.

3.1.1 Detailed Description

Definition at line 24 of file mpx_r2.h.

3.1.2 Field Documentation

3.1.2.1 unsigned char* mem::execADDR

Address of first INSTRUCTION.

Definition at line 27 of file mpx_r2.h.

3.1.2.2 unsigned char* mem::loadADDR

Address to load data to.

Definition at line 26 of file mpx_r2.h.

3.1.2.3 int mem::size

Number of words in memory.

Definition at line 25 of file mpx_r2.h.

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

- src/mpx_r2.h

3.2 mpx_cmd Struct Reference

`#include <mpx_cmd.h>`

Data Fields

- char * cmd_name
- struct mpx_cmd * next
- void(* cmd_function)(int argc, char *argv[ ])

3.2.1 Detailed Description

Definition at line 13 of file mpx_cmd.h.

3.2.2 Field Documentation

3.2.2.1 void(* mpx_cmd::cmd_function)(int argc, char *argv[ ])

Definition at line 16 of file mpx_cmd.h.

3.2.2.2 char* mpx_cmd::cmd_name

Definition at line 14 of file mpx_cmd.h.

3.2.2.3 struct mpx_cmd* mpx_cmd::next

Definition at line 15 of file mpx_cmd.h.

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

- src/mpx_cmd.h

3.3 page Struct Reference

`#include <mpx_r2.h>`

Data Fields

- PCB * process
  pointer to the PCB structure
- unsigned char * left
  pointer to the left PCB structure
- unsigned char * right
  pointer to the right PCB structure

3.3.1 Detailed Description

Definition at line 44 of file mpx_r2.h.

3.3.2 Field Documentation

3.3.2.1 unsigned char* page::left

pointer to the left PCB structure

Definition at line 46 of file mpx_r2.h.

3.3.2.2 PCB* page::process

pointer to the PCB structure

Definition at line 45 of file mpx_r2.h.

3.3.2.3 unsigned char* page::right

pointer to the right PCB structure

Definition at line 47 of file mpx_r2.h.

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

- src/mpx_r2.h

3.4 process Struct Reference

```
#include <mpx_r2.h>
```

Data Fields

- char name [STRLEN]
  character array containing 16 characters plus space for null

- signed char classType
  class of process APPLICATION or SYSTEM
- signed char priority
  process priority ranges from -128 to +127
- signed char state
  stores the current states of the process
- MEMDSC * memdsc
  stores the description of the ADDRESS SPACE for the process
- STACKDSC * stackdsc
  stores the description of the stack for each process;

3.4.1 Detailed Description

Definition at line 35 of file mpx_r2.h.

3.4.2 Field Documentation

3.4.2.1 signed char process::classType

class of process APPLICATION or SYSTEM

Definition at line 37 of file mpx_r2.h.

3.4.2.2 MEMDSC* process::memdsc

stores the description of the ADDRESS SPACE for the process

Definition at line 40 of file mpx_r2.h.

3.4.2.3 char process::name[STRLEN]

character array containing 16 characters plus space for null

Definition at line 36 of file mpx_r2.h.

3.4.2.4 signed char process::priority

process priority ranges from -128 to +127

Definition at line 38 of file mpx_r2.h.

3.4.2.5 STACKDSC* process::stackdsc

stores the description of the stack for each process;

Definition at line 41 of file mpx_r2.h.

3.4.2.6 signed char process::state

stores the current states of the process

Definition at line 39 of file mpx_r2.h.

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

- src/mpx_r2.h

3.5 root Struct Reference

`#include <mpx_r2.h>`

Data Fields

- int count
- unsigned char * node

3.5.1 Detailed Description

Definition at line 50 of file mpx_r2.h.

3.5.2 Field Documentation

3.5.2.1 int root::count

Definition at line 51 of file mpx_r2.h.

3.5.2.2 unsigned char* root::node

Definition at line 52 of file mpx_r2.h.

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

- src/mpx_r2.h

3.6 stack Struct Reference

`#include <mpx_r2.h>`

Data Fields

- unsigned char * top
  pointer to the top of the stack
- unsigned char * base
  pointer to the bottom of the stack

3.6.1 Detailed Description

Definition at line 30 of file mpx_r2.h.

3.6.2 Field Documentation

3.6.2.1 unsigned char* stack::base

pointer to the bottom of the stack

Definition at line 32 of file mpx_r2.h.

3.6.2.2 unsigned char* stack::top

pointer to the top of the stack

Definition at line 31 of file mpx_r2.h.

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

- src/mpx_r2.h

Chapter 4

File Documentation

4.1 src/mpx.c File Reference

```
#include "mpx_cmd.h"
#include "mpx_util.h"
#include "mpx_supt.h"
```

Functions

- void main ()

  Entry Point of MPX.

4.1.1 Function Documentation

4.1.1.1 void main ( )

Entry Point of MPX.

This is the entry point of MPX, it calls the mpx_command_loop. The Command Loop function starts the display of the Welcome Message and the inital home screen of MPX.

Definition at line 10 of file mpx.c.

```
{
        int err;
        sys_init( MODULE_R1 + MODULE_R2 ); //System initilization
        err = mpx_command_loop();
        errorDecode(err);
}
```

4.2 src/mpx.c

```
00001 #include "mpx_cmd.h"
00002 #include "mpx_util.h"
00003 #include "mpx_supt.h"
```

```
00004
00010 void main(){
00011         int err;
00012         sys_init( MODULE_R1 + MODULE_R2 ); //System initilization
00013         err = mpx_command_loop();
00014         errorDecode(err);
00015 }
```

4.3 src/mpx_cmd.c File Reference

```
#include "mpx_cmd.h"
#include "mpx_util.h"
#include "mpx_r2.h"
#include "mpx_supt.h"
#include "mystdlib.h"
#include <string.h>
#include <stdio.h>
```

Functions

- void mpx_add_command (char *cmd_name, void(*cmd_function)(int argc, char *argv[ ]))
- int mpx_command_loop (void)

  This function displays the Main Screen for mpx.

- void mpxcmd_load (int argc, char *argv[ ])

  This function displays the Directory containing the MPX process files.

- void mpxcmd_help (int argc, char *argv[ ])

  This is a user menu funtion designed to give info about other functions takes one or no inputs.

- void mpxcmd_version (int argc, char *argv[ ])

  The Version function displays MPX version information.

- void mpxcmd_prompt (void)

  The Prompt function allows the user to change the default prompt.

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

  The Date function allows the user to display or change the date of the MPX system.

- void mpxcmd_exit (int argc, char *argv[ ])

  The Exit function allows the user to confirm if they want to exit MPX.

Variables

- char prompt_str [MAX_LINE] = "MPX> "

  Prompt sting stores the default Prompt for MPX.

- char * welcome_message_str = "\n\n Welcome to Perpetual Motion Squad's Operating System.\n\n (type 'help commands') for a list of available commands.)\n\n"

  Welocome Message String stores the Welcome Message for MPX.

- char * anykey_str = "\n<<Press Enter to Continue.>>"

  Any Key String stores the value of the prompt for the user to press return.

- mpx_cmd_t * cmd_head = NULL

4.3.1 Function Documentation

4.3.1.1 void mpx_add_command (char * *cmd_name*, void(*)(int argc, char *argv[ ]) *cmd_function*)

Definition at line 20 of file mpx_cmd.c.

```
{

  /* allocate a command object */
  mpx_cmd_t *command = (mpx_cmd_t*) sys_alloc_mem( sizeof(mpx_cmd_t) ); /*
FIXME need to check for error from alloc func. */

  /* allocate and populate the command name member. */
  command->cmd_name = sys_alloc_mem( strlen(cmd_name)+1 );
  strcpy( command->cmd_name, cmd_name );

  /* populate the command function member. */
  command->cmd_function = cmd_function;

  /* be sure to set the next-command pointer member to NULL, since this wil
l be the new last command. */
  command->next = NULL;

  /* add the command to the global list of commands. */
  if ( cmd_head == NULL ) {
          cmd_head = command;
  } else {
          mpx_cmd_t *last_command = cmd_head;
          while ( last_command->next != NULL ) {
                  last_command = last_command->next;
          }
          last_command->next = command;
  }
}
```

4.3.1.2 int mpx_command_loop (void)

This function displays the Main Screen for mpx.

MPX Command Loop Function dispalays the Main Screen for MPX and functions as the control loop for MPX.

Definition at line 50 of file mpx_cmd.c.

```
{
```

```
  char cmd_line[MAX_LINE];
  char *cmd_argv[MAX_ARGS+1];
  int  cmd_argc;
  int  i;
  mpx_cmd_t *command;

  mpx_add_command( "help", mpxcmd_help );
  mpx_add_command("load", mpxcmd_load );
  mpx_add_command("date", mpxcmd_date );
  mpx_add_command("exit", mpxcmd_exit );
  mpx_add_command("version", mpxcmd_version );
  mpx_add_command("create",mpxcmd_create_PCB);
  mpx_add_command("delete",mpxcmd_delete_PCB);
  mpx_add_command("block",mpxcmd_block);
  mpx_add_command("unblock",mpxcmd_unblock);
  mpx_add_command("suspend",mpxcmd_suspend);
  mpx_add_command("resume",mpxcmd_resume);
  mpx_add_command("setPriority",mpxcmd_setPriority);
  mpx_add_command("show",mpxcmd_show_PCB);
  mpx_add_command("showAll",mpxcmd_showAll_PCB);
  mpx_add_command("showReady",mpxcmd_showReady_PCB);
  mpx_add_command("showBlocked",mpxcmd_showBlocked_PCB);

  for(;;){ /* infinite loop */

          mpx_cls();

          printf("%s", welcome_message_str);

          printf("%s", prompt_str);

          cmd_argc = 0;

          mpx_readline(cmd_line, MAX_LINE-1);

          cmd_argv[0] = strtok(cmd_line, " ");
          cmd_argc++;

          /* cmd_line is invalidated after this point; use cmd_argv[] inste
ad. */

          for(i=0; i<MAX_ARGS; i++){
                  cmd_argv[cmd_argc] = strtok(NULL, " ");
                  if( cmd_argv[cmd_argc] == NULL ){
                          break;
                  }
                  cmd_argc++;
          }

          /* handle too-many-args error condition. */
          if (i == MAX_ARGS && strtok(NULL, " ") != NULL) {
                  printf("ERROR: Argument list too long.\n");
                  printf("%s", anykey_str); mpxprompt_anykey();
                  continue;
          }

          /* run the command function that the user requested,
           * or print an error message if it is not valid. */
          command = cmd_head;
          while (command != NULL) {
                  if ( strcmp(command->cmd_name, cmd_argv[0]) == 0 ) {
                          command->cmd_function( cmd_argc, cmd_argv );
                          break;
                  }
                  command = command->next;
          }
```

```
                /* if we did not find the requested command in the list of comman
      ds,
                 * print an appropriate error message. */
                if ( command == NULL ) {
                        printf("Invalid command.\n");
                        printf("%s", anykey_str); mpxprompt_anykey();
                }
        }
}
```

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

The Date function allows the user to display or change the date of the MPX system.

Definition at line 237 of file mpx_cmd.c.

```
                                        {
        date_rec date;
        sys_get_date(&date);
        printf("\n");
        printf("  System Date:\n");
        printf("    %2d/%2d/%4d\n", date.month, date.day, date.year);
        printf("   (mm/dd/yyyy)\n");
        printf("\n");
        printf("Change it (y/n)? ");
        if( mpxprompt_yn() ) {
                int is_leapyear;
                int max_days;

                printf("\n");

                printf("  New YEAR:  "); date.year      = mpxprompt_int();
                if( !(date.year >=1900 && date.year < 10000) ){
                        /* invalid year entered. */
                        printf("\nInvalid year entered.\n");
                        printf("%s", anykey_str); mpxprompt_anykey();
                        return;
                }

                is_leapyear = ((date.year%4==0 && date.year%100!=0)||(date.year%4
      00==0));

                printf("  New MONTH: "); date.month     = mpxprompt_int();

                switch (date.month) {
                        case 1:
                        case 3:
                        case 5:
                        case 7:
                        case 8:
                        case 10:
                        case 12:
                                max_days = 31;
                        break;

                        case 4:
                        case 6:
                        case 9:
                        case 11:
                                max_days = 30;
                        break;

                        case 2:
                                if( is_leapyear ) {
```

```
                                        max_days = 29;
                                } else {
                                        max_days = 28;
                                }
                        break;

                        default:
                                /* invalid month entered. */
                                printf("\nInvalid month entered.\n");
                                printf("%s", anykey_str); mpxprompt_anykey();
                                return;
                        /* break;  commented out to prevent turbo c++ "unreachabl
      e code" warning. */
                }

                printf("  New DAY:   "); date.day       = mpxprompt_int();

                if( !(date.day > 0 && date.day <= max_days) ){
                        /* invalid day entered. */
                        printf("\nInvalid day entered.\n");
                        printf("%s", anykey_str); mpxprompt_anykey();
                        return;
                }

                /* set the system date. */
                if( sys_set_date(&date) == 0 ){
                        printf("Date successfully set!\n");
                } else {
                        printf("WARNING:\n");
                        printf("sys_set_date() returned error.\n");
                        printf("Date may not have been set.\n");
                }
                printf("%s", anykey_str); mpxprompt_anykey();
        }
        return;
}
```

4.3.1.4 void mpxcmd_exit (int *argc*, char * *argv[ ]*)

The Exit function allows the user to confirm if they want to exit MPX.

Definition at line 321 of file mpx_cmd.c.

```
                                        {
        printf("\n");
        printf("Are you sure you want to terminate MPX?\n");
        if( mpxprompt_yn() ) {
                printf("EXITING.\n");
                sys_exit();
        }
}
```

4.3.1.5 void mpxcmd_help (int *argc*, char * *argv[ ]*)

This is a user menu funtion designed to give info about other functions takes one or no inputs.

Definition at line 162 of file mpx_cmd.c.

```
                                        {
        FILE *fp;
        long fileSize;
```

```
        char* buffer;
        char fileName[100];
        size_t data;
        strcpy(fileName,argv[1]);
        sprintf(buffer,"help\\%s\.txt",fileName);


        if(argc==2){ // specific function help
                fp=fopen(buffer,"r");
                fseek(fp,0,SEEK_END);
                fileSize=ftell(fp);
                rewind(fp);
                buffer = (char*) sys_alloc_mem(sizeof(char)*fileSize);
                data = fread (buffer,1,fileSize,fp);

                printf("%s",buffer);
        }
        else if(argc==1){ // general help
                fp=fopen("help\\help.txt","r");
                fseek(fp,0,SEEK_END);
                fileSize=ftell(fp);
                rewind(fp);
                buffer = (char*) sys_alloc_mem(sizeof(char)*fileSize);
                data = fread (buffer,1,fileSize,fp);
                printf("%s",buffer);
        }
        else{
                printf("Wrong number of arguments used or no such command");
                return;
        }
        fclose(fp);
        printf("%s", anykey_str); mpxprompt_anykey();
        return;
}
```

4.3.1.6 void mpxcmd_load (int *argc*, char * *argv[ ]*)

This function displays the Directory containing the MPX process files.

Definition at line 129 of file mpx_cmd.c.

```
                                        {
        char buf[10];
        char line_buf[MAX_LINE];
        long file_size;
        int  num_mpx_files = 0;

        mpx_cls();

        if( sys_open_dir(NULL) != 0 ){
                printf("WARNING: Failed to open MPX directory!\n");
                printf("%s", anykey_str); mpxprompt_anykey();
                return;
        }

        mpx_pager_init("  Contents of MPX Directory (.mpx Files):\n  ============
    ============================\n    SIZE        NAME\n    ----------  --------------
    ----------------------------\n");
        while( sys_get_entry(buf, 9, &file_size) == 0 ){
                /* snprintf(&line_buf, MAX_LINE, "    %10ld  %s", file_size, buf)
    ; */
                sprintf(&line_buf, "    %10ld  %s", file_size, buf);
                mpx_pager(&line_buf);
                num_mpx_files++;
```

```
        }

        sys_close_dir();

        if (num_mpx_files == 0) {
                printf("\n There aren't any .mpx files in the MPX directory!\n\n"
      );
        }

        printf("%s", anykey_str); mpxprompt_anykey();
        return;
}
```

4.3.1.7 void mpxcmd_prompt (void)

The Prompt function allows the user to change the default prompt.

Definition at line 225 of file mpx_cmd.c.

```
                    {
        printf("\n");
        printf("  Current prompt is: \"%s\"\n", prompt_str);
        printf("\n");
        printf("Enter new prompt: ");
        mpx_readline( prompt_str, MAX_LINE );

        printf("%s", anykey_str); mpxprompt_anykey();
        return;
}
```

4.3.1.8 void mpxcmd_version (int *argc*, char * *argv[ ]*)

The Version function displays MPX version information.

Definition at line 201 of file mpx_cmd.c.

```
                                        {
        mpx_cls();
        printf("\n");
        printf("  ============================================\n");
        printf("  = MPX System Version R1 - September 17, 2010 =\n");
        printf("  ============================================\n");
        printf("\n");
        printf("      by the members of PERPETUAL MOTION SQUAD:\n");
        printf("                         --------- ------ -----\n");
        printf("\n");
        printf("             *  Paul Prince  *\n");
        printf("\n");
        printf("                      *  Nicholas Yanak  *\n");
        printf("\n");
        printf("                            *  Nathan Clay  *\n");
        printf("\n");
        printf("\n");
        printf("  WVU Fall 2010 CS450 w/ Lec. Camille Hayhearst\n");

        printf("%s", anykey_str); mpxprompt_anykey();
        return;
}
```

4.3.2 Variable Documentation

4.3.2.1 char* anykey_str = "\n<<Press Enter to Continue.>>"

Any Key String stores the value of the prompt for the user to press return.

Definition at line 14 of file mpx_cmd.c.

4.3.2.2 mpx_cmd_t* cmd_head = NULL

Definition at line 17 of file mpx_cmd.c.

4.3.2.3 char prompt_str[MAX_LINE] = "MPX> "

Prompt sting stores the default Prompt for MPX.

Definition at line 12 of file mpx_cmd.c.

4.3.2.4 char* welcome_message_str = "\n\n Welcome to Perpetual Motion Squad's Operating System.\n\n (type 'help commands') for a list of available commands.)\n\n"

Welocome Message String stores the Welcome Message for MPX.

Definition at line 13 of file mpx_cmd.c.

4.4 src/mpx_cmd.c

```
00001 #include "mpx_cmd.h"
00002 #include "mpx_util.h"
00003 #include "mpx_r2.h"
00004 #include "mpx_supt.h"
00005 #include "mystdlib.h"
00006 #include <string.h>
00007 #include <stdio.h>
00008
00009
00010 /* Strings */
00011
00012 char prompt_str[MAX_LINE]       = "MPX> ";
00013 char *welcome_message_str       = "\n\n  Welcome to Perpetual Motion Squad's Oper
      ating System.\n\n    (type 'help commands') for a list of available commands.)\n\
      n";
00014 char *anykey_str                = "\n<<Press Enter to Continue.>>";
00017 mpx_cmd_t *cmd_head = NULL;
00018
00019
00020 void mpx_add_command( char *cmd_name, void(*cmd_function)(int argc, char *argv[])
       ) {
00021
00022         /* allocate a command object */
00023         mpx_cmd_t *command = (mpx_cmd_t*) sys_alloc_mem( sizeof(mpx_cmd_t) ); /*
      FIXME need to check for error from alloc func. */
00024
00025         /* allocate and populate the command name member. */
00026         command->cmd_name = sys_alloc_mem( strlen(cmd_name)+1 );
00027         strcpy( command->cmd_name, cmd_name );
00028
00029         /* populate the command function member. */
```

```
00030         command->cmd_function = cmd_function;
00031
00032         /* be sure to set the next-command pointer member to NULL, since this wil
      l be the new last command. */
00033         command->next = NULL;
00034
00035         /* add the command to the global list of commands. */
00036         if ( cmd_head == NULL ) {
00037                 cmd_head = command;
00038         } else {
00039                 mpx_cmd_t *last_command = cmd_head;
00040                 while ( last_command->next != NULL ) {
00041                         last_command = last_command->next;
00042                 }
00043                 last_command->next = command;
00044         }
00045 }
00046
00050 int mpx_command_loop (void) {
00051
00052         char cmd_line[MAX_LINE];
00053         char *cmd_argv[MAX_ARGS+1];
00054         int  cmd_argc;
00055         int  i;
00056         mpx_cmd_t *command;
00057
00058         mpx_add_command( "help", mpxcmd_help );
00059         mpx_add_command("load", mpxcmd_load );
00060         mpx_add_command("date", mpxcmd_date );
00061         mpx_add_command("exit", mpxcmd_exit );
00062         mpx_add_command("version", mpxcmd_version );
00063         mpx_add_command("create",mpxcmd_create_PCB);
00064         mpx_add_command("delete",mpxcmd_delete_PCB);
00065         mpx_add_command("block",mpxcmd_block);
00066         mpx_add_command("unblock",mpxcmd_unblock);
00067         mpx_add_command("suspend",mpxcmd_suspend);
00068         mpx_add_command("resume",mpxcmd_resume);
00069         mpx_add_command("setPriority",mpxcmd_setPriority);
00070         mpx_add_command("show",mpxcmd_show_PCB);
00071         mpx_add_command("showAll",mpxcmd_showAll_PCB);
00072         mpx_add_command("showReady",mpxcmd_showReady_PCB);
00073         mpx_add_command("showBlocked",mpxcmd_showBlocked_PCB);
00074
00075         for(;;){ /* infinite loop */
00076
00077                 mpx_cls();
00078
00079                 printf("%s", welcome_message_str);
00080
00081                 printf("%s", prompt_str);
00082
00083                 cmd_argc = 0;
00084
00085                 mpx_readline(cmd_line, MAX_LINE-1);
00086
00087                 cmd_argv[0] = strtok(cmd_line, " ");
00088                 cmd_argc++;
00089
00090                 /* cmd_line is invalidated after this point; use cmd_argv[] inste
      ad. */
00091
00092                 for(i=0; i<MAX_ARGS; i++){
00093                         cmd_argv[cmd_argc] = strtok(NULL, " ");
00094                         if( cmd_argv[cmd_argc] == NULL ){
00095                                 break;
00096                         }
00097                         cmd_argc++;
```

```
00098                 }
00099
00100                 /* handle too-many-args error condition. */
00101                 if (i == MAX_ARGS && strtok(NULL, " ") != NULL) {
00102                         printf("ERROR: Argument list too long.\n");
00103                         printf("%s", anykey_str); mpxprompt_anykey();
00104                         continue;
00105                 }
00106
00107                 /* run the command function that the user requested,
00108                  * or print an error message if it is not valid. */
00109                 command = cmd_head;
00110                 while (command != NULL) {
00111                         if ( strcmp(command->cmd_name, cmd_argv[0]) == 0 ) {
00112                                 command->cmd_function( cmd_argc, cmd_argv );
00113                                 break;
00114                         }
00115                         command = command->next;
00116                 }
00117
00118                 /* if we did not find the requested command in the list of comman
      ds,
00119                  * print an appropriate error message. */
00120                 if ( command == NULL ) {
00121                          printf("Invalid command.\n");
00122                          printf("%s", anykey_str); mpxprompt_anykey();
00123                 }
00124         }
00125 }
00126
00129 void mpxcmd_load (int argc, char *argv[]) {
00130         char buf[10];
00131         char line_buf[MAX_LINE];
00132         long file_size;
00133         int  num_mpx_files = 0;
00134
00135         mpx_cls();
00136
00137         if( sys_open_dir(NULL) != 0 ){
00138                 printf("WARNING: Failed to open MPX directory!\n");
00139                 printf("%s", anykey_str); mpxprompt_anykey();
00140                 return;
00141         }
00142
00143         mpx_pager_init("  Contents of MPX Directory (.mpx Files):\n  ============
      ==========================\n    SIZE        NAME\n    ----------  --------------
      ----------------------------\n");
00144         while( sys_get_entry(buf, 9, &file_size) == 0 ){
00145                 /* snprintf(&line_buf, MAX_LINE, "    %10ld  %s", file_size, buf)
      ; */
00146                 sprintf(&line_buf, "    %10ld  %s", file_size, buf);
00147                 mpx_pager(&line_buf);
00148                 num_mpx_files++;
00149         }
00150
00151         sys_close_dir();
00152
00153         if (num_mpx_files == 0) {
00154                 printf("\n There aren't any .mpx files in the MPX directory!\n\n"
      );
00155         }
00156
00157         printf("%s", anykey_str); mpxprompt_anykey();
00158         return;
00159 }
00160
00162 void mpxcmd_help(int argc, char *argv[]){
```

```
00163         FILE *fp;
00164         long fileSize;
00165         char* buffer;
00166         char fileName[100];
00167         size_t data;
00168         strcpy(fileName,argv[1]);
00169         sprintf(buffer,"help\\%s\.txt",fileName);
00170 
00171 
00172         if(argc==2){ // specific function help
00173                 fp=fopen(buffer,"r");
00174                 fseek(fp,0,SEEK_END);
00175                 fileSize=ftell(fp);
00176                 rewind(fp);
00177                 buffer = (char*) sys_alloc_mem(sizeof(char)*fileSize);
00178                 data = fread (buffer,1,fileSize,fp);
00179 
00180                 printf("%s",buffer);
00181         }
00182         else if(argc==1){ // general help
00183                 fp=fopen("help\\help.txt","r");
00184                 fseek(fp,0,SEEK_END);
00185                 fileSize=ftell(fp);
00186                 rewind(fp);
00187                 buffer = (char*) sys_alloc_mem(sizeof(char)*fileSize);
00188                 data = fread (buffer,1,fileSize,fp);
00189                 printf("%s",buffer);
00190         }
00191         else{
00192                 printf("Wrong number of arguments used or no such command");
00193                 return;
00194         }
00195         fclose(fp);
00196         printf("%s", anykey_str); mpxprompt_anykey();
00197         return;
00198 }
00199 
00201 void mpxcmd_version (int argc, char *argv[]) {
00202         mpx_cls();
00203         printf("\n");
00204         printf("  ============================================\n");
00205         printf("  = MPX System Version R1 - September 17, 2010 =\n");
00206         printf("  ============================================\n");
00207         printf("\n");
00208         printf("      by the members of PERPETUAL MOTION SQUAD:\n");
00209         printf("                         --------- ------ -----\n");
00210         printf("\n");
00211         printf("             *  Paul Prince  *\n");
00212         printf("\n");
00213         printf("                      *  Nicholas Yanak  *\n");
00214         printf("\n");
00215         printf("                               *  Nathan Clay  *\n");
00216         printf("\n");
00217         printf("\n");
00218         printf("  WVU Fall 2010 CS450 w/ Lec. Camille Hayhearst\n");
00219 
00220         printf("%s", anykey_str); mpxprompt_anykey();
00221         return;
00222 }
00223 
00225 void mpxcmd_prompt (void) {
00226         printf("\n");
00227         printf("  Current prompt is: \"%s\"\n", prompt_str);
00228         printf("\n");
00229         printf("Enter new prompt: ");
00230         mpx_readline( prompt_str, MAX_LINE );
00231 
```

```
00232         printf("%s", anykey_str); mpxprompt_anykey();
00233         return;
00234 }
00235
00237 void mpxcmd_date (int argc, char *argv[]) {
00238         date_rec date;
00239         sys_get_date(&date);
00240         printf("\n");
00241         printf("  System Date:\n");
00242         printf("    %2d/%2d/%4d\n", date.month, date.day, date.year);
00243         printf("   (mm/dd/yyyy)\n");
00244         printf("\n");
00245         printf("Change it (y/n)? ");
00246         if( mpxprompt_yn() ) {
00247                 int is_leapyear;
00248                 int max_days;
00249
00250                 printf("\n");
00251
00252                 printf("  New YEAR:  "); date.year      = mpxprompt_int();
00253                 if( !(date.year >=1900 && date.year < 10000) ){
00254                         /* invalid year entered. */
00255                         printf("\nInvalid year entered.\n");
00256                         printf("%s", anykey_str); mpxprompt_anykey();
00257                         return;
00258                 }
00259
00260                 is_leapyear = ((date.year%4==0 && date.year%100!=0)||(date.year%4
      00==0));
00261
00262                 printf("  New MONTH: "); date.month     = mpxprompt_int();
00263
00264                 switch (date.month) {
00265                         case 1:
00266                         case 3:
00267                         case 5:
00268                         case 7:
00269                         case 8:
00270                         case 10:
00271                         case 12:
00272                                 max_days = 31;
00273                         break;
00274
00275                         case 4:
00276                         case 6:
00277                         case 9:
00278                         case 11:
00279                                 max_days = 30;
00280                         break;
00281
00282                         case 2:
00283                                 if( is_leapyear ) {
00284                                         max_days = 29;
00285                                 } else {
00286                                         max_days = 28;
00287                                 }
00288                         break;
00289
00290                         default:
00291                                 /* invalid month entered. */
00292                                 printf("\nInvalid month entered.\n");
00293                                 printf("%s", anykey_str); mpxprompt_anykey();
00294                                 return;
00295                         /* break;  commented out to prevent turbo c++ "unreachabl
      e code" warning. */
00296                 }
00297
```

```
00298                   printf("  New DAY:   "); date.day       = mpxprompt_int();
00299
00300                   if( !(date.day > 0 && date.day <= max_days) ){
00301                           /* invalid day entered. */
00302                           printf("\nInvalid day entered.\n");
00303                           printf("%s", anykey_str); mpxprompt_anykey();
00304                           return;
00305                   }
00306
00307                   /* set the system date. */
00308                   if( sys_set_date(&date) == 0 ){
00309                           printf("Date successfully set!\n");
00310                   } else {
00311                           printf("WARNING:\n");
00312                           printf("sys_set_date() returned error.\n");
00313                           printf("Date may not have been set.\n");
00314                   }
00315                   printf("%s", anykey_str); mpxprompt_anykey();
00316           }
00317           return;
00318 }
00319
00321 void mpxcmd_exit (int argc, char *argv[]) {
00322           printf("\n");
00323           printf("Are you sure you want to terminate MPX?\n");
00324           if( mpxprompt_yn() ) {
00325                   printf("EXITING.\n");
00326                   sys_exit();
00327           }
00328 }
```

4.5 src/mpx_cmd.h File Reference

Data Structures

- struct mpx_cmd

Defines

- #define MAX_LINE 1024
- #define MAX_ARGS 10

Typedefs

- typedef struct mpx_cmd mpx_cmd_t

Functions

- int mpx_command_loop (void)

  This function displays the Main Screen for mpx.

- void mpxcmd_exit (int argc, char *argv[ ])

  The Exit function allows the user to confirm if they want to exit MPX.

- void mpxcmd_help (int argc, char *argv[ ])

This is a user menu funtion designed to give info about other functions takes one or no inputs.

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

  This function displays the Directory containing the MPX process files.

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

  The Date function allows the user to display or change the date of the MPX system.

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

  The Version function displays MPX version information.

- void mpxcmd_prompt (void)

  The Prompt function allows the user to change the default prompt.

4.5.1 Define Documentation

4.5.1.1 #define MAX_ARGS 10

Definition at line 8 of file mpx_cmd.h.

4.5.1.2 #define MAX_LINE 1024

Definition at line 7 of file mpx_cmd.h.

4.5.2 Typedef Documentation

4.5.2.1 typedef struct mpx_cmd mpx_cmd_t

4.5.3 Function Documentation

4.5.3.1 int mpx_command_loop (void)

This function displays the Main Screen for mpx.

MPX Command Loop Function dispalays the Main Screen for MPX and functions as the control loop for MPX.

Definition at line 50 of file mpx_cmd.c.

```
{

char cmd_line[MAX_LINE];
char *cmd_argv[MAX_ARGS+1];
int  cmd_argc;
int  i;
mpx_cmd_t *command;

mpx_add_command( "help", mpxcmd_help );
mpx_add_command("load", mpxcmd_load );
mpx_add_command("date", mpxcmd_date );
mpx_add_command("exit", mpxcmd_exit );
mpx_add_command("version", mpxcmd_version );
mpx_add_command("create",mpxcmd_create_PCB);
```

```
        mpx_add_command("delete",mpxcmd_delete_PCB);
        mpx_add_command("block",mpxcmd_block);
        mpx_add_command("unblock",mpxcmd_unblock);
        mpx_add_command("suspend",mpxcmd_suspend);
        mpx_add_command("resume",mpxcmd_resume);
        mpx_add_command("setPriority",mpxcmd_setPriority);
        mpx_add_command("show",mpxcmd_show_PCB);
        mpx_add_command("showAll",mpxcmd_showAll_PCB);
        mpx_add_command("showReady",mpxcmd_showReady_PCB);
        mpx_add_command("showBlocked",mpxcmd_showBlocked_PCB);

        for(;;){ /* infinite loop */

                mpx_cls();

                printf("%s", welcome_message_str);

                printf("%s", prompt_str);

                cmd_argc = 0;

                mpx_readline(cmd_line, MAX_LINE-1);

                cmd_argv[0] = strtok(cmd_line, " ");
                cmd_argc++;

                /* cmd_line is invalidated after this point; use cmd_argv[] inste
      ad. */

                for(i=0; i<MAX_ARGS; i++){
                        cmd_argv[cmd_argc] = strtok(NULL, " ");
                        if( cmd_argv[cmd_argc] == NULL ){
                                break;
                        }
                        cmd_argc++;
                }

                /* handle too-many-args error condition. */
                if (i == MAX_ARGS && strtok(NULL, " ") != NULL) {
                        printf("ERROR: Argument list too long.\n");
                        printf("%s", anykey_str); mpxprompt_anykey();
                        continue;
                }

                /* run the command function that the user requested,
                 * or print an error message if it is not valid. */
                command = cmd_head;
                while (command != NULL) {
                        if ( strcmp(command->cmd_name, cmd_argv[0]) == 0 ) {
                                command->cmd_function( cmd_argc, cmd_argv );
                                break;
                        }
                        command = command->next;
                }

                /* if we did not find the requested command in the list of comman
      ds,
                 * print an appropriate error message. */
                if ( command == NULL ) {
                         printf("Invalid command.\n");
                         printf("%s", anykey_str); mpxprompt_anykey();
                }
        }
}
```

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

The Date function allows the user to display or change the date of the MPX system.

Definition at line 237 of file mpx_cmd.c.

```
                                        {
  date_rec date;
  sys_get_date(&date);
  printf("\n");
  printf("  System Date:\n");
  printf("    %2d/%2d/%4d\n", date.month, date.day, date.year);
  printf("   (mm/dd/yyyy)\n");
  printf("\n");
  printf("Change it (y/n)? ");
  if( mpxprompt_yn() ) {
          int is_leapyear;
          int max_days;

          printf("\n");

          printf("  New YEAR:  "); date.year      = mpxprompt_int();
          if( !(date.year >=1900 && date.year < 10000) ){
                  /* invalid year entered. */
                  printf("\nInvalid year entered.\n");
                  printf("%s", anykey_str); mpxprompt_anykey();
                  return;
          }

          is_leapyear = ((date.year%4==0 && date.year%100!=0)||(date.year%4
00==0));

          printf("  New MONTH: "); date.month     = mpxprompt_int();

          switch (date.month) {
                  case 1:
                  case 3:
                  case 5:
                  case 7:
                  case 8:
                  case 10:
                  case 12:
                          max_days = 31;
                  break;

                  case 4:
                  case 6:
                  case 9:
                  case 11:
                          max_days = 30;
                  break;

                  case 2:
                          if( is_leapyear ) {
                                  max_days = 29;
                          } else {
                                  max_days = 28;
                          }
                  break;

                  default:
                          /* invalid month entered. */
                          printf("\nInvalid month entered.\n");
                          printf("%s", anykey_str); mpxprompt_anykey();
                          return;
                  /* break;  commented out to prevent turbo c++ "unreachabl
e code" warning. */
```

```
                }

                printf("  New DAY:   "); date.day       = mpxprompt_int();

                if( !(date.day > 0 && date.day <= max_days) ){
                        /* invalid day entered. */
                        printf("\nInvalid day entered.\n");
                        printf("%s", anykey_str); mpxprompt_anykey();
                        return;
                }

                /* set the system date. */
                if( sys_set_date(&date) == 0 ){
                        printf("Date successfully set!\n");
                } else {
                        printf("WARNING:\n");
                        printf("sys_set_date() returned error.\n");
                        printf("Date may not have been set.\n");
                }
                printf("%s", anykey_str); mpxprompt_anykey();
        }
        return;
}
```

4.5.3.3 void mpxcmd_exit (int *argc*, char * *argv[ ]*)

The Exit function allows the user to confirm if they want to exit MPX.

Definition at line 321 of file mpx_cmd.c.

```
                                        {
        printf("\n");
        printf("Are you sure you want to terminate MPX?\n");
        if( mpxprompt_yn() ) {
                printf("EXITING.\n");
                sys_exit();
        }
}
```

4.5.3.4 void mpxcmd_help (int *argc*, char * *argv[ ]*)

This is a user menu funtion designed to give info about other functions takes one or no inputs.

Definition at line 162 of file mpx_cmd.c.

```
                                        {
        FILE *fp;
        long fileSize;
        char* buffer;
        char fileName[100];
        size_t data;
        strcpy(fileName,argv[1]);
        sprintf(buffer,"help\\%s\.txt",fileName);


        if(argc==2){ // specific function help
                fp=fopen(buffer,"r");
                fseek(fp,0,SEEK_END);
                fileSize=ftell(fp);
                rewind(fp);
                buffer = (char*) sys_alloc_mem(sizeof(char)*fileSize);
```

```
                data = fread (buffer,1,fileSize,fp);

                printf("%s",buffer);
        }
        else if(argc==1){ // general help
                fp=fopen("help\\help.txt","r");
                fseek(fp,0,SEEK_END);
                fileSize=ftell(fp);
                rewind(fp);
                buffer = (char*) sys_alloc_mem(sizeof(char)*fileSize);
                data = fread (buffer,1,fileSize,fp);
                printf("%s",buffer);
        }
        else{
                printf("Wrong number of arguments used or no such command");
                return;
        }
        fclose(fp);
        printf("%s", anykey_str); mpxprompt_anykey();
        return;
}
```

4.5.3.5 void mpxcmd_load (int *argc*, char * *argv[ ]*)

This function displays the Directory containing the MPX process files.

Definition at line 129 of file mpx_cmd.c.

```
                                        {
        char buf[10];
        char line_buf[MAX_LINE];
        long file_size;
        int  num_mpx_files = 0;

        mpx_cls();

        if( sys_open_dir(NULL) != 0 ){
                printf("WARNING: Failed to open MPX directory!\n");
                printf("%s", anykey_str); mpxprompt_anykey();
                return;
        }

        mpx_pager_init("  Contents of MPX Directory (.mpx Files):\n  ============
    ============================\n    SIZE        NAME\n    ----------  --------------
    ----------------------------\n");
        while( sys_get_entry(buf, 9, &file_size) == 0 ){
                /* snprintf(&line_buf, MAX_LINE, "    %10ld  %s", file_size, buf)
    ; */
                sprintf(&line_buf, "    %10ld  %s", file_size, buf);
                mpx_pager(&line_buf);
                num_mpx_files++;
        }

        sys_close_dir();

        if (num_mpx_files == 0) {
                printf("\n There aren't any .mpx files in the MPX directory!\n\n"
    );
        }

        printf("%s", anykey_str); mpxprompt_anykey();
        return;
}
```

4.5.3.6 void mpxcmd_prompt (void)

The Prompt function allows the user to change the default prompt.

Definition at line 225 of file mpx_cmd.c.

```
                    {
        printf("\n");
        printf("  Current prompt is: \"%s\"\n", prompt_str);
        printf("\n");
        printf("Enter new prompt: ");
        mpx_readline( prompt_str, MAX_LINE );

        printf("%s", anykey_str); mpxprompt_anykey();
        return;
}
```

4.5.3.7 void mpxcmd_version (int *argc*, char * *argv[ ]*)

The Version function displays MPX version information.

Definition at line 201 of file mpx_cmd.c.

```
                                        {
        mpx_cls();
        printf("\n");
        printf("  ================================================\n");
        printf("  = MPX System Version R1 - September 17, 2010 =\n");
        printf("  ================================================\n");
        printf("\n");
        printf("      by the members of PERPETUAL MOTION SQUAD:\n");
        printf("                         --------- ------ -----\n");
        printf("\n");
        printf("             *  Paul Prince  *\n");
        printf("\n");
        printf("                      *  Nicholas Yanak  *\n");
        printf("\n");
        printf("                           *  Nathan Clay  *\n");
        printf("\n");
        printf("\n");
        printf("  WVU Fall 2010 CS450 w/ Lec. Camille Hayhearst\n");

        printf("%s", anykey_str); mpxprompt_anykey();
        return;
}
```

4.6 src/mpx_cmd.h

```
00001 #ifndef MPX_CMD_HFILE
00002 #define MPX_CMD_HFILE
00003
00004
00005 /* Symbolic Constants */
00006
00007 #define MAX_LINE        1024
00008 #define MAX_ARGS        10
00009
00010
00011 /* Types */
00012
```

```
00013 typedef struct mpx_cmd {
00014         char *cmd_name;
00015         struct mpx_cmd *next;
00016         void (*cmd_function)(int argc, char *argv[]);
00017 } mpx_cmd_t;
00018
00019
00020 /* Prototypes */
00021
00022 int             mpx_command_loop(void);
00023 void    mpxcmd_exit             (int argc, char *argv[]);
00024 void    mpxcmd_help             (int argc, char *argv[]);
00025 void    mpxcmd_load             (int argc, char *argv[]);
00026 void    mpxcmd_date             (int argc, char *argv[]);
00027 void    mpxcmd_version  (int argc, char *argv[]);
00028 void    mpxcmd_prompt   (void);
00029
00030
00031 #endif
```

4.7 src/mpx_r2.c File Reference

```
#include "mpx_r2.h"
#include "mpx_supt.h"
#include "mystdlib.h"
#include "mpx_util.h"
#include <string.h>
#include <stdio.h>
```

Functions

- PCB * allocate_PCB (void)

  Allocates the memory for a new Process Control Block and returns the pointer to the new PCB location in memory.

- int free_PCB (PCB *pointer)

  This function releases all allocated memory related to a PCB.

- int setup_PCB (PCB *pointer, char *Name, int classType, int state, int priority)

  This Function initializes the contents of a PCB and checks the values if correct returns 0 if not returns 1.

- char * string_PCB (PCB *pointer)

  This function returns a character string with PCB information formatted.

- void insert_PCB (PCB *PCBpointer)

  This function inserts a PCB into its aproprate PCB Queue.

- void insert_PORDR (PCB *PCBpointer, ROOT *queueROOT)

  This function inserts into a queue a element sorted by its priority lower number (higher priority) to high number(lower priority).

- void insert_FIFO (PCB *PCBpointer, ROOT *queueROOT)

In this function we grow the queque to the right no matter of the Priority of the PCB.

- PCB * find_PCB (char *name)

  This function findes a PCB by its identifier (name) and returns a pointer to its structures location.

- void remove_PCB (PCB *process)

  This function removes a pcb and dealocates its resouces takes in a pointer to a PCBs location.

- void mpxcmd_create_PCB (int argc, char *argv[ ])

  This is a user function that interacts with the user to create a PCB structure.

- PCB * copy_PCB (PCB *pointer)

  This function preforms a deep copy of a PCB.

- void mpxcmd_delete_PCB (int argc, char *argv[ ])

  This is a user function in the menu to delete a process it takes the process name as input.

- void mpxcmd_block (int argc, char *argv[ ])

  This is a user function in the menu that puts a process in the blocked state it takes the process name as input.

- void mpxcmd_unblock (int argc, char *argv[ ])

  This is a user function in the menu that puts a process in the unblocked state it takes the process name as input.

- void mpxcmd_suspend (int argc, char *argv[ ])

  This is a user function in the menu that puts a process in the suspend state it takes the process name as input.

- void mpxcmd_resume (int argc, char *argv[ ])

  This is a user function in the menu that puts a process in the ready state if previously blocked and blocked if previously suspended it takes the process name as input.

- void mpxcmd_setPriority (int argc, char *argv[ ])

  This is a user function from the menu it changes the priority of a PCB and takes the name and desired priority as inputs80ij.

- void mpxcmd_show_PCB (int argc, char *argv[ ])

  This is a user command from the menu it is used to show information about a specific PCB.

- void mpxcmd_showAll_PCB (int argc, char *argv[ ])

  This is a user functions that shows name and state of all processes.

- void mpxcmd_showReady_PCB (int argc, char *argv[ ])

  This is a user function that shows all non-suspended processes followed by suspended processes.

- void mpxcmd_showBlocked_PCB (int argc, char *argv[ ])

  This is a user function that shows all blocked processes followed by non-blocked processes.

Variables

- ROOT * rQueue
- ROOT * wsQueue

4.7.1 Function Documentation

4.7.1.1 PCB* allocate_PCB (void)

Allocates the memory for a new Process Control Block and returns the pointer to the new PCB location in memory.

< pointer to the new PCB

< pointer to the Memory Descriptor

<pointer to the stack descriptor

< pointer to the stack low address

Definition at line 16 of file mpx_r2.c.

```
                    {
  PCB *newPCB;
  int i;
  MEMDSC *newMemDsc;
  STACKDSC *newStackDsc;
  unsigned char *stack;


  // Allocate memory to each of the disctenct parts of the PCB
  newStackDsc = (STACKDSC*) sys_alloc_mem(sizeof(STACKDSC));
  newMemDsc = (MEMDSC*) sys_alloc_mem(sizeof(MEMDSC));
  newPCB = (PCB*) sys_alloc_mem(sizeof(PCB));
  stack = (unsigned char*) sys_alloc_mem(STACKSIZE*sizeof(unsigned char));

  if ( stack == NULL ||
           newStackDsc == NULL ||
           newMemDsc == NULL ||
           newPCB == NULL ) return NULL;

  //Setup Memory Descriptor with Default Values for Module 2
  newMemDsc -> size = 0;
  newMemDsc -> loadADDR = NULL;
  newMemDsc -> execADDR = NULL;

  //Setup the Stack

  memset(stack,0,STACKSIZE*sizeof(unsigned char));//ZERO out Stack to aid i
n debug....
  newStackDsc -> base = stack; // x86 arch Stacks start at the Higest value

  newStackDsc -> top  = stack[STACKSIZE-1];// and go to lowest or n - 2 for
 Word alloc

  //Bundling Opereations of Stack Descripter Bellow
  newPCB -> stackdsc = newStackDsc;  // stack descriptor is placed in the P
CB

  //Bundling Operations of Memory Descriptor
  newPCB -> memdsc = newMemDsc; // memory descriptor is placed in the PCB

  return newPCB;
```

```
};
```

4.7.1.2 PCB* copy_PCB (PCB * *pointer*)

This function preforms a deep copy of a PCB.

Definition at line 404 of file mpx_r2.c.

```
                        {
                PCB *tempPCB = allocate_PCB();
                memcpy(tempPCB,pointer,sizeof(PCB));
                memcpy(tempPCB -> memdsc, pointer -> memdsc , sizeof(MEMDSC));
                memcpy(tempPCB -> stackdsc ,pointer -> stackdsc, sizeof(STACKDSC)
      );
        return tempPCB;
}
```

4.7.1.3 PCB* find_PCB (char * *name*)

This function findes a PCB by its identifier (name) and returns a pointer to its structures location.

Definition at line 286 of file mpx_r2.c.

```
                        {
        ELEM *incr;
        incr =  rQueue -> node; //set node to the first node in the queque
        while ( strcmp(name,incr -> process -> name ) != 0 && incr != NULL){ // P
      rocess with the lowest priority goes first
                        incr= incr -> right; // progrees to the right
        }
        if (incr == NULL ){
        incr =  wsQueue -> node; //set node to the first node in the queque
        while ( strcmp(name,incr -> process -> name ) != 0 && incr != NULL){ // P
      rocess with the lowest priority goes first
                        incr= incr -> right; // progrees to the right
        }
        }
        if ( incr -> process != NULL && incr != NULL ){
                return incr->process;
        }else{
                return NULL;
        }

}
```

4.7.1.4 int free_PCB (PCB * *pointer*)

This function releases all allocated memory related to a PCB.

< is a pointer to the stack descriptor

< is a pointer to the base location of the stack

< is a pointer to a Memory Descriptor

< holder for error capture on use of sys_free_mem

Definition at line 58 of file mpx_r2.c.

```
                                                                {
        STACKDSC *stackdscptr = pointer -> stackdsc;
        unsigned char *stack = stackdscptr -> base;
        MEMDSC *memptr = pointer -> memdsc;

        int err;

        //Free Stack First
        err = sys_free_mem(stack);
        if( err < 0 ) return err;
        //Second free Stack Descriptor
        err = sys_free_mem(stackdscptr);
        if( err < 0 ) return err;
        //Third free Memory Descriptor
        err = sys_free_mem(memptr);
        if( err < 0 ) return err;
        //Finaly free Process Control block
        err = sys_free_mem(pointer);
        if(err < 0 ) return err;

        return 0; //freed mem ok
}
```

4.7.1.5 void insert_FIFO (PCB * *PCBpointer*, ROOT * *queueROOT*)

In this function we grow the queque to the right no matter of the Priority of the PCB.

Definition at line 255 of file mpx_r2.c.

```
                                                { //FIXME: NO ERROR HANDLING
        ELEM *node; // declare node of type element
        ELEM *incr;


        node = sys_alloc_mem( sizeof(ELEM)); // allocate Memory for node
        node -> process = PCBpointer;// add the PCB to the node

        if( queueROOT -> node == NULL ){ // if this is the first element ever in
      the queque
                node -> left = NULL; // set the link left to null
                node -> right = NULL;// set the link right to null
                queueROOT -> node = node; // Set the first element in the queque
      to node of Type Element
                queueROOT -> count += 1; // increase count by one
                return; //exit out first node is in queque.
        }


        /* INSERT INTO THE QUEQUE IN FIFO ORDER*/
        incr = queueROOT -> node; //set node to the first node in the queque
        while( incr -> right != NULL ){
                incr = incr -> right; // progress forward to the right of the que
      que
        }
         incr -> right = node;
         node -> left = incr; //set left to previous node
         node -> right = NULL; // set right to null
         queueROOT -> count += 1; // increase count by one as the size of the que
      que has grown by one

         return;

}
```

4.7.1.6 void insert_PCB (PCB * *PCBpointer*)

This function inserts a PCB into its aproprate PCB Queue.

Definition at line 141 of file mpx_r2.c.

```
                                                                    {
   int ORD;

   if ( PCBpointer -> state == READY || PCBpointer -> state == RUNNING ){
                ORD  = PORDR;
        }
        if( PCBpointer -> state == BLOCKED ||
                PCBpointer -> state == SUSPENDED_READY ||
                PCBpointer -> state == SUSPENDED_BLOCKED ){
                ORD  = FIFO;
        }

   switch(ORD){
                case PORDR:
                        insert_PORDR(PCBpointer,rQueue);
                        break;
                case FIFO:
                        insert_FIFO(PCBpointer,wsQueue);
                        break;
                default:
                        printf("ORDER not Valid");
                        break;
                };
}
```

4.7.1.7 void insert_PORDR (PCB * *PCBpointer*, ROOT * *queueROOT*)

This function inserts into a queue a element sorted by its priority lower number (higher priority) to high number(lower priority).

Definition at line 166 of file mpx_r2.c.

```
                                                { //FIXME: NO ERROR CHECKING

        ELEM *node; // declare node of type element
        ELEM *incr;
        ELEM *temp1;
        char line[MAX_LINE];
        char *lp;
        lp = &line;
        node = sys_alloc_mem( sizeof(ELEM)); // allocate Memory for node
        node -> process = PCBpointer;// add the PCB to the node

        if( queueROOT -> node == NULL ){ // if this is the first element ever in
      the queque
                node -> left = NULL;
                node -> right = NULL;
                queueROOT -> node = node; // Set the first element in the queque
      to node of Type Element
                queueROOT -> count += 1; // increase count by one
                return; //exit out first node is in queque.
        }

        incr = queueROOT -> node; //set node to the first node in the queque
        while ( incr -> process -> priority <= node -> process -> priority  ){ //
       Process with the lowest priority goes first
                     lp = string_PCB(incr->process);
```

```
                printf("%s\n",lp);
                if( incr->right == NULL) break;
                incr = incr -> right; // progrees to the right

}

/* There are three cases to check for head, tail, and middle*/

/*head case*/
if ( incr -> left == NULL && incr-> right == NULL){
        if( incr -> process -> priority <= node -> process -> priority ){

                node-> left = incr;
                node-> right = NULL;
                incr->right = node;
                queueROOT->count +=1;
        }else{
                node->left = NULL;
                node->right = incr;
                incr->left = node;
                queueROOT -> node = node; //set quequeROOT to new head
                queueROOT ->count +=1;
        }
        return;
}
if( incr -> left == NULL && incr->right != NULL ){
        node->left = NULL;
        node->right = incr;
        incr->left = node;
        queueROOT -> node = node; //set quequeROOT to new head
        queueROOT ->count +=1;
        return;
}

/*tail case*/
if( incr -> left != NULL && incr->right == NULL ){

        if( incr -> process -> priority <= node -> process -> priority ){

                node-> left = incr;
                node-> right = NULL;
                incr->right = node;
                queueROOT->count +=1;
                return;
        }else{
                incr = incr -> left; //decrement incr
                temp1 = incr -> right;
                incr->right = node;
                node->right = temp1;
                node->left = incr;
                temp1->left = node;
                queueROOT->count +=1;
                return;
        }

}

/*middle case*/
if( incr -> left != NULL && incr->right != NULL){
        incr = incr -> left;
        temp1 = incr -> right;
        incr->right = node;
        node->right = temp1;
        node->left = incr;
        temp1->left = node;
        queueROOT->count +=1;
        return;
```

```
        }
}
```

4.7.1.8 void mpxcmd_block (int *argc*, char * *argv[ ]*)

This is a user function in the menu that puts a process in the blocked state it takes the process name as input.

Definition at line 430 of file mpx_r2.c.

```
                                        {
        if(argc==2){
                char name[STRLEN];

                PCB *pointer;
                PCB *tempPCB;
                int buffs = STRLEN;

                strcpy(name,argv[1]);

                pointer = find_PCB(name);
                if ( pointer != NULL){
                        tempPCB = copy_PCB(pointer);
                        remove_PCB(pointer);
                        if( tempPCB -> state == READY || tempPCB -> state ==
    RUNNING ) tempPCB -> state = BLOCKED;
                        if( tempPCB -> state == SUSPENDED_READY ) tempPCB -> stat
    e = SUSPENDED_BLOCKED;
                        insert_PCB(tempPCB);
                }else{
                        printf("Process Name not found!");
                        return;
                }
        }
        else{
                printf("Wrong number of arguments used");
                return;
        }
}
```

4.7.1.9 void mpxcmd_create_PCB (int *argc*, char * *argv[ ]*)

This is a user function that interacts with the user to create a PCB structure.

Definition at line 369 of file mpx_r2.c.

```
                                                {
        static int count;
        char name[STRLEN];
        char line[MAX_LINE];
        int type;
        int priority;

        PCB *newPCB = allocate_PCB();

        if( count == ZERO ){
                rQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
                wsQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
        }
```

```
        printf("Process Name: \n");
        mpx_readline(name, STRLEN);
        printf("Process Class Type ( Application 0 or System  1): \n" );
        type= mpxprompt_int();
        printf("Process Priority (-128 to 127): \n");
        priority = mpxprompt_int();



        if ( setup_PCB(newPCB,&name,type,READY,priority) == 1){
                printf("Incrorrect information entered.");
                mpxprompt_anykey();
                return;
        }

        insert_PCB(newPCB);
        count++;//Update the number of times the function has run.
}
```

4.7.1.10 void mpxcmd_delete_PCB (int *argc*, char * *argv[ ]*)

This is a user function in the menu to delete a process it takes the process name as input.

Definition at line 412 of file mpx_r2.c.

```
                                                {
        if (argc == 2){
        char name[STRLEN];
        PCB *pointer;
        strcpy(name,argv[1]);

        pointer = find_PCB(name);

        if ( pointer != NULL){
                remove_PCB(pointer);
        }else{
                printf("Process Name not found!");
                return;
        }
        }
}
```

4.7.1.11 void mpxcmd_resume (int *argc*, char * *argv[ ]*)

This is a user function in the menu that puts a process in the ready state if previously blocked and blocked if previously suspended it takes the process name as input.

Definition at line 514 of file mpx_r2.c.

```
                                        {
        if(argc==2){
                char name[STRLEN];
                PCB *pointer;
                PCB *tempPCB;
                int buffs = STRLEN;

                strcpy(name,argv[1]);

                pointer = find_PCB(name);
                if ( pointer != NULL){
                        tempPCB = copy_PCB(pointer);
```

```
                        remove_PCB(pointer);
                        if( tempPCB -> state == SUSPENDED_READY ) tempPCB -> stat
      e = READY;
                        if( tempPCB -> state == SUSPENDED_BLOCKED ) tempPCB -> st
      ate = BLOCKED;
                        insert_PCB(tempPCB);
                }else{
                        printf("Process Name not found!");
                        return;
                }
        }
        else{
                printf("Wrong number of arguments used");
                return;
        }
}
```

4.7.1.12 void mpxcmd_setPriority (int *argc*, char * *argv[ ]*)

This is a user function from the menu it changes the priority of a PCB and takes the name and desired priority as inputs80ij.

Definition at line 542 of file mpx_r2.c.

```
                                                {
        if(argc==3){
                char name[STRLEN];
                PCB *pointer;
                int priority;
                PCB *tempPCB;
                int buffs = STRLEN;
                priority  = atoi(argv[2]);
                strcpy(name,argv[1]);
                if( priority <= 128 || priority >= -127){ ;}else{
                        printf("Number entered out of range!");
                        mpxprompt_anykey();
                        return;
                }
                pointer = find_PCB(name);
                if ( pointer != NULL){
                        pointer -> priority = priority;
                        if( pointer -> state == READY ){
                                        tempPCB = copy_PCB(pointer);
                                        remove_PCB(pointer);
                                        insert_PCB(tempPCB);
                                        }
                }else{
                        printf("Process Name not found!");
                        mpxprompt_anykey();
                        return;
                }
        }
        else{
                printf("Wrong number of arguments used");
                mpxprompt_anykey();
                return;
        }
}
```

4.7.1.13 void mpxcmd_show_PCB (int *argc*, char * *argv[ ]*)

This is a user command from the menu it is used to show information about a specific PCB.

Definition at line 579 of file mpx_r2.c.

```
                                                {
        if(argc==2){
                char name[STRLEN];
                PCB *pointer;
                char class[30];
                char state[45];
                int buffs = STRLEN;
                char line[MAX_LINE];
                char* lp;
                lp = &line;

                strcpy(name,argv[1]);

                pointer = find_PCB(name);

                if ( pointer != NULL){
                        printf("%s\n",string_PCB(pointer));
                        mpxprompt_anykey();
                }else{
                        printf("Process Name not found!");
                        mpxprompt_anykey();
                        return;
                }
        }
        else{
                printf("Wrong number of arguments used");
                mpxprompt_anykey();
                return;
        }
}
```

4.7.1.14 void mpxcmd_showAll_PCB (int *argc*, char * *argv[]*)

This is a user functions that shows name and state of all processes.

Definition at line 611 of file mpx_r2.c.

```
                                                { // Pagination function needs add
      ed !!Function still needs work!!
        if(argc==1){
                ELEM *incr;
                PCB *pointer;
                char line[MAX_LINE];
                char* lp;
                char class[30];
                char state[45];
                //set node to the first node in the queque
                lp = &line;
                mpx_pager_init(" All PCB's In Queue:\n -------------------------
      ---------------------------\n");

                printf("%d", rQueue -> count);
                mpxprompt_anykey();
                if( rQueue -> count > 0 ){
                incr = rQueue -> node;
                while( incr != NULL ){

                        pointer = incr -> process;

                        lp = string_PCB(pointer);
                        mpx_pager(lp);
```

```
                    incr = incr -> right; // progress forward to the right of
    the queue
                }
            }
                printf("%d", wsQueue -> count);
                mpxprompt_anykey();
                if(wsQueue -> count > 0){
                incr = wsQueue -> node;
                while( incr != NULL ){
                        pointer = incr -> process;


                        lp = string_PCB(pointer);
                        mpx_pager(lp);

                        incr = incr -> right; // progress forward to the right of
    the queue
                }
                }
     }
     else{
                printf("Wrong number of arguments used");
                return;
     }
     mpxprompt_anykey();
}
```

4.7.1.15 void mpxcmd_showBlocked_PCB (int *argc*, char * *argv[ ]*)

This is a user function that shows all blocked processes followed by non-blocked processes.

Definition at line 699 of file mpx_r2.c.

```
                                                { // Pagination function needs
    added !!Function still needs work!!
     if(argc==1){
                ELEM *incr;
                PCB *pointer;
                char line[MAX_LINE];
                char* lp;
                char class[30];
                char state[45];
                lp = &line;
                mpx_pager_init(" All PCB's Blocked State in Queues:\n -----------
---------------------------------------\n");

                incr = wsQueue -> node;//set node to the first node in the queque

                while( incr  != NULL ){
                        pointer = incr -> process;
                        if ( pointer -> state == SUSPENDED_BLOCKED || pointer ->
state == BLOCKED ){
                        lp = string_PCB(pointer);
                        mpx_pager(lp);
                        }
                        incr = incr -> right; // progress forward to the right of
     the queue                      incr = incr -> right; // progress forward to the
right of the queque
                }
      }
      else{
                printf("Wrong number of arguments used");
                return;
      }
```

```
        mpxprompt_anykey();
}
```

4.7.1.16 void mpxcmd_showReady_PCB (int *argc*, char * *argv[ ]*)

This is a user function that shows all non-suspended processes followed by suspended processes.

Definition at line 660 of file mpx_r2.c.

```
                                                { // Pagination function needs a
      dded !!Function still needs work!!
        if(argc==1){
                ELEM *incr;
                PCB *pointer;
                char line[MAX_LINE];
                char* lp;
                char class[30];
                char state[45];
                incr = rQueue -> node;//set node to the first node in the queque
                lp = &line;
                mpx_pager_init(" All PCB's Ready State in Queues:\n -------------
      ----------------------------------------\n");
                while( incr != NULL ){

                        pointer = incr -> process;
                        if ( pointer -> state == READY){
                        lp = string_PCB(pointer);
                        mpx_pager(lp);
                        }
                        incr = incr -> right; // progress forward to the right of
       the queque
                }

                incr = wsQueue -> node;//set node to the first node in the queque

                while( incr != NULL ){
                        pointer = incr -> process;
                        if ( pointer -> state == SUSPENDED_READY){
                        lp = string_PCB(pointer);
                        mpx_pager(lp);
                        }
                        incr = incr -> right; // progress forward to the right of
       the queque                      incr = incr -> right; // progress forward to the
      right of the queque
                }
        }
        else{
                printf("Wrong number of arguments used");
                return;
        }
        mpxprompt_anykey();
}
```

4.7.1.17 void mpxcmd_suspend (int *argc*, char * *argv[ ]*)

This is a user function in the menu that puts a process in the suspend state it takes the process name as input.

Definition at line 487 of file mpx_r2.c.

```
{
```

```
        if(argc==2){
                char name[STRLEN];
                PCB *pointer;
                PCB *tempPCB;
                int buffs = STRLEN;
                strcpy(name,argv[1]);

                pointer = find_PCB(name);
                if ( pointer != NULL){
                        tempPCB = copy_PCB(pointer);
                        remove_PCB(pointer);
                        if( tempPCB -> state == READY || tempPCB -> state ==
    RUNNING ) tempPCB -> state = SUSPENDED_READY;
                        if( tempPCB -> state == BLOCKED ) tempPCB -> state = SUSP
    ENDED_BLOCKED;
                        insert_PCB(tempPCB);
                }else{
                        printf("Process Name not found!");
                        return;
                }
        }
        else{
                printf("Wrong number of arguments used");
                return;
        }
}
```

4.7.1.18 void mpxcmd_unblock (int *argc*, char * *argv[ ]*)

This is a user function in the menu that puts a process in the unblocked state it takes the process name as input.

Definition at line 459 of file mpx_r2.c.

```
                                        {
        if(argc==2){
                char name[STRLEN];
                PCB *pointer;
                PCB *tempPCB;
                int buffs = STRLEN;

                strcpy(name,argv[1]);

                pointer = find_PCB(name);
                if ( pointer != NULL){
                        tempPCB = copy_PCB(pointer);
                        remove_PCB(pointer);
                        if( tempPCB -> state == BLOCKED ) tempPCB -> state = READ
    Y;
                        if( tempPCB -> state == SUSPENDED_BLOCKED ) tempPCB -> st
    ate = SUSPENDED_READY;
                        insert_PCB(tempPCB);
                }else{
                        printf("Process Name not found!");
                        return;
                }
        }
        else{
                printf("Wrong number of arguments used");
                return;
        }
}
```

4.7.1.19 void remove_PCB (PCB * *process*)

This function removes a pcb and dealocates its resouces takes in a pointer to a PCBs location.

Definition at line 306 of file mpx_r2.c.

```
                        {
 ROOT *queue;
 ELEM *incr;
 ELEM *temp1;
 ELEM *temp2;

 if ( process -> state == READY || process -> state == RUNNING ){
         queue = rQueue;
 }
 if( process -> state == BLOCKED ||
         process -> state == SUSPENDED_READY ||
         process -> state == SUSPENDED_BLOCKED ){
         queue = wsQueue;
 }
 /* last in queue */
 if ( queue -> count == 1 ){
         incr = queue-> node;
         free_PCB(incr->process);
         sys_free_mem(queue->node);
         queue -> node = NULL;
         queue -> count -=1;

         return;
 }
 incr = queue-> node; //set node to the first node in the queque
 while ( (incr -> process != process ) && incr != NULL ){ // find the same
process
                 incr = incr -> right; // progrees to the right
 }
 /* There are three cases to check for head, tail, and middle*/



 /*head case*/
 if( incr -> left == NULL && incr->right != NULL ){
         temp1 = incr -> right;
         temp1 -> left  = NULL;
         queue -> node = temp1; //set queueROOT to new head
         queue ->count -=1;
 }

 /*tail case*/
 if( incr -> left != NULL && incr->right == NULL ){
         temp1 = incr-> left;
         temp1 -> right  = NULL;
         queue -> count -=1;
 }

 /*middle case*/
 if( incr -> left != NULL && incr->right != NULL){
         temp1 = incr -> left;
         temp1 -> right = incr -> right;
         temp2 = incr -> right;
         temp2 -> left = incr -> left;
         queue -> count -=1;
 }
 //Deallocate mem
 free_PCB(process);
 sys_free_mem(incr);

 return;
```

```
        }
```

4.7.1.20 int setup_PCB (PCB * *pointer*, char * *Name*, int *classType*, int *state*, int *priority*)

This Function initializes the contents of a PCB and checks the values if correct returns 0 if not returns 1.

Definition at line 83 of file mpx_r2.c.

```
      {//FIXME: NO DATA VV
        int ret;
        char *name = pointer -> name;
        ret = 0;
        strcpy(name,Name);

        if( find_PCB(name) == NULL){
                if( classType == 1 || classType == 0 ){
                        pointer -> classType = classType;
                }else{
                        ret = 1;
                }
                 if( state == BLOCKED ||
                         state == SUSPENDED_READY ||
                     state == SUSPENDED_BLOCKED ||
                         state == READY ||
                         state == RUNNING )
                {
                pointer -> state = state;
                }else{
                        ret = 1;
                }
                if( priority <= 127 && priority >= -128){
                        pointer -> priority = priority;
                }else{
                        ret = 1;
                }
        }else{
                ret = 1;
        }
        return ret;
}
```

4.7.1.21 char* string_PCB (PCB * *pointer*)

This function returns a character string with PCB information formatted.

Definition at line 116 of file mpx_r2.c.

```
                                {
        char line_buf[MAX_LINE];
        char *name = pointer -> name;
        signed char *classType = pointer -> classType;
        signed char *stateType = pointer -> state;
        signed char *priority = pointer -> priority;
        char class[60];
        char state[60];

        if( classType == APPLICATION ) strcpy( class, "Application");
        if( classType == SYSTEM ) strcpy( class, "System" );

        if( stateType == RUNNING ) strcpy(state,"Running");
```

```
        if( stateType == READY ) strcpy( state ,"Ready" );
        if( stateType == BLOCKED ) strcpy( state ,"Blocked");
        if( stateType == SUSPENDED_READY ) strcpy(state ,"Suspended Ready");
        if ( stateType == SUSPENDED_BLOCKED ) strcpy( state,"Suspended Blocked" )
       ;


    sprintf(&line_buf,"Name: %s  Class: %s State: %s Priority: %d ", name, class,
       state,priority);

        return line_buf;
}
```

4.7.2 Variable Documentation

4.7.2.1 ROOT* rQueue

Definition at line 9 of file mpx_r2.c.

4.7.2.2 ROOT* wsQueue

Definition at line 10 of file mpx_r2.c.

4.8 src/mpx_r2.c

```
00001 #include "mpx_r2.h"
00002 #include "mpx_supt.h"
00003 #include "mystdlib.h"
00004 #include "mpx_util.h"
00005 #include <string.h>
00006 #include <stdio.h>
00007
00008
00009 ROOT *rQueue; // Remember to reserve memory for these
00010 ROOT *wsQueue;
00011
00012
00016 PCB *allocate_PCB( void ){
00017         PCB *newPCB;
00018         int i;
00019         MEMDSC *newMemDsc;
00020         STACKDSC *newStackDsc;
00021         unsigned char *stack;
00022
00023
00024         // Allocate memory to each of the disctenct parts of the PCB
00025         newStackDsc = (STACKDSC*) sys_alloc_mem(sizeof(STACKDSC));
00026         newMemDsc = (MEMDSC*) sys_alloc_mem(sizeof(MEMDSC));
00027         newPCB = (PCB*) sys_alloc_mem(sizeof(PCB));
00028         stack = (unsigned char*) sys_alloc_mem(STACKSIZE*sizeof(unsigned char));
00029
00030         if ( stack == NULL ||
00031                  newStackDsc == NULL ||
00032                  newMemDsc == NULL ||
00033                  newPCB == NULL ) return NULL;
00034
00035         //Setup Memory Descriptor with Default Values for Module 2
00036         newMemDsc -> size = 0;
00037         newMemDsc -> loadADDR = NULL;
00038         newMemDsc -> execADDR = NULL;
```

```
00039
00040         //Setup the Stack
00041
00042         memset(stack,0,STACKSIZE*sizeof(unsigned char));//ZERO out Stack to aid i
      n debug....
00043         newStackDsc -> base = stack; // x86 arch Stacks start at the Higest value

00044         newStackDsc -> top  = stack[STACKSIZE-1];// and go to lowest or n - 2 for
       Word alloc
00045
00046         //Bundling Opereations of Stack Descripter Bellow
00047         newPCB -> stackdsc = newStackDsc;  // stack descriptor is placed in the P
      CB
00048
00049         //Bundling Operations of Memory Descriptor
00050         newPCB -> memdsc = newMemDsc; // memory descriptor is placed in the PCB
00051
00052         return newPCB;
00053
00054 };
00058 int free_PCB( PCB *pointer /*< [in] is a pointer to a PCB  */ ){
00059         STACKDSC *stackdscptr = pointer -> stackdsc;
00060         unsigned char *stack = stackdscptr -> base;
00061         MEMDSC *memptr = pointer -> memdsc;
00062
00063         int err;
00064
00065         //Free Stack First
00066         err = sys_free_mem(stack);
00067         if( err < 0 ) return err;
00068         //Second free Stack Descriptor
00069         err = sys_free_mem(stackdscptr);
00070         if( err < 0 ) return err;
00071         //Third free Memory Descriptor
00072         err = sys_free_mem(memptr);
00073         if( err < 0 ) return err;
00074         //Finaly free Process Control block
00075         err = sys_free_mem(pointer);
00076         if(err < 0 ) return err;
00077
00078         return 0; //freed mem ok
00079 }
00080
00082 //FIXME: Move to allocate, Create to setup
00083 int setup_PCB( PCB *pointer, char *Name, int classType, int state, int priority )
      {//FIXME: NO DATA VV
00084         int ret;
00085         char *name = pointer -> name;
00086         ret = 0;
00087         strcpy(name,Name);
00088
00089         if( find_PCB(name) == NULL){
00090                 if( classType == 1 || classType == 0 ){
00091                         pointer -> classType = classType;
00092                 }else{
00093                         ret = 1;
00094                 }
00095                  if( state == BLOCKED ||
00096                          state == SUSPENDED_READY ||
00097                      state == SUSPENDED_BLOCKED ||
00098                          state == READY ||
00099                          state == RUNNING )
00100                 {
00101                 pointer -> state = state;
00102                 }else{
00103                         ret = 1;
00104                 }
```

```
00105                 if( priority <= 127 && priority >= -128){
00106                         pointer -> priority = priority;
00107                 }else{
00108                         ret = 1;
00109                 }
00110         }else{
00111                 ret = 1;
00112         }
00113         return ret;
00114 }
00116 char *string_PCB( PCB *pointer){
00117         char line_buf[MAX_LINE];
00118         char *name = pointer -> name;
00119         signed char *classType = pointer -> classType;
00120         signed char *stateType = pointer -> state;
00121         signed char *priority = pointer -> priority;
00122         char class[60];
00123         char state[60];
00124
00125         if( classType == APPLICATION ) strcpy( class, "Application");
00126         if( classType == SYSTEM ) strcpy( class, "System" );
00127
00128         if( stateType == RUNNING ) strcpy(state,"Running");
00129         if( stateType == READY ) strcpy( state ,"Ready" );
00130         if( stateType == BLOCKED ) strcpy( state ,"Blocked");
00131         if( stateType == SUSPENDED_READY ) strcpy(state ,"Suspended Ready");
00132         if ( stateType == SUSPENDED_BLOCKED ) strcpy( state,"Suspended Blocked" )
      ;
00133
00134
00135     sprintf(&line_buf,"Name: %s  Class: %s State: %s Priority: %d ", name, class,
      state,priority);
00136
00137         return line_buf;
00138 }
00139
00141 void insert_PCB(PCB *PCBpointer/*< pointer to a PCB to insert*/ ){
00142    int ORD;
00143
00144    if ( PCBpointer -> state == READY || PCBpointer -> state == RUNNING ){
00145                 ORD  = PORDR;
00146         }
00147         if( PCBpointer -> state == BLOCKED ||
00148                 PCBpointer -> state == SUSPENDED_READY ||
00149                 PCBpointer -> state == SUSPENDED_BLOCKED ){
00150                 ORD  = FIFO;
00151         }
00152
00153    switch(ORD){
00154                 case PORDR:
00155                         insert_PORDR(PCBpointer,rQueue);
00156                         break;
00157                 case FIFO:
00158                         insert_FIFO(PCBpointer,wsQueue);
00159                         break;
00160                 default:
00161                         printf("ORDER not Valid");
00162                         break;
00163                 };
00164 }
00166 void insert_PORDR( PCB *PCBpointer, ROOT *queueROOT ){ //FIXME: NO ERROR CHECKING

00167         ELEM *node; // declare node of type element
00168         ELEM *incr;
00169         ELEM *temp1;
00170         char line[MAX_LINE];
00171         char *lp;
```

```
00172         lp = &line;
00173         node = sys_alloc_mem( sizeof(ELEM)); // allocate Memory for node
00174         node -> process = PCBpointer;// add the PCB to the node
00175
00176         if( queueROOT -> node == NULL ){ // if this is the first element ever in
      the queque
00177                 node -> left = NULL;
00178                 node -> right = NULL;
00179                 queueROOT -> node = node; // Set the first element in the queque
      to node of Type Element
00180                 queueROOT -> count += 1; // increase count by one
00181                 return; //exit out first node is in queque.
00182         }
00183
00184         incr = queueROOT -> node; //set node to the first node in the queque
00185         while ( incr -> process -> priority <= node -> process -> priority  ){ //
       Process with the lowest priority goes first
00186                     lp = string_PCB(incr->process);
00187                         printf("%s\n",lp);
00188                         if( incr->right == NULL) break;
00189                         incr = incr -> right; // progrees to the right
00190
00191         }
00192
00193         /* There are three cases to check for head, tail, and middle*/
00194
00195         /*head case*/
00196         if ( incr -> left == NULL && incr-> right == NULL){
00197                 if( incr -> process -> priority <= node -> process -> priority ){

00198                         node-> left = incr;
00199                         node-> right = NULL;
00200                         incr->right = node;
00201                         queueROOT->count +=1;
00202                 }else{
00203                         node->left = NULL;
00204                         node->right = incr;
00205                         incr->left = node;
00206                         queueROOT -> node = node; //set quequeROOT to new head
00207                         queueROOT ->count +=1;
00208                 }
00209                 return;
00210         }
00211         if( incr -> left == NULL && incr->right != NULL ){
00212                 node->left = NULL;
00213                 node->right = incr;
00214                 incr->left = node;
00215                 queueROOT -> node = node; //set quequeROOT to new head
00216                 queueROOT ->count +=1;
00217                 return;
00218         }
00219
00220         /*tail case*/
00221         if( incr -> left != NULL && incr->right == NULL ){
00222
00223                 if( incr -> process -> priority <= node -> process -> priority ){

00224                         node-> left = incr;
00225                         node-> right = NULL;
00226                         incr->right = node;
00227                         queueROOT->count +=1;
00228                         return;
00229                 }else{
00230                         incr = incr -> left; //decrement incr
00231                         temp1 = incr -> right;
00232                         incr->right = node;
00233                         node->right = temp1;
```

```
00234                           node->left = incr;
00235                           temp1->left = node;
00236                           queueROOT->count +=1;
00237                           return;
00238                   }
00239
00240           }
00241
00242           /*middle case*/
00243           if( incr -> left != NULL && incr->right != NULL){
00244                   incr = incr -> left;
00245                   temp1 = incr -> right;
00246                   incr->right = node;
00247                   node->right = temp1;
00248                   node->left = incr;
00249                   temp1->left = node;
00250                   queueROOT->count +=1;
00251                   return;
00252           }
00253 }
00255 void insert_FIFO( PCB *PCBpointer, ROOT *queueROOT){ //FIXME: NO ERROR HANDLING
00256           ELEM *node; // declare node of type element
00257           ELEM *incr;
00258
00259
00260           node = sys_alloc_mem( sizeof(ELEM)); // allocate Memory for node
00261           node -> process = PCBpointer;// add the PCB to the node
00262
00263           if( queueROOT -> node == NULL ){ // if this is the first element ever in
      the queque
00264                   node -> left = NULL; // set the link left to null
00265                   node -> right = NULL;// set the link right to null
00266                   queueROOT -> node = node; // Set the first element in the queque
      to node of Type Element
00267                   queueROOT -> count += 1; // increase count by one
00268                   return; //exit out first node is in queque.
00269           }
00270
00271
00272           /* INSERT INTO THE QUEQUE IN FIFO ORDER*/
00273           incr = queueROOT -> node; //set node to the first node in the queque
00274           while( incr -> right != NULL ){
00275                   incr = incr -> right; // progress forward to the right of the que
      que
00276           }
00277            incr -> right = node;
00278            node -> left = incr; //set left to previous node
00279            node -> right = NULL; // set right to null
00280            queueROOT -> count += 1; // increase count by one as the size of the que
      que has grown by one
00281
00282            return;
00283
00284 }
00286 PCB *find_PCB( char *name){
00287           ELEM *incr;
00288           incr =  rQueue -> node; //set node to the first node in the queque
00289           while ( strcmp(name,incr -> process -> name ) != 0 && incr != NULL){ // P
      rocess with the lowest priority goes first
00290                           incr= incr -> right; // progrees to the right
00291           }
00292           if (incr == NULL ){
00293           incr =  wsQueue -> node; //set node to the first node in the queque
00294           while ( strcmp(name,incr -> process -> name ) != 0 && incr != NULL){ // P
      rocess with the lowest priority goes first
00295                           incr= incr -> right; // progrees to the right
00296           }
```

```
00297         }
00298         if ( incr -> process != NULL && incr != NULL ){
00299                 return incr->process;
00300         }else{
00301                 return NULL;
00302         }
00303
00304 }
00306 void remove_PCB( PCB *process ){
00307         ROOT *queue;
00308         ELEM *incr;
00309         ELEM *temp1;
00310         ELEM *temp2;
00311
00312         if ( process -> state == READY || process -> state == RUNNING ){
00313                 queue = rQueue;
00314         }
00315         if( process -> state == BLOCKED ||
00316                 process -> state == SUSPENDED_READY ||
00317                 process -> state == SUSPENDED_BLOCKED ){
00318                 queue = wsQueue;
00319         }
00320         /* last in queue */
00321         if ( queue -> count == 1 ){
00322                 incr = queue-> node;
00323                 free_PCB(incr->process);
00324                 sys_free_mem(queue->node);
00325                 queue -> node = NULL;
00326                 queue -> count -=1;
00327
00328                 return;
00329         }
00330         incr = queue-> node; //set node to the first node in the queque
00331         while ( (incr -> process != process ) && incr != NULL ){ // find the same
      process
00332                         incr = incr -> right; // progrees to the right
00333         }
00334         /* There are three cases to check for head, tail, and middle*/
00335
00336
00337
00338         /*head case*/
00339         if( incr -> left == NULL && incr->right != NULL ){
00340                 temp1 = incr -> right;
00341                 temp1 -> left  = NULL;
00342                 queue -> node = temp1; //set queueROOT to new head
00343                 queue ->count -=1;
00344         }
00345
00346         /*tail case*/
00347         if( incr -> left != NULL && incr->right == NULL ){
00348                 temp1 = incr-> left;
00349                 temp1 -> right  = NULL;
00350                 queue -> count -=1;
00351         }
00352
00353         /*middle case*/
00354         if( incr -> left != NULL && incr->right != NULL){
00355                 temp1 = incr -> left;
00356                 temp1 -> right = incr -> right;
00357                 temp2 = incr -> right;
00358                 temp2 -> left = incr -> left;
00359                 queue -> count -=1;
00360         }
00361         //Deallocate mem
00362         free_PCB(process);
00363         sys_free_mem(incr);
```

```
00364
00365         return;
00366         }
00367
00369 void mpxcmd_create_PCB(int argc, char *argv[]){
00370         static int count;
00371         char name[STRLEN];
00372         char line[MAX_LINE];
00373         int type;
00374         int priority;
00375
00376         PCB *newPCB = allocate_PCB();
00377
00378         if( count == ZERO ){
00379                 rQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
00380                 wsQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
00381         }
00382
00383
00384         printf("Process Name: \n");
00385         mpx_readline(name, STRLEN);
00386         printf("Process Class Type ( Application 0 or System  1): \n" );
00387         type= mpxprompt_int();
00388         printf("Process Priority (-128 to 127): \n");
00389         priority = mpxprompt_int();
00390
00391
00392
00393         if ( setup_PCB(newPCB,&name,type,READY,priority) == 1){
00394                 printf("Incrorrect information entered.");
00395                 mpxprompt_anykey();
00396                 return;
00397         }
00398
00399         insert_PCB(newPCB);
00400         count++;//Update the number of times the function has run.
00401 }
00402
00404 PCB *copy_PCB(PCB *pointer){
00405                 PCB *tempPCB = allocate_PCB();
00406                 memcpy(tempPCB,pointer,sizeof(PCB));
00407                 memcpy(tempPCB -> memdsc, pointer -> memdsc , sizeof(MEMDSC));
00408                 memcpy(tempPCB -> stackdsc ,pointer -> stackdsc, sizeof(STACKDSC)
      );
00409         return tempPCB;
00410 }
00412 void mpxcmd_delete_PCB(int argc, char *argv[]){
00413         if (argc == 2){
00414         char name[STRLEN];
00415         PCB *pointer;
00416         strcpy(name,argv[1]);
00417
00418         pointer = find_PCB(name);
00419
00420         if ( pointer != NULL){
00421                 remove_PCB(pointer);
00422         }else{
00423                 printf("Process Name not found!");
00424                 return;
00425         }
00426         }
00427 }
00428
00430 void mpxcmd_block(int argc, char *argv[]){
00431         if(argc==2){
00432                 char name[STRLEN];
00433
```

```
00434                 PCB *pointer;
00435                 PCB *tempPCB;
00436                 int buffs = STRLEN;
00437 
00438                 strcpy(name,argv[1]);
00439 
00440                 pointer = find_PCB(name);
00441                 if ( pointer != NULL){
00442                         tempPCB = copy_PCB(pointer);
00443                         remove_PCB(pointer);
00444                         if( tempPCB -> state == READY || tempPCB -> state ==
      RUNNING ) tempPCB -> state = BLOCKED;
00445                         if( tempPCB -> state == SUSPENDED_READY ) tempPCB -> stat
      e = SUSPENDED_BLOCKED;
00446                         insert_PCB(tempPCB);
00447                 }else{
00448                         printf("Process Name not found!");
00449                         return;
00450                 }
00451         }
00452         else{
00453                 printf("Wrong number of arguments used");
00454                 return;
00455         }
00456 }
00457 
00459 void mpxcmd_unblock(int argc, char *argv[]){
00460         if(argc==2){
00461                 char name[STRLEN];
00462                 PCB *pointer;
00463                 PCB *tempPCB;
00464                 int buffs = STRLEN;
00465 
00466                 strcpy(name,argv[1]);
00467 
00468                 pointer = find_PCB(name);
00469                 if ( pointer != NULL){
00470                         tempPCB = copy_PCB(pointer);
00471                         remove_PCB(pointer);
00472                         if( tempPCB -> state == BLOCKED ) tempPCB -> state = READ
      Y;
00473                         if( tempPCB -> state == SUSPENDED_BLOCKED ) tempPCB -> st
      ate = SUSPENDED_READY;
00474                         insert_PCB(tempPCB);
00475                 }else{
00476                         printf("Process Name not found!");
00477                         return;
00478                 }
00479         }
00480         else{
00481                 printf("Wrong number of arguments used");
00482                 return;
00483         }
00484 }
00485 
00487 void mpxcmd_suspend(int argc, char *argv[]){
00488         if(argc==2){
00489                 char name[STRLEN];
00490                 PCB *pointer;
00491                 PCB *tempPCB;
00492                 int buffs = STRLEN;
00493                 strcpy(name,argv[1]);
00494 
00495                 pointer = find_PCB(name);
00496                 if ( pointer != NULL){
00497                         tempPCB = copy_PCB(pointer);
00498                         remove_PCB(pointer);
```

```
00499                           if( tempPCB -> state == READY || tempPCB -> state ==
      RUNNING ) tempPCB -> state = SUSPENDED_READY;
00500                           if( tempPCB -> state == BLOCKED ) tempPCB -> state = SUSP
      ENDED_BLOCKED;
00501                           insert_PCB(tempPCB);
00502                   }else{
00503                           printf("Process Name not found!");
00504                           return;
00505                   }
00506           }
00507           else{
00508                   printf("Wrong number of arguments used");
00509                   return;
00510           }
00511 }
00512
00514 void mpxcmd_resume(int argc, char *argv[]){
00515           if(argc==2){
00516                   char name[STRLEN];
00517                   PCB *pointer;
00518                   PCB *tempPCB;
00519                   int buffs = STRLEN;
00520
00521                   strcpy(name,argv[1]);
00522
00523                   pointer = find_PCB(name);
00524                   if ( pointer != NULL){
00525                           tempPCB = copy_PCB(pointer);
00526                           remove_PCB(pointer);
00527                           if( tempPCB -> state == SUSPENDED_READY ) tempPCB -> stat
      e = READY;
00528                           if( tempPCB -> state == SUSPENDED_BLOCKED ) tempPCB -> st
      ate = BLOCKED;
00529                           insert_PCB(tempPCB);
00530                   }else{
00531                           printf("Process Name not found!");
00532                           return;
00533                   }
00534           }
00535           else{
00536                   printf("Wrong number of arguments used");
00537                   return;
00538           }
00539 }
00540
00542 void mpxcmd_setPriority(int argc, char *argv[]){
00543           if(argc==3){
00544                   char name[STRLEN];
00545                   PCB *pointer;
00546                   int priority;
00547                   PCB *tempPCB;
00548                   int buffs = STRLEN;
00549                   priority  = atoi(argv[2]);
00550                   strcpy(name,argv[1]);
00551                   if( priority <= 128 || priority >= -127){ ;}else{
00552                           printf("Number entered out of range!");
00553                           mpxprompt_anykey();
00554                           return;
00555                   }
00556                   pointer = find_PCB(name);
00557                   if ( pointer != NULL){
00558                           pointer -> priority = priority;
00559                           if( pointer -> state == READY ){
00560                                           tempPCB = copy_PCB(pointer);
00561                                           remove_PCB(pointer);
00562                                           insert_PCB(tempPCB);
00563                                           }
```

```
00564                 }else{
00565                         printf("Process Name not found!");
00566                         mpxprompt_anykey();
00567                         return;
00568                 }
00569         }
00570         else{
00571                 printf("Wrong number of arguments used");
00572                 mpxprompt_anykey();
00573                 return;
00574         }
00575 }
00576
00577
00579 void mpxcmd_show_PCB(int argc, char *argv[]){
00580         if(argc==2){
00581                 char name[STRLEN];
00582                 PCB *pointer;
00583                 char class[30];
00584                 char state[45];
00585                 int buffs = STRLEN;
00586                 char line[MAX_LINE];
00587                 char* lp;
00588                 lp = &line;
00589
00590                 strcpy(name,argv[1]);
00591
00592                 pointer = find_PCB(name);
00593
00594                 if ( pointer != NULL){
00595                         printf("%s\n",string_PCB(pointer));
00596                         mpxprompt_anykey();
00597                 }else{
00598                         printf("Process Name not found!");
00599                         mpxprompt_anykey();
00600                         return;
00601                 }
00602         }
00603         else{
00604                 printf("Wrong number of arguments used");
00605                 mpxprompt_anykey();
00606                 return;
00607         }
00608 }
00609
00611 void mpxcmd_showAll_PCB(int argc, char *argv[]){ // Pagination function needs add
      ed !!Function still needs work!!
00612         if(argc==1){
00613                 ELEM *incr;
00614                 PCB *pointer;
00615                 char line[MAX_LINE];
00616                 char* lp;
00617                 char class[30];
00618                 char state[45];
00619                 //set node to the first node in the queque
00620                 lp = &line;
00621                 mpx_pager_init(" All PCB's In Queue:\n -------------------------
      ---------------------------\n");
00622
00623                 printf("%d", rQueue -> count);
00624                 mpxprompt_anykey();
00625                 if( rQueue -> count > 0 ){
00626                 incr = rQueue -> node;
00627                 while( incr != NULL ){
00628
00629                         pointer = incr -> process;
00630
```

```
00631                           lp = string_PCB(pointer);
00632                           mpx_pager(lp);
00633
00634                           incr = incr -> right; // progress forward to the right of
      the queque
00635                   }
00636               }
00637                   printf("%d", wsQueue -> count);
00638                   mpxprompt_anykey();
00639                   if(wsQueue -> count > 0){
00640                   incr = wsQueue -> node;
00641                   while( incr != NULL ){
00642                           pointer = incr -> process;
00643
00644
00645                           lp = string_PCB(pointer);
00646                           mpx_pager(lp);
00647
00648                           incr = incr -> right; // progress forward to the right of
      the queque
00649                   }
00650                   }
00651           }
00652           else{
00653                   printf("Wrong number of arguments used");
00654                   return;
00655           }
00656           mpxprompt_anykey();
00657 }
00658
00660 void mpxcmd_showReady_PCB(int argc, char *argv[]){ // Pagination function needs a
      dded !!Function still needs work!!
00661           if(argc==1){
00662                   ELEM *incr;
00663                   PCB *pointer;
00664                   char line[MAX_LINE];
00665                   char* lp;
00666                   char class[30];
00667                   char state[45];
00668                   incr = rQueue -> node;//set node to the first node in the queque
00669                   lp = &line;
00670                   mpx_pager_init(" All PCB's Ready State in Queues:\n -------------
      ----------------------------------------\n");
00671                   while( incr != NULL ){
00672
00673                           pointer = incr -> process;
00674                           if ( pointer -> state == READY){
00675                           lp = string_PCB(pointer);
00676                           mpx_pager(lp);
00677                           }
00678                           incr = incr -> right; // progress forward to the right of
      the queque
00679                   }
00680
00681                   incr = wsQueue -> node;//set node to the first node in the queque

00682                   while( incr != NULL ){
00683                           pointer = incr -> process;
00684                           if ( pointer -> state == SUSPENDED_READY){
00685                           lp = string_PCB(pointer);
00686                           mpx_pager(lp);
00687                           }
00688                           incr = incr -> right; // progress forward to the right of
      the queque                      incr = incr -> right; // progress forward to the
      right of the queque
00689                   }
00690           }
```

```
00691         else{
00692                 printf("Wrong number of arguments used");
00693                 return;
00694         }
00695         mpxprompt_anykey();
00696 }
00697
00699 void mpxcmd_showBlocked_PCB(int argc, char *argv[]){ // Pagination function needs
       added !!Function still needs work!!
00700         if(argc==1){
00701                 ELEM *incr;
00702                 PCB *pointer;
00703                 char line[MAX_LINE];
00704                 char* lp;
00705                 char class[30];
00706                 char state[45];
00707                 lp = &line;
00708                 mpx_pager_init(" All PCB's Blocked State in Queues:\n ----------
      ----------------------------------------\n");
00709
00710                 incr = wsQueue -> node;//set node to the first node in the queque

00711                 while( incr  != NULL ){
00712                         pointer = incr -> process;
00713                         if ( pointer -> state == SUSPENDED_BLOCKED || pointer ->
      state == BLOCKED ){
00714                         lp = string_PCB(pointer);
00715                         mpx_pager(lp);
00716                         }
00717                         incr = incr -> right; // progress forward to the right of
       the queque                       incr = incr -> right; // progress forward to the
      right of the queque
00718                 }
00719         }
00720         else{
00721                 printf("Wrong number of arguments used");
00722                 return;
00723         }
00724         mpxprompt_anykey();
00725 }
00726
00727
```

4.9 src/mpx_r2.h File Reference

Data Structures

- struct mem
- struct stack
- struct process
- struct page
- struct root

Defines

- #define RUNNING 0

  state is Defined as 0

- #define READY 1

state is Defined as 1

- #define BLOCKED 2
  state is defined as 2
- #define SUSPENDED_READY 3
  is defined by 3
- #define SUSPENDED_BLOCKED 4
  is defined by 4
- #define SYSTEM 1
  is defined as 1
- #define APPLICATION 0
  is defined as 0
- #define STACKSIZE 1024
  is the size of the stack in Bytes
- #define STRLEN 16
  is the length of a string for name
- #define PORDR 1
  is the Priority Order flag
- #define FIFO 0
  is the First In First Out Order flag
- #define ZERO 0
- #define MAX_LINE 1024

Typedefs

- typedef struct mem MEMDSC
- typedef struct stack STACKDSC
- typedef struct process PCB
- typedef struct page ELEM
- typedef struct root ROOT

Functions

- PCB * alloocate_PCB (void)
- int free_PCB (PCB *pointer)
  This function releases all allocated memory related to a PCB.
- int setup_PCB (PCB *pointer, char *name, int classType, int state, int priority)
  This Function initializes the contents of a PCB and checks the values if correct returns 0 if not returns 1.

- void insert_PCB (PCB *PCBpointer)

  This function inserts a PCB into its aproprate PCB Queue.

- void insert_PORDR (PCB *PCBpointer, ROOT *quequeROOT)

  This function inserts into a queue a element sorted by its priority lower number (higher priority) to high number(lower priority).

- void insert_FIFO (PCB *PCBpointer, ROOT *quequeROOT)

  In this function we grow the queque to the right no matter of the Priority of the PCB.

- PCB * find_PCB (char *name)

  This function findes a PCB by its identifier (name) and returns a pointer to its structures location.

- void mpxcmd_create_PCB (int argc, char *argv[ ])

  This is a user function that interacts with the user to create a PCB structure.

- void mpxcmd_delete_PCB (int argc, char *argv[ ])

  This is a user function in the menu to delete a process it takes the process name as input.

- void mpxcmd_block (int argc, char *argv[ ])

  This is a user function in the menu that puts a process in the blocked state it takes the process name as input.

- void mpxcmd_unblock (int argc, char *argv[ ])

  This is a user function in the menu that puts a process in the unblocked state it takes the process name as input.

- void mpxcmd_suspend (int argc, char *argv[ ])

  This is a user function in the menu that puts a process in the suspend state it takes the process name as input.

- void mpxcmd_resume (int argc, char *argv[ ])

  This is a user function in the menu that puts a process in the ready state if previously blocked and blocked if previously suspended it takes the process name as input.

- void mpxcmd_setPriority (int argc, char *argv[ ])

  This is a user function from the menu it changes the priority of a PCB and takes the name and desired priority as inputs80ij.

- void mpxcmd_show_PCB (int argc, char *argv[ ])

  This is a user command from the menu it is used to show information about a specific PCB.

- void mpxcmd_showAll_PCB (int argc, char *argv[ ])

  This is a user functions that shows name and state of all processes.

- void mpxcmd_showReady_PCB (int argc, char *argv[ ])

  This is a user function that shows all non-suspended processes followed by suspended processes.

- void mpxcmd_showBlocked_PCB (int argc, char *argv[ ])

  This is a user function that shows all blocked processes followed by non-blocked processes.

4.9.1 Define Documentation

4.9.1.1 #define APPLICATION 0

is defined as 0

Definition at line 12 of file mpx_r2.h.

4.9.1.2 #define BLOCKED 2

state is defined as 2

Definition at line 6 of file mpx_r2.h.

4.9.1.3 #define FIFO 0

is the First In First Out Order flag

Definition at line 18 of file mpx_r2.h.

4.9.1.4 #define MAX_LINE 1024

Definition at line 21 of file mpx_r2.h.

4.9.1.5 #define PORDR 1

is the Priority Order flag

Definition at line 17 of file mpx_r2.h.

4.9.1.6 #define READY 1

state is Defined as 1

Definition at line 5 of file mpx_r2.h.

4.9.1.7 #define RUNNING 0

state is Defined as 0

Definition at line 4 of file mpx_r2.h.

4.9.1.8 #define STACKSIZE 1024

is the size of the stack in Bytes

Definition at line 14 of file mpx_r2.h.

4.9.1.9 #define STRLEN 16

is the length of a string for name

Definition at line 15 of file mpx_r2.h.

4.9.1.10 #define SUSPENDED_BLOCKED 4

is defined by 4

Definition at line 9 of file mpx_r2.h.

4.9.1.11 #define SUSPENDED_READY 3

is defined by 3

Definition at line 8 of file mpx_r2.h.

4.9.1.12 #define SYSTEM 1

is defined as 1

Definition at line 11 of file mpx_r2.h.

4.9.1.13 #define ZERO 0

Definition at line 19 of file mpx_r2.h.

4.9.2 Typedef Documentation

4.9.2.1 typedef struct page ELEM

4.9.2.2 typedef struct mem MEMDSC

4.9.2.3 typedef struct process PCB

4.9.2.4 typedef struct root ROOT

4.9.2.5 typedef struct stack STACKDSC

4.9.3 Function Documentation

4.9.3.1 PCB* alloocate_PCB (void)

4.9.3.2 PCB* find_PCB (char * *name*)

This function findes a PCB by its identifier (name) and returns a pointer to its structures location.

Definition at line 286 of file mpx_r2.c.

```
                    {
  ELEM *incr;
  incr =  rQueue -> node; //set node to the first node in the queque
  while ( strcmp(name,incr -> process -> name ) != 0 && incr != NULL){ // P
rocess with the lowest priority goes first
                incr= incr -> right; // progrees to the right
  }
```

```
        if (incr == NULL ){
        incr =  wsQueue -> node; //set node to the first node in the queque
        while ( strcmp(name,incr -> process -> name ) != 0 && incr != NULL){ // P
      rocess with the lowest priority goes first
                        incr= incr -> right; // progrees to the right
        }
        }
        if ( incr -> process != NULL && incr != NULL ){
                return incr->process;
        }else{
                return NULL;
        }

}
```

4.9.3.3 int free_PCB (PCB * *pointer*)

This function releases all allocated memory related to a PCB.

< is a pointer to the stack descriptor

< is a pointer to the base location of the stack

< is a pointer to a Memory Descriptor

< holder for error capture on use of sys_free_mem

Definition at line 58 of file mpx_r2.c.

```
                                                                {
        STACKDSC *stackdscptr = pointer -> stackdsc;
        unsigned char *stack = stackdscptr -> base;
        MEMDSC *memptr = pointer -> memdsc;

        int err;

        //Free Stack First
        err = sys_free_mem(stack);
        if( err < 0 ) return err;
        //Second free Stack Descriptor
        err = sys_free_mem(stackdscptr);
        if( err < 0 ) return err;
        //Third free Memory Descriptor
        err = sys_free_mem(memptr);
        if( err < 0 ) return err;
        //Finaly free Process Control block
        err = sys_free_mem(pointer);
        if(err < 0 ) return err;

        return 0; //freed mem ok
}
```

4.9.3.4 void insert_FIFO (PCB * *PCBpointer*, ROOT * *queueROOT*)

In this function we grow the queque to the right no matter of the Priority of the PCB.

Definition at line 255 of file mpx_r2.c.

```
                                                { //FIXME: NO ERROR HANDLING
        ELEM *node; // declare node of type element
        ELEM *incr;
```

```
        node = sys_alloc_mem( sizeof(ELEM)); // allocate Memory for node
        node -> process = PCBpointer;// add the PCB to the node

        if( queueROOT -> node == NULL ){ // if this is the first element ever in
      the queque
                node -> left = NULL; // set the link left to null
                node -> right = NULL;// set the link right to null
                queueROOT -> node = node; // Set the first element in the queque
      to node of Type Element
                queueROOT -> count += 1; // increase count by one
                return; //exit out first node is in queque.
        }


        /* INSERT INTO THE QUEQUE IN FIFO ORDER*/
        incr = queueROOT -> node; //set node to the first node in the queque
        while( incr -> right != NULL ){
                incr = incr -> right; // progress forward to the right of the que
      que
        }
         incr -> right = node;
         node -> left = incr; //set left to previous node
         node -> right = NULL; // set right to null
         queueROOT -> count += 1; // increase count by one as the size of the que
      que has grown by one

         return;

}
```

4.9.3.5 void insert_PCB (PCB * *PCBpointer*)

This function inserts a PCB into its aproprate PCB Queue.

Definition at line 141 of file mpx_r2.c.

```
                                                                        {
    int ORD;

    if ( PCBpointer -> state == READY || PCBpointer -> state == RUNNING ){
                ORD  = PORDR;
        }
        if( PCBpointer -> state == BLOCKED ||
                PCBpointer -> state == SUSPENDED_READY ||
                PCBpointer -> state == SUSPENDED_BLOCKED ){
                ORD  = FIFO;
        }

    switch(ORD){
                case PORDR:
                        insert_PORDR(PCBpointer,rQueue);
                        break;
                case FIFO:
                        insert_FIFO(PCBpointer,wsQueue);
                        break;
                default:
                        printf("ORDER not Valid");
                        break;
                };
}
```

4.9.3.6 void insert_PORDR (PCB * *PCBpointer*, ROOT * *queueROOT*)

This function inserts into a queue a element sorted by its priority lower number (higher priority) to high number(lower priority).

Definition at line 166 of file mpx_r2.c.

```
                                                { //FIXME: NO ERROR CHECKING

  ELEM *node; // declare node of type element
  ELEM *incr;
  ELEM *temp1;
  char line[MAX_LINE];
  char *lp;
  lp = &line;
  node = sys_alloc_mem( sizeof(ELEM)); // allocate Memory for node
  node -> process = PCBpointer;// add the PCB to the node

  if( queueROOT -> node == NULL ){ // if this is the first element ever in
the queque
          node -> left = NULL;
          node -> right = NULL;
          queueROOT -> node = node; // Set the first element in the queque
to node of Type Element
          queueROOT -> count += 1; // increase count by one
          return; //exit out first node is in queque.
  }

  incr = queueROOT -> node; //set node to the first node in the queque
  while ( incr -> process -> priority <= node -> process -> priority  ){ //
 Process with the lowest priority goes first
              lp = string_PCB(incr->process);
                  printf("%s\n",lp);
                  if( incr->right == NULL) break;
                  incr = incr -> right; // progrees to the right

  }

  /* There are three cases to check for head, tail, and middle*/

  /*head case*/
  if ( incr -> left == NULL && incr-> right == NULL){
          if( incr -> process -> priority <= node -> process -> priority ){

                  node-> left = incr;
                  node-> right = NULL;
                  incr->right = node;
                  queueROOT->count +=1;
          }else{
                  node->left = NULL;
                  node->right = incr;
                  incr->left = node;
                  queueROOT -> node = node; //set quequeROOT to new head
                  queueROOT ->count +=1;
          }
          return;
  }
  if( incr -> left == NULL && incr->right != NULL ){
          node->left = NULL;
          node->right = incr;
          incr->left = node;
          queueROOT -> node = node; //set quequeROOT to new head
          queueROOT ->count +=1;
          return;
  }

  /*tail case*/
```

```
        if( incr -> left != NULL && incr->right == NULL ){

                if( incr -> process -> priority <= node -> process -> priority ){

                        node-> left = incr;
                        node-> right = NULL;
                        incr->right = node;
                        queueROOT->count +=1;
                        return;
                }else{
                        incr = incr -> left; //decrement incr
                        temp1 = incr -> right;
                        incr->right = node;
                        node->right = temp1;
                        node->left = incr;
                        temp1->left = node;
                        queueROOT->count +=1;
                        return;
                }

        }

        /*middle case*/
        if( incr -> left != NULL && incr->right != NULL){
                incr = incr -> left;
                temp1 = incr -> right;
                incr->right = node;
                node->right = temp1;
                node->left = incr;
                temp1->left = node;
                queueROOT->count +=1;
                return;
        }
}
```

4.9.3.7 void mpxcmd_block (int *argc*, char * *argv[ ]*)

This is a user function in the menu that puts a process in the blocked state it takes the process name as input.

Definition at line 430 of file mpx_r2.c.

```
                                        {
        if(argc==2){
                char name[STRLEN];

                PCB *pointer;
                PCB *tempPCB;
                int buffs = STRLEN;

                strcpy(name,argv[1]);

                pointer = find_PCB(name);
                if ( pointer != NULL){
                        tempPCB = copy_PCB(pointer);
                        remove_PCB(pointer);
                        if( tempPCB -> state == READY || tempPCB -> state ==
    RUNNING ) tempPCB -> state = BLOCKED;
                        if( tempPCB -> state == SUSPENDED_READY ) tempPCB -> stat
    e = SUSPENDED_BLOCKED;
                        insert_PCB(tempPCB);
                }else{
                        printf("Process Name not found!");
                        return;
```

```
                }
        }
        else{
                printf("Wrong number of arguments used");
                return;
        }
}
```

4.9.3.8 void mpxcmd_create_PCB (int *argc*, char * *argv[ ]*)

This is a user function that interacts with the user to create a PCB structure.

Definition at line 369 of file mpx_r2.c.

```
                                                {
        static int count;
        char name[STRLEN];
        char line[MAX_LINE];
        int type;
        int priority;

        PCB *newPCB = allocate_PCB();

        if( count == ZERO ){
                rQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
                wsQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
        }


        printf("Process Name: \n");
        mpx_readline(name, STRLEN);
        printf("Process Class Type ( Application 0 or System  1): \n" );
        type= mpxprompt_int();
        printf("Process Priority (-128 to 127): \n");
        priority = mpxprompt_int();



        if ( setup_PCB(newPCB,&name,type,READY,priority) == 1){
                printf("Incrorrect information entered.");
                mpxprompt_anykey();
                return;
        }

        insert_PCB(newPCB);
        count++;//Update the number of times the function has run.
}
```

4.9.3.9 void mpxcmd_delete_PCB (int *argc*, char * *argv[ ]*)

This is a user function in the menu to delete a process it takes the process name as input.

Definition at line 412 of file mpx_r2.c.

```
                                                {
        if (argc == 2){
        char name[STRLEN];
        PCB *pointer;
        strcpy(name,argv[1]);

        pointer = find_PCB(name);
```

```
        if ( pointer != NULL){
                remove_PCB(pointer);
        }else{
                printf("Process Name not found!");
                return;
        }
        }
}
```

4.9.3.10 void mpxcmd_resume (int *argc*, char * *argv[ ]*)

This is a user function in the menu that puts a process in the ready state if previously blocked and blocked if previously suspended it takes the process name as input.

Definition at line 514 of file mpx_r2.c.

```
                                        {
        if(argc==2){
                char name[STRLEN];
                PCB *pointer;
                PCB *tempPCB;
                int buffs = STRLEN;

                strcpy(name,argv[1]);

                pointer = find_PCB(name);
                if ( pointer != NULL){
                        tempPCB = copy_PCB(pointer);
                        remove_PCB(pointer);
                        if( tempPCB -> state == SUSPENDED_READY ) tempPCB -> stat
      e = READY;
                        if( tempPCB -> state == SUSPENDED_BLOCKED ) tempPCB -> st
      ate = BLOCKED;
                        insert_PCB(tempPCB);
                }else{
                        printf("Process Name not found!");
                        return;
                }
        }
        else{
                printf("Wrong number of arguments used");
                return;
        }
}
```

4.9.3.11 void mpxcmd_setPriority (int *argc*, char * *argv[ ]*)

This is a user function from the menu it changes the priority of a PCB and takes the name and desired priority as inputs80ij.

Definition at line 542 of file mpx_r2.c.

```
                                                {
        if(argc==3){
                char name[STRLEN];
                PCB *pointer;
                int priority;
                PCB *tempPCB;
                int buffs = STRLEN;
```

```
                priority  = atoi(argv[2]);
                strcpy(name,argv[1]);
                if( priority <= 128 || priority >= -127){ ;}else{
                        printf("Number entered out of range!");
                        mpxprompt_anykey();
                        return;
                }
                pointer = find_PCB(name);
                if ( pointer != NULL){
                        pointer -> priority = priority;
                        if( pointer -> state == READY ){
                                        tempPCB = copy_PCB(pointer);
                                        remove_PCB(pointer);
                                        insert_PCB(tempPCB);
                                        }
                }else{
                        printf("Process Name not found!");
                        mpxprompt_anykey();
                        return;
                }
        }
        else{
                printf("Wrong number of arguments used");
                mpxprompt_anykey();
                return;
        }
}
```

4.9.3.12 void mpxcmd_show_PCB (int *argc*, char * *argv[ ]*)

This is a user command from the menu it is used to show information about a specific PCB.

Definition at line 579 of file mpx_r2.c.

```
                                                {
        if(argc==2){
                char name[STRLEN];
                PCB *pointer;
                char class[30];
                char state[45];
                int buffs = STRLEN;
                char line[MAX_LINE];
                char* lp;
                lp = &line;

                strcpy(name,argv[1]);

                pointer = find_PCB(name);

                if ( pointer != NULL){
                        printf("%s\n",string_PCB(pointer));
                        mpxprompt_anykey();
                }else{
                        printf("Process Name not found!");
                        mpxprompt_anykey();
                        return;
                }
        }
        else{
                printf("Wrong number of arguments used");
                mpxprompt_anykey();
                return;
        }
}
```

4.9.3.13 void mpxcmd_showAll_PCB (int *argc*, char * *argv[ ]*)

This is a user functions that shows name and state of all processes.

Definition at line 611 of file mpx_r2.c.

```
                                                { // Pagination function needs add
      ed !!Function still needs work!!
        if(argc==1){
                ELEM *incr;
                PCB *pointer;
                char line[MAX_LINE];
                char* lp;
                char class[30];
                char state[45];
                //set node to the first node in the queque
                lp = &line;
                mpx_pager_init(" All PCB's In Queue:\n -------------------------
      ---------------------------\n");

                printf("%d", rQueue -> count);
                mpxprompt_anykey();
                if( rQueue -> count > 0 ){
                incr = rQueue -> node;
                while( incr != NULL ){

                        pointer = incr -> process;

                        lp = string_PCB(pointer);
                        mpx_pager(lp);

                        incr = incr -> right; // progress forward to the right of
       the queque
                }
             }
                printf("%d", wsQueue -> count);
                mpxprompt_anykey();
                if(wsQueue -> count > 0){
                incr = wsQueue -> node;
                while( incr != NULL ){
                        pointer = incr -> process;


                        lp = string_PCB(pointer);
                        mpx_pager(lp);

                        incr = incr -> right; // progress forward to the right of
       the queque
                }
                }
         }
         else{
                printf("Wrong number of arguments used");
                return;
         }
         mpxprompt_anykey();
}
```

4.9.3.14 void mpxcmd_showBlocked_PCB (int *argc*, char * *argv[ ]*)

This is a user function that shows all blocked processes followed by non-blocked processes.

Definition at line 699 of file mpx_r2.c.

```
                                                { // Pagination function needs
       added !!Function still needs work!!
        if(argc==1){
                ELEM *incr;
                PCB *pointer;
                char line[MAX_LINE];
                char* lp;
                char class[30];
                char state[45];
                lp = &line;
                mpx_pager_init(" All PCB's Blocked State in Queues:\n -----------
      ----------------------------------------\n");

                incr = wsQueue -> node;//set node to the first node in the queque

                while( incr  != NULL ){
                        pointer = incr -> process;
                        if ( pointer -> state == SUSPENDED_BLOCKED || pointer ->
      state == BLOCKED ){
                        lp = string_PCB(pointer);
                        mpx_pager(lp);
                        }
                        incr = incr -> right; // progress forward to the right of
       the queque                       incr = incr -> right; // progress forward to the
      right of the queque
                }
          }
          else{
                printf("Wrong number of arguments used");
                return;
          }
          mpxprompt_anykey();
}
```

4.9.3.15 void mpxcmd_showReady_PCB (int *argc*, char * *argv[ ]*)

This is a user function that shows all non-suspended processes followed by suspended processes.

Definition at line 660 of file mpx_r2.c.

```
                                                { // Pagination function needs a
      dded !!Function still needs work!!
          if(argc==1){
                ELEM *incr;
                PCB *pointer;
                char line[MAX_LINE];
                char* lp;
                char class[30];
                char state[45];
                incr = rQueue -> node;//set node to the first node in the queque
                lp = &line;
                mpx_pager_init(" All PCB's Ready State in Queues:\n -------------
      ----------------------------------------\n");
                while( incr != NULL ){

                        pointer = incr -> process;
                        if ( pointer -> state == READY){
                        lp = string_PCB(pointer);
                        mpx_pager(lp);
                        }
                        incr = incr -> right; // progress forward to the right of
       the queque
                }
```

```
                incr = wsQueue -> node;//set node to the first node in the queque

                while( incr != NULL ){
                        pointer = incr -> process;
                        if ( pointer -> state == SUSPENDED_READY){
                        lp = string_PCB(pointer);
                        mpx_pager(lp);
                        }
                        incr = incr -> right; // progress forward to the right of
       the queque                       incr = incr -> right; // progress forward to the
      right of the queque
                }
          }
          else{
                printf("Wrong number of arguments used");
                return;
          }
          mpxprompt_anykey();
}
```

4.9.3.16 void mpxcmd_suspend (int *argc*, char * *argv[ ]*)

This is a user function in the menu that puts a process in the suspend state it takes the process name as input.

Definition at line 487 of file mpx_r2.c.

```
                                                {
          if(argc==2){
                char name[STRLEN];
                PCB *pointer;
                PCB *tempPCB;
                int buffs = STRLEN;
                strcpy(name,argv[1]);

                pointer = find_PCB(name);
                if ( pointer != NULL){
                        tempPCB = copy_PCB(pointer);
                        remove_PCB(pointer);
                        if( tempPCB -> state == READY || tempPCB -> state ==
      RUNNING ) tempPCB -> state = SUSPENDED_READY;
                        if( tempPCB -> state == BLOCKED ) tempPCB -> state = SUSP
      ENDED_BLOCKED;
                        insert_PCB(tempPCB);
                }else{
                        printf("Process Name not found!");
                        return;
                }
          }
          else{
                printf("Wrong number of arguments used");
                return;
          }
}
```

4.9.3.17 void mpxcmd_unblock (int *argc*, char * *argv[ ]*)

This is a user function in the menu that puts a process in the unblocked state it takes the process name as input.

Definition at line 459 of file mpx_r2.c.

```
                                        {
  if(argc==2){
          char name[STRLEN];
          PCB *pointer;
          PCB *tempPCB;
          int buffs = STRLEN;

          strcpy(name,argv[1]);

          pointer = find_PCB(name);
          if ( pointer != NULL){
                  tempPCB = copy_PCB(pointer);
                  remove_PCB(pointer);
                  if( tempPCB -> state == BLOCKED ) tempPCB -> state = READ
Y;
                  if( tempPCB -> state == SUSPENDED_BLOCKED ) tempPCB -> st
ate = SUSPENDED_READY;
                  insert_PCB(tempPCB);
          }else{
                  printf("Process Name not found!");
                  return;
          }
  }
  else{
          printf("Wrong number of arguments used");
          return;
  }
}
```

4.9.3.18 int setup_PCB (PCB * *pointer*, char * *Name*, int *classType*, int *state*, int *priority*)

This Function initializes the contents of a PCB and checks the values if correct returns 0 if not returns 1.

Definition at line 83 of file mpx_r2.c.

```
{//FIXME: NO DATA VV
  int ret;
  char *name = pointer -> name;
  ret = 0;
  strcpy(name,Name);

  if( find_PCB(name) == NULL){
          if( classType == 1 || classType == 0 ){
                  pointer -> classType = classType;
          }else{
                  ret = 1;
          }
           if( state == BLOCKED ||
                    state == SUSPENDED_READY ||
               state == SUSPENDED_BLOCKED ||
                    state == READY ||
                    state == RUNNING )
          {
          pointer -> state = state;
          }else{
                  ret = 1;
          }
          if( priority <= 127 && priority >= -128){
                  pointer -> priority = priority;
          }else{
                  ret = 1;
          }
  }else{
```

```
                ret = 1;
        }
        return ret;
}
```

4.10 src/mpx_r2.h

```
00001 #ifndef MPX_R2_HFILE
00002 #define MPX_R2_HFILE
00003 /* Symbolic Constants */
00004 #define RUNNING  0///< state is Defined as 0
00005 #define READY   1///< state is Defined as  1
00006 #define BLOCKED 2 ///< state is defined as 2
00007
00008 #define SUSPENDED_READY    3 ///< is defined by 3
00009 #define SUSPENDED_BLOCKED  4 ///< is defined by 4
00010
00011 #define SYSTEM  1 ///< is defined as 1
00012 #define APPLICATION  0  ///< is defined as 0
00013
00014 #define STACKSIZE   1024 ///< is the size of the stack in Bytes
00015 #define STRLEN    16///< is the length of a string for name
00016
00017 #define PORDR  1 ///< is the Priority Order flag
00018 #define FIFO   0 ///< is the First In First Out Order flag
00019 #define ZERO   0
00020
00021 #define MAX_LINE        1024
00022
00023 /* Type Definitions and Structures */
00024 typedef struct mem{
00025         int size;
00026         unsigned char *loadADDR;
00027         unsigned char *execADDR;
00028 }MEMDSC;
00029
00030 typedef struct stack{
00031         unsigned char *top;
00032         unsigned char *base;
00033 }STACKDSC;
00034
00035 typedef struct process{
00036         char name[STRLEN];
00037         signed char classType;
00038         signed char priority;
00039         signed char state;
00040         MEMDSC *memdsc;
00041         STACKDSC *stackdsc;
00042 }PCB;
00043
00044 typedef struct page{
00045         PCB *process;
00046         unsigned char *left;
00047         unsigned char *right;
00048 }ELEM;
00049
00050 typedef struct root{
00051         int count;
00052         unsigned char *node;
00053 }ROOT;
00054
00055
00056
00057 /* Functions Dec*/
00058 PCB *alloocate_PCB(void);
```

```
00059 int free_PCB( PCB *pointer);
00060 int setup_PCB( PCB *pointer, char *name, int classType, int state, int priority )
      ;
00061 void insert_PCB(PCB *PCBpointer/*< pointer to a PCB to insert*/ );
00062 void insert_PORDR( PCB *PCBpointer, ROOT *quequeROOT );
00063 void insert_FIFO( PCB *PCBpointer, ROOT *quequeROOT);
00064 PCB *find_PCB( char *name);
00065 void mpxcmd_create_PCB(int argc, char *argv[]);
00066 void mpxcmd_delete_PCB(int argc, char *argv[]);
00067 void mpxcmd_block(int argc, char *argv[]);
00068 void mpxcmd_unblock(int argc, char *argv[]);
00069 void mpxcmd_suspend(int argc, char *argv[]);
00070 void mpxcmd_resume(int argc, char *argv[]);
00071 void mpxcmd_setPriority(int argc, char *argv[]);
00072 void mpxcmd_show_PCB(int argc, char *argv[]);
00073 void mpxcmd_showAll_PCB(int argc, char *argv[]);
00074 void mpxcmd_showReady_PCB(int argc, char *argv[]);
00075 void mpxcmd_showBlocked_PCB(int argc, char *argv[]);
00076 #endif
```

4.11 src/mpx_util.c File Reference

```
#include "mpx_cmd.h"
#include "mpx_util.h"
#include "mpx_supt.h"
#include "mystdlib.h"
#include <string.h>
#include <stdio.h>
```

Defines

- #define LINES_PER_PAGE 23

Functions

- void mpx_pager (char *line_to_print)

  The pager function permits displaying output screen-full at a time.

- void mpx_pager_init (char *header)

  The pager initialization function must be used before the pager function.

- int mpxprompt_yn (void)

  The function Prompt y n prompts the user to answer a Yes or No question.

- char mpxprompt_anykey (void)

  The function Prompt Any key Prompts the user to press the return key.

- int mpxprompt_int (void)

  The function Prompt int reads the in the input from the user.

- void mpx_readline (char *buffer, int buflen)

Readline function reads in a line from the Terminal.

- int mpx_cls (void)

  Clear, blanks the screen.

- void errorDecode (int err)

  Decodes the errors thrown by various functions in the MPX suport files.

4.11.1 Define Documentation

4.11.1.1 #define LINES_PER_PAGE 23

Definition at line 8 of file mpx_util.c.

4.11.2 Function Documentation

4.11.2.1 void errorDecode (int *err*)

Decodes the errors thrown by various functions in the MPX suport files.

Parameters

`[in]` ***err*** The error value to decode.

Definition at line 111 of file mpx_util.c.

```
                                                        {
  switch( err ){
          case ERR_SUP_INVDEV:
                  printf("Invalid device ID");
                  break;
          case ERR_SUP_INVOPC:
                  printf("Invalid operation Code");
                  break;
          case ERR_SUP_INVPOS:
                  printf("Invalid character postition");
                  break;
          case ERR_SUP_RDFAIL:
                  printf("Read Failed"); // could be sysrec or sys get entr
y
                  break;
          case ERR_SUP_WRFAIL:
                  printf("Write Failed");
                  break;
          // ERR_SUP_INVMOD Exists in documentation but is not present in s
upport code?
          case ERR_SUP_INVMEM:
                  printf("Invalid memory block pointer");
                  break;
          case ERR_SUP_FRFAIL:
                  printf("Memory Freeing Op Failed");
                  break;
          case ERR_SUP_INVDAT:
                  printf("Invalid Date");
                  break;
          case ERR_SUP_DATNCH:
                  printf("Date not properly changed");
```

```
                break;
        case ERR_SUP_INVDIR:
                printf("Invalid name or no such directory");
                break;
        case ERR_SUP_DIROPN:
                printf("Error Opening Directory");
                break;
        case ERR_SUP_DIRNOP:
                printf("No directory is open");
                break;
        case ERR_SUP_NOENTR:
                printf("No more entries found");
                break;
        case ERR_SUP_NAMLNG:
                printf("The name was too long for the buffer");
                break;
        case ERR_SUP_DIRCLS:
                printf("Error closing the directory");
                break;
        default:
                printf("Unknown Error Code: %d /n",err);
                break;
    }
}
```

4.11.2.2 int mpx_cls (void)

Clear, blanks the screen.

Definition at line 99 of file mpx_util.c.

```
            {
    /* fixme: add error catching */
    int err = sys_req(CLEAR, TERMINAL, NULL, 0);

    if ( err != OK ) return err;

    return OK;
}
```

4.11.2.3 void mpx_pager (char * *line_to_print*)

The pager function permits displaying output screen-full at a time.

The line to output MUST NOT end with a

(newline) character.

Definition at line 19 of file mpx_util.c.

```
                                {

    if ( lines_printed == 0 ) {
            mpx_cls();
            printf("%s", page_header);
    }

    printf("%s\n", line_to_print);

    if ( (lines_printed != 0) && (lines_printed % (LINES_PER_PAGE-header_line
s) == 0)) {
```

```
                lines_printed = 0;
                printf("<<Press enter for MORE>>"); mpxprompt_anykey();
        } else {
                lines_printed++;
        }
}
```

4.11.2.4 void mpx_pager_init (char * *header*)

The pager initialization function must be used before the pager function.

If no per-page header is required, pass NULL for that parameter.

All lines in the header, including the last one, MUST end with a

(newline) character.

Definition at line 42 of file mpx_util.c.

```
                                {
        char *cur_pos   = header;

        page_header     = header;
        lines_printed   = 0;
        pages_printed   = 0;
        header_lines    = 0;

        if (header != NULL) {
                while (*cur_pos != '\0') {
                        if (*cur_pos == '\n') {
                                header_lines++;
                        }
                        cur_pos++;
                }
        }
}
```

4.11.2.5 void mpx_readline (char * *buffer*, int *buflen*)

Readline function reads in a line from the Terminal.

Parameters

[in,out] ***buffer*** Points to the sting being read.

[in] ***buflen*** Defines the maximum characters read.

Definition at line 88 of file mpx_util.c.

```
                                                                {
        int local_buflen = buflen;
        sys_req(READ, TERMINAL, buffer, &local_buflen);

        /* remove newline from end of string. */
        if( buffer[strlen(buffer)-1] == '\n' || buffer[strlen(buffer)-1] == '\r'
      ) {
                buffer[strlen(buffer)-1] = '\0';
        } /* FIXME: strlen() is unsafe; should use strnlen(). */
}
```

4.11.2.6 char mpxprompt_anykey (void)

The function Prompt Any key Prompts the user to press the return key.

Definition at line 71 of file mpx_util.c.

```
                        {
        /* user must press enter. */
        int buflen = 3;
        char buf[5];
        buf[0] = ' ';
        sys_req(READ, TERMINAL, buf, &buflen);
        return buf[0];
}
```

4.11.2.7 int mpxprompt_int (void)

The function Prompt int reads the in the input from the user.

Definition at line 81 of file mpx_util.c.

```
                        {
        char input[MAX_LINE];
        mpx_readline(input, MAX_LINE);
        return atoi(input);
}
```

4.11.2.8 int mpxprompt_yn (void)

The function Prompt y n prompts the user to answer a Yes or No question.

Definition at line 61 of file mpx_util.c.

```
                        {
        char yn = mpxprompt_anykey();
        if( yn == 'Y' || yn == 'y' ) {
                return 1; /* true */
        } else {
                return 0; /* false */
        }
}
```

4.12 src/mpx_util.c

```
00001 #include "mpx_cmd.h"
00002 #include "mpx_util.h"
00003 #include "mpx_supt.h"
00004 #include "mystdlib.h"
00005 #include <string.h>
00006 #include <stdio.h>
00007
00008 #define LINES_PER_PAGE 23
00009 static int lines_printed;
00010 static int pages_printed;
00011 static int header_lines;
00012 static char *page_header;
00013
```

```
00014
00019 void mpx_pager(char *line_to_print) {
00020
00021         if ( lines_printed == 0 ) {
00022                 mpx_cls();
00023                 printf("%s", page_header);
00024         }
00025
00026         printf("%s\n", line_to_print);
00027
00028         if ( (lines_printed != 0) && (lines_printed % (LINES_PER_PAGE-header_line
      s) == 0)) {
00029                 lines_printed = 0;
00030                 printf("<<Press enter for MORE>>"); mpxprompt_anykey();
00031         } else {
00032                 lines_printed++;
00033         }
00034 }
00035
00042 void mpx_pager_init(char *header) {
00043         char *cur_pos   = header;
00044
00045         page_header     = header;
00046         lines_printed   = 0;
00047         pages_printed   = 0;
00048         header_lines    = 0;
00049
00050         if (header != NULL) {
00051                 while (*cur_pos != '\0') {
00052                         if (*cur_pos == '\n') {
00053                                 header_lines++;
00054                         }
00055                         cur_pos++;
00056                 }
00057         }
00058 }
00059
00061 int mpxprompt_yn(void) {
00062         char yn = mpxprompt_anykey();
00063         if( yn == 'Y' || yn == 'y' ) {
00064                 return 1; /* true */
00065         } else {
00066                 return 0; /* false */
00067         }
00068 }
00069
00071 char mpxprompt_anykey(void) {
00072         /* user must press enter. */
00073         int buflen = 3;
00074         char buf[5];
00075         buf[0] = ' ';
00076         sys_req(READ, TERMINAL, buf, &buflen);
00077         return buf[0];
00078 }
00079
00081 int mpxprompt_int(void) {
00082         char input[MAX_LINE];
00083         mpx_readline(input, MAX_LINE);
00084         return atoi(input);
00085 }
00086
00088 void mpx_readline ( char *buffer , int buflen  ) {
00089         int local_buflen = buflen;
00090         sys_req(READ, TERMINAL, buffer, &local_buflen);
00091
00092         /* remove newline from end of string. */
00093         if( buffer[strlen(buffer)-1] == '\n' || buffer[strlen(buffer)-1] == '\r'
```

```
    ) {
00094                   buffer[strlen(buffer)-1] = '\0';
00095           } /* FIXME: strlen() is unsafe; should use strnlen(). */
00096 }
00097
00099 int mpx_cls (void) {
00100           /* fixme: add error catching */
00101           int err = sys_req(CLEAR, TERMINAL, NULL, 0);
00102
00103           if ( err != OK ) return err;
00104
00105           return OK;
00106 }
00107
00111 void errorDecode(int err){
00112           switch( err ){
00113                   case ERR_SUP_INVDEV:
00114                           printf("Invalid device ID");
00115                           break;
00116                   case ERR_SUP_INVOPC:
00117                           printf("Invalid operation Code");
00118                           break;
00119                   case ERR_SUP_INVPOS:
00120                           printf("Invalid character postition");
00121                           break;
00122                   case ERR_SUP_RDFAIL:
00123                           printf("Read Failed"); // could be sysrec or sys get entr
    y
00124                           break;
00125                   case ERR_SUP_WRFAIL:
00126                           printf("Write Failed");
00127                           break;
00128                   // ERR_SUP_INVMOD Exists in documentation but is not present in s
    upport code?
00129                   case ERR_SUP_INVMEM:
00130                           printf("Invalid memory block pointer");
00131                           break;
00132                   case ERR_SUP_FRFAIL:
00133                           printf("Memory Freeing Op Failed");
00134                           break;
00135                   case ERR_SUP_INVDAT:
00136                           printf("Invalid Date");
00137                           break;
00138                   case ERR_SUP_DATNCH:
00139                           printf("Date not properly changed");
00140                           break;
00141                   case ERR_SUP_INVDIR:
00142                           printf("Invalid name or no such directory");
00143                           break;
00144                   case ERR_SUP_DIROPN:
00145                           printf("Error Opening Directory");
00146                           break;
00147                   case ERR_SUP_DIRNOP:
00148                           printf("No directory is open");
00149                           break;
00150                   case ERR_SUP_NOENTR:
00151                           printf("No more entries found");
00152                           break;
00153                   case ERR_SUP_NAMLNG:
00154                           printf("The name was too long for the buffer");
00155                           break;
00156                   case ERR_SUP_DIRCLS:
00157                           printf("Error closing the directory");
00158                           break;
00159                   default:
00160                           printf("Unknown Error Code: %d /n",err);
00161                           break;
```

```
00162         }
00163 }
```

4.13 src/mpx_util.h File Reference

Functions

- void mpx_pager (char *line_to_print)

  The pager function permits displaying output screen-full at a time.

- void mpx_pager_init (char *header)

  The pager initialization function must be used before the pager function.

- int mpx_cls (void)

  Clear, blanks the screen.

- int mpxprompt_yn (void)

  The function Prompt y n prompts the user to answer a Yes or No question.

- void mpx_readline (char *buffer, int buflen)

  Readline function reads in a line from the Terminal.

- char mpxprompt_anykey (void)

  The function Prompt Any key Prompts the user to press the return key.

- int mpxprompt_int (void)

  The function Prompt int reads the in the input from the user.

- void errorDecode (int err)

  Decodes the errors thrown by various functions in the MPX suport files.

4.13.1 Function Documentation

4.13.1.1 void errorDecode (int *err*)

Decodes the errors thrown by various functions in the MPX suport files.

Parameters

`[in]` ***err*** The error value to decode.

Definition at line 111 of file mpx_util.c.

```
                                                        {
    switch( err ){
            case ERR_SUP_INVDEV:
                    printf("Invalid device ID");
                    break;
            case ERR_SUP_INVOPC:
                    printf("Invalid operation Code");
```

```
                break;
            case ERR_SUP_INVPOS:
                printf("Invalid character postition");
                break;
            case ERR_SUP_RDFAIL:
                printf("Read Failed"); // could be sysrec or sys get entr
y
                break;
            case ERR_SUP_WRFAIL:
                printf("Write Failed");
                break;
            // ERR_SUP_INVMOD Exists in documentation but is not present in s
upport code?
            case ERR_SUP_INVMEM:
                printf("Invalid memory block pointer");
                break;
            case ERR_SUP_FRFAIL:
                printf("Memory Freeing Op Failed");
                break;
            case ERR_SUP_INVDAT:
                printf("Invalid Date");
                break;
            case ERR_SUP_DATNCH:
                printf("Date not properly changed");
                break;
            case ERR_SUP_INVDIR:
                printf("Invalid name or no such directory");
                break;
            case ERR_SUP_DIROPN:
                printf("Error Opening Directory");
                break;
            case ERR_SUP_DIRNOP:
                printf("No directory is open");
                break;
            case ERR_SUP_NOENTR:
                printf("No more entries found");
                break;
            case ERR_SUP_NAMLNG:
                printf("The name was too long for the buffer");
                break;
            case ERR_SUP_DIRCLS:
                printf("Error closing the directory");
                break;
            default:
                printf("Unknown Error Code: %d /n",err);
                break;
        }
}
```

4.13.1.2 int mpx_cls (void)

Clear, blanks the screen.

Definition at line 99 of file mpx_util.c.

```
            {
    /* fixme: add error catching */
    int err = sys_req(CLEAR, TERMINAL, NULL, 0);

    if ( err != OK ) return err;

    return OK;
}
```

4.13.1.3 void mpx_pager (char * *line_to_print*)

The pager function permits displaying output screen-full at a time.

The line to output MUST NOT end with a

(newline) character.

Definition at line 19 of file mpx_util.c.

```
                                    {

        if ( lines_printed == 0 ) {
                mpx_cls();
                printf("%s", page_header);
        }

        printf("%s\n", line_to_print);

        if ( (lines_printed != 0) && (lines_printed % (LINES_PER_PAGE-header_line
      s) == 0)) {
                lines_printed = 0;
                printf("<<Press enter for MORE>>"); mpxprompt_anykey();
        } else {
                lines_printed++;
        }
}
```

4.13.1.4 void mpx_pager_init (char * *header*)

The pager initialization function must be used before the pager function.

If no per-page header is required, pass NULL for that parameter.

All lines in the header, including the last one, MUST end with a

(newline) character.

Definition at line 42 of file mpx_util.c.

```
                                   {
        char *cur_pos   = header;

        page_header     = header;
        lines_printed   = 0;
        pages_printed   = 0;
        header_lines    = 0;

        if (header != NULL) {
                while (*cur_pos != '\0') {
                        if (*cur_pos == '\n') {
                                header_lines++;
                        }
                        cur_pos++;
                }
        }
}
```

4.13.1.5 void mpx_readline (char * *buffer,* int *buflen*)

Readline function reads in a line from the Terminal.

Parameters

[in,out] ***buffer*** Points to the sting being read.

[in] ***buflen*** Defines the maximum characters read.

Definition at line 88 of file mpx_util.c.

```
                                                                    {
        int local_buflen = buflen;
        sys_req(READ, TERMINAL, buffer, &local_buflen);

        /* remove newline from end of string. */
        if( buffer[strlen(buffer)-1] == '\n' || buffer[strlen(buffer)-1] == '\r'
      ) {
                buffer[strlen(buffer)-1] = '\0';
        } /* FIXME: strlen() is unsafe; should use strnlen(). */
}
```

4.13.1.6 char mpxprompt_anykey (void)

The function Prompt Any key Prompts the user to press the return key.

Definition at line 71 of file mpx_util.c.

```
                        {
        /* user must press enter. */
        int buflen = 3;
        char buf[5];
        buf[0] = ' ';
        sys_req(READ, TERMINAL, buf, &buflen);
        return buf[0];
}
```

4.13.1.7 int mpxprompt_int (void)

The function Prompt int reads the in the input from the user.

Definition at line 81 of file mpx_util.c.

```
                    {
        char input[MAX_LINE];
        mpx_readline(input, MAX_LINE);
        return atoi(input);
}
```

4.13.1.8 int mpxprompt_yn (void)

The function Prompt y n prompts the user to answer a Yes or No question.

Definition at line 61 of file mpx_util.c.

```
                    {
        char yn = mpxprompt_anykey();
        if( yn == 'Y' || yn == 'y' ) {
                return 1; /* true */
        } else {
                return 0; /* false */
        }
}
```

4.14 src/mpx_util.h

```
00001 #ifndef MPX_UTIL_HFILE
00002 #define MPX_UTIL_HFILE
00003
00004 void    mpx_pager               (char *line_to_print);
00005 void    mpx_pager_init          (char *header);
00006 int     mpx_cls                 (void);
00007 int     mpxprompt_yn            (void);
00008 void    mpx_readline            (char *buffer, int buflen);
00009 char    mpxprompt_anykey        (void);
00010 int     mpxprompt_int           (void);
00011 void    errorDecode             (int err);
00012
00013 #endif
```

Index