PMOS Programmers Manual

1.0

Generated by Doxygen 1.7.1

Fri Dec 10 2010 01:08:37

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 context Struct Reference

`#include <MPX_R3.H>`

Data Fields

- unsigned int BP
- unsigned int DI
- unsigned int SI
- unsigned int DS
- unsigned int ES
- unsigned int DX
- unsigned int CX
- unsigned int BX
- unsigned int AX
- unsigned int IP
- unsigned int CS
- unsigned int FLAGS
- word BP
- word DI
- word SI
- word DS
- word ES
- word DX
- word CX
- word BX
- word AX
- word IP
- word CS
- word FLAGS

3.1.1 Detailed Description

Definition at line 25 of file MPX_R3.H.

3.1.2 Field Documentation

3.1.2.1 unsigned int context::AX

Definition at line 27 of file MPX_R3.H.

3.1.2.2 word context::AX

Definition at line 66 of file trmdrive.c.

3.1.2.3 unsigned int context::BP

Definition at line 26 of file MPX_R3.H.

3.1.2.4 word context::BP

Definition at line 65 of file trmdrive.c.

3.1.2.5 unsigned int context::BX

Definition at line 27 of file MPX_R3.H.

3.1.2.6 word context::BX

Definition at line 66 of file trmdrive.c.

3.1.2.7 unsigned int context::CS

Definition at line 28 of file MPX_R3.H.

3.1.2.8 word context::CS

Definition at line 67 of file trmdrive.c.

3.1.2.9 word context::CX

Definition at line 66 of file trmdrive.c.

3.1.2.10 unsigned int context::CX

Definition at line 27 of file MPX_R3.H.

3.1.2.11 unsigned int context::DI

Definition at line 26 of file MPX_R3.H.

3.1.2.12 word context::DI

Definition at line 65 of file trmdrive.c.

3.1.2.13 unsigned int context::DS

Definition at line 26 of file MPX_R3.H.

3.1.2.14 word context::DS

Definition at line 65 of file trmdrive.c.

3.1.2.15 word context::DX

Definition at line 66 of file trmdrive.c.

3.1.2.16 unsigned int context::DX

Definition at line 27 of file MPX_R3.H.

3.1.2.17 unsigned int context::ES

Definition at line 26 of file MPX_R3.H.

3.1.2.18 word context::ES

Definition at line 65 of file trmdrive.c.

3.1.2.19 unsigned int context::FLAGS

Definition at line 28 of file MPX_R3.H.

3.1.2.20 word context::FLAGS

Definition at line 67 of file trmdrive.c.

3.1.2.21 unsigned int context::IP

Definition at line 28 of file MPX_R3.H.

3.1.2.22 word context::IP

Definition at line 67 of file trmdrive.c.

3.1.2.23 word context::SI

Definition at line 65 of file trmdrive.c.

3.1.2.24 unsigned int context::SI

Definition at line 26 of file MPX_R3.H.

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

- src/MPX_R3.H
- src/trmdrive.c

3.2 device Struct Reference

`#include <MPX_R5.h>`

Data Fields

- int flag
- int * flag_ptr
- int status
- char * inbuff
- int * incount
- int indone
- char * outbuff
- int * outcount
- int outdone
- char ringbuf [size]
- int ringbufin
- int ringbufout
- int ringbufcount

3.2.1 Detailed Description

Definition at line 50 of file MPX_R5.h.

3.2.2 Field Documentation

3.2.2.1 int device::flag

Definition at line 51 of file MPX_R5.h.

3.2.2.2 int* device::flag_ptr

Definition at line 52 of file MPX_R5.h.

3.2.2.3 char* device::inbuff

Definition at line 54 of file MPX_R5.h.

3.2.2.4 int* device::incount

Definition at line 55 of file MPX_R5.h.

3.2.2.5 int device::indone

Definition at line 56 of file MPX_R5.h.

3.2.2.6 char* device::outbuff

Definition at line 57 of file MPX_R5.h.

3.2.2.7 int* device::outcount

Definition at line 58 of file MPX_R5.h.

3.2.2.8 int device::outdone

Definition at line 59 of file MPX_R5.h.

3.2.2.9 char device::ringbuf[size]

Definition at line 60 of file MPX_R5.h.

3.2.2.10 int device::ringbufcount

Definition at line 63 of file MPX_R5.h.

3.2.2.11 int device::ringbufin

Definition at line 61 of file MPX_R5.h.

3.2.2.12 int device::ringbufout

Definition at line 62 of file MPX_R5.h.

3.2.2.13 int device::status

Definition at line 53 of file MPX_R5.h.

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

- src/MPX_R5.h

3.3 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.

3.3.1 Detailed Description

Definition at line 43 of file mpx_r2.h.

3.3.2 Field Documentation

3.3.2.1 unsigned char* mem::execADDR

Address of first INSTRUCTION.

Definition at line 46 of file mpx_r2.h.

3.3.2.2 unsigned char* mem::loadADDR

Address to load data to.

Definition at line 45 of file mpx_r2.h.

3.3.2.3 int mem::size

Number of words in memory.

Definition at line 44 of file mpx_r2.h.

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

- src/mpx_r2.h

3.4 mpx_cmd Struct Reference

`#include <mpx_cmd.h>`

Data Fields

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

3.4.1 Detailed Description

Definition at line 32 of file mpx_cmd.h.

3.4.2 Field Documentation

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

Definition at line 35 of file mpx_cmd.h.

3.4.2.2 char* mpx_cmd::cmd_name

Definition at line 33 of file mpx_cmd.h.

3.4.2.3 struct mpx_cmd* mpx_cmd::next

Definition at line 34 of file mpx_cmd.h.

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

- src/mpx_cmd.h

3.5 page Struct Reference

`#include <mpx_r2.h>`

Data Fields

- PCB * process
  pointer to the PCB structure
- struct page * left
  pointer to the left PCB structure
- struct page * right
  pointer to the right PCB structure

3.5.1 Detailed Description

Definition at line 63 of file mpx_r2.h.

3.5.2 Field Documentation

3.5.2.1 struct page* page::left

pointer to the left PCB structure

Definition at line 65 of file mpx_r2.h.

3.5.2.2 PCB* page::process

pointer to the PCB structure

Definition at line 64 of file mpx_r2.h.

3.5.2.3 struct page* page::right

pointer to the right PCB structure

Definition at line 66 of file mpx_r2.h.

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

- src/mpx_r2.h

3.6 params Struct Reference

`#include <MPX_R3.H>`

Data Fields

- int op_code
- int device_id
- unsigned char * buf_addr
- int * cont_addr

3.6.1 Detailed Description

Definition at line 31 of file MPX_R3.H.

3.6.2 Field Documentation

3.6.2.1 unsigned char* params::buf_addr

Definition at line 34 of file MPX_R3.H.

3.6.2.2 int* params::cont_addr

Definition at line 35 of file MPX_R3.H.

3.6.2.3 int params::device_id

Definition at line 33 of file MPX_R3.H.

3.6.2.4 int params::op_code

Definition at line 32 of file MPX_R3.H.

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

- src/MPX_R3.H

3.7 process Struct Reference

`#include <mpx_r2.h>`

Data Fields

- char name [STRLEN]
  character array containing 16 characters plus space for null
- int classType
  class of process APPLICATION or SYSTEM
- int priority
  process priority ranges from -128 to +127
- int 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;

3.7.1 Detailed Description

Definition at line 54 of file mpx_r2.h.

3.7.2 Field Documentation

3.7.2.1 int process::classType

class of process APPLICATION or SYSTEM

Definition at line 56 of file mpx_r2.h.

3.7.2.2 MEMDSC* process::memdsc

stores the description of the ADDRESS SPACE for the process

Definition at line 59 of file mpx_r2.h.

3.7.2.3 char process::name[STRLEN]

character array containing 16 characters plus space for null

Definition at line 55 of file mpx_r2.h.

3.7.2.4 int process::priority

process priority ranges from -128 to +127

Definition at line 57 of file mpx_r2.h.

3.7.2.5 STACKDSC* process::stackdsc

stores the description of the stack for each process;

Definition at line 60 of file mpx_r2.h.

3.7.2.6 int process::state

stores the current states of the process

Definition at line 58 of file mpx_r2.h.

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

- src/mpx_r2.h

3.8 root Struct Reference

`#include <mpx_r2.h>`

Data Fields

- int count
- ELEM * node

3.8.1 Detailed Description

Definition at line 69 of file mpx_r2.h.

3.8.2 Field Documentation

3.8.2.1 int root::count

Definition at line 70 of file mpx_r2.h.

3.8.2.2 ELEM* root::node

Definition at line 71 of file mpx_r2.h.

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

- src/mpx_r2.h

3.9 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

3.9.1 Detailed Description

Definition at line 49 of file mpx_r2.h.

3.9.2 Field Documentation

3.9.2.1 unsigned char* stack::base

pointer to the bottom of the stack

Definition at line 51 of file mpx_r2.h.

3.9.2.2 unsigned char* stack::top

pointer to the top of the stack

Definition at line 50 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 <dos.h>
#include "mpx_supt.h"
#include "mpx_util.h"
#include "mpx_cmd.h"
#include "mpx_r4.h"
#include "mpx_r3.h"
#include "mpx_r2.h"
#include "mpx_r5.h"
#include "TRMDRIVE.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 38 of file MPX.C.

```
{
int err;
PCB *command_loop, *idlePCB;
STACKDSC *command_stack;
```

```
    tcontext *command_context;
    tcontext *tempContext;
    char dir[20] = "proc";
    char name[20] = "idle";
    char filename[20] = "IDLE";
    int sizex,offset,priority;
    int eventFlag, eventtFlag;
    char command[20] = "Command_Hand";
    sys_init( MODULE_R4 ); //System initilization
    sys_set_vec(sys_call);

    //Open Device Drivers
    //com_open( (int *) eventFlag, 1200);
   //trm_open(  (int *) eventtFlag );

    // Command Handler loop insertion
    command_loop = allocate_PCB();

    command_stack = command_loop -> stackdsc;

    command_stack->top = command_stack->base + STACKSIZE - sizeof(tcontext);

    command_context = (tcontext*) command_stack->top;

    command_context->DS = _DS;
    command_context->ES = _ES;
    command_context->CS = FP_SEG(&mpx_command_loop);
    command_context->IP = FP_OFF(&mpx_command_loop);
    command_context->FLAGS = 0x200;

    setup_PCB(command_loop,command,SYSTEM,READY,-127);

    insert_PCB(command_loop);

    //IDLE Process insertion

    sys_check_program(dir,filename,&sizex,&offset);

    idlePCB = allocate_PCB();
    setup_PCB(idlePCB,name,APPLICATION,READY,127);

    idlePCB->memdsc->loadADDR = sys_alloc_mem(sizex);
    idlePCB->memdsc->execADDR = idlePCB->memdsc->loadADDR + offset;

    idlePCB->stackdsc->top = idlePCB->stackdsc->base + STACKSIZE - sizeof(
  tcontext);

    tempContext = (tcontext *) (idlePCB->stackdsc->top);
    tempContext->ES = _ES;
    tempContext->DS = _DS;
    tempContext->CS = FP_SEG(idlePCB->memdsc->execADDR);
    tempContext->IP = FP_OFF(idlePCB->memdsc->execADDR);
    tempContext->FLAGS = 0x200;

    sys_load_program(idlePCB->memdsc->loadADDR,sizex,dir,filename);

    insert_PCB(idlePCB);


    dispatch();
    printf("exit sucess");
    //mpxprompt_anykey();
    sys_exit();
}
```

4.2 src/MPX.C

```
00001 /************************************************************************
00002         MPX: The MultiProgramming eXecutive
00003         Project to Accompany
00004         A Practical Approach to Operating Systems
00005         Malcolm G. Lane & James D. Mooney
00006         Copyright 1993, P.W.S. Kent Publishing Co., Boston, MA.
00007
00008         File Name:      mpx.c
00009
00010         Author: Nathaniel Clay and Nicholas Yanak
00011         Version: 1.1
00012         Date:  12/9/2010
00013
00014         Purpose: This is the startup program. It sets initial values
00015         and then calls the dispatcher.
00016
00017
00018         Environment: Windows XP 32 bit
00019
00020 ************************************************************************/
00021
00022 #include <dos.h>
00023 #include "mpx_supt.h"
00024 #include "mpx_util.h"
00025 #include "mpx_cmd.h"
00026 #include "mpx_r4.h"
00027 #include "mpx_r3.h"
00028 #include "mpx_r2.h"
00029 #include "mpx_r5.h"
00030 #include "TRMDRIVE.H"
00031
00032
00038 void main(){
00039         int err;
00040         PCB *command_loop, *idlePCB;
00041         STACKDSC *command_stack;
00042         tcontext *command_context;
00043         tcontext *tempContext;
00044         char dir[20] = "proc";
00045         char name[20] = "idle";
00046         char filename[20] = "IDLE";
00047         int sizex,offset,priority;
00048         int eventFlag, eventtFlag;
00049         char command[20] = "Command_Hand";
00050         sys_init( MODULE_R4 ); //System initilization
00051         sys_set_vec(sys_call);
00052
00053         //Open Device Drivers
00054         //com_open( (int *) eventFlag, 1200);
00055        //trm_open(  (int *) eventtFlag );
00056
00057         // Command Handler loop insertion
00058         command_loop = allocate_PCB();
00059
00060         command_stack = command_loop -> stackdsc;
00061
00062         command_stack->top = command_stack->base + STACKSIZE - sizeof(tcontext);
00063
00064         command_context = (tcontext*) command_stack->top;
00065
00066         command_context->DS = _DS;
00067         command_context->ES = _ES;
00068         command_context->CS = FP_SEG(&mpx_command_loop);
00069         command_context->IP = FP_OFF(&mpx_command_loop);
00070         command_context->FLAGS = 0x200;
```

```
00071
00072         setup_PCB(command_loop,command,SYSTEM,READY,-127);
00073
00074         insert_PCB(command_loop);
00075
00076         //IDLE Process insertion
00077
00078         sys_check_program(dir,filename,&sizex,&offset);
00079
00080         idlePCB = allocate_PCB();
00081         setup_PCB(idlePCB,name,APPLICATION,READY,127);
00082
00083         idlePCB->memdsc->loadADDR = sys_alloc_mem(sizex);
00084         idlePCB->memdsc->execADDR = idlePCB->memdsc->loadADDR + offset;
00085
00086         idlePCB->stackdsc->top = idlePCB->stackdsc->base + STACKSIZE - sizeof(
      tcontext);
00087
00088         tempContext = (tcontext *) (idlePCB->stackdsc->top);
00089         tempContext->ES = _ES;
00090         tempContext->DS = _DS;
00091         tempContext->CS = FP_SEG(idlePCB->memdsc->execADDR);
00092         tempContext->IP = FP_OFF(idlePCB->memdsc->execADDR);
00093         tempContext->FLAGS = 0x200;
00094
00095         sys_load_program(idlePCB->memdsc->loadADDR,sizex,dir,filename);
00096
00097         insert_PCB(idlePCB);
00098
00099
00100         dispatch();
00101         printf("exit sucess");
00102         //mpxprompt_anykey();
00103         sys_exit();
00104 }
```

4.3 src/MPX_CMD.C File Reference

```
#include "mpx_supt.h"
#include "mpx_cmd.h"
#include "mpx_util.h"
#include "mpx_r2.h"
#include "mpx_r3.h"
#include "mpx_r4.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*)

< simply adds if list empty

< traverses list until end is reached then adds

Definition at line 44 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.

< takes first word of user input as the command

< goes through the user input and takes each word as a token for later use as arguments in command

< searches the list for a matching command

Definition at line 74 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);
        mpx_add_command("goR3", mpxcmd_r3run);
        mpx_add_command("loadProc", loadProgram);
        mpx_add_command("terminate",terminateProcess);
        mpx_add_command("goR4", mpxcmd_gor4);
```

```
        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 263 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 347 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();
                sys_req(EXIT, NO_DEV, NULL, 0);
        }
}
```

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.

< opens the file

< goes to the end of the file

< finds out the size of the file

< returns to the beginning

< writes to the buffer the prints out

Definition at line 190 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.

< attempts to open the directory

< gets a file and puts the name in buf until none left

Definition at line 157 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 251 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 229 of file MPX_CMD.C.

```
                                        {
        mpx_cls();
        printf("\n");
        printf("  ================================================\n");
        printf("  = MPX System Version R6 - December  8, 2010 =\n");
        printf("  ================================================\n");
        printf("\n");
        printf("      by the members of PERPETUAL MOTION SQUAD:\n");
        printf("                         --------- ------ -----\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 Hayhurst\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 38 of file MPX_CMD.C.

4.3.2.2 mpx_cmd_t* cmd_head = NULL

Definition at line 41 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 36 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 37 of file MPX_CMD.C.

4.4 src/MPX_CMD.C

```
00001 /**************************************************************************
00002         MPX: The MultiProgramming eXecutive
00003         Project to Accompany
00004         A Practical Approach to Operating Systems
00005         Malcolm G. Lane & James D. Mooney
00006         Copyright 1993, P.W.S. Kent Publishing Co., Boston, MA.
00007 
00008         File Name:      mpx_cmd.c
00009 
00010         Author: Nathaniel Clay and Nicholas Yanak
00011         Version: 1.1
00012         Date:  12/9/2010
00013 
00014         Purpose: Contains functions and supporting code available in the
00015         main menu of PMOS.
00016 
00017 
00018         Environment: Windows XP 32 bit
00019 
00020 **************************************************************************/
00021 
00022 #include "mpx_supt.h"
00023 #include "mpx_cmd.h"
00024 #include "mpx_util.h"
00025 #include "mpx_r2.h"
00026 #include "mpx_r3.h"
00027 #include "mpx_r4.h"
00028 
00029 #include "mystdlib.h"
00030 #include <string.h>
00031 #include <stdio.h>
00032 
00033 
00034 /* Strings */
00035 
00036 char prompt_str[MAX_LINE]       = "MPX> ";
00037 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";
00038 char *anykey_str                = "\n<<Press Enter to Continue.>>";
00041 mpx_cmd_t *cmd_head = NULL;
00042 
00043 
00044 void mpx_add_command( char *cmd_name, void(*cmd_function)(int argc, char *argv[])
       ) {
00045 
00046         /* allocate a command object */
```

```
00047         mpx_cmd_t *command = (mpx_cmd_t*) sys_alloc_mem( sizeof(mpx_cmd_t) ); /*
      FIXME need to check for error from alloc func. */
00048
00049         /* allocate and populate the command name member. */
00050         command->cmd_name = sys_alloc_mem( strlen(cmd_name)+1 );
00051         strcpy( command->cmd_name, cmd_name );
00052
00053         /* populate the command function member. */
00054         command->cmd_function = cmd_function;
00055
00056         /* be sure to set the next-command pointer member to NULL, since this wil
      l be the new last command. */
00057         command->next = NULL;
00058
00059         /* add the command to the global list of commands. */
00060         if ( cmd_head == NULL ) {
00061                 cmd_head = command;
00062         } else {
00063                 mpx_cmd_t *last_command = cmd_head;
00064                 while ( last_command->next != NULL ) {
00065                         last_command = last_command->next;
00066                 }
00067                 last_command->next = command;
00068         }
00069 }
00070
00074 int mpx_command_loop (void) {
00075
00076         char cmd_line[MAX_LINE];
00077         char *cmd_argv[MAX_ARGS+1];
00078         int  cmd_argc;
00079         int  i;
00080         mpx_cmd_t *command;
00081
00082         mpx_add_command( "help", mpxcmd_help );
00083         mpx_add_command("load", mpxcmd_load );
00084         mpx_add_command("date", mpxcmd_date );
00085         mpx_add_command("exit", mpxcmd_exit );
00086         mpx_add_command("version", mpxcmd_version );
00087         mpx_add_command("create",mpxcmd_create_PCB);
00088         mpx_add_command("delete",mpxcmd_delete_PCB);
00089         mpx_add_command("block",mpxcmd_block);
00090         mpx_add_command("unblock",mpxcmd_unblock);
00091         mpx_add_command("suspend",mpxcmd_suspend);
00092         mpx_add_command("resume",mpxcmd_resume);
00093         mpx_add_command("setPriority",mpxcmd_setPriority);
00094         mpx_add_command("show",mpxcmd_show_PCB);
00095         mpx_add_command("showAll",mpxcmd_showAll_PCB);
00096         mpx_add_command("showReady",mpxcmd_showReady_PCB);
00097         mpx_add_command("showBlocked",mpxcmd_showBlocked_PCB);
00098         mpx_add_command("goR3", mpxcmd_r3run);
00099         mpx_add_command("loadProc", loadProgram);
00100         mpx_add_command("terminate",terminateProcess);
00101         mpx_add_command("goR4", mpxcmd_gor4);
00102
00103         for(;;){ /* infinite loop */
00104
00105                 mpx_cls();
00106
00107                 printf("%s", welcome_message_str);
00108
00109                 printf("%s", prompt_str);
00110
00111                 cmd_argc = 0;
00112
00113                 mpx_readline(cmd_line, MAX_LINE-1);
00114
```

```
00115                 cmd_argv[0] = strtok(cmd_line, " ");
00116                 cmd_argc++;
00117
00118                 /* cmd_line is invalidated after this point; use cmd_argv[] inste
      ad. */
00119
00120                 for(i=0; i<MAX_ARGS; i++){
00121                         cmd_argv[cmd_argc] = strtok(NULL, " ");
00122                         if( cmd_argv[cmd_argc] == NULL ){
00123                                 break;
00124                         }
00125                         cmd_argc++;
00126                 }
00127
00128                 /* handle too-many-args error condition. */
00129                 if (i == MAX_ARGS && strtok(NULL, " ") != NULL) {
00130                         printf("ERROR: Argument list too long.\n");
00131                         printf("%s", anykey_str); mpxprompt_anykey();
00132                         continue;
00133                 }
00134
00135                 /* run the command function that the user requested,
00136                  * or print an error message if it is not valid. */
00137                 command = cmd_head;
00138                 while (command != NULL) {
00139                         if ( strcmp(command->cmd_name, cmd_argv[0]) == 0 ) {
00140                                 command->cmd_function( cmd_argc, cmd_argv );
00141                                 break;
00142                         }
00143                         command = command->next;
00144                 }
00145
00146                 /* if we did not find the requested command in the list of comman
      ds,
00147                  * print an appropriate error message. */
00148                 if ( command == NULL ) {
00149                          printf("Invalid command.\n");
00150                          printf("%s", anykey_str); mpxprompt_anykey();
00151                 }
00152         }
00153 }
00154
00157 void mpxcmd_load (int argc, char *argv[]) {
00158         char buf[10];
00159         char line_buf[MAX_LINE];
00160         long file_size;
00161         int  num_mpx_files = 0;
00162
00163         mpx_cls();
00164
00165         if( sys_open_dir(NULL) != 0 ){
00166                 printf("WARNING: Failed to open MPX directory!\n");
00167                 printf("%s", anykey_str); mpxprompt_anykey();
00168                 return;
00169         }
00170
00171         mpx_pager_init("  Contents of MPX Directory (.mpx Files):\n  ============
      ============================\n    SIZE        NAME\n    ----------  --------------
      ------------------------------\n");
00172         while( sys_get_entry(buf, 9, &file_size) == 0 ){
00173                 /* snprintf(&line_buf, MAX_LINE, "    %10ld  %s", file_size, buf)
      ; */
00174                 sprintf(&line_buf, "    %10ld  %s", file_size, buf);
00175                 mpx_pager(&line_buf);
00176                 num_mpx_files++;
00177         }
00178
```

```
00179         sys_close_dir();
00180
00181         if (num_mpx_files == 0) {
00182                 printf("\n There aren't any .mpx files in the MPX directory!\n\n"
      );
00183         }
00184
00185         printf("%s", anykey_str); mpxprompt_anykey();
00186         return;
00187 }
00188
00190 void mpxcmd_help(int argc, char *argv[]){
00191         FILE *fp;
00192         long fileSize;
00193         char* buffer;
00194         char fileName[100];
00195         size_t data;
00196         strcpy(fileName,argv[1]);
00197         sprintf(buffer,"help\\%s\.txt",fileName);
00198
00199
00200         if(argc==2){ // specific function help
00201                 fp=fopen(buffer,"r");
00202                 fseek(fp,0,SEEK_END);
00203                 fileSize=ftell(fp);
00204                 rewind(fp);
00205                 buffer = (char*) sys_alloc_mem(sizeof(char)*fileSize);
00206                 data = fread (buffer,1,fileSize,fp);
00207
00208                 printf("%s",buffer);
00209         }
00210         else if(argc==1){ // general help
00211                 fp=fopen("help\\help.txt","r");
00212                 fseek(fp,0,SEEK_END);
00213                 fileSize=ftell(fp);
00214                 rewind(fp);
00215                 buffer = (char*) sys_alloc_mem(sizeof(char)*fileSize);
00216                 data = fread (buffer,1,fileSize,fp);
00217                 printf("%s",buffer);
00218         }
00219         else{
00220                 printf("Wrong number of arguments used or no such command");
00221                 return;
00222         }
00223         fclose(fp);
00224         printf("%s", anykey_str); mpxprompt_anykey();
00225         return;
00226 }
00227
00229 void mpxcmd_version (int argc, char *argv[]) {
00230         mpx_cls();
00231         printf("\n");
00232         printf("  ================================================\n");
00233         printf("  = MPX System Version R6 - December  8, 2010 =\n");
00234         printf("  ================================================\n");
00235         printf("\n");
00236         printf("      by the members of PERPETUAL MOTION SQUAD:\n");
00237         printf("                          --------- ------ -----\n");
00238         printf("\n");
00239         printf("                      *  Nicholas Yanak  *\n");
00240         printf("\n");
00241         printf("                             *  Nathan Clay  *\n");
00242         printf("\n");
00243         printf("\n");
00244         printf("  WVU Fall 2010 CS450 w/ Lec. Camille Hayhurst\n");
00245
00246         printf("%s", anykey_str); mpxprompt_anykey();
```

```
00247         return;
00248 }
00249
00251 void mpxcmd_prompt (void) {
00252         printf("\n");
00253         printf("  Current prompt is: \"%s\"\n", prompt_str);
00254         printf("\n");
00255         printf("Enter new prompt: ");
00256         mpx_readline( prompt_str, MAX_LINE );
00257
00258         printf("%s", anykey_str); mpxprompt_anykey();
00259         return;
00260 }
00261
00263 void mpxcmd_date (int argc, char *argv[]) {
00264         date_rec date;
00265         sys_get_date(&date);
00266         printf("\n");
00267         printf("  System Date:\n");
00268         printf("    %2d/%2d/%4d\n", date.month, date.day, date.year);
00269         printf("   (mm/dd/yyyy)\n");
00270         printf("\n");
00271         printf("Change it (y/n)? ");
00272         if( mpxprompt_yn() ) {
00273                 int is_leapyear;
00274                 int max_days;
00275
00276                 printf("\n");
00277
00278                 printf("  New YEAR:  "); date.year      = mpxprompt_int();
00279                 if( !(date.year >=1900 && date.year < 10000) ){
00280                         /* invalid year entered. */
00281                         printf("\nInvalid year entered.\n");
00282                         printf("%s", anykey_str); mpxprompt_anykey();
00283                         return;
00284                 }
00285
00286                 is_leapyear = ((date.year%4==0 && date.year%100!=0)||(date.year%4
      00==0));
00287
00288                 printf("  New MONTH: "); date.month     = mpxprompt_int();
00289
00290                 switch (date.month) {
00291                         case 1:
00292                         case 3:
00293                         case 5:
00294                         case 7:
00295                         case 8:
00296                         case 10:
00297                         case 12:
00298                                 max_days = 31;
00299                         break;
00300
00301                         case 4:
00302                         case 6:
00303                         case 9:
00304                         case 11:
00305                                 max_days = 30;
00306                         break;
00307
00308                         case 2:
00309                                 if( is_leapyear ) {
00310                                         max_days = 29;
00311                                 } else {
00312                                         max_days = 28;
00313                                 }
00314                         break;
```

```
00315
00316                           default:
00317                                   /* invalid month entered. */
00318                                   printf("\nInvalid month entered.\n");
00319                                   printf("%s", anykey_str); mpxprompt_anykey();
00320                                   return;
00321                           /* break;  commented out to prevent turbo c++ "unreachabl
      e code" warning. */
00322                   }
00323
00324                   printf("  New DAY:   "); date.day       = mpxprompt_int();
00325
00326                   if( !(date.day > 0 && date.day <= max_days) ){
00327                           /* invalid day entered. */
00328                           printf("\nInvalid day entered.\n");
00329                           printf("%s", anykey_str); mpxprompt_anykey();
00330                           return;
00331                   }
00332
00333                   /* set the system date. */
00334                   if( sys_set_date(&date) == 0 ){
00335                           printf("Date successfully set!\n");
00336                   } else {
00337                           printf("WARNING:\n");
00338                           printf("sys_set_date() returned error.\n");
00339                           printf("Date may not have been set.\n");
00340                   }
00341                   printf("%s", anykey_str); mpxprompt_anykey();
00342           }
00343           return;
00344 }
00345
00347 void mpxcmd_exit (int argc, char *argv[]) {
00348           printf("\n");
00349           printf("Are you sure you want to terminate MPX?\n");
00350           if( mpxprompt_yn() ) {
00351                   printf("EXITING.\n");
00352                   //sys_exit();
00353                   sys_req(EXIT, NO_DEV, NULL, 0);
00354           }
00355 }
```

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 27 of file mpx_cmd.h.

4.5.1.2 #define MAX_LINE 1024

Definition at line 26 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.

< takes first word of user input as the command

< goes through the user input and takes each word as a token for later use as arguments in command

< searches the list for a matching command

Definition at line 74 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);
  mpx_add_command("goR3", mpxcmd_r3run);
  mpx_add_command("loadProc", loadProgram);
  mpx_add_command("terminate",terminateProcess);
  mpx_add_command("goR4", mpxcmd_gor4);

  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 263 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 347 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();
                sys_req(EXIT, NO_DEV, NULL, 0);
        }
}
```

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.

< opens the file

< goes to the end of the file

< finds out the size of the file

< returns to the beginning

< writes to the buffer the prints out

Definition at line 190 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.

< attempts to open the directory

< gets a file and puts the name in buf until none left

Definition at line 157 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 251 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 229 of file MPX_CMD.C.

```
                                                {
        mpx_cls();
        printf("\n");
```

```
        printf("  ============================================\n");
        printf("  = MPX System Version R6 - December  8, 2010 =\n");
        printf("  ============================================\n");
        printf("\n");
        printf("      by the members of PERPETUAL MOTION SQUAD:\n");
        printf("                         --------- ------ -----\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 Hayhurst\n");

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

4.6 src/mpx_cmd.h

```
00001 /*****************************************************************************
00002         MPX: The MultiProgramming eXecutive
00003         Project to Accompany
00004         A Practical Approach to Operating Systems
00005         Malcolm G. Lane & James D. Mooney
00006         Copyright 1993, P.W.S. Kent Publishing Co., Boston, MA.
00007
00008         File Name:      mpx_cmd.h
00009
00010         Author: Nathaniel Clay and Nicholas Yanak
00011         Version: 1.1
00012         Date:  12/9/2010
00013
00014         Purpose: Header for mpx_cmd.c
00015
00016
00017         Environment: Windows XP 32 bit
00018
00019 *****************************************************************************/
00020 #ifndef MPX_CMD_HFILE
00021 #define MPX_CMD_HFILE
00022
00023
00024 /* Symbolic Constants */
00025
00026 #define MAX_LINE        1024
00027 #define MAX_ARGS        10
00028
00029
00030 /* Types */
00031
00032 typedef struct mpx_cmd {
00033         char *cmd_name;
00034         struct mpx_cmd *next;
00035         void (*cmd_function)(int argc, char *argv[]);
00036 } mpx_cmd_t;
00037
00038
00039 /* Prototypes */
00040
00041 int             mpx_command_loop(void);
00042 void    mpxcmd_exit             (int argc, char *argv[]);
00043 void    mpxcmd_help             (int argc, char *argv[]);
00044 void    mpxcmd_load             (int argc, char *argv[]);
00045 void    mpxcmd_date             (int argc, char *argv[]);
```

```
00046 void    mpxcmd_version  (int argc, char *argv[]);
00047 void    mpxcmd_prompt   (void);
00048
00049
00050 #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 queue 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.

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

  This function preforms a deep copy of a PCB.

- 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 = NULL

  declaring null roots for initial start of linked lists

- ROOT ∗ wsQueue = NULL

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

< counter

< pointer to the Memory Descriptor

<pointer to the stack descriptor

< pointer to the stack low address

< checks to make sure everything is allocated

Definition at line 37 of file mpx_r2.c.

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


  // Allocate memory to each of the distinct 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;// 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* find_PCB (char * *name*)

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

Definition at line 312 of file mpx_r2.c.

```
                    {
```

```
        ELEM *incr;
        incr =  rQueue -> node; //set node to the first node in the priority queu
      e
        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  FIFO queue
        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;
        }

}
```

4.7.1.3 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 80 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
}
```

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

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

Definition at line 281 of file mpx_r2.c.

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


        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 queue
                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 queue t
      o node of Type Element
                queueROOT -> count += 1; // increase count by one
                return; //exit out first node is in queue.
        }


        /* INSERT INTO THE queue IN FIFO ORDER*/
        incr = queueROOT -> node; //set node to the first node in the queue
        while( incr -> right != NULL ){
                incr = incr -> right; // progress forward to the right of the que
      ue
        }
         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
      ue has grown by one

         return;

}
```

4.7.1.5 void insert_PCB (PCB * *PCBpointer*)

This function inserts a PCB into its aproprate PCB Queue.

< used to keep track of which queue the PCB belongs in

< counter which keeps track of how many times insert has ran

< checks for first call of insert and allocates mem

< if ready or running insert into priority order

< if blocked or suspended insert into first in first out

Definition at line 163 of file mpx_r2.c.

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

    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;
                };
        count++;//Update the number of times the function has run.
}
```

4.7.1.6 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 194 of file mpx_r2.c.

```
                                                { //FIXME: NO ERROR CHECKING

        ELEM *node; // declare node of type element
        ELEM *incr; // used to traverse queue
        ELEM *temp1; // used for temporary storage
        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 queue
                node -> left = NULL;
                node -> right = NULL;
                queueROOT -> node = node; // Set the first element in the queue t
      o node of Type Element
                queueROOT -> count += 1; // increase count by one
                return; //exit out first node is in queue.
        }

        incr = queueROOT -> node; //set node to the first node in the queue
        while ( incr -> process -> priority <= node -> process -> priority  ){ //
       Process with the lowest priority goes first
                        if( incr->right == NULL) break; // if the end is reaached
       quit
                        incr = incr -> right; // progrees to the right

        }

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

        /*head case*/
        // if new pcb has lower priority than head make it the new head else put
      it afterwards
        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 queueROOT to new head
                        queueROOT ->count +=1;
                }
                return;
        }
        if( incr -> left == NULL && incr->right != NULL ){ // sets it after incr
                node->left = NULL;
                node->right = incr;
                incr->left = node;
                queueROOT -> node = node; //set queueROOT to new head
                queueROOT ->count +=1;
                return;
        }

        /*tail case*/
        // if new pcb has higher priority make it the new tail
        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*/
        // left-incr-node-right
        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;
        }
}
```

4.7.1.7 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 466 of file mpx_r2.c.

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

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

                strcpy(name,argv[1]);

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

4.7.1.8 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 401 of file mpx_r2.c.

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

        PCB *newPCB = allocate_PCB();

        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);

}
```

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

This function preforms a deep copy of a PCB.

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

Definition at line 448 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;
        }
        }
}
```

4.7.1.10 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 550 of file mpx_r2.c.

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

                strcpy(name,argv[1]);

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

4.7.1.11 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 578 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;
                }
                tempPCB = find_PCB(name);
                if ( tempPCB != NULL){
                        tempPCB -> priority = priority;
                        if( tempPCB -> state == READY ){
                                        //tempPCB = copy_PCB(pointer);
                                        remove_PCB(tempPCB);
                                        insert_PCB(tempPCB);
                                        }
                }else{
                        printf("Process Name not found!");
                        mpxprompt_anykey();
                        return;
                }
        }
        else{
                printf("Wrong number of arguments used");
                mpxprompt_anykey();
                return;
        }
}
```

4.7.1.12 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 615 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;
        }
}
```

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

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

Definition at line 647 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 queue
                lp = &line;
                mpx_pager_init(" All PCB's In Queue:\n -------------------------
      ---------------------------\n");

                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
                }
             }
                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();
}
```

4.7.1.14 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 731 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 queue
                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 queue
                }
        }
        else{
                printf("Wrong number of arguments used");
                return;
        }
        mpxprompt_anykey();
}
```

4.7.1.15 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 692 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 queue
```

```
                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 queue
                }

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

4.7.1.16 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 523 of file mpx_r2.c.

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

                tempPCB = find_PCB(name);
                if ( tempPCB != NULL){
                        //tempPCB = copy_PCB(tempPCB);
                        remove_PCB(tempPCB);
                        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;
        }
}
```

4.7.1.17 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 495 of file mpx_r2.c.

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

                strcpy(name,argv[1]);

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

4.7.1.18 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 332 of file mpx_r2.c.

```
                                {
        ROOT *queue;
        ELEM *incr; // traverses queue
        ELEM *temp1; // used to hold left and right pcb
        ELEM *temp2;

        if( find_PCB( process-> name ) == NULL ){ //case where pcb is not in queu
      e
                free_PCB(process); //deallocate mem
                return; // return
        }

        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 queue
 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); //what will this do if incr is null

 return;
 }
```

4.7.1.19 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.

< return int 0 or 1

< initially set to return valid setup

< sets the name variable in pcb to the function input variable Name

< performs a search by name to find the pcb exits if none is found

< sets the setup to return a failed attempt if type is not 0 or 1

< checks to make sure state is a valid number and if not sets setup to return a failure

<checks that priority is within the valid range and has setup return failure if not

< returns failure or sucess of setup

Definition at line 105 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;
}
```

4.7.1.20 char* string_PCB (PCB * *pointer*)

This function returns a character string with PCB information formatted.

< becomes classType

< becomes stateType

< returns formatted string

Definition at line 138 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 = NULL

declaring null roots for initial start of linked lists

Definition at line 28 of file mpx_r2.c.

4.7.2.2 ROOT* wsQueue = NULL

Definition at line 29 of file mpx_r2.c.

4.8 src/mpx_r2.c

```
00001 /************************************************************************
00002         MPX: The MultiProgramming eXecutive
00003         Project to Accompany
00004         A Practical Approach to Operating Systems
00005         Malcolm G. Lane & James D. Mooney
00006         Copyright 1993, P.W.S. Kent Publishing Co., Boston, MA.
00007
00008         File Name:      mpx_r2.c
00009
00010         Author: Nathaniel Clay and Nicholas Yanak
00011         Version: 1.1
00012         Date:  12/9/2010
00013
00014         Purpose: Provides several supporting functions such as allocation and sea
      rch
00015         for PCB's
00016
00017
00018         Environment: Windows XP 32 bit
00019
00020 ************************************************************************/
00021 #include "mpx_r2.h"
00022 #include "mpx_supt.h"
00023 #include "mystdlib.h"
00024 #include "mpx_util.h"
00025 #include <string.h>
00026 #include <stdio.h>
00027
00028 ROOT *rQueue=NULL;
```

```
00029 ROOT *wsQueue=NULL;
00030
00031
00032
00033
00037 PCB *allocate_PCB( void ){
00038         PCB *newPCB;
00039         int i;
00040         MEMDSC *newMemDsc;
00041         STACKDSC *newStackDsc;
00042         unsigned char *stack;
00043
00044
00045         // Allocate memory to each of the distinct parts of the PCB
00046         newStackDsc = (STACKDSC*) sys_alloc_mem(sizeof(STACKDSC));
00047         newMemDsc = (MEMDSC*) sys_alloc_mem(sizeof(MEMDSC));
00048         newPCB = (PCB*) sys_alloc_mem(sizeof(PCB));
00049         stack = (unsigned char*) sys_alloc_mem(STACKSIZE*sizeof(unsigned char));
00050
00051         if ( stack == NULL ||
00052                  newStackDsc == NULL ||
00053                  newMemDsc == NULL ||
00054                  newPCB == NULL ) return NULL;
00055
00056         //Setup Memory Descriptor with Default Values for Module 2
00057         newMemDsc -> size = 0;
00058         newMemDsc -> loadADDR = NULL;
00059         newMemDsc -> execADDR = NULL;
00060
00061         //Setup the Stack
00062
00063         memset(stack,0,STACKSIZE*sizeof(unsigned char));//ZERO out Stack to aid i
      n debug....
00064         newStackDsc -> base = stack; // x86 arch Stacks start at the Higest value

00065         newStackDsc -> top  = stack + STACKSIZE;// and go to lowest or n - 2 for
      Word alloc
00066
00067         //Bundling Opereations of Stack Descripter Bellow
00068         newPCB -> stackdsc = newStackDsc;  // stack descriptor is placed in the P
      CB
00069
00070         //Bundling Operations of Memory Descriptor
00071         newPCB -> memdsc = newMemDsc; // memory descriptor is placed in the PCB
00072
00073         return newPCB;
00074
00075 }
00076
00080 int free_PCB( PCB *pointer /*< [in] is a pointer to a PCB  */ ){
00081         STACKDSC *stackdscptr = pointer -> stackdsc;
00082         unsigned char *stack = stackdscptr -> base;
00083         MEMDSC *memptr = pointer -> memdsc;
00084
00085         int err;
00086
00087         //Free Stack First
00088         err = sys_free_mem(stack);
00089         if( err < 0 ) return err;
00090         //Second free Stack Descriptor
00091         err = sys_free_mem(stackdscptr);
00092         if( err < 0 ) return err;
00093         //Third free Memory Descriptor
00094         err = sys_free_mem(memptr);
00095         if( err < 0 ) return err;
00096         //Finaly free Process Control block
00097         err = sys_free_mem(pointer);
```

```
00098         if(err < 0 ) return err;
00099
00100         return 0; //freed mem ok
00101 }
00102
00104 //FIXME: Move to allocate, Create to setup
00105 int setup_PCB( PCB *pointer, char *Name, int classType, int state, int priority )
      {//FIXME: NO DATA VV
00106         int ret;
00107         char *name = pointer -> name;
00108         ret = 0;
00109         strcpy(name,Name);
00110
00111         if( find_PCB(name) == NULL){
00112                 if( classType == 1 || classType == 0 ){
00113                         pointer -> classType = classType;
00114                 }else{
00115                         ret = 1;
00116                 }
00117                  if( state == BLOCKED ||
00118                          state == SUSPENDED_READY ||
00119                      state == SUSPENDED_BLOCKED ||
00120                          state == READY ||
00121                          state == RUNNING )
00122                 {
00123                 pointer -> state = state;
00124                 }else{
00125                         ret = 1;
00126                 }
00127                 if( priority <= 127 && priority >= -128){
00128                         pointer -> priority = priority;
00129                 }else{
00130                         ret = 1;
00131                 }
00132         }else{
00133                 ret = 1;
00134         }
00135         return ret;
00136 }
00138 char *string_PCB( PCB *pointer){
00139         char line_buf[MAX_LINE];
00140         char *name = pointer -> name;
00141         signed char *classType = pointer -> classType;
00142         signed char *stateType = pointer -> state;
00143         signed char *priority = pointer -> priority;
00144         char class[60];
00145         char state[60];
00146
00147         if( classType == APPLICATION ) strcpy( class, "Application");
00148         if( classType == SYSTEM ) strcpy( class, "System" );
00149
00150         if( stateType == RUNNING ) strcpy(state,"Running");
00151         if( stateType == READY ) strcpy( state ,"Ready" );
00152         if( stateType == BLOCKED ) strcpy( state ,"Blocked");
00153         if( stateType == SUSPENDED_READY ) strcpy(state ,"Suspended Ready");
00154         if ( stateType == SUSPENDED_BLOCKED ) strcpy( state,"Suspended Blocked" )
       ;
00155
00156
00157     sprintf(&line_buf,"Name: %s  Class: %s State: %s Priority: %d ", name, class,
       state,priority);
00158
00159         return line_buf;
00160 }
00161
00163 void insert_PCB(PCB *PCBpointer/*< pointer to a PCB to insert*/ ){
00164    int ORD;
```

```
00165    static int count;
00166         if( count == ZERO ){
00167                 rQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
00168                 wsQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
00169         }
00170
00171    if ( PCBpointer -> state == READY || PCBpointer -> state == RUNNING ){
00172                 ORD  = PORDR;
00173         }
00174         if( PCBpointer -> state == BLOCKED ||
00175                 PCBpointer -> state == SUSPENDED_READY ||
00176                 PCBpointer -> state == SUSPENDED_BLOCKED ){
00177                 ORD  = FIFO;
00178         }
00179
00180    switch(ORD){
00181                 case PORDR:
00182                         insert_PORDR(PCBpointer,rQueue);
00183                         break;
00184                 case FIFO:
00185                         insert_FIFO(PCBpointer,wsQueue);
00186                         break;
00187                 default:
00188                         //printf("ORDER not Valid");
00189                         break;
00190                 };
00191         count++;//Update the number of times the function has run.
00192 }
00194 void insert_PORDR( PCB *PCBpointer, ROOT *queueROOT ){ //FIXME: NO ERROR CHECKING

00195         ELEM *node; // declare node of type element
00196         ELEM *incr; // used to traverse queue
00197         ELEM *temp1; // used for temporary storage
00198         node = sys_alloc_mem( sizeof(ELEM)); // allocate Memory for node
00199         node -> process = PCBpointer;// add the PCB to the node
00200
00201         if( queueROOT -> node == NULL ){ // if this is the first element ever in
      the queue
00202                 node -> left = NULL;
00203                 node -> right = NULL;
00204                 queueROOT -> node = node; // Set the first element in the queue t
      o node of Type Element
00205                 queueROOT -> count += 1; // increase count by one
00206                 return; //exit out first node is in queue.
00207         }
00208
00209         incr = queueROOT -> node; //set node to the first node in the queue
00210         while ( incr -> process -> priority <= node -> process -> priority  ){ //
       Process with the lowest priority goes first
00211                         if( incr->right == NULL) break; // if the end is reaached
       quit
00212                         incr = incr -> right; // progrees to the right
00213
00214         }
00215
00216         /* There are three cases to check for head, tail, and middle*/
00217
00218         /*head case*/
00219         // if new pcb has lower priority than head make it the new head else put
      it afterwards
00220         if ( incr -> left == NULL && incr-> right == NULL){
00221                 if( incr -> process -> priority <= node -> process -> priority ){

00222                         node-> left = incr;
00223                         node-> right = NULL;
00224                         incr->right = node;
00225                         queueROOT->count +=1;
```

```
00226                 }else{
00227                         node->left = NULL;
00228                         node->right = incr;
00229                         incr->left = node;
00230                         queueROOT -> node = node; //set queueROOT to new head
00231                         queueROOT ->count +=1;
00232                 }
00233                 return;
00234         }
00235         if( incr -> left == NULL && incr->right != NULL ){ // sets it after incr
00236                 node->left = NULL;
00237                 node->right = incr;
00238                 incr->left = node;
00239                 queueROOT -> node = node; //set queueROOT to new head
00240                 queueROOT ->count +=1;
00241                 return;
00242         }
00243
00244         /*tail case*/
00245         // if new pcb has higher priority make it the new tail
00246         if( incr -> left != NULL && incr->right == NULL ){
00247
00248                 if( incr -> process -> priority <= node -> process -> priority ){

00249                         node-> left = incr;
00250                         node-> right = NULL;
00251                         incr->right = node;
00252                         queueROOT->count +=1;
00253                         return;
00254                 }else{
00255                         incr = incr -> left; //decrement incr
00256                         temp1 = incr -> right;
00257                         incr->right = node;
00258                         node->right = temp1;
00259                         node->left = incr;
00260                         temp1->left = node;
00261                         queueROOT->count +=1;
00262                         return;
00263                 }
00264
00265         }
00266
00267         /*middle case*/
00268         // left-incr-node-right
00269         if( incr -> left != NULL && incr->right != NULL){
00270                 incr = incr -> left;
00271                 temp1 = incr -> right;
00272                 incr->right = node;
00273                 node->right = temp1;
00274                 node->left = incr;
00275                 temp1->left = node;
00276                 queueROOT->count +=1;
00277                 return;
00278         }
00279 }
00281 void insert_FIFO( PCB *PCBpointer, ROOT *queueROOT){ //FIXME: NO ERROR HANDLING
00282         ELEM *node; // declare node of type element
00283         ELEM *incr; // traverses the queue
00284
00285
00286         node = sys_alloc_mem( sizeof(ELEM)); // allocate Memory for node
00287         node -> process = PCBpointer;// add the PCB to the node
00288
00289         if( queueROOT -> node == NULL ){ // if this is the first element ever in
     the queue
00290                 node -> left = NULL; // set the link left to null
00291                 node -> right = NULL;// set the link right to null
```

```
00292                 queueROOT -> node = node; // Set the first element in the queue t
      o node of Type Element
00293                 queueROOT -> count += 1; // increase count by one
00294                 return; //exit out first node is in queue.
00295         }
00296
00297
00298         /* INSERT INTO THE queue IN FIFO ORDER*/
00299         incr = queueROOT -> node; //set node to the first node in the queue
00300         while( incr -> right != NULL ){
00301                 incr = incr -> right; // progress forward to the right of the que
      ue
00302         }
00303          incr -> right = node;
00304          node -> left = incr; //set left to previous node
00305          node -> right = NULL; // set right to null
00306          queueROOT -> count += 1; // increase count by one as the size of the que
      ue has grown by one
00307
00308          return;
00309
00310 }
00312 PCB *find_PCB( char *name){
00313         ELEM *incr;
00314         incr =  rQueue -> node; //set node to the first node in the priority queu
      e
00315         while ( strcmp(name,incr -> process -> name ) != 0 && incr != NULL){ // P
      rocess with the lowest priority goes first
00316                         incr= incr -> right; // progrees to the right
00317         }
00318         if (incr == NULL ){
00319         incr =  wsQueue -> node; //set node to the first node in the  FIFO queue
00320         while ( strcmp(name,incr -> process -> name ) != 0 && incr != NULL){ // P
      rocess with the lowest priority goes first
00321                         incr= incr -> right; // progrees to the right
00322         }
00323         }
00324         if ( incr -> process != NULL && incr != NULL ){
00325                 return incr->process;
00326         }else{
00327                 return NULL;
00328         }
00329
00330 }
00332 void remove_PCB( PCB *process ){
00333         ROOT *queue;
00334         ELEM *incr; // traverses queue
00335         ELEM *temp1; // used to hold left and right pcb
00336         ELEM *temp2;
00337
00338         if( find_PCB( process-> name ) == NULL ){ //case where pcb is not in queu
      e
00339                 free_PCB(process); //deallocate mem
00340                 return; // return
00341         }
00342
00343         if ( process -> state == READY || process -> state == RUNNING ){
00344                 queue = rQueue;
00345         }
00346         if( process -> state == BLOCKED ||
00347                 process -> state == SUSPENDED_READY ||
00348                 process -> state == SUSPENDED_BLOCKED ){
00349                 queue = wsQueue;
00350         }
00351         /* last in queue */
00352         if ( queue -> count == 1 ){
00353                 incr = queue-> node;
```

```
00354                //       free_PCB(incr->process);
00355                 //sys_free_mem(queue->node);
00356                 queue -> node = NULL;
00357                 queue -> count -=1;
00358 
00359                 return;
00360         }
00361         incr = queue-> node; //set node to the first node in the queue
00362         while ( (incr -> process != process ) && incr != NULL ){ // find the same
      process
00363                         incr = incr -> right; // progrees to the right
00364         }
00365         /* There are three cases to check for head, tail, and middle*/
00366 
00367 
00368 
00369         /*head case*/
00370         if( incr -> left == NULL && incr->right != NULL ){
00371                 temp1 = incr -> right;
00372                 temp1 -> left  = NULL;
00373                 queue -> node = temp1; //set queueROOT to new head
00374                 queue ->count -=1;
00375         }
00376 
00377         /*tail case*/
00378         if( incr -> left != NULL && incr->right == NULL ){
00379                 temp1 = incr-> left;
00380                 temp1 -> right  = NULL;
00381                 queue -> count -=1;
00382         }
00383 
00384         /*middle case*/
00385         if( incr -> left != NULL && incr->right != NULL){
00386                 temp1 = incr -> left;
00387                 temp1 -> right = incr -> right;
00388                 temp2 = incr -> right;
00389                 temp2 -> left = incr -> left;
00390                 queue -> count -=1;
00391         }
00392         //Deallocate mem
00393         //free_PCB(process);
00394         //sys_free_mem(incr); //what will this do if incr is null
00395 
00396         return;
00397         }
00398 
00400 // it prompts the user for information then attempts to allocate and setup the pc
      b then insert in the queue
00401 void mpxcmd_create_PCB(int argc, char *argv[]){
00402         char name[STRLEN];
00403         char line[MAX_LINE];
00404         int type;
00405         int priority;
00406 
00407         PCB *newPCB = allocate_PCB();
00408 
00409         printf("Process Name: \n");
00410         mpx_readline(name, STRLEN);
00411         printf("Process Class Type ( Application 0 or System  1): \n" );
00412         type= mpxprompt_int();
00413         printf("Process Priority (-128 to 127): \n");
00414         priority = mpxprompt_int();
00415 
00416 
00417 
00418         if ( setup_PCB(newPCB,&name,type,READY,priority) == 1){
00419                 printf("Incrorrect information entered.");
```

```
00420                 mpxprompt_anykey();
00421                 return;
00422         }
00423
00424         insert_PCB(newPCB);
00425
00426 }
00427
00429 //PCB *copy_PCB(PCB *pointer){
00430 //              PCB *tempPCB = allocate_PCB();
00431 //              tempPCB -> state = pointer -> state;
00432 //              tempPCB -> classType = pointer -> classType;
00433 //              strcpy(tempPCB->name, pointer -> name);
00434 //              tempPCB -> priority = pointer ->priority;
00435 //
00436 //              // MEMDSC copy
00437 //              tempPCB -> memdsc -> size = pointer -> memdsc -> size;
00438 //              tempPCB -> memdsc -> loadADDR = pointer -> memdsc -> loadADDR;
00439 //              tempPCB -> memdsc -> execADDR = pointer -> memdsc -> execADDR;
00440 //
00441 //              //STACKDSC copy
00442 //              memcpy(tempPCB->stackdsc->base,pointer -> stackdsc -> base, STACK
      SIZE);
00443 //
00444 //      return tempPCB;
00445 //}
00446
00448 void mpxcmd_delete_PCB(int argc, char *argv[]){
00449         if (argc == 2){
00450         char name[STRLEN];
00451         PCB *pointer;
00452         strcpy(name,argv[1]);
00453
00454         pointer = find_PCB(name);
00455
00456         if ( pointer != NULL){
00457                 remove_PCB(pointer);
00458         }else{
00459                 printf("Process Name not found!");
00460                 return;
00461         }
00462         }
00463 }
00464
00466 void mpxcmd_block(int argc, char *argv[]){
00467         if(argc==2){
00468                 char name[STRLEN];
00469
00470                 PCB *pointer;
00471                 PCB *tempPCB;
00472                 int buffs = STRLEN;
00473
00474                 strcpy(name,argv[1]);
00475
00476                 tempPCB = find_PCB(name);
00477                 if ( tempPCB != NULL){
00478                         //tempPCB = copy_PCB(pointer);
00479                         remove_PCB(tempPCB);
00480                         if( tempPCB -> state == READY || tempPCB -> state ==
      RUNNING ) tempPCB -> state = BLOCKED;
00481                         if( tempPCB -> state == SUSPENDED_READY ) tempPCB -> stat
      e = SUSPENDED_BLOCKED;
00482                         insert_PCB(tempPCB);
00483                 }else{
00484                         printf("Process Name not found!");
00485                         return;
00486                 }
```

```
00487         }
00488         else{
00489                 printf("Wrong number of arguments used");
00490                 return;
00491         }
00492 }
00493
00495 void mpxcmd_unblock(int argc, char *argv[]){
00496         if(argc==2){
00497                 char name[STRLEN];
00498                 PCB *pointer;
00499                 PCB *tempPCB;
00500                 int buffs = STRLEN;
00501
00502                 strcpy(name,argv[1]);
00503
00504                 tempPCB = find_PCB(name);
00505                 if ( tempPCB != NULL){
00506                         //tempPCB = copy_PCB(pointer);
00507                         remove_PCB(tempPCB);
00508                         if( tempPCB -> state == BLOCKED ) tempPCB -> state = READ
      Y;
00509                         if( tempPCB -> state == SUSPENDED_BLOCKED ) tempPCB -> st
      ate = SUSPENDED_READY;
00510                         insert_PCB(tempPCB);
00511                 }else{
00512                         printf("Process Name not found!");
00513                         return;
00514                 }
00515         }
00516         else{
00517                 printf("Wrong number of arguments used");
00518                 return;
00519         }
00520 }
00521
00523 void mpxcmd_suspend(int argc, char *argv[]){
00524         if(argc==2){
00525                 char name[STRLEN];
00526                 PCB *pointer;
00527                 PCB *tempPCB;
00528                 int buffs = STRLEN;
00529                 strcpy(name,argv[1]);
00530
00531                 tempPCB = find_PCB(name);
00532                 if ( tempPCB != NULL){
00533                         //tempPCB = copy_PCB(tempPCB);
00534                         remove_PCB(tempPCB);
00535                         if( tempPCB -> state == READY || tempPCB -> state ==
      RUNNING ) tempPCB -> state = SUSPENDED_READY;
00536                         if( tempPCB -> state == BLOCKED ) tempPCB -> state = SUSP
      ENDED_BLOCKED;
00537                         insert_PCB(tempPCB);
00538                 }else{
00539                         printf("Process Name not found!");
00540                         return;
00541                 }
00542         }
00543         else{
00544                 printf("Wrong number of arguments used");
00545                 return;
00546         }
00547 }
00548
00550 void mpxcmd_resume(int argc, char *argv[]){
00551         if(argc==2){
00552                 char name[STRLEN];
```

```
00553                 PCB *pointer;
00554                 PCB *tempPCB;
00555                 int buffs = STRLEN;
00556 
00557                 strcpy(name,argv[1]);
00558 
00559                 tempPCB = find_PCB(name);
00560                 if ( pointer != NULL){
00561                         //tempPCB = copy_PCB(pointer);
00562                         remove_PCB(tempPCB);
00563                         if( tempPCB -> state == SUSPENDED_READY ) tempPCB -> stat
      e = READY;
00564                         if( tempPCB -> state == SUSPENDED_BLOCKED ) tempPCB -> st
      ate = BLOCKED;
00565                         insert_PCB(tempPCB);
00566                 }else{
00567                         printf("Process Name not found!");
00568                         return;
00569                 }
00570         }
00571         else{
00572                 printf("Wrong number of arguments used");
00573                 return;
00574         }
00575 }
00576 
00578 void mpxcmd_setPriority(int argc, char *argv[]){
00579         if(argc==3){
00580                 char name[STRLEN];
00581                 PCB *pointer;
00582                 int priority;
00583                 PCB *tempPCB;
00584                 int buffs = STRLEN;
00585                 priority  = atoi(argv[2]);
00586                 strcpy(name,argv[1]);
00587                 if( priority <= 128 || priority >= -127){ ;}else{
00588                         printf("Number entered out of range!");
00589                         mpxprompt_anykey();
00590                         return;
00591                 }
00592                 tempPCB = find_PCB(name);
00593                 if ( tempPCB != NULL){
00594                         tempPCB -> priority = priority;
00595                         if( tempPCB -> state == READY ){
00596                                         //tempPCB = copy_PCB(pointer);
00597                                         remove_PCB(tempPCB);
00598                                         insert_PCB(tempPCB);
00599                                         }
00600                 }else{
00601                         printf("Process Name not found!");
00602                         mpxprompt_anykey();
00603                         return;
00604                 }
00605         }
00606         else{
00607                 printf("Wrong number of arguments used");
00608                 mpxprompt_anykey();
00609                 return;
00610         }
00611 }
00612 
00613 
00615 void mpxcmd_show_PCB(int argc, char *argv[]){
00616         if(argc==2){
00617                 char name[STRLEN];
00618                 PCB *pointer;
00619                 char class[30];
```

```
00620                   char state[45];
00621                   int buffs = STRLEN;
00622                   char line[MAX_LINE];
00623                   char* lp;
00624                   lp = &line;
00625 
00626                   strcpy(name,argv[1]);
00627 
00628                   pointer = find_PCB(name);
00629 
00630                   if ( pointer != NULL){
00631                           printf("%s\n",string_PCB(pointer));
00632                           mpxprompt_anykey();
00633                   }else{
00634                           printf("Process Name not found!");
00635                           mpxprompt_anykey();
00636                           return;
00637                   }
00638           }
00639           else{
00640                   printf("Wrong number of arguments used");
00641                   mpxprompt_anykey();
00642                   return;
00643           }
00644 }
00645 
00647 void mpxcmd_showAll_PCB(int argc, char *argv[]){ // Pagination function needs add
      ed !!Function still needs work!!
00648           if(argc==1){
00649                   ELEM *incr;
00650                   PCB *pointer;
00651                   char line[MAX_LINE];
00652                   char* lp;
00653                   char class[30];
00654                   char state[45];
00655                   //set node to the first node in the queue
00656                   lp = &line;
00657                   mpx_pager_init(" All PCB's In Queue:\n ------------------------
      ---------------------------\n");
00658 
00659                   if( rQueue -> count > 0 ){
00660                   incr = rQueue -> node;
00661                   while( incr != NULL ){
00662 
00663                           pointer = incr -> process;
00664 
00665                           lp = string_PCB(pointer);
00666                           mpx_pager(lp);
00667 
00668                           incr = incr -> right; // progress forward to the right of
       the queue
00669                   }
00670               }
00671                   if(wsQueue -> count > 0){
00672                   incr = wsQueue -> node;
00673                   while( incr != NULL ){
00674                           pointer = incr -> process;
00675 
00676 
00677                           lp = string_PCB(pointer);
00678                           mpx_pager(lp);
00679 
00680                           incr = incr -> right; // progress forward to the right of
       the queue
00681                   }
00682                   }
00683           }
```

```
00684         else{
00685                 printf("Wrong number of arguments used");
00686                 return;
00687         }
00688         mpxprompt_anykey();
00689 }
00690
00692 void mpxcmd_showReady_PCB(int argc, char *argv[]){ // Pagination function needs a
      dded !!Function still needs work!!
00693         if(argc==1){
00694                 ELEM *incr;
00695                 PCB *pointer;
00696                 char line[MAX_LINE];
00697                 char* lp;
00698                 char class[30];
00699                 char state[45];
00700                 incr = rQueue -> node;//set node to the first node in the queue
00701                 lp = &line;
00702                 mpx_pager_init(" All PCB's Ready State in Queues:\n ------------
      ----------------------------------------\n");
00703                 while( incr != NULL ){
00704
00705                         pointer = incr -> process;
00706                         if ( pointer -> state == READY){
00707                         lp = string_PCB(pointer);
00708                         mpx_pager(lp);
00709                         }
00710                         incr = incr -> right; // progress forward to the right of
       the queue
00711                 }
00712
00713                 incr = wsQueue -> node;//set node to the first node in the queue
00714                 while( incr != NULL ){
00715                         pointer = incr -> process;
00716                         if ( pointer -> state == SUSPENDED_READY){
00717                         lp = string_PCB(pointer);
00718                         mpx_pager(lp);
00719                         }
00720                         incr = incr -> right; // progress forward to the right of
       the queue                        incr = incr -> right; // progress forward to the
      right of the queue
00721                 }
00722         }
00723         else{
00724                 printf("Wrong number of arguments used");
00725                 return;
00726         }
00727         mpxprompt_anykey();
00728 }
00729
00731 void mpxcmd_showBlocked_PCB(int argc, char *argv[]){ // Pagination function needs
       added !!Function still needs work!!
00732         if(argc==1){
00733                 ELEM *incr;
00734                 PCB *pointer;
00735                 char line[MAX_LINE];
00736                 char* lp;
00737                 char class[30];
00738                 char state[45];
00739                 lp = &line;
00740                 mpx_pager_init(" All PCB's Blocked State in Queues:\n -----------
      ----------------------------------------\n");
00741
00742                 incr = wsQueue -> node;//set node to the first node in the queue
00743                 while( incr  != NULL ){
00744                         pointer = incr -> process;
00745                         if ( pointer -> state == SUSPENDED_BLOCKED || pointer ->
```

```
      state == BLOCKED ){
00746                         lp = string_PCB(pointer);
00747                         mpx_pager(lp);
00748                         }
00749                         incr = incr -> right; // progress forward to the right of
       the queue                        incr = incr -> right; // progress forward to the
      right of the queue
00750                 }
00751         }
00752         else{
00753                 printf("Wrong number of arguments used");
00754                 return;
00755         }
00756         mpxprompt_anykey();
00757 }
00758
00759
```

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 * 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.

- 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 queue 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 function preforms a deep copy of a PCB.

- 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.

- ROOT ∗ getRQueue ()
- ROOT ∗ getWSQueue ()
- void setRQueue (ROOT ∗root)
- void setWSQueue (ROOT ∗root)

4.9.1 Define Documentation

4.9.1.1 #define APPLICATION 0

is defined as 0

Definition at line 31 of file mpx_r2.h.

4.9.1.2 #define BLOCKED 2

state is defined as 2

Definition at line 25 of file mpx_r2.h.

4.9.1.3 #define FIFO 0

is the First In First Out Order flag

Definition at line 37 of file mpx_r2.h.

4.9.1.4 #define MAX_LINE 1024

Definition at line 40 of file mpx_r2.h.

4.9.1.5 #define PORDR 1

is the Priority Order flag

Definition at line 36 of file mpx_r2.h.

4.9.1.6 #define READY 1

state is Defined as 1

Definition at line 24 of file mpx_r2.h.

4.9.1.7 #define RUNNING 0

state is Defined as 0

Definition at line 23 of file mpx_r2.h.

4.9.1.8 #define STACKSIZE 1024

is the size of the stack in Bytes

Definition at line 33 of file mpx_r2.h.

4.9.1.9 #define STRLEN 16

is the length of a string for name

Definition at line 34 of file mpx_r2.h.

4.9.1.10 #define SUSPENDED_BLOCKED 4

is defined by 4

Definition at line 28 of file mpx_r2.h.

4.9.1.11 #define SUSPENDED_READY 3

is defined by 3

Definition at line 27 of file mpx_r2.h.

4.9.1.12 #define SYSTEM 1

is defined as 1

Definition at line 30 of file mpx_r2.h.

4.9.1.13 #define ZERO 0

Definition at line 38 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* 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

< counter

< pointer to the Memory Descriptor

<pointer to the stack descriptor

< pointer to the stack low address

< checks to make sure everything is allocated

Definition at line 37 of file mpx_r2.c.

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


// Allocate memory to each of the distinct 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;// 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;

}
```

4.9.3.2 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 312 of file mpx_r2.c.

```
                        {
        ELEM *incr;
        incr =  rQueue -> node; //set node to the first node in the priority queu
      e
        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  FIFO queue
        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 80 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 ROOT* getRQueue ( )

4.9.3.5 ROOT* getWSQueue ( )

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

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

Definition at line 281 of file mpx_r2.c.

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


        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 queue
                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 queue t
      o node of Type Element
                queueROOT -> count += 1; // increase count by one
                return; //exit out first node is in queue.
        }


        /* INSERT INTO THE queue IN FIFO ORDER*/
        incr = queueROOT -> node; //set node to the first node in the queue
        while( incr -> right != NULL ){
```

```
                incr = incr -> right; // progress forward to the right of the que
      ue
        }
         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
      ue has grown by one

         return;

}
```

4.9.3.7 void insert_PCB (PCB * *PCBpointer*)

This function inserts a PCB into its aproprate PCB Queue.

< used to keep track of which queue the PCB belongs in

< counter which keeps track of how many times insert has ran

< checks for first call of insert and allocates mem

< if ready or running insert into priority order

< if blocked or suspended insert into first in first out

Definition at line 163 of file mpx_r2.c.

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

   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;
                };
        count++;//Update the number of times the function has run.
}
```

4.9.3.8 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 194 of file mpx_r2.c.

```
                                                { //FIXME: NO ERROR CHECKING

  ELEM *node; // declare node of type element
  ELEM *incr; // used to traverse queue
  ELEM *temp1; // used for temporary storage
  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 queue
          node -> left = NULL;
          node -> right = NULL;
          queueROOT -> node = node; // Set the first element in the queue t
o node of Type Element
          queueROOT -> count += 1; // increase count by one
          return; //exit out first node is in queue.
  }

  incr = queueROOT -> node; //set node to the first node in the queue
  while ( incr -> process -> priority <= node -> process -> priority  ){ //
 Process with the lowest priority goes first
                  if( incr->right == NULL) break; // if the end is reaached
 quit
                  incr = incr -> right; // progrees to the right

  }

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

  /*head case*/
  // if new pcb has lower priority than head make it the new head else put
it afterwards
  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 queueROOT to new head
                  queueROOT ->count +=1;
          }
          return;
  }
  if( incr -> left == NULL && incr->right != NULL ){ // sets it after incr
          node->left = NULL;
          node->right = incr;
          incr->left = node;
          queueROOT -> node = node; //set queueROOT to new head
          queueROOT ->count +=1;
          return;
  }

  /*tail case*/
  // if new pcb has higher priority make it the new tail
  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*/
        // left-incr-node-right
        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;
        }
}
```

4.9.3.9 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 466 of file mpx_r2.c.

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

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

                strcpy(name,argv[1]);

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

4.9.3.10 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 401 of file mpx_r2.c.

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

        PCB *newPCB = allocate_PCB();

        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);

}
```

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

This function preforms a deep copy of a PCB.

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

Definition at line 448 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;
        }
        }
}
```

4.9.3.12 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 550 of file mpx_r2.c.

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

                strcpy(name,argv[1]);

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

4.9.3.13 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 578 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;
        }
        tempPCB = find_PCB(name);
        if ( tempPCB != NULL){
                tempPCB -> priority = priority;
                if( tempPCB -> state == READY ){
                        //tempPCB = copy_PCB(pointer);
                        remove_PCB(tempPCB);
                        insert_PCB(tempPCB);
                        }
        }else{
                printf("Process Name not found!");
                mpxprompt_anykey();
                return;
        }
    }
    else{
        printf("Wrong number of arguments used");
        mpxprompt_anykey();
        return;
    }
}
```

4.9.3.14 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 615 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;
    }
}
```

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

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

Definition at line 647 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 queue
                lp = &line;
                mpx_pager_init(" All PCB's In Queue:\n -------------------------
      ---------------------------\n");

                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
                }
             }
                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();
}
```

4.9.3.16 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 731 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 queue
                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 queue
                }
          }
          else{
                printf("Wrong number of arguments used");
                return;
          }
          mpxprompt_anykey();
}
```

4.9.3.17 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 692 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 queue
                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 queue
                }

                incr = wsQueue -> node;//set node to the first node in the queue
                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 queue                      incr = incr -> right; // progress forward to the
```

```
        right of the queue
                }
        }
        else{
                printf("Wrong number of arguments used");
                return;
        }
        mpxprompt_anykey();
}
```

4.9.3.18 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 523 of file mpx_r2.c.

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

                tempPCB = find_PCB(name);
                if ( tempPCB != NULL){
                        //tempPCB = copy_PCB(tempPCB);
                        remove_PCB(tempPCB);
                        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;
        }
}
```

4.9.3.19 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 495 of file mpx_r2.c.

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

                strcpy(name,argv[1]);
```

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

4.9.3.20 void setRQueue (ROOT * *root*)

4.9.3.21 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.

< return int 0 or 1

< initially set to return valid setup

< sets the name variable in pcb to the function input variable Name

< performs a search by name to find the pcb exits if none is found

< sets the setup to return a failed attempt if type is not 0 or 1

< checks to make sure state is a valid number and if not sets setup to return a failure

<checks that priority is within the valid range and has setup return failure if not

< returns failure or sucess of setup

Definition at line 105 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;
}
```

4.9.3.22 void setWSQueue (ROOT * *root*)

4.10 src/mpx_r2.h

```
00001 /**************************************************************************
00002         MPX: The MultiProgramming eXecutive
00003         Project to Accompany
00004         A Practical Approach to Operating Systems
00005         Malcolm G. Lane & James D. Mooney
00006         Copyright 1993, P.W.S. Kent Publishing Co., Boston, MA.
00007
00008         File Name:      mpx_r2.h
00009
00010         Author: Nathaniel Clay and Nicholas Yanak
00011         Version: 1.1
00012         Date:  12/9/2010
00013
00014         Purpose: This is the header for r2
00015
00016
00017         Environment: Windows XP 32 bit
00018
00019 **************************************************************************/
00020 #ifndef MPX_R2_HFILE
00021 #define MPX_R2_HFILE
00022 /* Symbolic Constants */
00023 #define RUNNING  0///< state is Defined as 0
00024 #define READY   1///< state is Defined as  1
00025 #define BLOCKED 2 ///< state is defined as 2
00026
00027 #define SUSPENDED_READY    3 ///< is defined by 3
00028 #define SUSPENDED_BLOCKED  4 ///< is defined by 4
00029
00030 #define SYSTEM  1 ///< is defined as 1
00031 #define APPLICATION  0  ///< is defined as 0
00032
00033 #define STACKSIZE   1024 ///< is the size of the stack in Bytes
00034 #define STRLEN    16///< is the length of a string for name
00035
00036 #define PORDR  1 ///< is the Priority Order flag
00037 #define FIFO   0 ///< is the First In First Out Order flag
00038 #define ZERO   0
00039
00040 #define MAX_LINE        1024
00041
00042 /* Type Definitions and Structures */
00043 typedef struct mem{
00044         int size;
00045         unsigned char *loadADDR;
```

```
00046         unsigned char *execADDR;
00047 }MEMDSC;
00048
00049 typedef struct stack{
00050         unsigned char *top;
00051         unsigned char *base;
00052 }STACKDSC;
00053
00054 typedef struct process{
00055         char name[STRLEN];
00056         int classType;
00057         int priority;
00058         int state;
00059         MEMDSC *memdsc;
00060         STACKDSC *stackdsc;
00061 }PCB;
00062
00063 typedef struct page{
00064         PCB *process;
00065         struct page *left;
00066         struct page *right;
00067 }ELEM;
00068
00069 typedef struct root{
00070         int count;
00071         ELEM *node;
00072 }ROOT;
00073
00074 /* Functions Dec*/
00075 PCB *allocate_PCB(void);
00076 int free_PCB( PCB *pointer);
00077 int setup_PCB( PCB *pointer, char *name, int classType, int state, int priority )
      ;
00078 void insert_PCB(PCB *PCBpointer/*< pointer to a PCB to insert*/ );
00079 void insert_PORDR( PCB *PCBpointer, ROOT *quequeROOT );
00080 void insert_FIFO( PCB *PCBpointer, ROOT *quequeROOT);
00081 PCB *find_PCB( char *name);
00082 void mpxcmd_create_PCB(int argc, char *argv[]);
00083 void mpxcmd_delete_PCB(int argc, char *argv[]);
00084 void mpxcmd_block(int argc, char *argv[]);
00085 void mpxcmd_unblock(int argc, char *argv[]);
00086 void mpxcmd_suspend(int argc, char *argv[]);
00087 void mpxcmd_resume(int argc, char *argv[]);
00088 void mpxcmd_setPriority(int argc, char *argv[]);
00089 void mpxcmd_show_PCB(int argc, char *argv[]);
00090 void mpxcmd_showAll_PCB(int argc, char *argv[]);
00091 void mpxcmd_showReady_PCB(int argc, char *argv[]);
00092 void mpxcmd_showBlocked_PCB(int argc, char *argv[]);
00093 ROOT * getRQueue();
00094 ROOT * getWSQueue();
00095 void setRQueue(ROOT *root );
00096 void setWSQueue(ROOT *root);
00097 #endif
```

4.11 src/mpx_r3.c File Reference

```
#include "dos.h"
#include "mpx_cmd.h"
#include "mpx_util.h"
#include "mpx_r2.h"
#include "mpx_r3.h"
```

```
#include "procs-r3.c"
#include "mpx_supt.h"
#include "mystdlib.h"
#include <string.h>
#include <stdio.h>
```

Functions

- PCB * getHead_PCB ()
- void interrupt sys_call (void)
- void interrupt dispatch (void)
- void mpxcmd_r3run (int argc, char *argv[ ])
- void mpxcmd_gor4 (int argc, char *argv[ ])

Variables

- PCB * cop
- PCB * HEAD
- ELEM * TEMP
- ROOT * Root
- STACKDSC * STACK
- tcontext * context_p
- tparams * param_p
- ROOT * rQueue

  declaring null roots for initial start of linked lists

- ROOT * wsQueue
- unsigned char sys_stack [SYS_STACK_SIZE]
- unsigned short ss_save = NULL
- unsigned short sp_save = NULL
- unsigned short new_ss = NULL
- unsigned short new_sp = NULL

4.11.1 Function Documentation

4.11.1.1 void interrupt dispatch (void)

Definition at line 117 of file mpx_r3.c.

```
                        {

  if ( sp_save == NULL ){ //saves the SS and SP from being overwritten if n
ot already done
          ss_save = _SS;
          sp_save = _SP;
          }
          HEAD = getHead_PCB();
          //STACK = HEAD -> stackdsc;

          // get a process from the ready queue then set the ss and sp to e
```

```
      xecute the new process
                if ( HEAD != NULL ){
                        cop = HEAD;
                        cop -> state = READY;
                        remove_PCB(HEAD);
                        STACK = cop -> stackdsc;
                        new_ss = FP_SEG(STACK -> top);
                        new_sp = FP_OFF(STACK -> top );
                        _SS = new_ss;
                        _SP = new_sp;
                }else{ // if no process left return
                        cop = NULL;
                        _SS = ss_save;
                        _SP = sp_save;
                        ss_save = NULL;
                        sp_save = NULL;
                }
                //_iret;
}
```

4.11.1.2 PCB * getHead_PCB ( )

Definition at line 147 of file mpx_r3.c.

```
                  {
                ELEM *incr;
                PCB  *pointer= NULL;

                incr = rQueue -> node;//set node to the first node in the queque
                if( incr  != NULL ){
                        pointer = incr -> process;
                        incr = incr -> right; // progress forward to the right of
       the queque                       incr = incr -> right; // progress forward to the
      right of the queque
                }
                return pointer;
}
```

4.11.1.3 void mpxcmd_gor4 (int *argc*, char * *argv[]*)

Definition at line 266 of file mpx_r3.c.

```
                                        {
        mpx_cls();
        dispatch();
        mpxprompt_anykey();
}
```

4.11.1.4 void mpxcmd_r3run (int *argc*, char * *argv[]*)

Definition at line 160 of file mpx_r3.c.

```
                                        {


        PCB *test1;
        PCB *test2;
```

```
PCB *test3;
PCB *test4;
PCB *test5;

STACKDSC *stack1;
STACKDSC *stack2;
STACKDSC *stack3;
STACKDSC *stack4;
STACKDSC *stack5;

tcontext *context1;
tcontext *context2;
tcontext *context3;
tcontext *context4;
tcontext *context5;

char name1[10] = "test1";
char name2[10] = "test2";
char name3[10] = "test3";
char name4[10] = "test4";
char name5[10] = "test5";



test1 = allocate_PCB();
test2 = allocate_PCB();
test3 = allocate_PCB();
test4 = allocate_PCB();
test5 = allocate_PCB();

stack1 = test1 -> stackdsc;
stack2 = test2 -> stackdsc;
stack3 = test3 -> stackdsc;
stack4 = test4 -> stackdsc;
stack5 = test5 -> stackdsc;

stack1 -> top = stack1 -> base + STACKSIZE - sizeof(tcontext);
stack2 -> top = stack2 -> base + STACKSIZE - sizeof(tcontext);
stack3 -> top = stack3 -> base + STACKSIZE - sizeof(tcontext);
stack4 -> top = stack4 -> base + STACKSIZE - sizeof(tcontext);
stack5 -> top = stack5 -> base + STACKSIZE - sizeof(tcontext);

context1 = (tcontext*) stack1 -> top;
context2 = (tcontext*) stack2 -> top;
context3 = (tcontext*) stack3 -> top;
context4 = (tcontext*) stack4 -> top;
context5 = (tcontext*) stack5 -> top;



context1->DS = _DS;
context1->ES = _ES;
context1->CS = FP_SEG(&test1_R3);
context1->IP = FP_OFF(&test1_R3);
context1->FLAGS = 0x200;

context2->DS = _DS;
context2->ES = _ES;
context2->CS = FP_SEG(&test2_R3);
context2->IP = FP_OFF(&test2_R3);
context2->FLAGS = 0x200;

context3->DS = _DS;
context3->ES = _ES;
context3->CS = FP_SEG(&test3_R3);
context3->IP = FP_OFF(&test3_R3);
context3->FLAGS = 0x200;
```

```
        context4->DS = _DS;
        context4->ES = _ES;
        context4->CS = FP_SEG(&test4_R3);
        context4->IP = FP_OFF(&test4_R3);
        context4->FLAGS = 0x200;

        context5->DS = _DS;
        context5->ES = _ES;
        context5->CS = FP_SEG(&test5_R3);
        context5->IP = FP_OFF(&test5_R3);
        context5->FLAGS = 0x200;
        context5->DS = _DS;

        setup_PCB(test1,name1,APPLICATION,READY, 1);
        setup_PCB(test2,name2,APPLICATION,READY, 2);
        setup_PCB(test3,name3,APPLICATION,READY, 3);
        setup_PCB(test4,name4,APPLICATION,READY, 4);
        setup_PCB(test5,name5,APPLICATION,READY, 5);

        insert_PCB(test1);
        insert_PCB(test2);
        insert_PCB(test3);
        insert_PCB(test4);
        insert_PCB(test5);


        mpx_cls();
        dispatch();
        mpxprompt_anykey();

}
```

4.11.1.5 void interrupt sys_call (void)

Definition at line 59 of file mpx_r3.c.

```
                                {


        cop-> stackdsc -> top = (unsigned char *) MK_FP(_SS, _SP);
        param_p = ( tparams*)(sizeof(tcontext) + cop -> stackdsc -> top);//code s
      upplied by GA bryan
        context_p = (tcontext*) cop -> stackdsc -> top;
        //SWITCH TO TEMP STACK by storing all of your variables in memory

        //replaces SS and SP
        new_ss = FP_SEG(sys_stack);
        new_sp = FP_OFF(sys_stack);
        new_sp += SYS_STACK_SIZE;
        _SS = new_ss;
        _SP = new_sp;

        // if the idle opcode is sent then change staate to ready and insert into
       queue
        if ( param_p -> op_code == IDLE ){
                        cop -> state = READY;
                        insert_PCB(cop);
                        cop = NULL;
        }

        // if the exit opcode is sent then remove and free the pcb
        if( param_p -> op_code == EXIT ){
                remove_PCB(cop);
```

```
                free_PCB(cop);
                cop = NULL;
        }

        // if ( param_p -> op_code == READ ){
                // if( param_p -> device_id == TERMINAL ){

                // }
                // if ( param_p -> device_id == COM_PORT ){

                // }
        // }
        // if ( param_p -> op_code == WRITE ){
                // if( param_p -> device_id == TERMINAL ){

                // }
                // if( param_p -> device_id == COM_PORT ){

                // }

        // }



        // FIXME NO ERROR CHECKING
        //context_p -> AX = 0;

        //CALL TO DISPATCHER
        dispatch();

}
```

4.11.2 Variable Documentation

4.11.2.1 tcontext * context_p

Definition at line 40 of file mpx_r3.c.

4.11.2.2 PCB* cop

Definition at line 35 of file mpx_r3.c.

4.11.2.3 PCB* HEAD

Definition at line 36 of file mpx_r3.c.

4.11.2.4 unsigned short new_sp = NULL

Definition at line 51 of file mpx_r3.c.

4.11.2.5 unsigned short new_ss = NULL

Definition at line 50 of file mpx_r3.c.

4.11.2.6 tparams * param_p

Definition at line 41 of file mpx_r3.c.

4.11.2.7 ROOT* Root

Definition at line 38 of file mpx_r3.c.

4.11.2.8 ROOT* rQueue

declaring null roots for initial start of linked lists

Definition at line 28 of file mpx_r2.c.

4.11.2.9 unsigned short sp_save = NULL

Definition at line 49 of file mpx_r3.c.

4.11.2.10 unsigned short ss_save = NULL

Definition at line 48 of file mpx_r3.c.

4.11.2.11 STACKDSC* STACK

Definition at line 39 of file mpx_r3.c.

4.11.2.12 unsigned char sys_stack[SYS_STACK_SIZE]

Definition at line 47 of file mpx_r3.c.

4.11.2.13 ELEM* TEMP

Definition at line 37 of file mpx_r3.c.

4.11.2.14 ROOT * wsQueue

Definition at line 29 of file mpx_r2.c.

4.12 src/mpx_r3.c

```
00001 /**************************************************************************
00002         MPX: The MultiProgramming eXecutive
00003         Project to Accompany
00004         A Practical Approach to Operating Systems
00005         Malcolm G. Lane & James D. Mooney
00006         Copyright 1993, P.W.S. Kent Publishing Co., Boston, MA.
00007
00008         File Name:      mpx_cmd.c
00009
```

```
00010         Author: Nathaniel Clay and Nicholas Yanak
00011         Version: 1.1
00012         Date:  12/9/2010
00013
00014         Purpose: Contains interrupt and dispatcher used for process and I/O
00015         management.
00016
00017
00018         Environment: Windows XP 32 bit
00019
00020 ************************************************************************/
00021
00022 #include "dos.h"
00023 #include "mpx_cmd.h"
00024 #include "mpx_util.h"
00025 #include "mpx_r2.h"
00026 #include "mpx_r3.h"
00027 #include "procs-r3.c"
00028 #include "mpx_supt.h"
00029 #include "mystdlib.h"
00030 #include <string.h>
00031 #include <stdio.h>
00032
00033 PCB *getHead_PCB();
00034
00035 PCB *cop;
00036 PCB *HEAD;
00037 ELEM *TEMP;
00038 ROOT *Root;
00039 STACKDSC *STACK;
00040 tcontext *context_p;
00041 tparams  *param_p;
00042 //IOCB termCB;
00043 //IOCB comCB;
00044
00045 extern ROOT *rQueue, *wsQueue; //link in the values for these in r2
00046
00047 unsigned char sys_stack[SYS_STACK_SIZE];
00048 unsigned short ss_save = NULL;
00049 unsigned short sp_save = NULL;
00050 unsigned short new_ss = NULL;
00051 unsigned short new_sp = NULL;
00052 tcontext *context_p;
00053 tparams  *param_p;
00054 //IOCB termCB;
00055 //IOCB comCB;
00056
00057
00058
00059 void interrupt sys_call(void){
00060
00061
00062         cop-> stackdsc -> top = (unsigned char *) MK_FP(_SS, _SP);
00063         param_p = ( tparams*)(sizeof(tcontext) + cop -> stackdsc -> top);//code s
      upplied by GA bryan
00064         context_p = (tcontext*) cop -> stackdsc -> top;
00065         //SWITCH TO TEMP STACK by storing all of your variables in memory
00066
00067         //replaces SS and SP
00068         new_ss = FP_SEG(sys_stack);
00069         new_sp = FP_OFF(sys_stack);
00070         new_sp += SYS_STACK_SIZE;
00071         _SS = new_ss;
00072         _SP = new_sp;
00073
00074         // if the idle opcode is sent then change staate to ready and insert into
       queue
```

```
00075         if ( param_p -> op_code == IDLE ){
00076                         cop -> state = READY;
00077                         insert_PCB(cop);
00078                         cop = NULL;
00079         }
00080
00081         // if the exit opcode is sent then remove and free the pcb
00082         if( param_p -> op_code == EXIT ){
00083                 remove_PCB(cop);
00084                 free_PCB(cop);
00085                 cop = NULL;
00086         }
00087
00088         // if ( param_p -> op_code == READ ){
00089                 // if( param_p -> device_id == TERMINAL ){
00090
00091                 // }
00092                 // if ( param_p -> device_id == COM_PORT ){
00093
00094                 // }
00095         // }
00096         // if ( param_p -> op_code == WRITE ){
00097                 // if( param_p -> device_id == TERMINAL ){
00098
00099                 // }
00100                 // if( param_p -> device_id == COM_PORT ){
00101
00102                 // }
00103
00104         // }
00105
00106
00107
00108
00109         // FIXME NO ERROR CHECKING
00110         //context_p -> AX = 0;
00111
00112         //CALL TO DISPATCHER
00113         dispatch();
00114
00115 }
00116
00117 void interrupt dispatch(void){
00118
00119         if ( sp_save == NULL ){ //saves the SS and SP from being overwritten if n
      ot already done
00120                 ss_save = _SS;
00121                 sp_save = _SP;
00122                 }
00123                 HEAD = getHead_PCB();
00124                 //STACK = HEAD -> stackdsc;
00125
00126                 // get a process from the ready queue then set the ss and sp to e
      xecute the new process
00127                 if ( HEAD != NULL ){
00128                         cop = HEAD;
00129                         cop -> state = READY;
00130                         remove_PCB(HEAD);
00131                         STACK = cop -> stackdsc;
00132                         new_ss = FP_SEG(STACK -> top);
00133                         new_sp = FP_OFF(STACK -> top );
00134                         _SS = new_ss;
00135                         _SP = new_sp;
00136                 }else{ // if no process left return
00137                         cop = NULL;
00138                         _SS = ss_save;
00139                         _SP = sp_save;
```

```
00140                           ss_save = NULL;
00141                           sp_save = NULL;
00142                   }
00143                   //_iret;
00144 }
00145
00146         // returns the head pcb of the ready queue
00147 PCB *getHead_PCB(){
00148                   ELEM *incr;
00149                   PCB  *pointer= NULL;
00150
00151                   incr = rQueue -> node;//set node to the first node in the queue
00152                   if( incr  != NULL ){
00153                           pointer = incr -> process;
00154                           incr = incr -> right; // progress forward to the right of
       the queue                       incr = incr -> right; // progress forward to the
      right of the queque
00155                   }
00156                   return pointer;
00157 }
00158
00159 // used to test r3 test processes
00160 void mpxcmd_r3run(int argc, char *argv[]){
00161
00162
00163         PCB *test1;
00164         PCB *test2;
00165         PCB *test3;
00166         PCB *test4;
00167         PCB *test5;
00168
00169         STACKDSC *stack1;
00170         STACKDSC *stack2;
00171         STACKDSC *stack3;
00172         STACKDSC *stack4;
00173         STACKDSC *stack5;
00174
00175         tcontext *context1;
00176         tcontext *context2;
00177         tcontext *context3;
00178         tcontext *context4;
00179         tcontext *context5;
00180
00181         char name1[10] = "test1";
00182         char name2[10] = "test2";
00183         char name3[10] = "test3";
00184         char name4[10] = "test4";
00185         char name5[10] = "test5";
00186
00187
00188
00189         test1 = allocate_PCB();
00190         test2 = allocate_PCB();
00191         test3 = allocate_PCB();
00192         test4 = allocate_PCB();
00193         test5 = allocate_PCB();
00194
00195         stack1 = test1 -> stackdsc;
00196         stack2 = test2 -> stackdsc;
00197         stack3 = test3 -> stackdsc;
00198         stack4 = test4 -> stackdsc;
00199         stack5 = test5 -> stackdsc;
00200
00201         stack1 -> top = stack1 -> base + STACKSIZE - sizeof(tcontext);
00202         stack2 -> top = stack2 -> base + STACKSIZE - sizeof(tcontext);
00203         stack3 -> top = stack3 -> base + STACKSIZE - sizeof(tcontext);
00204         stack4 -> top = stack4 -> base + STACKSIZE - sizeof(tcontext);
```

```
00205         stack5 -> top = stack5 -> base + STACKSIZE - sizeof(tcontext);
00206
00207         context1 = (tcontext*) stack1 -> top;
00208         context2 = (tcontext*) stack2 -> top;
00209         context3 = (tcontext*) stack3 -> top;
00210         context4 = (tcontext*) stack4 -> top;
00211         context5 = (tcontext*) stack5 -> top;
00212
00213
00214
00215         context1->DS = _DS;
00216         context1->ES = _ES;
00217         context1->CS = FP_SEG(&test1_R3);
00218         context1->IP = FP_OFF(&test1_R3);
00219         context1->FLAGS = 0x200;
00220
00221         context2->DS = _DS;
00222         context2->ES = _ES;
00223         context2->CS = FP_SEG(&test2_R3);
00224         context2->IP = FP_OFF(&test2_R3);
00225         context2->FLAGS = 0x200;
00226
00227         context3->DS = _DS;
00228         context3->ES = _ES;
00229         context3->CS = FP_SEG(&test3_R3);
00230         context3->IP = FP_OFF(&test3_R3);
00231         context3->FLAGS = 0x200;
00232
00233         context4->DS = _DS;
00234         context4->ES = _ES;
00235         context4->CS = FP_SEG(&test4_R3);
00236         context4->IP = FP_OFF(&test4_R3);
00237         context4->FLAGS = 0x200;
00238
00239         context5->DS = _DS;
00240         context5->ES = _ES;
00241         context5->CS = FP_SEG(&test5_R3);
00242         context5->IP = FP_OFF(&test5_R3);
00243         context5->FLAGS = 0x200;
00244         context5->DS = _DS;
00245
00246         setup_PCB(test1,name1,APPLICATION,READY, 1);
00247         setup_PCB(test2,name2,APPLICATION,READY, 2);
00248         setup_PCB(test3,name3,APPLICATION,READY, 3);
00249         setup_PCB(test4,name4,APPLICATION,READY, 4);
00250         setup_PCB(test5,name5,APPLICATION,READY, 5);
00251
00252         insert_PCB(test1);
00253         insert_PCB(test2);
00254         insert_PCB(test3);
00255         insert_PCB(test4);
00256         insert_PCB(test5);
00257
00258
00259         mpx_cls();
00260         dispatch();
00261         mpxprompt_anykey();
00262
00263 }
00264
00265 // used to test r4 test processes
00266 void mpxcmd_gor4(int argc, char *argv[]){
00267         mpx_cls();
00268         dispatch();
00269         mpxprompt_anykey();
00270 }
00271
```

00272

4.13 src/MPX_R3.H File Reference

Data Structures

- struct context
- struct params

Defines

- #define SYS_STACK_SIZE 200

Typedefs

- typedef struct context tcontext
- typedef struct params tparams

Functions

- void interrupt sys_call (void)
- void interrupt dispatch (void)
- void mpxcmd_r3run (int argc, char *argv[ ])
- void mpxcmd_gor4 (int argc, char *argv[ ])

4.13.1 Define Documentation

4.13.1.1 #define SYS_STACK_SIZE 200

Definition at line 23 of file MPX_R3.H.

4.13.2 Typedef Documentation

4.13.2.1 typedef struct context tcontext

4.13.2.2 typedef struct params tparams

4.13.3 Function Documentation

4.13.3.1 void interrupt dispatch (void)

Definition at line 117 of file mpx_r3.c.

```
                        {

  if ( sp_save == NULL ){ //saves the SS and SP from being overwritten if n
ot already done
          ss_save = _SS;
```

```
                sp_save = _SP;
                }
                HEAD = getHead_PCB();
                //STACK = HEAD -> stackdsc;

                // get a process from the ready queue then set the ss and sp to e
        xecute the new process
                if ( HEAD != NULL ){
                        cop = HEAD;
                        cop -> state = READY;
                        remove_PCB(HEAD);
                        STACK = cop -> stackdsc;
                        new_ss = FP_SEG(STACK -> top);
                        new_sp = FP_OFF(STACK -> top );
                        _SS = new_ss;
                        _SP = new_sp;
                }else{ // if no process left return
                        cop = NULL;
                        _SS = ss_save;
                        _SP = sp_save;
                        ss_save = NULL;
                        sp_save = NULL;
                }
                //_iret;
}
```

4.13.3.2 void mpxcmd_gor4 (int *argc*, char * *argv[ ]*)

Definition at line 266 of file mpx_r3.c.

```
                                        {
        mpx_cls();
        dispatch();
        mpxprompt_anykey();
}
```

4.13.3.3 void mpxcmd_r3run (int *argc*, char * *argv[ ]*)

Definition at line 160 of file mpx_r3.c.

```
                                        {


        PCB *test1;
        PCB *test2;
        PCB *test3;
        PCB *test4;
        PCB *test5;

        STACKDSC *stack1;
        STACKDSC *stack2;
        STACKDSC *stack3;
        STACKDSC *stack4;
        STACKDSC *stack5;

        tcontext *context1;
        tcontext *context2;
        tcontext *context3;
        tcontext *context4;
        tcontext *context5;
```

```
char name1[10] = "test1";
char name2[10] = "test2";
char name3[10] = "test3";
char name4[10] = "test4";
char name5[10] = "test5";



test1 = allocate_PCB();
test2 = allocate_PCB();
test3 = allocate_PCB();
test4 = allocate_PCB();
test5 = allocate_PCB();

stack1 = test1 -> stackdsc;
stack2 = test2 -> stackdsc;
stack3 = test3 -> stackdsc;
stack4 = test4 -> stackdsc;
stack5 = test5 -> stackdsc;

stack1 -> top = stack1 -> base + STACKSIZE - sizeof(tcontext);
stack2 -> top = stack2 -> base + STACKSIZE - sizeof(tcontext);
stack3 -> top = stack3 -> base + STACKSIZE - sizeof(tcontext);
stack4 -> top = stack4 -> base + STACKSIZE - sizeof(tcontext);
stack5 -> top = stack5 -> base + STACKSIZE - sizeof(tcontext);

context1 = (tcontext*) stack1 -> top;
context2 = (tcontext*) stack2 -> top;
context3 = (tcontext*) stack3 -> top;
context4 = (tcontext*) stack4 -> top;
context5 = (tcontext*) stack5 -> top;



context1->DS = _DS;
context1->ES = _ES;
context1->CS = FP_SEG(&test1_R3);
context1->IP = FP_OFF(&test1_R3);
context1->FLAGS = 0x200;

context2->DS = _DS;
context2->ES = _ES;
context2->CS = FP_SEG(&test2_R3);
context2->IP = FP_OFF(&test2_R3);
context2->FLAGS = 0x200;

context3->DS = _DS;
context3->ES = _ES;
context3->CS = FP_SEG(&test3_R3);
context3->IP = FP_OFF(&test3_R3);
context3->FLAGS = 0x200;

context4->DS = _DS;
context4->ES = _ES;
context4->CS = FP_SEG(&test4_R3);
context4->IP = FP_OFF(&test4_R3);
context4->FLAGS = 0x200;

context5->DS = _DS;
context5->ES = _ES;
context5->CS = FP_SEG(&test5_R3);
context5->IP = FP_OFF(&test5_R3);
context5->FLAGS = 0x200;
context5->DS = _DS;

setup_PCB(test1,name1,APPLICATION,READY, 1);
setup_PCB(test2,name2,APPLICATION,READY, 2);
```

```
        setup_PCB(test3,name3,APPLICATION,READY, 3);
        setup_PCB(test4,name4,APPLICATION,READY, 4);
        setup_PCB(test5,name5,APPLICATION,READY, 5);

        insert_PCB(test1);
        insert_PCB(test2);
        insert_PCB(test3);
        insert_PCB(test4);
        insert_PCB(test5);


        mpx_cls();
        dispatch();
        mpxprompt_anykey();

}
```

4.13.3.4 void interrupt sys_call (void)

Definition at line 59 of file mpx_r3.c.

```
                            {


        cop-> stackdsc -> top = (unsigned char *) MK_FP(_SS, _SP);
        param_p = ( tparams*)(sizeof(tcontext) + cop -> stackdsc -> top);//code s
      upplied by GA bryan
        context_p = (tcontext*) cop -> stackdsc -> top;
        //SWITCH TO TEMP STACK by storing all of your variables in memory

        //replaces SS and SP
        new_ss = FP_SEG(sys_stack);
        new_sp = FP_OFF(sys_stack);
        new_sp += SYS_STACK_SIZE;
        _SS = new_ss;
        _SP = new_sp;

        // if the idle opcode is sent then change staate to ready and insert into
       queue
        if ( param_p -> op_code == IDLE ){
                        cop -> state = READY;
                        insert_PCB(cop);
                        cop = NULL;
        }

        // if the exit opcode is sent then remove and free the pcb
        if( param_p -> op_code == EXIT ){
                remove_PCB(cop);
                free_PCB(cop);
                cop = NULL;
        }

        // if ( param_p -> op_code == READ ){
                // if( param_p -> device_id == TERMINAL ){

                // }
                // if ( param_p -> device_id == COM_PORT ){

                // }
        // }
        // if ( param_p -> op_code == WRITE ){
                // if( param_p -> device_id == TERMINAL ){

                // }
```

```
                // if( param_p -> device_id == COM_PORT ){

                // }

        // }



        // FIXME NO ERROR CHECKING
        //context_p -> AX = 0;

        //CALL TO DISPATCHER
        dispatch();

}
```

4.14 src/MPX_R3.H

```
00001 /*****************************************************************************
00002         MPX: The MultiProgramming eXecutive
00003         Project to Accompany
00004         A Practical Approach to Operating Systems
00005         Malcolm G. Lane & James D. Mooney
00006         Copyright 1993, P.W.S. Kent Publishing Co., Boston, MA.
00007
00008         File Name:      mpx_r3.h
00009
00010         Author: Nathaniel Clay and Nicholas Yanak
00011         Version: 1.1
00012         Date:  12/9/2010
00013
00014         Purpose: This is the header for r3
00015
00016
00017         Environment: Windows XP 32 bit
00018
00019 *****************************************************************************/
00020 #ifndef MPX_R3_HFILE
00021 #define MPX_R3_HFILE
00022
00023 #define SYS_STACK_SIZE 200
00024
00025 typedef struct context {
00026         unsigned int BP, DI, SI, DS, ES;
00027         unsigned int DX, CX, BX, AX;
00028         unsigned int IP, CS, FLAGS;
00029 } tcontext;
00030
00031 typedef struct params {
00032 int op_code;
00033 int device_id;
00034 unsigned char *buf_addr;
00035 int *cont_addr;
00036 } tparams;
00037 /*
00038 typedef struct IOCB {
00039         char *name;
00040         void ( funct *) (void);
00041         PCB * IO_OP;
00042 } IOCB;
00043 */
00044
00045 void interrupt sys_call(void);
00046 void interrupt dispatch(void);
```

```
00047 void mpxcmd_r3run(int argc, char *argv[]);
00048 void mpxcmd_gor4(int argc, char *argv[]);
00049 #endif
```

4.15 src/mpx_r4.c File Reference

```
#include "dos.h"
#include "mpx_cmd.h"
#include "mpx_util.h"
#include "mpx_r2.h"
#include "mpx_r3.h"
#include "mpx_r4.h"
#include "mpx_supt.h"
#include "mystdlib.h"
#include <string.h>
#include <stdio.h>
```

Functions

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

Variables

- ROOT * rQueue

  declaring null roots for initial start of linked lists

- ROOT * wsQueue
- void * loadAddr

4.15.1 Function Documentation

4.15.1.1 void loadProgram (int *argc,* char * *argv[ ]*)

Definition at line 36 of file mpx_r4.c.

```
                                        { //name,fileName,priority,path

        // sets up variables
        static int count;
        MEMDSC *tempMem;
        unsigned char temptop;
        char *dir, *name, *filename;
        int size,offset,priority;
        tcontext *tempContext;
        unsigned int *tempCS,*tempIP;
        STACKDSC *temp;
```

```
        // initializes values
        int err = 0;
        PCB *newPCB = allocate_PCB();
        tempMem=newPCB->memdsc;
        dir = (char*) sys_alloc_mem(30 * sizeof(char));
        name = (char*) sys_alloc_mem(30 * sizeof(char));
        filename = (char*) sys_alloc_mem(30 * sizeof(char));
        strcpy(dir,argv[4]);
        strcpy(name,argv[1]);
        strcpy(filename,argv[2]);
        priority = atoi(argv[3]);

        err = sys_check_program(dir,filename,&size,&offset);

        if((argc==5)||(127<=priority<=-128)&&( err==0)){ //checks for validity



              /*        if( count == ZERO ){ //If first process allocate queue
                        rQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
                        wsQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
                }   */

                setup_PCB(newPCB,name,APPLICATION,SUSPENDED_READY,priority);


                // sets up the adressess
                newPCB->memdsc->loadADDR= sys_alloc_mem(size);
                newPCB->memdsc->execADDR=newPCB->memdsc->loadADDR + offset;// is
      this the correct address?


                newPCB->memdsc->loadADDR= sys_alloc_mem(size);;
                newPCB->memdsc->execADDR=newPCB->memdsc->loadADDR + offset;


                //make sure all registers are properly set

                newPCB -> stackdsc-> top = newPCB -> stackdsc-> base + STACKSIZE
      - sizeof(tcontext);

                tempContext = (tcontext *) (newPCB -> stackdsc-> top);
                tempContext ->ES = _ES;
                tempContext ->DS = _DS;
                tempContext ->CS = FP_SEG(newPCB->memdsc->execADDR);
                tempContext ->IP = FP_OFF(newPCB->memdsc->execADDR);
                tempContext ->FLAGS = 0x200;

                // load the program into memory
                 err = sys_load_program(newPCB->memdsc->loadADDR,size,dir,filenam
      e);

                insert_PCB(newPCB);     // put pcb into a queue
                count++;//Update the number of times the function has run.

        }
        else{
                printf("Wrong or invalid arguments entered.");
        }
}
```

4.15.1.2 void terminateProcess (int *argc*, char * *argv[ ]*)

Definition at line 107 of file mpx_r4.c.

```
                                        {

        if (argc == 2){ // checks for args then searches for process
                char name[STRLEN];
                PCB *pointer;
                strcpy(name,argv[1]);
                pointer = find_PCB(name);

                if ( pointer != NULL){
                        remove_PCB(pointer);
                        free_PCB(pointer);
                }
        }

        else{
                printf("Wrong arguments entered.");
                return;
        }
}
```

4.15.2 Variable Documentation

4.15.2.1 void* loadAddr

Definition at line 33 of file mpx_r4.c.

4.15.2.2 ROOT* rQueue

declaring null roots for initial start of linked lists

Definition at line 28 of file mpx_r2.c.

4.15.2.3 ROOT * wsQueue

Definition at line 29 of file mpx_r2.c.

4.16 src/mpx_r4.c

```
00001 /************************************************************************
00002         MPX: The MultiProgramming eXecutive
00003         Project to Accompany
00004         A Practical Approach to Operating Systems
00005         Malcolm G. Lane & James D. Mooney
00006         Copyright 1993, P.W.S. Kent Publishing Co., Boston, MA.
00007
00008         File Name:      mpx_r4.c
00009
00010         Author: Nathaniel Clay and Nicholas Yanak
00011         Version: 1.1
00012         Date:  12/9/2010
00013
00014         Purpose: Contains function and support for adding and removing programs
00015         into/from memory
00016
00017
00018         Environment: Windows XP 32 bit
00019
00020 ************************************************************************/
```

```
00021 #include "dos.h"
00022 #include "mpx_cmd.h"
00023 #include "mpx_util.h"
00024 #include "mpx_r2.h"
00025 #include "mpx_r3.h"
00026 #include "mpx_r4.h"
00027 #include "mpx_supt.h"
00028 #include "mystdlib.h"
00029 #include <string.h>
00030 #include <stdio.h>
00031
00032 extern ROOT *rQueue, *wsQueue; //link in the values for these in r2
00033 void * loadAddr;
00034
00035 // loads a program into memory
00036 void loadProgram(int argc, char *argv[]){ //name,fileName,priority,path
00037
00038         // sets up variables
00039         static int count;
00040         MEMDSC *tempMem;
00041         unsigned char temptop;
00042         char *dir, *name, *filename;
00043         int size,offset,priority;
00044         tcontext *tempContext;
00045         unsigned int *tempCS,*tempIP;
00046         STACKDSC *temp;
00047
00048         // initializes values
00049         int err = 0;
00050         PCB *newPCB = allocate_PCB();
00051         tempMem=newPCB->memdsc;
00052         dir = (char*) sys_alloc_mem(30 * sizeof(char));
00053         name = (char*) sys_alloc_mem(30 * sizeof(char));
00054         filename = (char*) sys_alloc_mem(30 * sizeof(char));
00055         strcpy(dir,argv[4]);
00056         strcpy(name,argv[1]);
00057         strcpy(filename,argv[2]);
00058         priority = atoi(argv[3]);
00059
00060         err = sys_check_program(dir,filename,&size,&offset);
00061
00062         if((argc==5)||(127<=priority<=-128)&&( err==0)){ //checks for validity
00063
00064
00065
00066               /*        if( count == ZERO ){ //If first process allocate queue
00067                         rQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
00068                         wsQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
00069                 }   */
00070
00071                 setup_PCB(newPCB,name,APPLICATION,SUSPENDED_READY,priority);
00072
00073
00074                 // sets up the adressess
00075                 newPCB->memdsc->loadADDR= sys_alloc_mem(size);
00076                 newPCB->memdsc->execADDR=newPCB->memdsc->loadADDR + offset;// is
      this the correct address?
00077
00078
00079                 newPCB->memdsc->loadADDR= sys_alloc_mem(size);;
00080                 newPCB->memdsc->execADDR=newPCB->memdsc->loadADDR + offset;
00081
00082
00083                 //make sure all registers are properly set
00084
00085                 newPCB -> stackdsc-> top = newPCB -> stackdsc-> base + STACKSIZE
      - sizeof(tcontext);
```

```
00086
00087                   tempContext = (tcontext *) (newPCB -> stackdsc-> top);
00088                   tempContext ->ES = _ES;
00089                   tempContext ->DS = _DS;
00090                   tempContext ->CS = FP_SEG(newPCB->memdsc->execADDR);
00091                   tempContext ->IP = FP_OFF(newPCB->memdsc->execADDR);
00092                   tempContext ->FLAGS = 0x200;
00093
00094                   // load the program into memory
00095                    err = sys_load_program(newPCB->memdsc->loadADDR,size,dir,filenam
      e);
00096
00097                   insert_PCB(newPCB);     // put pcb into a queue
00098                   count++;//Update the number of times the function has run.
00099
00100           }
00101           else{
00102                   printf("Wrong or invalid arguments entered.");
00103           }
00104 }
00105
00106 // removes process from memory
00107 void terminateProcess(int argc, char *argv[]){
00108
00109           if (argc == 2){ // checks for args then searches for process
00110                   char name[STRLEN];
00111                   PCB *pointer;
00112                   strcpy(name,argv[1]);
00113                   pointer = find_PCB(name);
00114
00115                   if ( pointer != NULL){
00116                           remove_PCB(pointer);
00117                           free_PCB(pointer);
00118                   }
00119           }
00120
00121           else{
00122                   printf("Wrong arguments entered.");
00123                   return;
00124           }
00125 }
00126
```

4.17 src/mpx_r4.c.BASE.c File Reference

```
#include "dos.h"
#include "mpx_cmd.h"
#include "mpx_util.h"
#include "mpx_r2.h"
#include "mpx_r3.h"
#include "mpx_r4.h"
#include "mpx_supt.h"
#include "mystdlib.h"
#include <string.h>
#include <stdio.h>
```

Functions

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

Variables

- ROOT * rQueue

  declaring null roots for initial start of linked lists

- ROOT * wsQueue
- void * loadAddr

4.17.1 Function Documentation

4.17.1.1 void loadProgram (int *argc*, char * *argv[ ]*)

Definition at line 14 of file mpx_r4.c.BASE.c.

```
                                    { //name,fileName,priority,path

    static int count;
    MEMDSC *tempMem;
    unsigned char temptop;
    char *dir, *name, *filename;
    int size,offset,priority;
    tcontext *tempContext;
    unsigned int *tempCS,*tempIP;
    ROOT *tempRQueue,*tempWSQueue;
    STACKDSC *temp;

    int err = 0;
    PCB *newPCB = allocate_PCB();
    tempMem=newPCB->memdsc;
    dir = (char*) sys_alloc_mem(30 * sizeof(char));
    name = (char*) sys_alloc_mem(30 * sizeof(char));
    filename = (char*) sys_alloc_mem(30 * sizeof(char));
    strcpy(dir,argv[4]);
    strcpy(name,argv[1]);
    strcpy(filename,argv[2]);
    priority = atoi(argv[3]);

    err = sys_check_program(dir,filename,&size,&offset);
    if((argc==5)||(127<=priority<=-128)&&( err==0)){


          /*        if( count == ZERO ){ //If first process allocate queue
                    rQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
                    wsQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
            }   */

            setup_PCB(newPCB,name,APPLICATION,SUSPENDED_READY,priority);



            newPCB->memdsc->loadADDR= sys_alloc_mem(size);;
            newPCB->memdsc->execADDR=newPCB->memdsc->loadADDR + offset;// is
this the correct address?
```

```
                //make sure all registers are properly set

                newPCB -> stackdsc-> top = newPCB -> stackdsc-> base + STACKSIZE
      - sizeof(tcontext);

                tempContext = (tcontext *) (newPCB -> stackdsc-> top);
                tempContext ->ES = _ES;
                tempContext ->DS = _DS;
                tempContext ->CS = FP_SEG(newPCB->memdsc->execADDR);
                tempContext ->IP = FP_OFF(newPCB->memdsc->execADDR);
                tempContext ->FLAGS = 0x200;


                 err = sys_load_program(newPCB->memdsc->loadADDR,size,dir,filenam
      e);

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

        }
        else{
                printf("Wrong or invalid arguments entered.");
        }
}
```

4.17.1.2 void terminateProcess (int *argc*, char * *argv[ ]*)

Definition at line 77 of file mpx_r4.c.BASE.c.

```
                                                {

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

                if ( pointer != NULL){
                        remove_PCB(pointer);
                        free_PCB(pointer);
                }
        }

        else{
                printf("Wrong arguments entered.");
                return;
        }
}
```

4.17.2 Variable Documentation

4.17.2.1 void* loadAddr

Definition at line 13 of file mpx_r4.c.BASE.c.

4.17.2.2 ROOT* rQueue

declaring null roots for initial start of linked lists

Definition at line 28 of file mpx_r2.c.

4.17.2.3 ROOT * wsQueue

Definition at line 29 of file mpx_r2.c.

4.18 src/mpx_r4.c.BASE.c

```
00001 #include "dos.h"
00002 #include "mpx_cmd.h"
00003 #include "mpx_util.h"
00004 #include "mpx_r2.h"
00005 #include "mpx_r3.h"
00006 #include "mpx_r4.h"
00007 #include "mpx_supt.h"
00008 #include "mystdlib.h"
00009 #include <string.h>
00010 #include <stdio.h>
00011
00012 extern ROOT *rQueue, *wsQueue; //link in the values for these in r2
00013 void * loadAddr;
00014 void loadProgram(int argc, char *argv[]){ //name,fileName,priority,path
00015
00016         static int count;
00017         MEMDSC *tempMem;
00018         unsigned char temptop;
00019         char *dir, *name, *filename;
00020         int size,offset,priority;
00021         tcontext *tempContext;
00022         unsigned int *tempCS,*tempIP;
00023         ROOT *tempRQueue,*tempWSQueue;
00024         STACKDSC *temp;
00025
00026         int err = 0;
00027         PCB *newPCB = allocate_PCB();
00028         tempMem=newPCB->memdsc;
00029         dir = (char*) sys_alloc_mem(30 * sizeof(char));
00030         name = (char*) sys_alloc_mem(30 * sizeof(char));
00031         filename = (char*) sys_alloc_mem(30 * sizeof(char));
00032         strcpy(dir,argv[4]);
00033         strcpy(name,argv[1]);
00034         strcpy(filename,argv[2]);
00035         priority = atoi(argv[3]);
00036
00037         err = sys_check_program(dir,filename,&size,&offset);
00038         if((argc==5)||(127<=priority<=-128)&&( err==0)){
00039
00040
00041
00042               /*        if( count == ZERO ){ //If first process allocate queue
00043                         rQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
00044                         wsQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
00045                 }   */
00046
00047                 setup_PCB(newPCB,name,APPLICATION,SUSPENDED_READY,priority);
00048
00049
00050
00051                 newPCB->memdsc->loadADDR= sys_alloc_mem(size);;
00052                 newPCB->memdsc->execADDR=newPCB->memdsc->loadADDR + offset;// is
      this the correct address?
00053
00054                 //make sure all registers are properly set
00055
00056                 newPCB -> stackdsc-> top = newPCB -> stackdsc-> base + STACKSIZE
      - sizeof(tcontext);
```

```
00057
00058                   tempContext = (tcontext *) (newPCB -> stackdsc-> top);
00059                   tempContext ->ES = _ES;
00060                   tempContext ->DS = _DS;
00061                   tempContext ->CS = FP_SEG(newPCB->memdsc->execADDR);
00062                   tempContext ->IP = FP_OFF(newPCB->memdsc->execADDR);
00063                   tempContext ->FLAGS = 0x200;
00064
00065
00066                    err = sys_load_program(newPCB->memdsc->loadADDR,size,dir,filenam
      e);
00067
00068                   insert_PCB(newPCB);
00069                   count++;//Update the number of times the function has run.
00070
00071           }
00072           else{
00073                   printf("Wrong or invalid arguments entered.");
00074           }
00075 }
00076
00077 void terminateProcess(int argc, char *argv[]){
00078
00079           if (argc == 2){
00080                   char name[STRLEN];
00081                   PCB *pointer;
00082                   strcpy(name,argv[1]);
00083                   pointer = find_PCB(name);
00084
00085                   if ( pointer != NULL){
00086                           remove_PCB(pointer);
00087                           free_PCB(pointer);
00088                   }
00089           }
00090
00091           else{
00092                   printf("Wrong arguments entered.");
00093                   return;
00094           }
00095 }
00096
```

4.19 src/mpx_r4.c.LOCAL.c File Reference

```
#include "dos.h"
#include "mpx_cmd.h"
#include "mpx_util.h"
#include "mpx_r2.h"
#include "mpx_r3.h"
#include "mpx_r4.h"
#include "mpx_supt.h"
#include "mystdlib.h"
#include <string.h>
#include <stdio.h>
```

Functions

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

Variables

- ROOT * rQueue

  declaring null roots for initial start of linked lists

- ROOT * wsQueue
- void * loadAddr

4.19.1 Function Documentation

4.19.1.1 void loadProgram (int *argc*, char * *argv[ ]*)

Definition at line 16 of file mpx_r4.c.LOCAL.c.

```
                                    { //name,fileName,priority,path

// sets up variables
static int count;
MEMDSC *tempMem;
unsigned char temptop;
char *dir, *name, *filename;
int size,offset,priority;
tcontext *tempContext;
unsigned int *tempCS,*tempIP;
ROOT *tempRQueue,*tempWSQueue;
STACKDSC *temp;

// initializes values
int err = 0;
PCB *newPCB = allocate_PCB();
tempMem=newPCB->memdsc;
dir = (char*) sys_alloc_mem(30 * sizeof(char));
name = (char*) sys_alloc_mem(30 * sizeof(char));
filename = (char*) sys_alloc_mem(30 * sizeof(char));
strcpy(dir,argv[4]);
strcpy(name,argv[1]);
strcpy(filename,argv[2]);
priority = atoi(argv[3]);

err = sys_check_program(dir,filename,&size,&offset);

if((argc==5)||(127<=priority<=-128)&&( err==0)){ //checks for validity


      /*        if( count == ZERO ){ //If first process allocate queue
                rQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
                wsQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
        }   */

        setup_PCB(newPCB,name,APPLICATION,SUSPENDED_READY,priority);


        // sets up the adressess
        newPCB->memdsc->loadADDR= sys_alloc_mem(size);
```

```
                newPCB->memdsc->execADDR=newPCB->memdsc->loadADDR + offset;// is
        this the correct address?

                //make sure all registers are properly set

                newPCB -> stackdsc-> top = newPCB -> stackdsc-> base + STACKSIZE
        - sizeof(tcontext);

                tempContext = (tcontext *) (newPCB -> stackdsc-> top);
                tempContext ->ES = _ES;
                tempContext ->DS = _DS;
                tempContext ->CS = FP_SEG(newPCB->memdsc->execADDR);
                tempContext ->IP = FP_OFF(newPCB->memdsc->execADDR);
                tempContext ->FLAGS = 0x200;

                // load the program into memory
                 err = sys_load_program(newPCB->memdsc->loadADDR,size,dir,filenam
        e);

                insert_PCB(newPCB);     // put pcb into a queue
                count++;//Update the number of times the function has run.

          }
          else{
                printf("Wrong or invalid arguments entered.");
          }
}
```

4.19.1.2 void terminateProcess (int *argc*, char * *argv[ ]*)

Definition at line 83 of file mpx_r4.c.LOCAL.c.

```
                                                {

          if (argc == 2){ // checks for args then searches for process
                char name[STRLEN];
                PCB *pointer;
                strcpy(name,argv[1]);
                pointer = find_PCB(name);

                if ( pointer != NULL){
                        remove_PCB(pointer);
                        free_PCB(pointer);
                }
          }

          else{
                printf("Wrong arguments entered.");
                return;
          }
}
```

4.19.2 Variable Documentation

4.19.2.1 void* loadAddr

Definition at line 13 of file mpx_r4.c.LOCAL.c.

4.19.2.2 ROOT* rQueue

declaring null roots for initial start of linked lists

Definition at line 28 of file mpx_r2.c.

4.19.2.3 ROOT * wsQueue

Definition at line 29 of file mpx_r2.c.

4.20 src/mpx_r4.c.LOCAL.c

```
00001 #include "dos.h"
00002 #include "mpx_cmd.h"
00003 #include "mpx_util.h"
00004 #include "mpx_r2.h"
00005 #include "mpx_r3.h"
00006 #include "mpx_r4.h"
00007 #include "mpx_supt.h"
00008 #include "mystdlib.h"
00009 #include <string.h>
00010 #include <stdio.h>
00011
00012 extern ROOT *rQueue, *wsQueue; //link in the values for these in r2
00013 void * loadAddr;
00014
00015 // loads a program into memory
00016 void loadProgram(int argc, char *argv[]){ //name,fileName,priority,path
00017
00018         // sets up variables
00019         static int count;
00020         MEMDSC *tempMem;
00021         unsigned char temptop;
00022         char *dir, *name, *filename;
00023         int size,offset,priority;
00024         tcontext *tempContext;
00025         unsigned int *tempCS,*tempIP;
00026         ROOT *tempRQueue,*tempWSQueue;
00027         STACKDSC *temp;
00028
00029         // initializes values
00030         int err = 0;
00031         PCB *newPCB = allocate_PCB();
00032         tempMem=newPCB->memdsc;
00033         dir = (char*) sys_alloc_mem(30 * sizeof(char));
00034         name = (char*) sys_alloc_mem(30 * sizeof(char));
00035         filename = (char*) sys_alloc_mem(30 * sizeof(char));
00036         strcpy(dir,argv[4]);
00037         strcpy(name,argv[1]);
00038         strcpy(filename,argv[2]);
00039         priority = atoi(argv[3]);
00040
00041         err = sys_check_program(dir,filename,&size,&offset);
00042
00043         if((argc==5)||(127<=priority<=-128)&&( err==0)){ //checks for validity
00044
00045
00046
00047               /*        if( count == ZERO ){ //If first process allocate queue
00048                         rQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
00049                         wsQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
00050                 }   */
```

```
00051
00052                   setup_PCB(newPCB,name,APPLICATION,SUSPENDED_READY,priority);
00053
00054
00055                   // sets up the adressess
00056                   newPCB->memdsc->loadADDR= sys_alloc_mem(size);
00057                   newPCB->memdsc->execADDR=newPCB->memdsc->loadADDR + offset;// is
      this the correct address?
00058
00059                   //make sure all registers are properly set
00060
00061                   newPCB -> stackdsc-> top = newPCB -> stackdsc-> base + STACKSIZE
      - sizeof(tcontext);
00062
00063                   tempContext = (tcontext *) (newPCB -> stackdsc-> top);
00064                   tempContext ->ES = _ES;
00065                   tempContext ->DS = _DS;
00066                   tempContext ->CS = FP_SEG(newPCB->memdsc->execADDR);
00067                   tempContext ->IP = FP_OFF(newPCB->memdsc->execADDR);
00068                   tempContext ->FLAGS = 0x200;
00069
00070                   // load the program into memory
00071                    err = sys_load_program(newPCB->memdsc->loadADDR,size,dir,filenam
      e);
00072
00073                   insert_PCB(newPCB);     // put pcb into a queue
00074                   count++;//Update the number of times the function has run.
00075
00076           }
00077           else{
00078                   printf("Wrong or invalid arguments entered.");
00079           }
00080 }
00081
00082 // removes process from memory
00083 void terminateProcess(int argc, char *argv[]){
00084
00085           if (argc == 2){ // checks for args then searches for process
00086                   char name[STRLEN];
00087                   PCB *pointer;
00088                   strcpy(name,argv[1]);
00089                   pointer = find_PCB(name);
00090
00091                   if ( pointer != NULL){
00092                           remove_PCB(pointer);
00093                           free_PCB(pointer);
00094                   }
00095           }
00096
00097           else{
00098                   printf("Wrong arguments entered.");
00099                   return;
00100           }
00101 }
00102
```

4.21 src/mpx_r4.c.REMOTE.c File Reference

```
#include "dos.h"
#include "mpx_cmd.h"
#include "mpx_util.h"
#include "mpx_r2.h"
```

```
#include "mpx_r3.h"
#include "mpx_r4.h"
#include "mpx_supt.h"
#include "mystdlib.h"
#include <string.h>
#include <stdio.h>
```

Functions

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

Variables

- ROOT * rQueue

  declaring null roots for initial start of linked lists

- ROOT * wsQueue
- void * loadAddr

4.21.1 Function Documentation

4.21.1.1 void loadProgram (int *argc*, char * *argv[ ]*)

Definition at line 14 of file mpx_r4.c.REMOTE.c.

```
                                    { //name,fileName,priority,path

        static int count;
        MEMDSC *tempMem;
        unsigned char temptop;
        char *dir, *name, *filename;
        int size,offset,priority;
        tcontext *tempContext;
        unsigned int *tempCS,*tempIP;
        STACKDSC *temp;

        int err = 0;
        PCB *newPCB = allocate_PCB();
        tempMem=newPCB->memdsc;
        dir = (char*) sys_alloc_mem(30 * sizeof(char));
        name = (char*) sys_alloc_mem(30 * sizeof(char));
        filename = (char*) sys_alloc_mem(30 * sizeof(char));
        strcpy(dir,argv[4]);
        strcpy(name,argv[1]);
        strcpy(filename,argv[2]);
        priority = atoi(argv[3]);

        err = sys_check_program(dir,filename,&size,&offset);
        if((argc==5)||(127<=priority<=-128)&&( err==0)){


              /*        if( count == ZERO ){ //If first process allocate queue
```

```
                        rQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
                        wsQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
                }   */

                setup_PCB(newPCB,name,APPLICATION,SUSPENDED_READY,priority);



                newPCB->memdsc->loadADDR= sys_alloc_mem(size);;
                newPCB->memdsc->execADDR=newPCB->memdsc->loadADDR + offset;

                //make sure all registers are properly set

                newPCB -> stackdsc-> top = newPCB -> stackdsc-> base + STACKSIZE
      - sizeof(tcontext);

                tempContext = (tcontext *) (newPCB -> stackdsc-> top);
                tempContext ->ES = _ES;
                tempContext ->DS = _DS;
                tempContext ->CS = FP_SEG(newPCB->memdsc->execADDR);
                tempContext ->IP = FP_OFF(newPCB->memdsc->execADDR);
                tempContext ->FLAGS = 0x200;


                 err = sys_load_program(newPCB->memdsc->loadADDR,size,dir,filenam
      e);

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

        }
        else{
                printf("Wrong or invalid arguments entered.");
        }
}
```

4.21.1.2 void terminateProcess (int *argc*, char * *argv[ ]*)

Definition at line 76 of file mpx_r4.c.REMOTE.c.

```
                                                {

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

                if ( pointer != NULL){
                        remove_PCB(pointer);
                        free_PCB(pointer);
                }
        }

        else{
                printf("Wrong arguments entered.");
                return;
        }
}
```

4.21.2 Variable Documentation

4.21.2.1 void* loadAddr

Definition at line 13 of file mpx_r4.c.REMOTE.c.

4.21.2.2 ROOT* rQueue

declaring null roots for initial start of linked lists

Definition at line 28 of file mpx_r2.c.

4.21.2.3 ROOT * wsQueue

Definition at line 29 of file mpx_r2.c.

4.22 src/mpx_r4.c.REMOTE.c

```
00001 #include "dos.h"
00002 #include "mpx_cmd.h"
00003 #include "mpx_util.h"
00004 #include "mpx_r2.h"
00005 #include "mpx_r3.h"
00006 #include "mpx_r4.h"
00007 #include "mpx_supt.h"
00008 #include "mystdlib.h"
00009 #include <string.h>
00010 #include <stdio.h>
00011
00012 extern ROOT *rQueue, *wsQueue; //link in the values for these in r2
00013 void * loadAddr;
00014 void loadProgram(int argc, char *argv[]){ //name,fileName,priority,path
00015
00016         static int count;
00017         MEMDSC *tempMem;
00018         unsigned char temptop;
00019         char *dir, *name, *filename;
00020         int size,offset,priority;
00021         tcontext *tempContext;
00022         unsigned int *tempCS,*tempIP;
00023         STACKDSC *temp;
00024
00025         int err = 0;
00026         PCB *newPCB = allocate_PCB();
00027         tempMem=newPCB->memdsc;
00028         dir = (char*) sys_alloc_mem(30 * sizeof(char));
00029         name = (char*) sys_alloc_mem(30 * sizeof(char));
00030         filename = (char*) sys_alloc_mem(30 * sizeof(char));
00031         strcpy(dir,argv[4]);
00032         strcpy(name,argv[1]);
00033         strcpy(filename,argv[2]);
00034         priority = atoi(argv[3]);
00035
00036         err = sys_check_program(dir,filename,&size,&offset);
00037         if((argc==5)||(127<=priority<=-128)&&( err==0)){
00038
00039
00040
00041               /*        if( count == ZERO ){ //If first process allocate queue
00042                         rQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
```

```
00043                           wsQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
00044                   }   */
00045
00046                   setup_PCB(newPCB,name,APPLICATION,SUSPENDED_READY,priority);
00047
00048
00049
00050                   newPCB->memdsc->loadADDR= sys_alloc_mem(size);;
00051                   newPCB->memdsc->execADDR=newPCB->memdsc->loadADDR + offset;
00052
00053                   //make sure all registers are properly set
00054
00055                   newPCB -> stackdsc-> top = newPCB -> stackdsc-> base + STACKSIZE
      - sizeof(tcontext);
00056
00057                   tempContext = (tcontext *) (newPCB -> stackdsc-> top);
00058                   tempContext ->ES = _ES;
00059                   tempContext ->DS = _DS;
00060                   tempContext ->CS = FP_SEG(newPCB->memdsc->execADDR);
00061                   tempContext ->IP = FP_OFF(newPCB->memdsc->execADDR);
00062                   tempContext ->FLAGS = 0x200;
00063
00064
00065                    err = sys_load_program(newPCB->memdsc->loadADDR,size,dir,filenam
      e);
00066
00067                   insert_PCB(newPCB);
00068                   count++;//Update the number of times the function has run.
00069
00070           }
00071           else{
00072                   printf("Wrong or invalid arguments entered.");
00073           }
00074 }
00075
00076 void terminateProcess(int argc, char *argv[]){
00077
00078           if (argc == 2){
00079                   char name[STRLEN];
00080                   PCB *pointer;
00081                   strcpy(name,argv[1]);
00082                   pointer = find_PCB(name);
00083
00084                   if ( pointer != NULL){
00085                           remove_PCB(pointer);
00086                           free_PCB(pointer);
00087                   }
00088           }
00089
00090           else{
00091                   printf("Wrong arguments entered.");
00092                   return;
00093           }
00094 }
00095
```

4.23 src/mpx_r4.h File Reference

Functions

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

4.23.1 Function Documentation

4.23.1.1 void loadProgram (int *argc*, char * *argv[ ]*)

Definition at line 36 of file mpx_r4.c.

```
                                    { //name,fileName,priority,path

  // sets up variables
  static int count;
  MEMDSC *tempMem;
  unsigned char temptop;
  char *dir, *name, *filename;
  int size,offset,priority;
  tcontext *tempContext;
  unsigned int *tempCS,*tempIP;
  STACKDSC *temp;

  // initializes values
  int err = 0;
  PCB *newPCB = allocate_PCB();
  tempMem=newPCB->memdsc;
  dir = (char*) sys_alloc_mem(30 * sizeof(char));
  name = (char*) sys_alloc_mem(30 * sizeof(char));
  filename = (char*) sys_alloc_mem(30 * sizeof(char));
  strcpy(dir,argv[4]);
  strcpy(name,argv[1]);
  strcpy(filename,argv[2]);
  priority = atoi(argv[3]);

  err = sys_check_program(dir,filename,&size,&offset);

  if((argc==5)||(127<=priority<=-128)&&( err==0)){ //checks for validity


        /*        if( count == ZERO ){ //If first process allocate queue
                  rQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
                  wsQueue = (ROOT*) sys_alloc_mem(sizeof(ROOT));
          }   */

          setup_PCB(newPCB,name,APPLICATION,SUSPENDED_READY,priority);


          // sets up the adressess
          newPCB->memdsc->loadADDR= sys_alloc_mem(size);
          newPCB->memdsc->execADDR=newPCB->memdsc->loadADDR + offset;// is
this the correct address?


          newPCB->memdsc->loadADDR= sys_alloc_mem(size);;
          newPCB->memdsc->execADDR=newPCB->memdsc->loadADDR + offset;


          //make sure all registers are properly set

          newPCB -> stackdsc-> top = newPCB -> stackdsc-> base + STACKSIZE
- sizeof(tcontext);

          tempContext = (tcontext *) (newPCB -> stackdsc-> top);
          tempContext ->ES = _ES;
          tempContext ->DS = _DS;
          tempContext ->CS = FP_SEG(newPCB->memdsc->execADDR);
          tempContext ->IP = FP_OFF(newPCB->memdsc->execADDR);
          tempContext ->FLAGS = 0x200;
```

```
                // load the program into memory
                 err = sys_load_program(newPCB->memdsc->loadADDR,size,dir,filenam
      e);

                insert_PCB(newPCB);     // put pcb into a queue
                count++;//Update the number of times the function has run.

        }
        else{
                printf("Wrong or invalid arguments entered.");
        }
}
```

4.23.1.2 void terminateProcess (int *argc*, char * *argv[ ]*)

Definition at line 107 of file mpx_r4.c.

```
                                        {

        if (argc == 2){ // checks for args then searches for process
                char name[STRLEN];
                PCB *pointer;
                strcpy(name,argv[1]);
                pointer = find_PCB(name);

                if ( pointer != NULL){
                        remove_PCB(pointer);
                        free_PCB(pointer);
                }
        }

        else{
                printf("Wrong arguments entered.");
                return;
        }
}
```

4.24 src/mpx_r4.h

```
00001 /************************************************************************
00002         MPX: The MultiProgramming eXecutive
00003         Project to Accompany
00004         A Practical Approach to Operating Systems
00005         Malcolm G. Lane & James D. Mooney
00006         Copyright 1993, P.W.S. Kent Publishing Co., Boston, MA.
00007
00008         File Name:      mpx_r4.h
00009
00010         Author: Nathaniel Clay and Nicholas Yanak
00011         Version: 1.1
00012         Date:  12/9/2010
00013
00014         Purpose: This is the header file for r4.
00015
00016
00017         Environment: Windows XP 32 bit
00018
00019 ************************************************************************/
00020 #ifndef MPX_R4_HFILE
00021 #define MPX_R4_HFILE
00022
```

```
00023 void loadProgram(int argc, char *argv[]);
00024 void terminateProcess(int argc, char *argv[]);
00025
00026
00027 #endif
```

4.25 src/MPX_R5.C File Reference

```
#include "mpx_supt.h"
#include <stdlib.h>
#include <dos.h>
#include "MPX_R5.h"
```

Functions

- int com_open (int *eflag, int baudrate)
- int com_close (void)
- void interrupt level1 ()
- void level2Write ()
- void level2Read ()
- int com_read (char *buf_p, int *count_p)
- int com_write (char *buf_p, int *count_p)

4.25.1 Function Documentation

4.25.1.1 int com_close (void)

Definition at line 49 of file MPX_R5.C.

```
                {
        int mask;

        if(dcbPtr.flag != OPEN) //check that port is open
                return SERIAL_PORT_NOT_OPEN;

        dcbPtr.flag=CLOSED;
        disable(); //start enable
        mask = inportb(PIC_MASK);
        mask = mask | 0x10;
        outportb(PIC_MASK, mask);
        enable(); //end enable

        outportb( MS,0x00); // clears Modem Control Status
        outportb( INT_EN,0x00); // clears int_en
        setvect( INT_ID, oldfunc); //restore Microsoft interupt

        return 0;

}
```

4.25.1.2 int com_open (int * *eflag*, int *baudrate*)

Definition at line 6 of file MPX_R5.C.

```
                                                {
            long brd;
            int mask;

            if(eflag == NULL)
                    return INV_FLAG; // invalid flag
            if(baudrate <= 0)
                    return INV_BAUD;  // invalid baud
            if(dcbPtr.flag==OPEN) // Make sure that the device is not open.
                    return PORT_ALREADY_OPEN;

            dcbPtr.flag = OPEN;
            dcbPtr.flag_ptr = eflag;
            dcbPtr.status = IDLE;
            dcbPtr.ringbufin = 0;
            dcbPtr.ringbufout = 0;
            dcbPtr.ringbufcount = 0;

            oldfunc = getvect(INT_ID); //get the vector of the Windows compor
  t interupt handler
            setvect(INT_ID, &level1); //level1 is interrupt handler
            brd = 115200 / (long) baudrate; //calculate baud rate divisor


            outportb(LC, 0x80); //store 0x80 in line control register
            outportb(BRD_LSB, brd & 0xFF); //is Baud rate devisor LSB
            outportb(BRD_MSB,(brd>>8) & 0xFF); //is Baud rate devisor MSB
            outportb(LC, 0x03); //store 0x03 in line control register

            disable(); // disable interupts
            mask = inportb(PIC_MASK);
            mask = mask & 0xEF;
            outportb(PIC_MASK, mask);
            enable(); // enable interupts

            //enable level for COM1 in PIC Mask register
            //Store 0x08 in modem control register
            outportb( MC, 0x08);//enables serial interrupts
            //store 0x01 in interrupt enable register
            outportb( INT_EN, 0x01);//enables input ready interrupts

  return 0; // return zero if no error.
      }
```

4.25.1.3 int com_read (char * *buf_p*, int * *count_p*)

Definition at line 143 of file MPX_R5.C.

```
                                                {
            //Validate the supplied parameters.
            if(dcbPtr.flag != OPEN) //check if device is open
                    return READ_PORT_NOT_OPEN;
            if(dcbPtr.status != IDLE) //check if device is idle
                    return READ_DEV_BUSY;
            if( buf_p == NULL) //check if buffer is empty
                    return READ_INV_BUFF_ADD;
            if( &count_p == NULL) //check if count pointer is null
                    return READ_INV_COUNT;
      // Initialize the input buffer variables (not the ring buffer!) and set t
```

```
he status to reading.
          dcbPtr.inbuff = buf_p;
          dcbPtr.incount = count_p;
          dcbPtr.indone = 0;

          *(dcbPtr.flag_ptr) = FLAG_CLEAR; //clear event flag

          disable(); //disable interrupts
          dcbPtr.status=READ; //we are now reading

          /* Copy characters from the ring buffer to the requestor's buffer
,
          until the ring buffer is emptied, the requested count has been
          reached, or a CR (ENTER) code has been found. The copied
          characters should, of course, be removed from the ring buffer.
          Either input interrupts or all interrupts should be disabled
          during the copying. */

          while((dcbPtr.ringbufcount >0) && (dcbPtr.inbuff-1 !='\r' && (
dcbPtr.indone >= *(dcbPtr.incount)))){
                        *((dcbPtr.inbuff)) = dcbPtr.ringbuf[dcbPtr.
ringbufout];
                        dcbPtr.indone++;
                        dcbPtr.inbuff++;
                        dcbPtr.ringbufout = (dcbPtr.ringbufout+1)%size;
                        dcbPtr.ringbufcount--;

                } //end while

                enable(); //enable interrupts

          //the requestor buffer is not yet full
          if(dcbPtr.indone < *(dcbPtr.incount))
                  return 0;
          if(*(dcbPtr.inbuff-1) == '\r')
                  *(dcbPtr.inbuff-1) = '\0';
          else
                  *dcbPtr.inbuff = '\0';

          //Reset the DCB status to idle, set the event flag, and
          //return the actual count to the requestor's variable.
          dcbPtr.status = IDLE; //status back to IDLE
          *dcbPtr.flag_ptr = SET; //the event is over
          *dcbPtr.incount = dcbPtr.indone;
   return 0;
  }
```

4.25.1.4 int com_write (char * *buf_p*, int * *count_p*)

Definition at line 197 of file MPX_R5.C.

```
                                        {
          int mask;
          //Ensure that the input parameters are valid.
          if(dcbPtr.flag != OPEN) //check if device is open
                  return WRITE_PORT_NOT_OPEN;
          if(dcbPtr.status != IDLE) //check if device is idle
                  return WRITE_DEV_BUSY;
          if(buf_p == NULL) //check if buffer is empty
                  return WRITE_INV_BUFF_ADD;
          if(count_p == NULL) //check pointer is null
                  return WRITE_INV_COUNT;

          //Install the buffer pointer and counters in the DCB, and set the
```

```
 current status to writing.
          dcbPtr.outbuff = buf_p;
          dcbPtr.outcount = count_p;
          dcbPtr.outdone = 0;
          dcbPtr.status = WRITE;
          //Clear the caller's event flag.
          *dcbPtr.flag_ptr = FLAG_CLEAR;

          //Get the first character from the requestor's buffer and store i
t in the output register.
          outportb(BASE, *dcbPtr.outbuff);
          dcbPtr.outbuff++;
          dcbPtr.outdone++;


           //Enable write interrupts by setting bit 1 of the Interrupt Enab
le register.
           //This must be done by setting the register to the logical or of
 its previous
           //contents and 0x02.

          mask = inportb(INT_EN);
          mask = mask | 0x02;
          outportb(INT_EN,mask);

  return 0;
  }
```

4.25.1.5 void interrupt level1 ( )

Definition at line 70 of file MPX_R5.C.

```
                        {
          if(dcbPtr.flag != OPEN){
                  outportb(PIC_CMD, EOI);//clear interupt PIC command regis
ter
                  return;
          } else{
                  num = ((inportb(INT_ID_REG) & WHATINTERRUPTBIT));
                  if (num == 2) // 0000 0010 : write interrupt
                          level2Write();
                  if (num == 4) // 0000 0100 : read interrupt
                          level2Read();
                  outportb(PIC_CMD, EOI); //clear interupt PIC command regi
ster
          }
return;

   }
```

4.25.1.6 void level2Read ( )

Definition at line 110 of file MPX_R5.C.

```
                  {
          char new;
          char ret = '\r';

          new=inportb(BASE); //Read a character from the input register.
          if ( new != ret )
          outportb(BASE, new);// ECHO BACK
```

```
            //If the current status is not reading, store the character in th
      e ring buffer.
            if(dcbPtr.status != READ){
                    if(dcbPtr.ringbufcount != size){
                            dcbPtr.ringbuf[dcbPtr.ringbufin]= new;
                            dcbPtr.ringbufin = (dcbPtr.ringbufin+1)%size;
                            dcbPtr.ringbufcount++;
                    }
            }else{ //status is reading
                    *dcbPtr.inbuff = new;
                    dcbPtr.inbuff++;
                    dcbPtr.indone++;
                            //If the count is not completed and the character
       is not CR, return. Do not signal completion.
                    if(new== '\r' || (dcbPtr.indone ) >= *(dcbPtr.incount)){
                            if(*(dcbPtr.inbuff-1) == '\r'){
                                    *(dcbPtr.inbuff-1) = '\0';
                            }else{
                                    *dcbPtr.inbuff = '\0';
                            }
                            *dcbPtr.incount = dcbPtr.indone;
                            dcbPtr.status =IDLE;
                            *dcbPtr.flag_ptr = SET;
            }//end if
        }//end else

        }
```

4.25.1.7 void level2Write ( )

Definition at line 86 of file MPX_R5.C.

```
                          {

            int mask;

            if(dcbPtr. status != WRITE)
                    return; //Ignore the interrupt and return

            if(dcbPtr.outdone < *dcbPtr.outcount){
                    outportb(BASE, *dcbPtr.outbuff);
                    dcbPtr.outbuff++;
                    dcbPtr.outdone++;
                    return;
            }else{

                    dcbPtr.status = IDLE;
                    *dcbPtr.flag_ptr = SET;

                    mask = inportb(INT_EN);
                    mask = mask&~0x02;
                    outportb(INT_EN, mask);
                    return;
            }
        }
```

4.26 src/MPX_R5.C

```
00001 # include "mpx_supt.h"
00002 # include <stdlib.h>
00003 # include <dos.h>
```

```
00004 # include "MPX_R5.h"
00005
00006         int com_open (int *eflag, int baudrate){
00007                 long brd;
00008                 int mask;
00009
00010                 if(eflag == NULL)
00011                         return INV_FLAG; // invalid flag
00012                 if(baudrate <= 0)
00013                         return INV_BAUD;  // invalid baud
00014                 if(dcbPtr.flag==OPEN) // Make sure that the device is not open.
00015                         return PORT_ALREADY_OPEN;
00016
00017                 dcbPtr.flag = OPEN;
00018                 dcbPtr.flag_ptr = eflag;
00019                 dcbPtr.status = IDLE;
00020                 dcbPtr.ringbufin = 0;
00021                 dcbPtr.ringbufout = 0;
00022                 dcbPtr.ringbufcount = 0;
00023
00024                 oldfunc = getvect(INT_ID); //get the vector of the Windows compor
      t interupt handler
00025                 setvect(INT_ID, &level1); //level1 is interrupt handler
00026                 brd = 115200 / (long) baudrate; //calculate baud rate divisor
00027
00028
00029                 outportb(LC, 0x80); //store 0x80 in line control register
00030                 outportb(BRD_LSB, brd & 0xFF); //is Baud rate devisor LSB
00031                 outportb(BRD_MSB,(brd>>8) & 0xFF); //is Baud rate devisor MSB
00032                 outportb(LC, 0x03); //store 0x03 in line control register
00033
00034                 disable(); // disable interupts
00035                 mask = inportb(PIC_MASK);
00036                 mask = mask & 0xEF;
00037                 outportb(PIC_MASK, mask);
00038                 enable(); // enable interupts
00039
00040                 //enable level for COM1 in PIC Mask register
00041                 //Store 0x08 in modem control register
00042                 outportb( MC, 0x08);//enables serial interrupts
00043                 //store 0x01 in interrupt enable register
00044                 outportb( INT_EN, 0x01);//enables input ready interrupts
00045
00046     return 0; // return zero if no error.
00047         }
00048
00049         int com_close (void){
00050                 int mask;
00051
00052                 if(dcbPtr.flag != OPEN) //check that port is open
00053                         return SERIAL_PORT_NOT_OPEN;
00054
00055                 dcbPtr.flag=CLOSED;
00056                 disable(); //start enable
00057                 mask = inportb(PIC_MASK);
00058                 mask = mask | 0x10;
00059                 outportb(PIC_MASK, mask);
00060                 enable(); //end enable
00061
00062                 outportb( MS,0x00); // clears Modem Control Status
00063                 outportb( INT_EN,0x00); // clears int_en
00064                 setvect( INT_ID, oldfunc); //restore Microsoft interupt
00065
00066                 return 0;
00067
00068         }
00069
```

```
00070         void interrupt level1(){
00071                 if(dcbPtr.flag != OPEN){
00072                         outportb(PIC_CMD, EOI);//clear interupt PIC command regis
      ter
00073                         return;
00074                 } else{
00075                         num = ((inportb(INT_ID_REG) & WHATINTERRUPTBIT));
00076                         if (num == 2) // 0000 0010 : write interrupt
00077                                 level2Write();
00078                         if (num == 4) // 0000 0100 : read interrupt
00079                                 level2Read();
00080                         outportb(PIC_CMD, EOI); //clear interupt PIC command regi
      ster
00081                 }
00082     return;
00083
00084         }
00085
00086         void level2Write(){
00087
00088                 int mask;
00089
00090                 if(dcbPtr. status != WRITE)
00091                         return; //Ignore the interrupt and return
00092
00093                 if(dcbPtr.outdone < *dcbPtr.outcount){
00094                         outportb(BASE, *dcbPtr.outbuff);
00095                         dcbPtr.outbuff++;
00096                         dcbPtr.outdone++;
00097                         return;
00098                 }else{
00099
00100                         dcbPtr.status = IDLE;
00101                         *dcbPtr.flag_ptr = SET;
00102
00103                         mask = inportb(INT_EN);
00104                         mask = mask&~0x02;
00105                         outportb(INT_EN, mask);
00106                         return;
00107                 }
00108         }
00109
00110         void level2Read(){
00111                 char new;
00112                 char ret = '\r';
00113
00114                 new=inportb(BASE); //Read a character from the input register.
00115                 if ( new != ret )
00116                 outportb(BASE, new);// ECHO BACK
00117                 //If the current status is not reading, store the character in th
      e ring buffer.
00118                 if(dcbPtr.status != READ){
00119                         if(dcbPtr.ringbufcount != size){
00120                                 dcbPtr.ringbuf[dcbPtr.ringbufin]= new;
00121                                 dcbPtr.ringbufin = (dcbPtr.ringbufin+1)%size;
00122                                 dcbPtr.ringbufcount++;
00123                         }
00124                 }else{ //status is reading
00125                         *dcbPtr.inbuff = new;
00126                         dcbPtr.inbuff++;
00127                         dcbPtr.indone++;
00128                                 //If the count is not completed and the character
       is not CR, return. Do not signal completion.
00129                         if(new== '\r' || (dcbPtr.indone ) >= *(dcbPtr.incount)){
00130                                 if(*(dcbPtr.inbuff-1) == '\r'){
00131                                         *(dcbPtr.inbuff-1) = '\0';
00132                                 }else{
```

```
00133                                           *dcbPtr.inbuff = '\0';
00134                                   }
00135                                   *dcbPtr.incount = dcbPtr.indone;
00136                                   dcbPtr.status =IDLE;
00137                                   *dcbPtr.flag_ptr = SET;
00138                   }//end if
00139           }//end else
00140
00141           }
00142
00143           int com_read (char *buf_p,int *count_p){
00144                   //Validate the supplied parameters.
00145                   if(dcbPtr.flag != OPEN) //check if device is open
00146                           return READ_PORT_NOT_OPEN;
00147                   if(dcbPtr.status != IDLE) //check if device is idle
00148                           return READ_DEV_BUSY;
00149                   if( buf_p == NULL) //check if buffer is empty
00150                           return READ_INV_BUFF_ADD;
00151                   if( &count_p == NULL) //check if count pointer is null
00152                           return READ_INV_COUNT;
00153           // Initialize the input buffer variables (not the ring buffer!) and set t
      he status to reading.
00154                   dcbPtr.inbuff = buf_p;
00155                   dcbPtr.incount = count_p;
00156                   dcbPtr.indone = 0;
00157
00158                   *(dcbPtr.flag_ptr) = FLAG_CLEAR; //clear event flag
00159
00160                   disable(); //disable interrupts
00161                   dcbPtr.status=READ; //we are now reading
00162
00163                   /* Copy characters from the ring buffer to the requestor's buffer
      ,
00164                   until the ring buffer is emptied, the requested count has been
00165                   reached, or a CR (ENTER) code has been found. The copied
00166                   characters should, of course, be removed from the ring buffer.
00167                   Either input interrupts or all interrupts should be disabled
00168                   during the copying. */
00169
00170                   while((dcbPtr.ringbufcount >0) && (dcbPtr.inbuff-1 !='\r' && (
      dcbPtr.indone >= *(dcbPtr.incount)))){
00171                                   *((dcbPtr.inbuff)) = dcbPtr.ringbuf[dcbPtr.
      ringbufout];
00172                                   dcbPtr.indone++;
00173                                   dcbPtr.inbuff++;
00174                                   dcbPtr.ringbufout = (dcbPtr.ringbufout+1)%size;
00175                                   dcbPtr.ringbufcount--;
00176
00177                           } //end while
00178
00179                           enable(); //enable interrupts
00180
00181                   //the requestor buffer is not yet full
00182                   if(dcbPtr.indone < *(dcbPtr.incount))
00183                           return 0;
00184                   if(*(dcbPtr.inbuff-1) == '\r')
00185                           *(dcbPtr.inbuff-1) = '\0';
00186                   else
00187                           *dcbPtr.inbuff = '\0';
00188
00189                   //Reset the DCB status to idle, set the event flag, and
00190                   //return the actual count to the requestor's variable.
00191                   dcbPtr.status = IDLE; //status back to IDLE
00192                   *dcbPtr.flag_ptr = SET; //the event is over
00193                   *dcbPtr.incount = dcbPtr.indone;
00194            return 0;
00195           }
```

```
00196
00197         int com_write (char *buf_p,int *count_p){
00198                 int mask;
00199                 //Ensure that the input parameters are valid.
00200                 if(dcbPtr.flag != OPEN) //check if device is open
00201                         return WRITE_PORT_NOT_OPEN;
00202                 if(dcbPtr.status != IDLE) //check if device is idle
00203                         return WRITE_DEV_BUSY;
00204                 if(buf_p == NULL) //check if buffer is empty
00205                         return WRITE_INV_BUFF_ADD;
00206                 if(count_p == NULL) //check pointer is null
00207                         return WRITE_INV_COUNT;
00208
00209                 //Install the buffer pointer and counters in the DCB, and set the
       current status to writing.
00210                 dcbPtr.outbuff = buf_p;
00211                 dcbPtr.outcount = count_p;
00212                 dcbPtr.outdone = 0;
00213                 dcbPtr.status = WRITE;
00214                 //Clear the caller's event flag.
00215                 *dcbPtr.flag_ptr = FLAG_CLEAR;
00216
00217                 //Get the first character from the requestor's buffer and store i
      t in the output register.
00218                 outportb(BASE, *dcbPtr.outbuff);
00219                 dcbPtr.outbuff++;
00220                 dcbPtr.outdone++;
00221
00222
00223                  //Enable write interrupts by setting bit 1 of the Interrupt Enab
      le register.
00224                  //This must be done by setting the register to the logical or of
       its previous
00225                  //contents and 0x02.
00226
00227                 mask = inportb(INT_EN);
00228                 mask = mask | 0x02;
00229                 outportb(INT_EN,mask);
00230
00231         return 0;
00232         }
00233
```

4.27 src/MPX_R5.h File Reference

Data Structures

- struct device

Defines

- #define INT_ID 0X0C
- #define BASE 0x3F8
- #define INT_EN BASE+1
- #define BRD_LSB BASE
- #define BRD_MSB BASE+1
- #define INT_ID_REG BASE+2
- #define LC BASE+3
- #define MC BASE+4

- #define LS BASE+5
- #define MS BASE+6
- #define PIC_MASK 0X21
- #define PIC_CMD 0x20
- #define EOI 0x20
- #define WHATINTERRUPTBIT 0x07
- #define size 256
- #define OPEN 1
- #define CLOSED 0
- #define SET 1
- #define FLAG_CLEAR 0
- #define IDLE 0
- #define READ 1
- #define WRITE 2
- #define NO_ERROR 0
- #define INV_FLAG -101
- #define INV_BAUD -102
- #define PORT_ALREADY_OPEN -103
- #define SERIAL_PORT_NOT_OPEN -201
- #define READ_PORT_NOT_OPEN -301
- #define READ_INV_BUFF_ADD -302
- #define READ_INV_COUNT -303
- #define READ_DEV_BUSY -304
- #define WRITE_PORT_NOT_OPEN -401
- #define WRITE_INV_BUFF_ADD -402
- #define WRITE_INV_COUNT -403
- #define WRITE_DEV_BUSY -404

Typedefs

- typedef struct device DCB

Functions

- int com_open (int *, int)
- int com_close (void)
- int com_read (char *, int *)
- int com_write (char *, int *)
- void interrupt level1 ()
- void level2Write ()
- void level2Read ()

Variables

- int callInt = 0
- int num
- DCB dcbPtr
- void interrupt(* oldfunc)(void)

4.27.1 Define Documentation

4.27.1.1 #define BASE 0x3F8

Definition at line 3 of file MPX_R5.h.

4.27.1.2 #define BRD_LSB BASE

Definition at line 5 of file MPX_R5.h.

4.27.1.3 #define BRD_MSB BASE+1

Definition at line 6 of file MPX_R5.h.

4.27.1.4 #define CLOSED 0

Definition at line 24 of file MPX_R5.h.

4.27.1.5 #define EOI 0x20

Definition at line 14 of file MPX_R5.h.

4.27.1.6 #define FLAG_CLEAR 0

Definition at line 26 of file MPX_R5.h.

4.27.1.7 #define IDLE 0

Definition at line 27 of file MPX_R5.h.

4.27.1.8 #define INT_EN BASE+1

Definition at line 4 of file MPX_R5.h.

4.27.1.9 #define INT_ID 0X0C

Definition at line 2 of file MPX_R5.h.

4.27.1.10 #define INT_ID_REG BASE+2

Definition at line 7 of file MPX_R5.h.

4.27.1.11 #define INV_BAUD -102

Definition at line 34 of file MPX_R5.h.

4.27.1.12 #define INV_FLAG -101

Definition at line 33 of file MPX_R5.h.

4.27.1.13 #define LC BASE+3

Definition at line 8 of file MPX_R5.h.

4.27.1.14 #define LS BASE+5

Definition at line 10 of file MPX_R5.h.

4.27.1.15 #define MC BASE+4

Definition at line 9 of file MPX_R5.h.

4.27.1.16 #define MS BASE+6

Definition at line 11 of file MPX_R5.h.

4.27.1.17 #define NO_ERROR 0

Definition at line 32 of file MPX_R5.h.

4.27.1.18 #define OPEN 1

Definition at line 23 of file MPX_R5.h.

4.27.1.19 #define PIC_CMD 0x20

Definition at line 13 of file MPX_R5.h.

4.27.1.20 #define PIC_MASK 0X21

Definition at line 12 of file MPX_R5.h.

4.27.1.21 #define PORT_ALREADY_OPEN -103

Definition at line 35 of file MPX_R5.h.

4.27.1.22 #define READ 1

Definition at line 28 of file MPX_R5.h.

4.27.1.23 #define READ_DEV_BUSY -304

Definition at line 40 of file MPX_R5.h.

4.27.1.24 #define READ_INV_BUFF_ADD -302

Definition at line 38 of file MPX_R5.h.

4.27.1.25 #define READ_INV_COUNT -303

Definition at line 39 of file MPX_R5.h.

4.27.1.26 #define READ_PORT_NOT_OPEN -301

Definition at line 37 of file MPX_R5.h.

4.27.1.27 #define SERIAL_PORT_NOT_OPEN -201

Definition at line 36 of file MPX_R5.h.

4.27.1.28 #define SET 1

Definition at line 25 of file MPX_R5.h.

4.27.1.29 #define size 256

Definition at line 20 of file MPX_R5.h.

4.27.1.30 #define WHATINTERRUPTBIT 0x07

Definition at line 17 of file MPX_R5.h.

4.27.1.31 #define WRITE 2

Definition at line 29 of file MPX_R5.h.

4.27.1.32 #define WRITE_DEV_BUSY -404

Definition at line 44 of file MPX_R5.h.

4.27.1.33 #define WRITE_INV_BUFF_ADD -402

Definition at line 42 of file MPX_R5.h.

4.27.1.34 #define WRITE_INV_COUNT -403

Definition at line 43 of file MPX_R5.h.

4.27.1.35 #define WRITE_PORT_NOT_OPEN -401

Definition at line 41 of file MPX_R5.h.

4.27.2 Typedef Documentation

4.27.2.1 typedef struct device DCB

4.27.3 Function Documentation

4.27.3.1 int com_close (void)

Definition at line 49 of file MPX_R5.C.

```
                    {
        int mask;

        if(dcbPtr.flag != OPEN) //check that port is open
                return SERIAL_PORT_NOT_OPEN;

        dcbPtr.flag=CLOSED;
        disable(); //start enable
        mask = inportb(PIC_MASK);
        mask = mask | 0x10;
        outportb(PIC_MASK, mask);
        enable(); //end enable

        outportb( MS,0x00); // clears Modem Control Status
        outportb( INT_EN,0x00); // clears int_en
        setvect( INT_ID, oldfunc); //restore Microsoft interupt

        return 0;

}
```

4.27.3.2 int com_open (int *, int)

Definition at line 6 of file MPX_R5.C.

```
                                        {
        long brd;
        int mask;

        if(eflag == NULL)
                return INV_FLAG; // invalid flag
        if(baudrate <= 0)
                return INV_BAUD;  // invalid baud
        if(dcbPtr.flag==OPEN) // Make sure that the device is not open.
                return PORT_ALREADY_OPEN;

        dcbPtr.flag = OPEN;
        dcbPtr.flag_ptr = eflag;
        dcbPtr.status = IDLE;
```

```
            dcbPtr.ringbufin = 0;
            dcbPtr.ringbufout = 0;
            dcbPtr.ringbufcount = 0;

            oldfunc = getvect(INT_ID); //get the vector of the Windows compor
  t interupt handler
            setvect(INT_ID, &level1); //level1 is interrupt handler
            brd = 115200 / (long) baudrate; //calculate baud rate divisor


            outportb(LC, 0x80); //store 0x80 in line control register
            outportb(BRD_LSB, brd & 0xFF); //is Baud rate devisor LSB
            outportb(BRD_MSB,(brd>>8) & 0xFF); //is Baud rate devisor MSB
            outportb(LC, 0x03); //store 0x03 in line control register

            disable(); // disable interupts
            mask = inportb(PIC_MASK);
            mask = mask & 0xEF;
            outportb(PIC_MASK, mask);
            enable(); // enable interupts

            //enable level for COM1 in PIC Mask register
            //Store 0x08 in modem control register
            outportb( MC, 0x08);//enables serial interrupts
            //store 0x01 in interrupt enable register
            outportb( INT_EN, 0x01);//enables input ready interrupts

return 0; // return zero if no error.
    }
```

4.27.3.3 int com_read (char *, int *)

Definition at line 143 of file MPX_R5.C.

```
                                        {
            //Validate the supplied parameters.
            if(dcbPtr.flag != OPEN) //check if device is open
                    return READ_PORT_NOT_OPEN;
            if(dcbPtr.status != IDLE) //check if device is idle
                    return READ_DEV_BUSY;
            if( buf_p == NULL) //check if buffer is empty
                    return READ_INV_BUFF_ADD;
            if( &count_p == NULL) //check if count pointer is null
                    return READ_INV_COUNT;
  // Initialize the input buffer variables (not the ring buffer!) and set t
  he status to reading.
            dcbPtr.inbuff = buf_p;
            dcbPtr.incount = count_p;
            dcbPtr.indone = 0;

            *(dcbPtr.flag_ptr) = FLAG_CLEAR; //clear event flag

            disable(); //disable interrupts
            dcbPtr.status=READ; //we are now reading

            /* Copy characters from the ring buffer to the requestor's buffer
  ,
            until the ring buffer is emptied, the requested count has been
            reached, or a CR (ENTER) code has been found. The copied
            characters should, of course, be removed from the ring buffer.
            Either input interrupts or all interrupts should be disabled
            during the copying. */

            while((dcbPtr.ringbufcount >0) && (dcbPtr.inbuff-1 !='\r' && (
```

```
dcbPtr.indone >= *(dcbPtr.incount))){
                        *((dcbPtr.inbuff)) = dcbPtr.ringbuf[dcbPtr.
ringbufout];
                        dcbPtr.indone++;
                        dcbPtr.inbuff++;
                        dcbPtr.ringbufout = (dcbPtr.ringbufout+1)%size;
                        dcbPtr.ringbufcount--;

                } //end while

                enable(); //enable interrupts

        //the requestor buffer is not yet full
        if(dcbPtr.indone < *(dcbPtr.incount))
                return 0;
        if(*(dcbPtr.inbuff-1) == '\r')
                *(dcbPtr.inbuff-1) = '\0';
        else
                *dcbPtr.inbuff = '\0';

        //Reset the DCB status to idle, set the event flag, and
        //return the actual count to the requestor's variable.
        dcbPtr.status = IDLE; //status back to IDLE
        *dcbPtr.flag_ptr = SET; //the event is over
        *dcbPtr.incount = dcbPtr.indone;
  return 0;
 }
```

4.27.3.4 int com_write (char *, int *)

Definition at line 197 of file MPX_R5.C.

```
                                        {
        int mask;
        //Ensure that the input parameters are valid.
        if(dcbPtr.flag != OPEN) //check if device is open
                return WRITE_PORT_NOT_OPEN;
        if(dcbPtr.status != IDLE) //check if device is idle
                return WRITE_DEV_BUSY;
        if(buf_p == NULL) //check if buffer is empty
                return WRITE_INV_BUFF_ADD;
        if(count_p == NULL) //check pointer is null
                return WRITE_INV_COUNT;

        //Install the buffer pointer and counters in the DCB, and set the
 current status to writing.
        dcbPtr.outbuff = buf_p;
        dcbPtr.outcount = count_p;
        dcbPtr.outdone = 0;
        dcbPtr.status = WRITE;
        //Clear the caller's event flag.
        *dcbPtr.flag_ptr = FLAG_CLEAR;

        //Get the first character from the requestor's buffer and store i
t in the output register.
        outportb(BASE, *dcbPtr.outbuff);
        dcbPtr.outbuff++;
        dcbPtr.outdone++;


         //Enable write interrupts by setting bit 1 of the Interrupt Enab
le register.
         //This must be done by setting the register to the logical or of
 its previous
```

```
                //contents and 0x02.

               mask = inportb(INT_EN);
               mask = mask | 0x02;
               outportb(INT_EN,mask);

       return 0;
       }
```

4.27.3.5 void interrupt level1 ( )

Definition at line 70 of file MPX_R5.C.

```
                              {
               if(dcbPtr.flag != OPEN){
                       outportb(PIC_CMD, EOI);//clear interupt PIC command regis
    ter
                       return;
               } else{
                       num = ((inportb(INT_ID_REG) & WHATINTERRUPTBIT));
                       if (num == 2) // 0000 0010 : write interrupt
                               level2Write();
                       if (num == 4) // 0000 0100 : read interrupt
                               level2Read();
                       outportb(PIC_CMD, EOI); //clear interupt PIC command regi
    ster
               }
   return;

      }
```

4.27.3.6 void level2Read ( )

Definition at line 110 of file MPX_R5.C.

```
                 {
               char new;
               char ret = '\r';

               new=inportb(BASE); //Read a character from the input register.
               if ( new != ret )
               outportb(BASE, new);// ECHO BACK
               //If the current status is not reading, store the character in th
    e ring buffer.
               if(dcbPtr.status != READ){
                       if(dcbPtr.ringbufcount != size){
                               dcbPtr.ringbuf[dcbPtr.ringbufin]= new;
                               dcbPtr.ringbufin = (dcbPtr.ringbufin+1)%size;
                               dcbPtr.ringbufcount++;
                       }
               }else{ //status is reading
                       *dcbPtr.inbuff = new;
                       dcbPtr.inbuff++;
                       dcbPtr.indone++;
                               //If the count is not completed and the character
     is not CR, return. Do not signal completion.
                       if(new== '\r' || (dcbPtr.indone ) >= *(dcbPtr.incount)){
                               if(*(dcbPtr.inbuff-1) == '\r'){
                                       *(dcbPtr.inbuff-1) = '\0';
                               }else{
                                       *dcbPtr.inbuff = '\0';
```

```
                        }
                        *dcbPtr.incount = dcbPtr.indone;
                        dcbPtr.status =IDLE;
                        *dcbPtr.flag_ptr = SET;
        }//end if
}//end else

}
```

4.27.3.7 void level2Write ( )

Definition at line 86 of file MPX_R5.C.

```
                {

        int mask;

        if(dcbPtr. status != WRITE)
                return; //Ignore the interrupt and return

        if(dcbPtr.outdone < *dcbPtr.outcount){
                outportb(BASE, *dcbPtr.outbuff);
                dcbPtr.outbuff++;
                dcbPtr.outdone++;
                return;
        }else{

                dcbPtr.status = IDLE;
                *dcbPtr.flag_ptr = SET;

                mask = inportb(INT_EN);
                mask = mask&~0x02;
                outportb(INT_EN, mask);
                return;
        }
}
```

4.27.4 Variable Documentation

4.27.4.1 int callInt = 0

Definition at line 47 of file MPX_R5.h.

4.27.4.2 DCB dcbPtr

Definition at line 67 of file MPX_R5.h.

4.27.4.3 int num

Definition at line 48 of file MPX_R5.h.

4.27.4.4 void interrupt(* oldfunc)(void)

Definition at line 77 of file MPX_R5.h.

4.28 src/MPX_R5.h

```
00001         //COM1 Addresses
00002 #define INT_ID 0X0C // interupt ID for windows interupt table
00003 #define BASE 0x3F8  //com1 base address
00004 #define INT_EN BASE+1  // interupt enable
00005 #define BRD_LSB BASE // LSB Baud Rate Devisor
00006 #define BRD_MSB BASE+1 // MSB Baud Rate Devisor
00007 #define INT_ID_REG BASE+2
00008 #define LC BASE+3 // Line control register
00009 #define MC BASE+4 // Modem Control Register
00010 #define LS BASE+5 // Line control status
00011 #define MS BASE+6 // Modem Control status
00012 #define PIC_MASK 0X21 // Programmable Interupt Controler Mask
00013 #define PIC_CMD 0x20  // Programmable Interupt Controler Command
00014 #define EOI 0x20
00015
00016 //Used in interrupt1
00017 #define WHATINTERRUPTBIT 0x07
00018
00019 //ring buffer size
00020 #define size 256
00021
00022 //Flag states
00023 #define OPEN 1 //device open
00024 #define CLOSED 0 //device closed
00025 #define SET 1 //event flag set
00026 #define FLAG_CLEAR 0 //event flag cleared
00027 #define IDLE   0
00028 #define READ   1
00029 #define WRITE  2
00030
00031 //Error values returned
00032 #define NO_ERROR 0 //no error
00033 #define INV_FLAG -101 //invalid even flad pointer
00034 #define INV_BAUD -102 //invalid baud rate divisor
00035 #define PORT_ALREADY_OPEN -103 //port already open
00036 #define SERIAL_PORT_NOT_OPEN -201 //serial port not open
00037 #define READ_PORT_NOT_OPEN -301 //port not open
00038 #define READ_INV_BUFF_ADD -302 //invalid buffer address
00039 #define READ_INV_COUNT -303 //invalid count address or count value
00040 #define READ_DEV_BUSY -304 //device is busy
00041 #define WRITE_PORT_NOT_OPEN -401
00042 #define WRITE_INV_BUFF_ADD -402
00043 #define WRITE_INV_COUNT -403
00044 #define WRITE_DEV_BUSY -404
00045
00046 //global variables
00047 int callInt=0;
00048 int num;
00049
00050 typedef struct device{
00051         int flag; //indicate if device is open or closed.
00052         int *flag_ptr; //pointer to event flag
00053         int  status; //status code: IDLE, READ, WRITE
00054         char *inbuff; //pointer to requester's buffer; read data placed here
00055         int *incount; //max number of chars can be placed in requester's buffer
00056         int indone; //number of chars that have been placed in requester's buffer

00057         char *outbuff;
00058         int *outcount;
00059         int outdone;
00060         char ringbuf[size]; //ring buffer
00061         int ringbufin; //where write next char will be placed
00062         int ringbufout; //where remove next char from buffer
00063         int ringbufcount; //number of stored but not read chars from buffer
00064 }DCB;
```

```
00065
00066 //typedef struct device DCB;
00067 DCB dcbPtr;
00068
00069 //prototypes
00070 int com_open(int *, int );
00071 int com_close(void);
00072 int com_read(char *, int *);
00073 int com_write(char *, int *);
00074 void interrupt level1();
00075 void level2Write();
00076 void level2Read();
00077 void interrupt (*oldfunc) (void);
00078
```

4.29 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.

4.29.1 Define Documentation

4.29.1.1 #define LINES_PER_PAGE 23

Definition at line 8 of file mpx_util.c.

4.29.2 Function Documentation

4.29.2.1 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;
        }
}
```

4.29.2.2 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;
}
```

4.29.2.3 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++;
```

```
        }
}
```

4.29.2.4 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++;
                }
        }
}
```

4.29.2.5 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(). */
}
```

4.29.2.6 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];
}
```

4.29.2.7 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);
}
```

4.29.2.8 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 */
        }
}
```

4.30 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 }
```

4.31 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.

4.31.1 Function Documentation

4.31.1.1 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;
        }
}
```

4.31.1.2 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;
}
```

4.31.1.3 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++;
        }
}
```

4.31.1.4 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++;
                }
        }
}
```

4.31.1.5 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(). */
}
```

4.31.1.6 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];
}
```

4.31.1.7 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);
}
```

4.31.1.8 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 */
        }
}
```

4.32 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
```

4.33 src/procs-r3.c File Reference

```
#include "mpx_supt.h"
```

Defines

- #define RC_1 1
- #define RC_2 2
- #define RC_3 3
- #define RC_4 4
- #define RC_5 5

Functions

- void test1_R3 ()
- void test2_R3 ()
- void test3_R3 ()
- void test4_R3 ()
- void test5_R3 ()

4.33.1 Define Documentation

4.33.1.1 #define RC_1 1

Definition at line 43 of file procs-r3.c.

4.33.1.2 #define RC_2 2

Definition at line 44 of file procs-r3.c.

4.33.1.3 #define RC_3 3

Definition at line 45 of file procs-r3.c.

4.33.1.4 #define RC_4 4

Definition at line 46 of file procs-r3.c.

4.33.1.5 #define RC_5 5

Definition at line 47 of file procs-r3.c.

4.33.2 Function Documentation

4.33.2.1 void test1_R3 ( )

Definition at line 83 of file procs-r3.c.

4.33.2.2 void test2_R3 ( )

Definition at line 106 of file procs-r3.c.

4.33.2.3 void test3_R3 ( )

Definition at line 129 of file procs-r3.c.

4.33.2.4 void test4_R3 ( )

Definition at line 152 of file procs-r3.c.

4.33.2.5 void test5_R3 ( )

Definition at line 175 of file procs-r3.c.

4.34 src/procs-r3.c

```
00001 /**********************************************************
00002         MPX: The MultiProgramming eXecutive
00003         Project to Accompany
00004         A Practical Approach to Operating Systems
00005         Malcolm G. Lane & James D. Mooney
00006         Copyright 1993, P.W.S. Kent Publishing Co., Boston, MA.
00007
00008         File Name: procs-r3.c
00009
00010         Author: M.G.Lane, J. Mooney
00011         Version: 2.0
00012         Date: 01/03/93
00013
00014         Purpose: Process Management command procedures
00015
00016
00017         Environments: This file is system independent.
00018
00019         Procedures:
00020                               test1_R3 - test process
```

```
00021                               test2_R3 - test process
00022                               test3_R3 - test process
00023                               test4_R3 - test process
00024                               test5_R3 - test process
00025
00026 ****************************************************************************
00027
00028   Change Log:
00029
00030         05/16/88  mgl   Initial Version
00031         07/17/88  mgl   Final documentation changes
00032         08/13/92  jdm   Update for Version 2.0
00033         12/23/92  jdm   Revised count values
00034         12/28/92  jdm   changed file name, moved to support
00035         01/03/93  jdm   changed test procedure names
00036
00037 ****************************************************************************/
00038
00039 #include "mpx_supt.h"
00040
00041 /* loop counts */
00042
00043 #define RC_1 1
00044 #define RC_2 2
00045 #define RC_3 3
00046 #define RC_4 4
00047 #define RC_5 5
00048
00049
00050
00051
00052
00053
00054 /*
00055         Procedures: testx_R3 (x = 1, 2, 3, 4, 5)
00056
00057         Purpose: test processes for Module R3
00058
00059
00060         Parameters: none
00061
00062         Return value: none
00063
00064         Calls:  sys_req
00065                 printf
00066
00067         Globals: none
00068
00069         Algorithm:
00070
00071            Each process prints a message to the screen and gives up
00072            control to the dispatcher using sys_req.  Each process
00073            loops a certain number of times, displaying a message to
00074            the screen inside the loop.  (test1 loops 5 times, test2
00075            loops 10, test3 loops 15, test4 loops 20, and test5 loops
00076            25 times).  Each test process eventually requests
00077            termination.  If a dispatcher dispatches a test process
00078            after it requested termination, it prints a message
00079            indicating so, and the process starts over.
00080 */
00081
00082
00083 void test1_R3()
00084 {
00085         int ix; /* loop index */
00086
00087         /* repeat forever if termination fails */
```

```
00088         while (TRUE) {
00089
00090                 /* loop for the prescribed number of times */
00091                 for (ix=1; ix <= RC_1; ix++) {
00092
00093                         /* give up control to the dispatcher */
00094                         printf("test1 dispatched; loop count = %d\n",ix);
00095                         sys_req(IDLE, NO_DEV, NULL, 0);
00096                 }
00097
00098                 /* request termination */
00099                 sys_req(EXIT, NO_DEV, NULL, 0);
00100
00101                 /* display error message if dispatched again */
00102                 printf ("test1 dispatched after it exited!!!\n");
00103         }
00104  }
00105
00106 void test2_R3()
00107 {
00108         int ix; /* loop index */
00109
00110         /* repeat forever if termination fails */
00111         while (TRUE) {
00112
00113                 /* loop for the prescribed number of times */
00114                 for (ix=1; ix <= RC_2; ix++) {
00115
00116                         /* give up control to the dispatcher */
00117                         printf("test2 dispatched; loop count = %d\n",ix);
00118                         sys_req(IDLE, NO_DEV, NULL, 0);
00119                 }
00120
00121                 /* request termination */
00122                 sys_req(EXIT, NO_DEV, NULL, 0);
00123
00124                 /* display error message if dispatched again */
00125                 printf ("test2 dispatched after it exited!!!\n");
00126         }
00127  }
00128
00129 void test3_R3()
00130 {
00131         int ix; /* loop index */
00132
00133         /* repeat forever if termination fails */
00134         while (TRUE) {
00135
00136                 /* loop for the prescribed number of times */
00137                 for (ix=1; ix <= RC_3; ix++) {
00138
00139                         /* give up control to the dispatcher */
00140                         printf("test3 dispatched; loop count = %d\n",ix);
00141                         sys_req(IDLE, NO_DEV, NULL, 0);
00142                 }
00143
00144                 /* request termination */
00145                 sys_req(EXIT, NO_DEV, NULL, 0);
00146
00147                 /* display error message if dispatched again */
00148                 printf ("test3 dispatched after it exited!!!\n");
00149         }
00150  }
00151
00152 void test4_R3()
00153 {
00154         int ix; /* loop index */
```

```
00155
00156         /* repeat forever if termination fails */
00157         while (TRUE) {
00158
00159                 /* loop for the prescribed number of times */
00160                 for (ix=1; ix <= RC_4; ix++) {
00161
00162                         /* give up control to the dispatcher */
00163                         printf("test4 dispatched; loop count = %d\n",ix);
00164                         sys_req(IDLE, NO_DEV, NULL, 0);
00165                 }
00166
00167                 /* request termination */
00168                 sys_req(EXIT, NO_DEV, NULL, 0);
00169
00170                 /* display error message if dispatched again */
00171                 printf ("test4 dispatched after it exited!!!\n");
00172         }
00173  }
00174
00175 void test5_R3()
00176 {
00177         int ix; /* loop index */
00178
00179         /* repeat forever if termination fails */
00180         while (TRUE) {
00181
00182                 /* loop for the prescribed number of times */
00183                 for (ix=1; ix <= RC_5; ix++) {
00184
00185                         /* give up control to the dispatcher */
00186                         printf("test5 dispatched; loop count = %d\n",ix);
00187                         sys_req(IDLE, NO_DEV, NULL, 0);
00188                 }
00189
00190                 /* request termination */
00191                 sys_req(EXIT, NO_DEV, NULL, 0);
00192
00193                 /* display error message if dispatched again */
00194                 printf ("test5 dispatched after it exited!!!\n");
00195         }
00196  }
00197
00198 /* END OF FILE */
```

4.35 src/trmdrive.c File Reference

```
#include <dos.h>
#include "mpx_supt.h"
#include "trmdrive.h"
```

Data Structures

- struct context

Defines

- #define PIC_CMD 0x20
- #define PIC_MASK 0x21

- #define KBD_LEVEL 1
- #define SET 1
- #define RESET 0
- #define CR 0x0D
- #define LF 0x0A
- #define BS 0x08
- #define ESC 0x1B
- #define DEV_IDLE 0
- #define DEV_READ 1
- #define DEV_WRITE 2
- #define KBD_INTNUM (0x08 + KBD_LEVEL)
- #define OPEN_FILE 0x3D
- #define CLOSE_FILE 0x3E
- #define WRITE_FILE 0x40
- #define GET_CHAR 0x06
- #define WRITE_ONLY 0x01
- #define MAX_XPOS 79
- #define MAX_YPOS 23

Typedefs

- typedef unsigned short word
- typedef unsigned char byte
- typedef struct context context

Functions

- void interrupt kbd_ihand (void)
- void clear_scr (void)
- int goto_xy (int xval, int yval)
- void out_char (char ch)
- int trm_open (int *ef_p)
- trm_close (void)
- int trm_read (char *buf_p, int *count_p)
- int trm_write (char *buf_p, int *count_p)
- int trm_clear (void)
- int trm_gotoxy (int xval, int yval)
- void trm_getc (void)

Variables

- union REGS regs
- struct SREGS segs
- struct {
  flag open
  int status
  int * eflag_p
  char * out_buf_p
  int * out_count_p

int out_max
int out_ctr
char * in_buf_p
int * in_count_p
int in_max
int in_ctr
} dcb_trm

- void interrupt(* old_kbhand_p)(void) = NULL
- int con_handle
- int pendc = 0

4.35.1 Define Documentation

4.35.1.1 #define BS 0x08

Definition at line 86 of file trmdrive.c.

4.35.1.2 #define CLOSE_FILE 0x3E

Definition at line 99 of file trmdrive.c.

4.35.1.3 #define CR 0x0D

Definition at line 84 of file trmdrive.c.

4.35.1.4 #define DEV_IDLE 0

Definition at line 90 of file trmdrive.c.

4.35.1.5 #define DEV_READ 1

Definition at line 91 of file trmdrive.c.

4.35.1.6 #define DEV_WRITE 2

Definition at line 92 of file trmdrive.c.

4.35.1.7 #define ESC 0x1B

Definition at line 87 of file trmdrive.c.

4.35.1.8 #define GET_CHAR 0x06

Definition at line 101 of file trmdrive.c.

4.35.1.9 #define KBD_INTNUM (0x08 + KBD_LEVEL)

Definition at line 95 of file trmdrive.c.

4.35.1.10 #define KBD_LEVEL 1

Definition at line 78 of file trmdrive.c.

4.35.1.11 #define LF 0x0A

Definition at line 85 of file trmdrive.c.

4.35.1.12 #define MAX_XPOS 79

Definition at line 105 of file trmdrive.c.

4.35.1.13 #define MAX_YPOS 23

Definition at line 106 of file trmdrive.c.

4.35.1.14 #define OPEN_FILE 0x3D

Definition at line 98 of file trmdrive.c.

4.35.1.15 #define PIC_CMD 0x20

Definition at line 74 of file trmdrive.c.

4.35.1.16 #define PIC_MASK 0x21

Definition at line 75 of file trmdrive.c.

4.35.1.17 #define RESET 0

Definition at line 83 of file trmdrive.c.

4.35.1.18 #define SET 1

Definition at line 82 of file trmdrive.c.

4.35.1.19 #define WRITE_FILE 0x40

Definition at line 100 of file trmdrive.c.

4.35.1.20 #define WRITE_ONLY 0x01

Definition at line 103 of file trmdrive.c.

4.35.2 Typedef Documentation

4.35.2.1 typedef unsigned char byte

Definition at line 62 of file trmdrive.c.

4.35.2.2 typedef struct context context

4.35.2.3 typedef unsigned short word

Definition at line 61 of file trmdrive.c.

4.35.3 Function Documentation

4.35.3.1 void clear_scr (void)

Definition at line 585 of file trmdrive.c.

4.35.3.2 int goto_xy (int *xval*, int *yval*)

Definition at line 609 of file trmdrive.c.

4.35.3.3 void interrupt kbd_ihand (void)

Definition at line 447 of file trmdrive.c.

4.35.3.4 void out_char (char *ch*)

Definition at line 652 of file trmdrive.c.

4.35.3.5 int trm_clear (void)

Definition at line 393 of file trmdrive.c.

4.35.3.6 trm_close (void)

Definition at line 235 of file trmdrive.c.

4.35.3.7 void trm_getc (void)

Definition at line 480 of file trmdrive.c.

4.35.3.8 int trm_gotoxy (int *xval*, int *yval*)

Definition at line 422 of file trmdrive.c.

4.35.3.9 int trm_open (int * *ef_p*)

Definition at line 176 of file trmdrive.c.

4.35.3.10 int trm_read (char * *buf_p*, int * *count_p*)

Definition at line 282 of file trmdrive.c.

4.35.3.11 int trm_write (char * *buf_p*, int * *count_p*)

Definition at line 331 of file trmdrive.c.

4.35.4 Variable Documentation

4.35.4.1 int con_handle

Definition at line 131 of file trmdrive.c.

4.35.4.2 struct { ... } dcb_trm

4.35.4.3 int* eflag_p

Definition at line 116 of file trmdrive.c.

4.35.4.4 char* in_buf_p

Definition at line 121 of file trmdrive.c.

4.35.4.5 int* in_count_p

Definition at line 122 of file trmdrive.c.

4.35.4.6 int in_ctr

Definition at line 124 of file trmdrive.c.

4.35.4.7 int in_max

Definition at line 123 of file trmdrive.c.

4.35.4.8 void interrupt(* old_kbhand_p)(void) = NULL

Definition at line 128 of file trmdrive.c.

4.35.4.9 flag open

Definition at line 114 of file trmdrive.c.

4.35.4.10 char* out_buf_p

Definition at line 117 of file trmdrive.c.

4.35.4.11 int* out_count_p

Definition at line 118 of file trmdrive.c.

4.35.4.12 int out_ctr

Definition at line 120 of file trmdrive.c.

4.35.4.13 int out_max

Definition at line 119 of file trmdrive.c.

4.35.4.14 int pendc = 0

Definition at line 134 of file trmdrive.c.

4.35.4.15 union REGS regs

Definition at line 109 of file trmdrive.c.

4.35.4.16 struct SREGS segs

Definition at line 110 of file trmdrive.c.

4.35.4.17 int status

Definition at line 115 of file trmdrive.c.

4.36 src/trmdrive.c

```
00001 /*****************************************************************
00002         MPX: The MultiProgramming eXecutive
00003         Project to Accompany
00004         A Practical Approach to Operating Systems
00005         Malcolm G. Lane & James D. Mooney
00006         Copyright 1993, P.W.S. Kent Publishing Co., Boston, MA.
00007
00008         File Name: trmdrive.c
00009
00010         Authors: M.G. Lane, A. Ghosal, J. Mooney
```

```
00011         Version: 2.1b
00012         Date: 11/10/93
00013
00014         Purpose: Terminal (Console) Driver
00015
00016         This module is a direct driver for keyboard input and
00017         screen output.  These devices are collectively called
00018         the "terminal."  Note that the screen output does *not*
00019         use interrupts.
00020
00021         This version does not support simultaneous input and
00022         output, or typeahead.  Note that the screen driver does
00023         *not* use interrupts, and does *not* call IO_complete.
00024
00025         This driver uses BIOS functions for screen output and
00026         keyboard input.  It should be portable across all
00027         keyboards and video modes.
00028
00029         Environments: IBM-PC, TURBO-C.
00030
00031         Procedures:     trm_open        open the terminal
00032                         trm_close       close the terminal
00033                         trm_read        begin a keyboard read
00034                         trm_getc        process keyboard characters
00035                         trm_write       perform a screen write
00036                         trm_clear       clear the screen
00037                         trm_gotoxy      set cursor position
00038                         kbd_ihand       handle keyboard interrupts
00039                         trm_getc        get processed characters
00040
00041
00042 ************************************************************************
00043
00044         Change Log:
00045
00046         04/21/88  akg   revise IOFIN parameters
00047         05/08/88  mgl   enhance driver, add tab support.
00048         05/09/88  mgl   change to far pointers for con_read and con_write
00049         08/02/88  mgl   enabled code for 8088, 8086 that had been
00050                               conditionally omitted.
00051         12/11/92  jdm   restructured for MPX 2.0
00052         12/30/92  jdm   corrected name conflict (clear_scr)
00053         03/30/93  jdm   corrected errors in trm_getc
00054         03/30/93  jdm   final version for V2.0b
00055         11/10/93  jdm   updated for large model; removed IO_complete
00056
00057 ************************************************************************/
00058
00059 #include <dos.h>
00060
00061 typedef unsigned short word;
00062 typedef unsigned char byte;
00063
00064 typedef struct context {
00065         word BP, DI, SI, DS, ES;
00066         word DX, CX, BX, AX;
00067         word IP, CS, FLAGS;
00068         } context;
00069
00070 #include "mpx_supt.h"
00071 #include "trmdrive.h"
00072
00073 /* define 8259 PIC ports */
00074 #define PIC_CMD         0x20
00075 #define PIC_MASK        0x21
00076
00077 /* keyboard interrupt level */
```

```
00078 #define KBD_LEVEL 1
00079
00080
00081 /* general definitions */
00082 #define SET             1
00083 #define RESET           0
00084 #define CR              0x0D
00085 #define LF              0x0A
00086 #define BS              0x08
00087 #define ESC             0x1B
00088
00089 /* DCB status codes */
00090 #define DEV_IDLE        0
00091 #define DEV_READ        1
00092 #define DEV_WRITE       2
00093
00094 /* define keyboard interrupt number */
00095 #define KBD_INTNUM      (0x08 + KBD_LEVEL)
00096
00097 /* MSDOS interrupt parameters */
00098 #define OPEN_FILE       0x3D
00099 #define CLOSE_FILE      0x3E
00100 #define WRITE_FILE      0x40
00101 #define GET_CHAR        0x06
00102
00103 #define WRITE_ONLY      0x01
00104
00105 #define MAX_XPOS        79
00106 #define MAX_YPOS        23
00107
00108 /* register structures for MS-DOS interrupts */
00109 union REGS regs;
00110 struct SREGS segs;
00111
00112 /* DCB structure */
00113 struct {
00114         flag    open;
00115         int     status;
00116         int     *eflag_p;
00117         char    *out_buf_p;
00118         int     *out_count_p;
00119         int     out_max;
00120         int     out_ctr;
00121         char    *in_buf_p;
00122         int     *in_count_p;
00123         int     in_max;
00124         int     in_ctr;
00125         } dcb_trm = {FALSE, DEV_IDLE};
00126
00127 /* ptr to saved keyboard interrupt vector */
00128 void interrupt (* old_kbhand_p) (void) = NULL;
00129
00130 /* MS-DOS handle for console (screen) output */
00131 int con_handle;
00132
00133 /* pending keyboard character count */
00134 int pendc = 0;
00135
00136 /*extern flag mpx_active;*/
00137
00138 /* declare keyboard interrupt handler */
00139 void interrupt kbd_ihand(void);
00140
00141 /* declare local procedures */
00142 void clear_scr (void);
00143 int goto_xy (int xval, int yval);
00144 void out_char(char ch);
```

```
00145
00146
00147 /*
00148         Procedure: trm_open
00149
00150         Purpose: initialize the terminal
00151
00152
00153         Parameters:     int *ef_p       ptr to event flag
00154
00155         Return value: error code, or zero if OK
00156
00157         Calls:  getvect, setvect
00158                 clear_scr
00159                 intdos(OPEN_FILE)
00160
00161         Globals: old_kbhand_p, dcb_trm, regs
00162
00163         Errors: ERR_TRM_OP_INVEFP invalid event flag parameter
00164                 ERR_TRM_OP_ALROPN device already open
00165                 ERR_TRM_OP_OPFAIL open failed
00166
00167         The keyboard is assumed to be already "open", with interrupts
00168         enabled.  We insert the MPX interrupt handler so keystrokes
00169         will be detected;  this will in turn call the MS-DOS handler
00170         for scan code processing, then collect the processed code.
00171
00172         The screen is accessed through MS-DOS handles.  We open a
00173         private handle for CON and clear the screen.  We assume that
00174         ANSI.SYS is loaded.
00175 */
00176 int trm_open (int *ef_p)
00177 {
00178         int     rval;
00179         int     err;
00180
00181         /* validate parameter */
00182         if (ef_p == NULL) return (ERR_TRM_OP_INVEFP);
00183
00184         /* if already open, return error code */
00185         if (dcb_trm.open) return (ERR_TRM_OP_ALROPN);
00186
00187         /* initialize DCB */
00188         dcb_trm.eflag_p = ef_p;
00189         dcb_trm.open = TRUE;
00190         dcb_trm.status = DEV_IDLE;
00191
00192         /* clear character counter */
00193         pendc = 0;
00194
00195         /* open CON for output, get handle */
00196         regs.h.ah = (byte) OPEN_FILE;
00197         regs.h.al = (byte) WRITE_ONLY;
00198         regs.x.dx = (word) "CON";
00199         rval = intdos(&regs, &regs);
00200         if (regs.x.cflag!=0) return(ERR_TRM_OP_OPFAIL);
00201         con_handle = rval;
00202
00203         /* save MS-DOS keyboard handler addr, setup MPX handler */
00204         if (old_kbhand_p==NULL) {
00205                 old_kbhand_p = getvect(KBD_INTNUM);
00206         }
00207         setvect(KBD_INTNUM,&kbd_ihand);
00208
00209         /* clear the screen */
00210         err = trm_clear();
00211         err = err;
```

```
00212
00213         return (OK);
00214 }
00215
00216
00217 /*
00218         Procedure: trm_close
00219
00220         Purpose: close the terminal
00221
00222
00223         Parameters:     none
00224
00225         Return value: error code, or zero if OK
00226
00227         Calls:  setvect
00228                 intdos (CLOSE_FILE)
00229
00230         Globals: old_kbhand_p, dcb_trm, regs
00231
00232         Errors: ERR_TRM_CL_NOTOPN       terminal not open
00233                 ERR_TRM_CL_CLFAIL       close failed
00234 */
00235 trm_close(void)
00236 {
00237         int     err;
00238
00239         /* if not open, return error code */
00240         if (dcb_trm.eflag_p == NULL) return(ERR_TRM_CL_NOTOPN);
00241
00242         /* clear the open flag */
00243         dcb_trm.open = FALSE;
00244
00245         /* restore MS-DOS interrupt vector */
00246         if (old_kbhand_p!=NULL) {
00247                 setvect(KBD_INTNUM, old_kbhand_p);
00248         }
00249         old_kbhand_p = NULL;
00250
00251         /* close CON */
00252         regs.h.ah = (byte) CLOSE_FILE;
00253         regs.x.bx = (word) con_handle;
00254         err = intdos(&regs, &regs);
00255         err = err;
00256         if (regs.x.cflag!=0) return(ERR_TRM_CL_CLFAIL);
00257
00258         return(OK);
00259 }
00260
00261
00262
00263 /*
00264         Procedure: trm_read
00265
00266         Purpose: initiate block input from the keyboard
00267
00268         Parameters:     char *buf_p     ptr to buffer
00269                         int *count_p ptr to count
00270
00271         Return value: error code, or zero if OK
00272
00273         Calls:  out_char
00274
00275         Globals: dcb_trm
00276
00277         Errors: ERR_TRM_RD_INVBUF invalid buffer parameter
00278                 ERR_TRM_RD_INVCNT invalid count parameter
```

```
00279                 ERR_TRM_RD_NOTOPN device not open
00280                 ERR_TRM_RD_DVBUSY device busy
00281 */
00282 int trm_read (char *buf_p, int *count_p)
00283 {
00284         /* check for valid parameters */
00285         if (buf_p == NULL) return (ERR_TRM_RD_INVBUF);
00286         if (count_p == NULL) return (ERR_TRM_RD_INVCNT);
00287         if (*count_p <= 0) return (ERR_TRM_RD_INVCNT);
00288
00289         /* check terminal status */
00290         if (dcb_trm.eflag_p == NULL) return(ERR_TRM_RD_NOTOPN);
00291         if (dcb_trm.status != DEV_IDLE) return(ERR_TRM_RD_DVBUSY);
00292
00293         /* setup DCB */
00294         dcb_trm.in_buf_p = buf_p;
00295         *dcb_trm.in_buf_p = NULCH;
00296         dcb_trm.in_count_p = count_p;
00297         dcb_trm.in_max = *count_p;
00298         dcb_trm.in_ctr = 0;
00299         dcb_trm.status = DEV_READ;
00300
00301         /* clear caller's event flag */
00302         *dcb_trm.eflag_p = RESET;
00303
00304         return (OK);
00305 }
00306
00307
00308 /*
00309         Procedure: trm_write
00310
00311         Purpose: perform block output to the screen
00312
00313         This routine is NOT interrupt driven, and does
00314         NOT call IO_complete!
00315
00316
00317         Parameters:     char *buf_p     ptr to buffer
00318                         int *count_p ptr to count
00319
00320         Return value: error code, or zero if OK
00321
00322         Calls:  out_char
00323
00324         Globals: dcb_trm
00325
00326         Errors: ERR_TRM_WR_INVBUF invalid buffer parameter
00327                 ERR_TRM_WR_INVCNT invalid count parameter
00328                 ERR_TRM_WR_NOTOPN device not open
00329                 ERR_TRM_WR_DVBUSY device busy
00330 */
00331 int trm_write (char *buf_p, int *count_p)
00332 {
00333
00334         char    ch;
00335
00336         /* check for valid parameters */
00337         if (buf_p == NULL) return (ERR_TRM_WR_INVBUF);
00338         if (count_p == NULL) return (ERR_TRM_WR_INVCNT);
00339         if (*count_p < 0) return (ERR_TRM_WR_INVCNT);
00340
00341         /* check terminal status */
00342         if (dcb_trm.eflag_p == NULL) return(ERR_TRM_WR_NOTOPN);
00343         if (dcb_trm.status != DEV_IDLE) return(ERR_TRM_WR_DVBUSY);
00344
00345         /* setup DCB */
```

```
00346         dcb_trm.out_buf_p = buf_p;
00347         dcb_trm.out_count_p = count_p;
00348         dcb_trm.out_max = *count_p;
00349         dcb_trm.out_ctr = 0;
00350         dcb_trm.status = DEV_WRITE;
00351 
00352         /* clear caller's event flag */
00353         *dcb_trm.eflag_p = RESET;
00354 
00355         /* output the characters */
00356         while (dcb_trm.out_ctr < dcb_trm.out_max) {
00357                 ch = *dcb_trm.out_buf_p++;
00358                 dcb_trm.out_ctr++;
00359                 out_char(ch);
00360                 if (ch==CR) out_char(LF);
00361                 if (ch==LF) out_char(CR);
00362         }
00363 
00364         /* reset DCB status */
00365         dcb_trm.status = DEV_IDLE;
00366 
00367         /* return count, set event flag */
00368         *dcb_trm.out_count_p = dcb_trm.out_ctr;
00369         *dcb_trm.eflag_p = SET;
00370 
00371         return (OK);
00372 }
00373 
00374 
00375 
00376 /*
00377         Procedure: trm_clear
00378 
00379         Purpose: clear the terminal screen
00380 
00381 
00382         Parameters:     none
00383 
00384         Return value: error code, or zero if OK
00385 
00386         Calls:  clear_scr
00387 
00388         Globals: none
00389 
00390         Errors: none
00391 */
00392 
00393 int trm_clear(void)
00394 {
00395         int err;
00396 
00397         err = goto_xy(0,0);
00398         err = err;
00399         clear_scr();
00400 
00401         return(OK);
00402 }
00403 
00404 /*
00405         Procedure: trm_gotoxy
00406 
00407         Purpose: position cursor
00408 
00409 
00410         Parameters:     int xval        requested x position (0 - 79}
00411                         int yval        requested y position (0 - 23)
00412 
```

```
00413         Return value: error code, or zero if OK
00414
00415         Calls:  gotoxy
00416
00417         Globals: none
00418
00419         Errors: ERR_TRM_XY_INVPOS invalid x or y parameter
00420 */
00421
00422 int trm_gotoxy(int xval, int yval)
00423 {
00424         int err;
00425
00426         err = goto_xy(xval, yval);
00427         if (err != OK) return(ERR_TRM_XY_INVPOS);
00428
00429         return(OK);
00430 }
00431
00432
00433 /*
00434         Procedure: kbd_ihand
00435
00436         Purpose: keyboard interrupt handler
00437
00438
00439         Parameters: none
00440
00441         Return value: none
00442
00443         Calls:  MSDOS keyboard handler
00444
00445         Globals: old_kbhand_p, pendc
00446 */
00447 void interrupt kbd_ihand(void)
00448 {
00449         /*if (mpx_active) return;*/
00450
00451         /* let MS-DOS process the character */
00452         (*old_kbhand_p)();
00453
00454         /* increment counter */
00455         pendc++;
00456
00457
00458 }
00459
00460
00461 /*
00462         Procedure: trm_getc
00463
00464         Purpose: process pending keyboard characters
00465
00466
00467         Parameters:     none
00468
00469         Return value: none
00470
00471         Calls:  intdos(GET_CHAR)
00472                 out_char
00473                 IO_complete
00474
00475         Globals: dcb_trm, regs
00476                 pendc
00477
00478         Errors: none
00479 */
```

```
00480 void trm_getc(void)
00481 {
00482         char    nextch;
00483         int     err;
00484         int     finish;
00485
00486
00487         /* process any pending characters until block finished */
00488         finish = FALSE;
00489         while ((pendc > 0) && !finish) {
00490
00491                 /* get the processed character, if any */
00492                 regs.h.ah = (byte) GET_CHAR;
00493                 regs.h.dl = 0xFF;
00494                 err = intdos(&regs, &regs);
00495                 err = err;
00496                 nextch = (char) regs.h.al;
00497
00498
00499                 /* if no character present, ignore */
00500                 if (regs.x.flags & 0x40){
00501                        /*pendc = 1;*/
00502                 }
00503
00504
00505                 /* if char = 0, get the function code & ignore */
00506                 else if (nextch==NULCH) {
00507                         regs.h.ah = (byte) GET_CHAR;
00508                         regs.h.dl = 0xFF;
00509                         err = intdos(&regs, &regs);
00510                         err = err;
00511                 }
00512
00513
00514                 else {
00515                         /* if CR, store newline & advance cursor */
00516                         if (nextch==CR) {
00517                                 out_char(CR);
00518                                 out_char(LF);
00519                                 *dcb_trm.in_buf_p++ = '\n';
00520                                 dcb_trm.in_ctr++;
00521                         }
00522
00523                         /* if backspace, delete prev. char, if any */
00524                         else if (nextch==BS) {
00525                                 if (dcb_trm.in_ctr > 0) {
00526                                         out_char(BS);
00527                                         out_char(' ');
00528                                         out_char(BS);
00529                                         dcb_trm.in_ctr--;
00530                                         dcb_trm.in_buf_p--;
00531                                 }
00532                         }
00533
00534                         /* otherwise, just store & echo */
00535                         else {
00536                                 out_char(nextch);
00537                                 *dcb_trm.in_buf_p++ = nextch;
00538                                 dcb_trm.in_ctr++;
00539                         }
00540
00541                         *dcb_trm.in_buf_p = NULCH;
00542
00543                         /* terminate on CR (ENTER) */
00544                         if (nextch == CR)  {
00545                                 finish = TRUE;
00546                         }
```

```
00547
00548                           /* otherwise, terminate if buffer full */
00549                           else if (dcb_trm.in_ctr >= dcb_trm.in_max) {
00550                                   out_char(CR);
00551                                   out_char(LF);
00552                                   finish = TRUE;
00553                           }
00554                   }
00555
00556           /* decrement character counter */
00557           pendc--;
00558           }
00559
00560           /* cleanup if terminating */
00561           if (finish) {
00562                   dcb_trm.status = DEV_IDLE;
00563                   *dcb_trm.in_count_p = dcb_trm.in_ctr;
00564                   *dcb_trm.eflag_p = SET;
00565           }
00566
00567           return;
00568 }
00569
00570
00571 /*
00572           Procedure: clear_scr
00573
00574           Purpose: clear the terminal screen
00575
00576
00577           Parameters: none
00578
00579           Return value: none
00580
00581           Calls:  out_char
00582
00583           Globals: none
00584 */
00585 void clear_scr(void)
00586 {
00587
00588           out_char(ESC);
00589           out_char('[');
00590           out_char('2');
00591           out_char('J');
00592
00593 }
00594
00595 /*
00596           Procedure: goto_xy
00597
00598           Purpose: position the cursor
00599
00600           Parameters:     int xval        horizontal position (0 - 79)
00601                           int yval        vertical position (0 - 23)
00602
00603           Return value: error code; zero if OK
00604
00605           Calls: out_char
00606
00607           Globals: none
00608 */
00609 int goto_xy(int xval, int yval)
00610 {
00611
00612           int xdh, xdl;
00613           int ydh, ydl;
```

```
00614         char digtab[10] = {'0','1','2','3','4','5','6','7','8','9'};
00615
00616         if ((xval < 0) || (xval > MAX_XPOS)) return(-1);
00617         if ((yval < 0) || (yval > MAX_YPOS)) return(-1);
00618
00619         xdh = (xval%100)/10;
00620         xdl = (xval%10);
00621         ydh = (yval%100)/10;
00622         ydl = (yval%10);
00623
00624         out_char(ESC);
00625         out_char('[');
00626         out_char(digtab[ydh]);
00627         out_char(digtab[ydl]);
00628         out_char(';');
00629         out_char(digtab[xdh]);
00630         out_char(digtab[xdl]);
00631         out_char('H');
00632
00633
00634
00635         return(OK);
00636 }
00637
00638 /*
00639         Procedure: out_char
00640
00641         Purpose: output a character
00642
00643
00644         Parameters: ch  character to output
00645
00646         Return value: none
00647
00648         Calls:  intdosx (WRITE_FILE)
00649
00650         Globals: none
00651 */
00652 void out_char(ch)
00653 char ch;
00654 {
00655         char    chbuf;
00656         int     err;
00657
00658         chbuf = ch;
00659         regs.h.ah = (byte) WRITE_FILE;
00660         regs.x.bx = con_handle;
00661         regs.x.cx = 1;
00662         regs.x.dx = FP_OFF(&chbuf);
00663         segs.ds = FP_SEG(&chbuf);
00664         segs.es = FP_SEG(&chbuf);
00665         err = intdosx(&regs, &regs, &segs);
00666         err = err;
00667 }
00668
00669 /* END OF FILE */
00670
00671
00672
```

4.37 src/trmdrive.h File Reference

Defines

- #define ERR_TRM_OP_INVEFP (-101)
- #define ERR_TRM_OP_ALROPN (-102)
- #define ERR_TRM_OP_OPFAIL (-103)
- #define ERR_TRM_CL_NOTOPN (-201)
- #define ERR_TRM_CL_CLFAIL (-202)
- #define ERR_TRM_RD_NOTOPN (-301)
- #define ERR_TRM_RD_INVBUF (-302)
- #define ERR_TRM_RD_INVCNT (-303)
- #define ERR_TRM_RD_DVBUSY (-304)
- #define ERR_TRM_WR_NOTOPN (-401)
- #define ERR_TRM_WR_INVBUF (-402)
- #define ERR_TRM_WR_INVCNT (-403)
- #define ERR_TRM_WR_DVBUSY (-404)
- #define ERR_TRM_XY_INVPOS (-601)

Functions

- int trm_open (int *ef_p)
- int trm_close (void)
- int trm_read (char *buf_p, int *count_p)
- void trm_getc (void)
- int trm_write (char *buf_p, int *count_p)
- int trm_clear (void)
- int trm_gotoxy (int xpos, int ypos)

4.37.1 Define Documentation

4.37.1.1 #define ERR_TRM_CL_CLFAIL (-202)

Definition at line 39 of file trmdrive.h.

4.37.1.2 #define ERR_TRM_CL_NOTOPN (-201)

Definition at line 38 of file trmdrive.h.

4.37.1.3 #define ERR_TRM_OP_ALROPN (-102)

Definition at line 35 of file trmdrive.h.

4.37.1.4 #define ERR_TRM_OP_INVEFP (-101)

Definition at line 34 of file trmdrive.h.

4.37.1.5 #define ERR_TRM_OP_OPFAIL (-103)

Definition at line 36 of file trmdrive.h.

4.37.1.6 #define ERR_TRM_RD_DVBUSY (-304)

Definition at line 44 of file trmdrive.h.

4.37.1.7 #define ERR_TRM_RD_INVBUF (-302)

Definition at line 42 of file trmdrive.h.

4.37.1.8 #define ERR_TRM_RD_INVCNT (-303)

Definition at line 43 of file trmdrive.h.

4.37.1.9 #define ERR_TRM_RD_NOTOPN (-301)

Definition at line 41 of file trmdrive.h.

4.37.1.10 #define ERR_TRM_WR_DVBUSY (-404)

Definition at line 49 of file trmdrive.h.

4.37.1.11 #define ERR_TRM_WR_INVBUF (-402)

Definition at line 47 of file trmdrive.h.

4.37.1.12 #define ERR_TRM_WR_INVCNT (-403)

Definition at line 48 of file trmdrive.h.

4.37.1.13 #define ERR_TRM_WR_NOTOPN (-401)

Definition at line 46 of file trmdrive.h.

4.37.1.14 #define ERR_TRM_XY_INVPOS (-601)

Definition at line 51 of file trmdrive.h.

4.37.2 Function Documentation

4.37.2.1 int trm_clear (void)

Definition at line 393 of file trmdrive.c.

4.37.2.2 int trm_close (void)

Definition at line 235 of file trmdrive.c.

4.37.2.3 void trm_getc (void)

Definition at line 480 of file trmdrive.c.

4.37.2.4 int trm_gotoxy (int *xpos*, int *ypos*)

Definition at line 422 of file trmdrive.c.

4.37.2.5 int trm_open (int * *ef_p*)

Definition at line 176 of file trmdrive.c.

4.37.2.6 int trm_read (char * *buf_p*, int * *count_p*)

Definition at line 282 of file trmdrive.c.

4.37.2.7 int trm_write (char * *buf_p*, int * *count_p*)

Definition at line 331 of file trmdrive.c.

4.38 src/trmdrive.h

```
00001 /************************************************************
00002         MPX: The MultiProgramming eXecutive
00003         Project to Accompany
00004         A Practical Approach to Operating Systems
00005         Malcolm G. Lane & James D. Mooney
00006         Copyright 1993, P.W.S. Kent Publishing Co., Boston, MA.
00007
00008         File Name: trmdrive.h
00009
00010         Authors: M.G. Lane, J. Mooney
00011         Version: 2.1b
00012         Date: 11/10/93
00013
00014         Purpose: Terminal Driver (Header file)
00015
00016         Environments: IBM-PC, TURBO-C.
00017
00018
00019
00020
00021 ************************************************************
00022   Change Log:
00023
00024         12/01/92  jdm   separated from prtdrive.c
00025         12/08/92  jdm   removed hand;er declaration
00026         12/11/92  jdm   converted to trmdrive.c
00027         03/19/93  jdm   final version for V2.0b
00028         11/10/93  jdm   updated for large model
```

```
00029
00030 ************************************************************/
00031
00032 /* device error codes */
00033
00034 #define ERR_TRM_OP_INVEFP       (-101)
00035 #define ERR_TRM_OP_ALROPN       (-102)
00036 #define ERR_TRM_OP_OPFAIL       (-103)
00037
00038 #define ERR_TRM_CL_NOTOPN       (-201)
00039 #define ERR_TRM_CL_CLFAIL       (-202)
00040
00041 #define ERR_TRM_RD_NOTOPN       (-301)
00042 #define ERR_TRM_RD_INVBUF       (-302)
00043 #define ERR_TRM_RD_INVCNT       (-303)
00044 #define ERR_TRM_RD_DVBUSY       (-304)
00045
00046 #define ERR_TRM_WR_NOTOPN       (-401)
00047 #define ERR_TRM_WR_INVBUF       (-402)
00048 #define ERR_TRM_WR_INVCNT       (-403)
00049 #define ERR_TRM_WR_DVBUSY       (-404)
00050
00051 #define ERR_TRM_XY_INVPOS       (-601)
00052
00053 /* driver function prototypes */
00054
00055 /* trm_open: open the terminal */
00056 /*      RETURNS: error code, or zero if ok */
00057 int trm_open (int *ef_p /* ptr to event flag */
00058         );
00059
00060 /* trm_close: close the terminal */
00061 /*      RETURNS: error code, or zero if ok */
00062 int trm_close (void);
00063
00064 /* trm_read: begin block keyboard input */
00065 /*      RETURNS: error code, or zero if ok */
00066 int trm_read (char *buf_p, /* ptr to buffer */
00067                 int *count_p /* ptr to count value */
00068         );
00069
00070 /* trm_getc: process keyboard characters */
00071 void trm_getc (void);
00072
00073
00074 /* trm_write: perform block screen output */
00075 /*      RETURNS: error code, or zero if ok */
00076 int trm_write (char *buf_p, /* ptr to buffer */
00077                 int *count_p /* ptr to count value */
00078         );
00079
00080 /* trm_clear: clear the screen */
00081 /*      RETURNS: error code, or zero if ok */
00082 int trm_clear (void);
00083
00084 /* trm_gotoxy: go to specified position */
00085 /*      RETURNS: error code, or zero if ok */
00086 int trm_gotoxy (int xpos, /* horizontal position: 0 - 79 */
00087                 int ypos  /* vertical position: 0 - 23 */
00088         );
00089
00090 /* END OF FILE */
00091
```

Index