F.R.I.D.A.Y.

Generated by Doxygen 1.9.6

Chapter 1

F.R.I.D.A.Y.

NOTE: Run `` doxygen Doxyfile `` to generate full documentation.

1.1 F.R.I.D.A.Y. Programmer Manual R1

1.1.0.1 Contents

1.1.1 1. Overview

F.R.I.D.A.Y. is a light-weight OS built to run on QEMU. You can use this documentation to extend the existing systems and add more functionality.

1.1.2 2. OS Lifecycle

When the OS kernel is booted, the first function `kmain()` is called. This function bootstraps most of the core functionality of the OS. Once bootstrapping is done, control is passed to `comhand()` for the command handler.

Once the command handler has finished, `comhand()` will return, thus giving control back to `kmain()`. `kmain()` then begins the shutdown process and exits.

1.1.3 3. Extending Systems

1.1.3.1 3.i. kmain() and Startup

`kmain()` is the first function called after the bootloader for the OS. This function is located in `kmain.c` and is responsible for bootstrapping most of the OS' core functionality. After all core systems have been initialized, full control is passed to the `comhand()` function in `comhand.c`. If something needs to be initialized, put the method call for it before the call to `comhand()`.

1.1.3.2 3.ii. The Command Handler

`comhand()` is what defines the OS' command handling system. When `kmain()` calls this function, the command handler welcomes the user and begins listening for user input. The command handler requests user input via a `sys_req()` call. The input gathered from this method is then used to run the command that matches the input, if any.

1.1.3.3 3.iii. Registering a Command

All commands are 'registered' via the `comm_funcs` array inside `comhand.c`. This array contains pointers to functions that follow the format:

```
bool cmd_((COMMAND_NAME))(const char *command);
```

Note that the name of the method is **not** required to be followed, but should to maintain convention. Any new command **should** be placed in user space, preferably in the `commands.c` file. The return value of the function should signify if the command matched the **label** of the command. i.e. the command `help junk-option1 junk-option2` should still return true for the help command, even though the options are not valid.

Command Function The start of a command function should resemble:

```
bool cmd_name(const char *comm)
{
    const char *label = "name";

    if (!matches_cmd(comm, label))
        return false;

    //cmd logic
    return true;
}
```

Use the `matches_cmd(const char *cmd, const char *label)` function to check if the command's label matches.

Adding Command to Help Once you've added a command, you should add a help message for it. Use the `help_messages` array to add an instance of the `help_info` struct. Doing so should resemble:

```
{.str_label = "name", .help_message = "The %s command does X and then does Y.\nYou should include Z arguments
```

After adding this, running `help name` command will then recognize the added struct and return the `help_↩ message` formatted with the command's name.

1.1.3.4 4. Conclusion

The information above covers most important information on how to extend F.R.I.D.A.Y. Please use the included Doxygen documentation for more information on how the internal systems work. If you'd like to learn how to use the system from a user's perspective, please refer to our `User Manual`

Chapter 2

Preparing Windows for MPX Development

Windows is not suited for native MPX development. Instead, you will need to set up a Linux distribution in a either virtual machine or using the Windows Subsystem for Linux (WSL).

2.1 Virtual Machine

The recommended virtual machine is LOUD, the LCSEE Optimized Ubuntu Distribution. Follow the directions at https://lcseesystems.wvu.edu/services/loud. If you encounter issues where the virtual machine hangs at a black screen, or is unusably slow (indicated by a turtle icon in the bottom-right hand corner of the VirtualBox window), consider WSL instead.

2.2 WSL

WSL is an optional component of Windows 10 and later. First, you will need to ensure that WSL itself is enabled, and that a distribution is installed. Open an elevated Command Prompt or PowerShell window by pressing the Windows Key + X, and choosing "Command Prompt (Admin)" or "PowerShell (Admin)". In this window, run:

```
wsl --install -d ubuntu
```

This will enable WSL if it isn't already, and install Ubuntu along with it. If WSL wasn't already installed, you may need to reboot before you can launch an Ubuntu window. The first time you open an Ubuntu window, you'll be prompted to create a username and password. This will become the local account within the Ubuntu environment, and the password will become the one you need later to run commands with `sudo`. Once the account is set up, follow the steps for Ubuntu below.

2.3 Preparing Ubuntu and Other Debian Derivatives for MPX Development

Ubuntu is the primary development environment for MPX and the basis for LOUD, so no extensive preparation is needed. Simply open a terminal window and run the following commands:

```
sudo apt update
sudo apt install -y clang make nasm git binutils-i686-linux-gnu qemu-system-x86 gdb
```

2.4 Preparing macOS for MPX Development

All commands need to be run from a Terminal. You should be able to find the Terminal application in the `/Applications` folder of your system's internal disk. Alternatively, pressing Command+Space and typing `Terminal` should bring it up.

2.4.1 Install XCode Tools

First, you need to install the XCode development tools. This includes the compiler, clang, and GNU make. This command will open a pop-up window for confirmation. Once confirmed, this may take some time to complete.

```
xcode-select --install
```

2.4.2 Install Homebrew

Next, install the Homebrew package manager from `https://brew.sh`. There should be a command under the label "Install Homebrew" that you can copy and paste into your Terminal window. Note that this makes use of the XCode tools installed in the first step, so that **must** be complete prior to this step.

It is likely that installing Homebrew will prompt you for your password so that it can elevate privileges using `sudo`. This is the same password you use to unlock your account when you turn on your system.

Note also that once the command you paste from the web site completes, there are a few additional steps you need to take to finalize the installation of Homebrew. In your terminal window, there will be some output beginning with the bold words `**==> Next steps:**`. You must follow the instructions in your Terminal window to complete the Homebrew installation.

2.4.3 Install Remaining Tools

Once Homebrew is installed, you can easily install NASM, QEMU, the cross-linker, and cross-debugger.

```
brew install nasm qemu i686-elf-binutils i386-elf-gdb
```

If you get an error here, make sure that you followed the `**==> Next steps:**` portion of the Homebrew installation process. You may need to open a new Terminal window for the changes to take effect.

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 alarm_params Struct Reference

The parameters used to pass into the alarm function.

Public Attributes

- int * **time_ptr**

  A pointer to where the time is stored.
- char * **str_ptr**

  A pointer to where the message is stored.
- time_zone_t * **time_zone**

  The timezone used to create the alarm.
- unsigned char **buffer** [100]

  The data to store.

5.1.1 Detailed Description

The parameters used to pass into the alarm function.

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

- kernel/alarm.c

5.2 context Struct Reference

The context to save onto a PCB.

```
#include <pcb.h>
```

Public Attributes

- int **gs**

  The segment registers.
- int **fs**
- int **es**
- int **ds**
- int **ss**
- int **edi**

  The general purpose registers.
- int **esi**
- int **ebp**
- int **esp**
- int **ebx**
- int **edx**
- int **ecx**
- int **eax**
- int **eip**

  The status control registers, ordered as they are added for interrupts.
- int **cs**
- int **eflags**

5.2.1 Detailed Description

The context to save onto a PCB.

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

- include/mpx/pcb.h

5.3 gdt_descriptor Struct Reference

Public Attributes

- uint16_t **size**
- struct gdt_entry * **base**

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

- kernel/core.c

5.4 gdt_entry Struct Reference

Public Attributes

- uint16_t **limit_low**
- uint16_t **base_low**

  first 16 bits of limit
- uint8_t **base_mid**

  first 16 bits of base
- uint8_t **access**

  bits 16-23 of base
- uint8_t **flags**

  next 8 bits; access flags
- uint8_t **base_high**

  page granularity, size

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

- kernel/core.c

5.5 help_info Struct Reference

Used to store information on a specific label of the 'help' command.

Public Attributes

- char * **str_label** [15]

  The label of the command for the help message.
- char * **help_message**

  The help message to send for this struct.

5.5.1 Detailed Description

Used to store information on a specific label of the 'help' command.

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

- user/commands.c

5.6 idt_descriptor Struct Reference

The metadata for the IDT.

Public Attributes

- uint16_t **size**
- struct idt_entry * **base**

5.6.1 Detailed Description

The metadata for the IDT.

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

- kernel/core.c

5.7 idt_entry Struct Reference

A single entry in the IDT.

Public Attributes

- uint16_t **base_low**
- uint16_t **sselect**

  offset bits 0..15
- uint8_t **zero**

  stack selector in gdt or ldt
- uint8_t **flags**

  this stays zero; unused
- uint16_t **base_high**

  attributes

5.7.1 Detailed Description

A single entry in the IDT.

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

- kernel/core.c

5.8 line_entry Struct Reference

Used to store a specific line previously entered.

Public Attributes

- void * **_dont_use_1**

  These are a hacky way to use linked lists without excessive allocation (temp until R5)
- void * **_dont_use_2**

  These are a hacky way to use linked lists without excessive allocation (temp until R5)
- char * line

  The line that was entered.
- size_t **line_length**

  The line's length, not including the null terminator.

5.8.1 Detailed Description

Used to store a specific line previously entered.

5.8.2 Member Data Documentation

5.8.2.1 line

```
char* line_entry::line
```

The line that was entered.

Does not include null terminator.

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

- kernel/serial.c

5.9 linked_list_ Struct Reference

The main linked list structure.

```
#include <linked_list.h>
```

Public Attributes

- int **_size**

  The size of the linked list.
- int **_max_size**

  The maximum size of the linked list, set to -1 for infinite.
- int(* **sort_func**)(void *, void *)

  A pointer to the sorting function.
- ll_node * **_first**

  The first node in the linked list.
- ll_node * **_last**

  The second node in the linked list.

5.9.1 Detailed Description

The main linked list structure.

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

- include/linked_list.h

5.10 linked_list_node_ Struct Reference

The node used for all linked lists.

`#include <linked_list.h>`

Public Attributes

- void * **_item**

  The pointer to the item we're storing.
- struct linked_list_node_ * **_next**

  The next node in the list.

5.10.1 Detailed Description

The node used for all linked lists.

Note that

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

- include/linked_list.h

5.11 page_dir Struct Reference

Public Attributes

- page_table * **tables** [1024]
- uint32_t **tables_phys** [1024]

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

- kernel/core.c

5.12 page_entry Struct Reference

Public Attributes

- uint32_t **present**:1
- uint32_t **writeable**:1
- uint32_t **usermode**:1
- uint32_t **accessed**:1
- uint32_t **dirty**:1
- uint32_t **reserved**:7
- uint32_t **frameaddr**:20

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

- kernel/core.c

5.13 page_table Struct Reference

Public Attributes

- page_entry **pages** [1024]

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

- kernel/core.c

5.14 pcb Struct Reference

The definition of a process control block.

```
#include <pcb.h>
```

Public Attributes

- void * **_next**

  This exists as an extremely hacky way to use them in the linked list without allocating memory.
- void * **_item**

  This exists as an extremely hacky way to use them in the linked list without allocating memory.
- const char * **name**

  The name of the PCB, max length of 8.
- enum pcb_class **process_class**

  The process class type.
- int **priority**

  Integer priority of PCB, 0-9, lower = higher priority;.
- enum pcb_exec_state **exec_state**

  The execution state of this PCB.
- enum pcb_dispatch_state **dispatch_state**

  The dispatch state of this PCB.
- void * **stack_ptr**

  A pointer to the next available byte in the stack.
- unsigned char **stack** [PCB_STACK_SIZE]

  The stack itself.

5.14.1 Detailed Description

The definition of a process control block.

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

- include/mpx/pcb.h

Chapter 6

File Documentation

6.1 bomb_catcher.h

```
00001 //
00002 // Created by Andrew Bowie on 1/27/23.
00003 //
00004
00005 #ifndef F_R_I_D_A_Y_BOMB_CATCHER_H
00006 #define F_R_I_D_A_Y_BOMB_CATCHER_H
00007
00011 void start_bombcatcher(void);
00012
00013 #endif //F_R_I_D_A_Y_BOMB_CATCHER_H
```

6.2 include/cli.h File Reference

Contains useful commands for interfacing with the CLI.

Functions

- void set_cli_prompt (const char *prompt)

  Sets the CLI prompt to be used when prompting input.
- void set_cli_history (bool enabled)

  Sets if the CLI is enabled.
- void set_command_formatting (bool enabled)

  If command color formatting should be enabled.
- void set_invisible (bool enabled)

  Sets if the input for the line should be invisible.
- void set_tab_completions (bool enabled)

  Sets if the input should use tab completions.

6.2.1 Detailed Description

Contains useful commands for interfacing with the CLI.

6.2.2 Function Documentation

6.2.2.1 set_cli_history()

```
void set_cli_history (
            bool enabled )
```

Sets if the CLI is enabled.

Parameters

enabled	if the CLI should be enabled.

6.2.2.2 set_cli_prompt()

```
void set_cli_prompt (
            const char * prompt )
```

Sets the CLI prompt to be used when prompting input.

Can be set to NULL if no prompt should be printed.

Parameters

prompt	the prompt to use.

6.2.2.3 set_command_formatting()

```
void set_command_formatting (
            bool enabled )
```

If command color formatting should be enabled.

Parameters

enabled	if it should be enabled.

6.2.2.4 set_invisible()

```
void set_invisible (
```

```
            bool enabled )
```

Sets if the input for the line should be invisible.

Parameters

enabled	if it's enabled or not.

6.2.2.5 set_tab_completions()

```
void set_tab_completions (
            bool enabled )
```

Sets if the input should use tab completions.

Parameters

enabled	if it's enabled or not.

6.3 cli.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Andrew Bowie on 1/27/23.
00003 //
00004
00005 #ifndef F_R_I_D_A_Y_CLI_H
00006 #define F_R_I_D_A_Y_CLI_H
00007
00018 void set_cli_prompt(const char *prompt);
00019
00024 void set_cli_history(bool enabled);
00025
00030 void set_command_formatting(bool enabled);
00031
00036 void set_invisible(bool enabled);
00037
00042 void set_tab_completions(bool enabled);
00043
00044 #endif //F_R_I_D_A_Y_CLI_H
```

6.4 color.h

```
00001 //
00002 // Created by Andrew Bowie on 1/27/23.
00003 //
00004
00005 #ifndef F_R_I_D_A_Y_COLOR_H
00006 #define F_R_I_D_A_Y_COLOR_H
00007
00014 typedef struct {
00016     const char *color_label;
00018     const int color_num;
00019 } color_t;
00020
00025 void set_output_color(const color_t *color);
00026
00031 const color_t *get_output_color(void);
```

```
00032
00038 const color_t *get_color(const char *label);
00039
00044 const color_t **get_colors(void);
00045
00046 #endif //F_R_I_D_A_Y_COLOR_H
```

6.5 include/commands.h File Reference

This file contains headers for commands run by the command handler.

```
#include "stdbool.h"
```

Functions

- const char * find_best_match (const char *cmd)

  Finds the best match for the given command, or NULL if it doesn't match OR matches multiple OR is equal to the command.
- bool command_exists (const char *cmd)

  Checks if the given command exists.
- bool cmd_version (const char *comm)

  The version command, used to handle when the user asks for a version number.
- bool cmd_shutdown (const char *comm)

  The shutdown command.
- bool cmd_get_time_menu (const char *comm)

  The get time command, used to get the time on the system.
- bool cmd_help (const char *comm)

  The help command, used to help the user when they are struggling.
- bool cmd_set_time (const char *comm)

  The set time command, used to set time user wants.
- bool cmd_set_date (const char *comm)

  The set date command, used to set time user wants.
- bool cmd_set_tz (const char *comm)

  The set timezone command, used to set the system timezone.
- bool cmd_clear (const char *comm)

  The clear command, used to clear the console.
- bool cmd_color (const char *comm)

  The color command, used to change text color for the terminal.
- bool cmd_yield (const char *comm)

  the yield command, causes the command handler to yield immediately.
- bool cmd_pcb (const char *comm)

  The pcb command, used to interact with the pcb system.
- bool cmd_alarm (const char *comm)

6.5.1 Detailed Description

This file contains headers for commands run by the command handler.

6.5.2 Function Documentation

6.5.2.1 cmd_alarm()

```
bool cmd_alarm (
            const char * comm )
```

Parameters

comm	
message	

Returns

Authors

Jared Crowley

6.5.2.2 cmd_clear()

```
bool cmd_clear (
            const char * comm )
```

The clear command, used to clear the console.

Parameters

comm	the command string.

Returns

true if it was handled, false if not.

6.5.2.3 cmd_color()

```
bool cmd_color (
            const char * comm )
```

The color command, used to change text color for the terminal.

Parameters

comm	the command string.

Returns

true if it was handled, false if not.

6.5.2.4 cmd_get_time_menu()

```
bool cmd_get_time_menu (
            const char * comm )
```

The get time command, used to get the time on the system.

Parameters

comm	the command string.

Returns

true if the command was handled, false if not.

6.5.2.5 cmd_help()

```
bool cmd_help (
            const char * comm )
```

The help command, used to help the user when they are struggling.

Parameters

comm	the command string.

Returns

true if it was handled, false if not.

6.5.2.6 cmd_pcb()

```
bool cmd_pcb (
            const char * comm )
```

The pcb command, used to interact with the pcb system.

Parameters

comm	the command string.

Returns

true if it was handled, false if not.

6.5.2.7 cmd_set_date()

```
bool cmd_set_date (
            const char * comm )
```

The set date command, used to set time user wants.

Parameters

comm	the command string.

Returns

true if it was handled, false if not.

6.5.2.8 cmd_set_time()

```
bool cmd_set_time (
            const char * comm )
```

The set time command, used to set time user wants.

Parameters

comm	the command string.

Returns

true if it was handled, false if not.

6.5.2.9 cmd_set_tz()

```
bool cmd_set_tz (
            const char * comm )
```

The set timezone command, used to set the system timezone.

Parameters

comm	the command string.

Returns

true if it was handled, false if not.

6.5.2.10 cmd_shutdown()

```
bool cmd_shutdown (
            const char * comm )
```

The shutdown command.

If ran, will re-prompt the user for confirmation.

Parameters

comm	the command string.

Returns

true if the command was handled, false if not.

6.5.2.11 cmd_version()

```
bool cmd_version (
            const char * comm )
```

The version command, used to handle when the user asks for a version number.

Must Include Compilation date

Parameters

comm	the command string.

Returns

true if the command was handled, false if not.

6.5.2.12 cmd_yield()

```
bool cmd_yield (
            const char * comm )
```

the yield command, causes the command handler to yield immediately.

Parameters

comm	the command string.

Returns

true if it was handled, false if not.

6.5.2.13 command_exists()

```
bool command_exists (
            const char * cmd )
```

Checks if the given command exists.

Parameters

cmd	the command to check for.

Returns

true if it does, false if not.

Authors

Andrew Bowie

6.5.2.14 find_best_match()

```
const char * find_best_match (
            const char * cmd )
```

Finds the best match for the given command, or NULL if it doesn't match OR matches multiple OR is equal to the command.

Parameters

cmd	the command.

Returns

the best match for it.

Authors

Andrew Bowie

6.6 commands.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Andrew Bowie on 1/18/23.
00003 //
00004
00005 #ifndef F_R_I_D_A_Y_COMMANDS_H
00006 #define F_R_I_D_A_Y_COMMANDS_H
00007
00008 #include "stdbool.h"
00009
00023 const char *find_best_match(const char *cmd);
00024
00031 bool command_exists(const char *cmd);
00032
00038 bool cmd_version(const char *comm);
00039
00045 bool cmd_shutdown(const char *comm);
00046
00052 bool cmd_get_time_menu(const char *comm);
00053
00059 bool cmd_help(const char *comm);
00060
00066 bool cmd_set_time(const char* comm);
00067
00073 bool cmd_set_date(const char* comm);
00074
00080 bool cmd_set_tz(const char *comm);
00081
00087 bool cmd_clear(const char *comm);
00088
00094 bool cmd_color(const char *comm);
00095
00101 bool cmd_yield(const char *comm);
00102
00103
00109 bool cmd_pcb(const char *comm);
00110
00111 bool cmd_alarm(const char *comm);
00112 #endif //F_R_I_D_A_Y_COMMANDS_H
```

6.7 include/ctype.h File Reference

A subset of standard C library functions.

Functions

- int isspace (int c)

  Determine if a character is whitespace.
- int isdigit (int c)

  Determine if a character is a digit.
- int todigit (int c)

  Return int value of character if is digit.
- int isupper (int c)

Determine if a character is uppercase.

- int islower (int c)

  Determine if a character is lowercase.

- int tolower (int c)

  Converts the given character to lowercase.

- int toupper (int c)

  Converts the given character to uppercase.

6.7.1 Detailed Description

A subset of standard C library functions.

6.7.2 Function Documentation

6.7.2.1 isdigit()

```
int isdigit (
            int c )
```

Determine if a character is a digit.

Parameters

c	Character to check

Returns

Non-zero if digit, 0 if not digit

6.7.2.2 islower()

```
int islower (
            int c )
```

Determine if a character is lowercase.

If the character is not alphabetical, 0 is returned.

Parameters

c	Character to check.

Returns

Non-zero if lower, 0 if not lower.

6.7.2.3 isspace()

```
int isspace (
            int c )
```

Determine if a character is whitespace.

Parameters

c	Character to check

Returns

Non-zero if space, 0 if not space

6.7.2.4 isupper()

```
int isupper (
            int c )
```

Determine if a character is uppercase.

If the character is not alphabetical, 0 is returned.

Parameters

c	Character to check.

Returns

Non-zero if upper, 0 if not upper.

6.7.2.5 todigit()

```
int todigit (
            int c )
```

Return int value of character if is digit.

Parameters

c	Character to check

Returns

Negative not digit, value of digit otherwise

6.7.2.6 tolower()

```
int tolower (
             int c )
```

Converts the given character to lowercase.

Parameters

c	the character to convert.

Returns

the lowercase character.

6.7.2.7 toupper()

```
int toupper (
             int c )
```

Converts the given character to uppercase.

Parameters

c	the character to convert.

Returns

the uppercase character.

6.8 ctype.h

Go to the documentation of this file.

```
00001 #ifndef MPX_CTYPE_H
00002 #define MPX_CTYPE_H
```

```
00003
00014 int isspace(int c);
00015
00021 int isdigit(int c);
00027 int todigit(int c);
00028
00035 int isupper(int c);
00036
00043 int islower(int c);
00044
00050 int tolower(int c);
00051
00057 int toupper(int c);
00058
00059 #endif
```

6.9 include/linked_list.h File Reference

This file represents the functionality and structure of a linked list.

Classes

- struct linked_list_node_

  The node used for all linked lists.
- struct linked_list_

  The main linked list structure.

Typedefs

- typedef struct linked_list_node_ ll_node

  The node used for all linked lists.
- typedef struct linked_list_ **linked_list**

  The main linked list structure.

6.9.1 Detailed Description

This file represents the functionality and structure of a linked list.

Any item added to this list, MUST contain the necessary data as defined by the ll_node type.

6.9.2 Typedef Documentation

6.9.2.1 ll_node

```
typedef struct linked_list_node_ ll_node
```

The node used for all linked lists.

Note that

6.10 linked_list.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Andrew Bowie on 9/18/22.
00003 //
00004
00005 #ifndef LINKEDLIST_H
00006 #define LINKEDLIST_H
00007
00017 typedef struct linked_list_node_
00018 {
00020     void *_item; //8 bytes
00022     struct linked_list_node_ *_next; //8 bytes
00023 } ll_node;
00024
00028 typedef struct linked_list_ {
00030     int _size;
00032     int _max_size;
00034     int (*sort_func)(void*, void*);
00036     ll_node *_first;
00038     ll_node *_last;
00039 } linked_list;
00040
00045 linked_list
00046 *nl_unbounded(void);
00047
00052 linked_list
00053 *nl_maxsize(int max_size);
00054
00061 ll_node
00062 *get_first_node(linked_list *list);
00063
00070 ll_node
00071 *next_node(ll_node *node);
00072
00079 void
00080 *get_item_node(ll_node *node);
00081
00086 int
00087 list_size(linked_list *list);
00088
00096 void *
00097 get_item(linked_list *list, int index);
00098
00104 void
00105 destroy_list(linked_list *list, int destroy_values);
00106
00113 int
00114 add_item(linked_list *list, void *item);
00115
00123 int
00124 add_item_index(linked_list *list, int index, void *item);
00125
00132 void
00133 remove_item(linked_list *list, int index);
00134
00141 int
00142 remove_item_ptr(linked_list *list, void *item_ptr);
00143
00151 void
00152 *remove_item_unsafe(linked_list *list, int index);
00153
00159 void
00160 set_sort_func(linked_list *list, int sort_func(void *, void *));
00161
00167 void
00168 for_each_il(linked_list *list, void call(void *node));
00169
00170 #endif //LINKEDLIST_H
```

6.11 include/math.h File Reference

A header full of useful math type functions.

Functions

- unsigned int ui_realmod (int x, int mod)

  Calculates the real modulo value of X modulo 'mod'.
- double pow (double a, double b)

  Calculates the Answer from a variable and a exponent.
- void s_rand (unsigned long long seed)

  Seeds the random number generator.
- unsigned int next_random (void)

  Returns the next random 30 bits from the LCRNG.

6.11.1 Detailed Description

A header full of useful math type functions.

6.11.2 Function Documentation

6.11.2.1 next_random()

```
unsigned int next_random (
            void  )
```

Returns the next random 30 bits from the LCRNG.

Returns

the next random number.

6.11.2.2 pow()

```
double pow (
            double a,
            double b )
```

Calculates the Answer from a variable and a exponent.

Parameters

a	is the variable
b	is the exponent

Returns

The new value from the a^b

6.11.2.3 s_rand()

```
void s_rand (
            unsigned long long seed )
```

Seeds the random number generator.

Parameters

seed	the seed.

6.11.2.4 ui_realmod()

```
unsigned int ui_realmod (
            int x,
            int mod )
```

Calculates the real modulo value of X modulo 'mod'.

Parameters

x	the value.
mod	the modulo.

Returns

the modulo value of x modulo 'mod'

6.12 math.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Andrew Bowie on 1/19/23.
00003 //
00004
00005 #ifndef F_R_I_D_A_Y_MATH_H
00006 #define F_R_I_D_A_Y_MATH_H
00007
00019 unsigned int ui_realmod(int x, int mod);
00020
00027 double pow(double a, double b);
00028
00033 void s_rand(unsigned long long seed);
00034
00039 unsigned int next_random(void);
00040 #endif //F_R_I_D_A_Y_MATH_H
```

6.13 include/memory.h File Reference

MPX-specific dynamic memory functions.

```
#include <stddef.h>
```

Functions

- void * sys_alloc_mem (size_t size)
  Allocate dynamic memory.
- int sys_free_mem (void *ptr)
  Free dynamic memory.
- void sys_set_heap_functions (void *(*alloc_fn)(size_t), int(*free_fn)(void *))
  Installs user-supplied heap management functions.

6.13.1 Detailed Description

MPX-specific dynamic memory functions.

6.13.2 Function Documentation

6.13.2.1 sys_alloc_mem()

```
void * sys_alloc_mem (
            size_t size )
```

Allocate dynamic memory.

Parameters

size	The amount of memory, in bytes, to allocate

Returns

NULL on error, otherwise the address of the newly allocated memory

6.13.2.2 sys_free_mem()

```
int sys_free_mem (
            void * ptr )
```

Free dynamic memory.

Parameters

ptr	The address of dynamically allocated memory to free

Returns

0 on success, non-zero on error

6.13.2.3 sys_set_heap_functions()

```
void sys_set_heap_functions (
             void *(*)(size_t) alloc_fn,
             int(*)(void *) free_fn )
```

Installs user-supplied heap management functions.

Parameters

alloc_fn	A function that dynamically allocates memory
free_fn	A function that frees dynamically allocated memory

6.14 memory.h

Go to the documentation of this file.

```
00001 #ifndef MPX_MEMORY_H
00002 #define MPX_MEMORY_H
00003
00004 #include <stddef.h>
00005
00016 void *sys_alloc_mem(size_t size);
00017
00023 int sys_free_mem(void *ptr);
00024
00030 void sys_set_heap_functions(void * (*alloc_fn)(size_t), int (*free_fn)(void *));
00031
00032 #endif
```

6.15 include/mpx/alarm.h File Reference

A header file for alarm functions.

Functions

- bool create_new_alarm (int *time_array, const char *message)

  Creates a new pcb that will display message at or after given time.
- bool shouldAlarm (const int *time_array, time_zone_t *time_zone)

  Checks current time to see if alarm should alert.

6.15.1 Detailed Description

A header file for alarm functions.

6.15.2 Function Documentation

6.15.2.1 create_new_alarm()

```
bool create_new_alarm (
            int * time_array,
            const char * message )
```

Creates a new pcb that will display message at or after given time.

Parameters

time_array	the time to display message
message	message to display

Returns

true if the alarm was created, false if it failed.

Author

Kolby Eisenhauer, Andrew Bowie

6.15.2.2 shouldAlarm()

```
bool shouldAlarm (
            const int * time_array,
            time_zone_t * time_zone )
```

Checks current time to see if alarm should alert.

Parameters

time_array	the time to display message
time_zone	the timezone the alarm was created in.

Returns

boolean if should alarm

Author

Kolby Eisenhauer

6.16 alarm.h

Go to the documentation of this file.

```
00001 #ifndef F_R_I_D_A_Y_ALARM_H
00002 #define F_R_I_D_A_Y_ALARM_H
00003
00016 bool create_new_alarm(int *time_array, const char* message);
00017
00025 bool shouldAlarm(const int *time_array, time_zone_t *time_zone);
00026 #endif
```

6.17 include/mpx/clock.h File Reference

Contains functions for interacting with the system clock.

```
#include "time_zone.h"
```

Functions

- const time_zone_t * get_clock_timezone (void)

  Gets the current timezone for the clock.
- void set_timezone (const time_zone_t *offset)

  Sets the timezone hour offset.
- int print_time (void)

  Prints the time and date of the system.
- int * adj_timezone (int time[6], int tz_offset_hr, int tz_offset_min)

  Adjusts the given time array to the specified timezone.
- int * get_time (int t_buf[7])

  Gets the time and stores it in the given array in the form: {year, month, date, week_day, hours, mins, seconds}.
- bool set_time_clock (unsigned int hr, unsigned int min, unsigned int sec)

  Sets the time of the system clock to the provided values.
- bool set_date_clock (unsigned int month, unsigned int day, unsigned int year)

  Sets the date of the system clock to the provided values.
- unsigned char decimal_to_bcd (unsigned int decimal)

  Converts the given decimal number to BCD.
- int bcd_to_decimal (unsigned char bcd)

  Converts the given BCD number to decimal.
- bool is_valid_date_or_time (int word_len, char buf[ ][word_len], int buff_len)

  Checks if the given array of time values is validly defined.
- unsigned int get_days_in_month (int month, int year)

  Gets the amount of days in the provided month and returns it in BCD.

6.17.1 Detailed Description

Contains functions for interacting with the system clock.

6.17.2 Function Documentation

6.17.2.1 adj_timezone()

```
int * adj_timezone (
            int time[6],
            int tz_offset_hr,
            int tz_offset_min )
```

Adjusts the given time array to the specified timezone.

Parameters

time	the time array, should be passed in with the format {year, month, date, week_day, hours, mins}.
tz_offset_hr	the hour offset.
tz_offset_min	the minute offset.

Returns

a pointer to the adjusted array.

6.17.2.2 bcd_to_decimal()

```
int bcd_to_decimal (
            unsigned char bcd )
```

Converts the given BCD number to decimal.

Parameters

bcd	the number to convert.

Returns

the converted number.

6.17.2.3 decimal_to_bcd()

```
unsigned char decimal_to_bcd (
            unsigned int decimal )
```

Converts the given decimal number to BCD.

Parameters

decimal	the number to convert.

Returns

the converted number.

6.17.2.4 get_clock_timezone()

```
const time_zone_t * get_clock_timezone (
            void  )
```

Gets the current timezone for the clock.

Returns

the timezone.

6.17.2.5 get_days_in_month()

```
unsigned int get_days_in_month (
            int month,
            int year )
```

Gets the amount of days in the provided month and returns it in BCD.

Parameters

month	the month of the year, in BCD.
year	the year, in BCD. (Used for leap years)

Returns

the amount of days in the month, in BCD.

6.17.2.6 get_time()

```
int * get_time (
            int t_buf[7] )
```

Gets the time and stores it in the given array in the form: {year, month, date, week_day, hours, mins, seconds}.

Parameters

t_buf	the buffer to store the time in. Can be NULL.

Returns

the time array.

6.17.2.7 is_valid_date_or_time()

```
bool is_valid_date_or_time (
            int word_len,
            char buf[][word_len],
            int buff_len )
```

Checks if the given array of time values is validly defined.

All strings in the array must be valid, positive, 2 digit numbers.

Parameters

word_len	the length of 2nd dimension of the array.
buf	the array.
buff_len	the length of the 1st dimension of the array.

Returns

if the provided array is valid.

6.17.2.8 print_time()

```
int print_time (
            void  )
```

Prints the time and date of the system.

Returns

0 if successful, negative if not.

6.17.2.9 set_date_clock()

```
bool set_date_clock (
            unsigned int month,
            unsigned int day,
            unsigned int year )
```

Sets the date of the system clock to the provided values.

Parameters

month	the month, in BCD.
day	the day, in BCD.
year	the year, in BCD.

Returns

true if the time was changed, false if the values were invalid.

6.17.2.10 set_time_clock()

```
bool set_time_clock (
            unsigned int hr,
            unsigned int min,
            unsigned int sec )
```

Sets the time of the system clock to the provided values.

Parameters

hr	the hours, in BCD.
min	the minutes, in BCD.
sec	the seconds, in BCD.

Returns

true if the time was changed, false if the values were invalid.

6.17.2.11 set_timezone()

```
void set_timezone (
            const time_zone_t * offset )
```

Sets the timezone hour offset.

Parameters

offset	the hour offset.

6.18 clock.h

Go to the documentation of this file.

```
00001 #ifndef F_R_I_D_A_Y_SET_TIME_H
00002 #define F_R_I_D_A_Y_SET_TIME_H
00003
00004 #include "time_zone.h"
00005
00015 const time_zone_t *get_clock_timezone(void);
00016
00021 void set_timezone(const time_zone_t *offset);
00022
00027 int print_time(void);
00028
00037 int *adj_timezone(int time[6], int tz_offset_hr, int tz_offset_min);
00038
00045 int *get_time(int t_buf[7]);
00046
00054 bool set_time_clock(unsigned int hr, unsigned int min, unsigned int sec);
00055
00063 bool set_date_clock(unsigned int month, unsigned int day, unsigned int year);
00064
00070 unsigned char decimal_to_bcd(unsigned int decimal);
00071
00077 int  bcd_to_decimal(unsigned char bcd);
00078
00087 bool is_valid_date_or_time(int word_len, char buf[][word_len], int buff_len);
00088
00095 unsigned int get_days_in_month(int month, int year);
00096 #endif
```

6.19 comhand.h

```
00001
00002 #ifndef F_R_I_D_A_Y_COMHAND_H
00003 #define F_R_I_D_A_Y_COMHAND_H
00004
00005 #define CMD_PROMPT "» "
00006
00017 void signal_shutdown(void);
00018
00022 void comhand(void);
00023
00024 #endif //F_R_I_D_A_Y_COMHAND_H
```

6.20 device.h

```
00001 #ifndef MPX_DEVICES_H
00002 #define MPX_DEVICES_H
00003
00004 typedef enum {
00005     COM1 = 0x3f8,
00006     COM2 = 0x2f8,
00007     COM3 = 0x3e8,
00008     COM4 = 0x2e8,
00009 } device;
00010
00011 #endif
```

6.21 include/mpx/gdt.h File Reference

Kernel functions to initialize the Global Descriptor Table.

Functions

- void **gdt_init** (void)

  Creates and installs the Global Descriptor Table.

6.21.1 Detailed Description

Kernel functions to initialize the Global Descriptor Table.

6.22 gdt.h

Go to the documentation of this file.

```
00001 #ifndef MPX_GDT_H
00002 #define MPX_GDT_H
00003
00010 void gdt_init(void);
00011
00012 #endif
```

6.23 include/mpx/interrupts.h File Reference

Kernel functions related to software and hardware interrupts.

Macros

- #define **sti**() __asm__ volatile ("sti")

  Disable interrupts.
- #define **cli**() __asm__ volatile ("cli")

  Enable interrupts.

Functions

- void irq_init (void)

  Installs the initial interrupt handlers for the first 32 IRQ lines.
- void pic_init (void)

  Initializes the programmable interrupt controllers and performs the necessary remapping of IRQs.
- void **idt_init** (void)

  Creates and installs the Interrupt Descriptor Table.
- void **idt_install** (int vector, void(∗handler)(void ∗))

  Installs an interrupt handler.

6.23.1 Detailed Description

Kernel functions related to software and hardware interrupts.

6.23.2 Function Documentation

6.23.2.1 irq_init()

```
void irq_init (
            void  )
```

Installs the initial interrupt handlers for the first 32 IRQ lines.

Most do a panic for now.

6.23.2.2 pic_init()

```
void pic_init (
            void  )
```

Initializes the programmable interrupt controllers and performs the necessary remapping of IRQs.

Leaves interrupts turned off.

6.24 interrupts.h

Go to the documentation of this file.

```
00001 #ifndef MPX_INTERRUPTS_H
00002 #define MPX_INTERRUPTS_H
00003
00010 #define sti() __asm__ volatile ("sti")
00011
00013 #define cli() __asm__ volatile ("cli")
00014
00019 void irq_init(void);
00020
00025 void pic_init(void);
00026
00028 void idt_init(void);
00029
00031 void idt_install(int vector, void (*handler)(void *));
00032
00033 #endif
```

6.25 include/mpx/io.h File Reference

Kernel macros to read and write I/O ports.

Macros

- #define outb(port, data) __asm__ volatile ("outb %%al, %%dx" :: "a" (data), "d" (port))

  Write one byte to an I/O port.
- #define inb(port)

  Read one byte from an I/O port.

6.25.1 Detailed Description

Kernel macros to read and write I/O ports.

6.25.2 Macro Definition Documentation

6.25.2.1 inb

```
#define inb(
              port )
```

Value:

```
({                                \
unsigned char r;                  \
__asm__ volatile ("inb %%dx, %%al" : "=a" (r) : "d" (port));  \
r;                                \
})
```

Read one byte from an I/O port.

Parameters

port	The port to read from

Returns

A byte of data read from the port

6.25.2.2 outb

```
#define outb(
              port,
              data )  __asm__ volatile ("outb %%al, %%dx" ::  "a" (data), "d" (port))
```

Write one byte to an I/O port.

Parameters

port	The port to write to
data	The byte to write to the port

6.26 io.h

Go to the documentation of this file.

```
00001 #ifndef MPX_IO_H
00002 #define MPX_IO_H
00003
00014 #define outb(port, data)                        \
00015     __asm__ volatile ("outb %%al, %%dx" :: "a" (data), "d" (port))
00016
00022 #define inb(port) ({                            \
```

```
00023       unsigned char r;                            \
00024       __asm__ volatile ("inb %%dx, %%al" : "=a" (r) : "d" (port));  \
00025       r;                                          \
00026     })
00027
00028 #endif
```

6.27 include/mpx/panic.h File Reference

Common system functions and definitions.

```
#include <stdnoreturn.h>
```

Functions

- noreturn __attribute__ ((no_caller_saved_registers)) void kpanic(const char *msg)

  Kernel panic.

6.27.1 Detailed Description

Common system functions and definitions.

6.27.2 Function Documentation

6.27.2.1 __attribute__()

```
noreturn __attribute__ (
             (no_caller_saved_registers)  ) const
```

Kernel panic.

Prints an error message and halts.

Parameters

msg	A message to display before halting

6.28 panic.h

Go to the documentation of this file.

```
00001 #ifndef MPX_PANIC_H
00002 #define MPX_PANIC_H
00003
```

```
00004 #include <stdnoreturn.h>
00005
00015 /*
00016  non-standard attribute is required for clang < 15
00017  */
00018 noreturn __attribute__((no_caller_saved_registers)) void kpanic(const char *msg);
00019
00020 #endif
```

6.29 include/mpx/pcb.h File Reference

This file contains all of the structure and functions for a PCB and its context.

```
#include "stdbool.h"
#include "stddef.h"
```

Classes

- struct pcb

  The definition of a process control block.
- struct context

  The context to save onto a PCB.

Macros

- #define **PCB_MAX_NAME_LEN** 8

  The maximum length of a PCB's name.
- #define **PCB_STACK_SIZE** 2048

  The initial size of a PCB's stack.

Enumerations

- enum pcb_class { **USER** = 0 , **SYSTEM** = 1 }

  The clas of a PCB.
- enum pcb_exec_state { **READY** = 0 , **RUNNING** = 1 , **BLOCKED** = 2 }

  The execution state of a PCB.
- enum pcb_dispatch_state { **NOT_SUSPENDED** = 0 , **SUSPENDED** = 1 }

  An enum of dispatch state for PCBs.

Functions

- void setup_queue (void)

  Sets up queue for PCBS.
- struct pcb * peek_next_pcb (void)

  Peeks the next available PCB, or returns NULL if it's empty.
- struct pcb * poll_next_pcb (void)

  Polls the next available PCB, or returns NULL if it's empty.
- struct pcb * pcb_alloc (void)

  Allocates memory for a PCB block.
- int pcb_free (struct pcb *pcb_ptr)

  Frees the memory associated with the given PCB block.

6.29.1 Detailed Description

This file contains all of the structure and functions for a PCB and its context.

6.29.2 Function Documentation

6.29.2.1 pcb_alloc()

```
struct pcb * pcb_alloc (
            void  )
```

Allocates memory for a PCB block.

Returns

A pointer to the allocated PCB.

Authors

Andrew Bowie, Kolby Eisenhauer

6.29.2.2 pcb_free()

```
int pcb_free (
            struct pcb * pcb_ptr )
```

Frees the memory associated with the given PCB block.

Parameters

pcb_ptr	the pointer to the pcb.

Returns

0 on success, non-zero on failure.

Authors

Andrew Bowie

6.29.2.3 peek_next_pcb()

```
struct pcb * peek_next_pcb (
            void  )
```

Peeks the next available PCB, or returns NULL if it's empty.

Returns

the next PCB or NULL.

6.29.2.4 poll_next_pcb()

```
struct pcb * poll_next_pcb (
            void  )
```

Polls the next available PCB, or returns NULL if it's empty.

Returns

the next PCB or NULL.

6.29.2.5 setup_queue()

```
void setup_queue (
            void  )
```

Sets up queue for PCBS.

Authors

Andrew Bowie

6.30 pcb.h

Go to the documentation of this file.

```
00001 #include "stdbool.h"
00002 #include "stddef.h"
00003 #ifndef MPX_PCB_H
00004 #define MPX_PCB_H
00005
00012 #define PCB_MAX_NAME_LEN 8
00014 #define PCB_STACK_SIZE 2048
00015
00017 enum pcb_class {
00018     USER = 0,
00019     SYSTEM = 1,
00020 };
00021
00023 enum pcb_exec_state {
00024     READY = 0,
00025     RUNNING = 1,
00026     BLOCKED = 2,
00027 };
00028
00030 enum pcb_dispatch_state {
00031     NOT_SUSPENDED = 0,
00032     SUSPENDED = 1,
00033 };
00034
00036 struct pcb {
00038     void *_next;
00040     void *_item;
00041
00043     const char *name;
00045     enum pcb_class process_class;
00047     int priority;
00049     enum pcb_exec_state exec_state;
00051     enum pcb_dispatch_state dispatch_state;
00053     void *stack_ptr;
00055     unsigned char stack[PCB_STACK_SIZE];
00056 };
00057
00059 struct context {
00061     int gs, fs, es, ds, ss;
00063     int edi, esi, ebp, esp, ebx, edx, ecx, eax;
00065     int eip, cs, eflags;
00066 };
00067
00072 void setup_queue(void);
00073
00078 struct pcb *peek_next_pcb(void);
00079
00084 struct pcb *poll_next_pcb(void);
00085
00092 struct pcb *pcb_alloc(void);
00093
00101 int pcb_free(struct pcb* pcb_ptr);
00102
00112 struct pcb *pcb_setup(const char *name, int class, int priority);
00113
00119 void pcb_insert(struct pcb* pcb_ptr);
00120
00127 struct pcb *pcb_find(const char *name);
00128
00136 bool pcb_remove(struct pcb *pcb_ptr);
00137
00147 bool generate_new_pcb(const char *name,
00148                       int priority,
00149                       enum pcb_class class,
00150                       void *begin_ptr,
00151                       const char *input,
00152                       size_t input_len,
00153                       size_t param_ptrs);
00154
00160 void exec_pcb_cmd(const char *comm);
00161
00162
00163 #endif
```

6.31 include/mpx/r3cmd.h File Reference

LoadR3 Loads the contents of R3 while cycling through each process.

Functions

- bool loadr3 (const char *comm)

6.31.1 Detailed Description

LoadR3 Loads the contents of R3 while cycling through each process.

6.31.2 Function Documentation

6.31.2.1 loadr3()

```
bool loadr3 (
            const char * comm )
```

Parameters

comm	the command.

Authors

Zachary Ebert

6.32 r3cmd.h

Go to the documentation of this file.

```
00001 #ifndef _r3cmd_H
00002 #define _r3cmd_H
00012 bool loadr3(const char *comm);
00013
00014 #endif
```

6.33 include/mpx/serial.h File Reference

Kernel functions and constants for handling serial I/O.

```
#include <stddef.h>
#include <mpx/device.h>
```

Functions

- int serial_init (device dev)

  Initializes devices for user input and output.
- int serial_out (device dev, const char *buffer, size_t len)

  Writes a buffer to a serial port.
- int serial_poll (device dev, char *buffer, size_t len)

  Reads a string from a serial port.

6.33.1 Detailed Description

Kernel functions and constants for handling serial I/O.

6.33.2 Function Documentation

6.33.2.1 serial_init()

```
int serial_init (
            device dev )
```

Initializes devices for user input and output.

Parameters

device	A serial port to initialize (COM1, COM2, COM3, or COM4)

Returns

0 on success, non-zero on failure

6.33.2.2 serial_out()

```
int serial_out (
            device dev,
            const char * buffer,
            size_t len )
```

Writes a buffer to a serial port.

Parameters

device	The serial port to output to
buffer	A pointer to an array of characters to output
len	The number of bytes to write

Returns

The number of bytes written

6.33.2.3 serial_poll()

```
int serial_poll (
              device dev,
              char * buffer,
              size_t len )
```

Reads a string from a serial port.

Parameters

device	The serial port to read data from
buffer	A buffer to write data into as it is read from the serial port
count	The maximum number of bytes to read

Returns

The number of bytes read on success, a negative number on failure

6.34 serial.h

Go to the documentation of this file.

```
00001 #ifndef MPX_SERIAL_H
00002 #define MPX_SERIAL_H
00003
00004 #include <stddef.h>
00005 #include <mpx/device.h>
00006
00017 int serial_init(device dev);
00018
00026 int serial_out(device dev, const char *buffer, size_t len);
00027
00036 int serial_poll(device dev, char *buffer, size_t len);
00037
00038 #endif
```

6.35 include/mpx/vm.h File Reference

Kernel functions for virtual memory and primitive allocation.

```
#include <stddef.h>
```

Functions

- void * kmalloc (size_t size, int align, void **phys_addr)

  Allocates memory from a primitive heap.
- void vm_init (void)

  Initializes the kernel page directory and initial kernel heap area.

6.35.1 Detailed Description

Kernel functions for virtual memory and primitive allocation.

6.35.2 Function Documentation

6.35.2.1 kmalloc()

```
void * kmalloc (
            size_t size,
            int align,
            void ** phys_addr )
```

Allocates memory from a primitive heap.

Parameters

size	The size of memory to allocate
align	If non-zero, align the allocation to a page boundary
phys_addr	If non-NULL, a pointer to a pointer that will hold the physical address of the new memory

Returns

The newly allocated memory

6.35.2.2 vm_init()

```
void vm_init (
            void  )
```

Initializes the kernel page directory and initial kernel heap area.

Performs identity mapping of the kernel frames such that the virtual addresses are equivalent to the physical addresses.

6.36 vm.h

Go to the documentation of this file.

```
00001 #ifndef MPX_VM_H
00002 #define MPX_VM_H
00003
00009 #include <stddef.h>
00010
00019 void *kmalloc(size_t size, int align, void **phys_addr);
00020
00026 void vm_init(void);
00027
00028 #endif
```

6.37 print_format.h

```
00001 //
00002 // Created by Andrew Bowie on 2/1/23.
00003 //
00004
00005 #ifndef F_R_I_D_A_Y_PRINT_FORMAT_H
00006 #define F_R_I_D_A_Y_PRINT_FORMAT_H
00007
00008 #include "color.h"
00009 #include "stdbool.h"
00010
00012 typedef enum {
00013     BOLD = 0,
00014     UNDERLINE = 1,
00015     ITALIC = 2,
00016     INVISIBLE = 3,
00017     INVERSE = 4,
00018     BLINKING = 5,
00019     STRIKETHROUGH = 6,
00020 } format_code_t;
00021
00028 bool is_format_code(format_code_t format_code);
00029
00036 void set_format_code(format_code_t format_code, bool active);
00037
00041 void clear_formats();
00042
00043 #endif //F_R_I_D_A_Y_PRINT_FORMAT_H
```

6.38 include/processes.h File Reference

Provided system process and user processes for testing.

Functions

- void **proc1** (void)

  A test process that prints a message then yields, exiting after 1 iteration.
- void **proc2** (void)

  A test process that prints a message then yields, exiting after 2 iterations.
- void **proc3** (void)

  A test process that prints a message then yields, exiting after 3 iterations.
- void **proc4** (void)

  A test process that prints a message then yields, exiting after 4 iterations.
- void **proc5** (void)

  A test process that prints a message then yields, exiting after 5 iterations.
- void sys_idle_process (void)

  System idle process.
- void comwrite (void)

  This process attempts to write a message to the serial device.
- void **comread** (void)

  This process writes a prompt to the serial device, and then reads user input which is then printed back to the device.
- void **iocom25** (void)

  This process attempts to write a message to the serial device 25 times and then exits.
- void **iocom** (void)

  This process attempts to write a message to the serial device until suspended and terminated.

6.38.1 Detailed Description

Provided system process and user processes for testing.

6.38.2 Function Documentation

6.38.2.1 comwrite()

```
void comwrite (
            void  )
```

This process attempts to write a message to the serial device.

This should be the first test process executed when testing R6.

6.38.2.2 sys_idle_process()

```
void sys_idle_process (
            void  )
```

System idle process.

Used in dispatching. It will be dispatched if NO other processes are available to execute. Must be a system process.

6.39 processes.h

Go to the documentation of this file.

```
00001 #ifndef MPX_PROCESSES_H
00002 #define MPX_PROCESSES_H
00003
00009 /* ************************************************************************
00010  The following functions are needed for Module R3.
00011 ************************************************************************ */
00012
00016 void proc1(void);
00017
00021 void proc2(void);
00022
00026 void proc3(void);
00027
00031 void proc4(void);
00032
00036 void proc5(void);
00037
00038 /* ************************************************************************
00039  The following function is needed for Module R4.
00040 ************************************************************************ */
00041
00046 void sys_idle_process(void);
00047
00048 /* ************************************************************************
00049  The following functions are needed for Module R6.
00050 ************************************************************************ */
00051
00056 void comwrite(void);
00057
00062 void comread(void);
00063
00067 void iocom25(void);
00068
00072 void iocom(void);
00073
00074 #endif
```

6.40 include/stdio.h File Reference

Contains useful functions for standard IO.

```
#include "stddef.h"
#include "stdbool.h"
```

Functions

- char getc (void)
  Reads a single ASCII character from standard input.
- char pollc (void)
  Polls a single ASCII character from standard input.
- char * gets (char *str_buf, size_t buf_len)
  Reads a string of input from the standard input source.
- void print (const char *str)
  Prints a null-terminated string to standard output.
- int printf (const char *str,...)
  Prints the string with formatting to standard outpu.
- void println (const char *str)
  Prints a null-terminated string, then a new line, to standard output.
- void **clearscr** (void)
  Clears the screen.

6.40.1 Detailed Description

Contains useful functions for standard IO.

6.40.2 Function Documentation

6.40.2.1 getc()

```
char getc (
            void  )
```

Reads a single ASCII character from standard input.

Returns

The character read

6.40.2.2 gets()

```
char * gets (
            char * str_buf,
            size_t buf_len )
```

Reads a string of input from the standard input source.

Parameters

str_buf	the buffer to store the string in.
buf_len	the amount of bytes to read. (The buffer should be at least one byte longer)

Returns

a pointer to the read array.

6.40.2.3 pollc()

```
char pollc (
            void  )
```

Polls a single ASCII character from standard input.

If no characters are available, 0 is returned.

Returns

The character polled.

6.40.2.4 print()

```
void print (
            const char * str )
```

Prints a null-terminated string to standard output.

Parameters

str	the string.

6.40.2.5 printf()

```
int printf (
            const char * str,
             ...  )
```

Prints the string with formatting to standard outpu.

Parameters

str	the string to print.
...	the formatting objects.

Returns

0 if successful, -1 if there was a formatting error.

6.40.2.6 println()

```
void println (
            const char * str )
```

Prints a null-terminated string, then a new line, to standard output.

Parameters

str	the string.

6.41 stdio.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Andrew Bowie on 1/13/23.
00003 //
00004
00005 #ifndef F_R_I_D_A_Y_STDIO_H
00006 #define F_R_I_D_A_Y_STDIO_H
00007
00008 #include "stddef.h"
00009 #include "stdbool.h"
00010
00020 char getc(void);
00021
00027 char pollc(void);
00028
00035 char *gets(char *str_buf, size_t buf_len);
00036
00041 void print(const char *str);
00042
00049 int printf(const char *str, ...);
00050
00055 void println(const char *str);
00056
00060 void clearscr(void);
00061
00062 #endif //F_R_I_D_A_Y_STDIO_H
```

6.42 include/stdlib.h File Reference

A subset of standard C library functions.

Functions

- int atoi (const char *s)

  Convert an ASCII string to an integer.
- char * itoa (int i, char *str_buf, int buf_len)

  Convert a signed integer to a string.
- char * itoa_base (int i, int base, char *str_buf, int buf_len)

  Convert a signed integer to a string.

6.42.1 Detailed Description

A subset of standard C library functions.

6.42.2 Function Documentation

6.42.2.1 atoi()

```
int atoi (
            const char * s )
```

Convert an ASCII string to an integer.

Parameters

s	A NUL-terminated string

Returns

The value of the string converted to an integer

6.42.2.2 itoa()

```
char * itoa (
            int i,
            char * str_buf,
            int buf_len )
```

Convert a signed integer to a string.

Parameters

i	the integer to convert
str_buf	the buffer to store the integer in
buf_len	the string buffer length

Returns

the created string from the integer

6.42.2.3 itoa_base()

```
char * itoa_base (
            int i,
            int base,
            char * str_buf,
            int buf_len )
```

Convert a signed integer to a string.

Parameters

i	the integer to convert
base	the base of the number
str_buf	the buffer to store the integer in
buf_len	the string buffer length

Returns

the created string from the integer

6.43 stdlib.h

Go to the documentation of this file.

```
00001 #ifndef MPX_STDLIB_H
00002 #define MPX_STDLIB_H
00003
00014 int atoi(const char *s);
00015
00023 char *itoa(int i, char *str_buf, int buf_len);
00024
00033 char *itoa_base(int i, int base, char *str_buf, int buf_len);
00034
00035 #endif
```

6.44 include/string.h File Reference

A subset of standard C library functions.

```
#include <stddef.h>
#include "stdarg.h"
#include "stdbool.h"
```

Functions

- bool first_label_matches (const char *str1, const char *label)

  Checks if the given string's first part matches the label.
- void * memcpy (void *restrict dst, const void *restrict src, size_t n)

  Copy a region of memory.
- void * memset (void *address, int c, size_t n)

  Fill a region of memory.
- char * strcpy (char *str_dest, const char *str_src, size_t maxlen)

  Copies the data from the string source into the string destination.
- int strcmp (const char *s1, const char *s2)

  Compares two strings.
- int strcicmp (const char *s1, const char *s2)

  Compares two strings, ignoring case.
- char * str_strip_whitespace (char *str, char *buffer, size_t buf_len)

  Strips leading and trailing whitespace from the given string.
- size_t strlen (const char *s)

  Returns the length of a string.
- char * str_to_upper (char *str, char *buffer, int buf_len)

  Converts the given string to upper case.
- char * str_to_lower (char *str, char *buffer, int buf_len)

  Converts the given string to lower case.
- char * **strtok** (char *restrict s1, const char *restrict s2)

  Split string into tokens TODO.
- char * sprintf (const char *format, char *str, size_t buf_len,...)

  Formats the string with normal C formatting options.
- char * vsprintf (const char *format, char *str, size_t buf_len, va_list varargs)

  Formats the string with normal C formatting options.
- char split_once_after (const char *string, const char *split_after, char buff[ ], int buff_len)

  Returns string located after where to split, orginal string returned if not split.
- bool starts_with (const char *string, const char *starts_with)

  Returns true if string starts with given string.
- bool ci_starts_with (const char *string, const char *prefix)

  Returns true if the string starts with the given prefix.
- int split (const char *string, char split_at, int word_length, char buff[ ][word_length], int words)

  Splits the given string at character saving into a 2D buffer.
- int substring (const char *string, int start, int end, char buff[ ], int buff_size)

  Splits the given string at character saving into a 2D buffer.

6.44.1 Detailed Description

A subset of standard C library functions.

6.44.2 Function Documentation

6.44.2.1 ci_starts_with()

```
bool ci_starts_with (
            const char * string,
            const char * prefix )
```

Returns true if the string starts with the given prefix.

Case is ignored.

Parameters

string	the string to be tested.
prefix	the prefix of the string.

Returns

true if the string starts with the prefix.

6.44.2.2 first_label_matches()

```
bool first_label_matches (
            const char * str1,
            const char * label )
```

Checks if the given string's first part matches the label.

Parameters

str1	the string.
label	the label.

Returns

if the string matches the label.

6.44.2.3 memcpy()

```
void * memcpy (
            void *restrict dst,
            const void *restrict src,
            size_t n )
```

Copy a region of memory.

Parameters

dst	The destination memory region
src	The source memory region
n	The number of bytes to copy

Returns

A pointer to the destination memory region

6.44.2.4 memset()

```
void * memset (
            void * address,
            int c,
            size_t n )
```

Fill a region of memory.

Parameters

address	The start of the memory region
c	The byte to fill memory with
n	The number of bytes to fill

Returns

A pointer to the filled memory region

6.44.2.5 split()

```
int split (
            const char * string,
            char split_at,
            int word_length,
            char buff[][word_length],
            int words )
```

Splits the given string at character saving into a 2D buffer.

Parameters

string	string to be split
split_at	character to split at
wordlength	length of the column dimension of buffer must match buff dimension
words	number of rows (words) available in buff

Returns

error codes 0 is successful, negative if not

6.44.2.6 split_once_after()

```
char split_once_after (
            const char * string,
            const char * split_after,
            char buff[],
            int buff_len )
```

Returns string located after where to split, orginal string returned if not split.

Parameters

string	string to be split
split↩ At	string that chooses where to split

Returns

the string split or not

6.44.2.7 sprintf()

```
char * sprintf (
            const char * format,
            char * str,
            size_t buf_len,
             ...  )
```

Formats the string with normal C formatting options.

Parameters

format	the string format.
str	the buffer to store the resulting string in.
buf_len	the length of the provided string buffer.
...	the formatting values.

Returns

the formatted string.

6.44.2.8 starts_with()

```
bool starts_with (
            const char * string,
            const char * starts_with )
```

Returns true if string starts with given string.

Parameters

string	string to be tested
starts_with	given string to start with

Returns

if string starts with starts_with string

6.44.2.9 str_strip_whitespace()

```
char * str_strip_whitespace (
            char * str,
            char * buffer,
            size_t buf_len )
```

Strips leading and trailing whitespace from the given string.

Parameters

str	the string to strip from.
buffer	the buffer to store the resulting string in, or NULL if the strip should be done in place.
buf_len	the length of the buffer.

Returns

a pointer to the resulting string, or NULL if it failed.

6.44.2.10 str_to_lower()

```
char * str_to_lower (
            char * str,
            char * buffer,
            int buf_len )
```

Converts the given string to lower case.

If the provided buffer is null, overwrites the original string.

Parameters

str	the original string.
buffer	the buffer to store the string in, or NULL if the original string should be overwritten.
buf_len	the length of the buffer. If buffer is NULL, can be any number.

Returns

a pointer to the lower case string, or NULL if the buffer was too small to store the resulting string.

6.44.2.11 str_to_upper()

```
char * str_to_upper (
            char * str,
            char * buffer,
            int buf_len )
```

Converts the given string to upper case.

If the provided buffer is null, overwrites the original string.

Parameters

str	the original string.
buffer	the buffer to store the string in, or NULL if the original string should be overwritten.
buf_len	the length of the buffer. If buffer is NULL, can be any number.

Returns

a pointer to the upper case string, or NULL if the buffer was too small to store the resulting string.

6.44.2.12 strcicmp()

```
int strcicmp (
            const char * s1,
            const char * s2 )
```

Compares two strings, ignoring case.

Parameters

s1	The first string to compare
s2	The second string to compare

Returns

0 if strings are equal, <0 if s1 is lexicographically before s2, >0 otherwise

6.44.2.13 strcmp()

```
int strcmp (
            const char * s1,
            const char * s2 )
```

Compares two strings.

Parameters

s1	The first string to compare
s2	The second string to compare

Returns

0 if strings are equal, <0 if s1 is lexicographically before s2, >0 otherwise

6.44.2.14 strcpy()

```
char * strcpy (
            char * str_dest,
            const char * str_src,
            size_t maxlen )
```

Copies the data from the string source into the string destination.

If maxlen is exceeded, it only copies that amount of chars over.

Parameters

str_dest	the string destination.
str_src	the string source.
maxlen	the maximum amount of bytes to copy. Note that maxlen does not include the null terminator.

Returns

a pointer to the string, or NULL if there was an error.

6.44.2.15 strlen()

```
size_t strlen (
            const char * s )
```

Returns the length of a string.

Parameters

s	A NUL-terminated string

Returns

The number of bytes in the string (not counting NUL terminator)

6.44.2.16 substring()

```
int substring (
            const char * string,
            int start,
            int end,
            char buff[],
            int buff_size )
```

Splits the given string at character saving into a 2D buffer.

Parameters

string	string to be spliced
start	index to start at
end	index to end at
buff	buffer to save result to
buff_size	length of buff

Returns

error codes 0 is successful, negative if not

6.44.2.17 vsprintf()

```
char * vsprintf (
            const char * format,
            char * str,
            size_t buf_len,
            va_list varargs )
```

Formats the string with normal C formatting options.

Parameters

format	the string format.
str	the buffer to store the resulting string in.
buf_len	the length of the provided string buffer.
...	the formatting values.

Returns

the formatted string.

6.45 string.h

Go to the documentation of this file.

```
00001 #ifndef MPX_STRING_H
00002 #define MPX_STRING_H
00003
00004 #include <stddef.h>
00005 #include "stdarg.h"
00006 #include "stdbool.h"
00007
00019 bool first_label_matches(const char *str1, const char *label);
00020
00028 void* memcpy(void * restrict dst, const void * restrict src, size_t n);
00029
00037 void* memset(void *address, int c, size_t n);
00038
00047 char *strcpy(char *str_dest, const char *str_src, size_t maxlen);
00048
00055 int strcmp(const char *s1, const char *s2);
00056
00063 int strcicmp(const char *s1, const char *s2);
00064
00073 char *str_strip_whitespace(char *str, char *buffer, size_t buf_len);
00074
00080 size_t strlen(const char *s);
00081
00093 char *str_to_upper(char *str, char *buffer, int buf_len);
00094
00106 char *str_to_lower(char *str, char *buffer, int buf_len);
00107
00112 char* strtok(char * restrict s1, const char * restrict s2);
00113
00122 char *sprintf(const char *format, char *str, size_t buf_len, ...);
00123
00132 char *vsprintf(const char *format, char *str, size_t buf_len, va_list varargs);
00133
00140 char split_once_after(const char* string, const char* split_after, char buff[], int buff_len);
00141
00148 bool starts_with(const char* string, const char* starts_with);
00149
00157 bool ci_starts_with(const char *string, const char *prefix);
00158
00167 int split(const char *string, char split_at, int word_length, char buff[][word_length], int words);
00168
00178 int substring(const char* string, int start, int end, char buff[], int buff_size);
00179 #endif
```

6.46 include/sys_req.h File Reference

System request function and constants.

```
#include <mpx/device.h>
```

Macros

- #define **INVALID_OPERATION** (-1)
- #define **INVALID_BUFFER** (-2)
- #define **INVALID_COUNT** (-3)

Enumerations

- enum **op_code** {
  EXIT , **IDLE** , **READ** , **WRITE** ,
  SHUTDOWN }

Functions

- int sys_req (op_code op,...)
  Request an MPX kernel operation.

6.46.1 Detailed Description

System request function and constants.

6.46.2 Function Documentation

6.46.2.1 sys_req()

```
int sys_req (
              op_code op,
               ...  )
```

Request an MPX kernel operation.

Parameters

op_code	One of READ, WRITE, IDLE, or EXIT
...	As required for READ or WRITE

Returns

Varies by operation

6.47 sys_req.h

Go to the documentation of this file.

```
00001 #ifndef MPX_SYS_REQ_H
00002 #define MPX_SYS_REQ_H
00003
00004 #include <mpx/device.h>
00005
00011 typedef enum {
00012     EXIT,
00013     IDLE,
00014     READ,
00015     WRITE,
00016     SHUTDOWN
00017 } op_code;
00018
00019 // error codes
00020 #define INVALID_OPERATION   (-1)
00021 #define INVALID_BUFFER      (-2)
00022 #define INVALID_COUNT       (-3)
00023
00030 int sys_req(op_code op, ...);
00031
00032 #endif
```

6.48 time_zone.h

```
00001
00002 #ifndef F_R_I_D_A_Y_TIME_ZONE_H
00003 #define F_R_I_D_A_Y_TIME_ZONE_H
00004
00011 typedef struct {
00013     const char *tz_label;
00015     const char *tz_longformat;
00017     const int tz_hour_offset;
00019     const int tz_minute_offset;
00021     const char* tz_city;
00022 } time_zone_t;
00023
00029 const time_zone_t **get_all_timezones(void);
00030
00036 const time_zone_t *get_timezone(const char *tz_label);
00037
00038 #endif //F_R_I_D_A_Y_TIME_ZONE_H
```

6.49 kernel/alarm.c File Reference

Contains logic to create alarms for the OS.

```
#include "stdio.h"
#include "stddef.h"
#include "mpx/pcb.h"
#include "string.h"
#include "mpx/clock.h"
#include "sys_req.h"
#include "stdlib.h"
```

Classes

- struct alarm_params

  The parameters used to pass into the alarm function.

Typedefs

- typedef struct alarm_params **alarm_structure**

  The parameters used to pass into the alarm function.

Functions

- bool is_time_after (const int *now, const int *check)

  Check if the given time array of hours, minutes, seconds is after the other.
- bool shouldAlarm (const int *time_array, time_zone_t *tz)

  Checks current time to see if alarm should alert.
- void alarm_function (int *time_array, const char *message, time_zone_t *time_zone)

  The alarm function used by the alarm processes.
- bool create_new_alarm (int *time_array, const char *message)

  Creates a new pcb that will display message at or after given time.

6.49.1 Detailed Description

Contains logic to create alarms for the OS.

6.49.2 Function Documentation

6.49.2.1 alarm_function()

```
void alarm_function (
            int * time_array,
            const char * message,
            time_zone_t * time_zone )
```

The alarm function used by the alarm processes.

Parameters

time_array	the time array to go off at.
creation_time	the time the alarm was created.
message	the message to send to the user.
time_zone	the timezone to use for the alarm.

Authors

Kolby Eisenhauer

6.49.2.2 create_new_alarm()

```
bool create_new_alarm (
            int * time_array,
            const char * message )
```

Creates a new pcb that will display message at or after given time.

Parameters

time_array	the time to display message
message	message to display

Returns

true if the alarm was created, false if it failed.

Author

Kolby Eisenhauer, Andrew Bowie

6.49.2.3 is_time_after()

```
bool is_time_after (
            const int * now,
            const int * check )
```

Check if the given time array of hours, minutes, seconds is after the other.

Parameters

now	the time array considered to be 'now'
check	the time to check at.

Returns

true if it is after.

6.49.2.4 shouldAlarm()

```
bool shouldAlarm (
            const int * time_array,
            time_zone_t * time_zone )
```

Checks current time to see if alarm should alert.

Parameters

time_array	the time to display message
time_zone	the timezone the alarm was created in.

Returns

boolean if should alarm

Author

Kolby Eisenhauer

6.50 kernel/sys_call.c File Reference

This file contains the sys_call function which is used to do context switching.

```
#include "mpx/pcb.h"
#include "sys_req.h"
```

Functions

- struct context * sys_call (op_code action, struct context *ctx)

  The main system call function, implementing the IDLE, EXIT, and SHUTDOWN system requests.

6.50.1 Detailed Description

This file contains the sys_call function which is used to do context switching.

6.50.2 Function Documentation

6.50.2.1 sys_call()

```
struct context * sys_call (
            op_code action,
            struct context * ctx )
```

The main system call function, implementing the IDLE, EXIT, and SHUTDOWN system requests.

Parameters

action	the action to perform.
ctx	the current PCB context.

Returns

a pointer to the next context to load.

Author

Andrew Bowie, Zachary Ebert, Kolby Eisenhauer