MacaroniOS

Version: R2

Generated by Doxygen 1.9.8

Chapter 1

Macaroni Penguins

CS450: Operating Systems Structure (Fall 2025)

See the repo at https://github.com/WVU-CS450/MacaroniPenguins.

1.1 GETTING STARTED

Install WSL if you need to:
```
wsl --install -d ubuntu
```

Clone this repo into a linux environment (WSL, Ubuntu, etc):
```
git clone https://github.com/WVU-CS450/MacaroniPenguins.git
```

Prep your linux environment by running the following commands:
```
sudo apt update
sudo apt install -y clang make nasm git binutils-i686-linux-gnu qemu-system-x86 gdb
```

Then run `make` and `./mpx.sh`.

For more information, either run the `help` command (or `help verbose`) inside of MacaroniOS, or consult the doc/USER-GUIDE.pdf.

1.2 CONTRIBUTING

After making changes, running `version` will show that your working directory is 'dirty'. This simply means that you have uncommitted changes.

Ensure you have checked out the correct branch and pulled its latest changes. Stage/add the relevant files before committing them.

Now you can run `make clean` and `make` again, run `./mpx.sh`, and finally run `version` to see your latest commit hash and showing that your working directory is 'clean'.

When you're done, add your contributions to dev/CONTRIBUTIONS.docx and save it as doc/CONTRIBUTIONS.pdf.

1.3 DOXYGEN

Install doxygen and dependancies:

```
sudo apt update
sudo apt install -y doxygen texlive-full texlive-latex-base texlive-latex-extra wslu
```

> If you get some errors, maybe try `sudo apt upgrade -y`. If you are getting an "WSL Interoperability is disabled" error, see `WSL issue 13449` and run the permenant fix codeblock.

Create the configuration file (convention is a Doxyfile):

```
doxygen -g Doxyfile
```

Edit the file to your liking, reference the `doxygen manual` if needed, then run doxygen:

```
doxygen
```

> When releasing a new version of MacaroniOS, remember to change the `PROJECT_NUMER` (to R1, R2, etc) and `OUTPUT_DIRECTORY` (from `dev/doxygen` to `doc`). Also remember to change `user/version.c`.

Then `cd` into the generated latex directory and run:

```
make pdf
```

In the same directory, a `refman.pdf` is generated. Save this file as `doc/PROGRAMMER-GUIDE.pdf`.

Alternatively, you can run the doxyfile shell script to automate everything above.

1.4 RELEASES

View all of the `Releases` or view previous documentation in `doc/.legacy/`.

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

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

Chapter 4

Data Structure Documentation

4.1 pcb Struct Reference

Collaboration diagram for pcb:

Data Fields

- char **name** [PCB_NAME_MAX_LEN]
- enum process_class **process_class**
- int **priority**
- enum execution_state **execution_state**
- enum dispatch_state **dispatch_state**
- char * **stack**
- char * **stack_ptr**
- struct pcb * **next**
- struct pcb * **prev**

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

- include/pcb.h

4.2 pcb_queue Struct Reference

Collaboration diagram for pcb_queue:

Data Fields

- struct pcb * **head**
- struct pcb * **tail**

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

- include/pcb.h

4.3 rtc_date_t Struct Reference

Data Fields

- uint8_t **day**
- uint8_t **month**
- uint8_t **year**

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

- include/clock.h

4.4 rtc_time_t Struct Reference

Data Fields

- uint8_t **second**
- uint8_t **minute**
- uint8_t **hour**

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

- include/clock.h

4.5 stores Struct Reference

Struct that stores data relating to the time , and Hours.

4.5.1 Detailed Description

Struct that stores data relating to the time , and Hours.

Struct that stores data relating to the date , and Year.

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

- include/clock.h

Chapter 5

File Documentation

5.1 include/clock.h File Reference

Handles accesses to the Real Time Clock (RTC)

```
#include <string.h>
#include <sys_req.h>
#include <stdint.h>
#include <mpx/interrupts.h>
#include <mpx/io.h>
```

Include dependency graph for clock.h:

5.2 clock.h

Go to the documentation of this file.

```
00001 #ifndef CLOCK_H
00002 #define CLOCK_H
00003
00004 #include <string.h>
00005 #include <sys_req.h>
00006 #include <stdint.h>
00007 #include <mpx/interrupts.h>
00008 #include <mpx/io.h>
00009
00019 typedef struct {
00020     uint8_t second;
00021     uint8_t minute;
00022     uint8_t hour;
00023 } rtc_time_t;
00024
00029 typedef struct {
00030     uint8_t day;
00031     uint8_t month;
00032     uint8_t year;   // Last two digits of the year
00033 } rtc_date_t;
00034
00035
00040 void get_time(rtc_time_t *time);
00041
00046 void set_time(const rtc_time_t *time);
00047
00052 void get_date(rtc_date_t *date);
00053
```

```
00058 void set_date(const rtc_date_t *date);
00059
00064 void print_time(rtc_time_t *time);
00065
00070 void print_date(const rtc_date_t *date);
00071
00075 void clock_help(void);
00076
00081 void clock_command(const char *args);
00082
00083 //---- Helper Functions ----//
00084
00090 void my_strcat(char *dest, const char *src);
00091
00097 void my_strcpy(char *dest, const char *src);
00098
00104 void rtc_write(uint8_t reg, uint8_t value);
00105
00110 uint8_t rtc_read(uint8_t reg);
00111
00117 uint8_t bin_to_bcd(uint8_t value);
00118
00124 uint8_t bcd_to_bin(uint8_t value);
00125
00129 void tz_correction(void);
00130
00131 #endif
```

5.3 include/comhand.h File Reference

Command handler interface for the OS. Reads from the polling input and executes commands.

This graph shows which files directly or indirectly include this file:

Functions

- void **com_startup** (void)

  Prints a welcome message and penguin ASCII art to the terminal.
- void trim_Input (char *str)

  Trim function to remove \n and \r from the string.
- void **comhand** (void)

  Enters a loop and waits for the user to input commands.

5.3.1 Detailed Description

Command handler interface for the OS. Reads from the polling input and executes commands.

5.3.2 Function Documentation

5.3.2.1 trim_Input()

```
void trim_Input (
            char * str )
```

Trim function to remove \n and \r from the string.

Parameters

str	string variable to trim

5.4 comhand.h

Go to the documentation of this file.

```
00001 #ifndef COMHAND_H
00002 #define COMHAND_H
00003
00013 void com_startup(void);
00014
00019 void trim_Input(char *str);
00020
00024 void comhand(void);
00025
00026 #endif
```

5.5 include/ctype.h File Reference

A subset of standard C library functions.

Functions

- int isspace (int c)

5.5.1 Detailed Description

A subset of standard C library functions.

5.5.2 Function Documentation

5.5.2.1 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

5.6 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
00016 #endif
```

5.7 include/exit.h File Reference

Header file for the exit command used in the command handler. Exits the terminal when called and confirmed by the user.

Functions

- void **exit_help** (void)
- int exit_command (const char *args)

  Begins the shutdown process when the user types 'exit' in the terminal. Confirmation by typing 'Y' or 'n' is then required to completely exit.

5.7.1 Detailed Description

Header file for the exit command used in the command handler. Exits the terminal when called and confirmed by the user.

Author

Caleb Edwards

5.7.2 Function Documentation

5.7.2.1 exit_command()

```
int exit_command (
            const char * args )
```

Begins the shutdown process when the user types 'exit' in the terminal. Confirmation by typing 'Y' or 'n' is then required to completely exit.

Parameters

arg_counter	Counts the number of arguments input.
arg_vector	Stores the arguments.

Returns

int return 1 to confirm exit and 0 to return to terminal.

5.8 exit.h

Go to the documentation of this file.

```
00001 #ifndef EXIT_H
00002 #define EXIT_H
00010 void exit_help(void);
00011
00020 int exit_command(const char *args);
00021
00022 #endif
```

5.9 include/help.h File Reference

Header for the help command used in command handler. Used to list the commands available to the user.

Functions

- void **help_message** (void)
- void **help_command** (const char *args)

  Prints all commands available into the terminal when the user types 'help' in the input.
- void **help_verbose** (void)

  Prints all commands individual help functions.

5.9.1 Detailed Description

Header for the help command used in command handler. Used to list the commands available to the user.

Author

Caleb Edwards

5.10 help.h

Go to the documentation of this file.

```
00001 #ifndef HELP_H
00002 #define HELP_H
00010 void help_message(void);
00011
00016 void help_command(const char *args);
00017
00021 void help_verbose(void);
00022 #endif
```

5.11 include/itoa.h File Reference

Declaration for interger-to-ASCII conversion.

This graph shows which files directly or indirectly include this file:

Functions

- void itoa (int num, char ∗buffer)

  Converts an integer to a C-string.

5.11.1 Detailed Description

Declaration for interger-to-ASCII conversion.

5.11.2 Function Documentation

5.11.2.1 itoa()

```
void itoa (
            int num,
            char * buffer )
```

Converts an integer to a C-string.

Parameters

num	The integer to convert.
buffer	Pointer to an array to store the string.

5.12 itoa.h

Go to the documentation of this file.

```
00001 #ifndef ITOA_H
00002 #define ITOA_H
00003
00015 void itoa(int num, char* buffer);
00016
00017 #endif
```

5.13 include/itoBCD.h File Reference

Function that converts an integer into a string that is representative of the binary coded decimal format of the input integer.

Functions

- void itoBCD (int num, char ∗buffer)

5.13.1 Detailed Description

Function that converts an integer into a string that is representative of the binary coded decimal format of the input integer.

5.13.2 Function Documentation

5.13.2.1 itoBCD()

```
void itoBCD (
            int num,
            char * buffer )
```

Convert an integer to an Binary Coded Decimal

Parameters

int	Integer being converted into a Binary Coded Decimal
s	A buffer to hold the created string

5.14 itoBCD.h

Go to the documentation of this file.

```
00001 #ifndef ITOBCD_H
00002 #define ITOBCD_H
00003
00017 void itoBCD(int num, char* buffer);
00018
00019 #endif
```

5.15 include/memory.h File Reference

MPX-specific dynamic memory functions.

`#include <stddef.h>`
Include dependency graph for memory.h:

Functions

- void * sys_alloc_mem (size_t size)
- int sys_free_mem (void *ptr)
- void sys_set_heap_functions (void *(*alloc_fn)(size_t), int(*free_fn)(void *))

5.15.1 Detailed Description

MPX-specific dynamic memory functions.

5.15.2 Function Documentation

5.15.2.1 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

5.15.2.2 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

5.15.2.3 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

5.16 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
```

5.17 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
```

5.18 include/mpx/gdt.h File Reference

Kernel functions to initialize the Global Descriptor Table.

Functions

- void gdt_init (void)

5.18.1 Detailed Description

Kernel functions to initialize the Global Descriptor Table.

5.18.2 Function Documentation

5.18.2.1 gdt_init()

```
void gdt_init (
             void  )
```

Creates and installs the Global Descriptor Table.

5.19 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
```

5.20 include/mpx/interrupts.h File Reference

Kernel functions related to software and hardware interrupts.

This graph shows which files directly or indirectly include this file:

Macros

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

Functions

- void irq_init (void)
- void pic_init (void)
- void idt_init (void)
- void idt_install (int vector, void(*handler)(void *))

5.20.1 Detailed Description

Kernel functions related to software and hardware interrupts.

5.20.2 Macro Definition Documentation

5.20.2.1 cli

```
#define cli( ) __asm__ volatile ("cli")
```

Disable interrupts

5.20.2.2 sti

```
#define sti( ) __asm__ volatile ("sti")
```

Enable interrupts

5.20.3 Function Documentation

5.20.3.1 idt_init()

```
void idt_init (
            void  )
```

Creates and installs the Interrupt Descriptor Table.

5.20.3.2 idt_install()

```
void idt_install (
            int vector,
            void(*)(void *) handler )
```

Installs an interrupt handler

5.20.3.3 irq_init()

```
void irq_init (
            void  )
```

Installs the initial interrupt handlers for the first 32 IRQ lines. Most do a panic for now.

5.20.3.4 pic_init()

```
void pic_init (
            void  )
```

Initializes the programmable interrupt controllers and performs the necessary remapping of IRQs. Leaves interrupts turned off.

5.21 interrupts.h

Go to the documentation of this file.

```
00001 #ifndef MPX_INTERRUPTS_H
00002 #define MPX_INTERRUPTS_H
00003
00010 #define cli() __asm__ volatile ("cli")
00011
00013 #define sti() __asm__ volatile ("sti")
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
```

5.22 include/mpx/io.h File Reference

Kernel macros to read and write I/O ports.

This graph shows which files directly or indirectly include this file:

Macros

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

5.22.1 Detailed Description

Kernel macros to read and write I/O ports.

5.22.2 Macro Definition Documentation

5.22.2.1 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

5.22.2.2 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

5.23 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 #pragma clang diagnostic ignored "-Wgnu-statement-expression"
00023 #define inb(port) ({                              \
00024       unsigned char r;                            \
```

```
00025       __asm__ volatile ("inb %%dx, %%al" : "=a" (r) : "d" (port));  \
00026       r;                                                            \
00027     })
00028
00029 #endif
```

5.24 include/mpx/panic.h File Reference

Common system functions and definitions.

`#include <stdnoreturn.h>`
Include dependency graph for panic.h:

Functions

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

5.24.1 Detailed Description

Common system functions and definitions.

5.24.2 Function Documentation

5.24.2.1 __attribute__()

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

Kernel panic. Prints an error message and halts.

Parameters

msg	A message to display before halting

5.25 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
```

5.26 include/mpx/serial.h File Reference

Kernel functions and constants for handling serial I/O.

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

Include dependency graph for serial.h:

Functions

- int serial_init (device dev)
- int serial_out (device dev, const char *buffer, size_t len)
- int serial_poll (device dev, char *buffer, size_t len)

5.26.1 Detailed Description

Kernel functions and constants for handling serial I/O.

5.26.2 Function Documentation

5.26.2.1 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

5.26.2.2 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

5.26.2.3 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

5.27 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
00035 int serial_poll(device dev, char *buffer, size_t len);
00036
00037 #endif
```

5.28 include/mpx/vm.h File Reference

Kernel functions for virtual memory and primitive allocation.

`#include <stddef.h>`
Include dependency graph for vm.h:

Functions

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

5.28.1 Detailed Description

Kernel functions for virtual memory and primitive allocation.

5.28.2 Function Documentation

5.28.2.1 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

5.28.2.2 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.

5.29 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
```

5.30 include/pcb.h File Reference

Process Control Block queue and stack functions.

`#include <stdint.h>`
Include dependency graph for pcb.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct pcb
- struct pcb_queue

Macros

- #define **PCB_NAME_MAX_LEN** 16
- #define **PCB_STACK_MIN_SIZE** 1024

Enumerations

- enum **process_class** { **CLASS_SYSTEM** = 0 , **CLASS_USER** = 1 }
- enum **execution_state** { **STATE_READY** = 0 , **STATE_RUNNING** = 1 , **STATE_BLOCKED** = 2 }
- enum **dispatch_state** { **DISPATCH_ACTIVE** = 0 , **DISPATCH_SUSPENDED** = 1 }

Functions

- struct pcb ∗ **pcb_allocate** (void)
- int **pcb_free** (struct pcb ∗ptr)
- struct pcb ∗ **pcb_setup** (const char ∗name, int process_class, int priority)
- struct pcb ∗ **pcb_find** (const char ∗name)
- void pcb_insert (struct pcb ∗ptr)

  Insert a PCB into the appropriate queue.
- int pcb_remove (struct pcb ∗ptr)

  Remove a PCB from its current queue.

Variables

- struct pcb_queue ready_queue
- struct pcb_queue **blocked_queue**
- struct pcb_queue **suspended_ready_queue**
- struct pcb_queue **suspended_blocked_queue**

5.30.1 Detailed Description

Process Control Block queue and stack functions.

Defines the data structure types and functions for operating the PCB queues.

- Ready Queue: PCBs waiting to be executed by priority.
- Blocked Queue: PCBs waiting on something else to happen before its ready.
- Suspended-Ready Queue: PCBs not in the stack but ready.
- Suspended-Blocked Queue: PCBs not in the stack and blocked.

5.30.2 Function Documentation

5.30.2.1 pcb_insert()

```
void pcb_insert (
            struct pcb * ptr )
```

Insert a PCB into the appropriate queue.

Chooses the correct queue based on the PCB's dispatch and execution state, then inserts it. If the target queue is the ready queue, PCBs are inserted by priority (FIFO within same priority). Otherwise, they are appended FIFO at the tail.

Parameters

ptr	Pointer to the PCB to insert.

5.30.2.2 pcb_remove()

```
int pcb_remove (
            struct pcb * ptr )
```

Remove a PCB from its current queue.

Locates the PCB's queue based on its state, then detaches it by fixing neighboring links. Clears the PCB's next/prev pointers.

Parameters

ptr	Pointer to the PCB to remove.

Returns

0 on success, -1 if PCB is NULL or not found in any queue.

5.30.3 Variable Documentation

5.30.3.1 ready_queue

```
struct pcb_queue ready_queue  [extern]
```

TODO Will add in error messaging at different points

5.31 pcb.h

Go to the documentation of this file.

```
00001
00014 #ifndef PCB_H
00015 #define PCB_H
00016 #include <stdint.h>
00017 #define PCB_NAME_MAX_LEN 16
00018 #define PCB_STACK_MIN_SIZE 1024
00019
00020 enum process_class{
00021     CLASS_SYSTEM = 0,
00022     CLASS_USER = 1
00023 };
00024
00025 enum execution_state{
00026     STATE_READY = 0,
00027     STATE_RUNNING = 1,
00028     STATE_BLOCKED = 2
00029 };
00030
00031 enum dispatch_state{
00032     DISPATCH_ACTIVE = 0,
00033     DISPATCH_SUSPENDED = 1
00034 };
00035
00036 struct pcb{
00037     char name[PCB_NAME_MAX_LEN];
00038     enum process_class process_class;
00039     int priority; // 0 (highest) to 9
00040     enum execution_state execution_state;
00041     enum dispatch_state dispatch_state;
00042     char* stack; // Dynamically allocated, might manually allocate based on memory management.
00043     char* stack_ptr;
00044     struct pcb* next;
00045     struct pcb* prev;
00046 };
00047
00048 struct pcb_queue{
00049     struct pcb* head;
00050     struct pcb* tail;
00051 };
00052
00053 // Queue initialization
00054 extern struct pcb_queue ready_queue;
00055 extern struct pcb_queue blocked_queue;
```

```
00056 extern struct pcb_queue suspended_ready_queue;
00057 extern struct pcb_queue suspended_blocked_queue;
00058
00059 // PCB Kernal Function
00060 struct pcb* pcb_allocate(void);
00061
00062 int pcb_free(struct pcb* ptr);
00063
00064 struct pcb* pcb_setup(const char* name, int process_class, int priority);
00065
00066 struct pcb* pcb_find(const char* name);
00067
00078 void pcb_insert(struct pcb* ptr);
00079
00089 int pcb_remove(struct pcb* ptr);
00090
00091 #endif
```

5.32 include/processes.h File Reference

Provided system process and user processes for testing.

Functions

- void proc1 (void)
- void proc2 (void)
- void proc3 (void)
- void proc4 (void)
- void proc5 (void)
- void sys_idle_process (void)

5.32.1 Detailed Description

Provided system process and user processes for testing.

5.32.2 Function Documentation

5.32.2.1 proc1()

```
void proc1 (
            void  )
```

A test process that prints a message then yields, exiting after 1 iteration.

5.32.2.2 proc2()

```
void proc2 (
            void  )
```

A test process that prints a message then yields, exiting after 2 iterations.

5.32.2.3 proc3()

```
void proc3 (
            void  )
```

A test process that prints a message then yields, exiting after 3 iterations.

5.32.2.4 proc4()

```
void proc4 (
            void  )
```

A test process that prints a message then yields, exiting after 4 iterations.

5.32.2.5 proc5()

```
void proc5 (
            void  )
```

A test process that prints a message then yields, exiting after 5 iterations.

5.32.2.6 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.

5.33 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 #endif
```

5.34 include/ready.h File Reference

Commands to suspend or resume a process.

`#include "pcb.h"`
Include dependency graph for ready.h:

Functions

- void **suspend_help** (void)

  Prints the help message for the suspend command.
- int suspend_pcb (const char *process_name)

  Puts a non-system process in the suspended state, and moves it to the appropriate queue.
- void suspend_command (const char *args)

  Handles command-line arguments for the suspend command.
- void **resume_help** (void)

  Prints the help message for the resume command.
- int resume_pcb (const char *process_name)

  Puts a process in the active (not suspended) state, and moves it to the appropriate queue.
- void resume_command (const char *args)

  Handles command-line arguments for the resume command.

5.34.1 Detailed Description

Commands to suspend or resume a process.

5.34.2 Function Documentation

5.34.2.1 resume_command()

```
void resume_command (
            const char * args )
```

Handles command-line arguments for the resume command.

Parameters

args	Command argument string (process name or "help").

5.34.2.2 resume_pcb()

```
int resume_pcb (
            const char * process_name )
```

Puts a process in the active (not suspended) state, and moves it to the appropriate queue.

Parameters

process_name	Process's name (checks for validity)

Returns

int 0 for success, -1 for invalid process/name, and 1 if already active.

5.34.2.3 suspend_command()

```
void suspend_command (
            const char * args )
```

Handles command-line arguments for the suspend command.

Parameters

args	Command argument string (process name or "help").

5.34.2.4 suspend_pcb()

```
int suspend_pcb (
            const char * process_name )
```

Puts a non-system process in the suspended state, and moves it to the appropriate queue.

Parameters

process_name	Process's name (checks for validity)

Returns

int 0 for success, -1 for invalid process/name, -2 if given a system process, and 1 if already suspended.

5.35 ready.h

Go to the documentation of this file.

```
00001 #ifndef PCB_READY_H
00002 #define PCB_READY_H
00003
00009 #include "pcb.h"
00010
```

```
00014 void suspend_help(void);
00015
00021 int suspend_pcb(const char* process_name);
00022
00027 void suspend_command(const char *args);
00028
00032 void resume_help(void);
00033
00039 int resume_pcb(const char* process_name);
00040
00045 void resume_command(const char *args);
00046
00047 #endif
```

5.36 setPriority.h

```
00001 #ifndef SETPRIORITY_H
00002 #define SETPRIORITY_H
00003
00004 #include <pcb.h>
00005 #include <sys_req.h>
00006 #include <string.h>
00007
00008 void set_priority_command(const char* args);
00009
00010 void set_priority_help(void);
00011
00012 void setPriority(char* name, int newPriority);
00013
00014
00015 #endif
```

5.37 showPCB.h

```
00001 #ifndef SHOWPCB_H
00002 #define SHOWPCB_H
00003
00004 #include <pcb.h>
00005 #include <sys_req.h>
00006 #include <string.h>
00007 #include <comhand.h>
00008 #include <itoa.h>
00009
00010 void show_command(const char* args);
00011
00012 void showPCB(const char* name);
00013
00014 void showReady(void);
00015
00016 void showBlocked(void);
00017
00018 void showAllPCB(void);
00019
00020 void show_pcb_help(void);
00021
00022 void showSuspended(void);
00023
00024 #endif
```

5.38 include/stdlib.h File Reference

A subset of standard C library functions.

Functions

- int atoi (const char *s)

5.38.1 Detailed Description

A subset of standard C library functions.

5.38.2 Function Documentation

5.38.2.1 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

5.39 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
00016 #endif
```

5.40 include/string.h File Reference

A subset of standard C library functions.

`#include <stddef.h>`
Include dependency graph for string.h: This graph shows which files directly or indirectly include this file:

Functions

- void * memcpy (void *restrict dst, const void *restrict src, size_t n)
- void * memset (void *address, int c, size_t n)
- int strcmp (const char *s1, const char *s2)
- int **strncmp** (const char *s1, const char *s2, unsigned int n)
- size_t strlen (const char *s)
- char * strtok (char *restrict s1, const char *restrict s2)

  Split string into tokens.
- char * strncpy (char *dest, const char *src, unsigned int num_of_chars)

  Copy a string with length limit.

5.40.1 Detailed Description

A subset of standard C library functions.

5.40.2 Function Documentation

5.40.2.1 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

5.40.2.2 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

5.40.2.3 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

5.40.2.4 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)

5.40.2.5 strncpy()

```
char * strncpy (
            char * dest,
            const char * src,
            unsigned int num_of_chars )
```

Copy a string with length limit.

Parameters

dest	Destination buffer.
src	Source string.
num_of_chars	Maximum number of characters to copy.

Returns

Pointer to destination buffer.

5.40.2.6 strtok()

```
char * strtok (
             char *restrict s1,
             const char *restrict s2 )
```

Split string into tokens.

Parameters

s1	String to tokenize (NULL to continue tokenizing).
s2	Delimiter characters.

Returns

Pointer to next token, or NULL if none.

5.41 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
00018 void* memcpy(void * restrict dst, const void * restrict src, size_t n);
00019
00027 void* memset(void *address, int c, size_t n);
00028
00035 int strcmp(const char *s1, const char *s2);
00036
00037 int strncmp(const char *s1, const char *s2, unsigned int n);
00038
00044 size_t strlen(const char *s);
00045
00053 char* strtok(char * restrict s1, const char * restrict s2);
00054
00063 char* strncpy(char* dest, const char* src, unsigned int num_of_chars);
00064 #endif
```

5.42 include/sys_req.h File Reference

System request function and constants.

`#include <mpx/device.h>`
Include dependency graph for sys_req.h: This graph shows which files directly or indirectly include this file:

Macros

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

Enumerations

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

Functions

- int sys_req (op_code op,...)

5.42.1 Detailed Description

System request function and constants.

5.42.2 Function Documentation

5.42.2.1 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

5.43 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 } op_code;
00017
00018 // error codes
00019 #define INVALID_OPERATION   (-1)
00020 #define INVALID_BUFFER      (-2)
00021 #define INVALID_COUNT       (-3)
00022
00029 int sys_req(op_code op, ...);
00030
00031 #endif
```

5.44 include/version.h File Reference

Displays the current version of MacaroniOS.

Macros

- #define **GIT_DATE** "unknown"
- #define **GIT_HASH** "unknown"
- #define **GIT_DIRTY** "unknown"

Functions

- void **version_help** (void)

  Prints help information related to the version command.
- void **version_latest** (void)

  Displays the latest version.
- void **version_history** (void)

  Displays the past and present versions.
- void version_command (const char *args)

  Main handler for the version command.

5.44.1 Detailed Description

Displays the current version of MacaroniOS.

5.44.2 Function Documentation

5.44.2.1 version_command()

```
void version_command (
            const char * args )
```

Main handler for the version command.

Parameters

args	The argument string passed after 'version'

5.45 version.h

Go to the documentation of this file.

```
00001 #ifndef VERSION_H
00002 #define VERSION_H
00003
00004 #ifndef GIT_DATE
00005 #define GIT_DATE "unknown"
00006 #endif
00007
00008 #ifndef GIT_HASH
00009 #define GIT_HASH "unknown"
00010 #endif
00011
00012 #ifndef GIT_DIRTY
00013 #define GIT_DIRTY "unknown"
00014 #endif
00015
00025 void version_help(void);
00026
00030 void version_latest(void);
00031
00035 void version_history(void);
00036
00041 void version_command(const char *args);
00042
00043 #endif
```

Index