tinyOS

Generated by Doxygen 1.9.4

1 Class Index 1

1.1 Class List . . . 1

2 File Index 3

2.1 File List . . . 3

3 Class Documentation 5

3.1 __attribute__ Struct Reference . . . 5

3.1.1 Detailed Description . . . 5

3.1.2 Member Data Documentation . . . 5

3.1.2.1 base . . . 5

3.1.2.2 flags . . . 5

3.1.2.3 hOffset . . . 6

3.1.2.4 limit . . . 6

3.1.2.5 lOffset . . . 6

3.1.2.6 sel . . . 6

3.1.2.7 zero . . . 6

3.2 reg Struct Reference . . . 6

3.2.1 Detailed Description . . . 7

3.2.2 Member Data Documentation . . . 7

3.2.2.1 cs . . . 7

3.2.2.2 ds . . . 7

3.2.2.3 eax . . . 8

3.2.2.4 ebp . . . 8

3.2.2.5 ebx . . . 8

3.2.2.6 ecx . . . 8

3.2.2.7 edi . . . 8

3.2.2.8 edx . . . 8

3.2.2.9 eflags . . . 9

3.2.2.10 eip . . . 9

3.2.2.11 errorCode . . . 9

3.2.2.12 esi . . . 9

3.2.2.13 esp . . . 9

3.2.2.14 intNo . . . 9

3.2.2.15 ss . . . 9

3.2.2.16 useresp . . . 9

4 File Documentation 11

4.1 apps/cliTools.c File Reference . . . 11

4.1.1 Detailed Description . . . 11

4.1.2 Macro Definition Documentation . . . 12

4.1.2.1 CALCPRES . . . 12

4.1.3 Function Documentation . . . 12

4.1.3.1 printBanner() . . . 12
4.1.3.2 userInputHandler() . . . 12
4.2 cliTools.c . . . 13
4.3 apps/cliTools.h File Reference . . . 14
4.3.1 Detailed Description . . . 14
4.3.2 Function Documentation . . . 14
4.3.2.1 printBanner() . . . 15
4.3.2.2 userInputHandler() . . . 15
4.4 cliTools.h . . . 16
4.5 drivers/ata.c File Reference . . . 16
4.5.1 Detailed Description . . . 16
4.5.2 Function Documentation . . . 16
4.5.2.1 readSectors() . . . 16
4.5.2.2 writeSectors() . . . 17
4.6 ata.c . . . 18
4.7 drivers/ata.h File Reference . . . 19
4.7.1 Detailed Description . . . 19
4.7.2 Macro Definition Documentation . . . 19
4.7.2.1 DRIVE_BUSY . . . 20
4.7.2.2 DRIVE_ERROR . . . 20
4.7.2.3 DRIVE_FAULTY . . . 20
4.7.2.4 DRIVE_READY . . . 20
4.7.2.5 DRIVE_WAITING . . . 20
4.7.3 Function Documentation . . . 20
4.7.3.1 readSectors() . . . 20
4.7.3.2 writeSectors() . . . 21
4.8 ata.h . . . 22
4.9 drivers/keyboard.c File Reference . . . 22
4.9.1 Detailed Description . . . 23
4.9.2 Function Documentation . . . 23
4.9.2.1 KBInit() . . . 23
4.9.3 Variable Documentation . . . 23
4.9.3.1 keycodeToAscii . . . 23
4.10 keyboard.c . . . 24
4.11 drivers/keyboard.h File Reference . . . 24
4.11.1 Detailed Description . . . 25
4.11.2 Macro Definition Documentation . . . 25
4.11.2.1 BACKSPACE . . . 25
4.11.2.2 ENTER . . . 26
4.11.2.3 SCMAX . . . 26
4.11.2.4 UP_ARROW . . . 26
4.11.2.5 USER_BUFF_SIZE . . . 26

4.11.3 Function Documentation . . . 26
4.11.3.1 KBInit() . . . 26
4.12 keyboard.h . . . 27
4.13 drivers/ports.c File Reference . . . 27
4.14 ports.c . . . 27
4.15 drivers/ports.h File Reference . . . 27
4.15.1 Detailed Description . . . 28
4.15.2 Function Documentation . . . 28
4.15.2.1 portByteIn() . . . 28
4.15.2.2 portByteOut() . . . 29
4.15.2.3 portWordIn() . . . 29
4.15.2.4 portWordOut() . . . 30
4.16 ports.h . . . 30
4.17 drivers/screen.c File Reference . . . 30
4.17.1 Detailed Description . . . 31
4.17.2 Function Documentation . . . 31
4.17.2.1 clearScreen() . . . 31
4.17.2.2 getVGAOffset() . . . 32
4.17.2.3 printStr() . . . 32
4.17.2.4 printStrAtPos() . . . 32
4.17.2.5 putchar() . . . 33
4.17.2.6 putcharAtPos() . . . 33
4.17.2.7 removeLastChar() . . . 34
4.17.2.8 setVGAOffset() . . . 34
4.18 screen.c . . . 35
4.19 screen.h . . . 36
4.20 kernel/idt.c File Reference . . . 37
4.20.1 Detailed Description . . . 37
4.20.2 Function Documentation . . . 37
4.20.2.1 setIDT() . . . 38
4.20.2.2 setIDTGate() . . . 38
4.20.3 Variable Documentation . . . 38
4.20.3.1 IDT . . . 38
4.20.3.2 IDTReg . . . 39
4.21 idt.c . . . 39
4.22 kernel/idt.h File Reference . . . 39
4.22.1 Detailed Description . . . 40
4.22.2 Macro Definition Documentation . . . 40
4.22.2.1 IDTNB . . . 40
4.22.2.2 KERNCS . . . 40
4.22.3 Function Documentation . . . 40
4.22.3.1 setIDT() . . . 41

4.22.3.2 setIDTGate() . . . 41
4.22.4 Variable Documentation . . . 41
4.22.4.1 IDT . . . 41
4.22.4.2 IDTReg . . . 42
4.23 idt.h . . . 42
4.24 kernel/isr.c File Reference . . . 42
4.24.1 Detailed Description . . . 43
4.24.2 Function Documentation . . . 43
4.24.2.1 IRQHandler() . . . 43
4.24.2.2 ISRHandler() . . . 44
4.24.2.3 ISRInstall() . . . 44
4.24.2.4 regInterruptHandler() . . . 45
4.24.3 Variable Documentation . . . 46
4.24.3.1 exceptionMessages . . . 46
4.24.3.2 interruptHandlers . . . 46
4.25 isr.c . . . 46
4.26 kernel/isr.h File Reference . . . 48
4.26.1 Detailed Description . . . 50
4.26.2 Macro Definition Documentation . . . 50
4.26.2.1 IRQ0 . . . 50
4.26.2.2 IRQ1 . . . 50
4.26.2.3 IRQ10 . . . 50
4.26.2.4 IRQ11 . . . 51
4.26.2.5 IRQ12 . . . 51
4.26.2.6 IRQ13 . . . 51
4.26.2.7 IRQ14 . . . 51
4.26.2.8 IRQ15 . . . 51
4.26.2.9 IRQ2 . . . 51
4.26.2.10 IRQ3 . . . 52
4.26.2.11 IRQ4 . . . 52
4.26.2.12 IRQ5 . . . 52
4.26.2.13 IRQ6 . . . 52
4.26.2.14 IRQ7 . . . 52
4.26.2.15 IRQ8 . . . 52
4.26.2.16 IRQ9 . . . 53
4.26.3 Typedef Documentation . . . 53
4.26.3.1 ISR . . . 53
4.26.4 Function Documentation . . . 53
4.26.4.1 IRQHandler() . . . 53
4.26.4.2 ISRHandler() . . . 54
4.26.4.3 ISRInstall() . . . 54
4.26.4.4 regInterruptHandler() . . . 55

4.27 isr.h . . . 55
4.28 kernel/kernel.c File Reference . . . 57
4.28.1 Detailed Description . . . 57
4.28.2 Function Documentation . . . 57
4.28.2.1 main() . . . 58
4.29 kernel.c . . . 58
4.30 kernel/timer.c File Reference . . . 58
4.30.1 Detailed Description . . . 59
4.30.2 Function Documentation . . . 59
4.30.2.1 getTick() . . . 59
4.30.2.2 initTimer() . . . 59
4.30.3 Variable Documentation . . . 60
4.30.3.1 tick . . . 60
4.31 timer.c . . . 60
4.32 kernel/timer.h File Reference . . . 60
4.32.1 Detailed Description . . . 61
4.32.2 Function Documentation . . . 61
4.32.2.1 getTick() . . . 61
4.32.2.2 initTimer() . . . 61
4.33 timer.h . . . 62
4.34 libs/memory.c File Reference . . . 62
4.34.1 Detailed Description . . . 63
4.34.2 Function Documentation . . . 63
4.34.2.1 calloc() . . . 63
4.34.2.2 mallok() . . . 63
4.34.2.3 mem_cpy() . . . 64
4.34.2.4 mem_set() . . . 64
4.34.3 Variable Documentation . . . 65
4.34.3.1 freeMemPosition . . . 65
4.35 memory.c . . . 65
4.36 libs/memory.h File Reference . . . 66
4.36.1 Detailed Description . . . 66
4.36.2 Function Documentation . . . 66
4.36.2.1 callok() . . . 66
4.36.2.2 mallok() . . . 67
4.36.2.3 mem_cpy() . . . 67
4.36.2.4 mem_set() . . . 68
4.37 memory.h . . . 68
4.38 libs/random.c File Reference . . . 68
4.38.1 Detailed Description . . . 69
4.38.2 Function Documentation . . . 69
4.38.2.1 randK() . . . 69

4.38.2.2 randomK() . . . 70
4.38.2.3 resetRandom() . . . 70
4.38.3 Variable Documentation . . . 70
4.38.3.1 rngIndex . . . 70
4.38.3.2 rngTable . . . 70
4.39 random.c . . . 71
4.40 random.h . . . 71
4.41 libs/strings.c File Reference . . . 71
4.41.1 Detailed Description . . . 72
4.41.2 Function Documentation . . . 72
4.41.2.1 append() . . . 72
4.41.2.2 biggestWord() . . . 73
4.41.2.3 getNthWord() . . . 73
4.41.2.4 hexToAscii() . . . 74
4.41.2.5 str_cmp() . . . 74
4.41.2.6 str_cpy() . . . 75
4.41.2.7 str_len() . . . 75
4.41.2.8 strReverse() . . . 76
4.41.2.9 strSplit() . . . 77
4.41.2.10 strToL() . . . 78
4.41.2.11 wordCount() . . . 78
4.42 strings.c . . . 79
4.43 libs/strings.h File Reference . . . 81
4.43.1 Detailed Description . . . 82
4.43.2 Function Documentation . . . 82
4.43.2.1 append() . . . 82
4.43.2.2 hexToAscii() . . . 83
4.43.2.3 str_cmp() . . . 84
4.43.2.4 str_len() . . . 84
4.43.2.5 strReverse() . . . 85
4.43.2.6 strSplit() . . . 85
4.43.2.7 strToL() . . . 86
4.43.2.8 wordCount() . . . 86
4.44 strings.h . . . 87
4.45 utils.c . . . 87
4.46 utils.h . . . 88

Index **89**

Chapter 1

Class Index

1.1 Class List

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

__attribute__ 5
reg
Representation of the asm register pushed in interrupts.asm more info inside kernel/interrupts.↩
asm 6

Chapter 2

File Index

2.1 File List

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

apps/cliTools.c
: Function library containing cool stuff for cli . . . 11

apps/cliTools.h
: Function containing usefull command line tools & function . . . 14

drivers/ata.c
: ATA Drivers library . . . 16

drivers/ata.h
: Headers of the ata driver library . . . 19

drivers/keyboard.c
: Function library related to keyboard and inputs . . . 22

drivers/keyboard.h
: Function library header related to keyboard and inputs . . . 24

drivers/ports.c
: Function library containing everything related to ports . . . 27

drivers/ports.h
: Function header library related to ports writing and reading . . . 27

drivers/screen.c
: Function library linked to the screen & display . . . 30

drivers/screen.h . . . 36

kernel/idt.c
: Function linked to the Interrupt descriptor table . . . 37

kernel/idt.h
: Library containing everything related to the Interrupt Descriptor Table . . . 39

kernel/isr.c
: Functions required to setup & use the interrupt service routines . . . 42

kernel/isr.h
: Headers containing everything related to the interrupt service routines . . . 48

kernel/kernel.c
: OS Entry point . . . 57

kernel/timer.c
: Function library related to timers, date and time . . . 58

kernel/timer.h
: Function headers library related to timers, date and time . . . 60

libs/memory.c
: Rewritten function related to memory model to replace the one in stdlib . . . 62

libs/memory.h
File where the memory model is defined . . . 66
libs/random.c
Function library to get pseudo random numbers . . . 68
libs/random.h . . . 71
libs/strings.c
Function library to imitate strings.c from the stdlib . . . 71
libs/strings.h
Functions header library to imitate strings.c from the stdlib . . . 81
libs/utils.c . . . 87
libs/utils.h . . . 88

Chapter 3

Class Documentation

3.1 __attribute__ Struct Reference

Public Attributes

- u16 lOffset
- u16 sel
- u8 zero
- u8 flags
- u16 hOffset
- u16 limit
- u32 base

3.1.1 Detailed Description

Definition at line 20 of file idt.h.

3.1.2 Member Data Documentation

3.1.2.1 base

```
u32 __attribute__::base
```

Definition at line 30 of file idt.h.

3.1.2.2 flags

```
u8 __attribute__::flags
```

Definition at line 24 of file idt.h.

3.1.2.3 hOffset

```
u16 __attribute__::hOffset
```

Definition at line 25 of file idt.h.

3.1.2.4 limit

```
u16 __attribute__::limit
```

Definition at line 29 of file idt.h.

3.1.2.5 lOffset

```
u16 __attribute__::lOffset
```

Definition at line 21 of file idt.h.

3.1.2.6 sel

```
u16 __attribute__::sel
```

Definition at line 22 of file idt.h.

3.1.2.7 zero

```
u8 __attribute__::zero
```

Definition at line 23 of file idt.h.

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

- kernel/idt.h

3.2 reg Struct Reference

representation of the asm register pushed in interrupts.asm more info inside kernel/interrupts.asm

```
#include <isr.h>
```

Public Attributes

- u32 ds
- u32 edi
- u32 esi
- u32 ebp
- u32 esp
- u32 ebx
- u32 edx
- u32 ecx
- u32 eax
- u32 intNo
- u32 errorCode
- u32 eip
- u32 cs
- u32 eflags
- u32 useresp
- u32 ss

3.2.1 Detailed Description

representation of the asm register pushed in interrupts.asm more info inside kernel/interrupts.asm

Definition at line 18 of file isr.h.

3.2.2 Member Data Documentation

3.2.2.1 cs

```
u32 reg::cs
```

Definition at line 22 of file isr.h.

3.2.2.2 ds

```
u32 reg::ds
```

Definition at line 19 of file isr.h.

3.2.2.3 eax

```
u32 reg::eax
```

Definition at line 20 of file isr.h.

3.2.2.4 ebp

```
u32 reg::ebp
```

Definition at line 20 of file isr.h.

3.2.2.5 ebx

```
u32 reg::ebx
```

Definition at line 20 of file isr.h.

3.2.2.6 ecx

```
u32 reg::ecx
```

Definition at line 20 of file isr.h.

3.2.2.7 edi

```
u32 reg::edi
```

Definition at line 20 of file isr.h.

3.2.2.8 edx

```
u32 reg::edx
```

Definition at line 20 of file isr.h.

3.2.2.9 eflags

```
u32 reg::eflags
```

Definition at line 22 of file isr.h.

3.2.2.10 eip

```
u32 reg::eip
```

Definition at line 22 of file isr.h.

3.2.2.11 errorCode

```
u32 reg::errorCode
```

Definition at line 21 of file isr.h.

3.2.2.12 esi

```
u32 reg::esi
```

Definition at line 20 of file isr.h.

3.2.2.13 esp

```
u32 reg::esp
```

Definition at line 20 of file isr.h.

3.2.2.14 intNo

```
u32 reg::intNo
```

Definition at line 21 of file isr.h.

3.2.2.15 ss

```
u32 reg::ss
```

Definition at line 22 of file isr.h.

3.2.2.16 useresp

```
u32 reg::useresp
```

Definition at line 22 of file isr.h.

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

- kernel/isr.h

Chapter 4

File Documentation

4.1 apps/cliTools.c File Reference

function library containing cool stuff for cli

```
#include "cliTools.h"
```

Include dependency graph for cliTools.c:

Macros

- #define CALCPRES 10

Functions

- void printBanner ()
  function to display on screen a cool banner
- void userInputHandler (char *buff)

4.1.1 Detailed Description

function library containing cool stuff for cli

Author

Théodore MARTIN

Version

0.1

Date

2023-03-22

Definition in file cliTools.c.

4.1.2 Macro Definition Documentation

4.1.2.1 CALCPRES

```
#define CALCPRES 10
```

Definition at line 10 of file cliTools.c.

4.1.3 Function Documentation

4.1.3.1 printBanner()

```
void printBanner ( )
```

function to display on screen a cool banner

Definition at line 12 of file cliTools.c.

```
00013 {
00014     printStr("    .---.                                                                  .-"'-.
      \n");
00015     printStr("    |   |                              '    _    \\       \n");
00016     printStr("    '---'              _...._   .--.  /   /` '.      \\     \n");
00017     printStr("    .---.                        .'          '.  |__|.      |     \\  '     \n");
00018     printStr("    |   |            .      .-.      ..--.|   '       |  '    \n");
00019     printStr("    |   |           |  '   '  ||  |\\    \\     / /     \n");
00020     printStr("    |   |   _   _   |  |   |  ||  | `.       ` ..'  / _    \n");
00021     printStr("    |   |  | '  / | |  |   |  ||  |    '-...-'`.'   |   \n");
00022     printStr("    |   | .'  | .'  | |  |   |  ||  |            .      | / \n");
00023     printStr("    |   | /   | /   | |  |   |  ||__|          .'.'| |// \n");
00024     printStr(" __.'       '|    ''.    | |  |   |  |           .'.'.-'  /  \n");
00025     printStr("|       ' '    .'|  '/|  |   |  |  |          .'       \\_.'      \n");
00026     printStr("|____.'       `-'  `--' '--'   '--'                      \n");
00027 }
```

4.1.3.2 userInputHandler()

```
void userInputHandler (
            char * buff )
```

Parameters

in	*buff*	the keyboard content buffer

Returns

void

Definition at line 29 of file cliTools.c.

```
00030 {
00031     u32 wordNumber = wordCount(buff, ' ');
00032     if (wordNumber == 0) {
00033         printStr("No command entered...\n");
00034     }
00035     string *words = strSplit(buff, ' ', &wordNumber);
00036     char *firstWord = words[0];
00037     if(!str_cmp(firstWord, "HELP")) {
00038         printStr("WIKI");
00039     } else if(!str_cmp(firstWord, "ADD")) {
00040         if ((wordNumber) != 3) {
00041             printStr("Pour utiliser la commande ADD, entrez ADD ainsi que deux nombres.\n");
00042             printStr("Exemple:  ADD 8.3 -5.6");
00043         } else {
00044             i64 num1 = (i64) strToL(words[1]);
00045             i64 num2 = (i64) strToL(words[2]);
00046             i64 res = num1 + num2;
00047             //char *buf = mallok(10 * sizeof(CALCPRES));
00048             //gcvt(res, CALCPRES-1, buff);
00049             printStr(words[1]);
00050             printStr(" + ");
00051             printStr(words[2]);
00052             printStr(" = ");
00053             //printStr(buff);
00054         }
00055     }
00056     else {
00057         printStr("JuniOs:  commande inconnue:  ");
00058         printStr(buff);
00059         printStr("\nPour obtenir une liste des commandes disponnibles, entrez \"HELP\"");
00060     }
00061 }
```

4.2 cliTools.c

Go to the documentation of this file.

```
00001
00009 #include "cliTools.h"
00010 #define CALCPRES 10
00011
00012 void printBanner()
00013 {
00014     printStr("    .---.                                                                      .-”’-.
    \n");
00015     printStr("    |   |                                ’    _     \\       \n");
00016     printStr("    ’---’              _...._   .--.  /   /` ’.       \\     \n");
00017     printStr("    .---.                        .’          ’.  |__|.       |     \\  ’     \n");
00018     printStr("    |   |           .      .-.      ..--.|   ’       |  ’    \n");
00019     printStr("    |   |           |  ’   ’  ||  |\\\    \\      / /     \n");
00020     printStr("    |   |  _    _  |  |   |  ||  | `.       ` ..’  / _    \n");
00021     printStr("    |   |  | ’  / | |  |   |  ||  |    ’-...-’`.’   |   \n");
00022     printStr("    |   | .’  | .’  | |  |   |  ||  |            .      | / \n");
00023     printStr("    |   | /  | /  | |  |   |  ||__|          .’.’| |// \n");
00024     printStr(" __.’      ’|   `’.    | |  |   |  |            .’.’.-’  /  \n");
00025     printStr("|      ’ ’   .’|  ’/|  |   |  |             .’       \\\_.’      \n");
00026     printStr("|____.’       `-’  `--’ ’--’   ’--’                          \n");
00027 }
00028
00029 void userInputHandler(char *buff)
00030 {
00031     u32 wordNumber = wordCount(buff, ' ');
00032     if (wordNumber == 0) {
00033         printStr("No command entered...\n");
00034     }
00035     string *words = strSplit(buff, ' ', &wordNumber);
00036     char *firstWord = words[0];
00037     if(!str_cmp(firstWord, "HELP")) {
00038         printStr("WIKI");
00039     } else if(!str_cmp(firstWord, "ADD")) {
00040         if ((wordNumber) != 3) {
00041             printStr("Pour utiliser la commande ADD, entrez ADD ainsi que deux nombres.\n");
00042             printStr("Exemple:  ADD 8.3 -5.6");
00043         } else {
00044             i64 num1 = (i64) strToL(words[1]);
00045             i64 num2 = (i64) strToL(words[2]);
00046             i64 res = num1 + num2;
00047             //char *buf = mallok(10 * sizeof(CALCPRES));
00048             //gcvt(res, CALCPRES-1, buff);
00049             printStr(words[1]);
00050             printStr(" + ");
00051             printStr(words[2]);
```

```
00052             printStr(" = ");
00053             //printStr(buff);
00054         }
00055     }
00056     else {
00057         printStr("JuniOs:  commande inconnue:  ");
00058         printStr(buff);
00059         printStr("\nPour obtenir une liste des commandes disponnibles, entrez \"HELP\"");
00060     }
00061 }
```

4.3 apps/cliTools.h File Reference

function containing usefull command line tools & function

```
#include "../drivers/screen.h"
#include "../drivers/keyboard.h"
#include "../libs/utils.h"
```

Include dependency graph for cliTools.h: This graph shows which files directly or indirectly include this file:

Functions

- void printBanner ()

  function to display on screen a cool banner
- void userInputHandler (char ∗)

4.3.1 Detailed Description

function containing usefull command line tools & function

Author

Théodore MARTIN

Version

0.1

Date

2023-03-22

Definition in file cliTools.h.

4.3.2 Function Documentation

4.3.2.1 printBanner()

```
void printBanner ( )
```

function to display on screen a cool banner

Definition at line 12 of file cliTools.c.

```
00013 {
00014     printStr("    .---.                                                                  .-"'-.
    \n");
00015     printStr("    |   |                              '    _     \\       \n");
00016     printStr("    '---'                 _...._   .--.  /   /` '.      \\     \n");
00017     printStr("    .---.                          .'          '.  |__|.       |     \\  '     \n");
00018     printStr("    |   |            .       .-.       ..--.|   '        |  '    \n");
00019     printStr("    |   |            |  '    '  ||  |\\     \\      / /     \n");
00020     printStr("    |   |   _    _   |  |    |  ||  | `.       ` ..'  / _    \n");
00021     printStr("    |   |  | '  / | |  |    |  ||  |    '-...-'`.'   |   \n");
00022     printStr("    |   | .'   | .'  | |   |    |  ||  |            .      | / \n");
00023     printStr("    |   | /   | /   | |   |    |  ||__|          .'.'| |// \n");
00024     printStr(" __.'       '|    `'.    | |   |   |  |           .'.'.-'  /  \n");
00025     printStr("|      ' '    .'|  '/|   |   |  |            .'       \\\_.'      \n");
00026     printStr("|____.'       `-'   `--' '--'   '--'                          \n");
00027 }
```

4.3.2.2 userInputHandler()

```
void userInputHandler (
            char * buff )
```

Parameters

in	*buff*	the keyboard content buffer

Returns

void

Definition at line 29 of file cliTools.c.

```
00030 {
00031     u32 wordNumber = wordCount(buff, ' ');
00032     if (wordNumber == 0) {
00033         printStr("No command entered...\n");
00034     }
00035     string *words = strSplit(buff, ' ', &wordNumber);
00036     char *firstWord = words[0];
00037     if(!str_cmp(firstWord, "HELP")) {
00038         printStr("WIKI");
00039     } else if(!str_cmp(firstWord, "ADD")) {
00040         if ((wordNumber) != 3) {
00041             printStr("Pour utiliser la commande ADD, entrez ADD ainsi que deux nombres.\n");
00042             printStr("Exemple:  ADD 8.3 -5.6");
00043         } else {
00044             i64 num1 = (i64) strToL(words[1]);
00045             i64 num2 = (i64) strToL(words[2]);
00046             i64 res = num1 + num2;
00047             //char *buf = mallok(10 * sizeof(CALCPRES));
00048             //gcvt(res, CALCPRES-1, buff);
00049             printStr(words[1]);
00050             printStr(" + ");
00051             printStr(words[2]);
00052             printStr(" = ");
00053             //printStr(buff);
00054         }
00055     }
00056     else {
00057         printStr("JuniOs:  commande inconnue:  ");
00058         printStr(buff);
00059         printStr("\nPour obtenir une liste des commandes disponnibles, entrez \"HELP\"");
00060     }
00061 }
```

4.4 cliTools.h

Go to the documentation of this file.

```
00001
00008 #ifndef __CLITOOLS__
00009 #define __CLITOOLS__
00010 #include "../drivers/screen.h"
00011 #include "../drivers/keyboard.h"
00012 #include "../libs/utils.h"
00013
00017 void printBanner();
00018
00026 void userInputHandler(char *);
00027
00028 #endif
```

4.5 drivers/ata.c File Reference

ATA Drivers library.

`#include "ata.h"`
Include dependency graph for ata.c:

Functions

- void readSectors (u32 addr, u32 LBA, u8 sectors)

  Reads the drive sectors.
- void writeSectors (u32 LBA, u8 sectors, u32 *bytes)

  Write data to the sectors.

4.5.1 Detailed Description

ATA Drivers library.

Author

Théodore MARTIN

Version

0.1

Date

2023-03-24

Definition in file ata.c.

4.5.2 Function Documentation

4.5.2.1 readSectors()

```
void readSectors (
            u32 addr,
            u32 LBA,
            u8 sectors )
```

Reads the drive sectors.

Parameters

in	*addr*	the address that will store the result
	LBA	the logical block address
	sectors	the number of sectors to read

Returns

void

Definition at line 31 of file ata.c.

```
00032 {
00033     waitUntilDriveNotBusy();
00034     /* standard procedure - described in https://wiki.osdev.org/ATA_PIO_Mode */
00035     /* setting up the registers */
00036     portByteOut(0x1F6,0xE0 | ((LBA >>24) & 0xF));
00037     portByteOut(0x1F2,sectors);
00038     portByteOut(0x1F3, (u8) LBA);
00039     portByteOut(0x1F4, (u8)(LBA >> 8));
00040     portByteOut(0x1F5, (u8)(LBA >> 16));
00041     /* sending the read command */
00042     portByteOut(0x1F7,0x20);
00043     u16 *castedAddr = (u16 *)addr;
00044     /* retrieving the data */
00045     u16 *maxAddr = (u16 *)addr + 256 * sectors;
00046     for (; castedAddr < maxAddr; castedAddr += 256){
00047         waitUntilDriveNotBusy();
00048         waitUntilDriveNotFaulty();
00049         for (u8 j = 0; j != 255; ++j) {
00050             *(castedAddr + j) = portWordIn(0x1F0);
00051         }
00052     }
00053 }
```

4.5.2.2 writeSectors()

```
void writeSectors (
            u32 LBA,
            u8 sectors,
            u32 * bytes )
```

Write data to the sectors.

Parameters

LBA	the LBA target address
sectors	the number of sectors to write
bytes	the localisation of the bytes to write

Returns

void

Definition at line 55 of file ata.c.

```
00056 {
00057     waitUntilDriveNotBusy();
00058     /* standard procedure - described in https://wiki.osdev.org/ATA_PIO_Mode */
00059     /* setting up the registers */
```

```
00060     portByteOut(0x1F6,0xE0 | ((LBA >>24) & 0xF));
00061     portByteOut(0x1F2,sectors);
00062     portByteOut(0x1F3, (u8) LBA);
00063     portByteOut(0x1F4, (u8)(LBA >> 8));
00064     portByteOut(0x1F5, (u8)(LBA >> 16));
00065     /* sending the write command */
00066     portByteOut(0x1F7,0x30);
00067     for (u8 i = 0; i < sectors; ++i) {
00068         waitUntilDriveNotBusy();
00069         waitUntilDriveNotFaulty();
00070         for (u8 j = 0; j != 255; ++j) {
00071             portWordOut(0x1F0, *(bytes + j));
00072         }
00073     }
00074 }
```

4.6 ata.c

Go to the documentation of this file.

```
00001
00008 #include "ata.h"
00009
00015 static void waitUntilDriveNotBusy()
00016 {
00017     while (portByteIn(0x1F7) & DRIVE_BUSY);
00018 }
00019
00025 static void waitUntilDriveNotFaulty()
00026 {
00027     while (portByteIn(0x1F5) & DRIVE_FAULTY);
00028 }
00029
00030 //Needs testing
00031 void readSectors(u32 addr, u32 LBA, u8 sectors)
00032 {
00033     waitUntilDriveNotBusy();
00034     /* standard procedure - described in https://wiki.osdev.org/ATA_PIO_Mode */
00035     /* setting up the registers */
00036     portByteOut(0x1F6,0xE0 | ((LBA >>24) & 0xF));
00037     portByteOut(0x1F2,sectors);
00038     portByteOut(0x1F3, (u8) LBA);
00039     portByteOut(0x1F4, (u8)(LBA >> 8));
00040     portByteOut(0x1F5, (u8)(LBA >> 16));
00041     /* sending the read command */
00042     portByteOut(0x1F7,0x20);
00043     u16 *castedAddr = (u16 *)addr;
00044     /* retrieving the data */
00045     u16 *maxAddr = (u16 *)addr + 256 * sectors;
00046     for (; castedAddr < maxAddr; castedAddr += 256){
00047         waitUntilDriveNotBusy();
00048         waitUntilDriveNotFaulty();
00049         for (u8 j = 0; j != 255; ++j) {
00050             *(castedAddr + j) = portWordIn(0x1F0);
00051         }
00052     }
00053 }
00054
00055 void writeSectors(u32 LBA, u8 sectors, u32 *bytes)
00056 {
00057     waitUntilDriveNotBusy();
00058     /* standard procedure - described in https://wiki.osdev.org/ATA_PIO_Mode */
00059     /* setting up the registers */
00060     portByteOut(0x1F6,0xE0 | ((LBA >>24) & 0xF));
00061     portByteOut(0x1F2,sectors);
00062     portByteOut(0x1F3, (u8) LBA);
00063     portByteOut(0x1F4, (u8)(LBA >> 8));
00064     portByteOut(0x1F5, (u8)(LBA >> 16));
00065     /* sending the write command */
00066     portByteOut(0x1F7,0x30);
00067     for (u8 i = 0; i < sectors; ++i) {
00068         waitUntilDriveNotBusy();
00069         waitUntilDriveNotFaulty();
00070         for (u8 j = 0; j != 255; ++j) {
00071             portWordOut(0x1F0, *(bytes + j));
00072         }
00073     }
00074 }
```

4.7 drivers/ata.h File Reference

Headers of the ata driver library.

```
#include "../libs/utils.h"
#include "./ports.h"
```

Include dependency graph for ata.h: This graph shows which files directly or indirectly include this file:

Macros

- #define DRIVE_BUSY 0x80
- #define DRIVE_READY 0x40
- #define DRIVE_WAITING 0x08
- #define DRIVE_ERROR 0X01
- #define DRIVE_FAULTY 0X20

Functions

- void readSectors (u32, u32, u8)

  Reads the drive sectors.
- void writeSectors (u32, u8, u32 ∗)

  Write data to the sectors.

4.7.1 Detailed Description

Headers of the ata driver library.

Author

Théodore MARTIN

Version

0.1

Date

2023-03-24 https://wiki.osdev.org/ATA_PIO_Mode

Definition in file ata.h.

4.7.2 Macro Definition Documentation

4.7.2.1 DRIVE_BUSY

```
#define DRIVE_BUSY 0x80
```

Definition at line 14 of file ata.h.

4.7.2.2 DRIVE_ERROR

```
#define DRIVE_ERROR 0X01
```

Definition at line 17 of file ata.h.

4.7.2.3 DRIVE_FAULTY

```
#define DRIVE_FAULTY 0X20
```

Definition at line 18 of file ata.h.

4.7.2.4 DRIVE_READY

```
#define DRIVE_READY 0x40
```

Definition at line 15 of file ata.h.

4.7.2.5 DRIVE_WAITING

```
#define DRIVE_WAITING 0x08
```

Definition at line 16 of file ata.h.

4.7.3 Function Documentation

4.7.3.1 readSectors()

```
void readSectors (
            u32 addr,
            u32 LBA,
            u8 sectors )
```

Reads the drive sectors.

Parameters

in	*addr*	the address that will store the result
	LBA	the logical block address
	sectors	the number of sectors to read

Returns

void

Definition at line 31 of file ata.c.

```
00032 {
00033     waitUntilDriveNotBusy();
00034     /* standard procedure - described in https://wiki.osdev.org/ATA_PIO_Mode */
00035     /* setting up the registers */
00036     portByteOut(0x1F6,0xE0 | ((LBA >>24) & 0xF));
00037     portByteOut(0x1F2,sectors);
00038     portByteOut(0x1F3, (u8) LBA);
00039     portByteOut(0x1F4, (u8)(LBA >> 8));
00040     portByteOut(0x1F5, (u8)(LBA >> 16));
00041     /* sending the read command */
00042     portByteOut(0x1F7,0x20);
00043     u16 *castedAddr = (u16 *)addr;
00044     /* retrieving the data */
00045     u16 *maxAddr = (u16 *)addr + 256 * sectors;
00046     for (; castedAddr < maxAddr; castedAddr += 256){
00047         waitUntilDriveNotBusy();
00048         waitUntilDriveNotFaulty();
00049         for (u8 j = 0; j != 255; ++j) {
00050             *(castedAddr + j) = portWordIn(0x1F0);
00051         }
00052     }
00053 }
```

4.7.3.2 writeSectors()

```
void writeSectors (
            u32 LBA,
            u8 sectors,
            u32 * bytes )
```

Write data to the sectors.

Parameters

LBA	the LBA target address
sectors	the number of sectors to write
bytes	the localisation of the bytes to write

Returns

void

Definition at line 55 of file ata.c.

```
00056 {
00057     waitUntilDriveNotBusy();
00058     /* standard procedure - described in https://wiki.osdev.org/ATA_PIO_Mode */
00059     /* setting up the registers */
```

```
00060     portByteOut(0x1F6,0xE0 | ((LBA >>24) & 0xF));
00061     portByteOut(0x1F2,sectors);
00062     portByteOut(0x1F3, (u8) LBA);
00063     portByteOut(0x1F4, (u8)(LBA >> 8));
00064     portByteOut(0x1F5, (u8)(LBA >> 16));
00065     /* sending the write command */
00066     portByteOut(0x1F7,0x30);
00067     for (u8 i = 0; i < sectors; ++i) {
00068         waitUntilDriveNotBusy();
00069         waitUntilDriveNotFaulty();
00070         for (u8 j = 0; j != 255; ++j) {
00071             portWordOut(0x1F0, *(bytes + j));
00072         }
00073     }
00074 }
```

4.8 ata.h

Go to the documentation of this file.

```
00001
00009 #ifndef __ATA__
00010 #define __ATA__
00011 #include "../libs/utils.h"
00012 #include "./ports.h"
00013
00014 #define DRIVE_BUSY 0x80
00015 #define DRIVE_READY 0x40
00016 #define DRIVE_WAITING 0x08
00017 #define DRIVE_ERROR 0X01
00018 #define DRIVE_FAULTY 0X20
00019
00029 void readSectors(u32, u32, u8);
00030
00040 void writeSectors(u32, u8, u32*);
00041 #endif
```

4.9 drivers/keyboard.c File Reference

function library related to keyboard and inputs

```
#include "keyboard.h"
#include "../kernel/isr.h"
#include "ports.h"
#include "screen.h"
#include "../apps/cliTools.h"
```

Include dependency graph for keyboard.c:

Functions

- void KBInit ()

  function linking the bios keyboard interrupt to the callback This function links the bios keyboard interrupt obtained with IRQs to our keyboard callback. It allows us to setup a buffer and obtain user input. It allows us to retrieve the keycode pressed that we then convert in the KBCallback to ascii char, using an hard coded table.

Variables

- const char keycodeToAscii [ ]

4.9.1 Detailed Description

function library related to keyboard and inputs

Author

your name Théodore MARTIN

Version

0.1

Date

2023-03-22

Definition in file keyboard.c.

4.9.2 Function Documentation

4.9.2.1 KBInit()

```
void KBInit ( )
```

function linking the bios keyboard interrupt to the callback This function links the bios keyboard interrupt obtained with IRQs to our keyboard callback. It allows us to setup a buffer and obtain user input. It allows us to retrieve the keycode pressed that we then convert in the KBCallback to ascii char, using an hard coded table.

Definition at line 70 of file keyboard.c.

```
00071 {
00072     regInterruptHandler(IRQ1, KBCallback);
00073 }
```

4.9.3 Variable Documentation

4.9.3.1 keycodeToAscii

```
const char keycodeToAscii[]
```

Initial value:

```
= { '?', '?', '1', '2', '3', '4', '5', '6',
    '7', '8', '9', '0', '-', '=', '<', '?', 'A', 'Z', 'E', 'R', 'T', 'Y',
        'U', 'I', 'O', 'P', '[', ']', '?', '?', 'Q', 'S', 'D', 'F', 'G',
        'H', 'J', 'K', 'L', 'M', '\"', '`', '?', '\\', 'W', 'X', 'C', 'V',
        'B', 'N', ',', ';', '.', '/', '?', '?', '?', ' '}
```

Definition at line 21 of file keyboard.c.

4.10 keyboard.c

Go to the documentation of this file.

```
00001
00008 #include "keyboard.h"
00009 #include "../kernel/isr.h"
00010 #include "ports.h"
00011 #include "screen.h"
00012 #include "../apps/cliTools.h"
00013
00014 /* Global variables */
00015 static char userInput[USER_BUFF_SIZE];
00016 static char lastCommand[USER_BUFF_SIZE];
00017 static u16 userInputLength = 0;
00018 static u16 lastInputLength = 0;
00019
00020 /* Edited a bit for azerty */
00021 const char keycodeToAscii[] = { '?', '?', '1', '2', '3', '4', '5', '6',
00022     '7', '8', '9', '0', '-', '=', '<', '?', 'A', 'Z', 'E', 'R', 'T', 'Y',
00023         'U', 'I', 'O', 'P', '[', ']', '?', '?', 'Q', 'S', 'D', 'F', 'G',
00024         'H', 'J', 'K', 'L', 'M', '\"', '`', '?', '\\', 'W', 'X', 'C', 'V',
00025         'B', 'N', ',', ';', '.', '/', '?', '?', '?', ' '};
00026
00027
00028 /* Private functions */
00029 static void KBCallback()
00030 {
00031     /*char temp[129];*/
00032     u8 keycode = portByteIn(0x60);
00033     /*iToA(keycode, temp);
00034 printStr(temp);
00035 putchar('\n')*/
00036     if (keycode > SCMAX)
00037         return;
00038     if (keycode == BACKSPACE) {
00039         if (userInputLength != 0) {
00040             userInput[userInputLength] = ' ';
00041             userInputLength--;
00042             removeLastChar();
00043         }
00044     } else if (keycode == ENTER) {
00045         putchar('\n');
00046         userInputHandler(userInput);
00047         mem_cpy(userInput, lastCommand, userInputLength);
00048         lastInputLength = userInputLength;
00049         userInput[0] = '\0';
00050         userInputLength = 0;
00051         printStr("\nuser@JuniOs:~$");
00052     /*needs a fix, idk why but not working */
00053     } else if(keycode == UP_ARROW && userInputLength == 0 && lastInputLength != 0) {
00054         mem_cpy(lastCommand, userInput, lastInputLength);
00055         userInputLength = lastInputLength;
00056         printStr(userInput);
00057     }
00058     else {
00059         char chr = keycodeToAscii[(int)keycode];
00060         if (userInputLength < USER_BUFF_SIZE) {
00061             userInput[userInputLength] = chr;
00062             //append(userInput, chr);
00063             userInputLength++;
00064             putchar(chr);
00065         }
00066     }
00067 }
00068
00069 /* public functions */
00070 void KBInit()
00071 {
00072     regInterruptHandler(IRQ1, KBCallback);
00073 }
```

4.11 drivers/keyboard.h File Reference

function library header related to keyboard and inputs

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

Macros

- #define SCMAX 57
- #define ENTER 0x1C /∗ Ascii definition ∗/
- #define BACKSPACE 0x0E /∗Ascii definition ∗/
- #define USER_BUFF_SIZE 256 /∗ WARNING: TYPE USED FOR buffLength should match∗/
- #define UP_ARROW 27

Functions

- void KBInit ()

  function linking the bios keyboard interrupt to the callback This function links the bios keyboard interrupt obtained with IRQs to our keyboard callback. It allows us to setup a buffer and obtain user input. It allows us to retrieve the keycode pressed that we then convert in the KBCallback to ascii char, using an hard coded table.

4.11.1 Detailed Description

function library header related to keyboard and inputs

Author

Théodore MARTIN

Version

0.1

Date

2023-03-22

Definition in file keyboard.h.

4.11.2 Macro Definition Documentation

4.11.2.1 BACKSPACE

```
#define BACKSPACE 0x0E /*Ascii definition */
```

Definition at line 14 of file keyboard.h.

4.11.2.2 ENTER

```
#define ENTER 0x1C /* Ascii definition */
```

Definition at line 13 of file keyboard.h.

4.11.2.3 SCMAX

```
#define SCMAX 57
```

Definition at line 12 of file keyboard.h.

4.11.2.4 UP_ARROW

```
#define UP_ARROW 27
```

Definition at line 16 of file keyboard.h.

4.11.2.5 USER_BUFF_SIZE

```
#define USER_BUFF_SIZE 256 /* WARNING: TYPE USED FOR buffLength should match*/
```

Definition at line 15 of file keyboard.h.

4.11.3 Function Documentation

4.11.3.1 KBInit()

```
void KBInit ( )
```

function linking the bios keyboard interrupt to the callback This function links the bios keyboard interrupt obtained with IRQs to our keyboard callback. It allows us to setup a buffer and obtain user input. It allows us to retrieve the keycode pressed that we then convert in the KBCallback to ascii char, using an hard coded table.

Definition at line 70 of file keyboard.c.

```
00071 {
00072     regInterruptHandler(IRQ1, KBCallback);
00073 }
```

4.12 keyboard.h

Go to the documentation of this file.

```
00001
00009 #ifndef __KEYBOARD__
00010 #define __KEYBOARD__
00011
00012 #define SCMAX 57
00013 #define ENTER 0x1C /* Ascii definition */
00014 #define BACKSPACE 0x0E /*Ascii definition */
00015 #define USER_BUFF_SIZE 256 /* WARNING: TYPE USED FOR buffLength should match*/
00016 #define UP_ARROW 27
00024 void KBInit();
00025
00026 #endif
```

4.13 drivers/ports.c File Reference

function library containing everything related to ports

`#include "ports.h"`
Include dependency graph for ports.c:

4.14 ports.c

Go to the documentation of this file.

```
00001
00009 #include "ports.h"
00010
00011 u8 portByteIn(u16 port)
00012 {
00013     u8 result;
00014     /* Quick tutorial on __ASM__ syntax */
00015     /* based on AT&T, so in those case:  source, destination */
00016     /* "=a" sets the variable result to the value contained in eax */
00017     /* "d" (port) sets the register edx to the value contained in port*/
00018     /* "instruciton src, destination" :  output :  input */
00019     __asm__("in %%dx, %%al":  "=a"(result) :  "d"(port));
00020     return result;
00021 }
00022
00023 void portByteOut(u16 port, u8 data)
00024 {
00025     __asm__("out %%al, %%dx":  :"a"(data), "d"(port));
00026 }
00027
00028 u16 portWordIn(u16 port)
00029 {
00030     u16 result;
00031     __asm__("in %%dx, %%al":  "=a"(result) :  "d"(port));
00032     return result;
00033 }
00034
00035 void portWordOut(u16 port, u16 data)
00036 {
00037     __asm__("out %%al, %%dx":  :"a"(data), "d"(port));
00038 }
```

4.15 drivers/ports.h File Reference

function header library related to ports writing and reading

`#include "../libs/utils.h"`
Include dependency graph for ports.h: This graph shows which files directly or indirectly include this file:

Functions

- u8 portByteIn (u16)

  Read byte value from a port.
- void portByteOut (u16, u8)

  Write a byte to a port.
- u16 portWordIn (u16)

  Read a full word to a port.
- void portWordOut (u16, u16)

  write a word to a port

4.15.1 Detailed Description

function header library related to ports writing and reading

Author

Théodore MARTIN

Version

0.1

Date

2023-03-22

Definition in file ports.h.

4.15.2 Function Documentation

4.15.2.1 portByteIn()

```
u8 portByteIn (
            u16 port )
```

Read byte value from a port.

Parameters

in	*port*	the desired port to read

Returns

u8 the value held in the port

Definition at line 11 of file ports.c.

```
00012 {
00013     u8 result;
00014     /* Quick tutorial on __ASM__ syntax */
00015     /* based on AT&T, so in those case:  source, destination */
00016     /* "=a" sets the variable result to the value contained in eax */
00017     /* "d" (port) sets the register edx to the value contained in port*/
00018     /* "instruciton src, destination" :  output :  input */
00019     __asm__("in %%dx, %%al":  "=a"(result) :  "d"(port));
00020     return result;
00021 }
```

4.15.2.2 portByteOut()

```
void portByteOut (
             u16 port,
             u8 data )
```

Write a byte to a port.

Parameters

in	*port*	the desired port to write to
in	*data*	the desired value that sould be written

Definition at line 23 of file ports.c.

```
00024 {
00025     __asm__("out %%al, %%dx":  :"a"(data), "d"(port));
00026 }
```

4.15.2.3 portWordIn()

```
u16 portWordIn (
             u16 port )
```

Read a full word to a port.

Parameters

in	*port*	the desired port to read

Returns

u16 the value held in the port

Definition at line 28 of file ports.c.

```
00029 {
00030     u16 result;
00031     __asm__("in %%dx, %%al":  "=a"(result) :  "d"(port));
00032     return result;
00033 }
```

4.15.2.4 portWordOut()

```
void portWordOut (
             u16 port,
             u16 data )
```

write a word to a port

Parameters

in	*port*	the desired port to write to
in	*data*	the data that should be written

Definition at line 35 of file ports.c.

```
00036 {
00037     __asm__("out %%al, %%dx":  :"a"(data), "d"(port));
00038 }
```

4.16 ports.h

Go to the documentation of this file.

```
00001
00008 #ifndef __PORTS__
00009 #define __PORTS__
00010 #include "../libs/utils.h"
00017 u8 portByteIn(u16);
00018
00025 void portByteOut(u16, u8);
00026
00033 u16 portWordIn(u16);
00034
00041 void portWordOut(u16, u16);
00042
00043 #endif
```

4.17 drivers/screen.c File Reference

Function library linked to the screen & display.

```
#include "screen.h"
```

Include dependency graph for screen.c:

Functions

- u32 getVGAOffset ()

  Retrieving the current offset of the VGA "cursor".
- void setVGAOffset (u32 offset)

  Set the current VGA offset value to the input.
- void clearScreen ()

  Remove everything on screen and place cursor to top left.
- void putchar (char chr)

  print the char on the screen to the next position (cursor is automatic)
- void removeLastChar ()

deletes the last printed char on screen

- void putcharAtPos (char chr, u8 x, u8 y)

  function printing a char to a certain x/y position on screen
- void printStr (char ∗str)

  prints a string to the next position on screen (cursor is auto)
- void printStrAtPos (char ∗str, u8 x, u8 y)

  function printing a string at a certain x/y position

4.17.1 Detailed Description

Function library linked to the screen & display.

Header file for the screen & display library.

Author

Théodore MARTIN

Version

0.1

Date

22/04/2022

Definition in file screen.c.

4.17.2 Function Documentation

4.17.2.1 clearScreen()

```
void clearScreen ( )
```

Remove everything on screen and place cursor to top left.

Returns

void

Definition at line 67 of file screen.c.

```
00068 {
00069     u16 screenSize = MAX_COLS * MAX_ROWS;
00070     u8 *vga = (u8 *) VIDEO_MEMORY;
00071     for(u16 i = 0; i < screenSize; ++i) {
00072         vga[i * 2] = ' ';
00073         vga[i * 2 + 1] = WHITE_TEXT_BLACK_BACKGROUND;
00074     }
00075     setVGAOffset(getOffset(0,0));
00076 }
```

4.17.2.2 getVGAOffset()

```
u32 getVGAOffset ( )
```

Retrieving the current offset of the VGA "cursor".

Returns

u32, the requested offset

Definition at line 46 of file screen.c.

```
00046                     {
00047     portByteOut(0x3D4, 14); /* asking vga control register */
00048     u16 position = portByteIn(0x3D5); /* retrieving the position high bytes*/
00049     position = position « 8; /* shifting to the top bytes */
00050     portByteOut(0x3D4, 15); /*requesting low bytes*/
00051     position += portByteIn(0x3D5); /*adding the low bytes*/
00052     /* 1 vga cell is made of the char & its data (color & background) */
00053     /* We have to multiply the position by 2 to know where to write */
00054     return (u32) (position * 2);
00055 }
```

4.17.2.3 printStr()

```
void printStr (
             char * str )
```

prints a string to the next position on screen (cursor is auto)

Parameters

str	the string that should be printed

Definition at line 125 of file screen.c.

```
00126 {
00127     u32 size = str_len(str);
00128     for (u32 i = 0; i < size; ++i) {
00129         putchar(str[i]);
00130     }
00131 }
```

4.17.2.4 printStrAtPos()

```
void printStrAtPos (
             char * str,
             u8 x,
             u8 y )
```

function printing a string at a certain x/y position

Parameters

str	the string that should be printed
x	the x coordinates where the string should be (>0)
y	the y coordinates where the string should be (>0)

Definition at line 133 of file screen.c.

```
00134 {
00135     u32 size = str_len(str);
00136     for (u32 i = 0; i < size; ++i) {
00137         putcharAtPos(str[i], (x + i) % MAX_COLS, y + (x + i) / MAX_COLS);
00138     }
00139 }
```

4.17.2.5 putchar()

```
void putchar (
            char chr )
```

print the char on the screen to the next position (cursor is automatic)

Parameters

chr	char that should be printed

Returns

void

Definition at line 78 of file screen.c.

```
00079 {
00080     u32 offset = getVGAOffset();
00081     if (chr == '\n') {
00082         u32 missingColsToNewLine = (MAX_COLS - getCols(offset)) * 2;
00083         offset += missingColsToNewLine;
00084     } else if (chr == '\t') {
00085         offset += 8;
00086     } else {
00087         u8 *vga = (u8 *) VIDEO_MEMORY;
00088         vga[offset] = chr;
00089         vga[offset + 1] = WHITE_TEXT_BLACK_BACKGROUND;
00090         offset += 2;
00091     }
00092     /* scrolling the screen */
00093     if (offset >= MAX_COLS * MAX_ROWS * 2) {
00094         for (u8 i = 1; i < MAX_ROWS; ++i) {
00095             mem_cpy((i8 *) (getOffset(i, 0) + VIDEO_MEMORY),
00096                     (i8 *) (getOffset(i-1, 0) + VIDEO_MEMORY),
00097                     MAX_COLS * 2);
00098         }
00099         char *lastLine = (i8 *) (getOffset(MAX_ROWS - 1, 0) + VIDEO_MEMORY);
00100         for (u8 i = 0; i < MAX_COLS * 2; ++i)
00101             lastLine[i] = 0;
00102         offset -= 2 * MAX_COLS;
00103     }
00104     setVGAOffset(offset);
00105 }
```

4.17.2.6 putcharAtPos()

```
void putcharAtPos (
            char chr,
            u8 x,
            u8 y )
```

function printing a char to a certain x/y position on screen

Parameters

chr	the char that should be printed
x	the x coordinates (>0)
y	the y coordinates (>0)

Definition at line 116 of file screen.c.

```
00117 {
00118     i32 offset = getOffset(y, x);
00119     setVGAOffset(offset);
00120     u8 *vga = (u8 *) VIDEO_MEMORY;
00121     vga[offset] = chr;
00122     vga[offset + 1] = WHITE_TEXT_BLACK_BACKGROUND;
00123 }
```

4.17.2.7 removeLastChar()

```
void removeLastChar ( )
```

deletes the last printed char on screen

Returns

void

Definition at line 107 of file screen.c.

```
00108 {
00109     u32 offset = getVGAOffset() - 2;
00110     u8 *vga = (u8 *) VIDEO_MEMORY;
00111     vga[offset] = ' ';
00112     vga[offset + 1] = WHITE_TEXT_BLACK_BACKGROUND;
00113     setVGAOffset(offset);
00114 }
```

4.17.2.8 setVGAOffset()

```
void setVGAOffset (
            u32 offset )
```

Set the current VGA offset value to the input.

Parameters

in	*offset*	The desired offset new value

Returns

void

Definition at line 57 of file screen.c.

```
00058 {
00059     /* same as prior, but writing instead of reading */
00060     offset /= 2;
00061     portByteOut(0x3D4, 14);
00062     portByteOut(0x3D5, (u8)(offset » 8));
00063     portByteOut(0x3D4, 15);
00064     portByteOut(0x3D5, (u8)(offset & 0xFF));
00065 }
```

4.18 screen.c

Go to the documentation of this file.

```
00001
00008 #include "screen.h"
00009
00010 /* --- private functions --- */
00011
00019 static u32 getOffset(u8 row, u8 column)
00020 {
00021     return 2 * (row * MAX_COLS + column);
00022 }
00023
00030 static u8 getCols(u32 offset)
00031 {
00032     return (offset / 2) % MAX_COLS;
00033 }
00034
00041 static u8 getRows(u32 offset)
00042 {
00043     return (offset / 2) / MAX_COLS;
00044 }
00045
00046 u32 getVGAOffset() {
00047     portByteOut(0x3D4, 14); /* asking vga control register */
00048     u16 position = portByteIn(0x3D5); /* retrieving the position high bytes*/
00049     position = position « 8; /* shifting to the top bytes */
00050     portByteOut(0x3D4, 15); /*requesting low bytes*/
00051     position += portByteIn(0x3D5); /*adding the low bytes*/
00052     /* 1 vga cell is made of the char & its data (color & background) */
00053     /* We have to multiply the position by 2 to know where to write */
00054     return (u32) (position * 2);
00055 }
00056
00057 void setVGAOffset(u32 offset)
00058 {
00059     /* same as prior, but writing instead of reading */
00060     offset /= 2;
00061     portByteOut(0x3D4, 14);
00062     portByteOut(0x3D5, (u8)(offset » 8));
00063     portByteOut(0x3D4, 15);
00064     portByteOut(0x3D5, (u8)(offset & 0xFF));
00065 }
00066
00067 void clearScreen()
00068 {
00069     u16 screenSize = MAX_COLS * MAX_ROWS;
00070     u8 *vga = (u8 *) VIDEO_MEMORY;
00071     for(u16 i = 0; i < screenSize; ++i) {
00072         vga[i * 2] = ' ';
00073         vga[i * 2 + 1] = WHITE_TEXT_BLACK_BACKGROUND;
00074     }
00075     setVGAOffset(getOffset(0,0));
00076 }
00077
00078 void putchar(char chr)
00079 {
00080     u32 offset = getVGAOffset();
00081     if (chr == '\n') {
00082         u32 missingColsToNewLine = (MAX_COLS - getCols(offset)) * 2;
00083         offset += missingColsToNewLine;
00084     } else if (chr == '\t') {
00085         offset += 8;
00086     } else {
00087         u8 *vga = (u8 *) VIDEO_MEMORY;
00088         vga[offset] = chr;
00089         vga[offset + 1] = WHITE_TEXT_BLACK_BACKGROUND;
00090         offset += 2;
00091     }
00092     /* scrolling the screen */
00093     if (offset >= MAX_COLS * MAX_ROWS * 2) {
00094         for (u8 i = 1; i < MAX_ROWS; ++i) {
```

```
00095             mem_cpy((i8 *) (getOffset(i, 0) + VIDEO_MEMORY),
00096                     (i8 *) (getOffset(i-1, 0) + VIDEO_MEMORY),
00097                     MAX_COLS * 2);
00098         }
00099         char *lastLine = (i8 *) (getOffset(MAX_ROWS - 1, 0) + VIDEO_MEMORY);
00100         for (u8 i = 0; i < MAX_COLS * 2; ++i)
00101             lastLine[i] = 0;
00102         offset -= 2 * MAX_COLS;
00103     }
00104     setVGAOffset(offset);
00105 }
00106
00107 void removeLastChar()
00108 {
00109     u32 offset = getVGAOffset() - 2;
00110     u8 *vga = (u8 *) VIDEO_MEMORY;
00111     vga[offset] = ' ';
00112     vga[offset + 1] = WHITE_TEXT_BLACK_BACKGROUND;
00113     setVGAOffset(offset);
00114 }
00115
00116 void putcharAtPos(char chr, u8 x, u8 y)
00117 {
00118     i32 offset = getOffset(y, x);
00119     setVGAOffset(offset);
00120     u8 *vga = (u8 *) VIDEO_MEMORY;
00121     vga[offset] = chr;
00122     vga[offset + 1] = WHITE_TEXT_BLACK_BACKGROUND;
00123 }
00124
00125 void printStr(char *str)
00126 {
00127     u32 size = str_len(str);
00128     for (u32 i = 0; i < size; ++i) {
00129         putchar(str[i]);
00130     }
00131 }
00132
00133 void printStrAtPos(char *str, u8 x, u8 y)
00134 {
00135     u32 size = str_len(str);
00136     for (u32 i = 0; i < size; ++i) {
00137         putcharAtPos(str[i], (x + i) % MAX_COLS, y + (x + i) / MAX_COLS);
00138     }
00139 }
```

4.19 screen.h

```
00001
00008 #ifndef __SCREEN__
00009 #define __SCREEN__
00010 #include "./ports.h"
00011 #include "../libs/memory.h"
00012 #include "../libs/strings.h"
00013
00014 /* Constants definitions */
00015 #define VIDEO_MEMORY 0xb8000
00016 #define WHITE_TEXT_BLACK_BACKGROUND 0x0F
00017 #define MAX_ROWS 25
00018 #define MAX_COLS 80
00019
00020 /* Functions */
00021
00027 u32 getVGAOffset();
00028
00036 void setVGAOffset(u32);
00037
00043 void clearScreen();
00044
00052 void putchar(char);
00053
00059 void removeLastChar();
00060
00068 void putcharAtPos(char, u8, u8);
00069
00075 void printStr(char *);
00076
00084 void printStrAtPos(char *, u8, u8);
00085
00086 #endif
```

4.20 kernel/idt.c File Reference

function linked to the Interrupt descriptor table

`#include "idt.h"`
Include dependency graph for idt.c:

Functions

- void setIDTGate (u8 n, u32 handler)

  initialize the IDT Register (or gate)
- void setIDT ()

  Loads the lidt command, to load the IDT https://c9x.me/x86/html/file_module_x86_id_156.html.

Variables

- IDTGate IDT [IDTNB]
- IDTRegister IDTReg

4.20.1 Detailed Description

function linked to the Interrupt descriptor table

Author

Théodore MARTIN

Version

0.1

Date

2023-03-23

Definition in file idt.c.

4.20.2 Function Documentation

4.20.2.1 setIDT()

```
void setIDT ( )
```

Loads the lidt command, to load the IDT https://c9x.me/x86/html/file_module_x86_id_156.↩html.

Definition at line 26 of file idt.c.

```
00027 {
00028     /*  retrieving the adresses of the idt
00029 It will then be stored inside the IDTR (r for register) */
00030     IDTReg.base = (u32) &IDT;
00031     IDTReg.limit = IDTNB * sizeof(IDTGate) - 1;
00032     /* loading the idt */
00033     __asm__ __volatile__("lidt (%0)" :  :  "r"(&IDTReg));
00034     /* remember:  never load &IDT, always work with &IDTReg*/
00035 }
```

4.20.2.2 setIDTGate()

```
void setIDTGate (
            u8 n,
            u32 handler )
```

initialize the IDT Register (or gate)

Parameters

in	*n*	the gate number
in	*handler*	the function handling the interrupt

Definition at line 14 of file idt.c.

```
00015 {
00016     /*  filling the struct with the values
00017 explained here https://wiki.osdev.org/Interrupt_Descriptor_Table
00018 ("Structure on x86-64") */
00019     IDT[n].lOffset = low16(handler);
00020     IDT[n].sel = KERNCS;
00021     IDT[n].zero = 0;
00022     IDT[n].flags = 0x8E; /* p=1, dpl=0b00, type=0b1110 => type_attributes=0b1000_1110=0x8E */
00023     IDT[n].hOffset = high16(handler);
00024 }
```

4.20.3 Variable Documentation

4.20.3.1 IDT

```
IDTGate IDT[IDTNB]
```

Definition at line 11 of file idt.c.

4.20.3.2 IDTReg

```
IDTRegister IDTReg
```

Definition at line 12 of file idt.c.

4.21 idt.c

Go to the documentation of this file.

```
00001
00009 #include "idt.h"
00010
00011 IDTGate IDT[IDTNB];
00012 IDTRegister IDTReg;
00013
00014 void setIDTGate(u8 n, u32 handler)
00015 {
00016     /*  filling the struct with the values
00017 explained here https://wiki.osdev.org/Interrupt_Descriptor_Table
00018 ("Structure on x86-64") */
00019     IDT[n].lOffset = low16(handler);
00020     IDT[n].sel = KERNCS;
00021     IDT[n].zero = 0;
00022     IDT[n].flags = 0x8E; /* p=1, dpl=0b00, type=0b1110 => type_attributes=0b1000_1110=0x8E */
00023     IDT[n].hOffset = high16(handler);
00024 }
00025
00026 void setIDT()
00027 {
00028     /*  retrieving the adresses of the idt
00029 It will then be stored inside the IDTR (r for register) */
00030     IDTReg.base = (u32) &IDT;
00031     IDTReg.limit = IDTNB * sizeof(IDTGate) - 1;
00032     /* loading the idt */
00033     __asm__ __volatile__("lidt (%0)" :  :  "r"(&IDTReg));
00034     /* remember:  never load &IDT, always work with &IDTReg*/
00035 }
```

4.22 kernel/idt.h File Reference

library containing everything related to the Interrupt Descriptor Table

```
#include "../libs/utils.h"
```

Include dependency graph for idt.h: This graph shows which files directly or indirectly include this file:

Classes

- struct __attribute__

Macros

- #define KERNCS 0x08 /* kernel selector */
- #define IDTNB 256 /* number of interrupts - we need 256 else the kernel may panic */

Functions

- void setIDTGate (u8, u32)

  initialize the IDT Register (or gate)
- void setIDT ()

  Loads the lidt command, to load the IDT https://c9x.me/x86/html/file_module_x86_id_156.↩ html.

Variables

- IDTGate IDT [IDTNB]
- IDTRegister IDTReg

4.22.1 Detailed Description

library containing everything related to the Interrupt Descriptor Table

Author

Théodore MARTIN

Version

0.1

Date

2023-03-22

Definition in file idt.h.

4.22.2 Macro Definition Documentation

4.22.2.1 IDTNB

```
#define IDTNB 256 /* number of interrupts - we need 256 else the kernel may panic */
```

Definition at line 18 of file idt.h.

4.22.2.2 KERNCS

```
#define KERNCS 0x08 /* kernel selector */
```

Definition at line 17 of file idt.h.

4.22.3 Function Documentation

4.22.3.1 setIDT()

```
void setIDT ( )
```

Loads the lidt command, to load the IDT https://c9x.me/x86/html/file_module_x86_id_156.↩html.

Definition at line 26 of file idt.c.

```
00027 {
00028     /*  retrieving the adresses of the idt
00029 It will then be stored inside the IDTR (r for register) */
00030     IDTReg.base = (u32) &IDT;
00031     IDTReg.limit = IDTNB * sizeof(IDTGate) - 1;
00032     /* loading the idt */
00033     __asm__ __volatile__("lidt (%0)" :  :  "r"(&IDTReg));
00034     /* remember:  never load &IDT, always work with &IDTReg*/
00035 }
```

4.22.3.2 setIDTGate()

```
void setIDTGate (
            u8 n,
            u32 handler )
```

initialize the IDT Register (or gate)

Parameters

in	*n*	the gate number
in	*handler*	the function handling the interrupt

Definition at line 14 of file idt.c.

```
00015 {
00016     /*  filling the struct with the values
00017 explained here https://wiki.osdev.org/Interrupt_Descriptor_Table
00018 ("Structure on x86-64") */
00019     IDT[n].lOffset = low16(handler);
00020     IDT[n].sel = KERNCS;
00021     IDT[n].zero = 0;
00022     IDT[n].flags = 0x8E; /* p=1, dpl=0b00, type=0b1110 => type_attributes=0b1000_1110=0x8E */
00023     IDT[n].hOffset = high16(handler);
00024 }
```

4.22.4 Variable Documentation

4.22.4.1 IDT

```
IDTGate IDT[IDTNB]  [extern]
```

Definition at line 11 of file idt.c.

4.22.4.2 IDTReg

```
IDTRegister IDTReg  [extern]
```

Definition at line 12 of file idt.c.

4.23 idt.h

Go to the documentation of this file.

```
00001
00008 #ifndef __IDT__
00009 #define __IDT__
00010 #include "../libs/utils.h"
00011 /* Interrupt description table - storing interrupts
00012 * Table containing telling the CPU where the ISR (interrupt service routines)
00013 * are stored, one per interrupt.
00014 * https://wiki.osdev.org/IDT
00015 */
00016
00017 #define KERNCS 0x08 /* kernel selector */
00018 #define IDTNB 256 /* number of interrupts - we need 256 else the kernel may panic */
00019
00020 typedef struct {
00021     u16 lOffset;
00022     u16 sel; /*always KERNCS*/
00023     u8 zero;
00024     u8 flags;
00025     u16 hOffset;
00026 } __attribute__((packed)) IDTGate; /*using packed to get a fixed size*/
00027
00028 typedef struct {
00029     u16 limit;
00030     u32 base;
00031 } __attribute__((packed)) IDTRegister;
00032
00033 /* Global variables */
00034 extern IDTGate IDT[IDTNB];
00035 extern IDTRegister IDTReg;
00036
00044 void setIDTGate(u8, u32);
00045
00050 void setIDT();
00051
00052 #endif
```

4.24 kernel/isr.c File Reference

functions required to setup & use the interrupt service routines

```
#include "isr.h"
```

Include dependency graph for isr.c:

Functions

- void ISRInstall ()

  install all ISR at once with default stuff By default, if an interrupt is detected, the system will print the interrupt number, and the exception message
- void ISRHandler (reg r)

  called by asm when an interrupt is detected - mostly critical cpu things
- void regInterruptHandler (u8 n, ISR handler)

  sets the handler of the given ISR
- void IRQHandler (reg r)

  called by the asm code - runs the defined function reacting to an IRQ

Variables

- ISR interruptHandlers [IDTNB]
- char * exceptionMessages [ ]

4.24.1 Detailed Description

functions required to setup & use the interrupt service routines

Author

Théodore MARTIN

Version

0.1

Date

2023-03-22

Definition in file isr.c.

4.24.2 Function Documentation

4.24.2.1 IRQHandler()

```
void IRQHandler (
            reg r )
```

called by the asm code - runs the defined function reacting to an IRQ

Parameters

in	*r*	the register that we react to

Returns

void

Definition at line 129 of file isr.c.

```
00130 {
00131     /* After every interrupt we need to send an EOI to the PICs
00132 * or they will not send another interrupt again */
00133     if (r.intNo >= 40) portByteOut(0xA0, 0x20); /* slave */
00134     portByteOut(0x20, 0x20); /* master */
00135
```

```
00136     /* Handle the interrupt in a more modular way */
00137     if (interruptHandlers[r.intNo] != 0) {
00138         ISR handler = interruptHandlers[r.intNo];
00139         handler(r);
00140     }
00141 }
```

4.24.2.2 ISRHandler()

```
void ISRHandler (
             reg r )
```

called by asm when an interrupt is detected - mostly critical cpu things

Parameters

in	*r*	the register we receive the interrupt from

Returns

void

Definition at line 113 of file isr.c.

```
00114 {
00115     printStr("Received interrupt:  ");
00116     char s[3];
00117     iToA(r.intNo, s);
00118     printStr(s);
00119     printStr("\n");
00120     printStr(exceptionMessages[r.intNo]);
00121     printStr("\n");
00122 }
```

4.24.2.3 ISRInstall()

```
void ISRInstall ( )
```

install all ISR at once with default stuff By default, if an interrupt is detected, the system will print the interrupt number, and the exception message

Returns

void

Definition at line 12 of file isr.c.

```
00013 {
00014     /* init IDT */
00015     setIDTGate(0, (u32)isr0);
00016     setIDTGate(1, (u32)isr1);
00017     setIDTGate(2, (u32)isr2);
00018     setIDTGate(3, (u32)isr3);
00019     setIDTGate(4, (u32)isr4);
00020     setIDTGate(5, (u32)isr5);
00021     setIDTGate(6, (u32)isr6);
00022     setIDTGate(7, (u32)isr7);
00023     setIDTGate(8, (u32)isr8);
```

```
00024     setIDTGate(9, (u32)isr9);
00025     setIDTGate(10, (u32)isr10);
00026     setIDTGate(11, (u32)isr11);
00027     setIDTGate(12, (u32)isr12);
00028     setIDTGate(13, (u32)isr13);
00029     setIDTGate(14, (u32)isr14);
00030     setIDTGate(15, (u32)isr15);
00031     setIDTGate(16, (u32)isr16);
00032     setIDTGate(17, (u32)isr17);
00033     setIDTGate(18, (u32)isr18);
00034     setIDTGate(19, (u32)isr19);
00035     setIDTGate(20, (u32)isr20);
00036     setIDTGate(21, (u32)isr21);
00037     setIDTGate(22, (u32)isr22);
00038     setIDTGate(23, (u32)isr23);
00039     setIDTGate(24, (u32)isr24);
00040     setIDTGate(25, (u32)isr25);
00041     setIDTGate(26, (u32)isr26);
00042     setIDTGate(27, (u32)isr27);
00043     setIDTGate(28, (u32)isr28);
00044     setIDTGate(29, (u32)isr29);
00045     setIDTGate(30, (u32)isr30);
00046     setIDTGate(31, (u32)isr31);
00047     /* Remapping the PIC */
00048     portByteOut(0x20, 0x11);
00049     portByteOut(0xA0, 0x11);
00050     portByteOut(0x21, 0x20);
00051     portByteOut(0xA1, 0x28);
00052     portByteOut(0x21, 0x04);
00053     portByteOut(0xA1, 0x02);
00054     portByteOut(0x21, 0x01);
00055     portByteOut(0xA1, 0x01);
00056     portByteOut(0x21, 0x0);
00057     portByteOut(0xA1, 0x0);
00058     /* Installing IRQs */
00059     setIDTGate(32, (u32)irq0);
00060     setIDTGate(33, (u32)irq1);
00061     setIDTGate(34, (u32)irq2);
00062     setIDTGate(35, (u32)irq3);
00063     setIDTGate(36, (u32)irq4);
00064     setIDTGate(37, (u32)irq5);
00065     setIDTGate(38, (u32)irq6);
00066     setIDTGate(39, (u32)irq7);
00067     setIDTGate(40, (u32)irq8);
00068     setIDTGate(41, (u32)irq9);
00069     setIDTGate(42, (u32)irq10);
00070     setIDTGate(43, (u32)irq11);
00071     setIDTGate(44, (u32)irq12);
00072     setIDTGate(45, (u32)irq13);
00073     setIDTGate(46, (u32)irq14);
00074     setIDTGate(47, (u32)irq15);
00075     setIDT();
00076 }
```

4.24.2.4 regInterruptHandler()

```
void regInterruptHandler (
            u8 n,
            ISR handler )
```

sets the handler of the given ISR

Parameters

in	*n*	the register number
in	*handler*	the ISR that will be linked

Returns

void

Definition at line 124 of file isr.c.

```
00125 {
00126     interruptHandlers[n] = handler;
00127 }
```

4.24.3 Variable Documentation

4.24.3.1 exceptionMessages

```
char* exceptionMessages[]
```

Definition at line 78 of file isr.c.

4.24.3.2 interruptHandlers

```
ISR interruptHandlers[IDTNB]
```

Definition at line 10 of file isr.c.

4.25 isr.c

Go to the documentation of this file.

```
00001
00008 #include "isr.h"
00009
00010 ISR interruptHandlers[IDTNB];
00011
00012 void ISRInstall()
00013 {
00014     /* init IDT */
00015     setIDTGate(0, (u32)isr0);
00016     setIDTGate(1, (u32)isr1);
00017     setIDTGate(2, (u32)isr2);
00018     setIDTGate(3, (u32)isr3);
00019     setIDTGate(4, (u32)isr4);
00020     setIDTGate(5, (u32)isr5);
00021     setIDTGate(6, (u32)isr6);
00022     setIDTGate(7, (u32)isr7);
00023     setIDTGate(8, (u32)isr8);
00024     setIDTGate(9, (u32)isr9);
00025     setIDTGate(10, (u32)isr10);
00026     setIDTGate(11, (u32)isr11);
00027     setIDTGate(12, (u32)isr12);
00028     setIDTGate(13, (u32)isr13);
00029     setIDTGate(14, (u32)isr14);
00030     setIDTGate(15, (u32)isr15);
00031     setIDTGate(16, (u32)isr16);
00032     setIDTGate(17, (u32)isr17);
00033     setIDTGate(18, (u32)isr18);
00034     setIDTGate(19, (u32)isr19);
00035     setIDTGate(20, (u32)isr20);
00036     setIDTGate(21, (u32)isr21);
00037     setIDTGate(22, (u32)isr22);
00038     setIDTGate(23, (u32)isr23);
00039     setIDTGate(24, (u32)isr24);
00040     setIDTGate(25, (u32)isr25);
00041     setIDTGate(26, (u32)isr26);
00042     setIDTGate(27, (u32)isr27);
00043     setIDTGate(28, (u32)isr28);
00044     setIDTGate(29, (u32)isr29);
```

```
00045     setIDTGate(30, (u32)isr30);
00046     setIDTGate(31, (u32)isr31);
00047     /* Remapping the PIC */
00048     portByteOut(0x20, 0x11);
00049     portByteOut(0xA0, 0x11);
00050     portByteOut(0x21, 0x20);
00051     portByteOut(0xA1, 0x28);
00052     portByteOut(0x21, 0x04);
00053     portByteOut(0xA1, 0x02);
00054     portByteOut(0x21, 0x01);
00055     portByteOut(0xA1, 0x01);
00056     portByteOut(0x21, 0x0);
00057     portByteOut(0xA1, 0x0);
00058     /* Installing IRQs */
00059     setIDTGate(32, (u32)irq0);
00060     setIDTGate(33, (u32)irq1);
00061     setIDTGate(34, (u32)irq2);
00062     setIDTGate(35, (u32)irq3);
00063     setIDTGate(36, (u32)irq4);
00064     setIDTGate(37, (u32)irq5);
00065     setIDTGate(38, (u32)irq6);
00066     setIDTGate(39, (u32)irq7);
00067     setIDTGate(40, (u32)irq8);
00068     setIDTGate(41, (u32)irq9);
00069     setIDTGate(42, (u32)irq10);
00070     setIDTGate(43, (u32)irq11);
00071     setIDTGate(44, (u32)irq12);
00072     setIDTGate(45, (u32)irq13);
00073     setIDTGate(46, (u32)irq14);
00074     setIDTGate(47, (u32)irq15);
00075     setIDT();
00076 }
00077
00078 char *exceptionMessages[] = {
00079     "Division By Zero",
00080     "Debug",
00081     "Non Maskable Interrupt",
00082     "Breakpoint",
00083     "Into Detected Overflow",
00084     "Out of Bounds",
00085     "Invalid Opcode",
00086     "No Coprocessor",
00087     "Double Fault",
00088     "Coprocessor Segment Overrun",
00089     "Bad TSS",
00090     "Segment Not Present",
00091     "Stack Fault",
00092     "General Protection Fault",
00093     "Page Fault",
00094     "Unknown Interrupt",
00095     "Coprocessor Fault",
00096     "Alignment Check",
00097     "Machine Check",
00098     "Reserved",
00099     "Reserved",
00100     "Reserved",
00101     "Reserved",
00102     "Reserved",
00103     "Reserved",
00104     "Reserved",
00105     "Reserved",
00106     "Reserved",
00107     "Reserved",
00108     "Reserved",
00109     "Reserved",
00110     "Reserved"
00111 };
00112
00113 void ISRHandler(reg r)
00114 {
00115     printStr("Received interrupt:  ");
00116     char s[3];
00117     iToA(r.intNo, s);
00118     printStr(s);
00119     printStr("\n");
00120     printStr(exceptionMessages[r.intNo]);
00121     printStr("\n");
00122 }
00123
00124 void regInterruptHandler(u8 n, ISR handler)
00125 {
00126     interruptHandlers[n] = handler;
00127 }
00128
00129 void IRQHandler(reg r)
00130 {
00131     /* After every interrupt we need to send an EOI to the PICs
```

```
00132 * or they will not send another interrupt again */
00133     if (r.intNo >= 40) portByteOut(0xA0, 0x20); /* slave */
00134     portByteOut(0x20, 0x20); /* master */
00135 
00136     /* Handle the interrupt in a more modular way */
00137     if (interruptHandlers[r.intNo] != 0) {
00138         ISR handler = interruptHandlers[r.intNo];
00139         handler(r);
00140     }
00141 }
```

4.26 kernel/isr.h File Reference

headers containing everything related to the interrupt service routines

```
#include "idt.h"
#include "../libs/utils.h"
#include "../drivers/screen.h"
```

Include dependency graph for isr.h: This graph shows which files directly or indirectly include this file:

Classes

- struct reg

  representation of the asm register pushed in interrupts.asm more info inside kernel/interrupts.asm

Macros

- #define IRQ0 32
- #define IRQ1 33
- #define IRQ2 34
- #define IRQ3 35
- #define IRQ4 36
- #define IRQ5 37
- #define IRQ6 38
- #define IRQ7 39
- #define IRQ8 40
- #define IRQ9 41
- #define IRQ10 42
- #define IRQ11 43
- #define IRQ12 44
- #define IRQ13 45
- #define IRQ14 46
- #define IRQ15 47

Typedefs

- typedef void(∗ ISR) (reg)

Functions

- void **isr0** ()
- void **isr1** ()
- void **isr2** ()
- void **isr3** ()
- void **isr4** ()
- void **isr5** ()
- void **isr6** ()
- void **isr7** ()
- void **isr8** ()
- void **isr9** ()
- void **isr10** ()
- void **isr11** ()
- void **isr12** ()
- void **isr13** ()
- void **isr14** ()
- void **isr15** ()
- void **isr16** ()
- void **isr17** ()
- void **isr18** ()
- void **isr19** ()
- void **isr20** ()
- void **isr21** ()
- void **isr22** ()
- void **isr23** ()
- void **isr24** ()
- void **isr25** ()
- void **isr26** ()
- void **isr27** ()
- void **isr28** ()
- void **isr29** ()
- void **isr30** ()
- void **isr31** ()
- void **irq0** ()
- void **irq1** ()
- void **irq2** ()
- void **irq3** ()
- void **irq4** ()
- void **irq5** ()
- void **irq6** ()
- void **irq7** ()
- void **irq8** ()
- void **irq9** ()
- void **irq10** ()
- void **irq11** ()
- void **irq12** ()
- void **irq13** ()
- void **irq14** ()
- void **irq15** ()
- void ISRInstall ()

  install all ISR at once with default stuff By default, if an interrupt is detected, the system will print the interrupt number, and the exception message
- void ISRHandler (reg)

  called by asm when an interrupt is detected - mostly critical cpu things

- void regInterruptHandler (u8, ISR)
  sets the handler of the given ISR
- void IRQHandler (reg r)
  called by the asm code - runs the defined function reacting to an IRQ

4.26.1 Detailed Description

headers containing everything related to the interrupt service routines

Author

Théodore MARTIN

Version

0.1

Date

2023-03-22

Definition in file isr.h.

4.26.2 Macro Definition Documentation

4.26.2.1 IRQ0

```
#define IRQ0 32
```

Definition at line 28 of file isr.h.

4.26.2.2 IRQ1

```
#define IRQ1 33
```

Definition at line 29 of file isr.h.

4.26.2.3 IRQ10

```
#define IRQ10 42
```

Definition at line 38 of file isr.h.

4.26.2.4 IRQ11

```
#define IRQ11 43
```

Definition at line 39 of file isr.h.

4.26.2.5 IRQ12

```
#define IRQ12 44
```

Definition at line 40 of file isr.h.

4.26.2.6 IRQ13

```
#define IRQ13 45
```

Definition at line 41 of file isr.h.

4.26.2.7 IRQ14

```
#define IRQ14 46
```

Definition at line 42 of file isr.h.

4.26.2.8 IRQ15

```
#define IRQ15 47
```

Definition at line 43 of file isr.h.

4.26.2.9 IRQ2

```
#define IRQ2 34
```

Definition at line 30 of file isr.h.

4.26.2.10 IRQ3

```
#define IRQ3 35
```

Definition at line 31 of file isr.h.

4.26.2.11 IRQ4

```
#define IRQ4 36
```

Definition at line 32 of file isr.h.

4.26.2.12 IRQ5

```
#define IRQ5 37
```

Definition at line 33 of file isr.h.

4.26.2.13 IRQ6

```
#define IRQ6 38
```

Definition at line 34 of file isr.h.

4.26.2.14 IRQ7

```
#define IRQ7 39
```

Definition at line 35 of file isr.h.

4.26.2.15 IRQ8

```
#define IRQ8 40
```

Definition at line 36 of file isr.h.

4.26.2.16 IRQ9

```
#define IRQ9 41
```

Definition at line 37 of file isr.h.

4.26.3 Typedef Documentation

4.26.3.1 ISR

```
typedef void(* ISR) (reg)
```

Definition at line 25 of file isr.h.

4.26.4 Function Documentation

4.26.4.1 IRQHandler()

```
void IRQHandler (
             reg r )
```

called by the asm code - runs the defined function reacting to an IRQ

Parameters

in	*r*	the register that we react to

Returns

void

Definition at line 129 of file isr.c.

```
00130 {
00131     /* After every interrupt we need to send an EOI to the PICs
00132 * or they will not send another interrupt again */
00133     if (r.intNo >= 40) portByteOut(0xA0, 0x20); /* slave */
00134     portByteOut(0x20, 0x20); /* master */
00135
00136     /* Handle the interrupt in a more modular way */
00137     if (interruptHandlers[r.intNo] != 0) {
00138         ISR handler = interruptHandlers[r.intNo];
00139         handler(r);
00140     }
00141 }
```

4.26.4.2 ISRHandler()

```
void ISRHandler (
            reg r )
```

called by asm when an interrupt is detected - mostly critical cpu things

Parameters

in	*r*	the register we receive the interrupt from

Returns

void

Definition at line 113 of file isr.c.

```
00114 {
00115     printStr("Received interrupt:  ");
00116     char s[3];
00117     iToA(r.intNo, s);
00118     printStr(s);
00119     printStr("\n");
00120     printStr(exceptionMessages[r.intNo]);
00121     printStr("\n");
00122 }
```

4.26.4.3 ISRInstall()

```
void ISRInstall ( )
```

install all ISR at once with default stuff By default, if an interrupt is detected, the system will print the interrupt number, and the exception message

Returns

void

Definition at line 12 of file isr.c.

```
00013 {
00014     /* init IDT */
00015     setIDTGate(0, (u32)isr0);
00016     setIDTGate(1, (u32)isr1);
00017     setIDTGate(2, (u32)isr2);
00018     setIDTGate(3, (u32)isr3);
00019     setIDTGate(4, (u32)isr4);
00020     setIDTGate(5, (u32)isr5);
00021     setIDTGate(6, (u32)isr6);
00022     setIDTGate(7, (u32)isr7);
00023     setIDTGate(8, (u32)isr8);
00024     setIDTGate(9, (u32)isr9);
00025     setIDTGate(10, (u32)isr10);
00026     setIDTGate(11, (u32)isr11);
00027     setIDTGate(12, (u32)isr12);
00028     setIDTGate(13, (u32)isr13);
00029     setIDTGate(14, (u32)isr14);
00030     setIDTGate(15, (u32)isr15);
00031     setIDTGate(16, (u32)isr16);
00032     setIDTGate(17, (u32)isr17);
00033     setIDTGate(18, (u32)isr18);
00034     setIDTGate(19, (u32)isr19);
00035     setIDTGate(20, (u32)isr20);
00036     setIDTGate(21, (u32)isr21);
00037     setIDTGate(22, (u32)isr22);
```

```
00038     setIDTGate(23, (u32)isr23);
00039     setIDTGate(24, (u32)isr24);
00040     setIDTGate(25, (u32)isr25);
00041     setIDTGate(26, (u32)isr26);
00042     setIDTGate(27, (u32)isr27);
00043     setIDTGate(28, (u32)isr28);
00044     setIDTGate(29, (u32)isr29);
00045     setIDTGate(30, (u32)isr30);
00046     setIDTGate(31, (u32)isr31);
00047     /* Remapping the PIC */
00048     portByteOut(0x20, 0x11);
00049     portByteOut(0xA0, 0x11);
00050     portByteOut(0x21, 0x20);
00051     portByteOut(0xA1, 0x28);
00052     portByteOut(0x21, 0x04);
00053     portByteOut(0xA1, 0x02);
00054     portByteOut(0x21, 0x01);
00055     portByteOut(0xA1, 0x01);
00056     portByteOut(0x21, 0x0);
00057     portByteOut(0xA1, 0x0);
00058     /* Installing IRQs */
00059     setIDTGate(32, (u32)irq0);
00060     setIDTGate(33, (u32)irq1);
00061     setIDTGate(34, (u32)irq2);
00062     setIDTGate(35, (u32)irq3);
00063     setIDTGate(36, (u32)irq4);
00064     setIDTGate(37, (u32)irq5);
00065     setIDTGate(38, (u32)irq6);
00066     setIDTGate(39, (u32)irq7);
00067     setIDTGate(40, (u32)irq8);
00068     setIDTGate(41, (u32)irq9);
00069     setIDTGate(42, (u32)irq10);
00070     setIDTGate(43, (u32)irq11);
00071     setIDTGate(44, (u32)irq12);
00072     setIDTGate(45, (u32)irq13);
00073     setIDTGate(46, (u32)irq14);
00074     setIDTGate(47, (u32)irq15);
00075     setIDT();
00076 }
```

4.26.4.4 regInterruptHandler()

```
void regInterruptHandler (
              u8 n,
              ISR handler )
```

sets the handler of the given ISR

Parameters

in	*n*	the register number
in	*handler*	the ISR that will be linked

Returns

void

Definition at line 124 of file isr.c.

```
00125 {
00126     interruptHandlers[n] = handler;
00127 }
```

4.27 isr.h

Go to the documentation of this file.

```
00001
00008 #ifndef __ISR__
00009 #define __ISR__
00010 #include "idt.h"
00011 #include "../libs/utils.h"
00012 #include "../drivers/screen.h"
00013
00018 typedef struct {
00019     u32 ds;
00020     u32 edi, esi, ebp, esp, ebx, edx, ecx, eax; /* Pushed by pusha.  */
00021     u32 intNo, errorCode;
00022     u32 eip, cs, eflags, useresp, ss; /* pushed by proc by default*/
00023 } reg;
00024
00025 typedef void (*ISR)(reg);
00026
00027 /* IRQs define */
00028 #define IRQ0 32
00029 #define IRQ1 33
00030 #define IRQ2 34
00031 #define IRQ3 35
00032 #define IRQ4 36
00033 #define IRQ5 37
00034 #define IRQ6 38
00035 #define IRQ7 39
00036 #define IRQ8 40
00037 #define IRQ9 41
00038 #define IRQ10 42
00039 #define IRQ11 43
00040 #define IRQ12 44
00041 #define IRQ13 45
00042 #define IRQ14 46
00043 #define IRQ15 47
00044
00045 /*  ISRs & IRQs reserved for CPU exceptions, defined in ASM
00046 See kernel/interrupts.asm for more info*/
00047 extern void isr0();
00048 extern void isr1();
00049 extern void isr2();
00050 extern void isr3();
00051 extern void isr4();
00052 extern void isr5();
00053 extern void isr6();
00054 extern void isr7();
00055 extern void isr8();
00056 extern void isr9();
00057 extern void isr10();
00058 extern void isr11();
00059 extern void isr12();
00060 extern void isr13();
00061 extern void isr14();
00062 extern void isr15();
00063 extern void isr16();
00064 extern void isr17();
00065 extern void isr18();
00066 extern void isr19();
00067 extern void isr20();
00068 extern void isr21();
00069 extern void isr22();
00070 extern void isr23();
00071 extern void isr24();
00072 extern void isr25();
00073 extern void isr26();
00074 extern void isr27();
00075 extern void isr28();
00076 extern void isr29();
00077 extern void isr30();
00078 extern void isr31();
00079
00080 extern void irq0();
00081 extern void irq1();
00082 extern void irq2();
00083 extern void irq3();
00084 extern void irq4();
00085 extern void irq5();
00086 extern void irq6();
00087 extern void irq7();
00088 extern void irq8();
00089 extern void irq9();
00090 extern void irq10();
00091 extern void irq11();
00092 extern void irq12();
00093 extern void irq13();
00094 extern void irq14();
00095 extern void irq15();
00096
00097 /* functions */
```

```
00105 void ISRInstall();
00106
00114 void ISRHandler(reg);
00115
00124 void regInterruptHandler(u8 , ISR);
00125
00133 void IRQHandler(reg r);
00134
00135
00136
00137 #endif
```

4.28 kernel/kernel.c File Reference

OS Entry point.

```
#include "../drivers/screen.h"
#include "../apps/cliTools.h"
#include "isr.h"
#include "idt.h"
#include "timer.h"
#include "../drivers/keyboard.h"
#include "../libs/memory.h"
```

Include dependency graph for kernel.c:

Functions

- void main ()

  the kernel entry point

4.28.1 Detailed Description

OS Entry point.

Author

Théodore MARTIN

Version

0.1

Date

2023-03-22

Definition in file kernel.c.

4.28.2 Function Documentation

4.28.2.1 main()

```
void main ( )
```

the kernel entry point

Returns

void

Definition at line 24 of file kernel.c.

```
00025 {
00026     /* Inits */
00027     ISRInstall();
00028     initTimer(50);
00029     KBInit();
00030     __asm__ volatile("sti");
00031     clearScreen();
00032     printStr("Hello, and Welcome to\n");
00033     printBanner();
00034     printStr("The future of the operating system, living like it's 1984\n");
00035     printStr("user@JuniOs:~$");
00036 }
```

4.29 kernel.c

Go to the documentation of this file.

```
00001
00009 #include "../drivers/screen.h"
00010 #include "../apps/cliTools.h"
00011 #include "isr.h"
00012 #include "idt.h"
00013 #include "timer.h"
00014 #include "../drivers/keyboard.h"
00015 #include "../libs/memory.h"
00016
00017 //https://github.com/cfenollosa/os-tutorial
00018
00024 void main()
00025 {
00026     /* Inits */
00027     ISRInstall();
00028     initTimer(50);
00029     KBInit();
00030     __asm__ volatile("sti");
00031     clearScreen();
00032     printStr("Hello, and Welcome to\n");
00033     printBanner();
00034     printStr("The future of the operating system, living like it's 1984\n");
00035     printStr("user@JuniOs:~$");
00036 }
```

4.30 kernel/timer.c File Reference

function library related to timers, date and time

```
#include "timer.h"
#include "../drivers/ports.h"
#include "isr.h"
```

Include dependency graph for timer.c:

Functions

- void initTimer (u32 f)

  Function linking our timer callback to the IRQ interruption to setup our timer. Everytime a tick passes, it will call our callback that will then increase the timer value.
- u32 getTick ()

  Get the current tick value.

Variables

- u32 tick = 0

4.30.1 Detailed Description

function library related to timers, date and time

Author

Théodore MARTIN

Version

0.1

Date

2023-03-22

Definition in file timer.c.

4.30.2 Function Documentation

4.30.2.1 getTick()

```
u32 getTick ( )
```

Get the current tick value.

Returns

u32 the tick value

Definition at line 31 of file timer.c.

```
00031                 {
00032     return tick;
00033 }
```

4.30.2.2 initTimer()

```
void initTimer (
            u32 f )
```

Function linking our timer callback to the IRQ interruption to setup our timer. Everytime a tick passes, it will call our callback that will then increase the timer value.

Parameters

in	*freq*	the frequency of the timer. used to make it faster or slower

Definition at line 22 of file timer.c.

```
00023 {
00024     regInterruptHandler(IRQ0, callback);
00025     u32 d = 1193180 / f; /* hardware clock is at 1193180 Hz*/
00026     portByteOut(0x43, 0x36);
00027     portByteOut(0x40, (u8)(d & 0xFF));
00028     portByteOut(0x40, (u8)((d » 8) & 0xFF));
00029 }
```

4.30.3 Variable Documentation

4.30.3.1 tick

```
u32 tick = 0
```

Definition at line 13 of file timer.c.

4.31 timer.c

Go to the documentation of this file.

```
00001
00008 #include "timer.h"
00009 #include "../drivers/ports.h"
00010 #include "isr.h"
00011
00012 /* Global tick count */
00013 u32 tick = 0;
00014
00015 /* private functions */
00016 static void callback()
00017 {
00018     tick++;
00019 }
00020
00021 /* public functions */
00022 void initTimer(u32 f)
00023 {
00024     regInterruptHandler(IRQ0, callback);
00025     u32 d = 1193180 / f; /* hardware clock is at 1193180 Hz*/
00026     portByteOut(0x43, 0x36);
00027     portByteOut(0x40, (u8)(d & 0xFF));
00028     portByteOut(0x40, (u8)((d » 8) & 0xFF));
00029 }
00030
00031 u32 getTick() {
00032     return tick;
00033 }
```

4.32 kernel/timer.h File Reference

function headers library related to timers, date and time

```
#include "../libs/utils.h"
```

Include dependency graph for timer.h: This graph shows which files directly or indirectly include this file:

Functions

- void initTimer (u32)

  Function linking our timer callback to the IRQ interruption to setup our timer. Everytime a tick passes, it will call our callback that will then increase the timer value.
- u32 getTick ()

  Get the current tick value.

4.32.1 Detailed Description

function headers library related to timers, date and time

Author

Théodore MARTIN

Version

0.1

Date

2023-03-22

Definition in file timer.h.

4.32.2 Function Documentation

4.32.2.1 getTick()

```
u32 getTick ( )
```

Get the current tick value.

Returns

u32 the tick value

Definition at line 31 of file timer.c.

```
00031                {
00032     return tick;
00033 }
```

4.32.2.2 initTimer()

```
void initTimer (
              u32 f )
```

Function linking our timer callback to the IRQ interruption to setup our timer. Everytime a tick passes, it will call our callback that will then increase the timer value.

Parameters

in	*freq*	the frequency of the timer. used to make it faster or slower

Definition at line 22 of file timer.c.

```
00023 {
00024     regInterruptHandler(IRQ0, callback);
00025     u32 d = 1193180 / f; /* hardware clock is at 1193180 Hz*/
00026     portByteOut(0x43, 0x36);
00027     portByteOut(0x40, (u8)(d & 0xFF));
00028     portByteOut(0x40, (u8)((d » 8) & 0xFF));
00029 }
```

4.33 timer.h

Go to the documentation of this file.

```
00001
00008 #ifndef __TIMER__
00009 #define __TIMER__
00010 #include "../libs/utils.h"
00017 void initTimer(u32);
00018
00024 u32 getTick();
00025
00026 #endif
```

4.34 libs/memory.c File Reference

rewritten function related to memory model to replace the one in stdlib

`#include "memory.h"`
Include dependency graph for memory.c:

Functions

- void mem_cpy (char *src, char *dest, u32 nbBytes)

  standard memory copy
- void mem_set (u8 *dest, u8 value, u32 len)

  standard memory set
- u32 mallok (u32 size)

  standard malloc
- u32 calloc (u32 nbObj, u32 size)

Variables

- u32 freeMemPosition = 0x10000

4.34.1 Detailed Description

rewritten function related to memory model to replace the one in stdlib

Author

Théodore MARTIN

Version

0.1

Date

2023-03-22

Definition in file memory.c.

4.34.2 Function Documentation

4.34.2.1 calloc()

```
u32 calloc (
            u32 nbObj,
            u32 size )
```

Definition at line 42 of file memory.c.

```
00043 {
00044     i32 realSize = nbObj * size;
00045     u32 baseAddr = mallok(realSize);
00046     mem_set((char *)baseAddr, 0, nbObj);
00047     return baseAddr;
00048 }
```

4.34.2.2 mallok()

```
u32 mallok (
            u32 size )
```

standard malloc

Parameters

in	*size*	the size that should allocated

Returns

u32 free memory localisation

Definition at line 31 of file memory.c.

```
00031                           {
00032     if (freeMemPosition & 0xFFFFF000) {
00033         freeMemPosition &= 0xFFFFF000;
00034         freeMemPosition += 0x1000;
00035     }
00036     //if (physicalPos) *physicalPos = freeMemPos;
00037     u32 position = freeMemPosition;
00038     freeMemPosition += size;
00039     return position;
00040 }
```

4.34.2.3 mem_cpy()

```
void mem_cpy (
            char * src,
            char * dest,
            u32 nbBytes )
```

standard memory copy

Parameters

in	*src*	the source of the copy
out	*dest*	the destination of the copy
in	*how*	many things should be copied

return void

Definition at line 15 of file memory.c.

```
00016 {
00017     for (u32 i = 0; i < nbBytes; ++i) {
00018         *dest++ = *src++;
00019     }
00020 }
```

4.34.2.4 mem_set()

```
void mem_set (
            u8 * dest,
            u8 value,
            u32 len )
```

standard memory set

Parameters

out	*dest*	where the data should be set
in	*value*	what is used to initialize the data
in	*len*	how many data should be set

Returns

void

Definition at line 23 of file memory.c.

```
00024 {
00025     u8 *temp = (u8 *)dest;
00026     for (; len != 0; --len) {
00027         *temp++ = value;
00028     }
00029 }
```

4.34.3 Variable Documentation

4.34.3.1 freeMemPosition

```
u32 freeMemPosition = 0x10000
```

Definition at line 12 of file memory.c.

4.35 memory.c

Go to the documentation of this file.

```
00001
00008 #include "memory.h"
00009
00010 /* Global variable */
00011 /* This is where we have some free memory, as the kernel starts at 0x1000 */
00012 u32 freeMemPosition = 0x10000;
00013
00014 /* Utilities - functions of the libc */
00015 void mem_cpy(char *src, char *dest, u32 nbBytes)
00016 {
00017     for (u32 i = 0; i < nbBytes; ++i) {
00018         *dest++ = *src++;
00019     }
00020 }
00021
00022 /* used to fill space of size len from dest as value */
00023 void mem_set(u8 *dest, u8 value, u32 len)
00024 {
00025     u8 *temp = (u8 *)dest;
00026     for (; len != 0; --len) {
00027         *temp++ = value;
00028     }
00029 }
00030
00031 u32 mallok(u32 size /*int align, u32 *physicalPos*/) {
00032     if (freeMemPosition & 0xFFFFF000) {
00033         freeMemPosition &= 0xFFFFF000;
00034         freeMemPosition += 0x1000;
00035     }
00036     //if (physicalPos) *physicalPos = freeMemPos;
00037     u32 position = freeMemPosition;
00038     freeMemPosition += size;
00039     return position;
00040 }
00041
00042 u32 calloc(u32 nbObj, u32 size)
00043 {
00044     i32 realSize = nbObj * size;
00045     u32 baseAddr = mallok(realSize);
00046     mem_set((char *)baseAddr, 0, nbObj);
00047     return baseAddr;
00048 }
```

4.36 libs/memory.h File Reference

File where the memory model is defined.

```
#include "utils.h"
```

Include dependency graph for memory.h: This graph shows which files directly or indirectly include this file:

Functions

- void mem_cpy (char *, char *, u32)
  standard memory copy
- void mem_set (u8 *, u8, u32)
  standard memory set
- u32 mallok (u32)
  standard malloc
- u32 callok (u32, u32)
  recreation of the standard calloc

4.36.1 Detailed Description

File where the memory model is defined.

Author

Théodore MARTIN

Version

0.1

Date

2023-03-22

Definition in file memory.h.

4.36.2 Function Documentation

4.36.2.1 callok()

```
u32 callok (
            u32 ,
            u32  )
```

recreation of the standard calloc

Parameters

in	*nbObj*	the number of object to alloc
in	*size*	the size of 1 object

Returns

the position of the allocated memory

4.36.2.2 mallok()

```
u32 mallok (
            u32 size )
```

standard malloc

Parameters

in	*size*	the size that should allocated

Returns

u32 free memory localisation

Definition at line 31 of file memory.c.

```
00031                       {
00032     if (freeMemPosition & 0xFFFFF000) {
00033         freeMemPosition &= 0xFFFFF000;
00034         freeMemPosition += 0x1000;
00035     }
00036     //if (physicalPos) *physicalPos = freeMemPos;
00037     u32 position = freeMemPosition;
00038     freeMemPosition += size;
00039     return position;
00040 }
```

4.36.2.3 mem_cpy()

```
void mem_cpy (
            char * src,
            char * dest,
            u32 nbBytes )
```

standard memory copy

Parameters

in	*src*	the source of the copy
out	*dest*	the destination of the copy
in	*how*	many things should be copied

return void

Definition at line 15 of file memory.c.

```
00016 {
00017     for (u32 i = 0; i < nbBytes; ++i) {
00018         *dest++ = *src++;
00019     }
00020 }
```

4.36.2.4 mem_set()

```
void mem_set (
              u8 * dest,
              u8 value,
              u32 len )
```

standard memory set

Parameters

out	*dest*	where the data should be set
in	*value*	what is used to initialize the data
in	*len*	how many data should be set

Returns

void

Definition at line 23 of file memory.c.

```
00024 {
00025     u8 *temp = (u8 *)dest;
00026     for (; len != 0; --len) {
00027         *temp++ = value;
00028     }
00029 }
```

4.37 memory.h

Go to the documentation of this file.

```
00001
00008 #ifndef __MEMORY__
00009 #define __MEMORY__
00010 #include "utils.h"
00011
00021 void mem_cpy(char *, char *, u32);
00022
00032 void mem_set(u8 *, u8, u32);
00033
00041 u32 mallok(u32);
00042
00051 u32 callok(u32, u32);
00052 #endif
```

4.38 libs/random.c File Reference

function library to get pseudo random numbers

```
#include "random.h"
```

Include dependency graph for random.c:

Functions

- u8 randomK ()

  get a pseudo random integer between [0; 255]
- f32 randK ()

  get a pseudo random value between [0; 1]
- void resetRandom ()

  function to reset the pseudo random generator

Variables

- u8 rngTable [256]
- i32 rngIndex = 0

4.38.1 Detailed Description

function library to get pseudo random numbers

Author

Théodore MARTIN

Version

0.1

Date

2023-03-22

Definition in file random.c.

4.38.2 Function Documentation

4.38.2.1 randK()

```
f32 randK ( )
```

get a pseudo random value between [0; 1]

Returns

f32 the random

Definition at line 40 of file random.c.

```
00041 {
00042     rngIndex = (rngIndex+1)&0xFF;
00043     return rngTable[rngIndex] / 255;
00044 }
```

4.38.2.2 randomK()

```
u8 randomK ( )
```

get a pseudo random integer between [0; 255]

Returns

u8 the random

Definition at line 34 of file random.c.

```
00035 {
00036     rngIndex = (rngIndex+1)&0xFF;
00037     return rngTable[rngIndex];
00038 }
```

4.38.2.3 resetRandom()

```
void resetRandom ( )
```

function to reset the pseudo random generator

Definition at line 46 of file random.c.

```
00047 {
00048     rngIndex = 0;
00049 }
```

4.38.3 Variable Documentation

4.38.3.1 rngIndex

```
i32 rngIndex = 0
```

Definition at line 32 of file random.c.

4.38.3.2 rngTable

```
u8 rngTable[256]
```

Initial value:

```
= {
    121, 137, 85, 187, 232, 188, 91, 133, 178, 93, 101, 208, 40, 162,
    148, 84, 253, 62, 132, 19, 122, 246, 14, 240, 223, 115, 52, 159,
    141, 30, 80, 254, 193, 114, 21, 213, 37, 249, 156, 29, 197, 237,
    252, 63, 86, 73, 225, 39, 248, 120, 194, 128, 196, 131, 67, 34,
    221, 60, 176, 220, 24, 192, 104, 113, 10, 157, 139, 81, 78, 23,
    2, 174, 212, 224, 11, 108, 143, 79, 72, 218, 82, 69, 216, 111,
    27, 76, 145, 44, 107, 83, 173, 105, 36, 219, 71, 177, 142, 230,
    167, 242, 222, 119, 26, 239, 48, 180, 245, 25, 0, 33, 153, 140,
    125, 170, 235, 147, 244, 169, 9, 151, 116, 59, 179, 127, 12, 250,
    96, 228, 203, 241, 149, 4, 20, 247, 61, 163, 118, 58, 202, 226,
    56, 171, 185, 199, 110, 22, 100, 68, 57, 152, 182, 158, 103, 189,
    175, 165, 77, 47, 43, 3, 89, 38, 161, 35, 135, 64, 209, 55,
    160, 99, 183, 31, 210, 211, 238, 217, 8, 92, 42, 201, 74, 1,
    181, 155, 205, 66, 6, 200, 144, 243, 13, 215, 45, 186, 236, 94,
    28, 184, 109, 102, 54, 126, 49, 17, 251, 138, 117, 136, 123, 65,
    146, 150, 97, 5, 75, 41, 198, 154, 95, 229, 129, 16, 98, 195,
    46, 15, 234, 32, 18, 227, 255, 88, 206, 190, 53, 70, 50, 166,
    112, 164, 90, 51, 168, 207, 130, 87, 124, 7, 191, 106, 233, 214,
    231, 172, 204, 134
}
```

Definition at line 10 of file random.c.

4.39 random.c

Go to the documentation of this file.

```
00001
00008 #include "random.h"
00009
00010 u8 rngTable[256] = {
00011     121, 137, 85, 187, 232, 188, 91, 133, 178, 93, 101, 208, 40, 162,
00012     148, 84, 253, 62, 132, 19, 122, 246, 14, 240, 223, 115, 52, 159,
00013     141, 30, 80, 254, 193, 114, 21, 213, 37, 249, 156, 29, 197, 237,
00014     252, 63, 86, 73, 225, 39, 248, 120, 194, 128, 196, 131, 67, 34,
00015     221, 60, 176, 220, 24, 192, 104, 113, 10, 157, 139, 81, 78, 23,
00016     2, 174, 212, 224, 11, 108, 143, 79, 72, 218, 82, 69, 216, 111,
00017     27, 76, 145, 44, 107, 83, 173, 105, 36, 219, 71, 177, 142, 230,
00018     167, 242, 222, 119, 26, 239, 48, 180, 245, 25, 0, 33, 153, 140,
00019     125, 170, 235, 147, 244, 169, 9, 151, 116, 59, 179, 127, 12, 250,
00020     96, 228, 203, 241, 149, 4, 20, 247, 61, 163, 118, 58, 202, 226,
00021     56, 171, 185, 199, 110, 22, 100, 68, 57, 152, 182, 158, 103, 189,
00022     175, 165, 77, 47, 43, 3, 89, 38, 161, 35, 135, 64, 209, 55,
00023     160, 99, 183, 31, 210, 211, 238, 217, 8, 92, 42, 201, 74, 1,
00024     181, 155, 205, 66, 6, 200, 144, 243, 13, 215, 45, 186, 236, 94,
00025     28, 184, 109, 102, 54, 126, 49, 17, 251, 138, 117, 136, 123, 65,
00026     146, 150, 97, 5, 75, 41, 198, 154, 95, 229, 129, 16, 98, 195,
00027     46, 15, 234, 32, 18, 227, 255, 88, 206, 190, 53, 70, 50, 166,
00028     112, 164, 90, 51, 168, 207, 130, 87, 124, 7, 191, 106, 233, 214,
00029     231, 172, 204, 134
00030 };
00031
00032 i32 rngIndex = 0;
00033
00034 u8 randomK()
00035 {
00036     rngIndex = (rngIndex+1)&0xFF;
00037     return rngTable[rngIndex];
00038 }
00039
00040 f32 randK()
00041 {
00042     rngIndex = (rngIndex+1)&0xFF;
00043     return rngTable[rngIndex] / 255;
00044 }
00045
00046 void resetRandom()
00047 {
00048     rngIndex = 0;
00049 }
```

4.40 random.h

```
00001
00008 #ifndef __K_RANDOM__
00009 #define __K_RANDOM__
00010 #include "utils.h"
00016 u8 randomK();
00017
00023 f32 randK();
00024
00029 void resetRandom();
00030
00031 #endif
```

4.41 libs/strings.c File Reference

function library to imitate strings.c from the stdlib

`#include "strings.h"`

Include dependency graph for strings.c:

Functions

- u32 str_len (char *str)
  computing the length of a string
- i8 str_cmp (char *str1, char *str2)
  compares 2 string
- void strReverse (char *str)
  reverses a string
- u32 str_cpy (string in, string out)
- void append (char *str, char n)
  adds the given char to the end of a string
- u32 wordCount (char *str, char del)
  counts the number of word separated by a delimitor in a string
- u32 biggestWord (char *str, char del)
- char * getNthWord (char *str, char del, u32 pos)
- string * strSplit (string toSplit, char del, u32 *wordNb)
  Split a string in an array of substrings.
- f64 strToL (char *str)
  converts a string to a float
- void hexToAscii (int n, char *str)
  converts a int to its hexadecimal value in a string

4.41.1 Detailed Description

function library to imitate strings.c from the stdlib

Author

Théodore MARTIN

Version

0.1

Date

2023-03-22

Definition in file strings.c.

4.41.2 Function Documentation

4.41.2.1 append()

```
void append (
            char * str,
            char n )
```

adds the given char to the end of a string

Parameters

out	*str*	the string where we should append the char
in	*chr*	the char that should be added

Returns

void

Definition at line 52 of file strings.c.

```
00053 {
00054     u32 appendPoint = str_len(str);
00055     str[appendPoint] = n;
00056     str[appendPoint + 1] = '\0';
00057 }
```

4.41.2.2 biggestWord()

```
u32 biggestWord (
            char * str,
            char del )
```

Definition at line 75 of file strings.c.

```
00076 {
00077     u32 biggest = 0;
00078     u32 temp = 0;
00079     while (*str++ != '\0') {
00080         if (*str == del) {
00081             if (temp > biggest)
00082                 biggest = temp;
00083             temp = 0;
00084         } else {
00085             temp++;
00086         }
00087     }
00088     return biggest;
00089 }
```

4.41.2.3 getNthWord()

```
char * getNthWord (
            char * str,
            char del,
            u32 pos )
```

Definition at line 91 of file strings.c.

```
00091                                                 {
00092     u32 startIndex = 0;
00093     u32 wordNb = 0;
00094     u32 size = 0;
00095     /*first, we're retrieving the starting index of the wanted word*/
00096     while (wordNb < pos && *(str+startIndex) != '\0') {
00097         if (*(str+startIndex) == del) {
00098             wordNb+=1;
00099         }
00100         startIndex+=1;
00101     }
00102     /*then, we retrieve the size of the said word*/
00103     while (*(str+startIndex+size) != del && *(str+startIndex+size) != '\0') {
00104         size+=1;
00105     }
00106     /*allocating memory to save the word*/
00107     char *result = (char *) mallok(size);
00108     /*copying it*/
00109     for (u32 i = startIndex; i < startIndex + size; ++i) {
00110         result[i - startIndex] = str[i];
00111     }
00112     return result;
00113 }
```

4.41.2.4 hexToAscii()

```
void hexToAscii (
            int n,
            char * str )
```

converts a int to its hexadecimal value in a string

Parameters

in	*n*	the number
out	*str*	where the result will be stored

Returns

void

Definition at line 198 of file strings.c.

```
00198                                   {
00199     append(str, '0');
00200     append(str, 'x');
00201     char zeros = 0;
00202     u32 tmp;
00203     int i;
00204     for (i = 28; i > 0; i -= 4) {
00205         tmp = (n >> i) & 0xF;
00206         if (tmp == 0 && zeros == 0) continue;
00207         zeros = 1;
00208         if (tmp > 0xA) append(str, tmp - 0xA + 'a');
00209         else append(str, tmp + '0');
00210     }
00211
00212     tmp = n & 0xF;
00213     if (tmp >= 0xA) append(str, tmp - 0xA + 'a');
00214     else append(str, tmp + '0');
00215 }
```

4.41.2.5 str_cmp()

```
i8 str_cmp (
            char * str1,
            char * str2 )
```

compares 2 string

Parameters

in	*str1*	the first string
in	*str2*	the second string

Returns

i8 0 if the strings are equals, else some magic int.

Definition at line 19 of file strings.c.

```
00020 {
00021     while ((*str1 != '\0' && *str2 != '\0') && *str1 == *str2) {
00022         *str1+=1;
00023         *str2+=1;
00024     }
00025     if (*str1 == *str2)
00026         return 0;
00027     else
00028         return *str1 - *str2;
00029 }
```

4.41.2.6 str_cpy()

```
u32 str_cpy (
              string in,
              string out )
```

Definition at line 42 of file strings.c.

```
00043 {
00044     if (in == NULL || out == NULL)
00045         return -1;
00046     while (*in != '\0') {
00047         *out++ = *in++;
00048     }
00049     return 0;
00050 }
```

4.41.2.7 str_len()

```
u32 str_len (
              char * str )
```

computing the length of a string

Parameters

in	*str*	the string we want the length of

Returns

u32 the string size

Definition at line 10 of file strings.c.

```
00011 {
00012     u32 result = 0;
00013     while(str[result] != '\0') {
00014         result++;
00015     }
00016     return result;
00017 }
```

4.41.2.8 strReverse()

```
void strReverse (
            char * str )
```

reverses a string

Parameters

out	*str*	the string that should and will be reversed

Returns

void

Definition at line 31 of file strings.c.

```
00032 {
00033     char temp;
00034     u32 size = str_len(str);
00035     for (u32 i = 0; i < size/2; ++i) {
00036         temp = str[i];
00037         str[i] = str[size - i - 1];
00038         str[size - i - 1] = temp;
00039     }
00040 }
```

4.41.2.9 strSplit()

```
string * strSplit (
            string toSplit,
            char del,
            u32 * wordNb )
```

Split a string in an array of substrings.

Parameters

in	*toSplit*	the string that should be splitted
in	*del*	the char separating each word
out	*wordNb*	the length of the returned array

Returns

string∗ an array containing the extracted substrings

Definition at line 136 of file strings.c.

```
00137 {
00138     if (toSplit == NULL)
00139         return NULL;
00140     u32 splitLen, biggestWordLen;
00141     *wordNb = 0;
00142     strLenBiggestWordAndWordNb(toSplit, del, &splitLen, &biggestWordLen, wordNb);
00143     string *res = NULL;
00144     res = (string *) mallok(*wordNb * sizeof(string));
00145     u32 count, index, individualSize;
00146     count = index = individualSize = 0;
00147     string buff = (string) mallok(biggestWordLen * sizeof(char));
00148     while (index <= splitLen && count < *wordNb) {
00149         if ((toSplit[index] == del && str_len(buff) != 0) || toSplit[index] == '\0') {
00150             res[count] = (string) mallok(individualSize * sizeof(char));
00151             if (res[count] == NULL)
00152                 return NULL;
00153             str_cpy(buff, res[count]);
00154             count += 1;
00155             individualSize = 0;
00156             buff[0] = '\0';
```

```
00157         } else if(toSplit[index] != del) {
00158             individualSize++;
00159             append(buff, toSplit[index]);
00160         }
00161         index++;
00162     }
00163     //freek(buff); //TODO: implement free
00164     return res;
00165 }
```

4.41.2.10 strToL()

```
f64 strToL (
            char * str )
```

converts a string to a float

Parameters

in	*str*	the string containing the number

Returns

f64 the extracted float

Definition at line 176 of file strings.c.

```
00177 {
00178     if (str == NULL || str[0] == '\0')
00179         return 0;
00180     f64 res = 0;
00181     bool neg = str[0] == '-';
00182     u32 index = neg ?  1 :  0;
00183     i32 wholePart = charIndex(str, '.');
00184     if (wholePart == -1) wholePart = str_len(str);
00185     while (str[index] != '\0') {
00186         if (str[index] == '.')  {
00187             wholePart++;
00188         } else {
00189 
00190             res += (str[index] - 48) * powk(10, wholePart - 1 - index);
00191         }
00192         index++;
00193     }
00194     res = neg ?  res * -1 :  res;
00195     return res;
00196 }
```

4.41.2.11 wordCount()

```
u32 wordCount (
            char * str,
            char del )
```

counts the number of word separated by a delimitor in a string

Parameters

in	*str*	the string that should be parsed
in	*del*	the delimitor

Returns

u32 the word number

Definition at line 59 of file strings.c.

```
00060 {
00061     if (*str == '\0')
00062         return 0;
00063
00064     u32 result = 1;
00065     bool stringDetected = false;
00066     while(*str++ != '\0') {
00067         if (*str == '\"')
00068             stringDetected = !stringDetected;
00069         if (*str == del && *(str+1) != del && !stringDetected)
00070             result++;
00071     }
00072     return result;
00073 }
```

4.42 strings.c

Go to the documentation of this file.

```
00001
00008 #include "strings.h"
00009
00010 u32 str_len(char *str)
00011 {
00012     u32 result = 0;
00013     while(str[result] != '\0') {
00014         result++;
00015     }
00016     return result;
00017 }
00018
00019 i8 str_cmp(char *str1, char *str2)
00020 {
00021     while ((*str1 != '\0' && *str2 != '\0') && *str1 == *str2) {
00022         *str1+=1;
00023         *str2+=1;
00024     }
00025     if (*str1 == *str2)
00026         return 0;
00027     else
00028         return *str1 - *str2;
00029 }
00030
00031 void strReverse(char *str)
00032 {
00033     char temp;
00034     u32 size = str_len(str);
00035     for (u32 i = 0; i < size/2; ++i) {
00036         temp = str[i];
00037         str[i] = str[size - i - 1];
00038         str[size - i - 1] = temp;
00039     }
00040 }
00041
00042 u32 str_cpy(string in, string out)
00043 {
00044     if (in == NULL || out == NULL)
00045         return -1;
00046     while (*in != '\0') {
00047         *out++ = *in++;
00048     }
00049     return 0;
00050 }
00051
00052 void append(char *str, char n)
00053 {
00054     u32 appendPoint = str_len(str);
00055     str[appendPoint] = n;
00056     str[appendPoint + 1] = '\0';
00057 }
00058
00059 u32 wordCount(char *str, char del)
00060 {
00061     if (*str == '\0')
00062         return 0;
```

```
00063
00064     u32 result = 1;
00065     bool stringDetected = false;
00066     while(*str++ != '\0') {
00067         if (*str == '\"')
00068             stringDetected = !stringDetected;
00069         if (*str == del && *(str+1) != del && !stringDetected)
00070             result++;
00071     }
00072     return result;
00073 }
00074
00075 u32 biggestWord(char *str, char del)
00076 {
00077     u32 biggest = 0;
00078     u32 temp = 0;
00079     while (*str++ != '\0') {
00080         if (*str == del) {
00081             if (temp > biggest)
00082                 biggest = temp;
00083             temp = 0;
00084         } else {
00085             temp++;
00086         }
00087     }
00088     return biggest;
00089 }
00090
00091 char *getNthWord(char *str, char del, u32 pos) {
00092     u32 startIndex = 0;
00093     u32 wordNb = 0;
00094     u32 size = 0;
00095     /*first, we're retrieving the starting index of the wanted word*/
00096     while (wordNb < pos && *(str+startIndex) != '\0') {
00097         if (*(str+startIndex) == del) {
00098             wordNb+=1;
00099         }
00100         startIndex+=1;
00101     }
00102     /*then, we retrieve the size of the said word*/
00103     while (*(str+startIndex+size) != del && *(str+startIndex+size) != '\0') {
00104         size+=1;
00105     }
00106     /*allocating memory to save the word*/
00107     char *result = (char *) mallok(size);
00108     /*copying it*/
00109     for (u32 i = startIndex; i < startIndex + size; ++i) {
00110         result[i - startIndex] = str[i];
00111     }
00112     return result;
00113 }
00114
00115 static void strLenBiggestWordAndWordNb(char *str, char del, u32 *size, u32 *biggest, u32 *wordNb)
00116 {
00117     u32 temp = 0;
00118     bool wordPreviouslyFound = false;
00119     *biggest = *size = *wordNb = 0;
00120     do {
00121         if (*str == del || *str == '\0') {
00122             if (temp > *biggest)
00123                 *biggest = temp;
00124             if (wordPreviouslyFound)
00125                 *(wordNb) += 1;
00126             wordPreviouslyFound = false;
00127             temp = 0;
00128         } else {
00129             wordPreviouslyFound = true;
00130             temp++;
00131         }
00132         *(size) += 1;
00133     } while (*str++ != '\0');
00134 }
00135
00136 string *strSplit(string toSplit, char del, u32 *wordNb)
00137 {
00138     if (toSplit == NULL)
00139         return NULL;
00140     u32 splitLen, biggestWordLen;
00141     *wordNb = 0;
00142     strLenBiggestWordAndWordNb(toSplit, del, &splitLen, &biggestWordLen, wordNb);
00143     string *res = NULL;
00144     res = (string *) mallok(*wordNb * sizeof(string));
00145     u32 count, index, individualSize;
00146     count = index = individualSize = 0;
00147     string buff = (string) mallok(biggestWordLen * sizeof(char));
00148     while (index <= splitLen && count < *wordNb) {
00149         if ((toSplit[index] == del && str_len(buff) != 0) || toSplit[index] == '\0') {
```

```
00150             res[count] = (string) mallok(individualSize * sizeof(char));
00151             if (res[count] == NULL)
00152                 return NULL;
00153             str_cpy(buff, res[count]);
00154             count += 1;
00155             individualSize = 0;
00156             buff[0] = '\0';
00157         } else if(toSplit[index] != del) {
00158             individualSize++;
00159             append(buff, toSplit[index]);
00160         }
00161         index++;
00162     }
00163     //freek(buff); //TODO: implement free
00164     return res;
00165 }
00166
00167 static i32 charIndex(char *str, char searched)
00168 {
00169     i32 res = 0;
00170     while(str[res] != searched && str[res] != '\0') {
00171         res++;
00172     }
00173     return str[res] == '\0' ?  -1 :  res;
00174 }
00175
00176 f64 strToL(char *str)
00177 {
00178     if (str == NULL || str[0] == '\0')
00179         return 0;
00180     f64 res = 0;
00181     bool neg = str[0] == '-';
00182     u32 index = neg ?  1 :  0;
00183     i32 wholePart = charIndex(str, '.');
00184     if (wholePart == -1) wholePart = str_len(str);
00185     while (str[index] != '\0') {
00186         if (str[index] == '.')  {
00187             wholePart++;
00188         } else {
00189
00190             res += (str[index] - 48) * powk(10, wholePart - 1 - index);
00191         }
00192         index++;
00193     }
00194     res = neg ?  res * -1 :  res;
00195     return res;
00196 }
00197
00198 void hexToAscii(int n, char *str) {
00199     append(str, '0');
00200     append(str, 'x');
00201     char zeros = 0;
00202     u32 tmp;
00203     int i;
00204     for (i = 28; i > 0; i -= 4) {
00205         tmp = (n >> i) & 0xF;
00206         if (tmp == 0 && zeros == 0) continue;
00207         zeros = 1;
00208         if (tmp > 0xA) append(str, tmp - 0xA + 'a');
00209         else append(str, tmp + '0');
00210     }
00211
00212     tmp = n & 0xF;
00213     if (tmp >= 0xA) append(str, tmp - 0xA + 'a');
00214     else append(str, tmp + '0');
00215 }
```

4.43 libs/strings.h File Reference

functions header library to imitate strings.c from the stdlib

```
#include "utils.h"
#include "memory.h"
```

Include dependency graph for strings.h: This graph shows which files directly or indirectly include this file:

Functions

- u32 str_len (char ∗)
  computing the length of a string
- i8 str_cmp (char ∗, char ∗)
  compares 2 string
- void strReverse (char ∗)
  reverses a string
- void append (char ∗, char)
  adds the given char to the end of a string
- u32 wordCount (char ∗, char)
  counts the number of word separated by a delimitor in a string
- string ∗ strSplit (string, char, u32 ∗)
  Split a string in an array of substrings.
- f64 strToL (char ∗)
  converts a string to a float
- void hexToAscii (int, char ∗)
  converts a int to its hexadecimal value in a string

4.43.1 Detailed Description

functions header library to imitate strings.c from the stdlib

Author

Théodore MARTIN

Version

0.1

Date

2023-03-22

Definition in file strings.h.

4.43.2 Function Documentation

4.43.2.1 append()

```
void append (
            char * str,
            char n )
```

adds the given char to the end of a string

Parameters

out	*str*	the string where we should append the char
in	*chr*	the char that should be added

Returns

void

Definition at line 52 of file strings.c.

```
00053 {
00054     u32 appendPoint = str_len(str);
00055     str[appendPoint] = n;
00056     str[appendPoint + 1] = '\0';
00057 }
```

4.43.2.2 hexToAscii()

```
void hexToAscii (
            int n,
            char * str )
```

converts a int to its hexadecimal value in a string

Parameters

in	*n*	the number
out	*str*	where the result will be stored

Returns

void

Definition at line 198 of file strings.c.

```
00198                                   {
00199     append(str, '0');
00200     append(str, 'x');
00201     char zeros = 0;
00202     u32 tmp;
00203     int i;
00204     for (i = 28; i > 0; i -= 4) {
00205         tmp = (n >> i) & 0xF;
00206         if (tmp == 0 && zeros == 0) continue;
00207         zeros = 1;
00208         if (tmp > 0xA) append(str, tmp - 0xA + 'a');
00209         else append(str, tmp + '0');
00210     }
00211
00212     tmp = n & 0xF;
00213     if (tmp >= 0xA) append(str, tmp - 0xA + 'a');
00214     else append(str, tmp + '0');
00215 }
```

4.43.2.3 str_cmp()

```
i8 str_cmp (
              char * str1,
              char * str2 )
```

compares 2 string

Parameters

in	*str1*	the first string
in	*str2*	the second string

Returns

i8 0 if the strings are equals, else some magic int.

Definition at line 19 of file strings.c.

```
00020 {
00021     while ((*str1 != '\0' && *str2 != '\0') && *str1 == *str2) {
00022         *str1+=1;
00023         *str2+=1;
00024     }
00025     if (*str1 == *str2)
00026         return 0;
00027     else
00028         return *str1 - *str2;
00029 }
```

4.43.2.4 str_len()

```
u32 str_len (
              char * str )
```

computing the length of a string

Parameters

in	*str*	the string we want the length of

Returns

u32 the string size

Definition at line 10 of file strings.c.

```
00011 {
00012     u32 result = 0;
00013     while(str[result] != '\0') {
00014         result++;
00015     }
00016     return result;
00017 }
```

4.43.2.5 strReverse()

```
void strReverse (
            char * str )
```

reverses a string

Parameters

out	*str*	the string that should and will be reversed

Returns

void

Definition at line 31 of file strings.c.

```
00032 {
00033     char temp;
00034     u32 size = str_len(str);
00035     for (u32 i = 0; i < size/2; ++i) {
00036         temp = str[i];
00037         str[i] = str[size - i - 1];
00038         str[size - i - 1] = temp;
00039     }
00040 }
```

4.43.2.6 strSplit()

```
string * strSplit (
             string toSplit,
             char del,
             u32 * wordNb )
```

Split a string in an array of substrings.

Parameters

in	*toSplit*	the string that should be splitted
in	*del*	the char separating each word
out	*wordNb*	the length of the returned array

Returns

string* an array containing the extracted substrings

Definition at line 136 of file strings.c.

```
00137 {
00138     if (toSplit == NULL)
00139         return NULL;
00140     u32 splitLen, biggestWordLen;
00141     *wordNb = 0;
00142     strLenBiggestWordAndWordNb(toSplit, del, &splitLen, &biggestWordLen, wordNb);
00143     string *res = NULL;
00144     res = (string *) mallok(*wordNb * sizeof(string));
00145     u32 count, index, individualSize;
```

```
00146     count = index = individualSize = 0;
00147     string buff = (string) mallok(biggestWordLen * sizeof(char));
00148     while (index <= splitLen && count < *wordNb) {
00149         if ((toSplit[index] == del && str_len(buff) != 0) || toSplit[index] == '\0') {
00150             res[count] = (string) mallok(individualSize * sizeof(char));
00151             if (res[count] == NULL)
00152                 return NULL;
00153             str_cpy(buff, res[count]);
00154             count += 1;
00155             individualSize = 0;
00156             buff[0] = '\0';
00157         } else if(toSplit[index] != del) {
00158             individualSize++;
00159             append(buff, toSplit[index]);
00160         }
00161         index++;
00162     }
00163     //freek(buff); //TODO: implement free
00164     return res;
00165 }
```

4.43.2.7 strToL()

```
f64 strToL (
            char * str )
```

converts a string to a float

Parameters

in	*str*	the string containing the number

Returns

f64 the extracted float

Definition at line 176 of file strings.c.

```
00177 {
00178     if (str == NULL || str[0] == '\0')
00179         return 0;
00180     f64 res = 0;
00181     bool neg = str[0] == '-';
00182     u32 index = neg ?  1 :  0;
00183     i32 wholePart = charIndex(str, '.');
00184     if (wholePart == -1) wholePart = str_len(str);
00185     while (str[index] != '\0') {
00186         if (str[index] == '.')  {
00187             wholePart++;
00188         } else {
00189 
00190             res += (str[index] - 48) * powk(10, wholePart - 1 - index);
00191         }
00192         index++;
00193     }
00194     res = neg ?  res * -1 :  res;
00195     return res;
00196 }
```

4.43.2.8 wordCount()

```
u32 wordCount (
            char * str,
            char del )
```

counts the number of word separated by a delimitor in a string

Parameters

in	*str*	the string that should be parsed
in	*del*	the delimitor

Returns

u32 the word number

Definition at line 59 of file strings.c.

```
00060 {
00061     if (*str == '\0')
00062         return 0;
00063
00064     u32 result = 1;
00065     bool stringDetected = false;
00066     while(*str++ != '\0') {
00067         if (*str == '\"')
00068             stringDetected = !stringDetected;
00069         if (*str == del && *(str+1) != del && !stringDetected)
00070             result++;
00071     }
00072     return result;
00073 }
```

4.44 strings.h

Go to the documentation of this file.

```
00001
00009 #ifndef __STRINGS__
00010 #define __STRINGS__
00011
00012 #include "utils.h"
00013 #include "memory.h"
00014
00022 u32 str_len(char *);
00023
00032 i8 str_cmp(char *, char *);
00033
00041 void strReverse(char *);
00042
00051 void append(char *, char);
00052
00061 u32 wordCount(char *, char);
00062
00072 string *strSplit(string, char, u32*);
00073
00081 f64 strToL(char *);
00082
00091 void hexToAscii(int ,char *);
00092
00093 #endif
```

4.45 utils.c

```
00001 #include "utils.h"
00002
00003 void iToA(i32 n, char *str) {
00004     int i, sign;
00005     if ((sign = n) < 0) n = -n;
00006     i = 0;
00007     do {
00008         str[i++] = n % 10 + '0';
00009     } while ((n /= 10) > 0);
00010
00011     if (sign < 0) str[i++] = '-';
00012     str[i] = '\0';
00013
00014     /* TODO: implement "reverse" */
00015 }
```

```
00016
00017 f64 powk(f64 base, i64 exp)
00018 {
00019     if (exp < 0) return 1 / powk(base, -exp);
00020     else if (exp == 1) return base;
00021     else if (exp == 0) return 1;
00022     else return base * powk(base, exp-1);
00023 }
```

4.46 utils.h

```
00001 #ifndef __UTILS__
00002 #define __UTILS__
00003
00004 /* stdint definitions */
00005 typedef unsigned char u8;/* max:  FF*/
00006 typedef unsigned short u16;/* max:  FFFF*/
00007 typedef unsigned int u32;/*max FFFF FFFF*/
00008 typedef unsigned long long u64;/*max FFFF FFFF FFFF FFFF*/
00009 typedef char i8;
00010 typedef short i16;
00011 typedef int i32;
00012 typedef long long i64;
00013 typedef u32 size_t;
00014 typedef u32 uintptr_t;
00015 typedef float f32;
00016 typedef double f64;
00017 /*string imitation*/
00018 typedef char* string;
00019
00020 /* boolean recreation */
00021 typedef u8 bool;
00022 #define true (1)
00023 #define false (0)
00024 /* you know why...  */
00025 #define NULL (0)
00026
00034 #define low16(addr) (u16)((addr) & 0xFFFF)
00035
00043 #define high16(addr) (u16)(((addr) » 16) & 0xFFFF)
00044
00053 void iToA(i32, char*);
00054
00063 f64 powk(f64, i64);
00064
00065 #endif
```

Index

__attribute__, 5
    base, 5
    flags, 5
    hOffset, 5
    limit, 6
    lOffset, 6
    sel, 6
    zero, 6

append
    strings.c, 72
    strings.h, 82
apps/cliTools.c, 11, 13
apps/cliTools.h, 14, 16
ata.c
    readSectors, 16
    writeSectors, 17
ata.h
    DRIVE_BUSY, 19
    DRIVE_ERROR, 20
    DRIVE_FAULTY, 20
    DRIVE_READY, 20
    DRIVE_WAITING, 20
    readSectors, 20
    writeSectors, 21

BACKSPACE
    keyboard.h, 25
base
    __attribute__, 5
biggestWord
    strings.c, 73

CALCPRES
    cliTools.c, 12
calloc
    memory.c, 63
callok
    memory.h, 66
clearScreen
    screen.c, 31
cliTools.c
    CALCPRES, 12
    printBanner, 12
    userInputHandler, 12
cliTools.h
    printBanner, 14
    userInputHandler, 15
cs
    reg, 7

DRIVE_BUSY
    ata.h, 19
DRIVE_ERROR
    ata.h, 20
DRIVE_FAULTY
    ata.h, 20
DRIVE_READY
    ata.h, 20
DRIVE_WAITING
    ata.h, 20
drivers/ata.c, 16, 18
drivers/ata.h, 19, 22
drivers/keyboard.c, 22, 24
drivers/keyboard.h, 24, 27
drivers/ports.c, 27
drivers/ports.h, 27, 30
drivers/screen.c, 30, 35
drivers/screen.h, 36
ds
    reg, 7

eax
    reg, 7
ebp
    reg, 8
ebx
    reg, 8
ecx
    reg, 8
edi
    reg, 8
edx
    reg, 8
eflags
    reg, 8
eip
    reg, 9
ENTER
    keyboard.h, 25
errorCode
    reg, 9
esi
    reg, 9
esp
    reg, 9
exceptionMessages
    isr.c, 46

flags
    __attribute__, 5

freeMemPosition
memory.c, 65

getNthWord
strings.c, 73
getTick
timer.c, 59
timer.h, 61
getVGAOffset
screen.c, 31

hexToAscii
strings.c, 73
strings.h, 83
hOffset
__attribute__, 5

IDT
idt.c, 38
idt.h, 41
idt.c
IDT, 38
IDTReg, 38
setIDT, 37
setIDTGate, 38
idt.h
IDT, 41
IDTNB, 40
IDTReg, 41
KERNCS, 40
setIDT, 40
setIDTGate, 41
IDTNB
idt.h, 40
IDTReg
idt.c, 38
idt.h, 41
initTimer
timer.c, 59
timer.h, 61
interruptHandlers
isr.c, 46
intNo
reg, 9
IRQ0
isr.h, 50
IRQ1
isr.h, 50
IRQ10
isr.h, 50
IRQ11
isr.h, 50
IRQ12
isr.h, 51
IRQ13
isr.h, 51
IRQ14
isr.h, 51
IRQ15
isr.h, 51
IRQ2
isr.h, 51
IRQ3
isr.h, 51
IRQ4
isr.h, 52
IRQ5
isr.h, 52
IRQ6
isr.h, 52
IRQ7
isr.h, 52
IRQ8
isr.h, 52
IRQ9
isr.h, 52
IRQHandler
isr.c, 43
isr.h, 53
ISR
isr.h, 53
isr.c
exceptionMessages, 46
interruptHandlers, 46
IRQHandler, 43
ISRHandler, 44
ISRInstall, 44
regInterruptHandler, 45
isr.h
IRQ0, 50
IRQ1, 50
IRQ10, 50
IRQ11, 50
IRQ12, 51
IRQ13, 51
IRQ14, 51
IRQ15, 51
IRQ2, 51
IRQ3, 51
IRQ4, 52
IRQ5, 52
IRQ6, 52
IRQ7, 52
IRQ8, 52
IRQ9, 52
IRQHandler, 53
ISR, 53
ISRHandler, 53
ISRInstall, 54
regInterruptHandler, 55
ISRHandler
isr.c, 44
isr.h, 53
ISRInstall
isr.c, 44
isr.h, 54

KBInit

keyboard.c, 23
keyboard.h, 26
KERNCS
idt.h, 40
kernel.c
main, 57
kernel/idt.c, 37, 39
kernel/idt.h, 39, 42
kernel/isr.c, 42, 46
kernel/isr.h, 48, 55
kernel/kernel.c, 57, 58
kernel/timer.c, 58, 60
kernel/timer.h, 60, 62
keyboard.c
KBInit, 23
keycodeToAscii, 23
keyboard.h
BACKSPACE, 25
ENTER, 25
KBInit, 26
SCMAX, 26
UP_ARROW, 26
USER_BUFF_SIZE, 26
keycodeToAscii
keyboard.c, 23

libs/memory.c, 62, 65
libs/memory.h, 66, 68
libs/random.c, 68, 71
libs/random.h, 71
libs/strings.c, 71, 79
libs/strings.h, 81, 87
libs/utils.c, 87
libs/utils.h, 88
limit
__attribute__, 6
lOffset
__attribute__, 6

main
kernel.c, 57
mallok
memory.c, 63
memory.h, 67
mem_cpy
memory.c, 64
memory.h, 67
mem_set
memory.c, 64
memory.h, 68
memory.c
calloc, 63
freeMemPosition, 65
mallok, 63
mem_cpy, 64
mem_set, 64
memory.h
callok, 66
mallok, 67
mem_cpy, 67
mem_set, 68

portByteIn
ports.h, 28
portByteOut
ports.h, 29
ports.h
portByteIn, 28
portByteOut, 29
portWordIn, 29
portWordOut, 29
portWordIn
ports.h, 29
portWordOut
ports.h, 29
printBanner
cliTools.c, 12
cliTools.h, 14
printStr
screen.c, 32
printStrAtPos
screen.c, 32
putchar
screen.c, 33
putcharAtPos
screen.c, 33

randK
random.c, 69
random.c
randK, 69
randomK, 69
resetRandom, 70
rngIndex, 70
rngTable, 70
randomK
random.c, 69
readSectors
ata.c, 16
ata.h, 20
reg, 6
cs, 7
ds, 7
eax, 7
ebp, 8
ebx, 8
ecx, 8
edi, 8
edx, 8
eflags, 8
eip, 9
errorCode, 9
esi, 9
esp, 9
intNo, 9
ss, 9
useresp, 9
regInterruptHandler

isr.c, 45
isr.h, 55
removeLastChar
screen.c, 34
resetRandom
random.c, 70
rngIndex
random.c, 70
rngTable
random.c, 70

SCMAX
keyboard.h, 26
screen.c
clearScreen, 31
getVGAOffset, 31
printStr, 32
printStrAtPos, 32
putchar, 33
putcharAtPos, 33
removeLastChar, 34
setVGAOffset, 34
sel
__attribute__, 6
setIDT
idt.c, 37
idt.h, 40
setIDTGate
idt.c, 38
idt.h, 41
setVGAOffset
screen.c, 34
ss
reg, 9
str_cmp
strings.c, 74
strings.h, 83
str_cpy
strings.c, 75
str_len
strings.c, 75
strings.h, 84
strings.c
append, 72
biggestWord, 73
getNthWord, 73
hexToAscii, 73
str_cmp, 74
str_cpy, 75
str_len, 75
strReverse, 75
strSplit, 77
strToL, 78
wordCount, 78
strings.h
append, 82
hexToAscii, 83
str_cmp, 83
str_len, 84
strReverse, 84
strSplit, 85
strToL, 86
wordCount, 86
strReverse
strings.c, 75
strings.h, 84
strSplit
strings.c, 77
strings.h, 85
strToL
strings.c, 78
strings.h, 86

tick
timer.c, 60
timer.c
getTick, 59
initTimer, 59
tick, 60
timer.h
getTick, 61
initTimer, 61

UP_ARROW
keyboard.h, 26
USER_BUFF_SIZE
keyboard.h, 26
useresp
reg, 9
userInputHandler
cliTools.c, 12
cliTools.h, 15

wordCount
strings.c, 78
strings.h, 86
writeSectors
ata.c, 17
ata.h, 21

zero
__attribute__, 6