Chapter 1

STDLIB functions

All functions are prefixed with an underscore character ('_').

1.1 init

The init function is supposed to be called at the very beginning of the program.

1.2 exception

The exception function is called whenever a syscall returns an error.

1.3 print_char

The print_char function prints the character at address (%rax) into the file descriptor stored in %r8.

1.4 printNumber

The printNumber function prints the value of %rax into the file descriptor stored in %r8.(in decimal)

1.5 printBinNumber

The same as printNumber, but this one writes it in binary.

1.6 printNewLine

Prints a new line character ('
n') into the file descriptor stored in %r8.

1.7 readValue

Reads the next value from the file descriptor stored in %r8(in decimal), ending with either a space, a new line character or the end of the buffer and stores it in %rax.

1.8 readChar

Reads a single char and stores the ASCII value in %rax from the file descriptor stored in %r8.

1.9 f_ro_open

Stands for **F**ile **R**ead **O**nly **OPEN**. Opens a file descriptor from the zero-terminated string whose first byte's address is stored in %rax and returns the file descriptor in %ax.

1.10 f_wo_open

Stands for **F**ile **W**rite **O**nly **OPEN**. %rax should contain the address of the first byte of the name(terminated by 0) and %rbx the file permissions to be used if the file does not exist. If the file exists it is truncated.

1.11 f_close

Closes the file descriptor stored in %rax.

1.12 swap

Swaps the values stored at the memory addresses of %rcx and %rdx.

1.13 sort

Bubble sorts the memory addresses stored in the range `[%rax,%rbx)`. %rbx - %rax should be equal to the number of elements that need to be sorted.

1.14 reverse_sort

Same as sort, but in the reverse order.

1.15 reverse

Reverses the memory between `[%rax,%rbx)`.

1.16 exit

Prints the code stored in %rax and exits. This function is called by the exception function.

1.17 prime

Checks if the number stored in %rax is prime or not. If it is prime, then %rax will have the value 1, else 0.

1.18 printString

Prints the string that begins at the memory location stored in %rax and has the length stored in %rbx.

1.19 Compiled C functions

All C functions that are used are defined in Appendix B.

Appendix A

The source code

```
# SYSCALL numbers

.equ SYS_READ,                      0
.equ SYS_WRITE,                     1
.equ SYS_OPEN,                      2
.equ SYS_CLOSE,                     3
.equ SYS_EXIT,                      60
.equ SYS_CREATE,                    85

# STANDARD STREAMS
.equ STDIN,                         0
.equ STDOUT,                        1
.equ STDERR,                        2

.section .bss
    .lcomm INTERNAL____READ_PTR,    16
    .lcomm INTERNAL____READ,        65536
    .lcomm MERGE_MEMORY,            524288

.section .rodata
    __exc: .ascii                   "An exception occured: "
.equ __exc_len,                     22
    digits: .byte                   48, 49, 50, 51, 52, 53, 54, 55, 56, 57
    new_line: .byte                 10
    __exit: .ascii                  "Process finished with exit code "
.equ __exit_len,                    32
```

```
.section .text

.global                             _init
.type                               _init, @function
_init:
movq   $0x0, INTERNAL____READ_PTR
movq   $0x0, INTERNAL____READ_PTR+8
retq

.global                             _exception
.type                               _exception, @function
_exception:
.cfi_startproc
negq    %rax
pushq   %rax
movq    $SYS_WRITE, %rax
movq    $STDERR, %rdi
movq    $__exc, %rsi
movq    $__exc_len, %rdx
syscall
movq    (%rsp), %rax
movl    $2, %r8d
call    _print_number
call    _print_new_line
popq    %rax
call    _exit
retq
.cfi_endproc

.global                             _print_char
.type                               _print_char, @function
_print_char:
.cfi_startproc
movq    %rax, %rsi
movq    $SYS_WRITE, %rax
movq    %r8, %rdi
movq    $1, %rdx
pushq   %r8
syscall
```

```
popq    %r8
or      %rax,%rax
jns     print_char.l1
call    _exception
print_char.l1:
mov     %rsi,%rax
ret
.cfi_endproc

.global                            _print_number
.type                              _print_number, @function
_print_number:
.cfi_startproc
pushq   %rax
pushq   %rdx
xor     %edx,%edx
movq    $10, %r15
idiv    %r15d
test    %eax,%eax
je      printNumber.l1
call    _print_number
printNumber.l1:
leaq    digits(%rdx), %rax
call    _print_char
popq    %rdx
popq    %rax
retq
.cfi_endproc

.global                            _print_bin_number
.type                              _print_bin_number, @function
_print_bin_number:
.cfi_startproc
pushq   %rax
pushq   %rdx
movq    %rax, %rdx
and     $1, %rdx
shr     $1, %rax
test    %rax,%rax
je      printBinNumber.l1
```

```
call    _print_bin_number
printBinNumber.l1:
leaq    digits(%rdx), %rax
call    _print_char
popq    %rdx
popq    %rax
retq
.cfi_endproc

.global                             _print_new_line
.type                               _print_new_line, @function
_print_new_line:
.cfi_startproc
movq    $SYS_WRITE, %rax
movq    %r8, %rdi
movq    $new_line, %rsi
movq    $1, %rdx
syscall
or      %rax,%rax
jns     printNewLine.l1
call    _exception
printNewLine.l1:
retq
.cfi_endproc

.global                             _read_value
.type                               _read_value, @function
_read_value:
.cfi_startproc
movq    INTERNAL____READ_PTR, %r11
movq    INTERNAL____READ_PTR+8, %r12
cmpq    %r12,%r11
jl      readValue.l1
movq    $SYS_READ, %rax
movq    %r8, %rdi
movq    $INTERNAL____READ, %rsi
movq    $65536, %rdx
syscall
or      %rax,%rax
jns     readValue.l2
```

```
call    _exception
readValue.l2:
movq    %rax,%rbx
cmpq    $STDIN, %r8
jne     readValue.l3
subq    $1, %rbx
readValue.l3:
movq    %rbx, INTERNAL____READ_PTR+8
movq    $0, %r10
jmp     readValue.l4
readValue.l1:
movq    %r11,%r10
movq    %r12, %rbx
readValue.l4:
movq    $0, %rax
readValue.l5:
movzxb  INTERNAL____READ(%r10), %rcx
cmpq    $0x20, %rcx
je      readValue.l6
cmpq    $0xa, %rcx
je      readValue.l6
cmpq    INTERNAL____READ_PTR+8, %r10
jge     readValue.l6
subq    $0x30, %rcx# rcx=rcx-'0'
incq    %r10
movq    $10, %r13
mul     %r13d
addq    %rcx, %rax
cmpq    %rbx, %r10
jl      readValue.l5
readValue.l6:
incq    %r10
cmpq    INTERNAL____READ_PTR+8, %r10
jge     readValue.l7
movzxb  INTERNAL____READ(%r10), %rcx
cmpq    $0xA, %rcx# '\n'
je      readValue.l6
cmpq    $0x20, %rcx# ' '
je      readValue.l6
readValue.l7:
```

```
movq    %r10, INTERNAL____READ_PTR
retq
.cfi_endproc

.global                              _read_char
.type                                _read_char, @function
_read_char:
.cfi_startproc
movq    INTERNAL____READ_PTR, %r11
movq    INTERNAL____READ_PTR+8, %r12
cmpq    %r12, %r11
jl      readChar.l1
movq    $SYS_READ, %rax
movq    %r8, %rdi
movq    $INTERNAL____READ, %rsi
movq    $1, %rdx
syscall
or      %rax,%rax
jns     readChar.l2
call    _exception
readChar.l2:
movq    %rax,%rbx
cmpq    $STDIN, %r8
jne     readChar.l3
subq    $1, %rbx
readChar.l3:
movq    %rbx, INTERNAL____READ_PTR+8
movq    $0, %r10
jmp     readChar.l4
readChar.l1:
movq    %r11, %r10
readChar.l4:
movzxb  INTERNAL____READ(%r10),%rax
incq    %r10
movq    %r10, INTERNAL____READ_PTR
retq
.cfi_endproc

.global                              _f_ro_open
.type                                _f_ro_open, @function
```

```
_f_ro_open:
.cfi_startproc
movq    $0, %rsi
movq    %rax, %rdi
movq    $SYS_OPEN, %rax
syscall
cmpq    $0, %rax
jl      f_ro_open.l1
retq
f_ro_open.l1:
call    _exception
retq
.cfi_endproc

.global                             _f_wo_open
.type                               _f_wo_open, @function
_f_wo_open:
.cfi_startproc
movq    $577, %rsi # O_TRUNC | O_CREAT | O_WRONLY
movq    %rbx, %rdx
movq    %rax, %rdi
movq    $SYS_OPEN, %rax
syscall
cmpq    $0x0, %rax
jl      f_wo_open.l1
retq
f_wo_open.l1:
call    _exception
retq
.cfi_endproc

.global                             _f_close
.type                               _f_close, @function
_f_close:
.cfi_startproc
movq    %rax, %rdi
movq    $SYS_CLOSE, %rax
syscall
or      %rax, %rax
jns     f_close.l1
```

```
call    _exception
f_close.l1:
retq
.cfi_endproc

.global                                 _swap
.type                                   _swap, @function
_swap:
.cfi_startproc
movq    (%rcx), %r9
movq    (%rdx), %r8
movq    %r8, (%rcx)
movq    %r9, (%rdx)
retq
.cfi_endproc

.global                                 _sort
.type                                   _sort, @function
_sort:
.cfi_startproc
subq    %rax, %rbx
sort.l1:
movq    $0x1, %r10
movq    $0x0, %r11
sort.l2:
movq    (%rax, %r10, 8), %r8
cmpq    -8(%rax, %r10, 8), %r8
jge     sort.l3
leaq    (%rax, %r10, 8), %rcx
leaq    -8(%rcx), %rdx
call    _swap
movq    $0x1, %r11
sort.l3:
incq    %r10
cmpq    %rbx, %r10
jl      sort.l2
cmpq    $0x0, %r11
jne     sort.l1
retq
.cfi_endproc
```

```
.global                              _reverse_sort
.type                                _reverse_sort, @function
_reverse_sort:
.cfi_startproc
movq    $0x0, %r11
subq    %rax,%rbx
reverse_sort.l1:
movq    $0x1, %r10
movq    $0x0, %r11
reverse_sort.l2:
movq    (%rax, %r10, 8), %r8
cmpq    -0x8(%rax, %r10, 8), %r8
jle     reverse_sort.l3
leaq    (%rax, %r10, 8), %rcx
leaq    -0x8(%rax, %r10, 8), %rdx
call    _swap
movq    $0x1, %r11
reverse_sort.l3:
incq    %r10
cmpq    %rbx, %r10
jl      reverse_sort.l2
cmpq    $0x0, %r11
jne     reverse_sort.l1
retq
.cfi_endproc

.global                              _reverse
.type                                _reverse, @function
_reverse:
.cfi_startproc
# rax = begin in memory
# rbx = begin in memory + size + 1
movq    %rax, %rcx
subq    $0x1, %rbx# rbx = begin in memory + size
subq    %rax, %rbx# rbx = size
leaq    (,%rbx,8), %rbx# rbx = size in memory
leaq    (%rax, %rbx), %rdx# rdx = end in memory
reverse.l1:
call    _swap # swaps rcx and rdx
```

```
addq    $0x8, %rcx
subq    $0x8, %rdx
cmpq    %rdx, %rcx
jl      reverse.l1
retq
.cfi_endproc

.global                                 _exit
.type                                   _exit, @function
_exit:
.cfi_startproc
pushq   %rax
movq    $SYS_WRITE, %rax
movq    $STDOUT, %rdi
movq    $__exit, %rsi
movq    $__exit_len, %rdx
syscall
movq    (%rsp), %rax
movq    $STDOUT, %r8
call    _print_number
call    _print_new_line
movq    $SYS_EXIT, %rax
popq    %rdi
syscall
retq
.cfi_endproc

.global                                 _print_string
.type                                   _print_string, @function
_print_string:
# rax = address of first char
# rbx = size
# r8 = target file descriptor
movq    %r8, %rdi
movq    %rax, %rsi
movq    %rbx, %rdx
pushq   %r8
movq    $SYS_WRITE, %rax
syscall
popq    %r8
```

```
or      %rax, %rax
jns     printString.l1
call    _exception
printString.l1:
retq

## Template:
# .globl <name>
# .type <name>, @function
# <name>:
#   .cfi_startproc
#   <code>
#   .cfi_endproc

.globl _prime
.type _prime, @function
_prime:
.cfi_startproc
#rax = value
cmpq    $2, %rax
jl      _prime.false # 0 and 1
cmpq    $4, %rax
jl      _prime.true # 2 and 3
movq    %rax, %r10
andq    $1, %r10
jz      _prime.false # if arg & 1 == 0 then arg is odd
movq    %rax, %rbx
movq    $0, %rdx
movq    $6, %r10
idiv    %r10
cmpq    $1, %rdx
je      _prime.l2
cmpq    $5, %rdx
je      _prime.l2
jmp     _prime.false # except for 2 and 3, there is no prime that cannot be written as 6n
_prime.l2:
movq    $3, %r10 # i
_prime.l1:
movq    %r10, %rax
xorq    %rdx, %rdx
```

```
imul    %r10
cmpq    %rbx, %rax
jg      _prime.true # i * i > arg
movq    %rbx, %rax
movq    $0, %rdx
idiv    %r10
cmpq    $0, %rdx
je      _prime.false # arg % i == 0
addq    $2, %r10
jmp     _prime.l1
_prime.true:
movq    $1, %rax
retq
_prime.false:
movq    $0, %rax
retq
.cfi_endproc
.size _prime, .-_prime

.global                                 _div_sum
.type                                   _div_sum, @function
_div_sum:
.cfi_startproc
# rbx = arg
# rcx = return value
movq    $1, %rcx
movq    $2, %rdi # i
_div_sum.l1:
movq    %rbx, %rax
xorq    %rdx, %rdx
div     %rdi
test    %rdx, %rdx
jne     _div_sum.l2
addq    %rdi, %rcx
_div_sum.l2:
incq    %rdi
cmpq    %rbx, %rdi
jle     _div_sum.l1
retq
.cfi_endproc
```

```
.size                                    _div_sum, .-_div_sum

.global                                  _perfect
.type                                    _perfect, @function
_perfect:
# rax = value
movq    %rax, %rbx
movq    $1, %rcx
movq    $2, %rdi
_perfect.l1:
movq    %rbx, %rax
xorq    %rdx, %rdx
div     %rdi
test    %rdx, %rdx
jne     _perfect.l2
add     %rdi, %rcx
_perfect.l2:
incq    %rdi
test    %rdi, %rdi
je      _perfect.false
leaq    (,%rdi,2), %rax
cmpq    %rbx, %rax
jle     _perfect.l1
cmpq    %rbx, %rcx
jne     _perfect.false
_perfect.true:
movq    $1, %rax
retq
_perfect.false:
movq    $0, %rax
retq
.size                                    _perfect, .-_perfect
```

Appendix B

The C functions

```
#include <stdlib.h>
#include <stdio.h>

long long MERGE_MEMORY[65536];

void merge(register long long* a, register long long *b, register long long s1, register
        register long long i=0, j=0, k=0;
        register long long* mem = MERGE_MEMORY;
        while(i < s1 && j < s2){
                if(*(a+i) < *(b+j)){
                        mem[k++] = *(a+(i++));
                }
                else{
                        mem[k++] = *(b+(j++));
                }
        }
        if(i == s1){
                while(j < s2){
                        mem[k++] = *(b+(j++));
                }
        }
        else{
                while(i < s1){
                        mem[k++] = *(a+(i++));
                }
        }
        k = 0;
        for(i = 0; i < s1; i++){
                *(a+i) = mem[k++];
        }
        for(j = 0; j < s2; j++){
                *(b+j) = mem[k++];
        }
}

void merge_sort(register void* a, register long long s){
        if(s <= 1)
                return;
        register long long middlepos = s / 2;
        register long long *midadr = (long long*)a + middlepos;
        merge_sort(a, middlepos);
        merge_sort((void*)((long long*)a + middlepos), s - middlepos);
        merge(a,  midadr, middlepos, s - middlepos);
}
```