1 Формулировка задания

Разработать программно-аппаратный прототип внешнего хранилища данных. В качестве места хранения данных выступает внутренняя память данных МК ATmega32 (EEPROM). Формат хранения данных соответствует упрощённой структуре файловой системы FAT. В качестве хранимых объектов могут выступать файлы и каталоги, каталоги образуют иерархическую структуру. Для каждого файла/каталога отводится набор кластеров (но не менее одного кластера на объект).

Разработанный програмный комплекс должен удовлетворять требованиям:

- Ввод pin-кода с регистра PORTB, длина 1 байт;
- Отображение на семисегментном индикаторе имени текущего каталога (в шестнадцатеричном формате);
- Прерывание Int0 переключает отображение имени последнего активного объекта / его адрес;

И выполнять команды:

- LoginAsUser (если pin-код не указан в файловой системе, то сохранение введённого pin-кода)
- LoginAsRoot (если pin-код не указан в файловой системе, то сохранение введённого pin-кода)
- Logout
- SetRootPin
- SetUserPin
- CheckClusterChain (Проверка корректности
- массива указателей)
- GetFileInfo / GetDirInfo
- CreateFile / CreateDirectory
- ListDirectory (получение имён объектов каталога)
- DeleteFile / DeleteDirectory
- ReadFile

2 Схема лабораторной установки

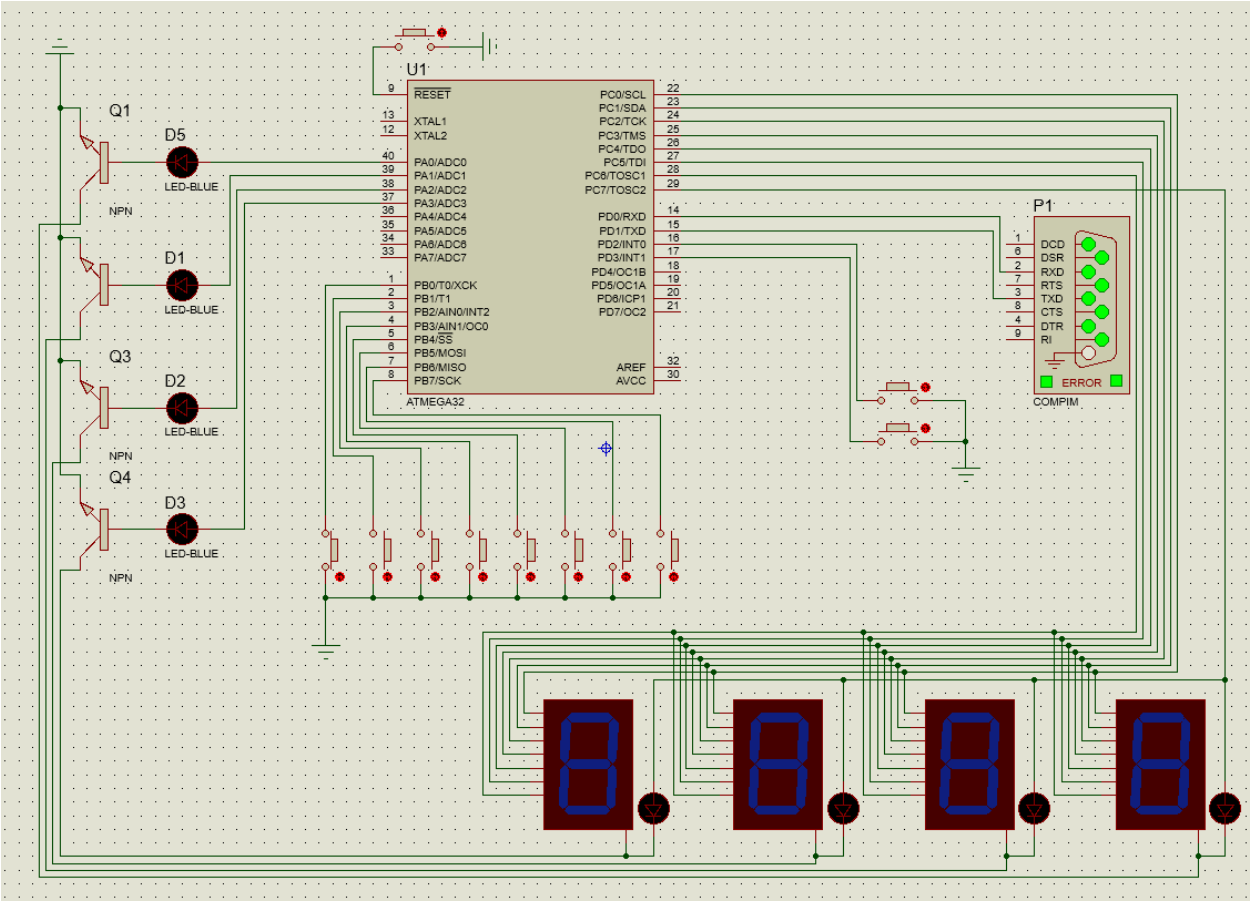


Рисунок 1 - Схема лабораторной установки

3 Форматы пересылаемых данных (в оба направления)

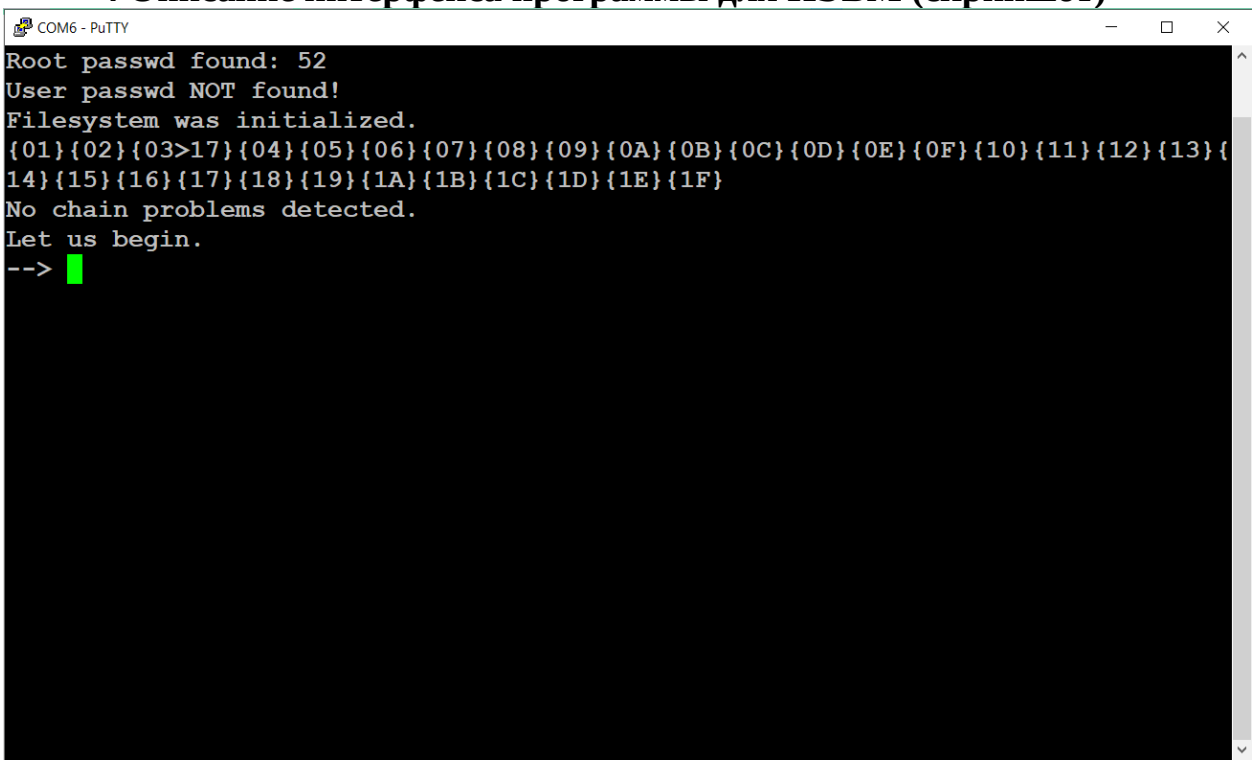
Передача команд осуществляется только в направлении ПК → МК:

Описание	Команда	1-й аргумент	Описание	2-й аргумент	Описание	3-й аргумент	Описание
Вывести список содержимого текущей директории	**ls**						
Вывести содержимое указанного файла	**cat**	**<NAME>**	Имя объекта в директории				
Сменить текущую директорию на указанную	**cd**	**<NAME>**	Имя объекта в директории				

Описание	Команда	1-й аргумент	Описание	2-й аргумент	Описание	3-й аргумент	Описание
Получить права указанного субъекта ФС	**login**	**<SUBJECT>**	Имя субъекта ФС: "**root**" / "**user**"				
Сбросить все имеющиеся права субъектов ФС	**logout**						
Сменить пароль субъекта ФС	**passwd**	**<SUBJECT>**	Имя субъекта ФС: "**root**" / "**user**"				
Отобразить информацию об объекте ФС	**info**	**<TYPE>**	Тип объекта ФС: директория – "**d**", файл – "**f**"	**<NAME>**	Имя объекта в директории		
Пометить объект ФС, как удалённый	**rm**	**<TYPE>**	Тип объекта ФС: директория – "**d**", файл – "**f**"	**<NAME>**	Имя объекта в директории		
Создать новый файл	**touch**	**<NAME>**	Имя нового файла	**<SUBJECT>**	Имя субъекта ФС - владельца: "**root**" / "**user**" / "**all**"	**<DATA>**	Строка, которая будет записана в файл
Создать новую директорию	**mkdir**	**<NAME>**	Имя нового файла	**<SUBJECT>**	Имя субъекта ФС - владельца: "**root**" /		

Описание	Команда	1-й аргумент	Описание	2-й аргумент	Описание	3-й аргумент	Описание
					"**user**" / "**all**"		
Проверка корректности массива указателей	**check**						

4 Описание интерфейса программы для ПЭВМ (скриншот)



МК имитирует терминал, для взаимодействия с которым нужно любое приложение для работы с COM портом.

5 Блок-схема алгоритма работы программы для МК

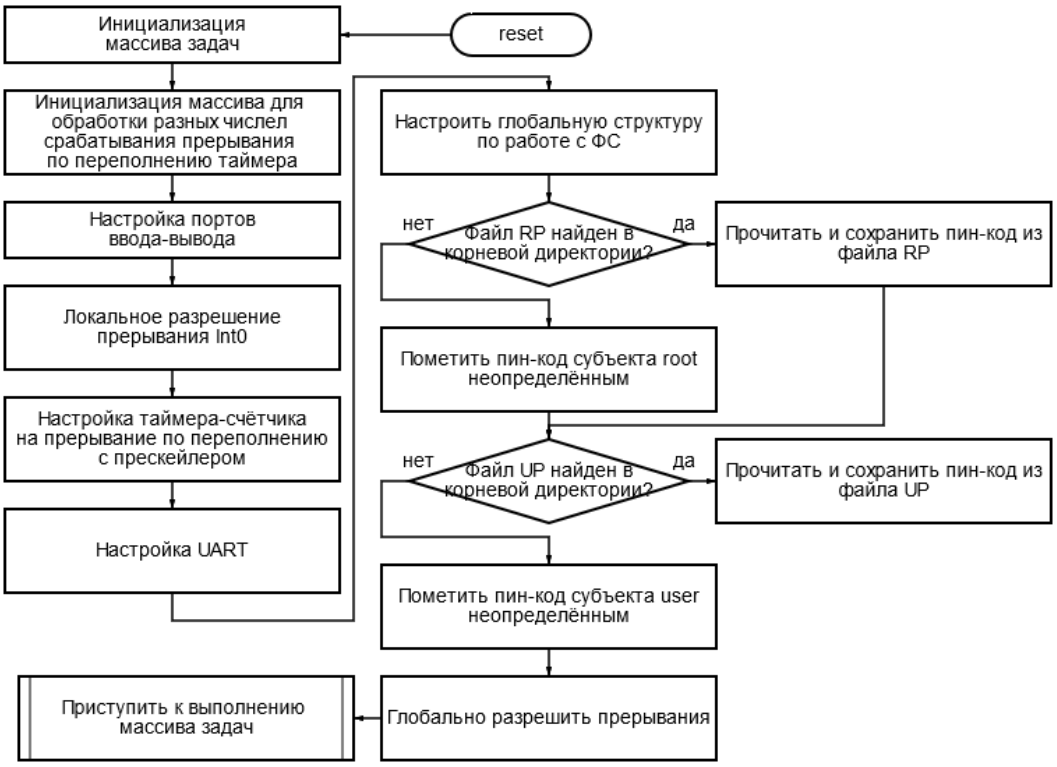


Рисунок 2 - Блок-схема инициализации основого цикла программы

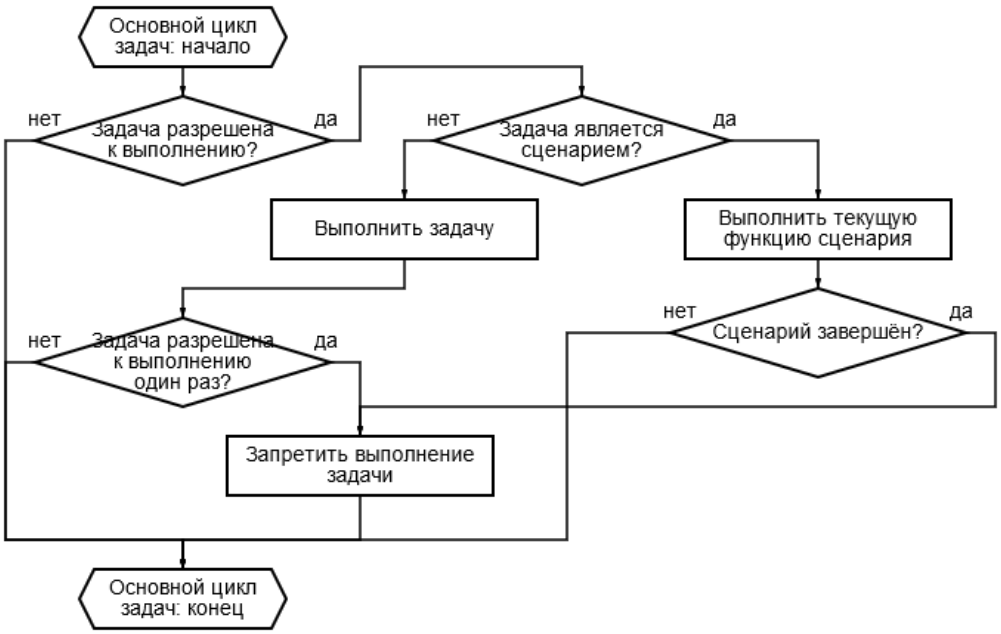


Рисунок 3 - Блок-схема исполнения основого цикла программы

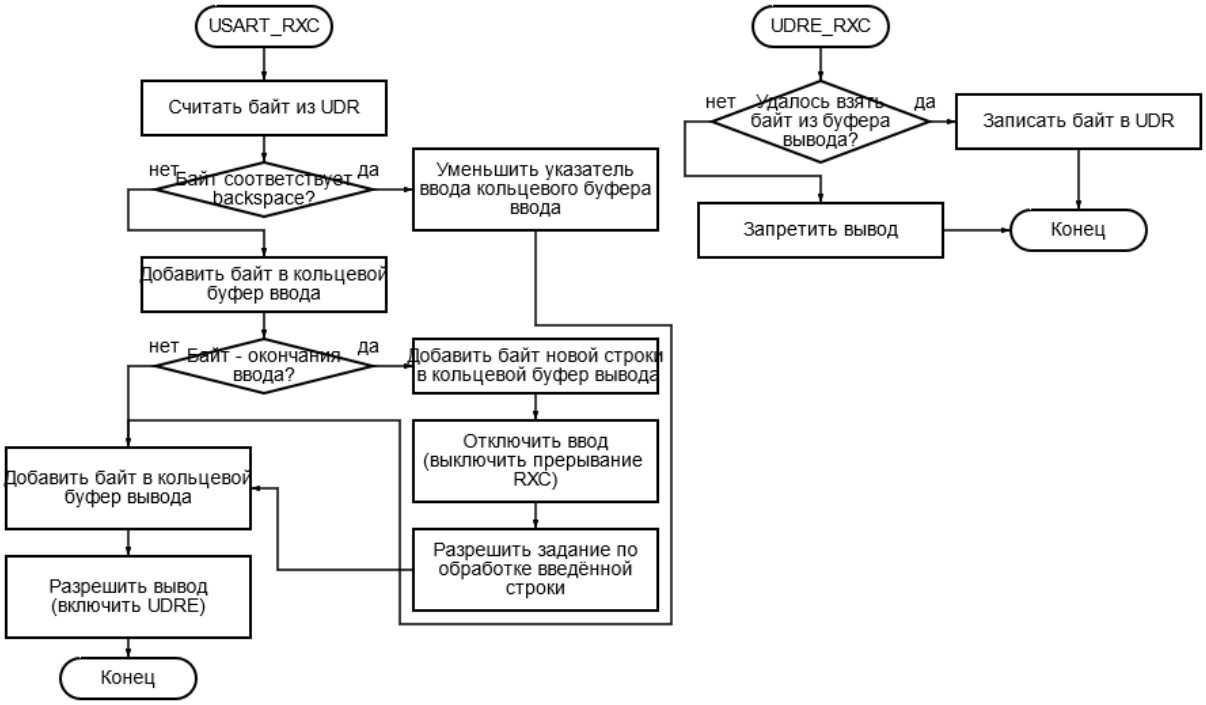


Рисунок 4 - Блок-схема обработки прерываний UART

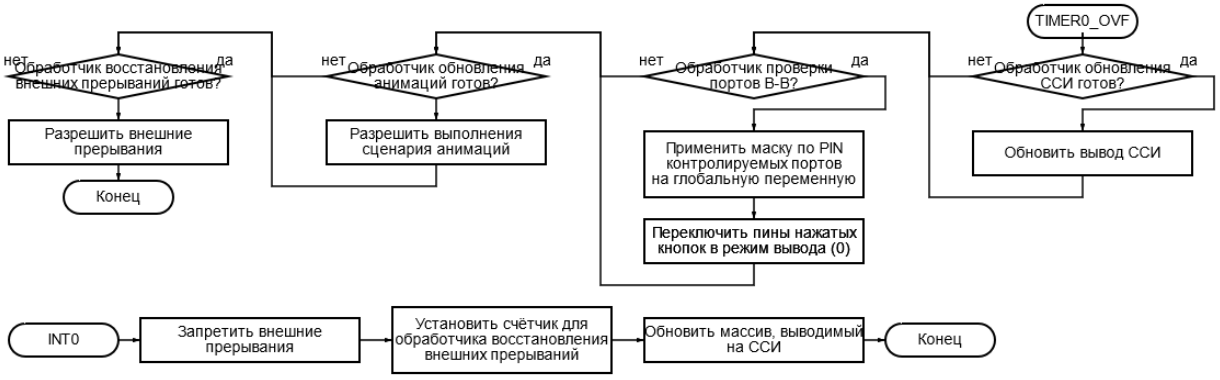


Рисунок 5 - Блок-схема обработчиков прерываний таймера и INT0

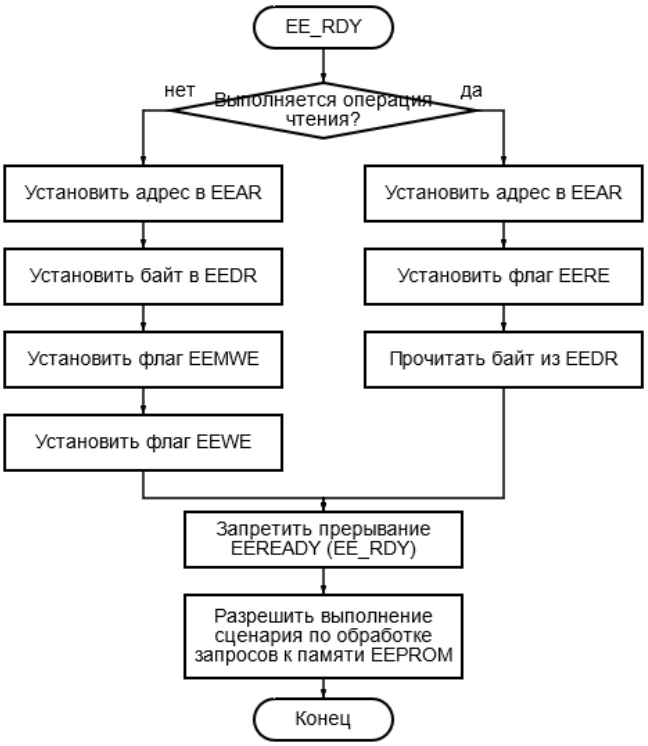


Рисунок 6 - Блок-схема обработки прерывания EEPROM READY

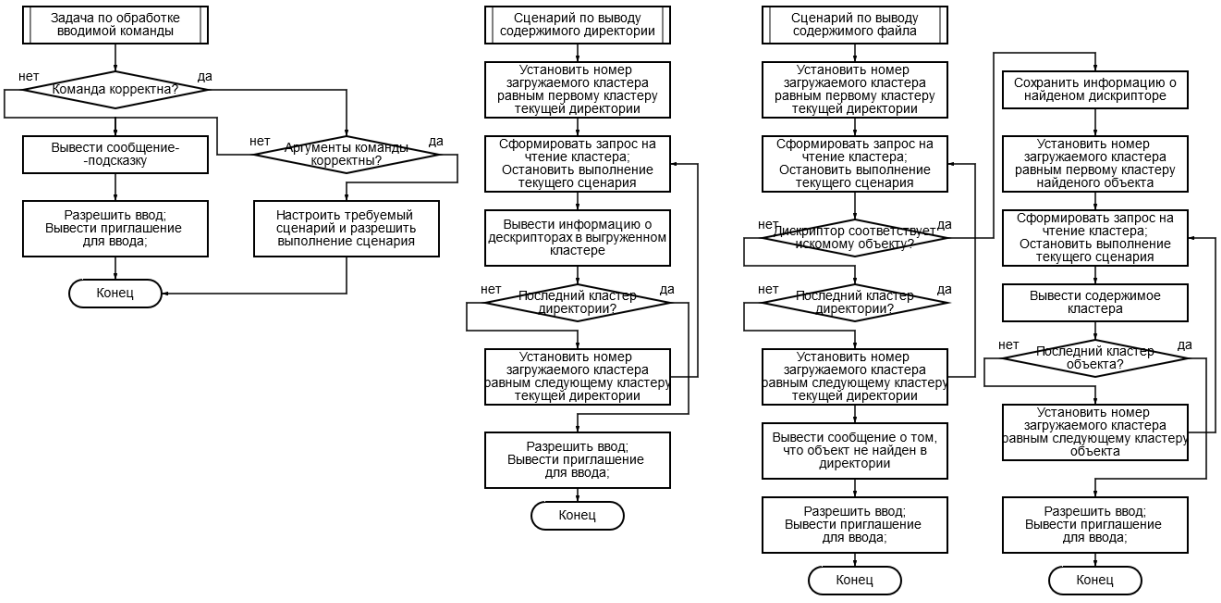


Рисунок 7 - Блок-схема задачи по обработке вводимой на ПК команды и сценариев по выводу содержимого объектов ФС

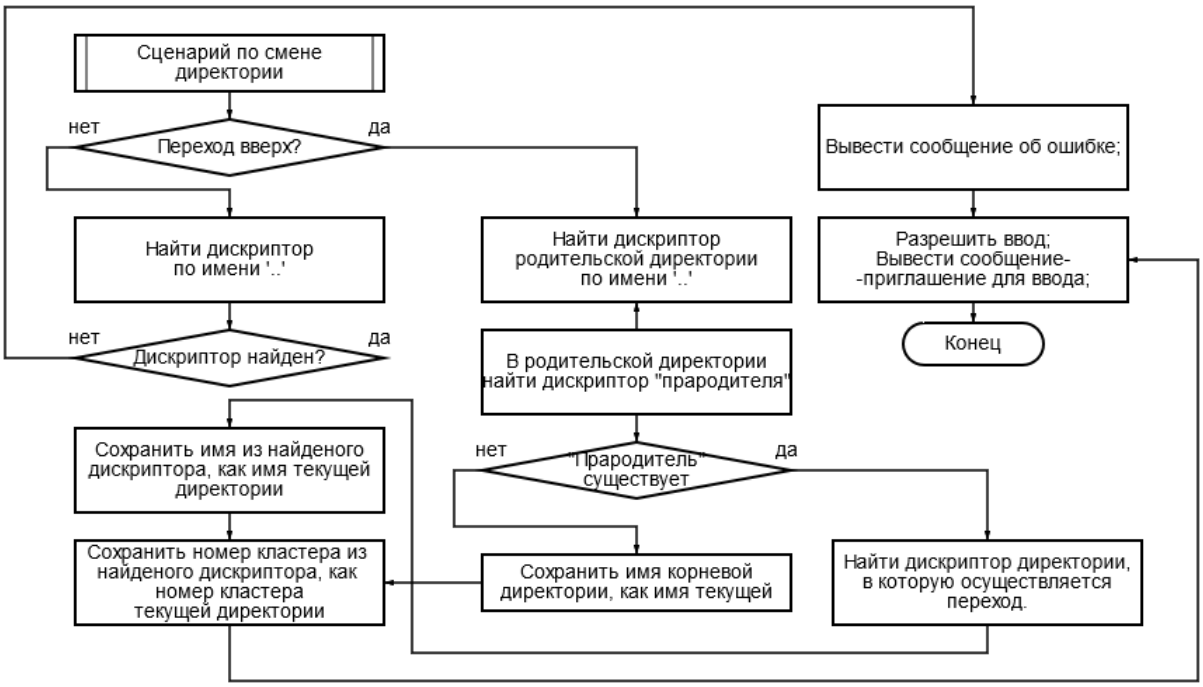


Рисунок 8 - Блок-схема сценария по смене текущей директории

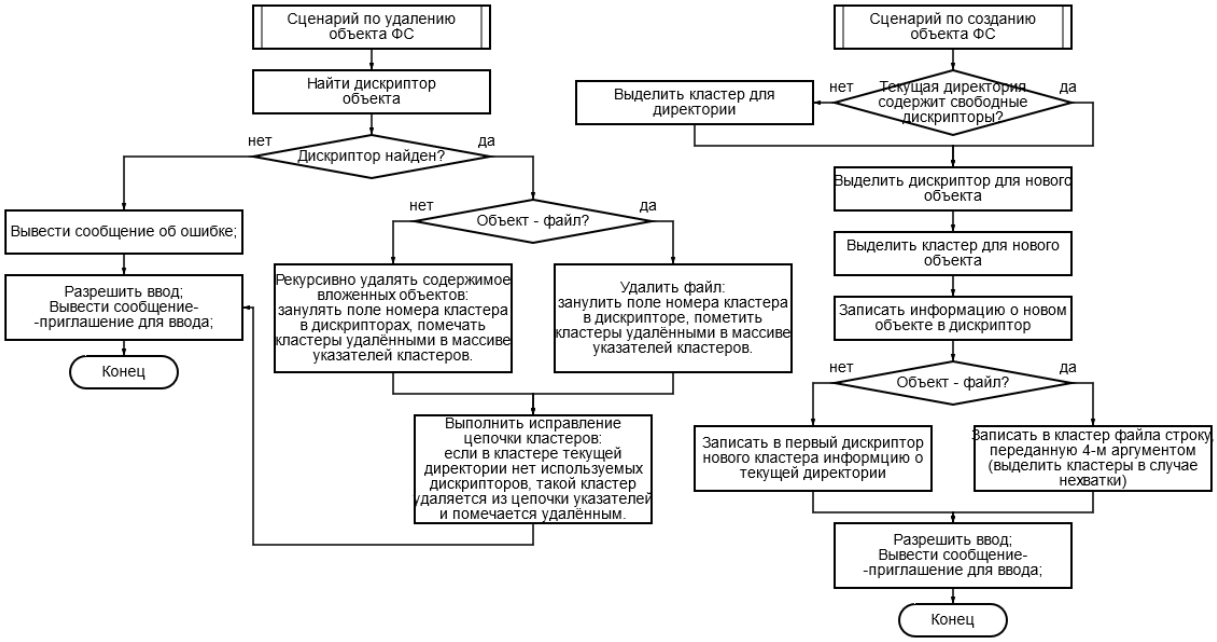


Рисунок 9 - Блок-схема сценариев по удалению и созданию объектов ФС

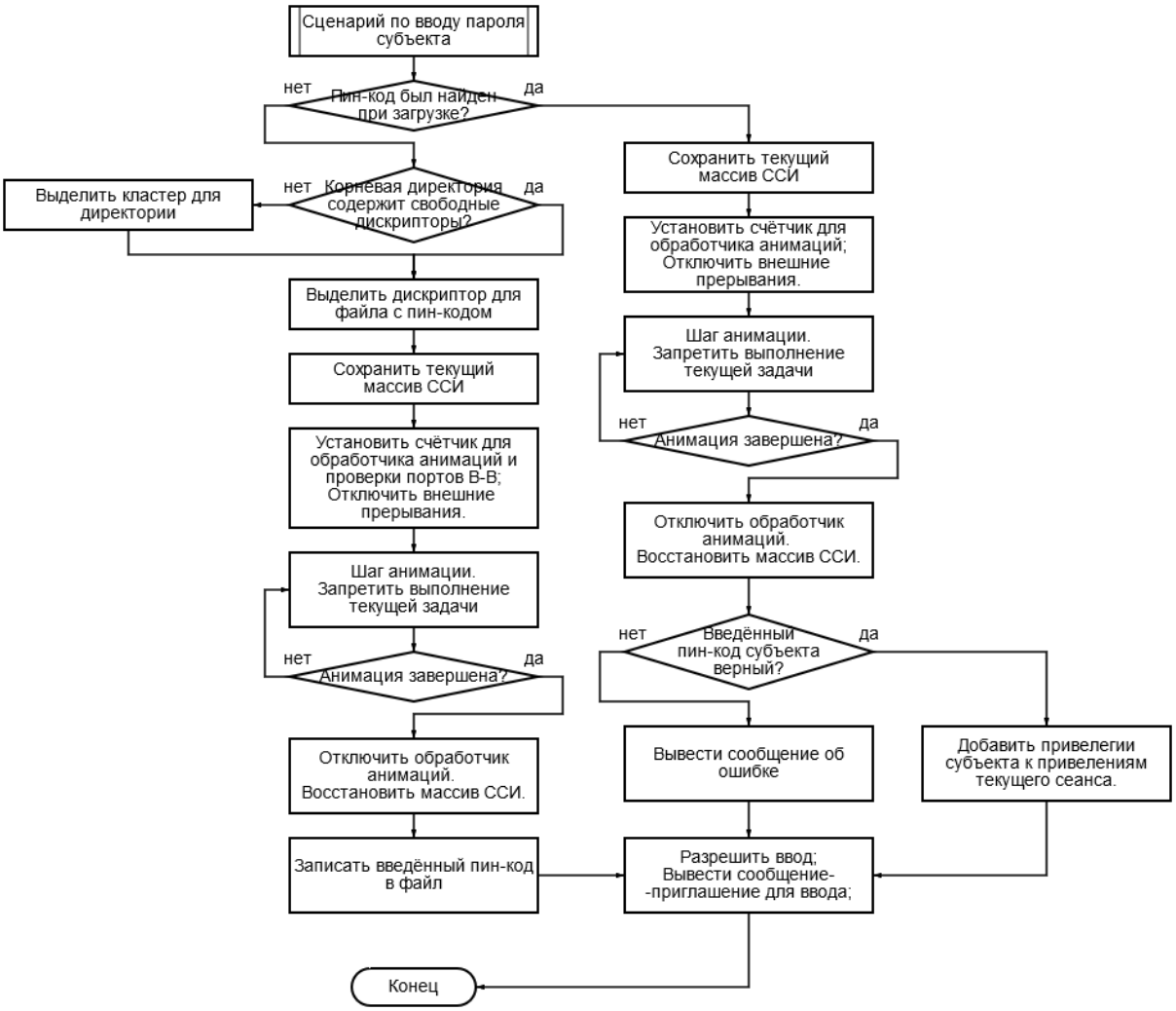


Рисунок 10 - Блок-схема сценария по вводу пин-кода субъекта

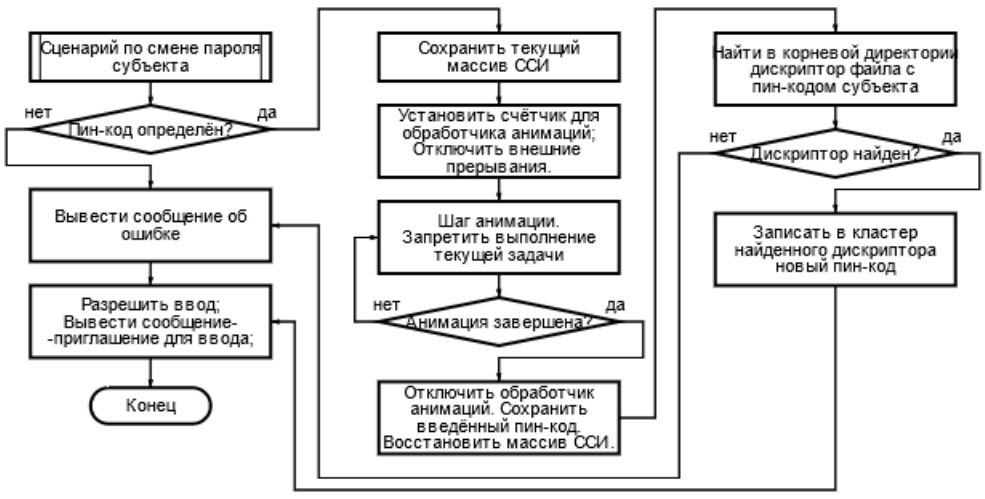


Рисунок 11 - Блок-схема сценария по смене пин-кода субъета

6 Блок-схема алгоритма работы программы для ПЭВМ

В рамках, реализацованного в этой работе, протокола общения ПК и МК через интерфейс UART требуется только ПО предназначенное для обычного взаимодействия с COM-портами ПК.

7 Результаты работы

В ходе выполнения лабораторной работы был разработан программно-аппаратный прототип внешнего хранилища данных на базе МК ATmega32. В файловой системе реализована мандатная система доступа. Объектами ФС являются файлы и директории. Доступ к объектам определяется привилегиями, которыми располагает текущая сессия. После инициализации ФС сессия имеет только общие привелегии, то есть имеет доступ ко всем объектам, принадлежность которых субъекту не определена.

Получение доступа к привелегиям субъектов осуществляется через проверку пин-кода, вводимого на PORTB.

Любое обращение к содержимому кластеров осуществляется только с помощью дискрипторов (кроме корневого каталога) для сравнения привелегий текущей сессии и привелегий, требуемых для работы с объектом. Объектами ФС, привелегиямии на изминение/создание (общими средставами работы с объектами ФС) которых не располагает ни один и субъектов, являются дисрипторы родительских директорий и файлы с пин-кодами в корневой директории.

8 Выводы по лабораторной работе

В ходе выполнения лабораторной работы был рассмотрен метод организации многозадачности на МК: сценарии. Данный метод показал свою эффективность в условиях, когда алгоритм состоит из подзадач, которые требуют ожидания завершения работы переферийного устройства, например, прочтения запрошеной области памяти EEPROM, или организации интерактивного взаимодействия с пользователем, например, проигрывание анимации на ССИ при вводе пин-кода на порте ввода-вывода. Одной из слабых сторон такого подхода является расположение последовательности подзадач в массиве, что усложняет ветвление при выполнение и расширение таких сценариев.

Так же был получен практический опыт организации простой ФС с мандатной моделью доступа и формулировании моделей работы с ней: структуры, хранящиеся в оперативной памяти, методы синхронизации вносимых изменений с данными в EEPROM.

Приложение 1

Комментированный листинг программы для МК на языке Си

Структура проекта имеет следующий вид:

- main.c
- lab/
  - lab.h
  - setup_hardware.c
  - setup_environment.c
  - setup_kernel.c
  - setup_interrupts.c
  - user_tasks.h
  - user_tasks.c
  - user_uart.c
- kernel/
  - _shared_macro.h
  - kernel.c
  - kernel.h
  - _buffer.c
  - _buffer.h
  - ssi.h
  - ssi.c
  - uart.h
  - uart.c
  - filesystem/
    - fs_configuration.h
    - fs_types.h
    - filesystem.h
    - memory.h
    - memory.c
    - fs_scripts.h
    - _fs_init.c
    - _fs_script__shared.c
    - _fs_script__cat.c
    - _fs_script__cd.c
    - _fs_script__create.c
    - _fs_script__info.c
    - _fs_script__ls.c
    - _fs_script__rm.c

main.c

```
/*
 * spbpu-4s-asvt-l11-new-kernel.c
 *
 * Created: 9/14/2021 11:35:04 PM
 * Author : mkentrru
 */

#include "lab/lab.h"



int main (void)
{
	setup_environment ();

	setup_kernel ();

	setup_hardware ();

	// Filesystem
	init_filesystem ();

	P__put__string (PSTR("Filesystem was initialized."), _true);

	get_predefined_configuration ();

	// ready to run...
	enable_interrupts (_true);


	if ( fs__check__cluster_chain () )
		P_locking_output__string (PSTR("Chain is broken!"), _true);
	else
		P_locking_output__string (PSTR("No chain problems detected."),
_true);

	P_locking_output__string (PSTR("Let us begin."), _true);

	locking_output__prompt ();


	run_kernel (K_TASK_COUNT);

}
```

lab/lab.h

```
/*
 * lab.h
 *
 * Created: 9/15/2021 12:50:29 AM
 *  Author: mkentrru
 */


#ifndef LAB_H_
#define LAB_H_

#include "../kernel/kernel.h"
```

```
#include "../kernel/filesystem/filesystem.h"

#include "../kernel/uart.h"
#include "../kernel/ssi.h"

#include "../kernel/_buffer.h"

#include "user_tasks.h"

#include <avr/pgmspace.h>

// kernel
#define HARDWARE_TASKS   (1) // MEM_TASK_ID - fs task

#define KERNEL_TASKS     (3) //:
#define t_input_handler              0 + HARDWARE_TASKS
      void input_handler_exec ();
#define fs_script_task               1 + HARDWARE_TASKS // fs scripts
#define inter_script_task            2  +  HARDWARE_TASKS  //  interaction
scripts

#define K_TASK_COUNT (HARDWARE_TASKS + KERNEL_TASKS)

//
#define TIMER0_HZ 488
#define HZTOC(a,t) (t/a)
#define K_INT_COUNT 1
#define timer                  0
extern struct ibind timer_binds[];
      #define timer_binds_count      4

            #define timer__ssi_cd                       0
            #define timer__ssi_cd_counter        3
            ibind_counter
      timer__ssi_cd_exec ();
            #define timer__ssi_cd_bind (timer_binds + timer__ssi_cd)

            #define timer__ioport_check                 1
            #define timer__ioport_check_counter  HZTOC(15, TIMER0_HZ)
            ibind_counter
      timer__ioport_check_exec ();
            #define      timer__ioport_check_bind      (timer_binds      +
timer__ioport_check)

            #define timer__animation             2
            #define timer__animation_counter     HZTOC(0.7, TIMER0_HZ)
            ibind_counter
      timer__animation__exec ();
            #define timer__animation_bind (timer_binds + timer__animation)

            #define timer__ext_int_cd                   3
            #define timer__ext_int_cd_counter    488
            ibind_counter
      timer__ext_int_cd__exec ();
            #define       timer__ext_int_cd_bind       (timer_binds       +
timer__ext_int_cd)

      void external_interrupts__cooldown ();

      #define enable_int0            GICR  setflag            _sf(INT0);
      #define disable_int0     GICR  dropflag    _df(INT0);

      #endif /* LAB_H_ */
```

lab/setup_hardware.c

```
/*
 * setup_hardware.c
 *
 * Created: 9/15/2021 12:57:23 AM
 *  Author: mkentrru
 */
#include "lab.h"

void init_buttons (){
	DDRD &= ~(1<<PD2);
	PORTD |= (1<<PD2);

	// DDRD     dropflag    _df(PD2);
	// PORTD    setflag           _sf(PD2);

	MCUCR setflag           _sf(ISC01); // from 1 to 0 is int
	GICR  setflag           _sf(INT0);
}


void init_timer0 (){
	// Timer0 // TIMER0_OVF_vect
	// Normal mode: TOV0 is set on TOP (0xFF)
	//WGM00 = 0;
	//WGM01 = 0;
	//TCCR0     dropflag    _df(WGM00) & _df(WGM01);
	// Compare Output Mode: OC0 is disconnected
	//COM01 = 0;
	//COM00 = 0;
	//TCCR0     dropflag    _df(COM01) & _df(COM00);
	// Clock Select:
	//CS02 = 1;
	//CS01 = 0;
	//CS00 = 0;
	// 010 - /8 prescaler
	// 011 - /64 pre => 488 HZ
	// 100 - /256 pre => 122 Hz
	// 101 - /1024 pre => 30 Hz
	TCCR0 setflag           _sf(CS01);
	TCCR0 setflag           _sf(CS00);


	// Interrupt mask
	//OCIE0 = 0; // Compare Match Interrupt Enable
	//TOIE0 = 1; // Overflow Interrupt Enable
	TIMSK setflag           _sf(TOIE0);
	//TIMSK     dropflag    _df(OCIE0);
}

void set_default_ports () {
	DDRA = 0xFF; PORTA = 0x00;
	DDRB = 0xFF; PORTB = 0x00;
	DDRC = 0xFF; PORTC = 0x00;
	DDRD = 0xFF; PORTD = 0x00;
}

void setup_hardware (){
	set_default_ports ();
	init_buttons ();
	init_default_ssi ();
	init_timer0 ();
	init_default_uart ();
}
```

lab/setup_environment.c

```
/*
 * setup_environment.c
 *
 * Created: 9/15/2021 12:52:28 AM
 *  Author: mkentrru
 */

#include "lab.h"


// Kernel

KERNEL_VAR task tasks [K_TASK_COUNT];

KERNEL_VAR ihandler ihandlers [K_INT_COUNT];
KERNEL_VAR struct ibind timer_binds  [timer_binds_count];
// UART
byte_t uart_input_buf    [uart_input_size];
byte_t uart_output_buf   [uart_output_size];
byte_t uart_error_buf    [uart_error_size];

_c_buffer uart_input;
_c_buffer uart_output;
_c_buffer uart_error;


// FS
_fs_context        fs_context;
_mem_quere         mem_quere;
_mem_task          mem_task; // not kernel task

script mem_script;

script fs__ls__script;
script fs__cat__script;
script fs__cd__script;
script fs__rm__script;
script fs__info__script;
script fs__rm__script;
script fs__touch__script;
script fs__mkdir__script;

script passwd_input__script;
script passwd_change__script;
script passwd_init__script;

// configure variables and etc
void setup_environment (){
      interrupts__last_state = _false;
      // UART
      _c_buffer__init (&uart_input, uart_input_buf, uart_input_size - 1);
      _c_buffer__init  (&uart_output,  uart_output_buf,  uart_output_size  -
1);
      _c_buffer__init (&uart_error, uart_error_buf, uart_error_size - 1);

      // Scripts
      mem_script__init ();
}


void get_predefined_configuration (){
```

```
	// SSI
	_ssi__default_configuration ();

	// EEPROM and FS
	stop_mem

	// UART
	enable_input
}
```

lab/setup_kernel.c

```
/*
 * setup_kernel.c
 *
 * Created: 9/15/2021 1:22:05 AM
 *  Author: mkentrru
 */

#include "lab.h"

void setup_kernel (){

	// to run mem tasks
	// init_task (MEM_TASK_ID, mem_script_task, TS_FLAGS_SELF_CONTROLED);
	init_task (MEM_TASK_ID, mem_script_task, TS_FLAGS_DISABLED);

	// default input
	init_task (t_input_handler,    input_handler_exec,
TS_FLAGS_ALLOWED_ONCE);

	// Filesystem
	// init_task (fs_script_task, _null, TS_FLAGS_SELF_CONTROLED);
	init_task (fs_script_task, _null, TS_FLAGS_DISABLED);

	// interactions: animations on passwd input and etc
	// init_task (inter_script_task, _null, TS_FLAGS_SELF_CONTROLED);
	init_task (inter_script_task, _null, TS_FLAGS_DISABLED);



	init_ihandler(timer, timer_binds, timer_binds_count);
		init_bind(timer_binds  +  timer__ssi_cd,  timer__ssi_cd_counter,
timer__ssi_cd_exec);
		init_bind(timer_binds   +   timer__ioport_check,   bind_disabled,
timer__ioport_check_exec);
		init_bind(timer_binds    +    timer__animation,    bind_disabled,
timer__animation__exec);
		init_bind(timer_binds    +    timer__ext_int_cd,    bind_disabled,
timer__ext_int_cd__exec);

}
```

lab/setup_interrupts.c

```
/*
 * setup_interrupts.c
 *
 * Created: 9/15/2021 1:27:44 AM
 *  Author: mkentrru
 */

#include "lab.h"
```

```
ISR(USART_RXC_vect){
	byte_t a = UDR;

	if (a == 0x08 || a == 0x7F) { // backspace
		_c_buffer__undo_input (_stdin);
	}
	else {
		_c_buffer__push(_stdin, a);

		if (a == _IEB) {
			_c_buffer__push (_stdout, _NLB);
			disable_input
			allow_task(t_input_handler);
		}
	}

	if(! _c_buffer__push(_stdout, a)){
		enable_output
	}

}

ISR(USART_UDRE_vect){
	byte_t a = 0;

	if (_c_buffer__pop(_stderr, & a) && _c_buffer__pop(_stdout, & a))
		disable_output
	else
		UDR = a;

}

ISR(TIMER0_OVF_vect){
	global_ihandler(timer);
}

ISR(INT0_vect){
	external_interrupts__cooldown ();
	fs__ssi__switch ();
	//locking_output__hex_byte (output_ssi_description[3]);
	//locking_output__hex_byte (output_ssi_description[2]);
	//locking_output__hex_byte (output_ssi_description[1]);
	//locking_output__hex_byte (output_ssi_description[0]);
	//locking_output__new_line ();

	// locking_output__hex_buffer (output_ssi_description, _SSI_SIZE);
}

ibind_counter timer__ext_int_cd__exec () {
	enable_int0
	return bind_disabled;
}


/*
	on interrupt i: // (EEPROM Ready Interrupt)
		if to_read:
			EEAR = a;
			EECR |= (1<<EERE);
			d = EEDR;
		else:
			EEAR = a;
			EEDR = d;
			EECR |= (1<<EEMWE);
			EECR |= (1<<EEWE);
```

```
            disable interrupt i

            allow task waiting
*/

ISR(EE_RDY_vect){

      if ( mem_quere.descr.to_read ) {
            EEAR = mem_quere.a;
            EECR |= (1<<EERE);
            mem_quere.d = EEDR;
      }
      else {
            EEAR = mem_quere.a;
            EEDR = mem_quere.d;
            EECR |= (1<<EEMWE);
            EECR |= (1<<EEWE);
      }
      stop_mem
      allow_task (mem_quere.waiting);
}
```

lab/user_tasks.h

```
/*
 * user_tasks.h
 *
 * Created: 9/21/2021 11:09:19 PM
 *  Author: mkentrru
 */


#ifndef USER_TASKS_H_
#define USER_TASKS_H_

typedef byte_t passwd_t;
extern passwd_t entered_passwd;
#endif /* USER_TASKS_H_ */
```

lab/user_tasks.c

```
/*
 * user_tasks.c
 *
 * Created: 9/21/2021 2:12:33 PM
 *  Author: mkentrru
 */

#include "lab.h"



ibind_counter timer__ssi_cd_exec () {
      _ssi_update ();
      return timer__ssi_cd_counter;
}

byte_t pin_b = 0x00;
#define passwd_input_pin PINB
#define passwd_input_pin__value pin_b

ibind_counter timer__ioport_check_exec () {
```

```
	passwd_input_pin__value &= passwd_input_pin;
	DDRB = ~passwd_input_pin__value;
	PORTB = passwd_input_pin__value;

	return timer__ioport_check_counter;
}


ibind_counter timer__animation__exec () {
	allow_task (inter_script_task);
	return timer__animation_counter;
}


#define passwd_input__animation_size _SSI_SIZE
byte_t passwd_input__animation_step = 0;

passwd_t entered_passwd = 0xFF;

byte_t ssi_animation_buffer [_SSI_SIZE];
#define ssi_animation_symbol 0x10


script_result passwd_input__entry () {

	passwd_input__animation_step = 0;
	passwd_input_pin__value = 0xFF;

	_ssi__store_description ();
	for ( byte_t a = 0; a < _SSI_SIZE; a++ ) {
		ssi_animation_buffer [a] = 0;
	}

	passwd_input_pin__value = 0xFF;
	DDRB = 0x00;
	PORTB = 0xFF;

	disable_int0

	put_string ("Enter passwd at PORTB.", _true);
	enable_bind (timer__ioport_check_bind, timer__ioport_check_counter);
	enable_bind (timer__animation_bind, timer__animation_counter);

	return script__next;
}

script_result passwd_input__animation_loop () {
	put_byte ('.');
	if ( passwd_input__animation_step < _SSI_SIZE) {
		ssi_animation_buffer [passwd_input__animation_step] =
		ssi_animation_symbol;

		_ssi__update_description    (symbols,    ssi_animation_buffer,
_SSI_SIZE, complex);
	}

	if (passwd_input__animation_step++ < passwd_input__animation_size) {
		disable_current_task ();
		return script__stay;
	}

	disable_bind (timer__ioport_check_bind);
	disable_bind (timer__animation_bind);

	return script__next;
```

```
}

script_result passwd_input__check () {
	entered_passwd = passwd_input_pin__value;
	new_line ();
	put_string ("You entered: ", _false);
	put_hex_byte (entered_passwd);
	new_line ();

	// login root
	if ( fs_context.access_control.current_try) {

		if ( ! fs_context.access_control.defined_root ) {
			// create RP

		}

		if ( fs_context.access_control.rp == entered_passwd ) {
			put_string ("Correct passwd.", _true);
			fs_context.access_control.root = _true;
			return script__next;
		}
	}
	// login user
	else
	if ( ! fs_context.access_control.current_try ) {

		if ( ! fs_context.access_control.defined_user ) {
			// create UP

		}

		if ( fs_context.access_control.up == entered_passwd ) {
			put_string ("Correct passwd.", _true);
			fs_context.access_control.user = _true;
			return script__next;
		}
	}

	put_string ("Wrong passwd.", _true);

	return script__next;
}

script_result passwd_input__outro () {
	DDRB = 0xFF;
	PORTB = 0x00;

	_ssi__restore_description ();
	put_prompt ();
	enable_int0

	return script__next;
}

#define passwd_input__script_size 4
script_result (* passwd_input_f [passwd_input__script_size]) (struct script
*)  = {
	passwd_input__entry,
	passwd_input__animation_loop,
	passwd_input__check,
	passwd_input__outro
};
```

```
	void passwd_input__script__init () {
		script__init   (&   passwd_input__script,   passwd_input__script_size,
_null, passwd_input_f );
	}

	void passwd_input__script_task () {
		script__step (& passwd_input__script);
	}

	script_result passwd_change__setup () {
		entered_passwd = passwd_input_pin__value;
		new_line ();
		put_string ("You entered: ", _false);
		put_hex_byte (entered_passwd);
		new_line ();

		if ( fs_context.access_control.current_try ) {
			fs_context.current_obj.iROOT = 1;
			fs_context.current_obj.nA = 'R';
		}
		else
		if ( ! fs_context.access_control.current_try ) {
			fs_context.current_obj.iUSER = 1;
			fs_context.current_obj.nA = 'U';
		}

		fs_context.current_obj.nB = 'P';
		fs_context.current_obj.TYPE = obj_type__file;


		fs_context.current_obj.KID = ROOT_KID;
		return script__next;
	}


	script_result passwd_change__write_changes () {

		fs_K__change
		(& fs_context.current_obj_buf, & fs_context.current_obj__memsync, 0,
		& entered_passwd, sizeof(passwd_t));

		return script__next;
	}

	script_result passwd_change__change_current () {
		if ( fs_context.access_control.current_try ) {
			fs_context.access_control.rp = entered_passwd;
		}
		else
		if ( ! fs_context.access_control.current_try ) {
			fs_context.access_control.up = entered_passwd;
		}

		P_locking_output__string   (PSTR("Passwd   successfully   changed."),
_true);

		return script__next;
	}

	script_result passwd_change__entry () {

		passwd_input__animation_step = 0;
		passwd_input_pin__value = 0xFF;

		_ssi__store_description ();
```

```
            for ( byte_t a = 0; a < _SSI_SIZE; a++ ) {
                  ssi_animation_buffer [a] = 0;
            }

            passwd_input_pin__value = 0xFF;
            DDRB = 0x00;
            PORTB = 0xFF;

            disable_int0

            put_string ("Enter passwd at PORTB.", _true);
            enable_bind (timer__ioport_check_bind, timer__ioport_check_counter);
            enable_bind (timer__animation_bind, timer__animation_counter);

            return script__next;
      }



      #define passwd_change__script_size 11
      script_result   (*   passwd_change_f   [passwd_change__script_size])   (struct
script *)  = {
            passwd_change__entry,                // 0: init input
            passwd_input__animation_loop,  // 1: input animation

            /*
                  1. Find FP_descr at ROOT_KID
                  2. Load FP_descr_KID
                  3. Change passwd at FP K
                  4. Sync K
            */
            passwd_change__setup,                // 2: store passwd and configure
searching

            fs__ls_load_kluster,                 // 3: load ROOT_KID Klaster
            fs__find__by_name,                   // 4: find FP at Klaster
            fs__ls_check_chain,                        //  5:  keep  searching  at
next Kl if not found
            fs__find__failure,                   // 6: if not found

            passwd_change__write_changes,  // 7: write new passwd
            fs__sync__current_K,                 // 8: sync FP Kl
            passwd_change__change_current, // 9: set new value at RAM
            passwd_input__outro                        // 10: outro
      };


      void passwd_change__script__init () {
            script__init  (&  passwd_change__script,  passwd_change__script_size,
_null, passwd_change_f );
      }

      void passwd_change__script_task () {
            script__step (& passwd_change__script);
      }

      KID source_folder_keeper = 0;
      script_result passwd_init__entry (script * s) {

            s->data = (void *) & entered_passwd;

            // login root
            if ( fs_context.access_control.current_try ) {
                  fs_context.current_obj.iROOT = 1;
                  fs_context.current_obj.iUSER = 0;
```

```
                fs_context.current_obj.nA = 'R';
        }
        else {
                fs_context.current_obj.iROOT = 0;
                fs_context.current_obj.iUSER = 1;
                fs_context.current_obj.nA = 'U';
        }

        fs_context.current_obj.nB = 'P';
        fs_context.current_obj.SIZE = 1;
        fs_context.current_obj.TYPE = obj_type__file;

        // to keep current dir

        //locking_output__string("Current dir: ", _false);
        //locking_output__hex_byte (fs_context.current_folder.KID);
        //locking_output__new_line ();

        source_folder_keeper = fs_context.current_folder.KID;
        fs_context.current_folder.KID = ROOT_KID;

        fs_context.descr_pointer = _null;
        return script__next;
}

script_result passwd_init__obj_exists () {
        P_locking_output__string(PSTR("Subject   passwd   file   exists!   But
how?.."), _true);
        locking_output__prompt ();
        return script__exit;
}

script_result passwd_init__input_outro () {
        entered_passwd = passwd_input_pin__value;
        locking_output__new_line ();
        P_locking_output__string (PSTR("You entered: "), _false);
        locking_output__hex_byte (entered_passwd);
        locking_output__new_line ();

        DDRB = 0xFF;
        PORTB = 0x00;
        _ssi__restore_description ();

        enable_int0

        return script__next;
}

script_result passwd_init__outro () {
        if ( fs_context.access_control.current_try ) {
                fs_context.access_control.rp = entered_passwd;
                fs_context.access_control.root = _true;
                fs_context.access_control.defined_root = _true;
        }
        else {
                fs_context.access_control.up = entered_passwd;
                fs_context.access_control.user = _true;
                fs_context.access_control.defined_user = _true;
        }

        fs_context.current_folder.KID = source_folder_keeper;

        //locking_output__string("Current dir: ", _false);
        //locking_output__hex_byte (fs_context.current_folder.KID);
        //locking_output__new_line ();
```

```
            locking_output__prompt ();
            return script__exit;
      }

      #define passwd_init__script_size 21
      script_result (* passwd_init_f [passwd_init__script_size]) (struct script
*)  = {
            passwd_init__entry,

            fs__ls_entry,                                 // 1: check access
            fs__ls_load_kluster,                  // 2: load K
            fs__create__check_same,               // 3: if obj is same: +3; else:
+1;
            fs__ls_check_chain,                           // 4: if chain ended: +1;
else: -2;

            fs__create__build_new_descr,    // 5: get free obj_descr and allocate
K for Obj (+2 to skip error)
            passwd_init__obj_exists,        // 6: if passwd file exists

            // enter new passwd
            passwd_input__entry,                  // 7: init input animation
            passwd_input__animation_loop,   // 8: animation loop step
            passwd_init__input_outro,             // 9: animation end

            // write new data to new file
            fs__ls_load_kluster,                  //  10:  load  K  witch  contains
free obj descriptor (dir)
            fs__create__write_new_descr,    // 11: write new descr to current dir
            fs__sync__intro,                      // 12: sync intro
            fs__sync__current_K,                  // 13: sync Dir

            fs__touch__write_data__setup,   // 14: get ready to write data
            fs__ls_load_kluster,                  // 15: load current K
            fs__touch__write_data,                // 16: write data to current K
            fs__sync__current_K,                  // 17: sync changes with memory
            fs__touch__write__check,        // 18: check if any data left
            fs__sync__intro,                      // 19: sync intro

            passwd_init__outro                    //  20:  store  new  passwd  at
context and set logged in
      };


      void passwd_init__script__init () {
            script__init (& passwd_init__script, passwd_init__script_size, _null,
passwd_init_f );
      }

      void passwd_init__script_task () {
            script__step (& passwd_init__script);
      }


      void external_interrupts__cooldown () {
            disable_int0
            enable_bind (timer__ext_int_cd_bind, timer__ext_int_cd_counter);
      }
```

lab/user_uart.c

```
      /*
       * user_uart.c
       *
```

```
 * Created: 9/15/2021 1:35:25 AM
 *  Author: mkentrru
 */

#include "lab.h"


#define CMD_NONE   0x00
#define CMD_WRONG  0x01
#define CMD_HELP   0x02
#define CMD_LOGIN  0x03
void cmd__login ();
#define CMD_LS           0x04
void cmd__ls ();
#define CMD_CAT          0x05
void cmd__cat ();
#define CMD_CD           0x06
void cmd__cd ();
#define CMD_LOGOUT 0x07
void cmd_logout ();
#define CMD_RM           0x08
void cmd__rm ();
#define CMD_INTRO  0x09
void cmd__intro ();
#define CMD_INFO   0x0A
void cmd__info ();
#define CMD_TOUCH  0x0B
void cmd__touch ();
#define CMD_MKDIR  0x0C
void cmd__mkdir ();
#define CMD_PASSWD 0x0D
void cmd__passwd ();
#define CMD_CHCHECK      0x0E
void cmd__chaincheck ();

struct _args args;


char * string_cut (char * s, char ** pos) {
	// skip \' ' at beginning
	while ( * s == ' ' ) {
		s++;
	}
	char * res = s;
	* pos = s;

	while ( valid_cmd_symbol( * s ) )
	s++;

	if (s == * pos) {
		* pos = _null;
		res = _null;
	}
	else {
		* pos = s;
	}

	return res;
}

exit_status substring_compare (char * a, char * b) {
	while ( valid_cmd_symbol(* a) || valid_cmd_symbol(* b) ) {
		if ( * a != * b ) return exit_failure;
		a++; b++;
	}
```

```
	return exit_success;
}

byte_t cover_input_string (char * s, struct _args * args ){
	args->argc = 0;

	char * a = s;
	char * r = s;
	while ( (r = string_cut (s, & a)) != _null && args->argc < MAX_ARGCOUNT ) {
		args->argv[args->argc++] = r;
		s = a;
	}

	if ( args->argc == 0 ) {
		return CMD_NONE;
	}

	if ( substring_compare(args->argv[0], "help") == exit_success )
return CMD_HELP;
	if ( substring_compare(args->argv[0], "ls") == exit_success )
return CMD_LS;
	if ( substring_compare(args->argv[0], "cat") == exit_success )
return CMD_CAT;
	if ( substring_compare(args->argv[0], "cd") == exit_success )
return CMD_CD;
	if ( substring_compare(args->argv[0], "login") == exit_success )
return CMD_LOGIN;
	if ( substring_compare(args->argv[0], "logout") == exit_success )
return CMD_LOGOUT;
	if ( substring_compare(args->argv[0], "passwd") == exit_success )
return CMD_PASSWD;
	if ( substring_compare(args->argv[0], "info") == exit_success )
return CMD_INFO;
	if ( substring_compare(args->argv[0], "rm") == exit_success )
return CMD_RM;
	if ( substring_compare(args->argv[0], "touch") == exit_success )
return CMD_TOUCH;
	if ( substring_compare(args->argv[0], "intro") == exit_success )
return CMD_INTRO;
	if ( substring_compare(args->argv[0], "mkdir") == exit_success )
return CMD_MKDIR;
	if ( substring_compare(args->argv[0], "check") == exit_success )
return CMD_CHCHECK;


	return CMD_WRONG;
}

exit_status check_username (struct _args * args) {
	if ( substring_compare(args->argv[1], "root") == exit_success ) {
		fs_context.access_control.current_try = 1;
		return exit_success;
	}
	else if ( substring_compare(args->argv[1], "user") == exit_success ) {
		fs_context.access_control.current_try = 0;
		return exit_success;
	}
	put_string("Wrong Subject name.", _true);
	return exit_failure;
}

exit_status check_objtype (struct _args * args) {
	if ( substring_compare(args->argv[1], "d") == exit_success ) {
```

```
                fs_context.current_obj.TYPE = obj_type__dir;
                return exit_success;
        }
        else if ( substring_compare(args->argv[1], "f") == exit_success ) {
                fs_context.current_obj.TYPE = obj_type__file;
                return exit_success;
        }

        return exit_failure;
}

const char cmd_error__wrong_command [] PROGMEM = "Try \'help\' for some
help.";
const char cmd_error__wrong_args [] PROGMEM = "Wrong arguments.
Try \'help\'.";
const char access_denied [] PROGMEM = "Access denied";
const char wrong_name [] PROGMEM = "Wrong name.";

#define none 0
_bool correct_argc (byte_t m, byte_t l) {
        if ( l == none ) {
                if ( args.argc == m )
                        return _true;
        }
        else {
                if ( args.argc >= m && args.argc <= l )
                        return _true;
        }
        P_locking_output__string (cmd_error__wrong_args, _true);
        locking_output__prompt ();
        return _false;
}

void input_handler_exec (){ // does not check if buffer is empty

        char command_buffer [uart_input_size + 1];


        //while ( !(_stdin_empty) ){

                _c_buffer_pos      end_pos = 0;

                if   (   _c_buffer__parse_string   (_stdin,   '\0',   &   end_pos,
command_buffer) ) {
                        // TODO: error message
                        _c_buffer__fix (_stdin);
                }

                switch ( cover_input_string(command_buffer, &args) ) {
                        case CMD_NONE:
                                put_prompt ();
                                break;

                        case CMD_WRONG:
                                P__put__string (cmd_error__wrong_command, _true);
                                put_prompt ();
                                break;

                        case CMD_HELP:
                                //   put_string("Ask   me   about   the   commands.",
_true);
                                put_prompt ();
                                break;

                        case CMD_LS:
```

```
                        if ( correct_argc (1, none) )
                              cmd__ls ();
                        break;

                  case CMD_CAT:
                        if ( correct_argc (2, none) )
                              cmd__cat ();
                        break;

                  case CMD_LOGIN:
                        if ( correct_argc (2, none) )
                              cmd__login ();
                        break;

                  case CMD_LOGOUT:
                        if ( correct_argc (1, none) )
                              cmd_logout ();
                        break;

                  case CMD_CD:
                        if ( correct_argc (2, none) )
                              cmd__cd ();
                        break;

                  case CMD_RM:
                        if ( correct_argc (3, none) )
                              cmd__rm ();

                        break;

                  case CMD_INFO:
                        if ( correct_argc (3, none) )
                              cmd__info ();

                        break;

                  case CMD_INTRO:
                        cmd__intro ();
                        break;

                  case CMD_TOUCH:
                        if ( correct_argc (3, 4) )
                              cmd__touch ();
                        break;
                  case CMD_MKDIR:
                        if ( correct_argc (3, none) )
                              cmd__mkdir ();
                        break;
                  case CMD_PASSWD:
                        if ( correct_argc (2, none) )
                              cmd__passwd ();
                        break;
                  case CMD_CHCHECK:
                        if ( correct_argc (1, none) )
                              cmd__chaincheck ();
                        break;

                  default:
                        // put_string ("Command is not covered yet. wtf",
_true);
                        put_prompt ();
                        break;
            }
```

```
                //_c_buffer__jump (_stdin, end_pos);
                //_c_buffer__next (_stdin); // to skip IEB
                _c_buffer__fix (_stdin);
        //}

        // enable_input
    }


    void cmd__ls () {
        P__put__string (PSTR("Dir: "), _false);
        put_byte (fs_context.current_folder.nA);
        put_byte (fs_context.current_folder.nB);
        new_line ();

        fs__ls__script__init ();
        allow_script (fs_script_task, fs__ls__script_task);
    }

    void cmd__cat () {

        fs_context.current_obj.nA = args.argv[1][0];
        fs_context.current_obj.nB = args.argv[1][1];
        fs_context.current_obj.TYPE = obj_type__file;

        P__put__string(PSTR("Show data of: "), _false);
        put_byte (fs_context.current_obj.nA);
        put_byte (fs_context.current_obj.nB);
        new_line ();

        fs__cat__script__init ();
        allow_script (fs_script_task, fs__cat__script_task);
    }

    void cmd__login () {
        if ( check_username (& args) ) {
            locking_output__prompt ();
        }
        else {
            // login root
            if     (     (fs_context.access_control.current_try     &&     !
fs_context.access_control.defined_root) ||
                (!       fs_context.access_control.current_try       &&       !
fs_context.access_control.defined_user) ) {
                    P_locking_output__string    (PSTR("Subject    passwd    is
undefined. Enter new passwd."), _true);

                    passwd_init__script__init ();
                    allow_script                            (inter_script_task,
passwd_init__script_task);
                }
                else {
                    passwd_input__script__init ();
                    allow_script                            (inter_script_task,
passwd_input__script_task);
                }

        }
    }

    void cmd_logout () {
        fs_context.access_control.root = _false;
        fs_context.access_control.user = _false;
        P__put__string (PSTR("Logged out."), _true);
        put_prompt ();
```

```
}

void cmd__cd () {
	fs_context.current_obj.nA = args.argv[1][0];
	fs_context.current_obj.nB = args.argv[1][1];
	fs_context.current_obj.TYPE = obj_type__dir;

	P__put__string (PSTR("Change Dir to: "), _false);
	put_byte (fs_context.current_obj.nA);
	put_byte (fs_context.current_obj.nB);

	fs__cd__script__init ();
	allow_script (fs_script_task, fs__cd__script_task);
}

void cmd__rm () {
	if ( check_objtype (& args) ) {
		P__put__string (PSTR("Wrong Obj type."), _true);
		put_prompt ();
	}
	else {
		fs_context.current_obj.nA = args.argv[2][0];
		fs_context.current_obj.nB = args.argv[2][1];

		P__put__string(PSTR("Remove: "), _false);
		put_byte (fs_context.current_obj.nA);
		put_byte (fs_context.current_obj.nB);
		new_line ();

		fs__rm__script__init ();
		allow_script (fs_script_task, fs__rm__script_task);
	}
}

void cmd__info () {
	if ( check_objtype (& args) ) {
		P__put__string (PSTR("Wrong Obj type."), _true);
		put_prompt ();
	}
	else {
		fs_context.current_obj.nA = args.argv[2][0];
		fs_context.current_obj.nB = args.argv[2][1];

		P__put__string (PSTR("Info:"), _true);

		fs__info__script__init ();
		allow_script (fs_script_task, fs__info__script_task);
	}
}


void cmd__intro () {
	locking_output__hex_buffer ((byte_t *) fs_context.intro.row, K_SIZE);
	mem_busy__read_area (ktoa(INTRO_KID), K_SIZE, (byte_t *) &
fs_context.intro.row);
	locking_output__hex_buffer ((byte_t *) fs_context.intro.row, K_SIZE);

	locking_output__hex_buffer ((byte_t *) & fs_context,
sizeof(_fs_context));

	locking_output__prompt ();
}


_bool correct_obj_rights () {
```

```
	char * s = args.argv [2];
	if ( substring_compare(s, "root") == exit_success ) {
		P_locking_output__string (PSTR("As Root Obj."), _true);
		if ( ! fs_context.access_control.root ) return _false;
		fs_context.current_obj.iROOT = 1;
		fs_context.current_obj.iUSER = 0;
		return _true;
	}
	else
	if ( substring_compare(s, "user") == exit_success ) {
		P_locking_output__string (PSTR("As User Obj."), _true);
		if ( ! fs_context.access_control.user ) return _false;
		fs_context.current_obj.iROOT = 0;
		fs_context.current_obj.iUSER = 1;
		return _true;
	}
	else
	if ( substring_compare(s, "all") == exit_success ) {
		P_locking_output__string (PSTR("As shared Obj."), _true);
		fs_context.current_obj.iROOT = 0;
		fs_context.current_obj.iUSER = 0;
		return _true;
	}
	P_locking_output__string (PSTR("Unknown user."), _true);
	return _false;
}


void cmd__touch () {
	char * d = _null;

	if ( ! valid_cmd_symbol(args.argv[1][0]) ||
	! valid_cmd_symbol(args.argv[1][1]) ) {
		P_locking_output__string (wrong_name, _true);
		locking_output__prompt ();
		return;
	}

	fs_context.current_obj.nA = args.argv[1][0];
	fs_context.current_obj.nB = args.argv[1][1];
	fs_context.current_obj.TYPE = obj_type__file;

	P_locking_output__string(PSTR("Create file: "), _false);
	locking_output__byte (fs_context.current_obj.nA);
	locking_output__byte (fs_context.current_obj.nB);
	locking_output__new_line ();

	if ( fs_obj_descr__shared (& fs_context.current_obj) || !
correct_obj_rights () ) {
		P_locking_output__string (access_denied, _true);
		locking_output__prompt ();
		return;
	}

	if ( args.argc == 4 ) {
		d = args.argv [3];
	}

	fs__touch__script__init (d);
	allow_script (fs_script_task, fs__touch__script_task);
}


void cmd__mkdir () {
	if ( ! valid_cmd_symbol(args.argv[1][0]) ||
```

```
            ! valid_cmd_symbol(args.argv[1][1]) ) {
                  P_locking_output__string (wrong_name, _true);
                  locking_output__prompt ();
                  return;
            }

            fs_context.current_obj.nA = args.argv[1][0];
            fs_context.current_obj.nB = args.argv[1][1];
            fs_context.current_obj.TYPE = obj_type__dir;

            P_locking_output__string(PSTR("Create file: "), _false);
            locking_output__byte (fs_context.current_obj.nA);
            locking_output__byte (fs_context.current_obj.nB);
            locking_output__new_line ();

            if   (   fs_obj_descr__shared   (&   fs_context.current_obj)   ||   !
correct_obj_rights () ) {
                  P_locking_output__string (access_denied, _true);
                  locking_output__prompt ();
                  return;
            }

            fs__mkdir__script__init ();
            allow_script (fs_script_task, fs__mkdir__script_task);
      }


      _bool correct_subject_rights () {
            char * s = args.argv [1];
            if ( substring_compare(s, "root") == exit_success ) {
                  P_locking_output__string (PSTR("Root passwd."), _true);
                  if ( ! fs_context.access_control.defined_root ) {
                        P_locking_output__string(PSTR("Root passwd is undefined.
Login first."), _true);
                        return _false;
                  }
                  if ( ! fs_context.access_control.root ) return _false;
                  fs_context.access_control.current_try = 1;
                  return _true;
            }
            else
            if ( substring_compare(s, "user") == exit_success ) {
                  P_locking_output__string (PSTR("USER passwd."), _true);
                  if ( ! fs_context.access_control.defined_user ) {
                        P_locking_output__string(PSTR("User passwd is undefined.
Login first."), _true);
                        return _false;
                  }
                  if ( ! fs_context.access_control.user ) return _false;
                  fs_context.access_control.current_try = 0;
                  return _true;
            }

            P_locking_output__string (PSTR("Unknown user."), _true);
            return _false;
      }


      void cmd__passwd () {

            P_locking_output__string(PSTR("Change "), _false);
            if ( ! correct_subject_rights () ) {
                  P_locking_output__string(PSTR("Request denied."), _true);
                  locking_output__prompt ();
                  return;
```

```
	}

	passwd_change__script__init ();
	allow_script (inter_script_task, passwd_change__script_task);
}


void cmd__chaincheck () {
	if ( fs__check__cluster_chain () ) {
		P_locking_output__string (PSTR("Chain is broken!"), _true);
	}
	else {
		P_locking_output__string (PSTR("No chain problems detected."),
_true);
	}

	locking_output__prompt ();
}
```

kernel/_shared_macro.h

```
/*
 * _shared_macro.h
 *
 * Created: 9/6/2021 6:35:11 PM
 *  Author: mkentrru
 */


#ifndef _SHARED_MACRO_H_
#define _SHARED_MACRO_H_


#include <stdint.h>

typedef uint8_t byte_t;
typedef uint8_t breg;

typedef uint16_t word_t;

#define _sf(a)           (1 << a)
#define _df(a)           (~(1 << a))
#define _dm(a, m)  (a & ~m)

#define setflag          |=
#define dropflag   &=

typedef uint8_t _bool;
#define _true      (_bool) 0x01
#define _false     (_bool) 0x00

#define loop while(1);

typedef uint8_t exit_status;

#define exit_success     0
#define exit_failure     1

#define _null 0

#define valid_cmd_symbol(c) ((c >= 'A' && c <= 'Z') ||\
 (c >= 'a' && c <= 'z') ||\
 (c >= '0' && c <= '9') || (c == '.'))

#endif /* _SHARED_MACRO_H_ */
```

kernel/kernel.h

```
/*
 * kernel.h
 *
 * Created: 9/6/2021 6:33:59 PM
 *  Author: mkentrru
 */


#ifndef KERNEL_H_
#define KERNEL_H_


#include "_shared_macro.h"
#include <avr/interrupt.h>
#include <avr/io.h>



//typedef enum{
	//MS_STOPPED,
	//MS_COUNTING
//} macro_state;



	// Tasks structures
#define TS_FLAGS_DISABLED			0x00
#define TS_FLAGS_ALLOWED		_sf(0)
#define TS_FLAGS_ALLOWED_ONCE	_sf(1)
#define TS_FLAGS_SELF_CONTROLED		_sf(2)
typedef byte_t task_state;

typedef void task_exec ();

typedef byte_t task_id_t;
typedef task_id_t task_count_t;
typedef struct {
	task_state s;
	task_exec * f;
} task;

// #define allow_task(id) (tasks + id)->s |= TS_FLAGS_ALLOWED;
extern task * current_task;
	// Interruption
typedef byte_t ihandler_id_t;

typedef uint16_t ibind_counter;

typedef byte_t ihandler_size;

typedef ibind_counter (ibind_handler) (void);
#define bind_disabled 0

struct ibind{
	ibind_counter c;   // count interruptions before handling
	ibind_handler * h; // local handler
};

typedef struct{
	struct ibind * b;  // binds
	ihandler_size c;   // count of binds
} ihandler;
```

```
#define instant_bind ((ibind_counter) 0x01)
#define disabled_bind ((ibind_counter) 0x00)

// Scripts

typedef byte_t script_result;
#define script__next     1
#define script__prev     -1
#define script__stay     0
#define script__exit     0xFF
#define script__error    script__exit

typedef byte_t script_id_t;

typedef struct script{

      script_id_t size;
      script_id_t current;

      void * data;

      script_result (** f) (struct script *);
} script;

void script__init (script * s, script_id_t size, void * data, script_result
(** f) (struct script *));
void script__step (script * s);



      // Kernel functions

void init_task (task_id_t, task_exec, task_state);
void allow_task (task_id_t id);
void allow_script (task_id_t id, void (* script_task) ());

void init_bind(struct ibind *, ibind_counter, ibind_handler *);
void init_ihandler(ihandler_id_t, struct ibind *, ihandler_size);

void local_ihandler(ihandler *);

extern _bool interrupts__last_state;
void enable_interrupts (_bool force);
void disable_interrupts ();

void enable_bind (struct ibind * b, ibind_counter c);
void disable_bind (struct ibind * b);

void lock_ext (struct ibind *, ibind_counter);
void unlock_ext ();

void run_kernel (task_count_t);



      /* User kernel configuration:
            1. Setup pipes
            2. Setup interruptions:
                  2.1 Init binds
                  2.2 Init handlers
            3. Setup tasks
      */

      /*
```

```
	Kernel units defined by user
	*/

#define KERNEL_VAR
#define PIPE_VAR

// extern KERNEL_VAR macro_state MS;

extern KERNEL_VAR task tasks[];
extern KERNEL_VAR ihandler ihandlers [];

extern KERNEL_VAR _bool ext_locked; // to lock INT0 and etc.

	// User defined functions

void setup_environment ();

// void configure_kernel ();
void setup_kernel ();

// void configure_hardware ();
void setup_hardware ();

void reset_hardware ();

void get_predefined_configuration ();

void switch_macro_state ();

void global_ihandler (ihandler_id_t);


extern task_id_t current_task_id;
void disable_current_task ();

#endif /* KERNEL_H_ */
```

kernel/kernel.c

```
/*
 * kernel.c
 *
 * Created: 9/6/2021 8:57:07 PM
 *  Author: mkentrru
 */

#include "kernel.h"

//void init_pipe (pipe * p, void * s){
	//p->s = s;
//}

void init_task (task_id_t tid, task_exec * f, task_state TS_FLAGS){
	task * t = (tasks + tid);
	// Task function
	t->f = f;

	// Task state flags
	t->s = TS_FLAGS;
}


void init_bind (struct ibind * ib, ibind_counter ib_counter, ibind_handler
* h){
	ib->c = ib_counter; // int counter
```

```
	ib->h = h; // int handler
}

void init_ihandler (ihandler_id_t hid, struct ibind * ib, ihandler_size hs)
{
	ihandler * h = (ihandlers + hid);
	h->b = ib; // handler binds
	h->c = hs; // binds count
}

void local_ihandler (ihandler * h){
	for (struct ibind * b = h->b; b < h->b + h->c; b++){
		// disabled: b->c == 0
		if (b->c && !--(b->c)) b->c = b->h();

	}
}


void enable_bind (struct ibind * b, ibind_counter c){
	b->c = c;
}

void disable_bind (struct ibind * b){
	b->c = bind_disabled;
}

//#include "uart.h"
task * current_task;
task_id_t current_task_id;
void run_kernel (task_count_t task_count){

	while (1){

		// bool running = false;
		for (current_task = tasks, current_task_id = 0;
		current_task < tasks + task_count;
		current_task++, current_task_id++
		){
			if (current_task->s & TS_FLAGS_ALLOWED) {
				current_task->f();
				// running = true;
				if (current_task->s & TS_FLAGS_ALLOWED_ONCE){
					current_task->s &= ~TS_FLAGS_ALLOWED;
					//put_string("Auto disabe: ", _false);
					//put_hex_byte (current_task_id);
					//new_line ();
				}
			}
		}
		// if (!running)   _SLEEP();?
	}
}

void global_ihandler (ihandler_id_t hid){
	breg cSREG = SREG;
	local_ihandler(ihandlers + hid);
	SREG = cSREG;
}

_bool interrupts__last_state = _false;

void enable_interrupts (_bool force){
	// if forcing to enable or last state
	if (force || interrupts__last_state) {
```

```
                sei();
                interrupts__last_state = _true;
        }
}


void disable_interrupts (){
        cli();
}


void lock_ext (struct ibind * b, ibind_counter c){
        ext_locked = _true;
        if (b){
                b->c = c;
        }
}

void unlock_ext (){
        ext_locked = _false;
}

void script__init (script * s, script_id_t size, void * data, script_result
(** f) (struct script *)) {

        s->size = size;
        s->current = 0;
        s->data = data;
        s->f = f;

}

void script__step (script * s) {
        byte_t res = 0;
        if ( s->current < s->size ) {

                res = s->f[s->current](s);
                s->current += res;
        }

        if ( s->current >= s->size || res == script__exit ){
                s->current = 0;
                disable_current_task ();
        }

}


void disable_current_task () {
        //put_string("disabling: ", _false);
        //put_hex_byte (current_task_id);
        //new_line ();
        (tasks + current_task_id)->s &= ~TS_FLAGS_ALLOWED;
}

//#include "uart.h"

void allow_task (task_id_t id) {
        //put_string("allowing: ", _false);
        //put_hex_byte (id);
        //new_line ();

        (tasks + id)->s |= TS_FLAGS_ALLOWED;
}
```

```
	void allow_script (task_id_t id, void (* script_task) ()) {
		//put_string("allowing: ", _false);
		//put_hex_byte (id);
		//new_line ();

		(tasks + id)->f = script_task;
		(tasks + id)->s |= TS_FLAGS_ALLOWED;
	}
```

kernel/_buffer.h

```
	/*
	 * _buffer.h
	 *
	 * Created: 9/6/2021 11:34:07 PM
	 *  Author: mkentrru
	 */


	#ifndef _BUFFER_H_
	#define _BUFFER_H_

	#include "_shared_macro.h"

	typedef byte_t _c_buffer_pos;


	typedef struct{

		byte_t * buf;
		byte_t mask;

		_c_buffer_pos in;
		_c_buffer_pos out;

		struct{
			_bool full: 1;
		}state;

	} _c_buffer;


	void _c_buffer__init (_c_buffer * c, byte_t * buf, byte_t mask);

	void _c_buffer__fix (_c_buffer * c);

	exit_status _c_buffer__push		(_c_buffer * c, byte_t d);

	exit_status _c_buffer__pop		(_c_buffer * c, byte_t * d);

	byte_t _c_buffer_slice (_c_buffer * c);

	exit_status _c_buffer__seek		(_c_buffer	*	c,	byte_t	s,
_c_buffer_pos * fpos);
	exit_status _c_buffer__parse_string (_c_buffer * c, byte_t d, _c_buffer_pos
* fpos, char * s);


	exit_status _c_buffer__pop_break (_c_buffer * c, byte_t * d, byte_t b);

	void _c_buffer__jump (_c_buffer * c, _c_buffer_pos pos);

	void _c_buffer__next (_c_buffer * c);
	exit_status _c_buffer__undo_input (_c_buffer * c);
```

```
void _c_buffer__skip (_c_buffer * c, byte_t pos);

exit_status _c_buffer_check (
      _c_buffer * c, _c_buffer_pos pos, _c_buffer_pos p_end,
      exit_status (*f) (byte_t));

#define inc_id(a, m) ((a + 1) & m)
#define dec_id(a, m) ((a - 1) & m)

#define _c_buffer__empty(b) (b->in == b->out && !(b->state.full))
#define _c_buffer__empty_v(b) (b.in == b.out && !(b.state.full))

#define _c_buffer__length(m, p1, p2) ((p2 - p1) & m)

#endif /* _BUFFER_H_ */
```

kernel/_buffer.c

```
/*
 * _buffer.c
 *
 * Created: 9/6/2021 11:34:16 PM
 *  Author: mkentrru
 */

#include "_buffer.h"

void _c_buffer__init (_c_buffer * c, byte_t * buf, byte_t mask){ // mask =
size - 1
      c->buf = buf;
      c->mask = mask;

      c->in = 0;
      c->out = 0;
      c->state.full = _false;
}

exit_status _c_buffer__push (_c_buffer * c, byte_t d){
      if (c->state.full) return exit_failure;

      c->buf[c->in] = d;
      c->in = inc_id(c->in, c->mask);

      if (c->in == c->out) c->state.full = _true;

      return exit_success;
}


exit_status _c_buffer__pop (_c_buffer * c, byte_t * d){
      if ( _c_buffer__empty(c) )     // empty
            return exit_failure;

      if (d) * d = c->buf[c->out];
      c->out = inc_id(c->out, c->mask);

      c->state.full = _false;

      return exit_success;
}

exit_status _c_buffer__undo_input (_c_buffer * c) {
      if (c->in != c->out) {
            c->in = dec_id(c->in, c->mask);
            return exit_success;
```

```
	}
	return exit_failure;
}

exit_status _c_buffer__seek (_c_buffer * c, byte_t s, _c_buffer_pos * fpos)
{
	if ( _c_buffer__empty(c) )
		return exit_failure;

	_c_buffer_pos pos = c->out;

	do{
		if (c->buf[pos] == s){
			* fpos = pos;
			return exit_success;
		}
		pos = inc_id(pos, c->mask);
	} while (pos != c->in);

	return exit_failure;
}

exit_status _c_buffer__parse_string
(_c_buffer * c, byte_t d, _c_buffer_pos * fpos, char * s) {

	if ( _c_buffer__empty(c) )
	return exit_failure;

	_c_buffer_pos pos = c->out;
	char a = 0;
	do{
		a = c->buf[pos];
		if (a == d || a == '\n' || a == '\0'){
			* fpos = pos;
			* s = '\0';
			return exit_success;
		}
		*(s++) = a;
		pos = inc_id(pos, c->mask);
	} while (pos != c->in);

	return exit_failure;

}


exit_status _c_buffer__pop_break (_c_buffer * c, byte_t * d, byte_t b){
	if (c->buf[c->out] == b) {
		return exit_failure;
	}

	* d = c->buf[c->out];

	c->out = inc_id (c->out, c->mask);
	c->state.full = _false;

	return exit_success;
}

byte_t _c_buffer__get (_c_buffer * c, _c_buffer_pos pos){
	return c->buf[pos & c->mask];
}

void _c_buffer__fix (_c_buffer * c){
	c->out = c->in;
```

```
	c->state.full = _false;
}

void _c_buffer__jump (_c_buffer * c, _c_buffer_pos pos){
	c->out = (pos & c->mask);
}

void _c_buffer__next (_c_buffer * c){
	c->out = inc_id(c->out, c->mask);
}


exit_status _c_buffer_check (
		_c_buffer * c,
		_c_buffer_pos pos, _c_buffer_pos p_end,
		exit_status (*f) (byte_t))
{
	while (pos != p_end){

		if (f(c->buf[pos]))
			return exit_failure;

		pos = inc_id(pos, c->mask);
	}
	return exit_success;
}

byte_t _c_buffer_slice (_c_buffer * c){
	byte_t d = c->buf[c->out];

	c->out = inc_id (c->out, c->mask);
	c->state.full = _false;

	return d;
}

void _c_buffer__skip (_c_buffer * c, byte_t pos){
	c->out = (c->out + pos) & c->mask;
}
```

kernel/ssi.h

```
/*
 * ssi.h
 *
 * Created: 9/21/2021 1:47:40 PM
 *  Author: mkentrru
 */


#ifndef SSI_H_
#define SSI_H_

#include "_shared_macro.h"
#include <avr/io.h>

#define ssi_output              PORTC
#define ssi_output_mask         0xFF
#define ssi_control                     PORTA
#define ssi_control_mask 0x0F

#define _SSI_SIZE 4
#define _SSI_SIZE_ASBYTE 2
```

```
	#define _ssi_A		0
	#define _ssi_B		1
	#define _ssi_C		2
	#define _ssi_D		3
	#define _ssi_E		4
	#define _ssi_F		5
	#define _ssi_G		6
	/*
			/	A	\
			F		B
			>	G	<
			E		C
			\	D	/
	*/

	#define _ssi_DP		7
	#define _ssi_hex_0 (_sf(_ssi_A)|_sf(_ssi_B)|_sf(_ssi_C)|_sf(_ssi_D)|
_sf(_ssi_E)|_sf(_ssi_F))
	#define _ssi_hex_1 (_sf(_ssi_B)|_sf(_ssi_C))
	#define _ssi_hex_2 (_sf(_ssi_A)|_sf(_ssi_B)|_sf(_ssi_D)|_sf(_ssi_E)|
_sf(_ssi_G))
	#define _ssi_hex_3 (_sf(_ssi_A)|_sf(_ssi_B)|_sf(_ssi_C)|_sf(_ssi_D)|
_sf(_ssi_G))
	#define _ssi_hex_4 (_sf(_ssi_B)|_sf(_ssi_C)|_sf(_ssi_F)|_sf(_ssi_G))
	#define _ssi_hex_5 (_sf(_ssi_A)|_sf(_ssi_C)|_sf(_ssi_D)|_sf(_ssi_F)|
_sf(_ssi_G))
	#define _ssi_hex_6 (_sf(_ssi_A)|_sf(_ssi_C)|_sf(_ssi_D)|_sf(_ssi_E)|
_sf(_ssi_F)|_sf(_ssi_G))
	#define _ssi_hex_7 (_sf(_ssi_A)|_sf(_ssi_B)|_sf(_ssi_C))
	#define _ssi_hex_8 (_sf(_ssi_A)|_sf(_ssi_B)|_sf(_ssi_C)|_sf(_ssi_D)|
_sf(_ssi_E)|_sf(_ssi_F)|_sf(_ssi_G))
	#define _ssi_hex_9 (_sf(_ssi_A)|_sf(_ssi_B)|_sf(_ssi_C)|_sf(_ssi_D)|
_sf(_ssi_F)|_sf(_ssi_G))

	#define _ssi_hex_A (_sf(_ssi_A)|_sf(_ssi_B)|_sf(_ssi_C)|_sf(_ssi_E)|
_sf(_ssi_F)|_sf(_ssi_G)|_sf(_ssi_DP))
	#define _ssi_hex_B (_sf(_ssi_A)|_sf(_ssi_B)|_sf(_ssi_C)|_sf(_ssi_D)|
_sf(_ssi_E)|_sf(_ssi_F)|_sf(_ssi_G)|_sf(_ssi_DP))
	#define _ssi_hex_C (_sf(_ssi_A)|_sf(_ssi_E)|_sf(_ssi_F)|_sf(_ssi_DP))
	#define _ssi_hex_D (_sf(_ssi_A)|_sf(_ssi_B)|_sf(_ssi_C)|_sf(_ssi_D)|
_sf(_ssi_E)|_sf(_ssi_F)|_sf(_ssi_DP))
	#define _ssi_hex_E (_sf(_ssi_A)|_sf(_ssi_D)|_sf(_ssi_E)|_sf(_ssi_F)|
_sf(_ssi_G)|_sf(_ssi_DP))
	#define _ssi_hex_F (_sf(_ssi_A)|_sf(_ssi_E)|_sf(_ssi_F)|_sf(_ssi_G)|
_sf(_ssi_DP))


	#define _ssi_sym_top	(_sf(_ssi_A))
	#define _ssi_sym_right	(_sf(_ssi_B)|_sf(_ssi_C))
	#define _ssi_sym_bottom	(_sf(_ssi_D))
	#define _ssi_sym_left	(_sf(_ssi_E)|_sf(_ssi_F))
	#define _ssi_digital_point	(_sf(_ssi_DP))


	//

	enum _ssi_modes {
		single,
		complex
	};

	enum _ssi_description_types{
		hex,
		symbols
	};
```

```
	extern byte_t output_ssi_description[4];
	extern enum _ssi_modes _ssi_mode;

	void _ssi__default_configuration ();

	void _ssi__set_word (byte_t a, byte_t b);
	void _ssi__update_description
	(enum _ssi_description_types t, byte_t * data, byte_t size, enum _ssi_modes
m);
	void _ssi_update ();


	void _ssi__store_description ();
	void _ssi__restore_description ();

	void init_default_ssi ();

	#endif /* SSI_H_ */
```

kernel/ssi.c

```
	/*
	 * ssi.c
	 *
	 * Created: 9/21/2021 1:48:07 PM
	 *  Author: mkentrru
	 */

	#include "ssi.h"

	byte_t output_ssi_description [_SSI_SIZE];
	byte_t output_ssi_description_backup [_SSI_SIZE];
	enum _ssi_modes _ssi_mode = single;
	enum _ssi_modes _ssi_mode_backup;

	byte_t _ssi__port_hex_code (byte_t v){

		switch (v){
			case 0:
			return _ssi_hex_0;
			break;

			case 1:
			return _ssi_hex_1;
			break;

			case 2:
			return _ssi_hex_2;
			break;

			case 3:
			return _ssi_hex_3;
			break;

			case 4:
			return _ssi_hex_4;
			break;

			case 5:
			return _ssi_hex_5;
			break;

			case 6:
			return _ssi_hex_6;
```

```
                break;

                case 7:
                return _ssi_hex_7;
                break;

                case 8:
                return _ssi_hex_8;
                break;

                case 9:
                return _ssi_hex_9;
                break;
                //
                case 0x0A:
                return _ssi_hex_A;
                break;

                case 0x0B:
                return _ssi_hex_B;
                break;

                case 0x0C:
                return _ssi_hex_C;
                break;

                case 0x0D:
                return _ssi_hex_D;
                break;

                case 0x0E:
                return _ssi_hex_E;
                break;

                case 0x0F:
                return _ssi_hex_F;
                break;

                default:
                return _ssi_digital_point;
                break;
        }
}

byte_t _ssi__port_simbols_code (byte_t v){
        switch (v){

                case 0x00:
                return 0x00;
                break;

                case 0x01:
                return _ssi_sym_top;
                break;

                case 0x02:
                return _ssi_sym_right;
                break;

                        case 0x03:
                        return _ssi_sym_top | _ssi_sym_right;
                        break;

                case 0x04:
                return _ssi_sym_bottom;
```

```
                  break;

                        case 0x07:
                        return _ssi_sym_top | _ssi_sym_right | _ssi_sym_bottom;
                        break;

                  case 0x08:
                  return _ssi_sym_left;
                  break;

                        case 0x0F:
                        return _ssi_sym_top | _ssi_sym_right | _ssi_sym_bottom |
_ssi_sym_left;
                        break;

                  case 0x10:
                  return _ssi_digital_point;
                  break;

                  default:
                  return _ssi_digital_point;
                  break;
            }

      }


      void _ssi__update_description
      (enum _ssi_description_types t, byte_t * data, byte_t size, enum _ssi_modes
m){
            switch (t){
                  case hex:
                        for (byte_t a = 0; a < size; a++){
                              if (a % 2)
                                    output_ssi_description         [a]         =
_ssi__port_hex_code(data[a >> 1] >> 4);
                              else
                                    output_ssi_description         [a]         =
_ssi__port_hex_code(data[a >> 1] & 0x0F);
                        }
                  break;

                  default:
                        for (byte_t a = 0; a < size; a++){
                              output_ssi_description            [a]            =
_ssi__port_simbols_code(data[a]);
                        }
                  break;
            };
            _ssi_mode = m;
      }

      void _ssi__set_word (byte_t a, byte_t b) {

            output_ssi_description [0] = _ssi__port_hex_code (b & 0x0F);
            output_ssi_description [1] = _ssi__port_hex_code (b >> 4);

            output_ssi_description [2] = _ssi__port_hex_code (a & 0x0F);
            output_ssi_description [3] = _ssi__port_hex_code (a >> 4);

            _ssi_mode = complex;
      }


      void _ssi_update (){
```

```
	static byte_t current_ssi_node = 0;
	ssi_control &= ~ssi_control_mask;
	ssi_control |= (1<<current_ssi_node);

	if ( _ssi_mode == complex ) {
		ssi_output = output_ssi_description [current_ssi_node];
	}
	else {
		ssi_output = * output_ssi_description;
	}

	current_ssi_node = (current_ssi_node + 1) % 4;
}


void init_default_ssi (){
	DDRA setflag ssi_control_mask;
	DDRC setflag ssi_output_mask;
}

void _ssi__default_configuration () {
	_ssi_mode = complex;
}

void _ssi__store_description (){
	_ssi_mode_backup = _ssi_mode;
	for (byte_t a = 0; a < _SSI_SIZE; a++){
		output_ssi_description_backup[a] = output_ssi_description[a];
		output_ssi_description[a] = 0;
	}
}

void _ssi__restore_description (){
	_ssi_mode = _ssi_mode_backup;
	for (byte_t a = 0; a < _SSI_SIZE; a++){
		output_ssi_description[a] = output_ssi_description_backup[a];
	}
}
```

kernel/uart.h

```
/*
 * uart.h
 *
 * Created: 9/15/2021 1:29:24 AM
 *  Author: mkentrru
 */


#ifndef UART_H_
#define UART_H_

#include "_shared_macro.h"
#include "_buffer.h"
#include <avr/io.h>
#include <avr/pgmspace.h>

#define enable_udre_int		UCSRB setflag	_sf(UDRIE);
#define disable_udre_int UCSRB dropflag	_df(UDRIE);

#define enable_rxc_int		UCSRB setflag	_sf(RXCIE);
#define disable_rxc_int		UCSRB dropflag	_df(RXCIE);

#define enable_input		enable_rxc_int
```

```
#define disable_input          disable_rxc_int

#define enable_output          enable_udre_int
#define disable_output         disable_udre_int


#define uart_input_size        0x100
#define uart_output_size 0x100
#define uart_error_size        0x10

extern _c_buffer uart_input;
extern _c_buffer uart_output;
extern _c_buffer uart_error;

#define _stdout          &uart_output
#define _stdin           &uart_input
#define _stderr          &uart_error

#define _stdin_buf uart_input.buf

#define _stdin_empty (_c_buffer__empty_v(uart_input))

#define  _stdin_buf_length(p1,  p2)  (_c_buffer__length(uart_input.mask,  p1,
p2))

#define _stdin_index(i) ((i) & uart_input.mask)


// #define _IBB 0x02 // input begin byte    (start of text)
#define _IEB '\r' // input end byte         (end of text)
// #define _OIBB 0x04 // output begin interrupt byte
#define _NLB '\n' // output end interrupt byte

/* commands */
#define _IOFFSET_CMD     0x01
#define _IOFFSET_DB            0x02
#define _CMD_NEW_STRING (byte_t) 'N'
#define _CMD_REM_STRING (byte_t) 'R'
#define _CMD_SET_SIZE    (byte_t) 'S'

/* errors */
#define _ERR_BUF   0x10
#define _ERR_BUF_INPUT_OVERFLOW            (_ERR_BUF + 1)
#define _ERR_BUF_OUTPUT_OVERFLOW     (_ERR_BUF + 2)
#define _ERR_BUF_EMPTY                     (_ERR_BUF + 3)

#define _ERR_CMD   0x20
#define _ERR_CMD_NO_IBB                    (_ERR_CMD + 1)
#define _ERR_CMD_NO_IEB                    (_ERR_CMD + 2)
#define _ERR_CMD_UNDEF                     (_ERR_CMD + 3)
#define _ERR_CMD_CB_JUNK             (_ERR_CMD + 4)
#define _ERR_CMD_REQ_DATA                  (_ERR_CMD + 5)
#define _ERR_CMD_INVALID_DATA        (_ERR_CMD + 6)

#define _ERR_USER  0x30




void init_default_uart ();
void put_buffer (byte_t * buf, byte_t s);
void put_hex_buffer (byte_t * buf, byte_t s);
void put_string (const char * buf, _bool new_line);
```

```
void put_error (byte_t err);
void put_extra_error (byte_t err, byte_t * buf);
byte_t get_byte ();



#define MAX_ARGCOUNT 4
struct _args {
	byte_t argc;
	char * argv[4];
};
extern struct _args args;

void put_byte (byte_t b);
void put_hex_byte (byte_t b);
void new_line ();

#define locking_output(b) { \
	while ( _c_buffer__push(_stdout, b) ) { \
		enable_output \
	} \
}

void locking_output__byte (byte_t b);
void locking_output__new_line ();
void locking_output__hex_byte (byte_t b);
void locking_output__hex_buffer (byte_t * buf, byte_t s);
void locking_output__string (const char * buf, _bool new_line);


/*
	FOR PROGMEM
*/

void P__put__string (const char * buf, _bool new_line);
void P_locking_output__string (const char * buf, _bool new_line);

#endif /* UART_H_ */
```

kernel/uart.c

```
/*
 * uart.c
 *
 * Created: 9/15/2021 1:33:16 AM
 *  Author: mkentrru
 */

#include "uart.h"

// #define BAUD 9600

void init_default_uart (){
	UCSRB |=    (1<<RXEN) | (1<<TXEN);
	UCSRC |=    (1<<URSEL)| (1<<UCSZ0) | (1<<UCSZ1);
	UBRRL =     51; //(8000000)/(16 * BAUD) - 1; // 51; // page 143
}

void put_buffer (byte_t * buf, byte_t s){
	// _c_buffer__push(_stdout, _IBB);

	for (byte_t pos = 0; pos < s; pos++){
		_c_buffer__push(_stdout, buf[pos]);
	}
	_c_buffer__push(_stdout, _IEB);
```

```
	enable_output
}

byte_t get_hex_char (byte_t b) {
	if (b > 0x10) return '.';
	if (b < 0x0A) return ('0' + b);
	return ('A' + (b - 0x0A));
}

void put_hex_buffer (byte_t * buf, byte_t s) {
	for (byte_t pos = 0; pos < s; pos++){
		byte_t b = buf[pos];
		_c_buffer__push(_stdout, get_hex_char( (b & 0xF0)>>4 ));
		_c_buffer__push(_stdout, get_hex_char( (b & 0x0F) ));
		_c_buffer__push(_stdout, ' ');
		if ( pos % 8 == 7) {
			_c_buffer__push(_stdout, '\n');
			_c_buffer__push(_stdout, '\r');
		}
	}

	_c_buffer__push(_stdout, '\n');
	_c_buffer__push(_stdout, '\r');

	enable_output
}

void put_string (const char * buf, _bool new_line){
	// _c_buffer__push(_stdout, _IBB);

	for (byte_t pos = 0;  buf[pos]; pos++){
		_c_buffer__push(_stdout, buf[pos]);
	}

	if (new_line) {
		_c_buffer__push(_stdout, '\n');
		_c_buffer__push(_stdout, '\r');
	}


	enable_output
}


//void put_error (byte_t err){
	//
	//// _c_buffer__push(_stderr, _OIBB);
	//// _c_buffer__push(_stderr, err);
	//// _c_buffer__push(_stderr, _OIEB);
	//
	//enable_output
//}
//
//void put_extra_error (byte_t err, byte_t * buf){
	//
	//_c_buffer__push(_stderr, _OIBB);
	//_c_buffer__push(_stderr, err);
	//
	//if (buf){
		//for (byte_t pos = 0;  buf[pos]; pos++){
			//_c_buffer__push(_stderr, buf[pos]);
		//}
	//}
	//_c_buffer__push(_stderr, _OIEB);
//
```

```
	//
	//enable_output
//}

byte_t get_byte (){
	return _c_buffer_slice (_stdin);
}



void put_byte (byte_t b) {
	_c_buffer__push(_stdout, b);
	enable_output
}

void put_hex_byte (byte_t b) {
	put_byte ( get_hex_char( (b & 0xF0)>>4 ) );
	put_byte ( get_hex_char( (b & 0x0F) ) );
	enable_output
}

void new_line () {
	_c_buffer__push(_stdout, '\n');
	_c_buffer__push(_stdout, '\r');
	enable_output
}

//void put_buffer (byte_t * buf, byte_t s){
	//// _c_buffer__push(_stdout, _IBB);
	//
	//for (byte_t pos = 0; pos < s; pos++){
		//_c_buffer__push(_stdout, buf[pos]);
	//}
	//_c_buffer__push(_stdout, _IEB);
	//
	//enable_output
//}



void locking_output__byte (byte_t b) {
	locking_output(b);
	enable_output
}

void locking_output__new_line () {
	locking_output__byte ('\n');
	locking_output__byte ('\r');
}

void locking_output__hex_byte (byte_t b) {
	locking_output__byte ( get_hex_char( (b & 0xF0)>>4 ) );
	locking_output__byte ( get_hex_char( (b & 0x0F) ) );
}

void locking_output__hex_buffer (byte_t * buf, byte_t s) {
	for (byte_t pos = 0; pos < s; pos++){
		byte_t b = buf[pos];
		locking_output__byte (get_hex_char( (b & 0xF0)>>4 ));
		locking_output__byte (get_hex_char( (b & 0x0F) ));
		locking_output__byte (' ');
		if ( pos % 8 == 7) locking_output__new_line ();
	}
```

```
	locking_output__new_line ();

	enable_output
}

void locking_output__string (const char * buf, _bool new_line) {
	// _c_buffer__push(_stdout, _IBB);

	for (byte_t pos = 0;  buf[pos]; pos++){
		locking_output__byte (buf[pos]);
	}

	if (new_line) locking_output__new_line ();

	enable_output
}


void P__put__string (const char * buf, _bool new_line) {
	byte_t a = 0;
	while ( (a = pgm_read_byte(buf)) != 0x00 ) {
		put_byte (a); buf++;
	}

	if (new_line) {
		_c_buffer__push(_stdout, '\n');
		_c_buffer__push(_stdout, '\r');
	}

	enable_output
}

void P_locking_output__string (const char * buf, _bool new_line) {
	byte_t a = 0;
	while ( (a = pgm_read_byte(buf)) != 0x00 ) {
		locking_output__byte (a); buf++;
	}

	if (new_line) locking_output__new_line ();

	enable_output
}
```

kernel/filesystem/fs_configuration.h

```
/*
 * fs_configuration.h
 *
 * Created: 11/1/2021 9:24:38 PM
 *  Author: mkentrru
 */


#ifndef FS_CONFIGURATION_H_
#define FS_CONFIGURATION_H_


#define K_SIZE 0x20
#define K_COUNT 0x20
#define MAX_OBJ_SIZE (K_SIZE * (K_COUNT - 1))

typedef uint8_t KID;

#define INTRO_KID  0
#define ROOT_KID   1
```

```
#define ktoa(a) (a * K_SIZE)

#define KP_END 0x20
#define KP_REM 0x40
#define KP_NAN 0x00


#define obj_type__file   1
#define obj_type__dir    0


typedef uint32_t memsync_control;

typedef byte_t passwd_t;
typedef byte_t kaddr_t;

#endif /* FS_CONFIGURATION_H_ */
```

kernel/filesystem/fs_types.h

```
/*
 * fs_types.h
 *
 * Created: 11/1/2021 9:25:54 PM
 *  Author: mkentrru
 */


#ifndef FS_TYPES_H_
#define FS_TYPES_H_

#include "memory.h"
#include "../kernel.h"

typedef struct{

      // byte_t name [UNIT_NAME_SIZE]; // 2 bytes

      byte_t nA; byte_t nB; // name

      KID KID:    5; // 1 byte
      byte_t TYPE:1;
      byte_t iROOT:     1;
      byte_t iUSER:     1;

      byte_t SIZE;

} fs_obj_descr;



union intro_flags {
      struct {
            byte_t fs_free_amount: 5;
            byte_t fs_type: 3;
      } as_flags;

      byte_t as_byte;
};


typedef union{
      byte_t row [K_SIZE];

      struct{
```

```
            union intro_flags flags;

            KID kp [K_SIZE - 1];

      } as_intro;

      struct{
            fs_obj_descr items [K_SIZE / sizeof(fs_obj_descr)];
      } as_directory;

} K_t; // k(c)lascter


typedef struct {
      K_t intro;                                          // 00:20

      K_t current_obj_buf;                              // 20:20

      memsync_control intro__memsync;                   // 40:04
      memsync_control current_obj__memsync;// 44:04

      fs_obj_descr current_folder;                // 48:04
      fs_obj_descr current_obj;                         // 4B:04


      KID search_stack [K_COUNT];                       // 50:20
      byte_t search_level;                              // 70:01
      fs_obj_descr * descr_pointer;               // 71:02

      struct {

            passwd_t up;
            passwd_t rp;

            _bool current_try: 1;

            _bool root: 1;
            _bool user: 1;
            _bool defined_root: 1;
            _bool defined_user: 1;

      } access_control;                                 //    73:3

      _bool ssi_showing_name;
} _fs_context;                                          //    76
#endif /* FS_TYPES_H_ */
```

kernel/filesystem/filesystem.h

```
/*
 * filesystem.h
 *
 * Created: 11/1/2021 10:27:29 PM
 *  Author: mkentrru
 */


#ifndef FILESYSTEM_H_
#define FILESYSTEM_H_

#include "../kernel.h"
#include "../uart.h"

#include "fs_configuration.h"
```

```
#include "fs_types.h"
#include "memory.h"
#include "fs_scripts.h"

int init_filesystem ();

exit_status fs__check__cluster_chain ();

void fs__ssi__update ();
void fs__ssi__switch ();


extern const char command_prompt [];

void put_prompt ();
void locking_output__prompt ();

#endif /* FILESYSTEM_H_ */
```

kernel/filesystem/memory.h

```
/*
 * memory.h
 *
 * Created: 11/1/2021 9:15:15 PM
 *  Author: mkentrru
 */


#ifndef MEMORY_H_
#define MEMORY_H_

#include "avr/io.h"

#include "../_shared_macro.h"
#include "../kernel.h"

typedef uint32_t memaddress;

// busy EEPROM
byte_t mem_busy__read (memaddress a);
void mem_busy__read_area (memaddress a, memaddress size, byte_t * d);
void mem_busy__write (memaddress a, byte_t d);
void mem_busy__write_area (memaddress a, memaddress size, byte_t * d);

// not busy EEPROM
#define MEM_TASK_ID      0
void mem_script__init ();
void mem_script_task ();

void mem__request (memaddress a, memaddress size, byte_t * d, task_id_t
tid, _bool to_read);



#define enable_eeprom_ready_int            EECR setflag _sf(EERIE);
#define disable_eeprom_ready_int     EECR dropflag _df(EERIE);

#define start_mem        enable_eeprom_ready_int
#define stop_mem         disable_eeprom_ready_int


typedef struct {
/*
      on interrupt i: // (EEPROM Ready Interrupt)
            if to_read:
```

```
                        EEAR = a;
                        EECR |= (1<<EERE);
                        d = EEDR;
                else:
                        EEAR = a;
                        EEDR = d;
                        EECR |= (1<<EEMWE);
                        EECR |= (1<<EEWE);

                disable interrupt i

                allow task waiting
*/
        struct {
                byte_t to_read:    1;
        } descr;

        task_id_t waiting;

        memaddress a;
        byte_t d;

} _mem_quere;

typedef struct {
        byte_t to_read:          1;

        memaddress a;
        memaddress size;
        memaddress done;

        byte_t * d;

        task_id_t caller;
} _mem_task;



extern _mem_quere mem_quere;
extern _mem_task mem_task;

/*
                MEMORY SCRIPT
*/

extern script mem_script;
void mem_script__init ();
void mem_script_task ();

#endif /* MEMORY_H_ */
```

kernel/filesystem/memory.c

```
/*
 * fs_memory.c
 *
 * Created: 11/1/2021 9:11:25 PM
 *  Author: mkentrru
 */

#include "memory.h"

// Eeprom memory
byte_t mem_busy__read (memaddress a) {
        while ( EECR & (1<<EEWE) );
```

```
	EEAR = a;
	EECR |= (1<<EERE);

	return EEDR;
}

void mem_busy__read_area (memaddress a, memaddress size, byte_t * d) {
	while ( size ) {
		* d = mem_busy__read (a);
		d++; a++; size--;
	}
}

void mem_busy__write (memaddress a, byte_t d) {
	while(EECR & (1<<EEWE));

	EEAR = a;
	EEDR = d;

	disable_interrupts ();
	EECR |= (1<<EEMWE);
	EECR |= (1<<EEWE);
	enable_interrupts (_false);
}

void mem_busy__write_area (memaddress a, memaddress size, byte_t * d) {
	while ( size ) {
		mem_busy__write (a, * d);
		d++; a++; size--;
	}
}


void mem__read (memaddress a, task_id_t tid) {
	mem_quere.a = a;

	mem_quere.descr.to_read = 1;
	mem_quere.waiting = tid;

	start_mem
}
void mem__write (memaddress a, byte_t d, task_id_t tid) {
	mem_quere.a = a;
	mem_quere.d = d;
	mem_quere.descr.to_read = 0;
	mem_quere.waiting = tid;

	start_mem
}

_bool is_mem_task_done () {
	if ( mem_task.done >= mem_task.size ) return _true;
	return _false;
}


script_result mem_task_step__control () {
	if ( is_mem_task_done() ) {
		// awake the one who asked
		allow_task (mem_task.caller);
		return script__exit;
	}
	return script__next;
}
```

```
	script_result mem_task_step__prepare () {
		// may be allowed instantly after mem_start
		disable_current_task ();
		if (mem_task.to_read == 1) {
			mem__read (mem_task.a, MEM_TASK_ID);
		}
		else {
			mem__write	(mem_task.a,	mem_task.d	[mem_task.done],
MEM_TASK_ID);
		}
		return script__next;
	}

	#define mem_script__check_done -2
	script_result mem_task_step__cover () {
		if (mem_task.to_read) {
			mem_task.d [mem_task.done] = mem_quere.d;
		}

		mem_task.done++;
		mem_task.a++;
		return mem_script__check_done;
	}

	void mem__request (memaddress a, memaddress size, byte_t * d, task_id_t
tid, _bool to_read) {
		if ( to_read ) mem_task.to_read = 1;
		else mem_task.to_read = 0;

		mem_task.a = a;
		mem_task.d = d;
		mem_task.done = 0;
		mem_task.size = size;
		mem_task.caller = tid;

		allow_task (MEM_TASK_ID);
		disable_current_task ();
	}

	#define mem__script_size 3
	script_result (* mem_script_f [mem__script_size]) (struct script *)  = {
		mem_task_step__control,
		mem_task_step__prepare,
		mem_task_step__cover
	};

	void mem_script__init () {
		script__init (& mem_script, mem__script_size, _null, mem_script_f );
	}

	void mem_script_task () {
		script__step (& mem_script);
	}
```

kernel/filesystem/fs_scripts.h

```
	/*
	 * fs_scripts.h
	 *
	 * Created: 11/1/2021 9:30:58 PM
	 *  Author: mkentrru
	 */
```

```
#ifndef FS_SCRIPTS_H_
#define FS_SCRIPTS_H_

#include "filesystem.h"

/*
            SHARED OBJECTS
*/


extern _fs_context fs_context;

/*
            MEMORY FUNCTIONS
*/

void  load_kluster (KID id, byte_t * d, task_id_t tid);
void  store_kluster (KID id, byte_t * d, task_id_t tid);


/*
            OBJ DESCR
*/

_bool fs_obj_descr__empty (fs_obj_descr * o);
_bool fs_obj_descr__removed (fs_obj_descr * o);
_bool fs_obj_descr__shared (fs_obj_descr * o);
_bool fs_obj_descr__good (fs_obj_descr * o);
_bool fs_obj_descr__free (fs_obj_descr * o);

_bool fs_obj_descr__access_deny (fs_obj_descr * o);
_bool fs_obj_descr__available (fs_obj_descr * o);

void fs_obj_descr__print (fs_obj_descr * o);
void fs_obj_descr__print_info (fs_obj_descr * o);

fs_obj_descr  *  fs_obj_descr__get_by_name  (byte_t  nA,  byte_t  nB,  byte_t
TYPE);
fs_obj_descr * fs_obj_descr__get_by_KID (KID id, byte_t TYPE);


void remove__obj (fs_obj_descr * o);

/*
            INTRO KLUSTER POINTERS
*/

_bool fs_kpointer__obj_deleted (KID id);
KID          fs_kpointer__get_first_kid (KID id);
KID          fs_kpointer__get_last_kid (KID id);
KID          fs_kpointer__get_next_kid (KID id);
void  fs_kpointer__mark_removed (KID id);
void  fs_kpointer__reduce_free_value (byte_t count);

/*
            KLUSTER CHANGES SYNC
*/

void  fs_K__change  (K_t  *  k,  memsync_control  *  ms,  kaddr_t  addr,  void  *
data, byte_t s);
void fs_K__mark_changed (memsync_control * ms);
exit_status  fs_K__get_changed  (memsync_control  *  ms,  memaddress  *  addr,
memaddress * size);
```

```
//    S           C           R           I           P           T
S//



/*
            ls - LIST DIRECTORY
*/

script_result fs__ls_entry ();
script_result fs__ls_load_kluster ();
#define fs__ls_chain_loop -2
script_result fs__ls_check_chain ();
script_result fs__outro ();

extern script fs__ls__script;
void fs__ls__script__init ();
void fs__ls__script_task ();


/*
            cat - DISPLAY FILE DATA
*/

script_result fs__find__by_name ();
script_result fs__find__by_KID ();
script_result fs__find__failure ();

extern script fs__cat__script;
void fs__cat__script__init ();
void fs__cat__script_task ();

/*
            cd - CHANGE DIRECTORY
*/



extern script fs__cd__script;
void fs__cd__script__init ();
void fs__cd__script_task ();

/*
            info - DISPLAY OBJECT INFO
*/

script_result fs__check__if_file ();
script_result fs__sub__setup ();
script_result fs__sub__entry ();
#define fs__sub__root_level 1
#define fs__sub__loop -4
#define fs__sub__dive -2
script_result fs__sub__control ();

extern script fs__info__script;
void fs__info__script__init ();
void fs__info__script_task ();

/*
            rm - REMOVE OBJECT (RECURSIVELY)
```

```
*/

script_result fs__sync__intro ();
script_result sync__K (script_result on_exit, KID id);

extern script fs__rm__script;
void fs__rm__script__init ();
void fs__rm__script_task ();

/*
            touch - CREATE FILE
*/

script_result fs__sync__current_K ();
script_result fs__create__check_same ();
script_result fs__create__build_new_descr ();
script_result fs__create__write_new_descr ();
script_result fs__touch__write_data__setup ();
script_result fs__touch__write_data ();
script_result fs__touch__write__check ();


extern script fs__touch__script;
void fs__touch__script__init (void * d);
void fs__touch__script_task ();


/*
            mkdir - CREATE DIRECTORY
*/

extern script fs__mkdir__script;
void fs__mkdir__script__init ();
void fs__mkdir__script_task ();


/*
            login - LOGIN AS ROOT/USER
*/

extern script passwd_input__script;
void passwd_input__script__init ();
void passwd_input__script_task ();

      // if *P file does not exists
extern script passwd_init__script;
void passwd_init__script__init ();
void passwd_init__script_task ();

/*
            passwd - CHANGE PASSWD
*/

extern script passwd_change__script;
void passwd_change__script__init ();
void passwd_change__script_task ();


/*
            SIMPLE SCRIPT STEPs
*/

script_result failure_dumb ();
script_result simply_end ();
```

```
extern fs_obj_descr tmp_descr;

#endif /* FS_SCRIPTS_H_ */
```

kernel/filesystem/_fs_init.c

```
/*
 * _fs_init.c
 *
 * Created: 11/1/2021 10:26:10 PM
 *  Author: mkentrru
 */

#include "fs_scripts.h"
#include "../ssi.h"
#include "../../lab/lab.h"


const char command_prompt [] = "> ";

int init_filesystem () {

	mem_busy__read_area	(ktoa(INTRO_KID),	K_SIZE,	(byte_t	*)	&
fs_context.intro.row);
	fs_context.intro__memsync = 0;

	fs_context.current_folder.KID = ROOT_KID;

	fs_context.current_obj.KID = fs_context.current_folder.KID;
	fs_context.current_obj__memsync = 0;

	fs_context.current_obj.nA = ':';
	fs_context.current_obj.nB = '/';
	fs_context.current_folder.nA = ':';
	fs_context.current_folder.nB = '/';


	fs_context.access_control.root = 0;
	fs_context.access_control.user = 0;
	fs_context.access_control.defined_root = 0;
	fs_context.access_control.defined_user = 0;

	fs_context.search_stack [0] = 0;
	fs_context.search_level = 0;

	fs_context.ssi_showing_name = _true;
	fs__ssi__update ();


	KID co = fs_context.current_obj.KID;
	fs_obj_descr * p = _null;

	do {

		mem_busy__read_area	(ktoa(co),	K_SIZE,	(byte_t	*)	&
fs_context.current_obj_buf.row);

		if ( ! fs_context.access_control.defined_root ) {
			p = fs_obj_descr__get_by_name ('R', 'P', obj_type__file);
			if ( p != _null && p->KID != KP_NAN ) {
				fs_context.access_control.rp	=	mem_busy__read	(
ktoa(p->KID) );
				fs_context.access_control.defined_root = _true;
				P_locking_output__string (PSTR("Root passwd found:
"), _false);
```

```
                              locking_output__hex_buffer     ((byte_t     *)     &
fs_context.access_control.rp, sizeof(passwd_t));
                        }
                  }

                  if ( ! fs_context.access_control.defined_user ) {
                        p = fs_obj_descr__get_by_name ('U', 'P', obj_type__file);
                        if ( p != _null && p->KID != KP_NAN ) {
                              fs_context.access_control.up   =   mem_busy__read   (
ktoa(p->KID) );
                              fs_context.access_control.defined_user = _true;
                              P_locking_output__string (PSTR("User passwd found:
"), _false);
                              locking_output__hex_buffer     ((byte_t     *)     &
fs_context.access_control.up, sizeof(passwd_t));
                        }
                  }

            } while ( (co = fs_kpointer__get_next_kid (co)) != KP_END &&
            !         (         fs_context.access_control.defined_root          &&
fs_context.access_control.defined_user ) );

            // if not first Kluster of ROOT_DIR loaded
            if ( fs_context.current_obj.KID != co ){
                  mem_busy__read_area
                  (ktoa(fs_context.current_obj.KID),   K_SIZE,   (byte_t   *)   &
fs_context.current_obj_buf.row);
            }

            if ( ! fs_context.access_control.defined_root )
                  P_locking_output__string   (PSTR("Root   passwd   NOT   found!"),
_true);
            if ( ! fs_context.access_control.defined_user )
                  P_locking_output__string   (PSTR("User   passwd   NOT   found!"),
_true);

            return exit_success;
      }


      void fs__ssi__update () {
            if ( fs_context.ssi_showing_name ) {
                  _ssi__set_word
                  (fs_context.current_folder.nA, fs_context.current_folder.nB);
            }
            else {
                  _ssi__set_word (0, fs_context.current_folder.KID);
            }
      }

      void fs__ssi__switch () {
            if      (      fs_context.ssi_showing_name      ==      _true      )
fs_context.ssi_showing_name = _false;
            else fs_context.ssi_showing_name = _true;

            fs__ssi__update ();
      }



      exit_status fs__check__cluster_chain () {
            memsync_control checked  = 0, mask = 1;
            KID * b = (KID *) & fs_context.intro.row, nid = ROOT_KID;

            for ( KID fid = ROOT_KID; fid < K_COUNT; fid++ ) {
```

```
                        mask = 1; mask <<= fid;

                        P_locking_output__string(PSTR("{"), _false);
                        locking_output__hex_byte(fid);

                        if ( * (b + fid) == KP_END || * (b + fid) == KP_NAN || * (b +
fid) == KP_REM ) {
                              P_locking_output__string(PSTR("}"), _false);
                              checked |= mask;
                              continue;
                        }

                        if ( ! (mask & checked) ) {

                              checked |= mask;

                              nid = fid;
                              while ( * (b + nid) != KP_END ) {

                                    nid = * (b + nid);

                                    P_locking_output__string(PSTR(">"), _false);
                                    locking_output__hex_byte(nid);

                                    if ( nid == KP_REM || nid == KP_NAN ) {
                                          // using wrong Kluster !
                                          P_locking_output__string(PSTR("}"), _true);
                                          P_locking_output__string         (PSTR("Chain
contains removed/undefined Kluster!"), _true);
                                          return exit_failure;
                                    }

                                    mask = 1; mask <<= nid;

                                    if ( mask & checked ) {
                                          // Kluster is used twice
                                          P_locking_output__string(PSTR("}"), _true);
                                          P_locking_output__string        (PSTR("Single
Kluster is used in different chains!"), _true);
                                          return exit_failure;
                                    }

                                    checked |= mask;

                              }
                        }

                        P_locking_output__string(PSTR("}"), _false);

                  }

                  locking_output__new_line ();
                  return exit_success;
            }

            void put_prompt () {
                  if ( fs_context.access_control.root ) put_byte ('r');
                  else put_byte ('-');

                  if ( fs_context.access_control.user ) put_byte ('u');
                  else put_byte ('-');

                  put_string (command_prompt, _false);
                  fs__ssi__update ();
```

```
	enable_input
}

void locking_output__prompt () {
	if ( fs_context.access_control.root ) locking_output__byte ('r');
	else locking_output__byte ('-');

	if ( fs_context.access_control.user ) locking_output__byte ('u');
	else locking_output__byte ('-');

	locking_output__string (command_prompt, _false);
	fs__ssi__update ();
	enable_input
}
```

kernel/filesystem/_fs_script__shared.c

```
/*
 * _fs_script__shared.c
 *
 * Created: 11/1/2021 9:37:28 PM
 *  Author: mkentrru
 */

#include "fs_scripts.h"

/*
		MEMORY
*/

void load_kluster (KID id, byte_t * d, task_id_t tid) {
	mem__request (ktoa(id), K_SIZE, d, tid, _true);
}

void store_kluster (KID id, byte_t * d, task_id_t tid) {
	mem__request (ktoa(id), K_SIZE, d, tid, _false);
}


/*
		OBJ DESCR
*/

_bool fs_obj_descr__empty (fs_obj_descr * o) {
	if (o->KID == 0) return _true;
	return _false;
}

_bool fs_obj_descr__shared (fs_obj_descr * o) {
	if ( o->TYPE == obj_type__dir && o->nA == '.' && o->nB == '.' )
		return _true;
	if ( fs_context.current_folder.KID == ROOT_KID &&
		o->TYPE == obj_type__file &&
		(o->nA == 'R' || o->nA == 'U')
		&& o->nB == 'P' )
		return _true;

	return _false;
}

_bool fs_obj_descr__good (fs_obj_descr * o) {
	// _bool fs_obj_descr__removed (fs_obj_descr * o)
	if (
	fs_obj_descr__empty (o) ||
```

```
		fs_obj_descr__shared (o) ||
		fs_obj_descr__removed (o) ) return _false;

		return _true;
	}

	_bool fs_obj_descr__free (fs_obj_descr * o) {
		if (
		fs_obj_descr__empty (o) ||
		fs_obj_descr__removed (o) ) return _true;

		return _false;
	}

	void fs_obj_descr__print (fs_obj_descr * o) {
		// "Type Owner Name FirstKID Size Removed"
		if ( o == _null ) return;
		if ( o->TYPE == obj_type__file) P_locking_output__string (PSTR("File
"), _false);
		else P_locking_output__string (PSTR("Dir   "), _false);

		if ( o->iROOT && o->iUSER ) P_locking_output__string(PSTR("????   "),
_false);
		else  if  (  o->iROOT  )  P_locking_output__string  (PSTR("Root     "),
_false);
		else  if  (  o->iUSER  )  P_locking_output__string  (PSTR("User     "),
_false);
		else P_locking_output__string (PSTR("All    "), _false);

		if ( valid_cmd_symbol(o->nA) && valid_cmd_symbol(o->nB) ) {
			locking_output__byte (o->nA);
			locking_output__byte (o->nB);
		}
		else {
			P_locking_output__string (PSTR("__"), _false);
		}

		P_locking_output__string (PSTR("    "), _false);
		locking_output__hex_byte (o->KID);

		P_locking_output__string (PSTR("        "), _false);
		locking_output__hex_byte (o->SIZE);

		P_locking_output__string (PSTR("     "), _false);
		if ( fs_obj_descr__removed (o) ) {
			P_locking_output__string (PSTR(" * "), _false);
		}

		locking_output__new_line ();
	}

	_bool fs_obj_descr__access_deny (fs_obj_descr * o) {

		if ( o->iUSER ) {
			if ( ! fs_context.access_control.defined_user ) {
				locking_output__new_line ();
				P_locking_output__string(PSTR("User  passwd  undefined."),
_true);
				return _true;
			}
			else if (! fs_context.access_control.user) {
				return _true;
			}
		}
```

```
            if ( o->iROOT ) {
                  if ( ! fs_context.access_control.defined_root ) {
                        locking_output__new_line ();
                        P_locking_output__string(PSTR("Root passwd undefined."),
_true);
                        return _true;
                  }
                  else if (! fs_context.access_control.root) {
                        return _true;
                  }
            }

            return _false;
      }

      _bool fs_obj_descr__removed (fs_obj_descr * o) {

            if ( o->KID == KP_NAN &&
            valid_cmd_symbol(o->nA) &&
            valid_cmd_symbol(o->nB) )
                  return _true;

            return _false;
      }

      _bool fs_obj_descr__available (fs_obj_descr * o) {
            if ( fs_obj_descr__access_deny (o) ) {
                  locking_output__new_line ();
                  P_locking_output__string  (PSTR("Access  to  Object  denied."),
_true);
                  return _false;
            }
            if ( fs_obj_descr__removed (o) ) {
                  locking_output__new_line ();
                  P_locking_output__string (PSTR("Object is removed."), _true);
                  return _false;
            }

            return _true;
      }

      fs_obj_descr  *  fs_obj_descr__get_by_name  (byte_t  nA,  byte_t  nB,  byte_t
TYPE) {
            fs_obj_descr * o = fs_context.current_obj_buf.as_directory.items;
            for ( byte_t i = 0; i < (K_SIZE / sizeof(fs_obj_descr)); i++, o++ ) {

                  if ( o->nA == nA && o->nB == nB && o->TYPE == TYPE ) {
                        return o;
                  }
            }
            return _null;
      }

      fs_obj_descr * fs_obj_descr__get_by_KID (KID id, byte_t TYPE) {
            fs_obj_descr * o = fs_context.current_obj_buf.as_directory.items;
            for ( byte_t i = 0; i < (K_SIZE / sizeof(fs_obj_descr)); i++, o++ ) {
                  if ( o->KID == id && o->TYPE == TYPE ) {
                        return o;
                  }
            }
            return _null;
      }

      void fs_obj_descr__print_info (fs_obj_descr * o) {
            KID nid =  fs_kpointer__get_first_kid (o->KID);
```

```
	KID * kp = fs_context.intro.as_intro.kp;
	P_locking_output__string (PSTR("Name: "), _false);
	locking_output__byte (o->nA);
	locking_output__byte (o->nB);
	locking_output__new_line ();

	P_locking_output__string (PSTR("Kluster chain: "), _false);

	if ( kp [nid - 1] == KP_REM ) {
		P_locking_output__string (PSTR("removed"), _true);
	}

	else {
		do {
			locking_output__hex_byte (nid);
			locking_output__byte ('>');
			nid = kp [nid - 1];
		} while ( nid != KP_REM && nid != KP_END );

		locking_output__new_line ();
		if ( o->TYPE == obj_type__file ) {
			P_locking_output__string (PSTR("Size: "), _false);
			locking_output__hex_byte (fs_context.current_obj.SIZE);
			locking_output__new_line ();
		}
	}
}

/*
		INTRO KLUSTER POINTERS
*/

_bool fs_kpointer__obj_deleted (KID id) {
	if (* (((KID *) fs_context.intro.row) + id) == KP_REM)
	return _true;
	return _false;
}

KID fs_kpointer__get_first_kid (KID id) {
	if (id == INTRO_KID) return INTRO_KID;

	KID * p = (KID *) fs_context.intro.row;
	for ( byte_t i = ROOT_KID; i < K_COUNT; i++ ) {
		if ( p [i] == id ){
			return fs_kpointer__get_first_kid (p [i]);
		}
	}

	return id;
}

KID fs_kpointer__get_last_kid (KID id) {
	KID * kp = (KID *) fs_context.intro.as_intro.kp;
	KID res = id, cid = id;
	do{
		res = kp [cid - 1];
		if ( res == KP_END ) return cid;
		cid = res;
	} while (res != KP_REM && res != KP_NAN);

	return KP_NAN;
}

KID fs_kpointer__get_next_kid (KID id) {
	if ( id == KP_END ) return KP_END;
```

```
            return * (((KID *) fs_context.intro.row) + id);
      }

      void fs_kpointer__mark_removed (KID id) {
            KID * p = (KID *) fs_context.intro.as_intro.kp;
            KID pid = fs_kpointer__get_first_kid (id),
            nid = 0;
            byte_t buf = KP_REM;
            do {

                  nid = p [pid - 1];
                  fs_K__change
                  (& fs_context.intro, & fs_context.intro__memsync, pid, & buf,
sizeof(buf));

            } while ( nid != KP_END && nid != KP_REM);

            union intro_flags f;
            f.as_byte = fs_context.intro.as_intro.flags.as_byte;
            if (f.as_flags.fs_free_amount < K_COUNT) {
                  f.as_flags.fs_free_amount++;
                  fs_K__change
                  (& fs_context.intro, & fs_context.intro__memsync, 0, &
f.as_byte, sizeof(byte_t));
            }
      }

      /*
                  KLUSTER CHANGES SYNC
      */

      void fs_K__change (K_t * k, memsync_control * ms, kaddr_t addr, void *
data, byte_t s) {

            byte_t * p = k->row + addr;
            byte_t * d = (byte_t *) data;
            memsync_control mask = 1;
            for ( byte_t a = 0; a < s; a++, p++ ) {
                  // locking_output__hex_byte((addr - a));
                  * p = d [a];
                  mask <<= (addr + a);
                  * ms |= mask;
                  mask = 1;
            }
      }

      void fs_K__mark_changed (memsync_control * ms) {
            * ms = (memsync_control) (-1);
      }

      exit_status fs_K__get_changed (memsync_control * ms, memaddress * addr,
memaddress * size) {
            memaddress s = 0;
            memaddress a = 0;
            memsync_control flag = 1;

            // skip '\0' at memsync
            while ( flag && ! (flag & * ms) ) {
                  a++;
                  flag <<= 1;
            }

            // no sense if flag has been turned to '0' with <<
            while ( flag & * ms ) {
                  s++;
```

```
                * ms &= ~flag;
                flag <<= 1;
        }

        // if no '1' has been found
        if ( ! s ) {
                return exit_failure;
        }

        * addr += a;
        * size = s;

        return exit_success;
}

/*
        SIMPLE SCRIPT STEPs
*/

script_result failure_dumb () {
        put_prompt ();
        return script__exit;
}
```

kernel/filesystem/_fs_script__cat.c

```
/*
 * _fs_script__cat.c
 *
 * Created: 11/1/2021 10:01:32 PM
 *  Author: mkentrru
 */

#include "fs_scripts.h"

script_result fs__find__by_name () {
        fs_obj_descr * o =
        fs_obj_descr__get_by_name
        (fs_context.current_obj.nA,                fs_context.current_obj.nB,
fs_context.current_obj.TYPE);

        if ( o != _null ) {

                if (! fs_obj_descr__available (o) ) {
                        put_prompt ();
                        return script__error;
                }

                fs_context.current_obj.iROOT = o->iROOT;
                fs_context.current_obj.iUSER = o->iUSER;

                // keep id where obj was found
                tmp_descr.KID = fs_context.current_obj.KID;

                fs_context.current_obj.KID = o->KID;

                fs_context.current_obj.SIZE = o->SIZE;

                fs_context.descr_pointer = o;

                return 3; // to skip chain check and failure msg
        }

        return 1; // keep going
```

```
	}

	script_result fs__find__by_KID () {
		fs_obj_descr * o =
		fs_obj_descr__get_by_KID
		(fs_context.current_obj.KID, fs_context.current_obj.TYPE);

		if ( o != _null ) {

			if (! fs_obj_descr__available (o) ) {
				put_prompt ();
				return script__error;
			}

			fs_context.current_obj.iROOT = o->iROOT;
			fs_context.current_obj.iUSER = o->iUSER;

			fs_context.current_obj.KID = o->KID;

			fs_context.current_obj.SIZE = 0;
			return 3; // to skip chain check and failure msg
		}
		return 1; // keep going
	}


	script_result fs__find__failure () {
		locking_output__new_line ();
		P_locking_output__string  (PSTR("There  is  no  such  Object  at  this
Dir."), _true);
		locking_output__prompt ();
		return script__error;
	}

	script_result fs__cat__display () {
		for (byte_t i = 0; i < K_SIZE; i++) {
			locking_output__byte (fs_context.current_obj_buf.row [i]);
		}
		// locking_output__new_line ();
		return script__next;
	}

	script_result fs__cat__outro () {
		locking_output__new_line ();
		locking_output__prompt ();
		return script__next;
	}


	#define fs__cat__script_size 9
	script_result  (*  fs__cat__script_f  [fs__cat__script_size])  (struct  script
*)  = {
		fs__ls_entry,                   //      0:      current_obj.KID      =
current_folder.KID
		fs__ls_load_kluster,     // 1: load current_obj.KID Kluster
		fs__find__by_name,       // 2: if found jumps to (5)
		fs__ls_check_chain,             // 3: if chain ended jumps to (4)
		fs__find__failure,       // 4: message about failure, exit
		// found, displaying: current_obj_kid = file fkid
		fs__ls_load_kluster,     // 5: load file cluster
		fs__cat__display,        // 6: display loop
		fs__ls_check_chain,             // 7: if file chain ended jumps to (8)
		fs__cat__outro                  // 8: end msg
	};
```

```
	void fs__cat__script__init () {
		script__init	(&	fs__cat__script,	fs__cat__script_size,	_null,
fs__cat__script_f );
	}

	void fs__cat__script_task () {
		script__step (& fs__cat__script);
	}
```

kernel/filesystem/_fs_script__cd.c

```
	/*
	 * _fs_script__cd.c
	 *
	 * Created: 11/1/2021 10:05:53 PM
	 *  Author: mkentrru
	 */

	#include "fs_scripts.h"


	script_result fs__cd_entry () {
		//locking_output__hex_byte (fs_context.current_folder.iROOT);
		//locking_output__hex_byte (fs_context.current_folder.iROOT);
		//locking_output__new_line ();

		if ( ! fs_obj_descr__shared (& fs_context.current_obj) &&
		! fs_obj_descr__available (& fs_context.current_folder) ) {
			put_prompt ();
			return script__error;
		}

		fs_context.current_obj.KID =
		fs_kpointer__get_first_kid (fs_context.current_folder.KID);

		return script__next;
	}

	script_result fs__cd__found () {
		fs_context.current_folder.KID = fs_context.current_obj.KID;
		fs_context.current_folder.iROOT = fs_context.current_obj.iROOT;
		fs_context.current_folder.iUSER = fs_context.current_obj.iUSER;

		fs_context.current_folder.nA = fs_context.current_obj.nA;
		fs_context.current_folder.nB = fs_context.current_obj.nB;

		fs_context.current_folder.SIZE = 0;
		fs_context.current_folder.TYPE = obj_type__dir;

		if ( fs_context.current_folder.nA == '.' &&
		fs_context.current_folder.nB == '.' ) {
			//// look for '..' at current folder
			fs_context.current_folder.nA = ':';
			fs_context.current_folder.nB = '/';

			return script__next;
		}

		locking_output__new_line ();
		locking_output__prompt ();
		return script__exit;
	}
```

```
	script_result fs__cd__preparent_found () {

		fs_context.current_obj.KID =
		fs_kpointer__get_first_kid (fs_context.current_obj.KID);

		return script__next;
	}


	script_result fs__cd__parent_outro () {

		locking_output__prompt ();
		return script__exit;
	}

	script_result fs__cd_find_at_preparent () {

		fs_obj_descr * o =
		fs_obj_descr__get_by_KID
		(fs_context.current_folder.KID, obj_type__dir);

		if ( o != _null ) {

			//if (! fs_obj_descr__available (o) ) {
			//put_prompt ();
			//return script__error;
			//}

			fs_context.current_folder.nA = o->nA;
			fs_context.current_folder.nB = o->nB;
			fs_context.current_folder.iROOT = o->iROOT;
			fs_context.current_folder.iUSER = o->iUSER;


			P_locking_output__string (PSTR(" -> "), _false);
			locking_output__byte (fs_context.current_folder.nA);
			locking_output__byte (fs_context.current_folder.nB);

			locking_output__new_line ();
			locking_output__prompt ();
			return script__exit; // to skip chain check and failure msg
		}
		return 1; // keep going
	}

	script_result fs__cd__no_pre () {
		P_locking_output__string (PSTR(" -> "), _false);
		locking_output__byte (fs_context.current_folder.nA);
		locking_output__byte (fs_context.current_folder.nB);

		locking_output__new_line ();
		locking_output__prompt ();
		return script__exit;
	}

	#define fs__cd__script_size 16
	script_result (* fs__cd__script_f [fs__cd__script_size]) (struct script *)
= {
		fs__cd_entry,                  //      0:      current_obj.KID      =
current_folder.KID
		fs__ls_load_kluster,     // 1: load current_obj.KID Kluster
		fs__find__by_name,       // 2: if found jumps to (5)
		fs__ls_check_chain,            // 3: if chain ended jumps to (4)
		fs__find__failure,             // 4: message about failure, exit
		// current_folder = current_obj (parent)
```

```
            fs__cd__found,                // 5: save found object KID

            fs__cd_entry,                 // 6: if '..' look for name at parent
            fs__ls_load_kluster,     // 7: load parent Kluster
            fs__find__by_name,       // 8: find '..' at parent Kluster
            fs__ls_check_chain,           // 9: loop for Kluster chains
            fs__cd__no_pre,               // 10: cannot find '..' at parent dir
            //  current_obj  =  '..'  ->  preparent  (current_obj->KID  =  preparent-
>KID)
            fs__cd__preparent_found, // 11: preparent KID found! -> current_obj

            fs__ls_load_kluster,          //  12:  find  descriptor  of  parent  at
preparent
            fs__cd_find_at_preparent,     // 13: find '..' at parent Kluster
            fs__ls_check_chain,           // 14: loop for Kluster chains
            failure_dumb,                 // 15
      };

      void fs__cd__script__init () {
            script__init    (&    fs__cd__script,    fs__cd__script_size,    _null,
fs__cd__script_f );
      }

      void fs__cd__script_task () {
            script__step (& fs__cd__script);
      }
```

kernel/filesystem/_fs_script__create.c

```
      /*
       * _fs_script__create.c
       *
       * Created: 11/2/2021 12:46:03 AM
       *  Author: mkentrru
       */

      #include "fs_scripts.h"

      // !! at least 1 is needed !!
      #define btoks(a) (1 + (a / K_SIZE))
      fs_obj_descr tmp_descr;

      script_result fs__sync__current_K () {

            memaddress offset = 0;
            memaddress addr = ktoa(fs_context.current_obj.KID);
            memaddress size = 0;

            // if found no changes
            if ( fs_K__get_changed (& fs_context.current_obj__memsync, & offset,
& size ) )
            return script__next;

            mem__request (addr + offset, size, fs_context.current_obj_buf.row +
offset, current_task_id, _false);

            return script__stay;
      }

      #define valid_file_symbol(c) ((c >= 0x20 && c <= 0x7E))

      script_result fs__touch__entry (struct script * s) {

            byte_t size = 0;
```

```
            P_locking_output__string (PSTR("Create file with data: "), _false);
            char * d = (char *) s->data;
            if ( d != _null ) {
                  while ( valid_file_symbol (* d) && size < MAX_OBJ_SIZE ) {
                        locking_output__byte (* d);
                        d++;
                        size++;
                  }
            }

            P_locking_output__string (PSTR(" ("), _false);
            locking_output__hex_byte (size);
            locking_output__byte (')');
            locking_output__new_line ();
            if ( btoks(size) >
            fs_context.intro.as_intro.flags.as_flags.fs_free_amount ) {
                  P__put__string (PSTR("No free space for this size."), _true);
            }
            fs_context.current_obj.SIZE = size;

            fs_context.current_folder.KID =
            fs_kpointer__get_first_kid (fs_context.current_folder.KID);

            * fs_context.search_stack = 0;

            // to restore current folder after all
            * (fs_context.search_stack + 1) = fs_context.current_folder.KID;

            fs_context.descr_pointer = _null;

            return script__next;
      }

      script_result fs__create__check_same () {
            fs_obj_descr * s = fs_context.current_obj_buf.as_directory.items;
            for             (fs_obj_descr             *             o             =
fs_context.current_obj_buf.as_directory.items;
            o < s + (K_SIZE / sizeof(fs_obj_descr)); o++
            ){
                  if ( fs_obj_descr__good(o) ) {
                        if (
                        o->TYPE == fs_context.current_obj.TYPE &&
                        o->nA == fs_context.current_obj.nA &&
                        o->nB == fs_context.current_obj.nB
                        ) {
                              return 3;
                        }
                  }
                  else if ( fs_obj_descr__free (o) ) {
                        // fs_obj_descr__print (o);
                        fs_context.descr_pointer = o;
                        * fs_context.search_stack = fs_context.current_obj.KID;

                        // P_locking_output__string (PSTR("Found at: "), _false);
                        //KID t = fs_context.current_obj.KID;
                        //locking_output__hex_buffer((byte_t     *)     &     t,
sizeof(KID));
                  }
            }

            return script__next;
      }

      script_result fs__create__same_found () {
            P__put__string (PSTR("Same Object exists."), _true);
```

```
		locking_output__prompt ();
		return script__exit;
	}

	script_result simply_end () {
		fs_context.current_folder.KID = * (fs_context.search_stack + 1);
		locking_output__new_line ();
		locking_output__prompt ();
		return script__exit;
	}


	void fs_kpointer__reduce_free_value (byte_t count) {
		union intro_flags f;
		f.as_byte = fs_context.intro.as_intro.flags.as_byte;
		if (f.as_flags.fs_free_amount) {
			f.as_flags.fs_free_amount -= count;
			fs_K__change
			(&  fs_context.intro,  &  fs_context.intro__memsync,  0,  &
f.as_byte, sizeof(byte_t));
		}
	}


	KID fs__K__allocate (KID fid) {

		KID * kp = (KID *) fs_context.intro.as_intro.kp;
		KID nid = ROOT_KID + 1;
		KID pid = KP_END;

		while ( kp [nid - 1] != KP_REM &&
		kp [nid - 1] != KP_NAN ) {
			nid++;
			if ( nid >= K_COUNT ) {
				P_locking_output__string (PSTR("! There are no Klusters
left !"), _true);
				return KP_NAN;
			}
		}


		//kp [pid - 1] = nid;
		if ( fid != KP_NAN ) {
			pid = fs_kpointer__get_last_kid (fid);
			fs_K__change
			(& fs_context.intro, & fs_context.intro__memsync, pid, & nid,
sizeof(byte_t));
		}

		//kp [nid - 1] = KP_END;
		P_locking_output__string(PSTR("Allocated: "), _false);
		locking_output__hex_byte (nid);
		locking_output__new_line ();

		pid = KP_END;
		fs_K__change
		(&  fs_context.intro,  &  fs_context.intro__memsync,  nid,  &  pid,
sizeof(byte_t));

		fs_kpointer__reduce_free_value (1);

		return nid;
	}
```

```
	script_result fs__create__build_new_descr () {

		P_locking_output__string(PSTR("Building new descriptor."), _true);

		KID nid = KP_NAN;
		if ( * fs_context.search_stack ) fs_context.current_folder.KID = *
fs_context.search_stack;

		// means contains no free descr
		if ( fs_context.descr_pointer == _null ) {
			P_locking_output__string (PSTR("No free descriptors found."),
_true);

			// 0: allocate new kluster for Dir
			//fs_context.current_folder.KID =
			//fs__K__allocate (fs_context.current_folder.KID);
			nid = fs__K__allocate (fs_context.current_folder.KID);
			if ( nid == KP_NAN ) {
				return 12;
			}
			fs_context.current_folder.KID = nid;
			// 1: take first descriptor
			fs_context.descr_pointer	=	(fs_obj_descr	*)
fs_context.current_obj_buf.as_directory.items;
		}

		// fs_obj_descr__print (fs_context.descr_pointer);

		tmp_descr.iROOT = fs_context.current_obj.iROOT;
		tmp_descr.iUSER = fs_context.current_obj.iUSER;
		nid = fs__K__allocate (KP_NAN);
		if ( nid == KP_NAN ) return 12;
		tmp_descr.KID = nid;

		tmp_descr.nA = fs_context.current_obj.nA;
		tmp_descr.nB = fs_context.current_obj.nB;
		tmp_descr.SIZE = fs_context.current_obj.SIZE;
		tmp_descr.TYPE = fs_context.current_obj.TYPE;
		// fs_obj_descr__print (& tmp_descr);

		fs_context.current_obj.KID = fs_context.current_folder.KID;

		return 2;
	}


	script_result fs__create__write_new_descr () {

		memaddress offset =
		(byte_t	*)	fs_context.descr_pointer	-	(byte_t	*)
fs_context.current_obj_buf.as_directory.items;

		//locking_output__hex_buffer		((byte_t		*)		&
fs_context.current_obj_buf.row, sizeof(K_t));
		//locking_output__hex_buffer		((byte_t		*)		&
fs_context.current_obj__memsync, sizeof(memsync_control));

		fs_K__change
		(&	fs_context.current_obj_buf,	&	fs_context.current_obj__memsync,
offset,
		& tmp_descr, sizeof(fs_obj_descr));

		//locking_output__new_line ();
```

```
            //locking_output__hex_buffer          ((byte_t          *)          &
fs_context.current_obj_buf.row, sizeof(K_t));
            //locking_output__hex_buffer          ((byte_t          *)          &
fs_context.current_obj__memsync, sizeof(memsync_control));

            return script__next;
      }




      script_result fs__touch__write_data__setup () {
            fs_context.current_obj.KID = tmp_descr.KID;

            return script__next;
      }

      script_result fs__touch__write_data (struct script * s) {
            fs_obj_descr * o = & tmp_descr;
            byte_t * data = (byte_t *) s->data;

            byte_t a = 0;

            for ( a = 0; a < K_SIZE && a < o->SIZE; a++, data++ ) {
                  fs_context.current_obj_buf.row [a] = * data;
            }

            o->SIZE -= a;
            s->data = data;

            // zero any data left
            while (a < K_SIZE) {
                  fs_context.current_obj_buf.row [a++] = 0x00;
            }

            //       locking_output__hex_buffer       ((byte_t       *)       &
fs_context.current_obj_buf.row, sizeof (K_t));

            fs_K__mark_changed (& fs_context.current_obj__memsync);
            return script__next;
      }


      script_result fs__touch__write__check () {
            // P_locking_output__string(PSTR("Checking if data left."), _true);
            fs_obj_descr * o = & tmp_descr;
            KID nid = 0;
            if ( o->SIZE ) {
                  nid = fs__K__allocate(tmp_descr.KID);
                  tmp_descr.KID = nid;
                  return -4;
            }

            P_locking_output__string(PSTR("Data written to file."), _true);

            return script__next;
      }

      #define fs__touch__script_size 18
      script_result   (*   fs__touch__script_f   [fs__touch__script_size])   (struct
script *)  = {
            fs__touch__entry,                    // 0: cound size and set KID to
load
```

```
            fs__ls_entry,                               // 1: check access
            fs__ls_load_kluster,                // 2: load K
            fs__create__check_same,             // 3: if obj is same: +3; else:
+1;
            fs__ls_check_chain,                         // 4: if chain ended: +1;
else: -2;

            fs__create__build_new_descr,   // 5: get free obj_descr and allocate
K for Obj (+2 to skip error)
            fs__create__same_found,             // 6: msg 'found' and exit

            fs__ls_load_kluster,                // 7: load K witch contains free
obj descriptor
            fs__create__write_new_descr,   // 8: write new descr to current dir
            fs__sync__intro,                    // 9: sync intro
            fs__sync__current_K,                // 10: sync Dir

            /*
                  write data:

            0.    Set c_obj KID = new_obj_descr
            1.    Load c_obj
            2.    Write data to c_obj
            3.    Sync current K
            4.    If data left:
                        allocate KID to c_obj
                        goto 1
                  Else:
                        sync intro
            */
            fs__touch__write_data__setup,  // 11: get ready to write data
            fs__ls_load_kluster,                // 12: load current K
            fs__touch__write_data,              // 13: write data to current K
            fs__sync__current_K,                // 14: sync changes with memory
            fs__touch__write__check,       // 15: check if any data left
            fs__sync__intro,                    // 16: sync intro

            simply_end                                  // 17: simpy exit
      };


      void fs__touch__script__init (void * d) {
            script__init   (&   fs__touch__script,   fs__touch__script_size,   d,
fs__touch__script_f );
      }

      void fs__touch__script_task () {
            script__step (& fs__touch__script);
      }


      script_result fs__mkdir__entry (struct script * s) {

            P_locking_output__string (PSTR("Create Directory."), _true);
            fs_context.current_obj.SIZE = 0;

            fs_context.current_folder.KID =
            fs_kpointer__get_first_kid (fs_context.current_folder.KID);

            * fs_context.search_stack = 0;

            // to restore current folder after all
            * (fs_context.search_stack + 1) = fs_context.current_folder.KID;

            fs_context.descr_pointer = _null;
```

```
	return script__next;
}


script_result fs__mkdir__write_parent_descr__setup () {
	fs_context.current_obj.KID = tmp_descr.KID;

	return script__next;
}


script_result fs__mkdir__write_parent_descr () {
	fs_obj_descr * o = & tmp_descr;

	//	locking_output__hex_buffer	((byte_t	*)	&
fs_context.current_obj_buf.row, sizeof (K_t));
	o->iROOT = 0;
	o->iUSER = 0;
	o->KID = * (fs_context.search_stack + 1);
	o->nA = '.';
	o->nB = '.';
	o->SIZE = 0;
	o->TYPE = obj_type__dir;

	fs_K__change	(&	fs_context.current_obj_buf,	&
fs_context.current_obj__memsync,
	0, o, sizeof (fs_obj_descr));

	return script__next;
}


#define fs__mkdir__script_size 17
script_result	(*	fs__mkdir__script_f	[fs__mkdir__script_size])	(struct
script *)	= {
	fs__mkdir__entry,			// 0: cound size and set KID to
load

	fs__ls_entry,				// 1: check access
	fs__ls_load_kluster,			// 2: load K
	fs__create__check_same,			// 3: if obj is same: +3; else:
+1;
	fs__ls_check_chain,			// 4: if chain ended: +1;
else: -2;

	fs__create__build_new_descr,	// 5: get free obj_descr and allocate
K for Obj (+2 to skip error)
	fs__create__same_found,			// 6: msg 'found' and exit

	fs__ls_load_kluster,			// 7: load K witch contains free
obj descriptor
	fs__create__write_new_descr,	// 8: write new descr to current dir
	fs__sync__intro,			// 9: sync intro
	fs__sync__current_K,			// 10: sync Dir

	fs__mkdir__write_parent_descr__setup,// 11: get ready to write data
	fs__ls_load_kluster,			// 12: load current K
	fs__mkdir__write_parent_descr, // 13:
	fs__sync__current_K,			// 14:
	fs__sync__intro,			// 15: sync intro

	simply_end				// 16: simpy exit
};
```

```
	void fs__mkdir__script__init () {
		script__init  (&  fs__mkdir__script,  fs__mkdir__script_size,  _null,
fs__mkdir__script_f );
	}

	void fs__mkdir__script_task () {
		script__step (& fs__mkdir__script);
	}
```

kernel/filesystem/_fs_script__info.c

```
	/*
	 * _fs_script__info.c
	 *
	 * Created: 11/1/2021 10:23:03 PM
	 *  Author: mkentrru
	 */

	#include "fs_scripts.h"

	script_result fs__check__if_file () {
		if ( fs_context.current_obj.TYPE == obj_type__file )
		return script__next;

		// show dir info
		fs_obj_descr__print_info (& fs_context.current_obj);
		return 2;
	}


	script_result fs__info__show_file () {

		fs_obj_descr__print_info (& fs_context.current_obj);

		locking_output__prompt ();
		return script__exit;
	}


	byte_t fs__dir_info__obj_count = 0;

	script_result fs__sub__setup () {

		// fs_context.tmp_obj.KID = fs_context.current_folder.KID;

		fs_context.search_level = fs__sub__root_level;

		fs_context.search_stack [0] = fs_context.current_folder.KID;
		fs_context.search_stack            [fs__sub__root_level]            =
fs_context.current_obj.KID;
		fs_context.search_stack [fs__sub__root_level + 1] = 0;

		fs__dir_info__obj_count = 0;

		return script__next;
	}

	script_result fs__sub__entry () {
		// locking_output__string ("Going deeper: ", _false);

		fs_context.current_folder.KID =
		fs_context.search_stack [fs_context.search_level];

		fs_context.current_obj.KID = fs_context.current_folder.KID;
```

```
            //locking_output__hex_byte (fs_context.current_obj.KID);
            //locking_output__new_line ();

            return script__next;
      }


      script_result fs__sub__control () {

            //locking_output__hex_byte (fs_context.search_level);

            if ( fs_context.search_level <= fs__sub__root_level ) {

                  // fs_context.current_folder.KID = fs_context.tmp_obj.KID;
                  fs_context.current_folder.KID = fs_context.search_stack [0];

                  P_locking_output__string (PSTR("Total: "), _false);
                  locking_output__hex_byte (fs__dir_info__obj_count);

                  locking_output__new_line ();
                  locking_output__prompt ();
                  return script__next;
            }

            fs_context.search_level--;
            // locking_output__string ("Loop.", _true);
            return fs__sub__loop;
      }

      void fs__info_dir__printf_obj (fs_obj_descr * o, byte_t l) {
            for ( byte_t a = 0; a < l; a++ ) {
                  P_locking_output__string (PSTR(".."), _false);
            }

            locking_output__byte (o->nA);
            locking_output__byte (o->nB);

            if ( o->TYPE == obj_type__dir ) {
                  P_locking_output__string (PSTR(":"), _false);
            }

            locking_output__new_line ();
      }

      script_result fs__info_dir__foreach () {

            // Obj from prev dive
            KID      *      last_sub_obj      =      fs_context.search_stack      +
fs_context.search_level + 1;
            fs_obj_descr * s = fs_context.current_obj_buf.as_directory.items;

            for             (fs_obj_descr             *             o             =
fs_context.current_obj_buf.as_directory.items;
            o < s + (K_SIZE / sizeof(fs_obj_descr)); o++
            ){

                  if ( * last_sub_obj != 0 ) {

                        if ( * last_sub_obj == o->KID ) {
                              * last_sub_obj = 0;
                        }
                  }
```

```
                else if ( ! ( fs_obj_descr__empty(o) || fs_obj_descr__shared
(o) ) ) {
                        if ( fs_obj_descr__available (o) ) {
                                fs__info_dir__printf_obj                        (o,
fs_context.search_level);
                                fs__dir_info__obj_count++;

                                // if dir -> go deeper
                                if ( o->TYPE == obj_type__dir ) {

                                        * last_sub_obj = o->KID;
                                        * (last_sub_obj + 1) = 0;
                                        fs_context.search_level++;

                                        return fs__sub__dive;
                                }
                        }
                }
        }
        // locking_output__new_line ();
        return script__next;
    }


    #define fs__info__script_size 13
    script_result (* fs__info__script_f [fs__info__script_size]) (struct script
*)  = {
        fs__ls_entry,                   //      0:      current_obj.KID      =
current_folder.KID
        fs__ls_load_kluster,    // 1: load current_obj.KID Kluster
        fs__find__by_name,      // 2: if found jumps to (5)
        fs__ls_check_chain,             // 3: if chain ended jumps to (4)
        fs__find__failure,      // 4: message about failure, exit

        fs__check__if_file,             // 5: if file: +1; else: +2;
        fs__info__show_file,    // 6: show file info

        fs__sub__setup,                 // 7: setup sub search

        fs__sub__entry,                 // 8: select search level - Dir
        fs__ls_load_kluster,    // 9: load Dir K-n
        fs__info_dir__foreach,          // 10: work around Dir
        fs__ls_check_chain,             // 11: check Dir K-chain
        fs__sub__control        // 12: check if finished
    };


    void fs__info__script__init () {
        script__init   (&   fs__info__script,   fs__info__script_size,   _null,
fs__info__script_f );
    }

    void fs__info__script_task () {
        script__step (& fs__info__script);
    }
```

kernel/filesystem/_fs_script__ls.c

```
    /*
     * fs_script__ls.c
     *
     * Created: 11/1/2021 9:34:49 PM
     *  Author: mkentrru
     */
```

```
#include "fs_scripts.h"


/*
	// ls - list directory items
void ls ():
	KID id = current_folder_id;
	cid = get_first_kluster_id (id);

	do:
		load_kluster (id)
		foreach obj_descr o in current_obj_buf:
			print o
	while (get_next_kluster_id (id) != K_END)
*/


script_result fs__ls_entry () {

	if ( ! fs_obj_descr__available (& fs_context.current_folder) ) {
		put_prompt ();
		return script__error;
	}

	fs_context.current_obj.KID =
		fs_kpointer__get_first_kid(fs_context.current_folder.KID);

	return script__next;
}

script_result fs__ls_info () {
	P_locking_output__string (PSTR("Type   Owner   Name   FirstKID   Size
Removed"), _true);
	return script__next;
}

script_result fs__ls_load_kluster () {

	load_kluster (fs_context.current_obj.KID,
	(byte_t *) & fs_context.current_obj_buf.row, current_task_id);

	return script__next;
}

script_result fs__ls_display_obj () {
	// put_string ("display", _true);
	fs_obj_descr * s = fs_context.current_obj_buf.as_directory.items;
	for		(fs_obj_descr		*		o		=
fs_context.current_obj_buf.as_directory.items;
	o < s + (K_SIZE / sizeof(fs_obj_descr)); o++
	){
		//if ( ! fs_obj_descr__empty(o) )
			fs_obj_descr__print (o);
	}
	return script__next;
}
#define fs__ls_chain_loop -2
script_result fs__ls_check_chain () {
	// put_string ("loop", _true);
	KID nid = fs_kpointer__get_next_kid (fs_context.current_obj.KID);
	if ( nid != KP_END ) {
		fs_context.current_obj.KID = nid;
		return fs__ls_chain_loop;
	}
	return script__next;
```

```
	}

	script_result fs__outro () {

		locking_output__prompt ();
		return script__next;
	}

	#define fs__ls__script_size 6
	script_result (* fs__ls__script_f [fs__ls__script_size]) (struct script *)
= {
		fs__ls_entry,			//	allowed	by	input_handler_exec
(uart)
		fs__ls_info,
		fs__ls_load_kluster,	// disabled till memory read
		fs__ls_display_obj,		// allowed by mem script
		fs__ls_check_chain,
		fs__outro
	};

	void fs__ls__script__init () {
		script__init	(&	fs__ls__script,	fs__ls__script_size,	_null,
fs__ls__script_f );
	}

	void fs__ls__script_task () {

		script__step (& fs__ls__script);
	}
```

kernel/filesystem/_fs_script__rm.c

```
	/*
	 * _fs_script__rm.c
	 *
	 * Created: 11/1/2021 10:49:02 PM
	 *  Author: mkentrru
	 */

	#include "fs_scripts.h"

	script_result fs__sync__intro () {
		memaddress offset = 0;
		memaddress addr = ktoa(INTRO_KID);
		memaddress size = 0;

		// if found no changes
		if ( fs_K__get_changed (& fs_context.intro__memsync, & offset, & size
) )
			return script__next;
		// P_locking_output__string (PSTR("Sync intro."), _true);
		mem__request (addr + offset, size, fs_context.intro.row + offset,
current_task_id, _false);

		return script__stay;
	}



	#define rm__file__jump_to__sync 7
	script_result fs__rm__file () {
		if ( fs_obj_descr__shared (& fs_context.current_obj) ) {
			P_locking_output__string (PSTR("Access denied."), _true);
			return 11;
```

```
		}

		else {
			remove__obj (fs_context.descr_pointer);
		}

		return script__next;
	}

	script_result sync__on_file_rm () {
		// P_locking_output__string(PSTR("Sync on file remove!"), _true);
		// locking_output__hex_byte (tmp_descr.KID);
		return sync__K (9, tmp_descr.KID);
	}

	byte_t removed_count = 0;

	void remove__obj (fs_obj_descr * o) {

		// fs_obj_descr__print (o);

		if ( ! fs_obj_descr__removed (o) ) {
			fs_kpointer__mark_removed (o->KID);

			memaddress offset =
			(byte_t		*)		o		-		(byte_t		*)
fs_context.current_obj_buf.as_directory.items;

			//	locking_output__hex_buffer	((byte_t	*)	&	offset,
sizeof(memaddress));

			o->KID = KP_NAN;
			//		locking_output__hex_buffer		((byte_t		*)		&
fs_context.current_obj_buf.row, sizeof (K_t));
			// fs_obj_descr__print (o);

			fs_K__change
			(&				fs_context.current_obj_buf,				&
fs_context.current_obj__memsync, offset, o, sizeof(fs_obj_descr));

			//		locking_output__hex_buffer		((byte_t		*)		&
fs_context.current_obj_buf.row, sizeof (K_t));
			removed_count++;
		}
	}


	script_result fs__rm_dir__foreach () {

		// Obj from prev dive
		KID	*	last_sub_obj	=	fs_context.search_stack	+
fs_context.search_level + 1;
		fs_obj_descr * s = fs_context.current_obj_buf.as_directory.items;

		for		(fs_obj_descr		*		o		=
fs_context.current_obj_buf.as_directory.items;
		o < s + (K_SIZE / sizeof(fs_obj_descr)); o++
		){

			if ( * last_sub_obj != 0 ) {

				if ( * last_sub_obj == o->KID ) {
					// found last object
					* last_sub_obj = 0;
					// remove
```

```
                        remove__obj (o);
                    }
                }

                else if ( fs_obj_descr__good (o) &&
                    fs_context.search_level >= fs__sub__root_level ) {

                    if ( fs_obj_descr__available (o) ) {

                        if ( o->TYPE == obj_type__dir ) {

                            * last_sub_obj = o->KID;
                            * (last_sub_obj + 1) = 0;
                            fs_context.search_level++;

                            return 1;
                        }
                        else {

                            // remove
                            remove__obj (o);
                        }
                    }
                    else {
                        P__put__string  (PSTR("Have  no  access  to  remove
some Objects inside."), _true);
                        fs_context.search_level = 0;
                        return 4;
                    }
                }
            }
            // locking_output__new_line ();
            return 2;
        }

        script_result fs__rm_check__if_file () {
            removed_count = 0;

            if ( fs_obj_descr__shared (& fs_context.current_obj) ) {
                P_locking_output__string (PSTR("Filesystem Obj."), _true);
                locking_output__prompt ();
                return script__exit;
            }

            if ( fs_context.current_obj.TYPE == obj_type__file )
            return script__next;

            return 3;
        }

        script_result fs__rm_sub__control () {

            if ( fs_context.search_level < fs__sub__root_level ) {

                fs_context.current_folder.KID = fs_context.search_stack [0];

                P_locking_output__string (PSTR("Removed: "), _false);
                locking_output__hex_byte (removed_count);

                locking_output__new_line ();

                return script__next;
            }

            fs_context.search_level--;
```

```
            // locking_output__string ("Loop.", _true);
            return -6;
      }

      script_result sync__K (script_result on_exit, KID id) {
            memaddress offset = 0;
            memaddress addr = ktoa(id);
            memaddress size = 0;

            // if found no changes
            if ( fs_K__get_changed (& fs_context.current_obj__memsync, & offset,
& size ) )
                  return on_exit;

            //P_locking_output__string(PSTR("Address: "), _false);
            //locking_output__hex_buffer ((byte_t *) & addr, sizeof(memaddress));
            //P_locking_output__string(PSTR("Offset: "), _false);
            //locking_output__hex_buffer     ((byte_t     *)     &     offset,
sizeof(memaddress));
            //P_locking_output__string(PSTR("Size: "), _false);
            //locking_output__hex_buffer ((byte_t *) & size, sizeof(memaddress));

            mem__request (addr + offset, size, fs_context.current_obj_buf.row +
offset, current_task_id, _false);

            return script__stay;
      }



      script_result sync__on_dive () {
            // P_locking_output__string(PSTR("Sync on dive"), _true);
            //memaddress offset = 0;
            //memaddress addr = ktoa(fs_context.current_obj.KID);
            //memaddress size = 0;
            //
            //// if found no changes
            //if  (  fs_K__get_changed  (&  fs_context.current_obj__memsync,  &
offset, & size ) )
                  //return -3;
            //
            //mem__request (addr + offset, size, fs_context.current_obj_buf.row +
offset, current_task_id, _false);
            //
            // return script__stay;

            return sync__K (-3, fs_context.current_obj.KID);
      }

      script_result sync__on_chain () {
            // P_locking_output__string(PSTR("Sync on loop"), _true);
            //memaddress offset = 0;
            //memaddress addr = ktoa(fs_context.current_obj.KID);
            //memaddress size = 0;
            //
            //// if found no changes
            //if  (  fs_K__get_changed  (&  fs_context.current_obj__memsync,  &
offset, & size ) )
                  //return 1;
            //
            //mem__request (addr + offset, size, fs_context.current_obj_buf.row +
offset, current_task_id, _false);
            //
            // return script__stay;
```

```
		return sync__K (1, fs_context.current_obj.KID);
	}

	script_result fs__rm_check_chain () {
		KID nid = fs_kpointer__get_next_kid (fs_context.current_obj.KID);
		if ( nid != KP_END ) {
			fs_context.current_obj.KID = nid;
			return -4;
		}
		return script__next;
	}

	script_result fs__rm__outro () {
		locking_output__prompt ();
		return script__next;
	}


	memsync_control kpointers_in_use = 0;

	script_result intro_fix__entry () {
		kpointers_in_use = 0;
		return script__next;
	}

	script_result intro_fix__check_empty () {
		//locking_output__hex_byte (fs_context.current_obj.KID);
		//locking_output__new_line ();
		KID cid = fs_context.current_obj.KID;
		memsync_control mask = 1;

		mask <<= cid;

		fs_obj_descr * s = fs_context.current_obj_buf.as_directory.items;
		for (fs_obj_descr * o = fs_context.current_obj_buf.as_directory.items;
		o < s + (K_SIZE / sizeof(fs_obj_descr)); o++
		){
			if ( ! fs_obj_descr__empty(o) ) {
				// fs_obj_descr__print (o);
				kpointers_in_use |= mask;
			}
		}

		//locking_output__hex_buffer((byte_t*) & kpointers_in_use,
sizeof(memsync_control));
		//locking_output__hex_buffer((byte_t*) & mask,
sizeof(memsync_control));
	//
		//locking_output__new_line ();
		return script__next;
	}

	script_result intro_fix__remove_unused () {

		KID fid = fs_kpointer__get_first_kid (fs_context.current_folder.KID),
		nid = fs_kpointer__get_next_kid (fid),
		LAST_ALIVE = fid, buf = 0;

		//P_locking_output__string(PSTR("Current folder and fid: "), _false);
		//locking_output__hex_byte (fs_context.current_folder.KID);
		//locking_output__hex_byte (fid);
		//locking_output__new_line ();

		memsync_control mask = 1;
```

```
            P_locking_output__string (PSTR("Fixing unused Klusters."), _true);

            while ( fid != KP_END && fid != KP_REM ) {
                  mask = 1;
                  mask <<= fid;
                  //      locking_output__hex_buffer((byte_t*)      &      mask,
sizeof(memsync_control));

                  // Kluster is unused
                  if ( ! (kpointers_in_use & mask) ) {
                        if ( LAST_ALIVE ) {
                              P_locking_output__string  (PSTR("Not  in  use:  "),
_false);
                              locking_output__hex_byte                    (fid);
locking_output__new_line ();

                              // fs_context.intro.as_intro.kp [LAST_ALIVE - 1] =
KP_END;
                              buf = KP_END;
                              fs_K__change
                              (&  fs_context.intro,  &  fs_context.intro__memsync,
LAST_ALIVE, & buf, sizeof(KID));
                              buf = KP_REM;
                              fs_K__change
                              (&  fs_context.intro,  &  fs_context.intro__memsync,
fid, & buf, sizeof(KID));

                              //locking_output__hex_buffer    ((byte_t    *)    &
LAST_ALIVE, sizeof(KID));
                              //locking_output__hex_buffer    ((byte_t    *)    &
fs_context.intro.row, sizeof (K_t));
                        }
                  }
                  else {

                        kpointers_in_use &= ~(mask);
                        if ( LAST_ALIVE ) {
                              //fs_context.intro.as_intro.kp  [LAST_ALIVE  -  1]  =
fid;
                              buf = fid;
                              fs_K__change
                              (&  fs_context.intro,  &  fs_context.intro__memsync,
LAST_ALIVE, & buf, sizeof(KID));
                        }
                        LAST_ALIVE = fid;
                  }

                  fid = nid;
                  nid = fs_kpointer__get_next_kid (fid);
            }

            //fs_context.intro.as_intro.kp [LAST_ALIVE - 1] = KP_END;
            buf = KP_END;
            fs_K__change
            (& fs_context.intro, & fs_context.intro__memsync, LAST_ALIVE, & buf,
sizeof(KID));

            return script__next;
      }


      #define fs__rm__script_size 23
      script_result (* fs__rm__script_f [fs__rm__script_size]) (struct script *)
= {
```

```
            fs__ls_entry,                   //      0:      current_obj.KID      =
current_folder.KID
            fs__ls_load_kluster,     // 1: load current_obj.KID Kluster
            fs__find__by_name,       // 2: if found jumps to (5)
            fs__ls_check_chain,             // 3: if chain ended jumps to (4)
            fs__find__failure,       // 4: message about failure, exit

            fs__rm_check__if_file,          // 5: if file: +1; else: +2;
            fs__rm__file,                   // 6: show file info
            sync__on_file_rm,

            fs__sub__setup,                 // 7: setup sub search

            fs__sub__entry,                 // 8: select search level - Dir
            fs__ls_load_kluster,     // 9: load Dir K-n
            fs__rm_dir__foreach,     // 10: work around Dir

            sync__on_dive,

            sync__on_chain,
            fs__rm_check_chain,             // 11: check Dir K-chain

            // memsync and intro fix
            fs__rm_sub__control,            // 12: check if finished

            fs__ls_entry,
            fs__ls_load_kluster,
            intro_fix__check_empty,
            fs__ls_check_chain,

            intro_fix__remove_unused,

            fs__sync__intro,         // 13: sync RAM intro and EEPROM intro
            fs__rm__outro                   // 14: simple outro
      };


      void fs__rm__script__init () {
            script__init    (&    fs__rm__script,    fs__rm__script_size,    _null,
fs__rm__script_f );
      }

      void fs__rm__script_task () {
            script__step (& fs__rm__script);
      }
```