Príloha C: Hlavičkové súbory

Súbor access_types.h

```
#ifndef _ACCESS_TYPES_H
#define _ACCESS_TYPES_H

#define NR_ACCESS_TYPES 8

#define AT_MEMBER       0
#define AT_RECEIVE      1
#define AT_SEND         2
#define AT_SEE          3
#define AT_CREATE       4
#define AT_ERASE        5
#define AT_ENTER        6
#define AT_CONTROL      7

//      vs_t            vs[NR_ACCESS_TYPES][MAX_VS_BITS/32];

#endif /* _ACCESS_TYPES_H */
```

Súbor constable.h

```
#ifndef _CONSTABLE_H
#define _CONSTABLE_H

#include <stdlib.h>
#include <string.h>
#include "types.h"
#include "language/error.h"

#define MAX(a,b)        ((a)>(b)?(a):(b))
#define MIN(a,b)        ((a)<(b)?(a):(b))

int init_error( const char *fmt, ...  );
int runtime( const char *fmt, ...  );
int fatal( const char *fmt, ...  );

#endif /* _CONSTABLE_H */
```

Súbor event.h

```
#ifndef _EVENT_H
#define _EVENT_H

#include <asm/types.h>
#include "types.h"
#include "object.h"

int event_mask_clear( event_mask_t *e );
int event_mask_setbit( event_mask_t *e, int b );
int event_mask_clrbit( event_mask_t *e, int b );
int event_mask_and( event_mask_t *e, event_mask_t *f );
int event_mask_or( event_mask_t *e, event_mask_t *f );
int event_mask_sub( event_mask_t *e, event_mask_t *f );

struct event_handler_s;
struct event_hadler_hash_s;

struct event_type_s {
        struct event_type_s     *next;
        struct class_s          *op[2];
        struct class_s          *object;
        event_mask_t            mask;
        struct event_hadler_hash_s      *handlers_hash;
        struct medusa_optype_s  m;
        struct class_s          operation_class;
};

extern struct event_type_s *global_event_types;
int event_type_add( struct medusa_optype_s *m, struct medusa_attribute_s
*a );
int event_type_hash_create( void );
struct event_type_s *event_type_find( u_int16_t classid, u_int16_t opid
);
struct event_type_s *event_type_find_name( char *name, struct class_s
*s, struct class_s *o, struct class_s *ob );
struct event_type_s *event_type_findset( struct class_s *c );

struct event_context_s;
struct tree_s;

struct event_handler_s {
        struct event_type_s     *type;
```

```
        int(*handler)(struct event_handler_s *,struct event_context_s
*);
        struct object_s         *local_vars;
        char                    data[0];
};
#define ALL_OBJ ((void*)1)

struct event_hadler_hash_s {
        struct event_hadler_hash_s      *next;
        struct event_handler_s          *handler;
        struct tree_s                   *op[2];
        int                             recursive[2];
};

struct event_hadler_hash_s *evhash_add( struct event_hadler_hash_s
**hash, struct event_handler_s *handler );
#define evhash_foreach(ev,hash) for((ev)=(hash);(ev)!=NULL;(ev)=(ev)->next)

int register_event_handler( struct event_handler_s *h, struct tree_s
*subject, int recursive_subject, struct tree_s *object, int
recursive_object );

struct event_context_s {
        struct object_s operation;
        struct object_s subject;
        struct object_s object;
        int             result;
        int             first;
        struct object_s *local_vars;
};

#define RESULT_UNKNOWN  -1
#define RESULT_ALLOW    0       /* YES */
#define RESULT_DENY     1       /* NO */
#define RESULT_SKIP     2       /* SKIP */
#define RESULT_OK       3       /* OK */

/*
RESULT_UNKNOWN -> *
RESULT_ALLOW -> RESULT_DENY | RESULT_SKIP
RESULT_DENY -> -
RESULT_SKIP -> RESULT_DENY
RESULT_OK -> RESULT_ALLOW | RESULT_DENY | RESULT_SKIP
*/
int evaluate_result( int old, int new );
```

```
int do_event( void *data );

#endif /* _EVENT_H */
```

Súbor medusa_object.h

```
#ifndef _MEDUSA_OBJECT_H
#define _MEDUSA_OBJECT_H

//#include <medusa/config.h>

#define MEDUSA_ATTRNAME_MAX     27
#define MEDUSA_CLASSNAME_MAX    12
#define MEDUSA_OPNAME_MAX       16

/*
 * Used by l2, l3, l4
 *
 * This file defines
 *             CLASS (defined in l2, used by l3, l4)   medusa_class_s
 *             OBJECT (defined in l2, used by l3, l4)  medusa_object_s
 *             ATTRIBUTE (defined in l2, used by
l4)   medusa_attribute_s
 */

/**/
/**/

/* describes one attribute in class (defined in l2 code, used by l4) */
struct medusa_attribute_s {
        u_int16_t offset;              /* offset of attribute in object
*/
        u_int16_t length;              /* bytes consumed by data */
        u_int8_t type;                 /* data type (MED_TYPE_xxx) */
        char name[MEDUSA_ATTRNAME_MAX]; /* string:  attribute name */
};

#define MED_TYPE_END            0x00    /* end of attribute list */
#define MED_TYPE_UNSIGNED       0x01    /* unsigned integer attr */
#define MED_TYPE_SIGNED         0x02    /* signed integer attr */
#define MED_TYPE_STRING         0x03    /* string attr */
#define MED_TYPE_BITMAP         0x04    /* bitmap attr */
```

```
#define MED_TYPE_READ_ONLY      0x80    /* this attribute is read-only
*/
#define MED_TYPE_PRIMARY_KEY    0x40    /* this attribute is used to
lookup object */

/**/
/**/

/* description of class (defined in l2 code, used by l3, l4) */
struct medusa_class_s {
        u_int16_t       classid;        /* (1,2,...):  unique identifier
of this class */
        u_int16_t       size;           /* size of object */
        char            name[MEDUSA_CLASSNAME_MAX];
};

/* description of access/operation type */
struct medusa_optype_s {
        u_int16_t       classid;        /* class, which acts as a
subject of this operation  --- !!!  subject nieje vhodne slovo.  */
        u_int16_t       opid;           /* (1,2,...):  unique identifier
of access type within class */
        u_int16_t       size;
        char            name[MEDUSA_OPNAME_MAX];
        u_int16_t       op_class[2];
        char            op_name[2][MEDUSA_ATTRNAME_MAX];
};

#endif
```

Súbor object.h

```
#ifndef _OBJECT_H
#define _OBJECT_H

#include "types.h"
#include "medusa_object.h"

struct object_s;

struct class_set_s {
        struct class_set_s      *next;
        int(*handler)(struct object_s *);
```

```
};

struct class_s {
        struct class_s                  *next;
        u_int16_t                       cinfo_ofset;
        u_int16_t                       cinfo_size;     /* v 32 bit
slovach */
        u_int32_t                       cinfo_mask;
        struct class_set_s              *set;
        u_int16_t                       event_offset;
        u_int16_t                       event_size;     /* v bajtoch */
        struct medusa_class_s           m;
        struct medusa_attribute_s       attr[0];
};

struct object_s {
        struct object_s *next;
        struct medusa_attribute_s attr;
        struct class_s  *class;
        char    *data;
};

struct class_s *get_class_by_name( char *name );
struct class_s *get_class_by_id( u_int16_t id );
struct medusa_attribute_s *get_attribute( struct class_s *c, char *name
);
struct class_s *add_class( struct medusa_class_s *mc, struct
medusa_attribute_s *a );
int class_alloc_cinfo( struct class_s *c );
int class_add_sethandler( struct class_s *c, int(*handler)(struct
object_s *) );
int object_is_same( struct object_s *a, struct object_s *b );

int object_clear_event( struct object_s *o );
int object_add_event( struct object_s *o, event_mask_t *e );
int object_set_inherit( struct object_s *o );
int object_clr_inherit( struct object_s *o );

int object_do_sethandler( struct object_s *o );

#endif
```

Súbor procmem.h

```
#ifndef _PROCMEM_H
#define _PROCMEM_H

#include "object.h"
#include "tree.h"
#include "event.h"

int procmem_getpid( struct object_s *subject );
int procmem_readstr( struct object_s *subject, u_int32_t addr, char
*buf, int maxlen );
int procmem_writestr( struct object_s *subject, u_int32_t addr, char
*buf );

int procmem_init( void );

#endif /* _PROCMEM_H */
```

Súbor subject.h

```
#ifndef _SUBJECT_H
#define _SUBJECT_H

#include "access_types.h"
#include "vs.h"

struct subject_s {
        vs_t    vs[NR_ACCESS_TYPES][MAX_VS_BITS/32];
        vs_t    recursive_vs[NR_ACCESS_TYPES][MAX_VS_BITS/32];
};

struct subject_get_s {
        struct subject_get_s *next;
        int(*get)(vs_t *vs,int n,struct object_s *o);
};

int subject_register( int(*get)(vs_t *vs,int n,struct object_s *o) );
int subject_get_vs( vs_t *vs, int n, struct object_s *o );

#endif /* _SUBJECT_H */
```

Súbor target.h

```
#ifndef _TARGET_H
#define _TARGET_H

#include "tree.h"
#include "event.h"
#include "access_types.h"

extern struct target_s *global_targets;

typedef struct ltree_s {
        struct ltree_s  *prev;
        struct tree_s   *tree;
        int             recursive;
} ltree_t;

typedef struct levent_s {
        struct levent_s         *prev;
        struct event_handler_s  *handler;
        struct target_s         *subject;
        struct target_s         *object;
} levent_t;

struct target_s {
        struct target_s *next;
        struct vs_s     *my_vs;
        vs_t            vs[NR_ACCESS_TYPES][MAX_VS_BITS/32];
        levent_t        *levent;
        ltree_t         *ltree;
        int             primary;
        char            name[0];
};

struct target_s *target_create( char *name, int primary );
struct target_s *target_find( char *name );
int target_add_path( struct target_s *target, int recursive, struct
tree_s *path );
int target_add_event( struct event_handler_s *handler, struct target_s
*subject, struct target_s *object );
int target_add_vs( struct target_s *target, int which, vs_t *vs );
vs_t *target_get_vs( struct target_s *target );


#endif /* _TARGET_H */
```

Súbor tree.h

```
#ifndef _TREE_H
#define _TREE_H

#include "vs.h"
#include "event.h"

struct tree_s;
struct target_s;

struct tree_type_s {
        char    *name;
        int     size;           /* size of node */
        struct class_s  *obj_class;     /* pre eventy */
        int(*init)(struct tree_s*);
        struct tree_type_s *child_type;
        int(*add_vs)(struct tree_s*,int recursive,int which,const vs_t
*vs);
};

struct tree_s {
        struct tree_s   *parent;
        struct tree_s   *child;         /* normalny potomkovia */
        struct tree_s   *reg;           /* potomkovia s regexpami */
        struct tree_s   *next;
        struct tree_s   *alt;           /* alternative */
        struct tree_type_s *type;
        char            *name;
        void            *extra;         /* precompiled regexp...  */
        struct target_s *primary_target;
        struct target_s *recursive_primary_target;
        event_mask_t    event;
        event_mask_t    recursive_event;
};

typedef struct tree_type_s tree_type_t;

extern struct tree_s global_root;

int register_tree_type( tree_type_t *type );
struct tree_s *create_path( char *path );
```

```
struct tree_s *find_one( struct tree_s *base, char *name );
int tree_is_offspring( struct tree_s *offspring, struct tree_s *ancestor
);

int tree_add_event( struct tree_s *p, int recursive, struct
event_handler_s *h );

int path_add_vs( char *path, int recursive, int which, const vs_t *vs );

char *tree_get_path( struct tree_s *t );

#endif
```

Súbor types.h

```
#ifndef _TYPES_H
#define _TYPES_H

#include <sys/types.h>

typedef u_int32_t event_mask_t;

#ifndef NULL
#define NULL    ((void*)0)
#endif

#endif /* _TYPES_H */
```

Súbor vs.h

```
#ifndef _VS_H
#define _VS_H

#include <asm/types.h>

#define MAX_VS_BITS     64

typedef __u32   vs_t;

struct vs_s {
        struct vs_s     *next;
```

```
        vs_t vs[MAX_VS_BITS/32];
        char name[0];
};

int vs_init( int max_vs );
struct vs_s *vs_alloc( char *name );
struct vs_s *vs_find( char *name );

void vs_clear( vs_t *to );
void vs_invert( vs_t *to );
void vs_set( const vs_t *from, vs_t *to );
void vs_add( const vs_t *from, vs_t *to );
void vs_sub( const vs_t *from, vs_t *to );
void vs_mask( const vs_t *from, vs_t *to );

int vs_test( const vs_t *test, const vs_t *vs );

#endif
```

Súbor basic/fs.h

```
#ifndef _FS_H
#define _FS_H

#include "../tree.h"
#include "../vs.h"

struct fs_s {
        struct tree_s   tree;
        struct {        /* kdev_t dev; */       /* ???  */
                        unsigned long ino;
               }        inode;
        vs_t vs[MAX_VS_BITS/32];                /* pre nastavovanie */
        vs_t recursive_vs[MAX_VS_BITS/32];      /* pre nastavovanie */
};

int fs_init( void );

#endif
```

Súbor basic/proc.h

```
#ifndef _PROC_H
#define _PROC_H

#include "../tree.h"
#include "../vs.h"
#include "../access_types.h"
#include "../subject.h"

struct proc_s {
        struct tree_s           tree;
        struct subject_s        s;
};

int proc_init( void );


#endif
```

Súbor comm0/comm.h

```
#ifndef _COMM_H
#define _COMM_H

#include "medusa.h"

int comm_vs_size( void );
char *comm_cmd(struct medusa_packet *msg, char *data);
int comm_open( char *filename );
int comm_do( void );

#endif /* _COMM_H */
```

Súbor comm0/medusa.h

```
/* Medusa DS9 header file
 */

#ifndef _MEDUSA_H_
#define _MEDUSA_H_

#define CONFIG_MEDUSA_SYSCALL
#define CONFIG_MEDUSA_FORCE
// #define CONFIG_MEDUSA_LINUX_CAPABILITIES

#define __KERNEL__
#define _LINUX_FS_H 1
#include <linux/capability.h>
#undef _LINUX_FS_H
#include <linux/kdev_t.h>
#undef __KERNEL__
#include <linux/sys.h>
#include <linux/types.h>


/* extension to struct task_struct */
struct med_proc {
        __u32 med_vs;           /* bitmap of virtual spaces (member) */
        __u32 med_vs_s;         /* see */
        __u32 med_vs_r;         /* read */
        __u32 med_vs_w;         /* write */
        __u16 med_act;          /* actions which need acknowledgement
from constable */
        __u16 med_iact;         /* iactions which need acknowledgement
from constable */
        __u32 med_user;         /* for free daemon use */
#ifdef CONFIG_MEDUSA_SYSCALL
        __u32 med_syscall[NR_syscalls / (sizeof(__u32) * 8)];   /*
bitmap of syscalls, which are reported */
#endif
#ifdef CONFIG_MEDUSA_FORCE
        void *force_code;       /* code to froce or NULL, kfree */
        int force_len;          /* force code length */
#endif
        __u32 cinfo[4];         /* info for constable */
};

/* extension to struct inode */
```

```
struct med_inode {
        __u32 med_vs;           /* bitmap of virtual spaces (member) */
        __u16 med_act;          /* actions which need acknowledgement
from constable */
#ifdef CONFIG_MEDUSA_LINUX_CAPABILITIES
        kernel_cap_t icap;              /* support for Linux
capabilities */
        kernel_cap_t pcap;
        kernel_cap_t ecap;
#endif /* CONFIG_MEDUSA_LINUX_CAPABILITIES */
        u_long cinfo;           /* info for constable */
        __u16 magic;            /* whether is it actual */
};

struct iattr;

#define MEDUSA_MAJOR    111
#define MP_SIZE         sizeof(struct medusa_packet)
#define BEGIN_VS        (0xffffffff)
#define ALL_VS          (0xffffffff)
#define NONE_VS         (0x0)
#define MIN_DATA_LEN    512
#define MAX_DATA_LEN    2048
#define B_SIZE          sizeof(struct communication_bufffer)

/* process actions
 * + SKIP - if answer is SKIP then operation doesn't execute but
 * it returns success, otherwise, SKIP does same as OK */
#define MPACT_FORK      0x0001  /* clone_flags */
#define MPACT_EXEC      0x0002  /* attempt - 0, done - 1 */
#define MPACT_EXIT      0x0004
#define MPACT_SETUID    0x0008  /* setuid, setreuid, !setgid, !setgroups
...  */
#define MPACT_KILL      0x0010  /* send signal, + SKIP */
#define MPACT_SOCKET    0x0020  /* socketcall ...  */
#define MPACT_PTRACE    0x0040  /* stop and send signal to be ptraced */
#define MPACT_PERM      0x0100  /* is in permission function */
#define MPACT_CH        0x0200  /* is in notify_change function */
#define MPACT_IOP       0x0400  /* it does inode operation (MIACT_) */
#define MPACT_CAP       0x0800  /* capable */
#define MPACT_MODULE    0x1000  /* create_module, init_module,
delete_module */
#define MPACT_SEXEC     0x2000  /* target = new uid, inof1 = inode vs
info2 =rcinfo + SKIP */
```

```
#define MPACT_MAY_CD    0x8000  /* temporarily, it can access all dirs
*/

struct m_proc_inf {
        int pid, parent_pid, child_pid, sibling_pid;    /* older sib.
*/
        int pgrp;
        unsigned short uid, euid, suid, fsuid;
        unsigned short gid, egid, sgid, fsgid;
        unsigned short luid;
        kernel_cap_t cap_effective, cap_inheritable, cap_permitted;
        struct med_proc mp;
};

/* inode actions
   (MED_YES doesn't work with inode ops yet, it defaults to MED_OK) */
#define MIACT_ACCESS    0x0001  /* if it finds this
                                   inode, it can redirect */
#define MIACT_CREATE    0x0002  /* + SKIP */
#define MIACT_LINK      0x0004  /* + SKIP */
#define MIACT_UNLINK    0x0008  /* */
#define MIACT_SYMLINK   0x0010  /* */
#define MIACT_MKDIR     0x0020  /* */
#define MIACT_RMDIR     0x0040  /* + SKIP */
#define MIACT_MKNOD     0x0080  /* */
#define MIACT_RENAME    0x0100  /* + SKIP */
#define MIACT_READLINK  0x0200  /* !!!  not yet, but can be helpful */
#define MIACT_FOLLOW    0x0400  /* !!!  what for ???  */
#define MIACT_TRUNCATE  0x0800  /* + SKIP */
#define MIACT_PERMISSION        0x1000  /* + SKIP */
#define MIACT_EXEC      0x2000  /* (inode, argv,envp, path), redir */
#define MIACT_INHERIT   0x4000  /* - inherit inode attributes */
#define MIACT_VALID     0x8000  /* - information is valid */

struct m_inode_inf {
        kdev_t dev;
        unsigned long ino;
        __kernel_mode_t mode;
        __kernel_nlink_t nlink;
        __kernel_uid_t uid;
        __kernel_gid_t gid;
        kdev_t rdev;
        struct med_inode mi;
};
```

```
/* cmd */
#define MED_VERSION     0
#define MED_IGET        1       /* Read information for inode */
struct medreq_iget {
        struct m_inode_inf inode;
        struct m_inode_inf parent;
        /* + name */
};
struct medans_iget {
        struct m_inode_inf inode;
};

#define MED_IACT        2       /* Report inode action */
struct medreq_iact {
        __u16 pact;             /* process action (0= inode action/
process action) */
        __u16 act;              /* sub-action which process attempts to
execute
                                 */
        struct m_inode_inf inode;
        struct m_proc_inf proc;
        __u32 info1;            /* additional information */
        __u32 info2;            /* additional information */
        /* +
         * data
         * (name)
         * */
};
struct medans_iact {
        struct m_proc_inf proc;
        /* ...  ???  */
        /* + redirection(pathname) */
};

#define MED_PACT        3       /* Report process action */
struct medreq_pact {
        __u16 act;              /* action which process attempts to
execute */
        struct m_proc_inf proc;
        struct m_proc_inf target;
        __u32 info1;            /* additional information */
        __u32 info2;            /* additional information */
};
struct medans_pact {
        struct m_proc_inf proc;
```

```
};

#define MED_TRACE       4       /* Report syscall */
struct medreq_trace {
        __u16 syscall;          /* syscall */
        struct m_proc_inf proc;
        __u32 arg[5];
};
struct medans_trace {
        struct m_proc_inf proc;
        __u32 arg[5];
        __u32 retval;           /* valid only when answer is MED_NO */
};


/* commands cmd to kernel from daemon */
#define MED_RESET       0x100
#define MED_GET_PROC    0x101
#define MED_SET_PROC    0x102
#define MED_FOREACH_PROC        0x103
#define MED_FORCE       0x104
#define MED_GETSTR      0x110
#define MED_PRINTK      0x111
struct medreq_cmd {
        __u32 arg[8];
        struct m_proc_inf proc;
};
struct medans_cmd {
        __u32 arg[8];
        struct m_proc_inf proc;
};

/* answer codes */
#define MED_ERR         -1      /* error */
#define MED_YES         0       /* permit operation */
#define MED_NO          1       /* fobid operation */
#define MED_SKIP        2       /* forbid operation, but return success
*/
#define MED_OK          3       /* permit operation, but proceed with
                                   standard system permission check if
any */

struct medusa_packet {
        int id;
        int cmd;
```

```
        int answer;
        int data_len;           /* kernel->daemon, daemon->kernel */
        union {
                struct medreq_iget r_iget;
                struct medans_iget a_iget;
                struct medreq_iact r_iact;
                struct medans_iact a_iact;
                struct medreq_pact r_pact;
                struct medans_pact a_pact;
                struct medreq_trace r_trace;
                struct medans_trace a_trace;
                struct medreq_cmd r_cmd;
                struct medans_cmd a_cmd;
        } u;
};

#define med_c_valid(MI) (((MI)->med_act & MIACT_VALID) ? 1 :  0)

#endif                          /* _MEDUSA_H_ */
```

Súbor domain/domain.h

```
#ifndef _DOMAIN_H
#define _DOMAIN_H

#include "../tree.h"
#include "../vs.h"
#include "../access_types.h"
#include "../subject.h"

struct domain_s {
        struct tree_s           tree;
        struct subject_s        s;
};

int domain_init( void );


#endif
```

Súbor language/error.h

```
#ifndef _ERROR_H
#define _ERROR_H

extern char *errstr;
extern char *Out_of_memory;

int error( const char *fmt, ...  );
int warning( const char *fmt, ...  );

#endif /* _ERROR_H */
```

Súbor language/execute.h

```
#ifndef _EXECUTE_H
#define _EXECUTE_H

#include "../event.h"

#define MAX_REG_SIZE    4096

struct register_s {
        struct medusa_attribute_s *attr;
        struct medusa_attribute_s tmp_attr;
        char *data;
        char buf[MAX_REG_SIZE];
};

void r_imm( struct register_s *r );
void r_sto( struct register_s *v, struct register_s *d );
void r_resize( struct register_s *v, int size );

int strnlen( char *s, int n );

void r_neg( struct register_s *v );
void r_not( struct register_s *v );
int r_nz( struct register_s *v );

void do_bin_op( int op, struct register_s *v, struct register_s *d );

u_int32_t data_load_var( struct event_context_s *c, char *name, char
*attr, struct medusa_attribute_s **a );
```

```
int execute_handler( struct event_handler_s *h, struct event_context_s
*c );

#endif /* _EXECUTE_H */
```

Súbor language/language.h

```
#ifndef _LANGUAGE_H
#define _LANGUAGE_H

#include <mmm/compiler.h>
#include <mmm/lex.h>

enum {
        Tpp = T|0x100,          /* ++ */
        Tmm,                    /* -- */
        TshiftL,                /* << */
        TshiftR,                /* >> */
        Tle,                    /* <= */
        Tge,                    /* >= */
        Teq,                    /* == */
        Tne,                    /* != */
        Tand,                   /* && */
        Txor,                   /* ^^ */
        Tor,                    /* || */

        Tif,                    /* if */
        Telse,                  /* else */
        Twhile,                 /* while */
        Tdo,                    /* do */
        Tfor,                   /* for */
        Tbreak,                 /* break */
        Tcontinue,              /* continue */
        Tswitch,                /* switch */
        Tcase,                  /* case */
        Tdefault,               /* default */
        Treturn,                /* return */
        Tgoto,                  /* goto */

        /* conf.  language */
        Tprimary,               /* primary */
        Ttarget,                /* target */
```

```
        Trecursive,             /* recursive */
        Taccess,                /* <access type> */

        T_id,                   /* <identifikator> */
        T_str,
        T_num,
};


enum {
        oNOP = 0|0x100,
        oLD0,
        oLD1,
        oLDI,
        oLTI,
        oLTS,
        oSTO,
        oSWP,
        oDEL,
        oIMM,
        oLDV,
        oLDS,
        oDUP,
        oADD,   /* poradie oADD - oXOR sa nesmie zmenit!  */
        oSUB,
        oMUL,
        oDIV,
        oMOD,
        oSHL,
        oSHR,
        oLT,
        oGT,
        oLE,
        oGE,
        oEQ,
        oNE,
        oOR,
        oAND,
        oXOR,
        oNOT,
        oNEG,

        oJR,
        oJIZ,
        oJNZ,
```

```
        oJSR,
        oRET,
};

#define START   N|0x0a00
#define SRET0   N|0x0aff
#define CMD     N|0x0c00
#define CMDS    N|0x0c30
#define S_val   N|0x0d00
#define S_exp   N|0x0e00
#define S_exp2  N|0x0e01
/* N|0x0e00-0x0e40 - reserved for expressions */

extern struct compile_tab_s conf_lang[];
extern struct compile_tab_s language[];
extern struct compile_tab_s value[];
extern struct compile_tab_s expression[];

extern lextab_t operators[];
extern lextab_t keywords[];
extern lexstattab_t lex_tab[];

extern struct compiler_err_class s_error;

#endif _LANGUAGE_H
```

Súbor language/variables.h

```
#ifndef _VARIABLES_H
#define _VARIABLES_H

#include "../object.h"
#include "../types.h"

struct sval_s {
        u_int32_t       type;
        u_int32_t       data;
};

struct object_s *alloc_var( char *name, struct medusa_attribute_s *attr,
struct class_s * class );
struct object_s *var_link( struct object_s **v, struct object_s *var );
struct object_s *get_var( struct object_s *v, char *name );
```

```
int var_to_sval( struct object_s *v, char *atr, struct sval_s *s );

#endif /* _VARIABLES_H */
```

Súbor rbac/rbac.h

```
#ifndef _RBAC_H
#define _RBAC_H

#include "../tree.h"
#include "../subject.h"
#include <sys/types.h>

struct user_assignment_s;
struct permission_assignment_s;
struct hierarchy_s;

struct user_s {                         /* U - users */
        struct user_s *next;
        struct user_assignment_s *ua;
        uid_t uid;
//      vs_t vs[MAX_VS_BITS/32];
        char name[0];
};

struct role_s {                         /* R - roles */
        struct role_s *next;
        int flag;                       /* for rbac_save */
        struct user_assignment_s *ua;
        struct permission_assignment_s *perm;
        struct hierarchy_s *sup;
        struct hierarchy_s *sub;
        vs_t vs[NR_ACCESS_TYPES][MAX_VS_BITS/32]; /* PA - permission
assignment */
//      vs_t recursive_vs_sup[NR_ACCESS_TYPES][MAX_VS_BITS/32];
//      vs_t recursive_vs_sub[NR_ACCESS_TYPES][MAX_VS_BITS/32];
        char name[0];
};

                                        /* P - permissions - existuju */

struct user_assignment_s {              /* UA - user assignment */
        struct user_assignment_s *next_user;
```

```
        struct user_assignment_s *next_role;
        struct user_s *user;
        struct role_s *role;
};

struct permission_assignment_s {
        struct permission_assignment_s *next;
        int n;
        int access[8];
        struct target_s *target[8];
};

struct session_s {                    /* S - sessions */
        struct session_s *next;
        struct user_s *user;
        struct user_assignment_s *roles;        /* osobitny zoznam */
};

struct hierarchy_s {                  /* H - hierarchy */
        struct hierarchy_s *next_sup;
        struct hierarchy_s *next_sub;
        struct role_s *sup_role;
        struct role_s *sub_role;
};


/* Obmedzenia */


/* Funkcie */

extern struct user_s *rbac_users;
extern struct role_s *rbac_roles;

struct user_s * rbac_user_add( char *name, uid_t uid );
struct user_s * rbac_user_find( char *name );
struct user_s * rbac_user_find_by_uid( uid_t uid );
//int rbac_all_users_add_vs( const vs_t *vs );
struct role_s * rbac_role_add( char *name );
int rbac_role_del( struct role_s *role );
struct role_s * rbac_role_find( char *name );
int rbac_roles_reinit_vs( void );
//int rbac_all_roles_add_vs( const vs_t *vs );
int rbac_add_ua( struct user_s *user, struct role_s *role );
int rbac_del_ua( struct user_s *user, struct role_s *role );
```

```
//void rbac_inherit_sub( struct role_s *r );
void rbac_inherit_sup( struct role_s *r );
int rbac_set_hierarchy( struct role_s *sup_role, struct role_s *sub_role
);
int rbac_del_hierarchy( struct role_s *sup_role, struct role_s *sub_role
);
int rbac_role_add_perm( struct role_s *role, int which, struct target_s
*t );
int rbac_role_del_perm( struct role_s *role, int which, struct target_s
*t );

/*
              +- users --- <user> \\  --- <role>
              |
              +- roles --- <role> \\ --- <user>
              |
        rbac -+- role --- <role> --- <role> ...
              |
              +- ROLE --- <role> --- <role> ...
              |
              +- permission --- <target> ---
[RECEIVE,SEND,SEE,CREATE,ERASE,ENTER,CONTROL]


 rbac/role - rekurzivne clenstvo smerom k podrolam
 rbac/ROLE - rekurzivne clenstvo smerom k nadrolam (je to vobec potrebne
)
 - schopnosti vzdy smerom k nadrolam.

Pravo:  priradene roli v konf subore:
        dvojica:  typ,target
        typ:  AT_RECEIVE AT_SEND AT_SEE AT_CREATE AT_ERASE AT_ENTER
AT_CONTROL

        AT_CONTROL rbac/user/<user>     - pravo priradovat pouzivatelovi
role
        AT_CONTROL rbac/role/<rola>     - pravo pridavat role
pouzivatelov
        AT_CONTROL rbac/role/<rola>     - pravo pridavat/uberat prava
        AT_CONTROL rbac/permission      - pravo pridavat/uberat prava

        AT_CREATE rbac/role             - pravo vytvarat role
        AT_CREATE rbac/ROLE             - pravo vytvarat role
        AT_CREATE rbac/role/<rola>/...  - pravo vytvarat podrole
        AT_CREATE rbac/ROLE/<rola>/...  - pravo vytvarat nadrole
```

```
        AT_ERASE - ako AT_CREATE ale rusit.
 */

struct rbac_user_s {            /* <user> */
        struct tree_s   tree;
        struct user_s   *user;
        vs_t vs[MAX_VS_BITS/32];
};

struct rbac_rusers_s {          /* rbac/users */
        struct tree_s   tree;
        vs_t vs[MAX_VS_BITS/32];
        vs_t recursive_vs[MAX_VS_BITS/32];
};

struct rbac_role_s {            /* <role> */
        struct tree_s   tree;
        vs_t vs[MAX_VS_BITS/32];
        vs_t recursive_vs[MAX_VS_BITS/32];
};

struct rbac_rrole_s {           /* rbac/role */
        struct tree_s   tree;
        vs_t vs[MAX_VS_BITS/32];
        vs_t recursive_vs[MAX_VS_BITS/32];
};

struct rbac_perm_s {            /* rbac/permission */
        struct tree_s   tree;
        vs_t vs[MAX_VS_BITS/32];
};

int rbac_get_user_vs( vs_t *vs, struct user_s *user );
int rbac_get_role_vs( vs_t *vs, struct role_s *role );

extern struct rbac_rusers_s *r_user;
extern struct rbac_rrole_s *r_role,*r_ROLE;
extern struct rbac_perm_s *r_perm;

int rbac_init( void );
int rbac_save( char *file, int rotate );
int rbac_adm( struct object_s *subject, char *op, char *s1, char *s2,
char *s3 );

#endif /* _RBAC_H */
```