Wichtige NR's

```
NR_REGS
NR_SEGS
NR_TASKS
NR_PROCS
NR_PROCS
NR_HOLES
```

Wichtige Strukturen

```
struct mem_map {
    vir_clicks mem_vir;
    phys_clicks mem_phys;
    vir_clicks mem_len;
};

EXTERN struct proc {
    int p_reg [NR_REGS];
    ...
    struct mem_map [NR_SEGS];
    ...
} proc [NR_TASKS + NR_PROCS];

PRIVATE struct hole {
    phys_clicks h_base;
    phys_clicks h_len;
    struct hole *h_next;
} hole [NR_HOLES];
```

```
        typedef     unsigned        vir_bytes;
        typedef     unsigned        vir_clicks;
        typedef     long            phys_bytes;
        typedef     unsigned        phys_clicks;
        typedef     int             signed_clicks;
```

Wichtig

1. Kernel
2. Memory Manager
3. File System, Dateisystem

```
kernel
mm
fs

h
kernel
mm
fs
```

```
h
    h/const.h
    h/callnr.h
    h/com.h
    h/error.h
    h/sgtty.h
    h/signal.h
    h/stat.h
    h/type.h
kernel
    kernel/const.h
    kernel/glo.h
    kernel/proc.h
    kernel/type.h
    kernel/main.c
    kernel/mpx88.s
    kernel/klib88.s
    kernel/proc.c
    kernel/memory.c
    kernel/floppy.c
    kernel/clock.c
    kernel/tty.c
    kernel/system.c
mm
    mm/const.h
    mm/glo.h
    mm/mproc.h
    mm/param.h
    mm/type.h
    mm/main.c
    mm/forkexit.c
    mm/exec.c
    mm/break.c
    mm/signal.c
    mm/getset.c
    mm/alloc.c

fs

h
    h/const.h
    h/callnr.h
    h/com.h
    h/error.h
    h/sgtty.h
        Datenstrukturen fuer IOCTL
            struct sggtyb {...}
            struct tchars {...}
    h/signal.h
        NR_SIGS
```

```
        N_SIGS
        SIGHUP
        SIGINT
        SIGQUIT
        SIGILL
        SIGTRAP
        SIGIOT
        SIGEMT
        ...
        SIGKILL
        ...
        SIGPIPE
        SIGSYS
        SIGALARM
        SIGTERM
    h/stat.h
    h/type.h
        typedef     unsigned        vir_bytes;
        typedef     unsigned        vir_clicks;
        typedef     long            phys_bytes;
        typedef     unsigned        phys_clicks;
        typedef     int             signed_clicks;
kernel
    kernel/const.h
        NR_REGS
        ES_REG
        DS_REG
        CS_REG
        SS_REG
        Interrupt Vektoren: DIVIDE_VECTOR, CLOCK_VECTOR, KEYBOARD_VECTOR, FLOPPY_VECTOR
        PIC 8259A
    kernel/glo.h
    kernel/proc.h
    kernel/type.h
    kernel/main.c
        main ()
        unexpected_int ()
        trap ()
        div_trap ()
        panic ()
        set_vec (vec_nr, addr, base_click)
    kernel/mpx88.s
        MINIX
        s_call
        tty_int
        lpr_int
        disk_int
        wini_int
        clock_int
        suprise
```

```
    trp
    divide
    save
    restart
    idle
    data
kernel/klib88.s
    phys_copy
    cp_mess
    port_out
    port_in
    lock
    unlock
    restore
    build_sig
    csv
    cret
    get_chrome
    dma_read
    vid_copy
    get_byte
    reboot
    wreboot
kernel/proc.c
    sys_call
    interrupt
    ready
    unready
    sched
    mini_send
    pick_proc
kernel/memory.c
    mem_task
    do_mem
    do_setup
kernel/floopy.c
    floppy_task
    do_rdwt
    dma_setup
    start_motor
    stop_motor
    seek
    transfer
    fdc_results
    fdc_out
    recalibrate
    reset
    clock_mess
    send_mess
kernel/clock.c
```

```
        clock_task
        do_setalarm
        do_get_time
        do_set_time
        do_clock_tick
        accounting
        init_clock
    kernel/tty.c
        tty_task // Hauptroutine
        do_charint
        in_char
        make_break
        echo
        chuck
        do_read
        rd_chars
        finish
        do_write
        do_ioctl
        do_cancel
        tty_reply
        sigchar
        keyboard
        console
        out_char
        scroll_screen
        flush
        move_to
        escape
        set_6845
        beep
        tty_init
        putc
        func_key
    kernel/system.c
        sys_task
        do_fork
        do_newmap
        do_exec
        do_xit
        do_getsp
        do_times
        do_abort
        do_sig
        do_copy
        cause_sig
        inform
        umap
mm
    mm/const.h
```

```
    mm/glo.h
    mm/mproc.h
    mm/param.h
    mm/type.h
    mm/main.c
        main
        get_work
        reply
        mm_init
        do_brk2
        set_map
    mm/forkexit.c
        do_fork
        do_mm_exit
        mm_exit
        do_wait
        cleanup
    mm/exec.c
        do_exec
        read_header
        new_mem
        patch_ptr
        load_seg
    mm/break.c
        do_brk
        adjust
        size_ok
        stack_fault
    mm/signal.c
        do_signal
        do_kill
        do_ksig
        check_sig
        sig_proc
        do_alarm
        set_alarm
        do_pause
        unpause
        dump_core
    mm/getset.c
        do_getset
    mm/alloc.c
        alloc_mem
        free_mem
        del_slot
        merge
        max_hole
        mem_init
fs
```

1 h/const.h

h/const.h

```
#define     EXTERN          extern
#define     PRIVATE         static
#define     PUBLIC
#define     FORWARD
#define     TRUE            1
#define     FALSE           0
...
#define     BLOCK_SIZE      1024
#define     SUPER_USER      (uid) 0
#define     MAJOR           8
#define     MINOR           0
#define     NR_TASKS        8
#define     NR_PROCS        16
#define     NR_SEGS         3
#define     T               0
#define     D               1
#define     S               2
```

Wichtig

1. Text
2. Data
3. Stack

Wichtig! Code = Text

Wichtig:

1. Text `T`
2. Data `D`
3. Stack `S`

Wichtig:

1. Text `T, 0`
2. Data `D, 1`
3. Stack `S, 2`

Wichtig:

1. `NR_TASKS`
2. `NR_PROCS`
3. `NR_SEGS`

So, achtung aufgepasst!

```
struct proc {

} proc [NR_TASKS + NR_PROCS]
```

1. Anzahl der Prozesse
2. Anzahl der Tasks

1. Im Speichermanager
   (a) `NR_PROCS` - Im Speichermanager, nur Prozesse
2. Im Kernel
   (a) `NR_TASKS` - im Kernel, Tasks
   (b) `NR_PROCS` - Kernel, Prozesse

Dinge mit `NR`

1. `NR_TASKS`
2. `NR_PROCS`
3. `NR_SEGS`

Dinge mit `NR` zur Struktur

1. `(NR_TASKS. proc)`
2. `(NR_PROCS, proc)`
3. `(NR_SEGS, mem_map`

Wichtig

1. `mem` - speicher
2. `map` - karte

Wichtig

1. `NR_SEGS`
2. `T`
3. `D`
4. `S`

Wichtig

1. `NR_SEGS, 3`, maximal grenze
2. `T, 0`, erstes Element
3. `D, 1`, zweites Element
4. `S, 2`, drittes Element

2 h/callnr.h

Wichtig

`h/callnr.h ^= die wichtigsten Systemaufrufe`

Alle lernen

```
#define         NCALLS          69
#define         EXIT            1
#define         FORK            2
#define         READ            3
#define         WRITE           4
#define         OPEN            5
#define         CLOSE           6
#define         WAIT            7
#define         CREAT           8
...
```

Die wichtigsten Systemaufrufe. Wichtig

`NCALLS ^= Anzahl der Systemaufrufe`

1. `exit ()`
2. `fork ()`
3. `read  ()`
4. `write ()`
5. `open ()`
6. `close ()`
7. `create ()`

Daneben gibt es, einfach lernen, egal, wie

```
exit
fork
read
write
open
close
wait
create
link
unlink
chdir
time
mknod
chmod
chown
```

```
brk
stat
lseek
getpid
mount
umount
setuid
getuid
stime
alarm
fstat
pause
utime
access
sync
kill
dup
pipe
times
setgid
getgid
signal
ioctl
exec
umask
chroot
```

3 h/com.h

Wichtig, Definitionen für Systemaufrufe
Botschaften

```
SEND            1   /* Code fuer Botschaften verschicken */
RECEIVE         2   /* Code fuer Botschaften empfangen */
BOTH            3   /* Code fuer SEND + RECIEVE */
ANY
```

Wichtig
Botschaften

```
HARDWARE
SYSTASK
CLOCK
MEM
FLOPPY
WINCHESTER
TTY
PRINTER
```

Wichtig: Botschaften, darin gibt es entsprechende Parameter - Achtung, koennte man meinen. Das sind Botschaften,

1. Fuer jedes Gerät eine Botschaft
   (a) Hardware
   (b) Systemaufrufe
   (c) Uhr
   (d) Arbeitsspeicher, `MEM`
   (e) Floppy Disk
   (f) Festplatte
   (g) TTY
   (h) Drucker
2. für alle diese gibt es Botschaften
3. Das sind keine Paramter, sondern Botschaften an das Gerät
4. Bei der Festplatte gibt es: `DISK_READ`, `DISK_WRITE`. Botschaft an das Gerät: Lesen, schreiben

```
HARDWARE
SYSTASK
    /* zum Beispiel */
    SYS_FORK
    SYS_EXEC
    SYS_COPY
    SYS_TIMES
CLOCK
    SET_ALARM
    CLOCK_TIME
    GET_TIME
    SET_TIME
    REAL_TIME
MEM
    RAM_DEV             /dev/ram
    MEM_DEV             /dev/mem
    KMEM_DEV            /dev/kmem
    NULL_DEV            /dev/null
FLOPPY
WINCHESTER
    DISKINT
    DISK_READ
    DISK_WRITE
    DISK_IOCTL
TTY/PRINTER
    ...
    TTY_READ
    TTY_WRITE
    TTY_IOCTL
```

4 h/error.h

Lauter Fehler

```
NERROR
OK
..
EIO
ENOMEM
ENODEV
...
```

5 h/type.h

```
typedef     unsigned        vir_bytes;
typedef     unsigned        vir_clicks;
typedef     long            phys_bytes;
typedef     unsigned        phys_clicks;
typedef     int             signed_clicks;
```

Lerne zu unterscheiden

1. Bytes
2. Clicks

Lerne zu unterscheiden

1. Virtuell
2. Physikalisch

1. Virtuelle Bytes
2. Virtuelle Clicks
3. Physikalische Bytes
4. Physikalische Clicks

```
struct mem_map {
    vir_clicks mem_vir;
    phys_clicks mem_phys;
    vir_clicks mem_len;
};
```

1. Anfang
2. Länge

Lerne zu unterscheiden

1. Anfang

2. Ende
3. Laenge

Wichtig:

1. Anfang
2. Länge

wichtig:

1. Angabe in Clicks nicht in Bytes
2. Anfang:
   (a) Virtuell
   (b) Physikalisch
3. Laenge: Virtuell

6 kernel/const.h

1. Register
2. Interrupt-Vektoren
3. `8259A`

1. Register
   (a) Übersicht
      i. `ax` 1
      ii. `bx` 2
      iii. `dx` 3
      iv. `cx` 4
      v. `si` 5
      vi. `di` 5
      vii. `sp` 7
      viii. `bp` 8
      ix. `es`
      x. `ss` 9
      xi. `ds` 10
      xii. `cs` 11
      xiii. `ip`
   (b) Definitionen
      i. Anzahl
      ii. Code
      iii. Daten
      iv. Stack

2. Interrupt-Vektoren
   (a) Geteilt durch 0
   (b) Uhr
   (c) Keyboard
   (d) XT-Wini
   (e) Floppy
   (f) Printer
   (g) Systemaufrufe
   (h) AT-Wini
3. `8259A`
   (a) IO-Port: `0x20`, I/O-Port Interrupt Controller `0xA0`, I/O-Port zweiter Interrupt Controller
   (b) Maske, 1. und 2.

```
#define     NR_REGS             11
#define     ES_REG              7
#define     DS_REG              8
#define     CS_REG              9
#define     SS_REG              10
```

Interrupt Vektoren

```
#define     DIVIDE_VECTOR       0
#define     CLOCK_VECTOR        8
#define     KEYBOARD_VECTOR     9
#define     XT_WINI_VECTOR      13
#define     FLOPPY_VECTOR       14
#define     PRINTER_VECTOR      15
#define     SYS_VECTOR          32
#define     ATA_WINI_VECTOR     118
```

8259A

```
#define     INT_CTL             0x20        /* I/O-Port fuer Interrupt Controller */
#define     INT_CTLMASK         0x21        /* Bits auf diesem Port setzen verbietet Inter
#define     INT2_CTL            0xA0
#define     INT2_MASK           0xA1
...
```

7 kernel/glo.h

8 kernel/proc.h

```
EXTERN struct proc {

} proc [NR_TASKS + NR_PROCS];
```

```
NR_TASKS!
NR_PROCS!

struct proc !

EXTERN struct proc {
    int p_...
} proc [NR_TASKS + NR_PROCS];

EXTERN struct proc {
    int p_...1
    int p_...2
    int p_...3
} proc [NR_TASKS + NR_PROCS];

EXTERN struct proc {
    int p_reg [NR_REGS];
    ...
    struct mem_map [NR_SEGS];
    ...
} proc [NR_TASKS + NR_PROCS];

NR_REGS !
NR_TASKS !
NR_PROCS !
NR_SEGS !
```

1. NR_REGS
2. NR_TASKS
3. NR_PROCS
4. NR_SEGS

OK

```
EXTERN struct proc {
    int p_reg [NR_REGS];
    ...
    struct mem_map [NR_SEGS];
    int p_pid;
    ...
} proc [NR_TASKS + NR_PROCS];
```

1. Register
2. Speicherabbild
3. Prozess ID

```
EXTERN struct proc {
    int p_reg [NR_REGS];
    int *p_sp;                          /* Stack pointer */
    struct pc_psw p_pcpsw;              /* pc und psw bei Interrupt gesichert */
    int p_flags;                        /* P_SLOT, ... nicht flags */
    struct mem_map [NR_SEGS];
    int *p_splimit;                     /* Niedrigster wert fuer den Stack */
    int p_pid;

    real_time user_time;
    real_time sys_time;
    real_time child_utime;
    real_time child_stime;
    real_time p_alarm;

    struct proc *p_callreq;             /* Anfang der Liste aller Prozesse die senden woll
    struct proc *p_sendlink;
    message *p_messbuf;                 /* Zeiger auf Puffer fuer die Botschaft */
    int p_getfrom;

    struct proc *p_nextready;
    int p_pending;
} proc [NR_TASKS + NR_PROCS];
```

9 kernel/type.h

```
struct pc_psw {
    int (*pc)();
    phys_clicks cs;
    unsigned psw;           /* Programm status word */
};
```

10 kernel/main.c

```
main ()
unexpected_int ()
trap ()
div_trap ()
panic ()
set_vec (vec_nr, addr, base_click)
```

11 kernel/mpx88.s

```
    kernel/mpx88.s
        MINIX
        s_call
        tty_int
        lpr_int
```

```
        disk_int
        wini_int
        clock_int
        suprise
        trp
        divide
        save
        restart
        idle
        data
```

1. TTY
2. Disk : Floppy-Disk
3. Winchester: Hard Drive
4. Clock
5. LPR - Drucker

6. Systemaufrufe

7. idle : nichts zu tun

Beispiele:

```
_tty_int:
    call save
    call _keyboard
    jmp _restart
_lpr_int:
    call save
    call _pr_char
    jmp _restart
...

_disk_int:
    call save
    mov _int_mess+2, *DISKINT
    mov ax, #int_mess
    push ax
    mov ax, *FLOPPY
    push ax
    call _interrupt
    jmp _start

_wini_int:
    ...
```

```
mov ax, *WINI
...
```

1. Unterste Ebene der Minix-kernels
2. Prozess und Botschaftsverwaltung wird abgewickelt
3. dazu in `proc.c`

4. Prozess
5. Botschaft

12 kernel/klib88.s

Wichtig:

1. Assembler-Hilfsroutinen
2. Zum Beispiel:
   (a) `port_in`
   (b) `port_out`

```
kernel/klib88.s
    phys_copy       // kopiert Daten innerhalb des Speichers
    cp_mess         // kopiert Botschaften von der Quelle zum ziel
    port_out
    port_in
    lock            // interrupts verbieten
    unlock          // interrupts zulassen
    restore
    build_sig
    csv
    cret
    get_chrome      // liefert 0 bei Monochrom Bildschirm, 1 bei Farbbildschirm
    dma_read
    vid_copy        // Diese Routine nimmt einen String von Zeichen und schreibt ihn a
    get_byte        // Holt Byte irgendwo aus dem Speicher
    reboot
    wreboot
```

13 kernel/proc.c

1. Prozessabarbeitung
2. Botschaftsabarbeitung

Wichtig:

1. `ready`: Prozess in Warteschlange
2. `unready`: Prozess aus Warteschlange entfernen
3. `sched`: Prozess zu lange gerechnet

```
sys_call
interrupt
ready
unready
sched
mini_send
pick_proc
```

14 kernel/memory.c

```
Treiber fuer vier wichtige Geraedateien
    /dev/null - null Device = Daten Senke
    /dev/mem  - ganzer Speicher
    /dev/kmem - vom kernel belegter Speicher
    /dev/ram  - RAM-Disk
```

1. Daten Senke `/dev/null`
2. Ganzer Speicher `/dev/mem`
3. Speicher, Kernel `/dev/kmem`
4. RAM-Disk `/dev/ram`
5. Treiber
6. Treiber, fuer alles, was im RAM als Gerät angesprochen wird, wie bei Floppy

```
mem_task
do_mem
do_setup
```

15 kernel/floppy.c

1. Treiber für Floppy-Disk-Controller
2. NEC PD765-Chip

```
IO-Ports:
DOR
FDC_STATUS
FDC_DATA
FDC_RATE
DMA_ADDR
```

```
Statusregister:
ST0
ST1
ST2
ST_CYL
ST_HEAD
ST_SEC
ST_PCN


Kommandobytes:
FDC_SEEK
FDC_READ
FDC_WRITE
FDC_RECALLIBRATE

Parameter
SECTOR_SIZE 512
HC_SIZE
NR_HEADS
```

Zum Beispiel

1. Bits fuer Motorsteuerung: `DOR`, `0x3f2`
2. Status register des Floppy-Disk-Controllers, `FDC_STATUS`, `0x3f4`
3. Datenregister des Floppy Disk Controllers: `FDC_DATA`, `0x3f5`
4. Register fuer Transferrate: `FDC_RATE`
5. Ports fuer die unteren 16 Bit der DMA-Addresse `DMA_ADDR`
6. Status Register 0 : `ST0`
7. Status Register 1 : `ST1`
8. Status Register 2 : `ST2`
9. Status Register 3 : `ST3`
10. Hier meldet der Controller Zylinder: `ST_CYL`
11. Hier meldet der Controller Koepfe: `ST_HEAD`
12. Hier meldet der Controller Sektoren: `ST_SEC`
13. Hier meldet der Controller aktuellen Zylinder: `ST_PCN`

1. I/O-Ports
2. Statusregsiter
3. Felder in I/O-Ports

4. Kommandobytes
5. Parameter, wie Sektorgroesse, ...
6. Fehlernummern: Positionierung `ERR_SEEK`

```
floppy_task // Hauptprogramm des Treibers
do_rdwt // Anforderung auf Platte zu lesen oder schreiben
dma_setup
start_motor
stop_motor
seek
transfer
fdc_results
fdc_out
recalibrate
reset
clock_mess
send_mess
```

16 kernel/clock.c

```
clock_task
do_setalarm
do_get_time
do_set_time
do_clock_tick
accounting
init_clock
```

17 kernel/tty.c

```
tty_task // Hauptroutine
do_charint
in_char
make_break
echo
chuck
do_read
rd_chars
finish
do_write
do_ioctl
do_cancel
tty_reply
sigchar
keyboard
console
out_char
```

```
scroll_screen
flush
move_to
escape
set_6845
beep
tty_init
putc
func_key
```

18 kernel/system.c

1. Zwischenstueck sowohl zwischen Filesystem und kernel
2. als auch zwischen Speicherverwaltung und Kernel

```
SYS_FORK
SYS_NEWMAP
SYS_EXEC
SYS_XIT
SYS_GETSP
SYS_TIMES
SYS_ABORT
SYS_SIG
SYS_COPY
```

```
sys_task
do_fork
do_newmap
do_exec
do_xit
do_getsp
do_times
do_abort
do_sig
do_copy
cause_sig
inform
umap
```

19 mm

19.1 mm/const.h

19.2 mm/glo.h

19.3 mm/mproc.h

```
EXTERN struct mproc {

} mproc [NR_PROCS];
```

```
EXTERN struct mproc {
    int mp_...
    int mp_...
    int mp_...
} mproc [NR_PROCS];
```

```
EXTERN struct mproc {
    struct mem_map mp_seg [NR_SEGS];
    ...

} mproc [NR_PROCS];
```

```
EXTERN struct mproc {
    struct mem_map mp_seg [NR_SEGS];
    char mp_exitstatus;
    char mp_sigstatus;
    int mp_pid;
    int mp_parent;
    int mp_procgrp;

    uid mp_realuid;
    uid mp_effuid;
    gid mp_realgid;
    gid mp_effgid;

    unshort mp_ignore;
    unshort mp_catch;
    int (*mp_func) ();
    unsigned mp_flags;
} mproc [NR_PROCS];
```

1. `mp_seg`
2. `mp_exitstatus`
3. `mp_sigstatus`
4. `mp_pid`
5. `mp_parent`
6. `mp_procgrp`
7. `mp_realuid`
8. `mp_effuid`
9. `mp_realgid`
10. `mp_effgid`
11. `mp_ignore`
12. `mp_catch`
13. `int (*mp_func) ()`
14. `mp_flags`

19.4 mm/param.h

19.5 mm/type.h

19.6 mm/main.c

```
main
get_work
reply
mm_init
do_brk2
set_map
```

19.7 mm/forkexit.c

```
do_fork
do_mm_exit
mm_exit
do_wait
cleanup
```

19.8 mm/exec.c

```
do_exec
read_header
new_mem
patch_ptr
load_seg
```

19.9 mm/break.c

```
do_brk
adjust
size_ok
stack_fault
```

19.10 mm/signal.c

```
do_signal
do_kill
do_ksig
check_sig
sig_proc
do_alarm
set_alarm
do_pause
unpause
dump_core
```

19.11 mm/getset.c

```
do_getset
```

19.12 mm/alloc.c

```
alloc_mem
free_mem
del_slot
merge
max_hole
mem_init
```

```
PRIVATE struct hole {
    phys_clicks h_base;
    phys_clicks h_len;
    struct hole *h_next;
} hole [NR_HOLES];
```

Ein Loch besteht aus einem

1. Anfang, Basis
2. Länge
3. Nächster Eintrag in der Freispeicherliste

Es gibt eine Liste mit

1. Zeiger auf Löcher, Liste
2. Zeiger auf unbenutzte Tabelleneinträge, `free_slots`

```
PRIVATE struct hole {
    phys_clicks h_base;
    phys_clicks h_len;
    struct hole *h_next;
} hole [NR_HOLES];

PRIVATE struct hole *hole_head;
PRIVATE struct hole *free_slots;

PUBLIC phys_clicks alloc_mem (phys_clicks clicks) {
    register struct hole *hp, *prev_ptr;
    phys_clicks old_base;


    hp = hole_head;
    while (hp != NIL_HOLE) {
        if (hp -> h_len >= clicks) {
            old_base = hp->h_base;
            hp->h_base += clicks;
            hp->h_len -= clicks;

            if (hp->h_len != 0)
                return old_base;
            del_slot (prev_ptr, hp);
```

```
            return old_base;
        }
        prev_ptr = hp;
        hp = hp->h_next;
    }
}

del_slot (struct hole *prev_ptr, struct hole *hp) {
    if (hp == hole_head)
        hole_head = hp->h_next;
    else
        prev_ptr->h_next = hp->h_next;
    hp->h_next = free_slots;
    free_slots = hp;
}
```

Alle funktionen des Speichers

```
main
get_work
reply
mm_init
do_brk2
set_map

do_fork
do_mm_exit
mm_exit
do_wait
cleanup

do_exec
read_header
new_mem
patch_ptr
load_seg

do_brk
adjust
size_ok
stack_fault

do_signal
do_kill
do_ksig
check_sig
sig_proc
do_alarm
set_alarm
do_pause
```

```
unpause
dump_core

do_getset

alloc_mem
free_mem
del_slot
merge
max_hole
mem_init
```