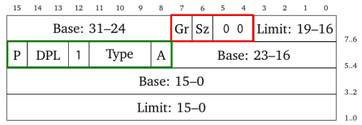
Datenstrukturen

GDT (global descriptor table)




```
⟨type definitions⟩+≡
 struct gdt_entry {
   uint limit_low    : 16;
   uint base_low     : 16;
   uint base_middle  :  8;
   uint access       :  8;
   uint flags        :  4;
   uint limit_high   :  4;
   uint base_high    :  8;
 };
```

```
⟨type definitions⟩+≡
 struct gdt_ptr {
   uint limit : 16;
   uint base  : 32;
 } __attribute__((packed));
```

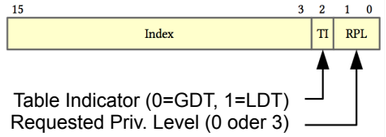
```
⟨global variables⟩+≡
 struct gdt_entry gdt[6];
 struct gdt_ptr gp;
```

– „Trick-GDT“ (base=0x40000000)
– normale GDT (base=0x00000000)

access	Code	Daten
kernel (R0)	10011010	10010010
user (R3)	11111010	11110010

No.	Offset	Funktion
0	0x00 (0)	Null-Deskriptor
1	0x08 (8)	Kernel, Code
2	0x10 (16)	Kernel, Daten
3	0x18 (24)	User, Code
4	0x20 (32)	User, Daten
5	0x28 (40)	TSS

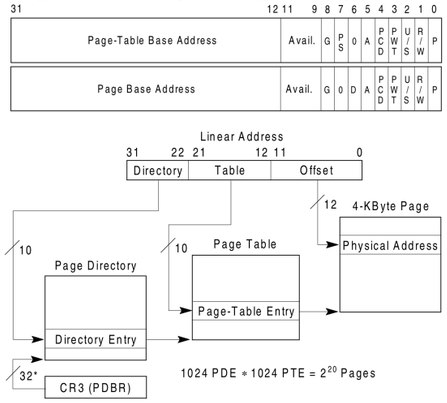
Segment Selector (CS, DS, …, SS)



System Calls

```
void *syscall_table[MAX_SYSCALLS];
```

Page Table Descriptor / Page Descriptor




```
⟨type definitions⟩+≡
 typedef struct {
   uint present          :  1;  //  0
   uint writeable        :  1;  //  1
   uint user_accessible  :  1;  //  2
   uint pwt              :  1;  //  3
   uint pcd              :  1;  //  4
   uint accessed         :  1;  //  5
   uint undocumented     :  1;  //  6
   uint zeroes           :  2;  //  8.. 7
   uint unused_bits      :  3;  // 11.. 9
   uint frame_addr       : 20;  // 31..12
 } page_table_desc;

 typedef struct { page_table_desc
   ptds[1024]; } page_directory;

 typedef struct {
   uint present          : 1;  //  0
   uint writeable        : 1;  //  1
   uint user_accessible  : 1;  //  2
   uint pwt              :  1;  //  3
   uint pcd              :  1;  //  4
   uint accessed         :  1;  //  5
   uint dirty            :  1;  //  6
   uint zeroes           :  2;  //  8.. 7
   uint unused_bits      :  3;  // 11.. 9
   uint frame_addr       : 20;  // 31..12
 } page_desc;

typedef struct { page_desc pds[1024]; }
  page_table;
```

```
⟨global variables⟩+≡
page_directory  kernel_pd;
page_table      kernel_pt;
page_table      kernel_pt_ram[16];
page_directory *current_pd = &kernel_pd;
page_table     *current_pt = &kernel_pt;
```

Address Space

```
⟨type definitions⟩+≡
 typedef struct {
   void         *pd;
   int           pid;
   short         status;
   memaddress    memstart, memend;
   unsigned int stacksize;
   memaddress    kstack_pt;
   unsigned int refcount;
   byte extra_kstacks;
 } address_space;
```

```
⟨global variables⟩+≡
 addr_space_id current_as
```

Thread Control Block, Blocked Queue, Lock

```
⟨type definitions⟩+≡
 typedef struct {
   thread_id  pid;          // process id
   thread_id  tid;          // thread id
   thread_id  ppid;         // parent process
   int        state;        // state of the process
   context_t  regs;         // context
   memaddress esp0;         // kernel stack pointer
   memaddress eip;          // program counter
   memaddress ebp;          // base pointer
   addr_space_id addr_space;
   thread_id next;          // id of ``next'' thread,
   thread_id prev;          // ``previous''  thread
   boolean used;
   int error;
   int exitcode;
   int waitfor;             // pid of the child that
                            // this process waits for
   char cmdline[CMDLINE_LENGTH];
   boolean   new;           // is this thread new?
   void      *top_of_thread_kstack;
                            // thread kernel stack
   int terminal;
   int files[MAX_PFD];
   char cwd[256];
   sighandler_t sighandlers[32];
   unsigned long sig_pending;
   unsigned long sig_blocked;
   word uid;                // user ID
   word gid;                // group ID
   word euid;               // effective user ID
   word egid;               // effective group ID
   word ruid;               // real user ID
   word rgid;               // real group ID
 } TCB;

typedef struct {
  thread_id next;     // id of ``next'' thread
  thread_id prev;     // id of ``previous'' thread
} blocked_queue;

typedef struct {
  short int     l;             // the lock
  boolean       used;          // using this lock?
  blocked_queue bq;            // queue for this lock
  char          lockname[20];  // name
} lock_t;
typedef lock_t *lock;
```

Funktionen

GDT und TSS

```
void fill_gdt_entry (int num, ulong base,
  ulong limit, uchar access, uchar gran);

void gdt_install();

void gdt_flush();

void tss_flush();
```

Page Table Descr. / Page Descr.

```
page_desc *fill_page_desc (page_desc *pd, uint present,
  uint writeable, uint user_accessible, uint dirty,
  memaddress frame_addr);

page_table_desc *fill_page_table_desc (page_table_desc
  *ptd, uint present, uint writeable, uint user_
  accessible, memaddress frame_addr);   // ohne dirty

#define KMAP(pd,frame)    fill_page_desc
  (pd, true, true, false, false, frame)

#define UMAP(pd,frame)    fill_page_desc
  (pd, true, true, true,  false, frame)

#define UMAPD(ptd, frame) fill_page_table_desc
  (ptd, true, true, true,  frame)

#define KMAPD(ptd, frame) fill_page_table_desc
  (ptd, true, true, false, frame)
```

Address Range	Usage	Access
0xD4000000 – 0xFFFFFFFF	*unused*	–
0xD0000000 – 0xD3FFFFFF	64 MByte Physical RAM (mapped)	K
0xC0000000 – 0xCFFFFFFF	Kernel Code and Data	K
0xBFFFF000 – 0xBFFFFFFF	Kernel Stack (4 KByte = one page)	K
0xB0000000 – 0xBFFFEFFF	*unused*	–
… – 0xAFFFFFFF	User Mode Stack	U
	User Mode Stack (grows automatically) Heap (can be grown with $sbrk_{158d}$)	(U)
0x00000000 – …	Process Code and Data	U

Pages und Frames anfordern / zurückgeben

```
void *request_new_page  ();
void *request_new_pages (int number_of_pages);
void release_page (unsigned int pageno);
void release_page_range (unsigned int start_pageno,
  unsigned int end_pageno);

int request_new_frame ();
void release_frame (int frame);
```

System Calls

```
void install_syscall_handler (int syscallno,
  void *syscall_handler);

void *syscall_table[MAX_SYSCALLS];
```

```
⟨linux system calls⟩+≡
#define __NR_exit          1
#define __NR_fork          2
#define __NR_read          3
#define __NR_write         4
#define __NR_open          5
#define __NR_close         6
...
```

```
⟨ulix system calls⟩+≡
#define __NR_get_errno     501
#define __NR_set_errno     502
...
```

Threads

```
int register_new_tcb
  (addr_space_id as_id);

void add_to_ready_queue
  (thread_id t);
void remove_from_ready_queue
  (thread_id t);

void add_to_blocked_queue
  (thread_id t, blocked_queue *bq);
void remove_from_blocked_queue
  (thread_id t, blocked_queue *bq);

void block (blocked_queue *q,
  int new_state);
void deblock (thread_id t,
  blocked_queue *q)
```