7. Speicherverwaltung

Speicherverwaltung

- Paging (Linux)
- Aufbau des Prozess-Speichers unter Linux
- `malloc()`, `calloc()`, `realloc()` und `free()`
- `memset()`
- `memcpy()`, `memcmp()`
- Alignment
- Anonymous Memory Mapping mit `mmap()`
- Linux OOM (Out of memory) Killer

Paging

- Linux verwendet **Paging** für die Speicherverwaltung
  - Seitengröße 4 KByte (Intel 32/64 bit) oder variabel: 4, 16, 64, 256 KByte (PPC32)

```
esser@ubu64:~$ cat getpagesize.c
#include <unistd.h>
#include <stdio.h>
int main () { printf ("page size: %d\n", getpagesize()); };
esser@ubu64:~$ gcc -o getpagesize getpagesize.c ; ./getpagesize
page size: 4096
esser@ubu64:~$ uname -m
x86_64
```

  - nutzt **Page Sharing** (z. B. für Bibliotheken) und **Copy-on-Write** (z. B. für `fork`)

Prozess-Speicher

- **Text-Segment**
  - Programm-Code, String-Literale, Konstanten
  - read-only, direkt auf Programmdatei gemappt
- **Stack**
  - lokale Variablen, Funktionsrückgabewerte, Rücksprungadressen
  - wächst und schrumpft nach Bedarf
- **Data-Segment**
  - globale, initialisierte Variablen
  - **Heap**, dynamischer Prozess-Speicher
  - verwaltet Prozess über `malloc()`, `free()` etc.

Prozess-Speicher

- **BSS-Segment**
  - nicht initialisierte globale Variablen
  - werden bei Prozess-Start auf 0 initialisiert

    (landen nicht in der Objektdatei eines Programms)
  - implementiert über Copy-on-Write-Mapping auf Seite mit Nullen
- **Mapped Files** (`mmap`)

Prozess-Speicher

- **Anonymous Memory Mappings**
  - für große Speicher-Anforderungen (`malloc`), die nicht auf dem Heap landen
  - `glibc` entscheidet abhängig von Größe, ob Anon-Mapping oder Heap (bis 128 K) verwendet wird
  - vermeidet Fragmentierung des Heap
  - sind schon mit 0 gefüllt

Beispiel für Speicher-Aufteilung

```
// memtest.c
// Hans-Georg Eßer, Systemprogrammierung
#include <stdio.h>    // printf
#include <unistd.h>   // sleep
#include <stdlib.h>   // malloc

char chararray[1024];              // BSS-Segment  (nicht init.)
const int i = 7;                   // Text-Segment (konstant)
int j = 4095;                      // Data-Segment (initialisiert)

void testfunc () {
  int array[4096];                 // Stack
  array[i] = 3; array[j] = 5;
  chararray[1] = 'B';
  printf ("Test: %d, %d\n", array[i], array[j]);
};

int main () {
  chararray[0] = 'A'; chararray[2] = '\0';
  testfunc ();
  printf ("Test: %s\n", chararray);
  char* s = malloc (40*1024*1024);  // 40 MByte, Heap
  sleep (20);
  return 0;
};
```


Linux, 32 Bit

```
[esser@quadamd:tmp]$ pmap $(pidof memtest)
30387:   ./memtest
08048000      4K r-x--  /tmp/memtest
08049000      4K r----  /tmp/memtest
0804a000      4K rw---  /tmp/memtest
b4f22000  40968K rw---    [ anon ]
b7724000   1496K r-x--  /lib/i386-linux-gnu/libc-2.13.so
b789a000      8K r----  /lib/i386-linux-gnu/libc-2.13.so
b789c000      4K rw---  /lib/i386-linux-gnu/libc-2.13.so
b789d000     12K rw---    [ anon ]
b78c5000     12K rw---    [ anon ]
b78c8000      4K r-x--    [ anon ]
b78c9000    120K r-x--  /lib/i386-linux-gnu/ld-2.13.so
b78e7000      4K r----  /lib/i386-linux-gnu/ld-2.13.so
b78e8000      4K rw---  /lib/i386-linux-gnu/ld-2.13.so
bf99a000    132K rw---    [ stack ]
 total    42776K

[esser@quadamd:tmp]$ cat /proc/$(pidof memtest)/maps
08048000-08049000 r-xp 00000000 08:04 1156248    /tmp/memtest
08049000-0804a000 r--p 00000000 08:04 1156248    /tmp/memtest
0804a000-0804b000 rw-p 00001000 08:04 1156248    /tmp/memtest
b4f22000-b7724000 rw-p 00000000 00:00 0
b7724000-b789a000 r-xp 00000000 08:04 1966089    /lib/i386-linux-gnu/libc-2.13.so
b789a000-b789c000 r--p 00176000 08:04 1966089    /lib/i386-linux-gnu/libc-2.13.so
b789c000-b789d000 rw-p 00178000 08:04 1966089    /lib/i386-linux-gnu/libc-2.13.so
b789d000-b78a0000 rw-p 00000000 00:00 0
b78c5000-b78c8000 rw-p 00000000 00:00 0
b78c8000-b78c9000 r-xp 00000000 00:00 0          [vdso]
b78c9000-b78e7000 r-xp 00000000 08:04 9569185    /lib/i386-linux-gnu/ld-2.13.so
b78e7000-b78e8000 r--p 0001d000 08:04 9569185    /lib/i386-linux-gnu/ld-2.13.so
b78e8000-b78e9000 rw-p 0001e000 08:04 9569185    /lib/i386-linux-gnu/ld-2.13.so
bf99a000-bf9bb000 rw-p 00000000 00:00 0          [stack]
```

Linux, 64 Bit

```
esser@ubu64:~$ pmap $(pidof memtest)
1839:   ./memtest
0000000000400000      4K r-x--  /home/esser/memtest
0000000000600000      4K r----  /home/esser/memtest
0000000000601000      4K rw---  /home/esser/memtest
00007f06b1887000  40964K rw---    [ anon ]
00007f06b4088000   1512K r-x--  /lib/libc-2.11.1.so
00007f06b4202000   2044K -----  /lib/libc-2.11.1.so
00007f06b4401000     16K r----  /lib/libc-2.11.1.so
00007f06b4405000      4K rw---  /lib/libc-2.11.1.so
00007f06b4406000     20K rw---    [ anon ]
00007f06b440b000    128K r-x--  /lib/ld-2.11.1.so
00007f06b460d000     12K rw---    [ anon ]
00007f06b4627000     12K rw---    [ anon ]
00007f06b462a000      4K r----  /lib/ld-2.11.1.so
00007f06b462b000      4K rw---  /lib/ld-2.11.1.so
00007f06b462c000      4K rw---    [ anon ]
00007fff5fc9b000     84K rw---    [ stack ]
00007fff5fdff000      4K r-x--    [ anon ]
ffffffffff600000      4K r-x--    [ anon ]
 total            44828K
esser@ubu64:~$ cat /proc/$(pidof memtest)/maps
00400000-00401000          r-xp 00000000 08:01 100804             /home/esser/memtest
00600000-00601000          r--p 00000000 08:01 100804             /home/esser/memtest
00601000-00602000          rw-p 00001000 08:01 100804             /home/esser/memtest
7f06b1887000-7f06b4088000  rw-p 00000000 00:00 0
7f06b4088000-7f06b4202000  r-xp 00000000 08:01 8765               /lib/libc-2.11.1.so
7f06b4202000-7f06b4401000  ---p 0017a000 08:01 8765               /lib/libc-2.11.1.so
7f06b4401000-7f06b4405000  r--p 00179000 08:01 8765               /lib/libc-2.11.1.so
7f06b4405000-7f06b4406000  rw-p 0017d000 08:01 8765               /lib/libc-2.11.1.so
7f06b4406000-7f06b440b000  rw-p 00000000 00:00 0
7f06b440b000-7f06b442b000  r-xp 00000000 08:01 8718               /lib/ld-2.11.1.so
7f06b460d000-7f06b4610000  rw-p 00000000 00:00 0
7f06b4627000-7f06b462a000  rw-p 00000000 00:00 0
7f06b462a000-7f06b462b000  r--p 0001f000 08:01 8718               /lib/ld-2.11.1.so
7f06b462b000-7f06b462c000  rw-p 00020000 08:01 8718               /lib/ld-2.11.1.so
7f06b462c000-7f06b462d000  rw-p 00000000 00:00 0
7fff5fc9b000-7fff5fcb0000  rw-p 00000000 00:00 0                  [stack]
7fff5fdff000-7fff5fe00000  r-xp 00000000 00:00 0                  [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0           [vsyscall]
```

```
esser@ubu64:~$ size memtest
 text    data     bss     dec     hex   filename
 1538     536    1056    3130    c3a   memtest
```


malloc()

- dynamisch Speicher allozieren
- verwendet Heap oder Anon-Map
- bei Nutzung des Heap: nicht initialisiert!

```
char *p;
p = malloc (2000)   // 2000 Bytes anfordern
if (!p) {
  // Fehler
  perror ("malloc");
}

...

free (p);
```


xmalloc()

- Test auf `malloc()`-Fehler wird oft in Wrapper `xmalloc()` integriert:

```
void *xmalloc (size_t size) {
  void *p;
  p = malloc (size);
  if (!p) {
    perror ("xmalloc");
    exit (EXIT_FAILURE);    // returns 1
  };
  return p;
};
```


calloc()

- Ähnlich `malloc()`, aber für **Arrays**
- Angabe von Anzahl und Elementgröße
- Speicher immer initialisiert (0)

```
struct mystruct { ... };
struct mystruct *p;
p = calloc (200, sizeof(struct mystruct));  // 200 Einträge

if (!p) {
  // Fehler
  perror ("calloc");
};
```

realloc()

- ändert die Größe von Speicher, der mit `malloc()` bzw. `calloc()` angefordert wurde
- verkleinern oder vergrößern
- Vorsicht: Rückgabewert ist Pointer für neuen Speicherbereich, der jetzt an anderer Stelle anfangen kann!

```
p = malloc (10*sizeof(struct xy));
...
r = realloc (p, 20*sizeof(struct xy));
if (!r) {
  // Fehler, p noch intakt!
}
...                  // evtl. r != p

free (r);            // nicht: free (p) !
```


free()

- gibt einen dynamisch reservierten Speicherbereich wieder frei
- darf nur für Rückgabewerte von `malloc()` oder `calloc()` aufgerufen werden!
- keine Freigabe von „Teilen“ möglich (→ `realloc`)
- Nach Freigabe Speicher nicht mehr nutzen!
- Doppelter `free()`-Aufruf schlägt fehl (Prog.-Abbruch)
- `free(NULL)` geht immer, ohne Wirkung

```
p = malloc(...); free (p);   // auch ok, wenn p==0
```


free()

- Shell-Variable `MALLOC_CHECK_` erlaubt Einsatz einer alternativen `malloc()`-Implementierung, die z. B. doppelte `free()`s erkennt

```
// free2.c
#include <stdlib.h>
#include <stdio.h>

int main () {
  int* p = malloc (200);
  free (p);
  free (p);  // Fehler!!
  printf ("nach 2x free\n");
};
```

```
esser@ubu64:~$ ./free2
(Programm bricht ab, Backtrace etc.)

esser@ubu64:~$ MALLOC_CHECK_=0 ./free2
nach 2x free

esser@ubu64:~$ MALLOC_CHECK_=1 ./free2
*** glibc detected *** ./free2: free():
invalid pointer: 0x0000000001780010 ***
nach 2x free

esser@ubu64:~$ MALLOC_CHECK_=2 ./free2
Abgebrochen
```


Verwaltung des Heap

- Es gibt verschiedene Speicher-Allokations-Routinen
- Linux-Programme (mit `glibc`) nutzen Variante von `dlmalloc` („Doug Lea's malloc“), http://g.oswego.edu/dl/html/malloc.html
  - Best-Fit (für Anforderungen >= 256 Byte, < 256 KByte)
  - nutzt OS-Features ab 256 KByte (→ Anon-Mapping)
  - Sonderbehandlung für kleine Anforderungen
- Beschreibung von `dlmalloc`:
  → ftp://g.oswego.edu/pub/misc/malloc.c
  → http://www.securecoding.cert.org/confluence/download/attachments/3524/04+Dynamic+Memory+v6.pdf (Folie 58-65)
- Tutorial: http://www.inf.udec.cl/~leo/Malloc_tutorial.pdf

memset()

- Speicher, der mit `malloc()` alloziert wurde, ist (evtl.) nicht **initialisiert**
- Das kann man mit `memset()` nachholen
- benötigt `#include <string.h>`

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>  // memset

int main () {
  int SIZE = 200;
  char* p = malloc (SIZE); strcpy (p, "Ohm-HS"); free (p);
  p = malloc (SIZE);
  printf ("%2x %2x %2x %2x %2x %2x %2x -- %s\n",
    p[0], p[1], p[2], p[3], p[4], p[5], p[6], p);
  memset (p, 0, SIZE);  // oder statt 0 beliebiges Füll-Byte
  printf ("%2x %2x %2x %2x %2x %2x %2x -- %s\n",
    p[0], p[1], p[2], p[3], p[4], p[5], p[6], p);
};
```

```
esser@ubu64:~$ ./memset
4f 68 6d 2d 48 53  0 -- Ohm-HS
 0  0  0  0  0  0  0 --
```


memcpy()

- kopiert einen Speicherbereich:

  `memcpy (ziel, quelle, laenge)`
- Rückgabewert: Zeiger auf `ziel`

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>  // memcpy, memset

int main () {
  int SIZE = 200;
  char* p = malloc (SIZE); strcpy (p, "Ohm-HS");
  char* q = malloc (SIZE);
  memcpy (q, p, SIZE);
  printf ("q: %2x %2x %2x %2x %2x %2x %2x -- %s\n",
    q[0], q[1], q[2], q[3], q[4], q[5], q[6], q);
};
```


memcpy() vs. strncpy()

- Zum Unterschied
  - `memcpy (ziel, quelle, laenge)`
  - `strncpy (ziel, quelle, laenge)`
- `strncpy()` nimmt auf Besonderheiten von **Strings** Rücksicht:
  - Ist Länge(`quelle`) < `laenge`, wird `ziel` mit Null-Bytes aufgefüllt
  - Inhalt in `quelle` nach erstem Null-Byte wird ignoriert
  - terminierendes Null-Byte bei `laenge` berücksichtigen
  - Aber: Ist `quelle` zu lang, entsteht ein *nicht*-0-terminierter String!

memcpy() vs. strncpy()

```
esser@ubu64:~$ cat strncpy.c
#include <string.h>
#include <stdio.h>

int main () {
  char quelle[] = "Vier";
  char ziel[]   = "ZZZZZZZZZ";
  strncpy (ziel, quelle, 4);   // kopiert 4 Bytes, ohne \0
  printf ("1. Versuch: %s\n", ziel);
  strncpy (ziel, quelle, 5);   // kopiert ganzen String mit \0
  printf ("2. Versuch: %s\n", ziel);
};

esser@ubu64:~$ ./strncpy
1. Versuch: VierZZZZZ
2. Versuch: Vier
```

memcmp()

- `memcmp (a, b, len)` vergleicht zwei Speicherbereiche

```
int main () {
  int SIZE = 200;
  char* p = malloc (SIZE); strcpy (p, "Ohm-HS");
  char* q = malloc (SIZE); char* s = malloc (SIZE);
  memcpy (q, p, SIZE);     memcpy (s, p, SIZE);
  s[0] = 'a';
  if (memcmp(p, q, SIZE) != 0) printf ("p, q verschieden\n");
  if (memcmp(p, s, SIZE) != 0) printf ("p, s verschieden\n");
};
```

- nicht (!) zum Vergleich von structs verwenden:

```
struct xy *a, *b;
memcmp (a, b, sizeof(struct xy))
```

sagt nicht unbedingt, ob a und b gleich sind

Alignment

- `malloc()` & Co. lassen allozierte Speicherbereiche immer an Adressen anfangen, die ein Vielfaches von 8 (32 Bit) bzw. 16 (64 Bit) sind

  → Variablen aller Typen sind „**naturally aligned**“

```
// malloc-align-test.c
#include <stdlib.h>
#include <stdio.h>
int main () {
  int* a = malloc(1);
  int* b = malloc(1);
  int* c = malloc(1);
  printf ("a: %p\n", a);
  // %p: Addresse eines Pointers
  printf ("b: %p\n", b);
  printf ("c: %p\n", c);
}
```

```
esser@ubu64:~$ uname -m
x86_64
esser@ubu64:~$ ./malloc-align-test
a: 0x6cc010
b: 0x6cc030
c: 0x6cc050
```

(Abstand: 0x20 = 32, Extraplatz für Verwaltungsdaten von `malloc`, → nächste Folie)

Extra-Platz bei malloc()

```
// malloc-test.c

#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>

void dump (char *filename, char *buf, int len) {
  int fd = open (filename, O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
  write (fd, buf, len);
  close (fd);
}

int main () {
  char *p1 = malloc(4);      // gibt: p1 = 0x23e6010
  strncpy (p1, "ABC", 4);
  dump ("out1", p1-16, 96);

  char *p2 = malloc(4);      // gibt: p2 = 0x23e6030,  Differenz 0x20
  strncpy (p2, "XYZ", 4);
  dump ("out2", p1-16, 96);

  free(p2);
  dump ("out3", p1-16, 96);
}
```


Extra-Platz bei malloc()

```
[esser@ohmvm:~]$ ./malloc-test

[esser@ohmvm:~]$ hexdump -C out1    # nach dem 1. malloc() / strncpy()
00000000  00 00 00 00 00 00 00 00  21 00 00 00 00 00 00 00  |........!.......|
00000010  41 42 43 00 00 00 00 00  00 00 00 00 00 00 00 00  |ABC.............|
00000020  00 00 00 00 00 00 00 00  e1 0f 02 00 00 00 00 00  |................|
...

[esser@ohmvm:~]$ hexdump -C out2    # nach dem 2. malloc() / strncpy()
00000000  00 00 00 00 00 00 00 00  21 00 00 00 00 00 00 00  |........!.......|
00000010  41 42 43 00 00 00 00 00  00 00 00 00 00 00 00 00  |ABC.............|
00000020  00 00 00 00 00 00 00 00  21 00 00 00 00 00 00 00  |........!.......|
00000030  58 59 5a 00 00 00 00 00  00 00 00 00 00 00 00 00  |XYZ.............|
00000040  00 00 00 00 00 00 00 00  c1 0f 02 00 00 00 00 00  |................|
...

[esser@ohmvm:~]$ hexdump -C out3    # nach dem free()
00000000  00 00 00 00 00 00 00 00  21 00 00 00 00 00 00 00  |........!.......|
00000010  41 42 43 00 00 00 00 00  00 00 00 00 00 00 00 00  |ABC.............|
00000020  00 00 00 00 00 00 00 00  21 00 00 00 00 00 00 00  |........!.......|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  c1 0f 02 00 00 00 00 00  |................|
...
```

Alignment

- Durch **Cast-Operationen** können Variablen entstehen, die nicht aligned sind:

```
// wrong-alignment.c
#include <stdio.h>

int main () {
  char *str = "ABCDEFGHIJK";
  char *c = str + 1;
  putchar (*c); printf ("\n");
  unsigned long *u1, *u2;
  u1 = (unsigned long *) str;
  u2 = (unsigned long *) c;
  printf ("Pointer c:  %p\n",  c);
  printf ("Pointer u1: %p\n",  u1);
  printf ("Inhalt u1:  %lx\n", *u1);
  printf ("Pointer u2: %p\n",  u2);
  printf ("Inhalt u2:  %lx\n", *u2);
};
```

```
esser@ubu64:~$ ./wrong-alignment
B
Pointer c:  0x40072d
Pointer u1: 0x40072c
Inhalt u1:  4847464544434241
Pointer u2: 0x40072d   <-- 0xd = 13
Inhalt u2:  4948474645444342
```

ASCII-Tabelle:

A	0x41	E	0x45
B	0x42	F	0x46
C	0x43	G	0x47
D	0x44	H	0x48

Anonymous Memory Mapping

- Alternative zur Nutzung des Heaps
- Jedes Anon-Mapping wie ein separater Heap …

```
void *p;
p = mmap (NULL,                         // do not care where
          512 * 1024,                   // 512 KB
          PROT_READ | PROT_WRITE,       // read/write
          MAP_ANONYMOUS | MAP_PRIVATE,  // anonymous, private
          -1,                           // fd (ignored)
          0);                           // offset (ignored)

if (p == MAP_FAILED)
  perror ("mmap");
else
  // 'p' points at 512 KB of anonymous memory...
```

Quelle: Robert Love, Linux System Programming

Alternative: `/dev/zero` mappen

```
void *p; int fd;
fd = open ("/dev/zero", O_RDWR);   // open /dev/zero for reading/writing
if (fd < 0) { perror ("open"); return -1; }

// map [0,page size) of /dev/zero
p = mmap (NULL,                    // do not care where
          getpagesize ( ),         // map one page
          PROT_READ | PROT_WRITE,  // map read/write
          MAP_PRIVATE,             // private mapping
          fd,                      // map /dev/zero
          0);                      // no offset

if (p == MAP_FAILED) {
          perror ("mmap");
          if (close (fd)) perror ("close");
          return -1;
}

if (close (fd)) perror ("close");     // close /dev/zero, no longer needed
// 'p' points at one page of memory, use it...
```

Quelle: Robert Love, Linux System Programming

Linux: Out of Memory (OOM)

```
// oom2.c,
#include <stdio.h>
#include <stdlib.h>

int main () {
  void *myblock = NULL;
  int count = 0;

  while(1) {
     myblock = (void *) malloc(1024*1024);
     if (!myblock) break;
     memset(myblock,1, 1024*1024);
     printf("Currently allocating %d MB\n",++count);
  }
  exit(0);
}
```

Quelle:
http://linuxdevcenter.com/pub/a/linux/2006/11/30/linux-out-of-memory.html

Programm wird nach kurzer Zeit abgebrochen, Meldung im Syslog (`dmesg`):

```
[12063.014255] Out of memory: kill process 3412 (oom2) score 136919 or a child
[12063.014259] Killed process 3412 (oom2)
```

Linux: Out of Memory (OOM)

- Wie entscheidet der OOM Killer?

  → berechnet für jeden Prozess einen „Score“

```
root@ubu64:/home/esser# pidof oom2
5206
root@ubu64:/home/esser# cat /proc/5206/oom_score
78066
root@ubu64:/home/esser# for i in /proc/[0-9]*; do echo -n
"$i : "; cat $i/oom_score; done | sort -n -k 2 -t ":" | tail
/proc/4080 : 66999
/proc/4076 : 67449
/proc/4077 : 76768
/proc/3948 : 77604
/proc/5206 : 78066
/proc/4112 : 83046
/proc/1 : 91499
/proc/3967 : 100795
/proc/4149 : 150386
/proc/3870 : 445033
```


Linux: Out of Memory (OOM)

- Kriterien für Score-Berechnung
  - Startwert: RSS (resident memory size) + Swap-Nutzung eines Prozess
  - Speichernutzung jedes Kindprozesses (außer Kernel Threads) wird hinzu addiert
  - Score von Prozessen, die „nice“ sind, erhöhen
  - Score von lang laufenden Prozessen reduzieren

  In Kernel 2.6.38 nicht berück-sichtigt

  - Score von Prozessen mit CAP_SYS_ADMIN (hat Root-Rechte) wird reduziert

Berechnung in mm/oom_kill.c im Kernel-Sourcecode.
Kriterien ändern sich von Version zu Version
Für Erläuterungen: siehe http://linux-mm.org/OOM_Killer