Speicherverwaltung und -Optimierung

Seminar “Effiziente Programmierung”

Mirko Hartung

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

2018-11-29

Gliederung

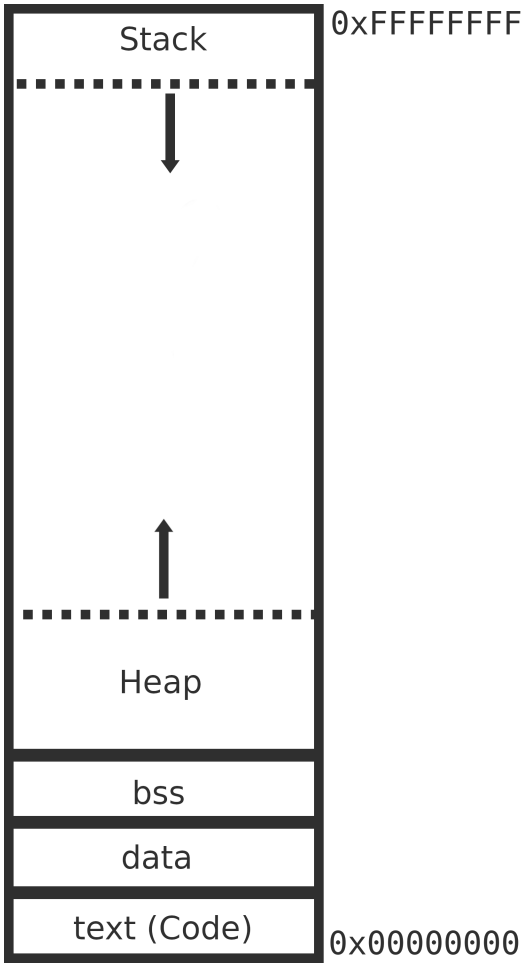
Motivation

- Speicherverwaltung
  - Verständnis von Leistungsengpässen
  - Wie sehen Anwendungen den physischen Speicher?
- Speicheroptimierung
  - Effiziente Nutzung des vorhandenen Speichers
  - Bewirkt die Miniaturisierung immer einen Performancegewinn?

Paging

- Speicher eines Programms abgebildet auf Speicherseiten
  - Jedes Programm hat einen eigenen Adressraum
  - Nicht das gesamte Programm muss im RAM gehalten werden
  - Betriebssystem hält wahrscheinlich aktive Seiten im Speicher
- Dynamisches Speichermanagment
  - Programme können neuen Speicher anfordern
  - *malloc*, *calloc* abstrahieren von Seiten
    $\Rightarrow$ Seiten haben dennoch Einfluss auf die Performance

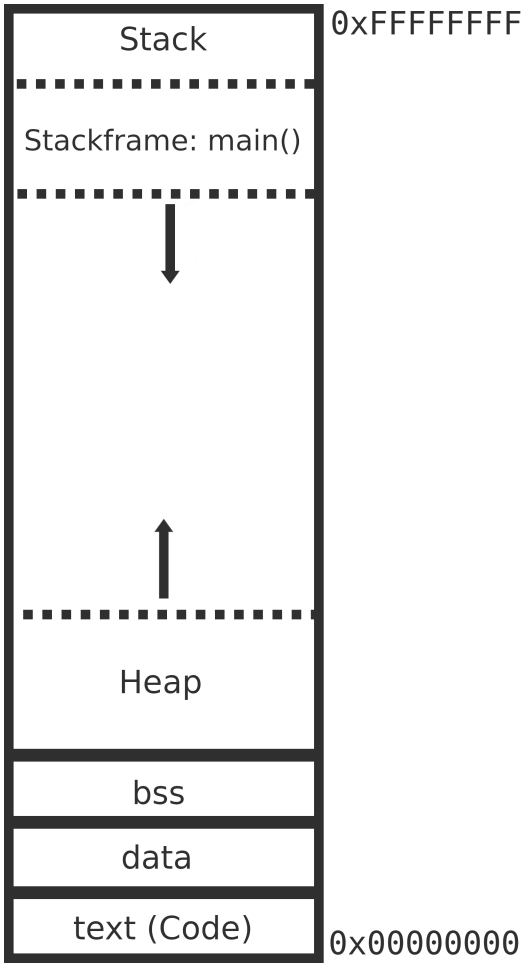
Speicherallokation



Layout im virtuellen Adressraum

- Stack “nach unten” wachsend
- Dynamisch allokierter Speicher wächst “nach oben”
- Statisch reservierter Speicher am Beginn des Adressraums

Speicherallokation




```
long counter; // bss
short number = 20; // data

int main(void)
{
  int i; // stack

  int *p; // stack

  p = malloc(sizeof (int)); // heap

  return 0;
}
```

Programmieren mit dynamischer Speicherallokation

- Arbeiten mit Pointern ist gefährlich
- Dennoch: Mehr Nutzen als Risiken
- free() führt eventuell zu Speicherlecks

```
struct s {
  int *val;
} *cont;

int *ptr;

void calc(void) {
  ptr = malloc(sizeof (int));
  *ptr = 245;
  free(ptr);

  cont = malloc(sizeof (s));
  cont->val = malloc(sizeof (int));

  free(cont->val);
  free(cont);
}
```

Programmieren mit dynamischer Speicherallokation

```
struct listElement {
  int value;
  struct listElement *next;
};

void deleteList(listElement* list) {
  if (list->next != NULL) {

    deleteList(list->next);

  }

  free(list);
}
```

Wie viel Speicher benötigt dieses Struct?

```
struct listElement {
  int value;
  struct listElement *next;
};
```

Struct und Type Alignment in C

- Struct Alignment
  - Speicheradressierung in x86 und ARM als Ursache
  - Speicherposition von Membervariablen nicht trivial
  - Padding führt zu verschwendeten Speicher
  - Performanceverlust durch Page/Cache-Misses
  - Manuelles Optimieren ist möglich!
- Struct Packing
  - Anweisung an den Compiler kein Padding zu verwenden
  - Zugriff auf Membervariablen ist rechenaufwändig
- Performance vs. Speicherplatz

Beispiel: Alignment im Struct

```
// Implizites Padding
struct containerA
{
  char c;
  int i;
  short s;
  long l;

};
```

```
// Explizites Padding
struct containerB
{
  char c;
  char pad1[3];
  int i;
  short s;
  char pad2[6];
  long l;

};
```

- sizeof(containerA) = ?

Lösung: Ändern der Reihenfolge

```
struct containerB
{
  char c;
  char pad1[3];
  int i;
  short s;
  char pad2[6];
  long l;
};
```

24 Byte

```
struct containerC
{
  long l;
  int i;
  short s;
  char c;
  char pad[1];
};
```

16 Byte

Lösung: Structure Packing

```
// Padding
struct containerB
{
   char c;
   char pad1[3];
   int i;
   short s;
   char pad2[6];
   long l;
};
```

```
// Kein Padding
#pragma pack(1)
struct containerD
{
   char c;
   char pad1[3];
   int i;
   short s;
   char pad2[6];
   long l;
};
```

- sizeof(containerD) = ?

Effizienz des Zugriffs auf Member

```
#pragma pack(1)
int main () {
  struct cont
  {
    int i;
    char c;
  } st;

  st.c = 'a';
  st.i = 20;
}
```

Effizienz des Zugriffs auf Member

Ohne Pragma

```
push    rbp
mov     rbp, rsp
mov     BYTE PTR [rbp-4], 97
mov     DWORD PTR [rbp-8], 20
mov     eax, 0
pop     rbp
ret
```

Mit Pragma

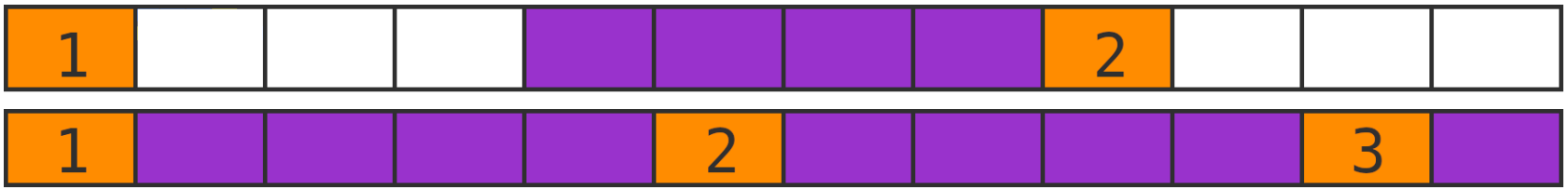
```
push    rbp
mov     rbp, rsp
mov     BYTE PTR [rbp-1], 97
mov     DWORD PTR [rbp-5], 20
mov     eax, 0
pop     rbp
ret
```

Performance? In synthetischen Tests Faktor 2x!
In der Praxis jedoch kaum messbar!

Padding und Arrays

- In C liegen Arrays im Speicher ohne Padding
- Sind die Speicherzugriffe optimal?

```
struct container
{
   char c;
   int i;
};
```



Unions in C

Idee: Verwende Speicherbereiche wieder, wenn Programmabschnitte unabhängig voneinander sind

- Vorteile
  - Gewinn an nutzbarem Speicher
  - Keine Performancekosten
- Nachteile
  - Erfordert viel manuelles Tuning
  - Programm wird komplex (-> schwer zu findene Bugs)
  - Nicht immer anwendbar
- Lohnt der Aufwand?

Unions in C

```
union floatOrInit
{
  float float;
  int int;
};

int main (void)
{
  union floatOrInt number;

  number.float = 4.0f;

  number.int = 17;
}
```

Datenkompression im Speicher

Vorteile

- Mehr Daten im Speicher als eigentlich vorhanden
- Lesen ist immer noch relativ schnell

Nachteile

- Nur auf statischen Daten sinnvoll
- Nicht alle Daten komprimieren gut
- Implementation aufwändig

$\Rightarrow$ Selber testen, ob die eigene Anwendung optimierbar ist!

Bibliotheken: *zlib*, *heatshrink*

Zusammenfassung

- Verwaltung
  - Komplex und fehleranfällig
  - Manuelle Verwaltung ist ein mächtiges Werkzeug
- Optimierung
  - Hoher Aufwand
  - Verbesserung schwer vorherzusagen

"Premature optimization is the root of all evil"

Donald Knuth

Quellen

- Performanceeinfluss von automatisierten Structurepacking
  http://www.alexonlinux.com/aligned-vs-unaligned-memory-access
  https://attractivechaos.wordpress.com/2013/05/02/does-packed-struct-hurt-performance-on-x86_64
- Tim Wischhof - Memory Management and Optimizations
  https://wr.informatik.uni-hamburg.de/_media/teaching/wintersemester_2017_2018/ep-1718-wischhof-memory-management-praesentation.pdf
- Eric S. Raymond - The Lost Art of Structure Packing
  http://www.catb.org/esr/structure-packing
- Andrew Tanenbaum Modern Operating Systems

Quellen

- Jakob Rieck - Pointers and dynamic memory management in C
  https://wr.informatik.uni-hamburg.de/_media/teaching/sommersemester_2014/cgk-14-rieck-zeig-e-und-dynamische-_speicherverwaltun-report.pdf
- A Survey on Computer System Memory Management and Optimization Techniques
  http://article.sapub.org/10.5923.j.ajca.20120103.01.html
- zlib
  https://www.zlib.net/
- heatshrink
  https://github.com/atomicobject/heatshrink