Speicherverwaltung in C

Tobias Gutzmann, Le Xuan Khanh, Robert Hartmann

19.04.2005

Inhalt

- Übersicht der wichtigsten Befehle
  - malloc, free, realloc
  - alloca, obstack, brk
- Speicherverwaltung in C
  - allg. Methodik der Speicherverwaltung
  - Garbage Collection in C?
- Speicherfehlersuche

Übersicht der wichtigsten Befehle

- malloc
- free
- realloc
- calloc
- memalign

malloc,free, realloc (stdlib.h)

- **void *malloc(size_t $size$);**

  fordert einen dynamischen Speicherbereich der Größe size in Byte an

  Rückgabe von Zeiger auf den Speicherbereich oder $NULL$, falls die Anforderung scheitert

- **void free(void *ptr);**

  Freigabe des Speichers, auf den der Zeiger ptr verweist, bei $NULL$ wird keine Operation ausgeführt

malloc, free, realloc (stdlib.h) (cont.)

- **void *realloc(void *ptr, size_t $size$);**

  ändert die Größe des Speicherbereiches des Zeigers ptr auf die Größe $size$ Byte

  Minimum(alte Größe, neue Größe) vom reservierten Bereich bleibt gleich

  neuer Speicherbereich bleibt uninitialisiert

malloc, free, realloc (stdlib.h) (cont.)

- **void *realloc(void *ptr, size_t $size$);** (cont.)

$$\begin{cases} ptr == NULL & \text{dann wie malloc}(size); \\ size == NULL & \text{dann wie free}(ptr)\text{,aber mit } NULL \text{ als Rückgabewert} \\ erfolgreich & \text{dann Rückgabe des Zeigers auf den neuen Bereich} \\ scheitert & \text{Rückgabewert } NULL \text{ und keine Veränderung} \end{cases}$$

Weitere Befehle

- **void *calloc(size_t $nmemb$, size_t $size$);**[1]
- **void *memalign(size_t $size$, size_t $alignment$);**
- da alles Bibliotheksfunktionen sind, können diese auch selbst implementiert werden

  beispielsweise, kann man sich die Zeilennummer und Datei mit speichern, um Speicherfehler aufspüren zu können. (mymalloc(size_t $size$,$\_\_FILE\_\_$,$\_\_LINE\_\_$))

[1] stdlib.h

Übersicht der wichtigsten Befehle

- alloca
- obstack
- brk

alloca, obstack, brk

- **void *alloca(size_t $size$);**

  Speicherbereich der Größe $size$ Byte auf dem *Stack* angelegt

  Freigabe erfolgt automatisch sobald der Aufrufer endet

  Problem: Keine wirkliche Funktion, sondern ein Makro.

  AB C99 wurde der Standard um VAL[2] erweitert. D.h. direkte Unterstützung der Sprache für dyn. Objektallocation auf dem Stack.

[2]Variable Array Length

alloca, obstack, brk (cont.)

- **int brk(void *$end\_data\_segment$);** - Systemaufruf

  Ziel: Halde oder Stack zu vergrößern oder zu verkleinern

  setzt das Ende des Datensegmentes auf den Wert $end\_data\_segment$

  mit *brk* kann *malloc* implementiert werden, Vergleich Betriebssystem

$$\begin{cases} true & 0 \\ false & -1, \text{und errno auf ENOMEM} \end{cases}$$

alloca, obstack, brk (cont.)

- **obstack - Bibliothek**

  Verwaltung von mehreren Stacks auf der Halde

  Objekte werden in *Chunks* gehalten, die vom Obstack-Objekt dynamisch verwaltet werden

  Hauptziel: dynamische Verwaltung von wachsenden Objekten.

alloca, obstack, brk (cont.)

- **obstack - Bibliothek (cont.)**

  1. obstack_chunk_alloc[3] und Obstack_chunk_free[4] müssen zuerst definiert werden
  2. Anlegen eines Obstack-Objektes (int obstack_init(struct obstack *obstack-ptr))
  3. obstack_copy(struct obstack *obstack-ptr, void *address, int size)
  4. obstack_grow(struct obstack *obstack-ptr, void *data, int size);

[3] # define obstack_chunk_alloc malloc
[4] # define obstack_chunk_free free

alloca, obstack,brk (cont.)

- **obstack - Bibliothek (cont.)**

  Anlegen eines Objektes mit unbekannter länge

  Speicherbelegung ist sequentiell, deshalb können mit *obstack_grow* Objekte nacheinander angelegt werden

  diese Objekte werden mit *void *obstack_finish(struct_obstack *obstack_ptr)* zu einem neuen Objekt zusammengeführt.

  Problem: Test bei jedem einfügen von Objekten, ob noch genügend Speicher vorhanden ist.
  Lösung: mittels $fast\ obstack$

Speicherverwaltung in C

- allg. Methodik der Speicherverwaltung
- Garbage Collection in C?

allg. Methodik der Speicherverwaltung

- Einteilung in zwei Phasen
  1. Suchen von unnützen Objekten
  2. Entfernen von unnützen Objekten

- einfachste Methode: Zeiger zählen[5] (Problem: Zyklisch verlinkte Objekte)

[5] reference counting

allg. Methodik der Speicherverwaltung (cont.)

- statt

  $d.f := p$

  nun

  $z := d.f;\ c := z.count;$
  $c := c + 1;\ z.count := c;$
  $if\ c == 0\ call\ putOnFreeList;$
  $d.f := p;\ c := p.count;$
  $c := c + 1;\ p.count := c;$

- Nachteil: es werden mehr Operationen gebraucht.

allg. Methodik der Speicherverwaltung (cont.)

- markieren und aufräumen[6] (Problem: Fragmentierung des Hauptspeichers)

  Markiere alle erreichbaren Objekte aus der Grundmenge(Register, Stack, Objekte).

  Gehe alle dyn. allozierten Obejkte durch und entferne die ohne Markierung.

[6] mark and sweep

Garbage Collection in C?

- GC Bibliothek von Böhm

  einfache Speicheranforderung über GC_malloc oder GC_realloc

  weitere Spezialbefehle wie GC_atomic_malloc siehe Bibliothek.

Speicherfehlersuche

– Typische Speicherfehler
  1. Nutzung von nicht initialisierten Variablen
  2. Nutzung von nicht deklariertem Speicher
  3. Nutzung von freigegebem Speicher
  4. Nutzung von Speicher der mit malloc(0) angefordert wurde
  5. Überlauf von Feldgrenzen
  6. Speicherlöcher

Speicherfehlersuche

- Manuelle Suche sehr schwer, insbesondere bei nebenläufigen Systemen
- Programm unterstützung von Valgrind, Electric Fence und gdb
- Bei Electric Fence nutzt man den Seitenfehler um ein Segmentation Fault auszulösen, dabei bedient man sich am Zugriffsschutz der MMU.
- dazu werden die Grenzen des angeforderte Speicher auf die Kachelgrenzen abgebildet.

Ende