Dynamische Speicherverwaltung

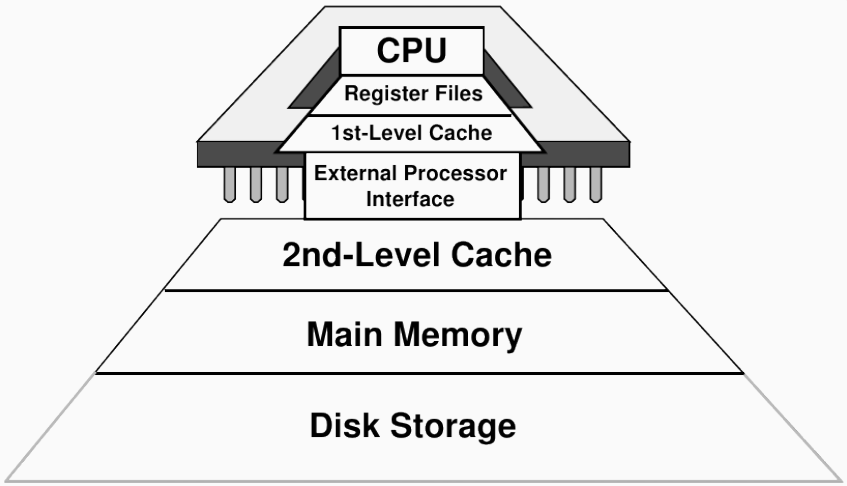
Tim Dobert

17.05.2013

Gliederung

Speicherverwaltung

- Verschiedene Arten von Speicher
- Virtuelle Speicherverwaltung $\rightarrow$ kontinuierlicher Adressraum
- Jedem Programm wird ein Teil davon zur Verfügung gestellt



[1]

Speichernutzung eines Programms

Speichernutzung eines Programms

- Constant Pool
  - Konstanten

Speichernutzung eines Programms

- Constant Pool
  - Konstanten
- Stack
  - Lokale Variablen
  - Parameter

Speichernutzung eines Programms

- Constant Pool
  - Konstanten
- Stack
  - Lokale Variablen
  - Parameter
- Heap
  - Freier Speicher, der alloziert werden kann

Verwendung

Verwendung

- Arrays dynamisch zur Laufzeit erzeugen und vergrößern
- Dynamisches Erzeugen und Löschen von Structs

Zur Übersetzungszeit muss nicht feststehen, wie viel Speicher gebraucht wird.

Anwendung in C

Allgemeines

- Das System reserviert Speicher und gibt seine Adresse zurück
- Die 4 wichtigsten Befehle sind in der Standardbibliothek
  $\Rightarrow$ #include <stdlib.h>

4 Grundbefehle

1. malloc
2. calloc
3. realloc
4. free

malloc

malloc reserviert Speicherplatz und gibt einen Pointer auf den Anfang zurück.

Signatur von malloc:

```
void* malloc(size_t)
```

Code Beispiel:

```
int* number = (int*) malloc(sizeof(int));
*number = 3;
//Es wurde eine 3 abgespeichert
```

calloc

calloc reserviert ebenfalls Speicherplatz und initialisiert diesen mit Null.

Signatur von calloc:

```
void* calloc(int, size_t)
```

Code Beispiel:

```
int* number = calloc(1, sizeof(int));
printf("%d" ,*number);
//Es wird eine 0 ausgegeben
```

realloc

realloc vergrößert oder verkleinert bereits reservierten Speicherplatz.

Signatur von realloc:

```
void* realloc(void*, size_t)
```

Code Beispiel:

```
int* array = malloc(10 * sizeof(int));
array = realloc(array, 20 * sizeof(int));
//z.B. array[15] kann jetzt verwendet werden
```

free

free gibt reservierten Speicher wieder frei.

Signatur von free:

```
void free(void*)
```

Code Beispiel:

```
int* number = malloc(sizeof(int));
free(number);
//z.B. *number = 3; führt jetzt zu Fehlern!
```

Beispielprogramm: Wachsendes Array

Ausschnitte des Codes 1

Aufbau des Arrays

```
struct DynamicArray{
    int *point;
    unsigned int size;
    unsigned int nextFree;
};
```

Initialisieren

```
void initArray(){
    array.size = 2;
    array.nextFree = 0;
    array.point = malloc(sizeof(int) * array.size);
}
```

Ausschnitte des Codes 2

Einfügen eines Elements

```
void add(int element)
{
    if(array.nextFree >= array.size)
    {
        array.size *= 2;
        int newSize = sizeof(int)*array.size;
        array.point = realloc(array.point, newSize);
    }
    array.point[array.nextFree] = element;
    ++array.nextFree;
}
```

Ausschnitte des Codes 3

Beenden des Programms

```
void finalize()
{
    printf("Speicherplatz wird freigegeben...\n");
    free(array.point);
    array.point = NULL;
}
```

Vergleich mit mit anderen Sprachen

Vergleich mit Java

- Java bietet keine vergleichbaren Befehle an
- Speicherplatz wird automatisch mit `new` alloziert
- Um die Freigabe kümmert sich der Garbage Collector

Vergleich mit C++

- Die gleichen Befehle sind verfügbar
- Normalerweise wird der `new`-Operator benutzt
- Freigabe mit `delete(*void)`

Praktische Tipps

Zwei beliebte Fehler

- Speicherplatz wird nicht freigegeben
  -> free am Ende des Sichtbarkeitsbereiches benutzen

- Der Zeiger zeigt auf freigegebenen Speicher
  -> nach `free(pnt)` sollte immer `pnt=NULL` folgen

Sonstige Tipps

- Sicherheitsvorkehrungen. Speicherfehler sind übel
  -> z.B. Größe des reservierten Speichers merken

- realloc benutzen, wenn möglich

Zusammenfassung

- Dynamische Speicherverwaltung nutzt den *Heap*.
- *malloc* und *calloc* reservieren Speicher
- *realloc* kann reservierten Speicher vergrößern/verkleinern
- *free* gibt Speicher wieder frei (wichtig!)
- Mit diesen Befehlen lassen sich leicht wachsende Strukturen implementieren
- Objektorientierte Sprachen abstrahieren durch `new`
- Aufmerksam sein und Zeiger auf *NULL* setzen

Quellen


Rechnerstrukturen Skript Kapitel 17,
http://tams.informatik.uni-hamburg.de/lectures/2012ws/vorlesung/rs/doc/rsWS12-18.pdf


C dynamic memory allocation,
http://en.wikipedia.org/wiki/C_dynamic_memory_allocation


Advanced Memory Management: Dynamic Allocation,
http://www.cprogramming.com/tutorial/dynamic_memory_allocation.html