Memory management in C++

Simon Rettberg

December 1, 2010

Topics

1. Address space of a process
2. Memory management in C++
3. Platform specific issues

Topics

1. Address space of a process
2. Memory management in C++
3. Platform specific issues

Virtual address space of a process

Schematics of the address space

- **Virtual:** Every process only sees its own address space
- Real memory usage grows with allocated heap space
- ...was already explained in second talk

Stack vs. heap

Stack:

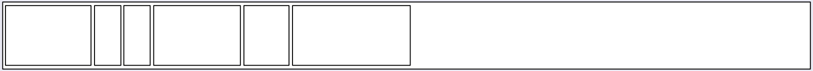
- Limited space
- "Automatic" freeing of variables
- Faster allocation of variables
- Variables that got allocated last will be deleted first

Heap:

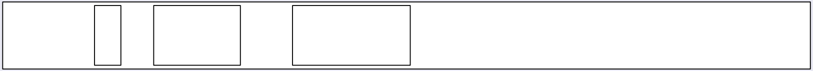
- Much more space (still limited)
- Manual freeing of variables and memory blocks
  C++: new/delete, C: malloc()/free()
- Management overhead when allocating/freeing
- Allocation and freeing can happen in any order
  $\rightarrow$ fragmentation can occur

Allocating and freeing memory

Heap after some allocations



A few blocks have been freed again



More allocations

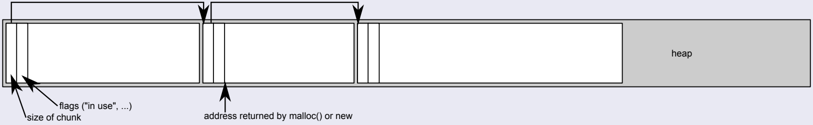
Keeping track of the heap

The heap needs to be managed:

- Keep track of which areas are allocated
- Which areas are free
- Where the heap currently ends

Different approaches to this.

Simplified example of heap management



Know your limits

C++ doesn't do any bounds checking for you

- Make sure your pointers stay inside buffers
- Don't use any pointers that point to freed memory
- Another source for horrible bugs is a double-delete/free

Breaking any of these rules can crash your program immediately, or even worse, produce really weird behaviour later on, so it takes you hours to track down the real source of the problem.

Buffer overflows can be exploited

```
int x = 5;
char temp[200];
gets(temp);
```

What happens on the stack

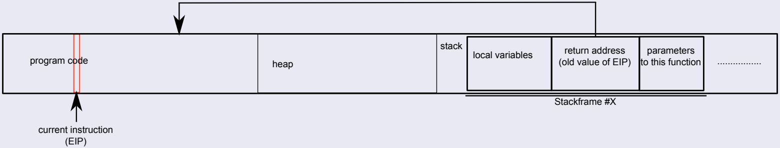
char temp[200];	int x = 5;	old EBP	return-address	....
<uninitialized>	0x00000005	0x31556900	0x12345678	

after executing "gets(temp);"
with user input "<200*x>JKLMDDDD9876"

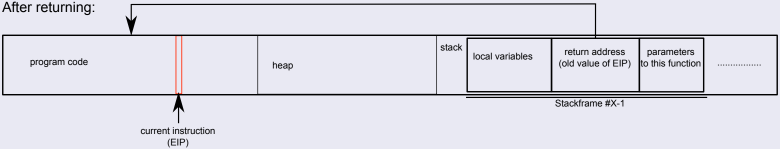
char temp[200];	int x = 5;	old EBP	return-address	....
"xxxxx....xx"	0x4d4c4b4a	0x44444444	0x36373839	
	"JKLM"	"DDDD"	"9876"	

Normal return from a function

What happens on return

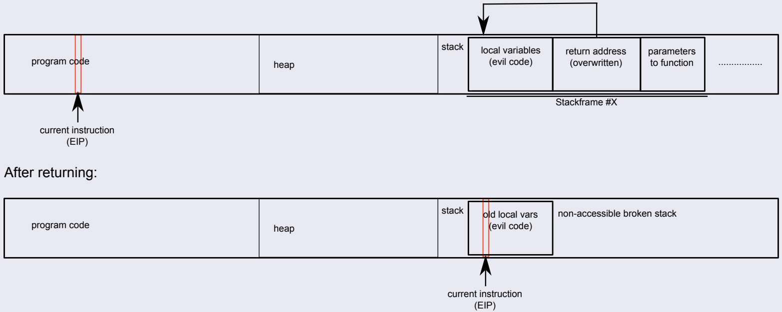


After returning:



Exploited return from a function

What happens on return



Topics

1. Address space of a process
2. Memory management in C++
3. Platform specific issues

Garbage collection in C++

- There is no fully automatic garbage collection in C++
$\rightarrow$ allocated memory has to be kept track of and freed if not needed anymore
- Who is the owner of an object or memory area and responsible for deleting
(important when dealing with libraries, especially C-only)
- In C++, thanks to classes having constructors and desctructors, it is easy to maintain a clear hierarchy

RAII

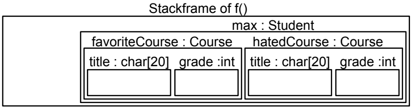
Resource Acquisition Is Initialization

- The object that allocates a memory block is also responsible for deleting it
- Objects can easily be nested this way (see next slides)
- Requires all classes to adhere to this concept

RAII

Automatic memory management in C++ (stack)

```
class Course {
        char title[20];
        int grade;
};

class Student {
        Course favoriteCourse;
        Course hatedCourse;
};

int f() {
        Student max;
} // leaving f will destroy max
```



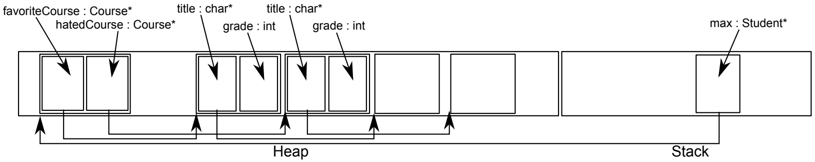
Automatic memory management in Java (heap)

```
class Course {
        char[] title = new char[20];
        int grade = 0;
}

class Student {
        Course favoriteCourse = new Course();
        Course hatedCourse = new Course();
}

int f() {
        Student max = new Student();
} // garbage collector of java will
// take care of cleaning memory up
```

RAII

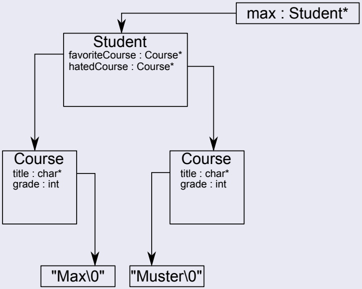
Manual memory management in C++ (heap)

```
class Course {
        char *title;
        int grade;
public:
        Course() { title = new char[20]; }
        ~Course() { delete[] title; }
};
class Student {
        Course *favoriteCourse, *hatedCourse;
public:
        Student() { favoriteCourse = new Course(); hatedCourse = new Course(); }
        ~Student() { delete favoriteCourse; delete hatedCourse; }
};
int f() {
        Student *max = new Student();
        delete max; // object hierarchy gets deleted
}
```



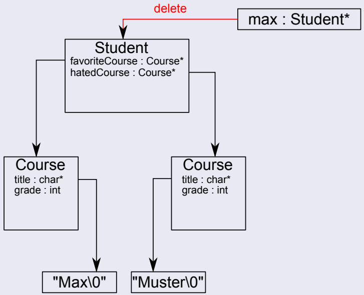
RAII - A simple example

deleting a simple object hierarchy



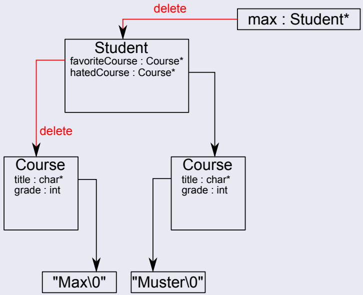
RAII - A simple example

deleting a simple object hierarchy



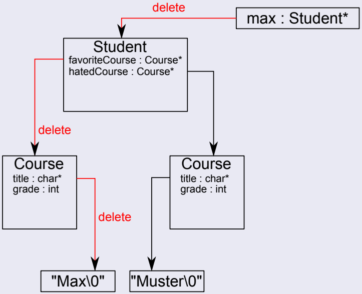
RAII - A simple example

deleting a simple object hierarchy



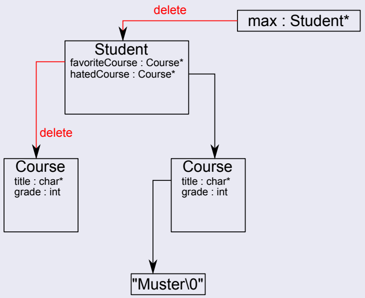
RAII - A simple example

deleting a simple object hierarchy



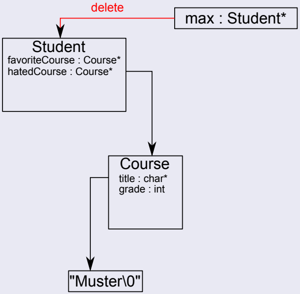
RAII - A simple example

deleting a simple object hierarchy



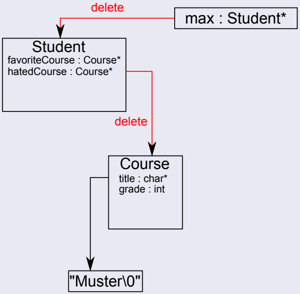
RAII - A simple example

deleting a simple object hierarchy



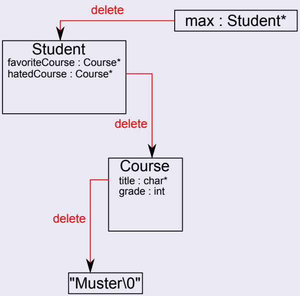
RAII - A simple example

deleting a simple object hierarchy



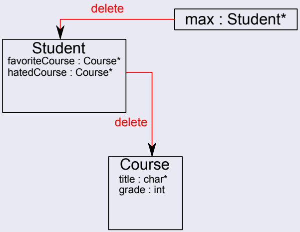
RAII - A simple example

deleting a simple object hierarchy



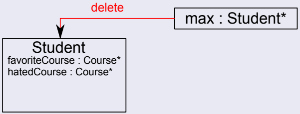
RAII - A simple example

deleting a simple object hierarchy



RAII - A simple example

deleting a simple object hierarchy



RAII - A simple example

deleting a simple object hierarchy

max : Student*

Live demo

Live demo: C++ vs. Java

Topics

1. Address space of a process
2. Memory management in C++
3. Platform specific issues

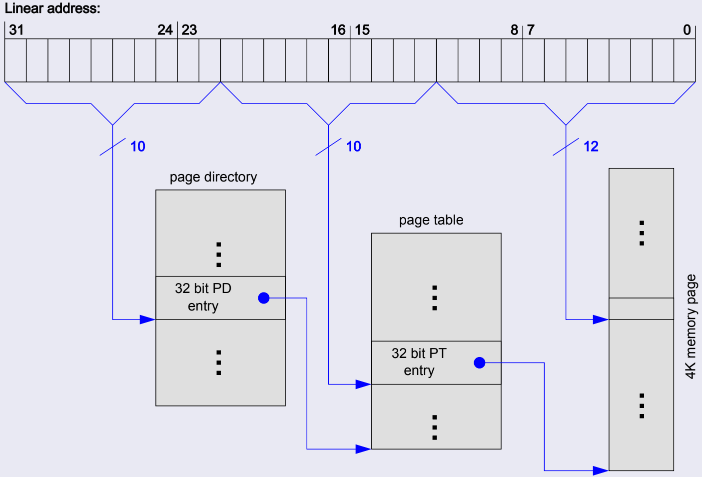
Paging (1)

What is the paging unit?

- Memory management unit
- Translation from virtual addresses (as seen by the process) to real addresses
- Supports swapping (move memory blocks to external storage if another process needs more physical memory)
- Fully transparent to processes
- Not a part of C++, affects memory management in general on modern architectures

Paging (2)

Translation from virtual to real addresses



Endianness

The endianness of a platform decides in which order multibyte integers are represented in memory

Example: `int i = 300; // 4 byte integer: 0x0000012C`

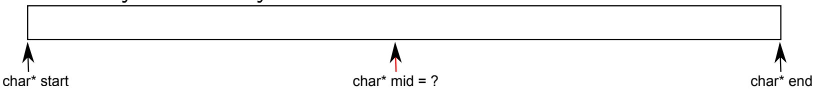
Little endian (eg. x86)	0x2C	0x01	0x00	0x00
Big endian (eg. PowerPC)	0x00	0x00	0x01	0x2C

→ Be careful when writing serializers or network apps (or 300 might become 738263040)

Why only 2/3 GB on 32bit?

- Last gigabyte is reserved for kernel libraries, drivers etc.
- On Windows by default only 2 GB (so highest bit of pointers is always 0)

Bonus question: Given two pointers to the beginning and the end of an array. How do you calculate the address of the middle?



Why only 2/3 GB on 32bit?

Assume `start = 0xA1112200`, `end = 0xA1112244`

Simple approach:

```
char *mid = (start + end) / 2;
```

```
   0xA1112200
 + 0xA1112244
 ------------
  0x142224444
```

Overflow! → `0x42224444 / 2 = 0x21112222`

Safe approach:

```
char *mid = start + (end - start) / 2;
```

```
   0xA1112244
 - 0xA1112200
 ------------
   0x00000044
```

→ `0x00000044 / 2 = 0x00000022`

```
   0xA1112200
 + 0x00000022
 ------------
   0xA1112222
```

No overflow, `0xA1112222` is correct.

That's all, folks! Any questions?