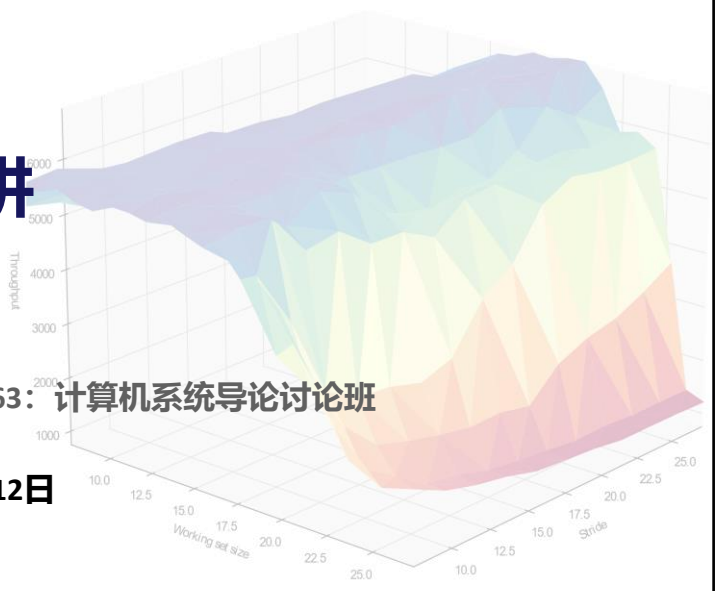
第二讲



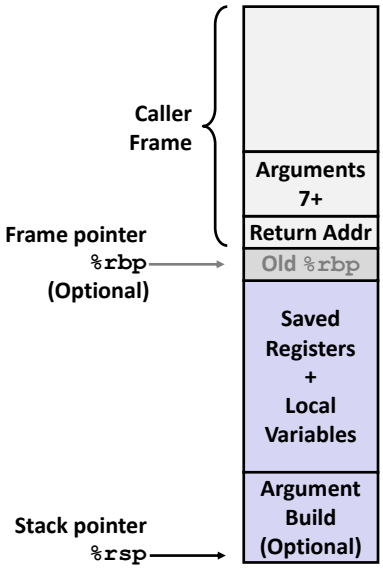
PKU 04832363：计算机系统导论讨论班

王畅

2021年10月12日

x86-64/Linux Stack Frame

- **Current Stack Frame ("Top" to Bottom)**
  - "Argument build:" Parameters for function about to call
  - Local variables If can't keep in registers
  - Saved register context
  - Old frame pointer (optional)
- **Caller Stack Frame**
  - Return address
    - Pushed by **`call`** instruction
  - Arguments for this call



2

需要仔细理解整个右边的栈帧是如何产生的。首先空白部分表示caller栈帧，即将调用callee（参数超过6个）时引入arguments部分，正式call时再引入返回地址。有时候需要存储%rbp，这是因为%rbp是被调用者保存寄存器，而%rbp经常需要用来作为帧基址（比如变长栈帧）。

Understanding Pointers & Arrays #1

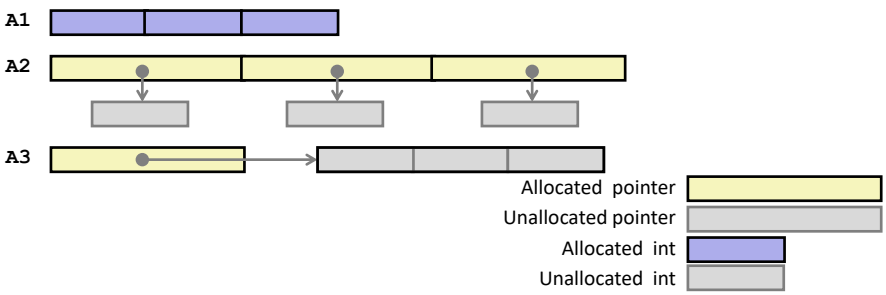
Decl	A1 , A2			*A1 , *A2		
	Comp	Bad	Size	Comp	Bad	Size
int A1[3]	Y	N	12	Y	N	4
int *A2	Y	N	8	Y	Y	4

- **Comp: Compiles (Y/N)**
- **Bad: Possible bad pointer reference (Y/N)**
- **Size: Value returned by `sizeof`**

简单，惟需要注意虽然数组的名称可以认为是指针，但其`size`是整个数组的大小。

Understanding Pointers & Arrays #2

Decl	An			*An			**An		
	Cmp	Bad	Size	Cmp	Bad	Size	Cmp	Bad	Size
int A1[3]	Y	N	12	Y	N	4	N	–	–
int *A2[3]	Y	N	24	Y	N	8	Y	Y	4
int (*A3)[3]	Y	N	8	Y	Y	12	Y	Y	4



4

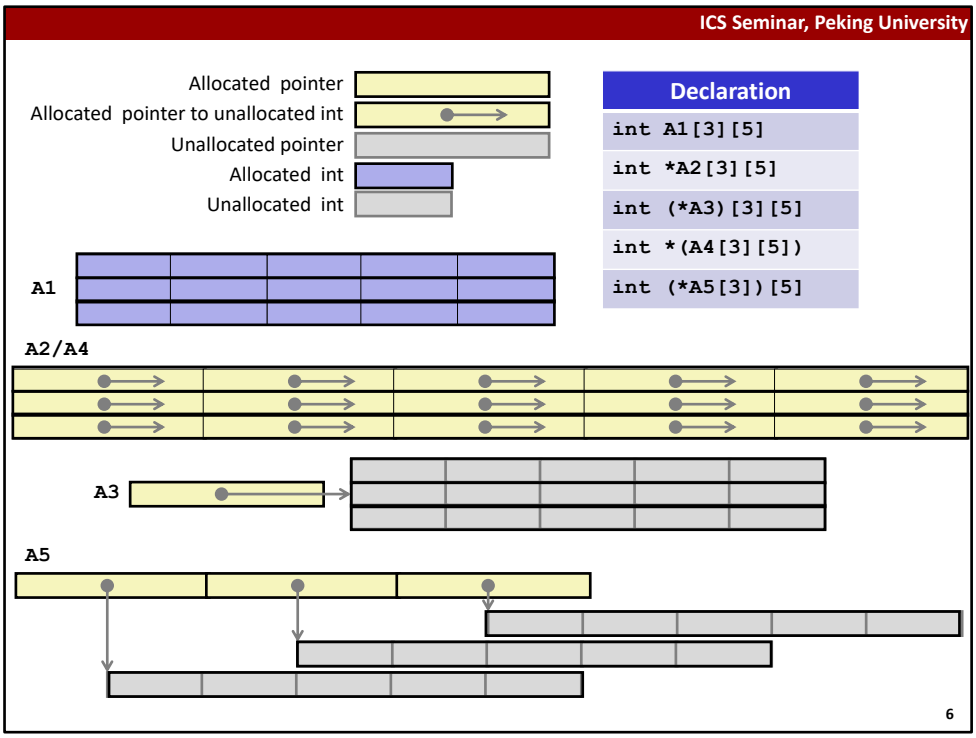
(*A3)[3]表示的是指向一个有3个元素的数组的基址的指针。换言之*A3是那个3个元素的数组的基址（因此取大小是12），**A3是第一个元素。

Understanding Pointers & Arrays #3

Decl	An			*An			**An		
	Cmp	Bad	Size	Cmp	Bad	Size	Cmp	Bad	Size
`int A1[3][5]`									
`int *A2[3][5]`									
`int (*A3)[3][5]`									
`int *(A4[3][5])`									
`int (*A5[3])[5]`									

- **Cmp: Compiles (Y/N)**
- **Bad: Possible bad pointer reference (Y/N)**
- **Size: Value returned by `sizeof`**

Decl	***An		
	Cmp	Bad	Size
`int A1[3][5]`			
`int *A2[3][5]`			
`int (*A3)[3][5]`			
`int *(A4[3][5])`			
`int (*A5[3])[5]`			



判断方法：在小括号中的表示的是指针数组本身的几何形状，在小括号外面的表示的是指针数组所指向的内容的几何形状。

原理：先声明类型，然后指明指向元素的类型（可与后面讲到的函数指针作类比）。

例如`(*A5[3])[5]`表示的是一个3个元素的指针数组（`A5`的`size`是`24`），每个数组元素是一个5元素数组的基址的指针；即`*A5`是第一个指针（取大小得到8），`**A5`是第一个5元素数组的基址（取大小得到`20`），`***A5`是该5元素数组的第一个数（取大小得到`4`）。

Understanding Pointers & Arrays #3

Decl	An			*An			**An		
	Cmp	Bad	Size	Cmp	Bad	Size	Cmp	Bad	Size
int A1[3][5]	Y	N	60	Y	N	20	Y	N	4
int *A2[3][5]	Y	N	120	Y	N	40	Y	N	8
int (*A3)[3][5]	Y	N	8	Y	Y	60	Y	Y	20
int *(A4[3][5])	Y	N	120	Y	N	40	Y	N	8
int (*A5[3])[5]	Y	N	24	Y	N	8	Y	Y	20

- Cmp: Compiles (Y/N)
- Bad: Possible bad pointer reference (Y/N)
- Size: Value returned by `sizeof`

Decl	***An		
	Cmp	Bad	Size
int A1[3][5]	N	-	-
int *A2[3][5]	Y	Y	4
int (*A3)[3][5]	Y	Y	4
int *(A4[3][5])	Y	Y	4
int (*A5[3])[5]	Y	Y	4

Activity 1

轮流回答问题

Question 1

- **下列程序共有8行输出，说出每行是什么**

```
int a[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};
int b[4], *c[4];
int(*d)[4] = a + 1;
1. cout << d[1] << endl;          0x7fffff6594e0
2. cout << *d[1] << endl;         9
3. cout << (*d)[1] << endl;       6
4. cout << *(*d + 1) << endl;     6
5. cout << a[2] << endl;          0x7fffff6594e0
6. cout << *a[2] << endl;         9
7. cout << (*a)[2] << endl;       3
8. cout << *(*a + 2) << endl;     3
```


5~8空是简单的。注意d表示的是一个指向4元素数组基址的指针。所以*d是{5, 6, 7, 8}的基址，d[1] = *(d+1)是{9, 10, 11, 12}的基址，*d[1]是其第一个元素9，(*d)[1]是{5, 6, 7, 8}的第2个元素，等等。

Question 2

- **下列代码声明的是什么？具体阐述**

```
int (*func(int (*input)(int), int arg))(int, int);
```


先回忆程设中学过的函数指针的基本概念。

作以下对比：

`int *pfun(int, int);`是一个返回值为整型指针的函数；

`int (*pfun)(int, int);`将`*`与`pfun`结合，表示`pfun`是一个指针，后面跟的括号表示它是函数指针，返回值是`int`，参数是双`int`。

同样，由于`*func`没有括号，说明`func`是一个函数，后面括号中`int (*input)(int), int arg`表示它的参数是一个返回`int`，输入单`int`的函数指针，以及一个`int`。`func`与前面的`*`结合，说明它的返回值是一个指针。然后再与后面的`(int, int)`结合，说明该指针指向的是一个函数。

以上声明可以用`typedef`简化：`typedef int (*ret_t)(int, int); ret_t func(int (*input)(int), int arg);`

Activity 2

问题求解

考点1：对齐

- Section 4 练习14、15
- Section 5 练习17、18、22、23、24
- 有可能不单独考，做综合题时要细心！

考点2：参数传递

- 六个传递参数寄存器的顺序
- Section 4 练习12

考点3：函数调用/栈管理（难点！）

- %rbp、%rsp寄存器的用法
- 调用者保存和被调用者保存
- 栈帧的建立和管理过程（重点）
- Section 4 练习13
- Section 5 练习19

考点4：综合汇编阅读（难点！）

- 期中考试两大区分点之一
- 先通读题目再分析
- 掌握基本技巧，熟练：作对应、画栈帧、推断
- 注意字节序和对齐要求！！！
- Section 4　习题16
- 历年试题：解答题第二题（下次选讲）

any questions?

Thanks & 感谢观看