Machine Prog: Data & Advanced

TA: 王非石 朱睿冬

Outline

- Array
- Structure
- Floating Point
- *Additional Exercise : Pointers & Dereference
- 可变长栈帧
- Buffer Overflow
- *Memory Layout

定长数组

循环遍历定长数组时，编译器会进行优化！

```
#define N 16
typedef int fix_matrix[N][N];
```

原始代码

```
/* Compute i,k of fixed matrix product */
int fix_prod_ele (fix_matrix A, fix_matrix B, long i, long k) {
    long j;
    int result = 0;

    for (j = 0; j < N; j++)
        result += A[i][j] * B[j][k];

    return result;
}
```

优化后的代码

```
/* Compute i,k of fixed matrix product */
int fix_prod_ele_opt(fix_matrix A, fix_matrix B, long i, long k) {
    int *Aptr = &A[i][0];    /* Points to elements in row i of A    */
    int *Bptr = &B[0][k];    /* Points to elements in column k of B */
    int *Bend = &B[N][k];    /* Marks stopping point for Bptr       */
    int result = 0;
    do {                          /* No need for initial test */
        result += *Aptr * *Bptr;  /* Add next product to sum  */
        Aptr ++;                  /* Move Aptr to next column */
        Bptr += N;                /* Move Bptr to next row    */
    } while (Bptr != Bend);       /* Test for stopping point  */
    return result;
}
```

定长数组

访问定长的高维数组时，编译器会进行优化！

```
#define N 16
typedef int fix_matrix[N][N];
```

```
/* Get element A[i][j] */
int fix_ele(fix_matrix A, size_t i, size_t j) {
  return A[i][j];
}
```

```
  # A in %rdi, i in %rsi, j in %rdx
  salq    $6, %rsi              # 64*i
  addq    %rsi, %rdi            # A + 64*i
  movl    (%rdi,%rdx,4), %eax   # Mem[A + 64*i + 4*j]
  ret
```

可变长数组

```
int var_ele(long n, int A[n][n], long i, long j) {
    return A[i][j];
}
```

```
int var_ele(long n, int A[n][n], long i, long j)
n in %rdi, A in %rsi, i in %rdx, j in %rcx
var_ele:
  imulq   %rdx, %rdi              Compute n · i
  leaq    (%rsi,%rdi,4), %rax     Compute xA + 4(n · i
  movl    (%rax,%rcx,4), %eax     Read from M[xA + 4(n · i) + 4j]
  ret
```

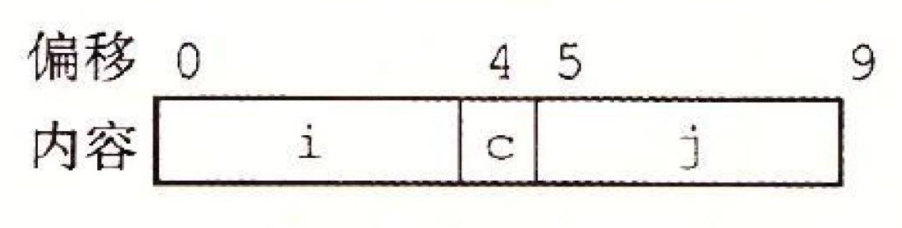
由于不定长，编译器不能用sal，leaq等指令加速乘法，被迫使用较慢的imulq

Structure

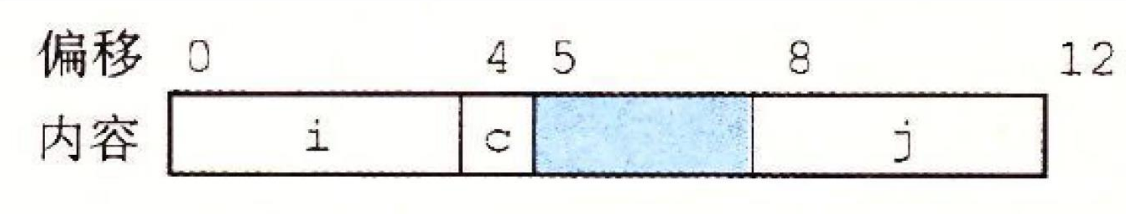
```
struct S1 {
    int  i;
    char c;
    int  j;
};
```

- 对于x86机器而言，处理未对齐的数据仍可正常运行，但是会影响效率；
- 对于其他的一些机器而言，处理未对齐的数据可能导致内存错误。

- 对齐前：



- 对齐后：

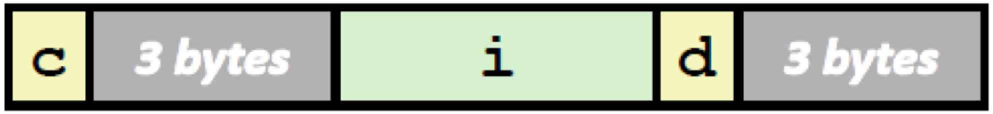


！注意结构的末尾可能需要对齐（结构数组）
！注意栈上的参数值总是8字节对齐

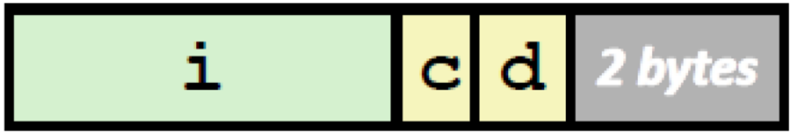
Structure

Saving Space：贪心思想

```
struct S4 {
  char c;
  int i;
  char d;
} *p;
```




```
struct S5 {
  int i;
  char c;
  char d;
} *p;
```



Union

Union中所有成员都从低地址开始存放，可据此判断大端/小端

Big Endian

f0	f1	f2	f3	f4	f5	f6	f7
c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
s[0]		s[1]		s[2]		s[3]	
i[0]				i[1]			
l[0]							

MSB → LSB (Print) MSB LSB

Little Endian

f0	f1	f2	f3	f4	f5	f6	f7
c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
s[0]		s[1]		s[2]		s[3]	
i[0]				i[1]			
l[0]							

LSB ← MSB (Print) LSB MSB

Output:

```
Characters 0-7 == [0xf0,0xf1,0xf2,0xf3,0xf4,0xf5,0xf6,0xf7]
Shorts     0-3 == [0xf1f0,0xf3f2,0xf5f4,0xf7f6]
Ints       0-1 == [0xf3f2f1f0,0xf7f6f5f4]
Long       0   == [0xf3f2f1f0]
```

Output on Sun:

```
Characters 0-7 == [0xf0,0xf1,0xf2,0xf3,0xf4,0xf5,0xf6,0xf7]
Shorts     0-3 == [0xf0f1,0xf2f3,0xf4f5,0xf6f7]
Ints       0-1 == [0xf0f1f2f3,0xf4f5f6f7]
Long       0   == [0xf0f1f2f3]
```

Floating Point

%xmm0 返回值

%xmm0~7 8个参数

%xmm0~15 所有均为caller saved

这部分了解即可，
往年没有深入考察过


```
float fadd(float x, float y)
{
    return x + y;
}
```

```
double dadd(double x, double y)
{
    return x + y;
}
```

```
  # x in %xmm0, y in %xmm1
  addss    %xmm1, %xmm0
  ret
```

```
  # x in %xmm0, y in %xmm1
   addsd    %xmm1, %xmm0
   ret
```

Additional Exercise : Pointers & Dereference

	`sizeof(A)`	What is `A`?
`int *A[3];`		
`int *(A[3]);`		
`int (*A)[3];`		
`int (*A[3]);`		
`int (*A[3])();`		
`int (*A[3])[5];`		

Hint:

Operator	Read as	Priority
`type A(parameters)`	Function with `parameters` and return type `type`	0
`type [N]`	Array of `N` `type`	1
`*`	Pointer to	2

Additional Exercise : Pointers & Dereference

	sizeof(A)	What is A?
int *A[3];	24	Array of 3 int*
int *(A[3]);	24	Array of 3 int*
int (*A)[3];	8	Pointer to array of 3 int
int (*A[3]);	24	Array of 3 int*
int (*A[3])();	24	Array of 3 pointers to a function with no parameter and return type int
int (*A[3])[5];	24	Array of size 3 pointers to array of size 5 of int

Hint:

Operator	Read as	Priority
type A(parameters)	Function with parameters and return type type	0
type [N]	Array of N type	1
*	Pointer to	2

可变长栈帧

```
long vframe(long n, long idx, long *q)  {
    long i;
    long *p[n];
    p[0] = &i;
    for (i = 1; i < n; i++)
        p[i] = q;
    return *p[idx];
}
```

```
long vframe(long n, long idx, long *q)
n in %rdi, idx in %rsi, q in %rdx
Only portions of code shown
vframe:
  pushq   %rbp                    Save old %rbp
  movq    %rsp, %rbp              Set frame pointer
  subq    $16, %rsp               Allocate space for i (%rsp = s1)
  leaq    22(,%rdi,8), %rax
  andq    $-16, %rax
  subq    %rax, %rsp              Allocate space for array p (%rsp = s2)
  leaq    7(%rsp), %rax
  shrq    $3, %rax
  leaq    0(,%rax,8), %r8         Set %r8 to &p[0]
  movq    %r8, %rcx               Set %rcx to &p[0] (%rcx = p)
    . . .
Code for initialization loop
i in %rax and on stack, n in %rdi, p in %rcx, q in %rdx
.L3:                            loop:
  movq    %rdx, (%rcx,%rax,8)     Set p[i] to q
  addq    $1, %rax                Increment i
  movq    %rax, -8(%rbp)          Store on stack
.L2:
  movq    -8(%rbp), %rax
  cmpq    %rdi, %rax
  jl      .L3
    . . .
Code for function exit
  leave                           Restore %rbp and %rsp
  ret                             Return
```

```
movq %rbp, %rsp        Set stack pointer to beginning of frame
popq %rbp              Restore saved %rbp and set stack ptr
                         to end of caller's frame
```

%rbp作为帧指针的作用：

- 在整个函数的执行过程中，%rbp始终指向函数栈的顶端（在返回地址和保存被调用者保存寄存器的值的下方）
- 利用固定长度的局部变量相对于%rbp的偏移量来引用它们
- leave指令释放整个栈帧

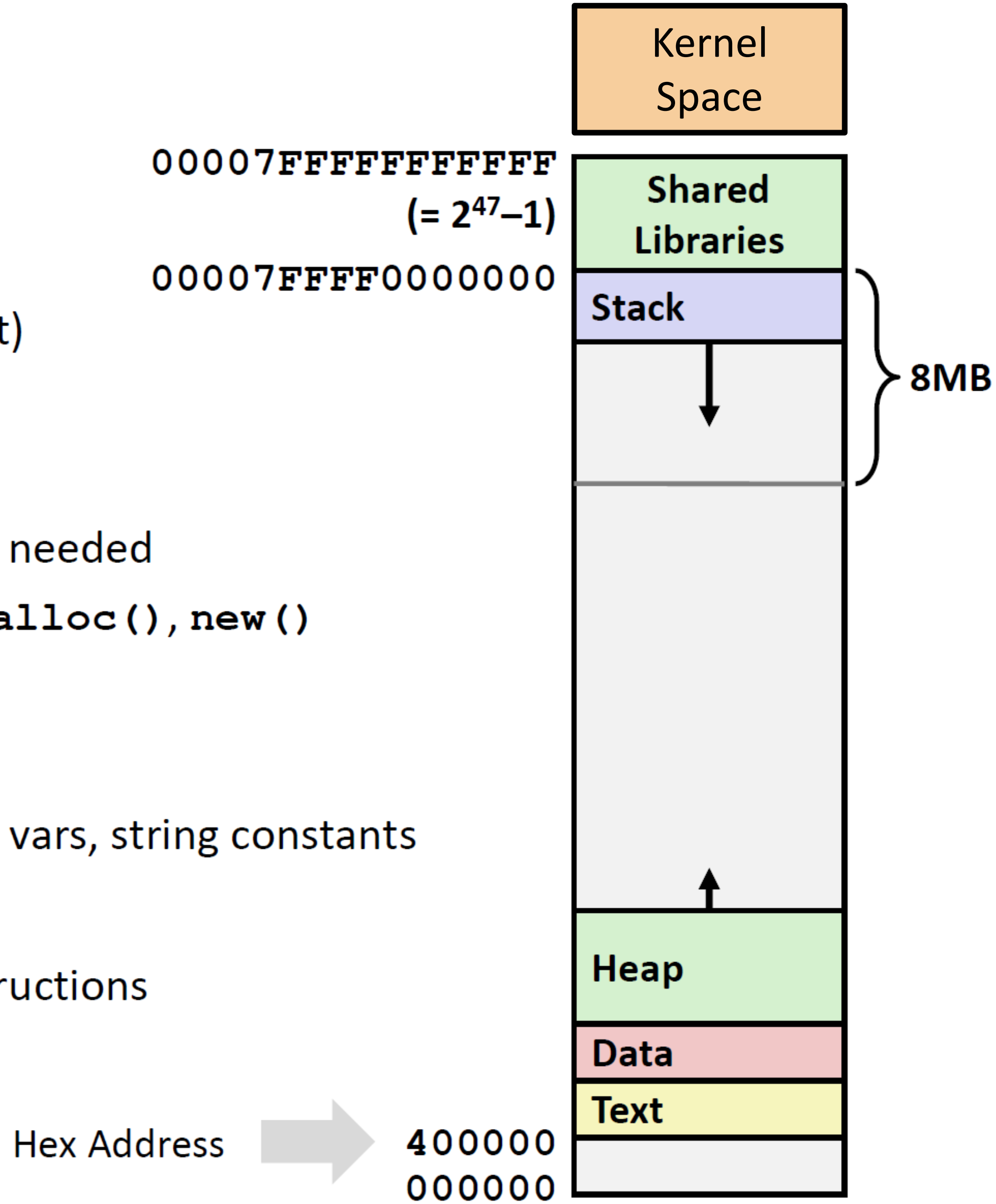
如何防御堆栈溢出攻击

- 用fgets规定输入字符串的大小。gets很不安全！
- 给内存增加可执行权限标记，禁止执行栈上的代码
- 栈偏移量随机化，无法事先确定数据地址
- 堆栈金丝雀：`%fs:0x28`

（了解即可）

Memory Layout

- **Stack**
  - Runtime stack (8MB limit)
  - E. g., local variables
- **Heap**
  - Dynamically allocated as needed
  - When call **`malloc(), calloc(), new()`**
- **Data**
  - Statically allocated data
  - E.g., global vars, **`static`** vars, string constants
- **Text / Shared Libraries**
  - Executable machine instructions
  - Read-only



Any Questions?