汇编语言程序设计

X86-32汇编编程 (补充)

目录

汇编示例程序-2

- 共享库文件
- 内存管理功能

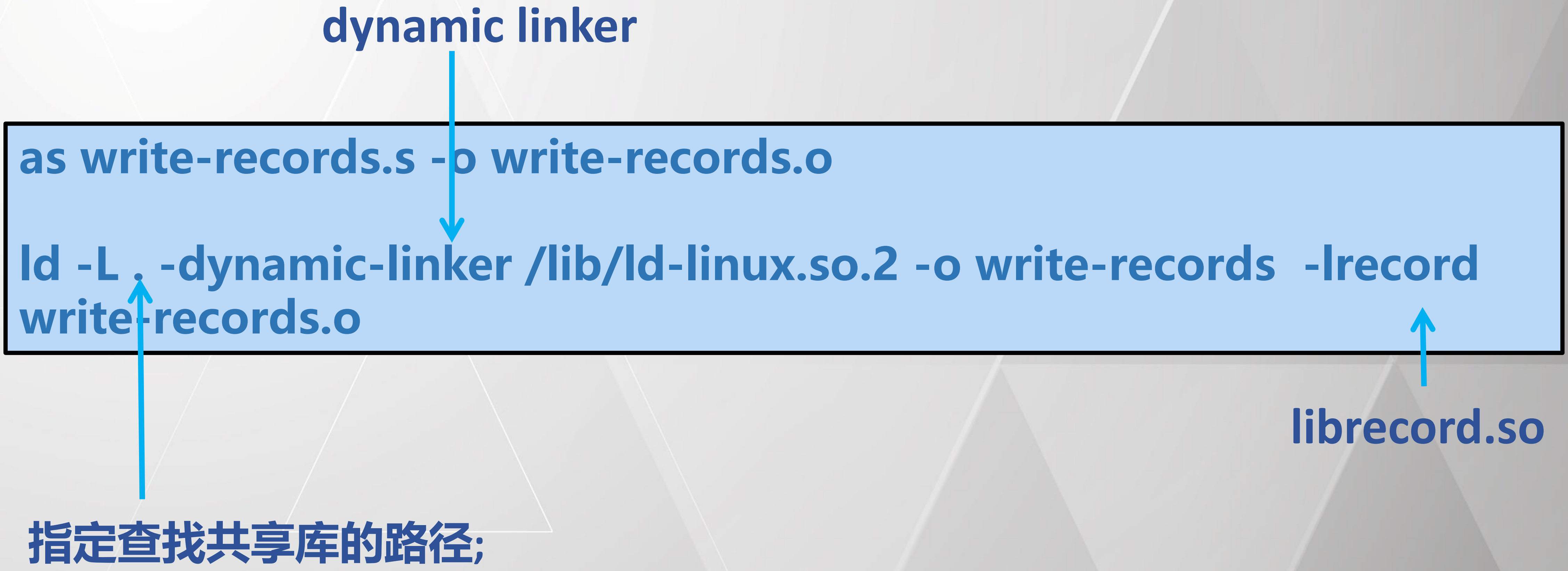
如何制作共享库文件

- 以“文件处理示例2——数据记录处理”为例

```
as write-record.s -o write-record.o
as read-record.s -o read-record.o
```

```
ld -shared  write-record.o read-record.o -o librecord.so
```

如何使用共享库文件



```
as write-records.s -o write-records.o

ld -L . -dynamic-linker /lib/ld-linux.so.2 -o write-records  -lrecord write-records.o
```

dynamic linker

librecord.so

指定查找共享库的路径;

运行时的注意事项

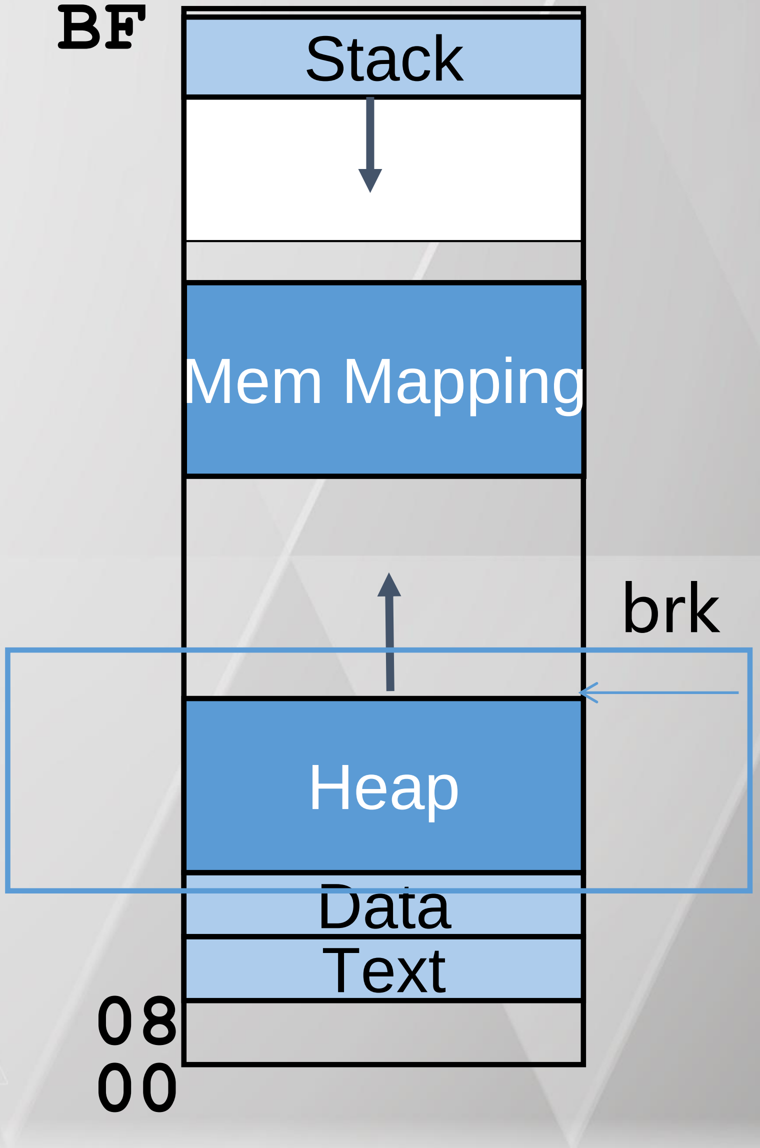
- **为定位共享库文件，dynamic linker默认在下列目录中依次搜索**
  - 环境变量LD_LIBRARY_PATH保存的路径中查找
  - 在文件/etc/ld.so.conf中列出的目录中搜索
  - 最后到默认的系统库文件目录中查找：先是/usr/lib ，然后是/lib等。

内存管理功能

堆（Heap）

- 程序动态分配的内存（C语言中malloc、free函数就是操作这一块区域）
- 当前堆的地址上限称为*system break*
  - 通过系统调用*brk(0)*可以获得
  - 或者是lib_c函数*sbrk(0)*
- *system break*可以按需调整
  - brk / sbrk

注意：直接访问超过(包括等于)*system break*的内存区域会引起“segmentation fault”



系统调用：brk

- **Call No.45 (%eax).**
  - Input
    - %ebx :欲设置的新的*break.*
  - Output
    - %eax: 成功，则返回新的*system break* ；否则返回当前的*system break* 。

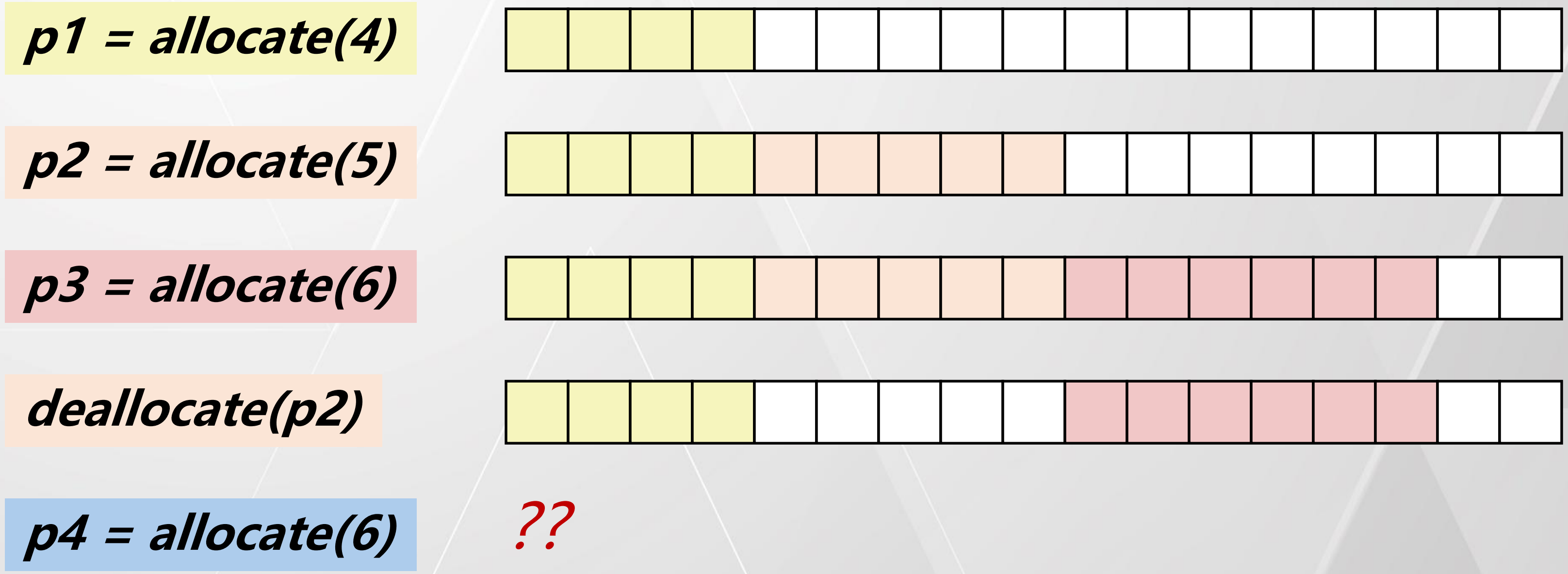
注意：失败时，与lib_c库里的同名调用的返回值不一样。

- **堆提供了一个用户进程可用的连续的内存空间，其上限可动态调整，但是系统调用本身无法提供灵活高效的内存分配/释放等管理功能。**

- **因此需要实现专门的内存管理模块，主要提供两类接口：**
  - 分配（输入：所需的内存大小；输出：分配的内存地址）
    - 可用的内存块如何跟踪与定位？
    - 如何选择合适的内存块进行分配？
    - 碎片如何处理？
  - 释放（输入：欲释放的内存地址）
    - 如何知道该指针指向的内存块的大小？
    - 如何复用释放的内存块？

lib_c函数 malloc / free 等即完成此功能。

简单示例-1： External Fragmentation



p1 = allocate(4)

p2 = allocate(5)

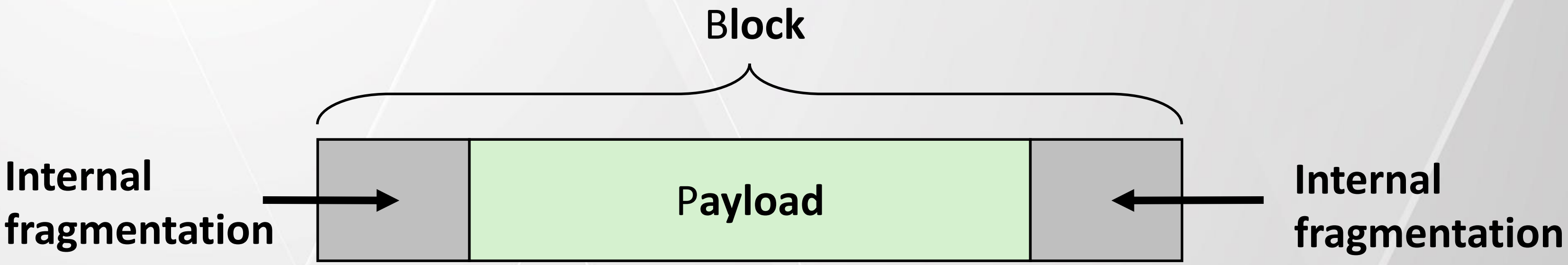
p3 = allocate(6)

deallocate(p2)

p4 = allocate(6)

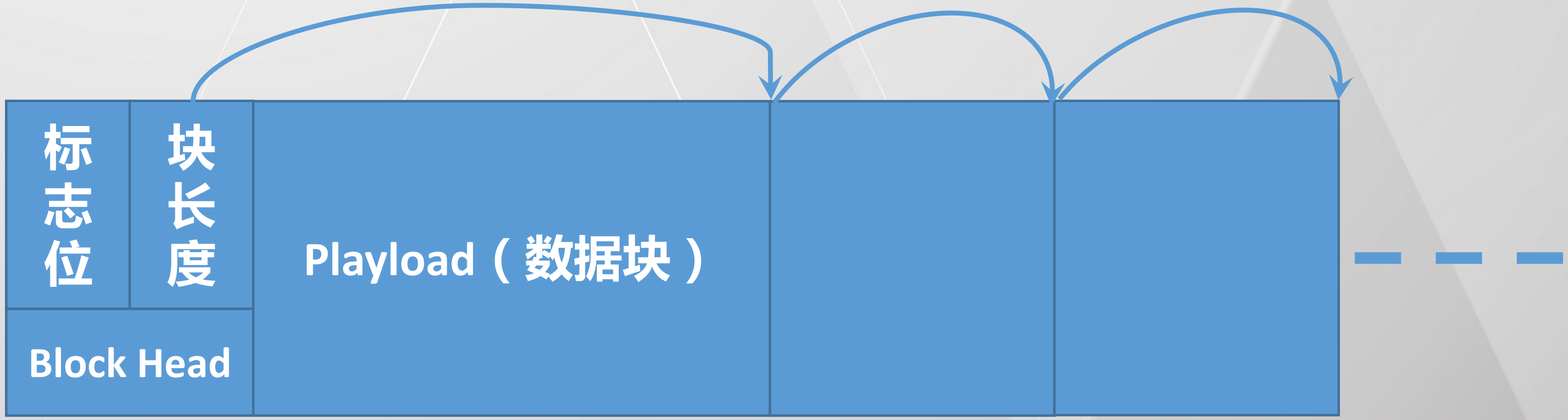
??

简单示例-2：Internal Fragmentation



内存管理功能示例，实现两个函数

- **allocate**
  - 从堆的起始地址开始，遍历每个内存块
  - 如果某一块标记为“可用”且其尺寸不小于所需求的大小，则将其标为“不可用”，并返回其数据块的地址，结束。
  - 否则继续遍历，如果没有合适的块，则通过系统调用增长system break。将新块标为“不可用”，并返回其数据块的地址，结束。
- **deallocate**
  - 直接置相应的标志位为“可用”。



```
section .data
 #This points to the beginning of the memory
heap_begin:
.long 0
#This points to one location past the memory we are managing
current_break:
.long 0

#size of space for memory region header
.equ HEADER_SIZE, 8
#Location of the "available" flag in the header
.equ HDR_AVAIL_OFFSET, 0
#Location of the size field in the header
.equ HDR_SIZE_OFFSET, 4

.equ UNAVAILABLE, 0
.equ AVAILABLE, 1
.equ SYS_BRK, 45                    #system call number for brk
.equ LINUX_SYSCALL, 0x80
```

```
# alloc.s
.section .text

.globl allocate_init
.type allocate_init,@function
allocate_init:
    pushl %ebp
    movl %esp, %ebp

    #If the brk system call is called with 0 in %ebx, it
    #returns the first invalid address
    movl $SYS_BRK, %eax
    movl $0, %ebx
    int $LINUX_SYSCALL
    movl %eax, current_break  #%eax now has the first invalid address
    movl %eax, heap_begin

    movl %ebp, %esp           #exit the function
    popl %ebp
    ret
```

```
  .globl allocate
  .type allocate, @function
  .equ ST_MEM_SIZE, 8        #stack position of the memory size to allocate
  allocate:
        pushl %ebp
        movl %esp, %ebp
        movl ST_MEM_SIZE(%ebp), %ecx        #%ecx will hold the size

        #we are looking for (which is the first and only parameter)
        movl heap_begin, %eax           #%eax will hold the search location
        movl current_break, %ebx        #%ebx will hold the current break

loop_begin:                             #we iterate through memory regions
        cmpl %ebx, %eax                 #need more memory if these are equal
        je move_break

        #grab the size of this memory
        movl HDR_SIZE_OFFSET(%eax), %edx
```

```
    cmpl $UNAVAILABLE, HDR_AVAIL_OFFSET(%eax)
    je next_location              #If unavailable, go to the next
    cmpl %edx, %ecx               #If available, check the size
    jle allocate_here             #big enough, go to allocate_here

next_location:
    addl $HEADER_SIZE, %eax
    addl %edx, %eax               #The total size of the memory
    Jmp  loop_begin               #go look at the next location

allocate_here:
    #if we’ ve made it here, that means that the region header of the
    #region to allocate is in %eax, mark space as unavailable
    movl $UNAVAILABLE, HDR_AVAIL_OFFSET(%eax)
    addl $HEADER_SIZE, %eax             #move %eax to the usable memory

    movl %ebp, %esp
    popl %ebp
    ret
```

```
move_break:
addl $HEADER_SIZE, %ebx      #add space for the headers structure
addl %ecx, %ebx              #add space to the break for the data
                             #requested

pushl %eax                   #save needed registers
movl $SYS_BRK, %eax          #reset the break
int $LINUX_SYSCALL
popl %eax                    #no error check?

#set this memory as unavailable, since we’ re about to give it away
movl $UNAVAILABLE, HDR_AVAIL_OFFSET(%eax)
movl %ecx, HDR_SIZE_OFFSET(%eax) #set the size of the memory
addl $HEADER_SIZE, %eax          #move %eax to the actual start of
                                 #usable memory.
movl %ebx, current_break         #save the new break

movl %ebp, %esp
popl %ebp
ret
```

```
.globl deallocate
.type deallocate,@function
.equ ST_MEMORY_SEG, 4

deallocate:
        movl ST_MEMORY_SEG(%esp), %eax
        #get the pointer to the real beginning of the memory
        subl $HEADER_SIZE, %eax
        #mark it as available
        movl $AVAILABLE, HDR_AVAIL_OFFSET(%eax)

        ret
```

该内存管理效率很低

- 分配时遍历所有的内存块，复杂度为O(n)
- 每次调用brk只是分配所需的内存大小，会导致系统调用次数过多
- Internal Fragmentation
  - 释放时应将相邻的available块合并。
- 如何改进（作为作业）？

如何使用

```
# read-records.s
...
.section .bss
   .lcomm record_buffer, RECORD_SIZE
```

vs.

```
.data
record_buffer_ptr:
         .long 0
```

```
pushl $record_buffer
call read_record
...
pushl $RECORD_FIRSTNAME + record_buffer
```

```
?
```