

Unix系统高级编程

进程映像和虚拟内存

Unit05

内存管理与进程映像



内存管理



内存管理

- 从底层硬件到上层应用，各层都提供了各自的内存管理接口

知识讲解



进程映像



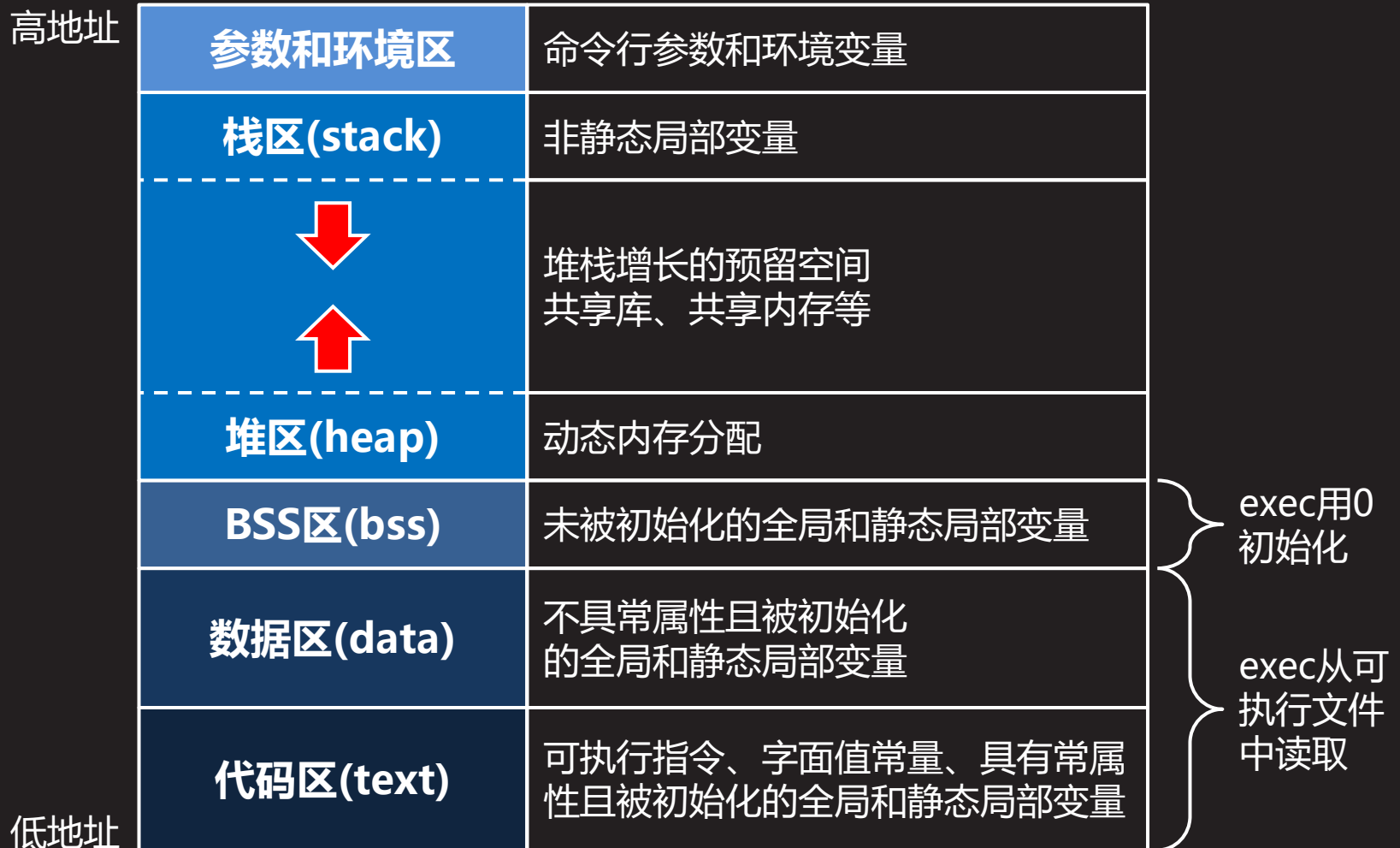
进程的内存布局

- 程序是保存在磁盘上的可执行文件
- 运行程序时，需要将可执行文件加载到内存，形成进程
- 一个程序(文件)可以同时存在多个进程(内存)
- 进程在内存空间中的布局形成进程映像，从低地址到高地址依次为
 - 代码区(text)
 - 数据区(data)
 - BSS区(bss)
 - 堆区(heap)
 - 栈区(stack)
 - 参数和环境区



进程的内存布局 (续1)

- 基于Intel架构的Linux系统中的进程内存布局



进程的内存布局

【参见：process.c】

- 进程的内存布局



进程映像文件

- 在/proc目录下可以看到很多以进程标识(PID)命名的子目录，这些子目录中存放的并不是真正的磁盘文件，而是操作系统内核为每个目前处于运行状态的进程，在内存中构建的审计数据结构，把它们挂载到文件系统中，只是为了方便系统管理人员实时监控系统的运行情况
 - 如/proc/2614目录下保存的就是2614进程的审计信息
- 在有关进程审计的诸多文件中，名为maps的文件反映了特定进程的内存布局，谓之进程映像文件

```
b738f000-b7533000 r-xp 00000000 08:01 983863 /lib/i386-  
linux-gnu/libc-2.15.so  
b76ff000-b7700000 r-xp 00000000 00:00 0 [vdso]  
b89fd000-b8a1e000 rw-p 00000000 00:00 0 [heap]  
bf906000-bf927000 rw-p 00000000 00:00 0 [stack]
```



进程映像文件 (续1)

- 进程映像文件中的每一行均包括六列，从左至右依次为
 - 形如 “bf8ff000-bf920000” 的地址范围
 - 形如 “rwxp” 的访问权限，其中
 - r** : 可读
 - w** : 可写
 - x** : 可执行
 - s/p** : 共享/私有
 - 形如 “000f6000” 的库内偏移地址
 - 形如 “08:01” 的映射文件设备号(主设备号:次设备号)
 - 形如 “983863” 的映射文件i节点号
 - 形如 “/lib/i386-linux-gnu/libc-2.15.so” 的库路径



进程映像文件 (续2)

- 进程映像文件中的每一行均包括六列，从左至右依次为
 - 上述最后一列也可能是[heap]、[stack]或者[vdso]
 - [heap] : 堆
 - [stack] : 栈
 - [vdso] : 虚拟动态共享库



size命令

- 通过size命令可以观察特定可执行程序的代码区(text)、数据区(data)和BSS区(bss)的大小，以字节为单位

```
$ size a.out
```

text	data	bss	dec	hex	filename
2628	268	28	2924	b0c	a.out

- Linux系统下的可执行程序、静态库和共享库的文件格式均采用ELF(Executable and Linkable Format, 可执行可链接文件格式)标准，该文件格式同时也是SVR4和Solaris等Unix操作系统的默认目标文件格式
- 符合ELF标准的可执行程序文件只包含代码区和数据区的内容，在进程加载阶段被exec函数读入进程的内存空间



虚拟内存



内存空间与地址空间



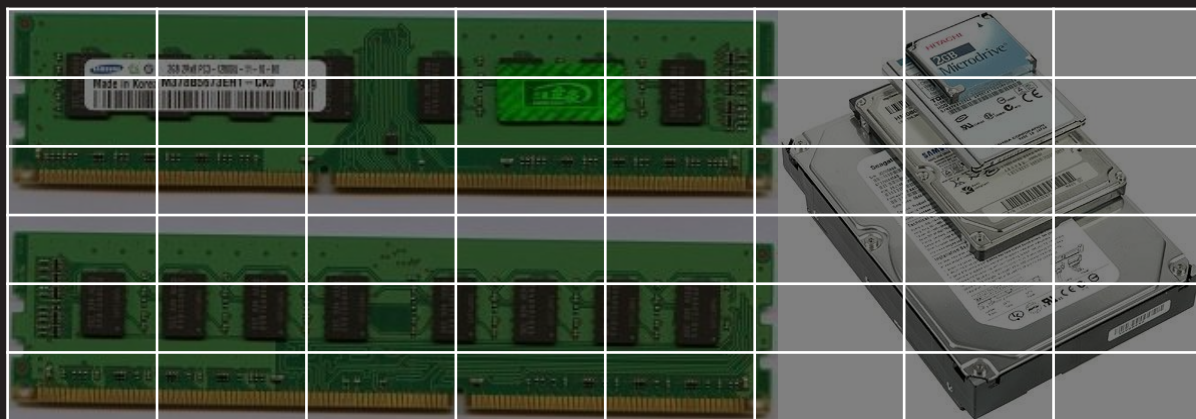
内存空间与地址空间

- 在系统中运行的每个进程，都拥有各自互独立的，4G字节大小的地址空间，谓之虚拟内存
- 用户程序中使用的都是地址空间中的虚拟内存，但真正存储代码和数据的却是永远无法被直接访问的物理内存
- 广义的物理内存不仅包括半导体内存，即通常所说的内存条，也包括磁盘上的交换分区(swap分区)或换页文件
- 当半导体内存不够用时，可以把一些长期闲置的代码和数据从半导体内存缓存到交换分区或换页文件中，这叫页面换出，一旦需要使用那些代码和数据，再把它们从交换分区或换页文件恢复到半导体内存中，这叫页面换入。因此，系统中的虚拟内存可以比半导体内存大很多



内存空间与地址空间（续1）

- 虚拟内存到物理内存，或者说地址空间到内存空间的映射关系，由操作系统内核通过内存映射表动态维护
- 虚拟内存技术一方面保护了系统的安全，把用户进程与内核进程，用户进程与用户进程，在对内存的访问上完全隔离开，另一方面又借助交换分区或换页文件，允许应用程序以一种完全透明的方式，使用比实际半导体内存更大的存储空间

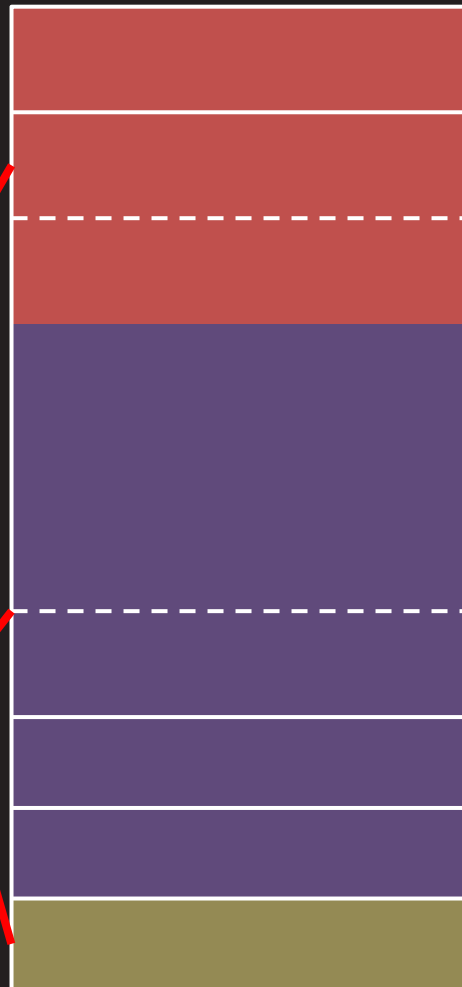
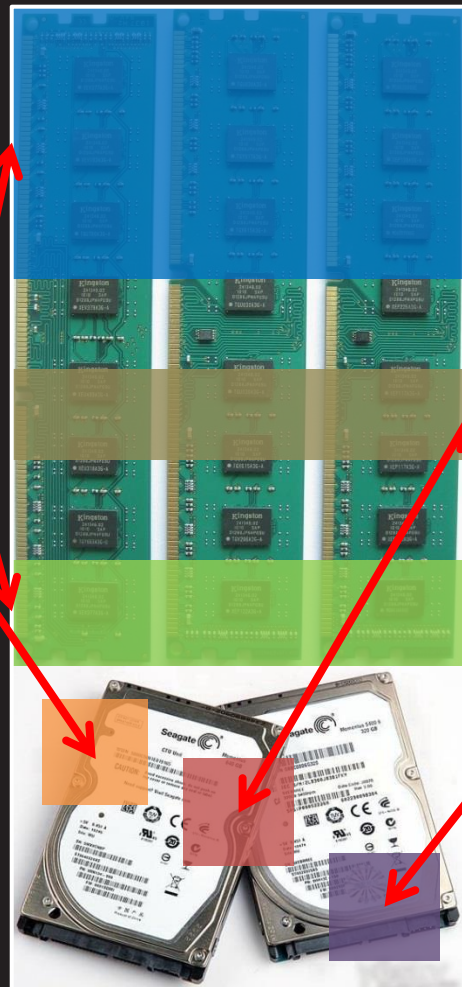
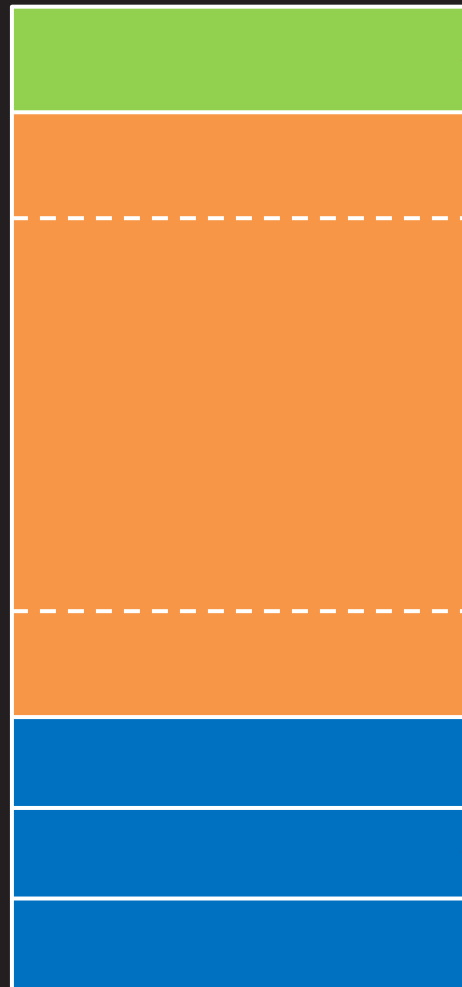


内存空间与地址空间 (续2)

进程A的地址空间
(虚拟内存)

系统的内存空间
(物理内存)

进程B的地址空间
(虚拟内存)



知识讲解



用户空间与内核空间



用户空间与内核空间

- 4G字节大小的进程地址空间分成两部分
 - 0~3G-1为用户空间，存放用户程序的代码和数据
 如某局部变量的地址0xbfc7fba0=3,217,554,336，约3G
 - 3G~4G-1为内核空间，存放系统内核的代码和数据
- 用户空间的代码不能直接访问内核空间的代码和数据，但可以通过系统调用进入内核态，间接地与系统内核交互

知识讲解



内存壁垒与段错误



内存壁垒与段错误

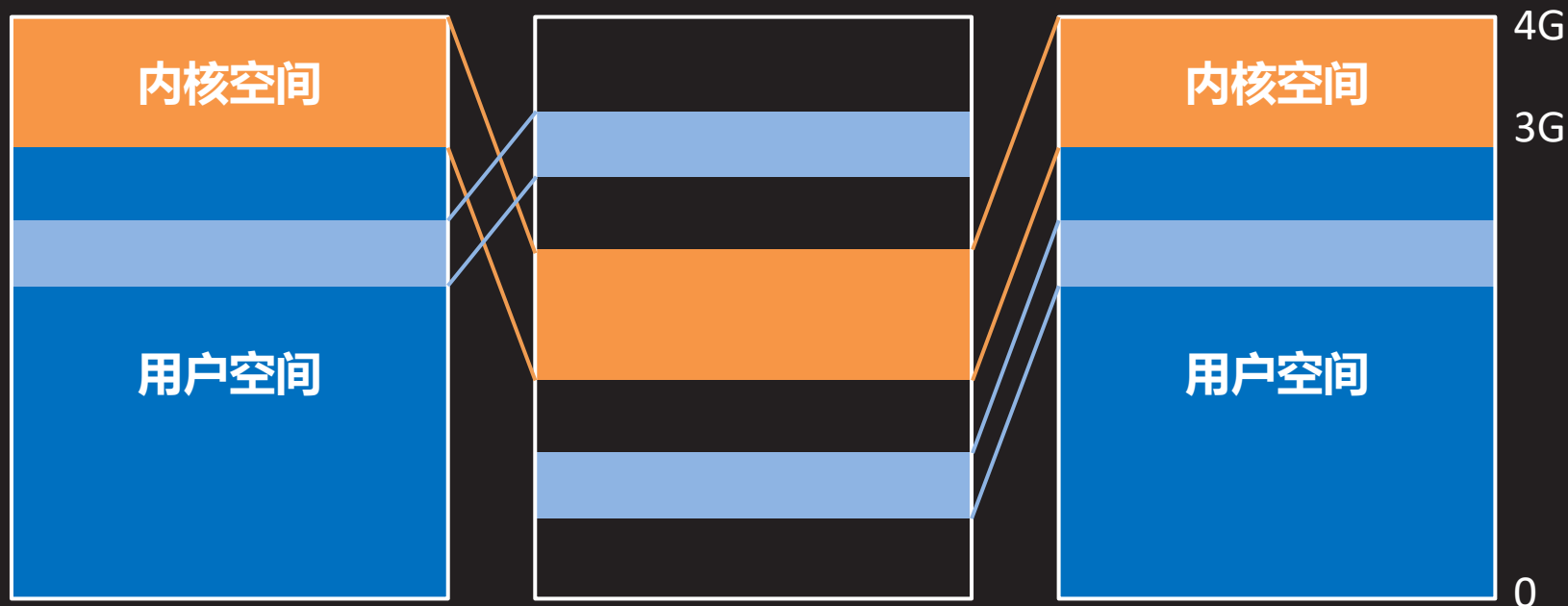
- 每个进程的用户空间都是 $0 \sim 3G-1$ ，但它们所对应的物理内存却是各自独立的，系统为每个进程的用户空间维护一张专属于该进程的内存映射表，记录虚拟内存到物理内存的对应关系，因此在不同进程之间交换虚拟内存地址是毫无意义的
- 所有进程的内核空间都是 $3G \sim 4G-1$ ，它们所对应的物理内存只有一份，系统为所有进程的内核空间维护一张内存映射表`init_mm.pgd`，记录虚拟内存到物理内存的对应关系，因此不同进程通过系统调用所访问的内核代码和数据是同一份



内存壁垒与段错误 (续1)

- 用户空间的内存映射表会随着进程的切换而切换，内核空间的内存映射表则无需随着进程的切换而切换
- 一切对虚拟内存的越权访问，都将导致段错误
 - 试图访问没有映射到物理内存的虚拟内存
 - 试图以非法方式访问虚拟内存，如对只读内存做写操作等

知识讲解



内存壁垒

【参见：vm.c】

课堂
练习

- 内存壁垒

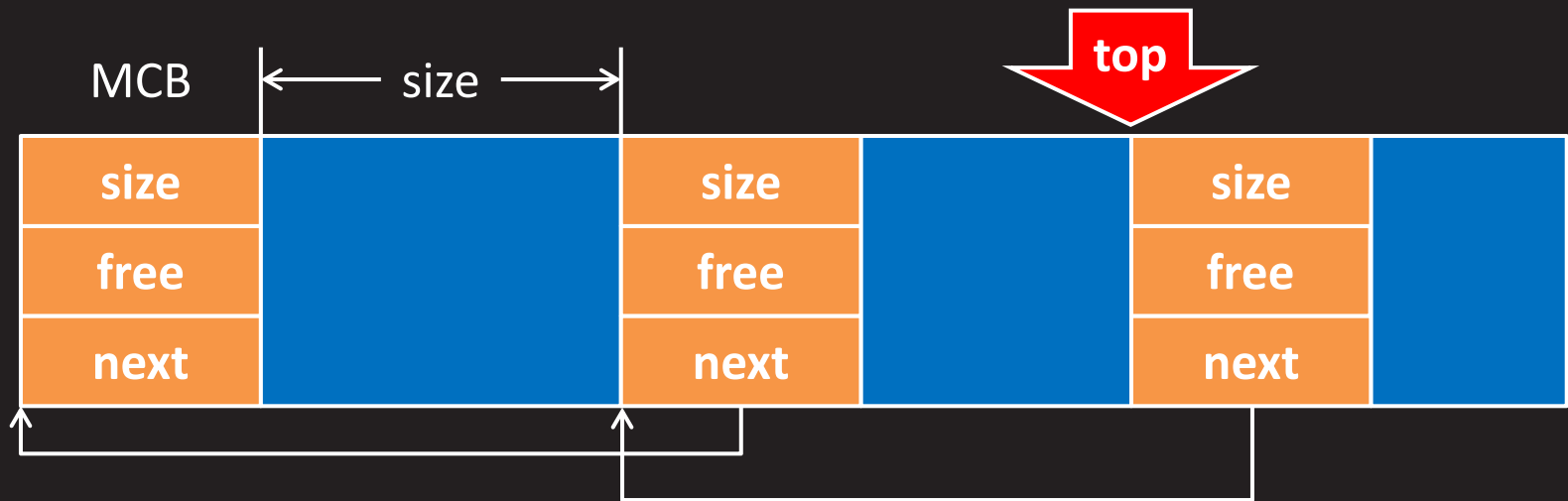


标准内存分配函数



标准内存分配函数

- 标准库提供的内存分配函数(malloc/calloc/realloc)在标准库内部维护一个线性链表，管理堆中动态分配的内存
- 标准内存分配函数在分配内存时会附加若干(通常是12个)字节，存放控制信息(MCB, 内存控制块)，该信息一旦被意外损坏，可能在后续操作中引发异常



标准内存分配函数 (续1)

- 虚拟内存到物理内存的映射以页(4K=4096字节)为单位

```
#include <unistd.h>

int getpagesize (void);
```

返回内存页的字节数

- 通过malloc函数首次分配内存，至少映射33页，即使通过free函数释放掉全部内存，最初的33页仍然保留

```
char* p = (char*)malloc (sizeof (char));
```



标准内存分配函数

【参见：page.c】

- 标准内存分配函数



总结和答疑

