Virtual Memory

李天驰游震邦

2021.12.5

Outline (第一部分)

- 地址空间
- 分页机制
  - 页表
  - 页表条目
- 优化页表
  - 时间优化：TLB
  - 空间优化：多级页表

Outline（第二部分）

- 页故障
- 内存映射
  - Fork
  - Execve
  - Mmap/munmap

Outline（第三部分）

- 动态内存分配
  - malloc/free
  - 垃圾收集器

Outline（第四部分）

- 内存安全

第一部分

地址空间

- 非负整数地址的有序集合
- 为什么要使用地址空间
  - 简化链接和加载
  - 内存保护的工具
  - 共享内存
  - 简单的物理内存分配

- 地址空间是主存、交换空间、内存映射I/O端口、ROM的抽象（而不是书中所说的主存和磁盘I/O设备的抽象）

分页

- 地址空间是操作系统提供的接口
- 分页是地址空间的实现
  - 并不只有这一种实现方式
- 核心问题
  - 将虚拟地址翻译为物理地址：使用页表
  - 同样不是唯一的实现方式

页表

- 简单的页表：是页表条目（PTE）的数组
- PTE
  - PTE为unsigned long
  - 考虑64位地址空间，页大小为2^{12}B。在现在的Intel/AMD架构中，地址空间大小为2^{48}B，因此只有36位用于地址翻译
  - PTE的其他位：
    - P：present位
    - 当P=1时，R/W, U/S, XD位的功能？
    - 当P=0时，可以被操作系统所用

页表优化

- 时间优化：TLB
  - 本质上是一个cache，而不是buffer
  - 在内存空间切换时，TLB需要被刷新
  - L1 TLB, i-TLB, d-TLB, L2 unified TLB

- 空间优化：多级页表

虚存翻译（多级页表）：2018年虚存大题

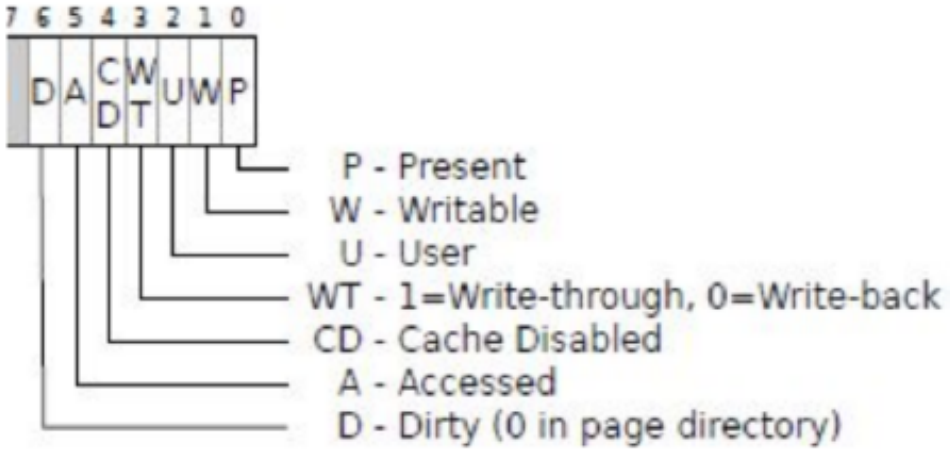
在进行地址翻译的过程中，操作系统需要借助页表(Page Table)的帮助。考虑一个32位的系统，页大小是4KB，页表项(Page Table Entry)大小是4字节(Byte)，如果不使用多级页表，常驻内存的页表一共需要_____页。

考虑下图已经显示的物理内存分配情况，在二级页表的情况下，已经显示的区域的页表需要占据_____页。

1024 5

VP0	已分配页
…	
VP1023	
VP1024	
…	
VP2047	
Gap	未分配页
1023 unallocated pages	
VP10239	已分配页
VP10240	
…	
VP11263	

IA32 体系采用小端法和二级页表。其中两级页表大小相同，页大小均为 4KB，结构也相同。TLB 采用直接映射。TLB 和页表每一项的后 7 位含义如下图所示。为简便起见，假设 TLB 和页表每一项的后 8~12 位都是 0 且不会被改变。注意后 7 位值为“27”则表示可读写。



当系统运行到某一时刻时，TLB 内容如下：

当系统运行到某一时刻时，TLB 内容如下：

索引	TLB 标记	内容	有效位
0	0x04013	0x3312D027	1
1	0x01000	0x24833020	0
2	0x005AE	0x00055004	1
3	0x00402	0x24AEE020	0
4	0x0AA00	0x0005505C	0
5	0x0000A	0x29DEE000	1
6	0x1AE82	0x00A23027	1
7	0x28DFC	0x00023000	0

一级页表的基地址为 0x0C23B00，物理内存中的部分内容如下：

地址	内容	地址	内容	地址	内容	地址	内容
00023000	E0	00023001	BE	00023002	EF	00023003	BE
00023120	83	00023121	C8	00023122	FD	00023123	12
00023200	23	00023201	FD	00023202	BC	00023203	DE
00023320	33	00023321	29	00023322	E5	00023323	D2
0005545C	97	0005545D	C2	0005545E	7B	0005545F	45
00055464	97	00055465	D2	00055466	7B	00055467	45
0C23B020	27	0C23B021	EB	0C23B022	AE	0C23B023	24
0C23B040	27	0C23B041	40	0C23B042	DE	0C23B043	29
0C23B080	05	0C23B081	5D	0C23B082	05	0C23B083	00
2314D200	23	2314D201	12	2314D202	DC	2314D203	0F
2314D220	A9	2314D221	45	2314D222	13	2314D223	D2
29DE404C	27	29DE404D	42	29DE404E	BA	29DE404F	00
29DE4400	D0	29DE4401	5C	29DE4402	B4	29DE4403	2A

此刻，系统先后试图对两个已经缓存在 cache 中的内存地址进行写操作，请分析**完成写之后**系统的状态（写的地址和上面的内存地址无交集），完成下面的填空。若不需要某次访问或者缺少所需信息，请填“\”。

第一次向地址 0xD7416560 写入内容，TLB 索引为：__6__，完成写之后该项 TLB 内容为：__0x00A23067__，

二级页表页表项地址为：______，物理地址为：__0x00A23560__。

第二次向地址 0x0401369B 写入内容，

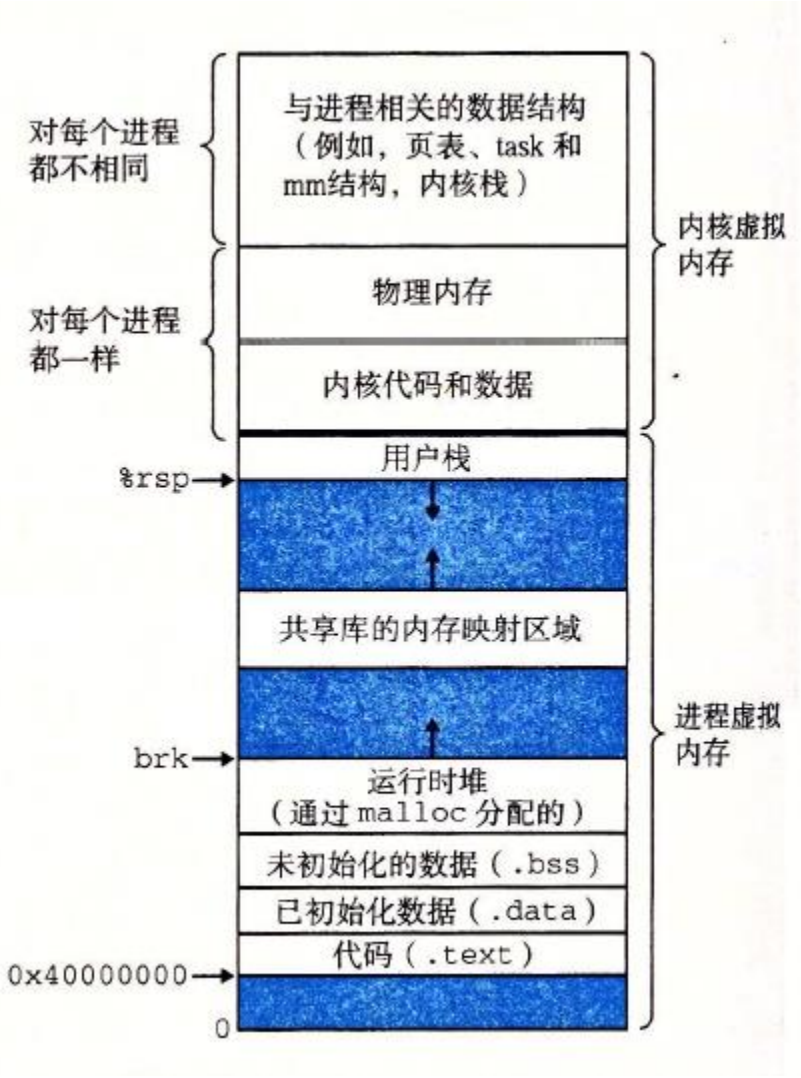
TLB 索引为：__3__，完成写之后该项 TLB 内容为：__0x00BA4067__

二级页表页表项地址为：__0x29DE404C__，物理地址为：__0x00BA469B__。

第二部分

Linux地址空间

- 物理内存区域可能只是整个物理内存空间的一部分，因为虚拟地址空间是48位，而物理地址空间是52位的
- .bss不是未初始化的数据，而是初始化成0的数据



虚拟内存区域

- Linux将虚拟内存组织成一些区域（也叫**段**）的集合
- 但是，硬件是不用段来管理的，这只是操作系统管理的一个方式
- 地址空间可以分页管理，也可以分段管理。但是，分段管理和这里的虚拟内存区域是不一样的，分段管理需要硬件和操作系统共同实现。

内存管理的数据结构

- 从程序员的角度来说，有两个重要的数据结构
  - 页表
  - vm_area_struct

页故障（page fault）

- 页故障（page fault）不仅仅是指物理页面的缺失，同时也包含权限错误
- 以下文本并不是一般保护故障（general protection fault），而是页故障（CSAPP 8.1.3）

一般保护故障。许多原因都会导致不为人知的一般保护故障(异常 13)，通常是因为一个程序引用了一个未定义的虚拟内存区域，或者因为程序试图写一个只读的文本段。Linux 不会尝试恢复这类故障。Linux shell 通常会把这种一般保护故障报告为“段故障(Segmentation fault)”。

页故障

- 页故障是定义在特权架构中的一种异常，以下是Intel定义的页故障

Interrupt 14—Page-Fault Exception (#PF)

Exception Class Fault.

Description

Indicates that, with paging enabled (the PG flag in the CR0 register is set), the processor detected one of the following conditions while using the page-translation mechanism to translate a linear address to a physical address:

- The P (present) flag in a page-directory or page-table entry needed for the address translation is clear, indicating that a page table or the page containing the operand is not present in physical memory.
- The procedure does not have sufficient privilege to access the indicated page (that is, a procedure running in user mode attempts to access a supervisor-mode page). If the SMAP flag is set in CR4, a page fault may also be triggered by code running in supervisor mode that tries to access data at a user-mode address. If the PKE flag is set in CR4, the PKRU register may cause page faults on data accesses to user-mode addresses with certain protection keys.
- Code running in user mode attempts to write to a read-only page. If the WP flag is set in CR0, the page fault will also be triggered by code running in supervisor mode that tries to write to a read-only page.
- An instruction fetch to a linear address that translates to a physical address in a memory page with the execute-disable bit set (for information about the execute-disable bit, see Chapter 4, “Paging”). If the SMEP flag is set in CR4, a page fault will also be triggered by code running in supervisor mode that tries to fetch an instruction from a user-mode address.

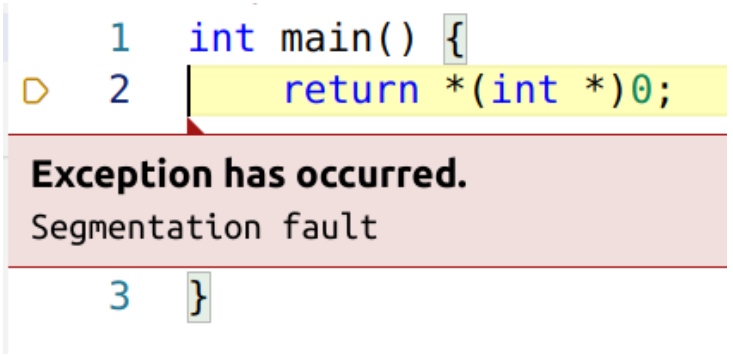
页故障的种类（basic）

- 地址无效：如NULL (0)
- 权限错误
  - R/W, XD, U/S
- 页面不在内存中

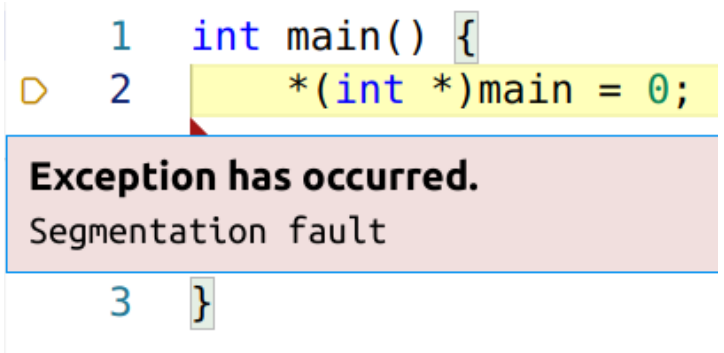
- 硬件需要检查PTE的哪些位
  - 先检查P位
  - 再检查R/W, XD, U/S

段故障（segmentation fault）

- 是一个信号，而不是异常
- 有一些页故障会引发这个信号
  - 地址无效



  - 权限错误



内存映射

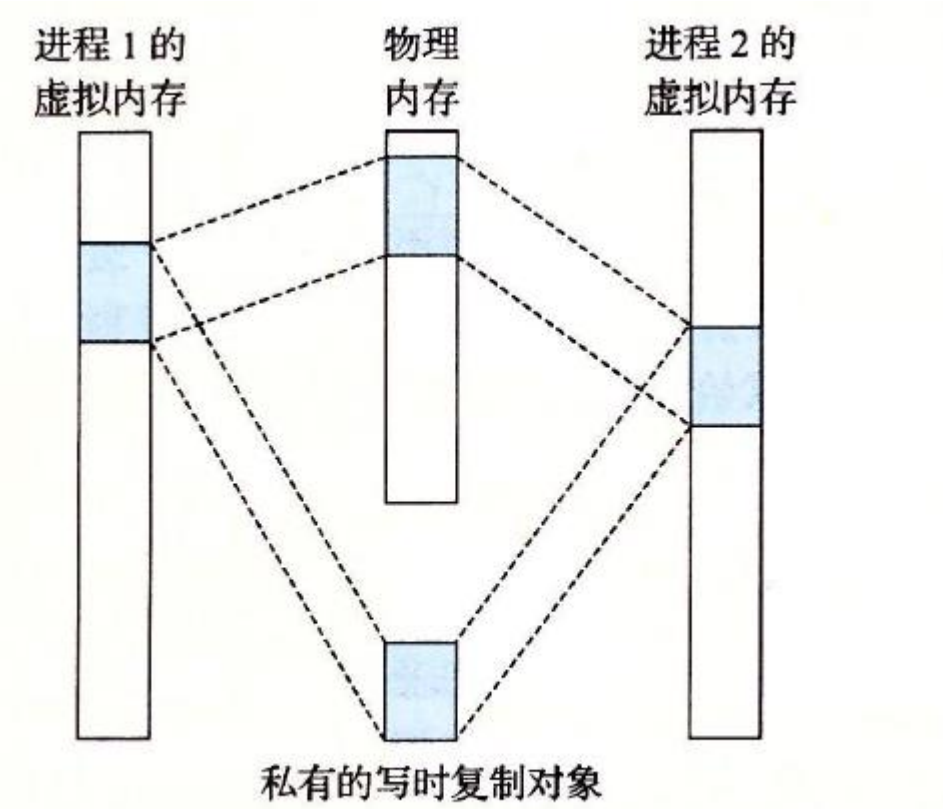
- 普通文件（目录文件不能被mmap映射）

```c
#include "csapp.h"

int main() {
    int dir_fd = Open("./mmap.c", O_RDONLY, S_IRUSR);
    Mmap(NULL, 10, PROT_READ, MAP_SHARED, dir_fd, 0);
}
```

  - mmap error: No such device
- 匿名文件（全是二进制0）

共享对象



a）两个进程都映射了私有的写时复制对象之后

进程创建

- fork
  - COW(copy on write)，借助R/W位实现
- execve
  - lazy loading

12、进程 P1 通过 fork()函数产生一个子进程 P2。假设执行 fork()函数之前，进程 P1 占用了 53 个（用户态的）物理页，则 fork 函数之后，进程 P1 和进程 P2 共占用_____个（用户态的）物理页；假设执行 fork()函数之前进程 P1 中有一个可读写的物理页，则执行 fork()函数之后，进程 P1 对该物理页的页表项权限为_____。上述两个空格对应内容应该是（　　）

A．53，读写　B．53，只读　C．106，读写　D．106，只读

B

mmap

- https://man7.org/linux/man-pages/man2/mmap.2.html

第三部分

为什么要动态内存分配

- 三个原因
  - 最根本的是对象的生命周期，可能比它的创建者的生命周期要长
    - 在C语言中有三种类型的生命周期：自动（栈中的对象）、动态（堆中的对象）、静态（.data .bss）
  - 变长对象
    - 可以被放在变长栈帧中，但是不高效、不安全
  - 大的对象可能会造成栈溢出

堆管理

- 三种方式
  - 手动管理：C
  - 运行时管理（垃圾收集器）：JVM, Go, Python
  - 编译时管理：Rust

  - 最后一种在教材中没有提到

动态内存分配API

- malloc/calloc/realloc
  - calloc会初始化
  - C++ new会初始化
- free
- brk/sbrk

动态内存管理的目标

- 最大化吞吐率
- 最大化内存利用率
  - 碎片
    - 内部碎片
    - 外部碎片

动态内存管理实现

- 链表
  - 隐式链表
  - 显示链表
  - 分离链表

- 适配
  - 首次适配
  - 下次适配
  - 最佳适配
- 伙伴系统：在固定大小的物理内存分配中很实用

7. 下列与虚拟内存有关的说法中哪些是不对的？

A. 操作系统为每个进程提供一个独立的页表，用于将其虚拟地址空间映射到物理地址空间。

B. MMU 使用页表进行地址翻译时，虚拟地址的虚拟页面偏移与物理地址的物理页面偏移是相同的。

C. 若某个进程的工作集大小超出了物理内存的大小，则可能出现抖动现象。

D. 动态内存分配管理，采用双向链表组织空闲块，使得首次适配的分配与释放均是空闲块数量的线性时间。

答案：D 首次释放不一定是空闲块数量的线性时间

13. 动态管理器分配策略中，最适合“最佳适配算法”的空白区组织方式是：

A. 按大小递减顺序排列

B. 按大小递增顺序排列

C. 按地址由小到大排列

D. 按地址由大到小排列

答案：B。

说明：最佳适应算法的空白区是按小大递增顺利链接在一起。

17. 动态内存管理中，可能会造成空闲链表中，小空闲块，即“碎片”，比较集中的算法是（ ）

A. 首次适配算法　　B. 下次适配算法

C. 最佳适配算法　　D. 以上三种算法无明显区别

【答案】A

【说明】考察空闲链表的不同搜索分配策略的性质。

垃圾收集器

- 可达图（reachability graph）
- Mark & Sweep

- 为什么在C语言中工作得不好？

- 为什么不用引用计数？（有循环引用）

第四部分

内存安全

- https://en.wikipedia.org/wiki/Memory_safety
- 分配、赋值、访问、释放
- 如果任何一个步骤出现错误，或者他们的顺序有错，就会产生内存不安全的情况
- 建议结合其他编程语言讲解，例如
  - Java，基础语法与C++相似
  - Python，同学都会的语言
  - Go，语法简单，和C相似