Introduction to Computer Systems Recitation

——Virtual Memory – Systems

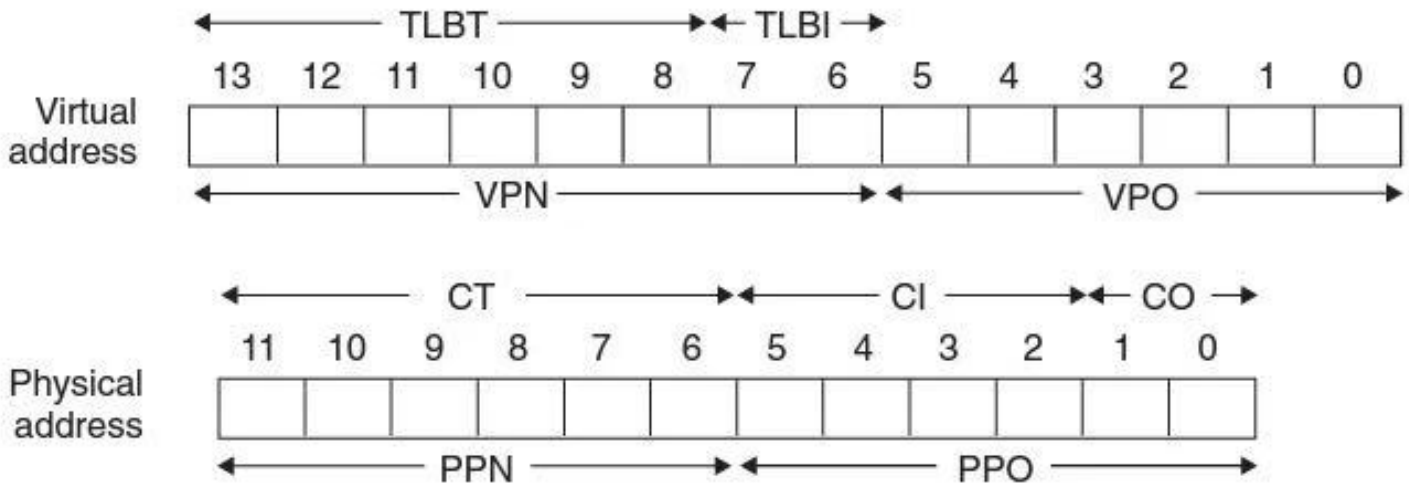
Guo Jiarui

1900012974
ntguojiarui@pku.edu.cn
Peking University

December 10, 2020

1. Simple memory system example
   - It should not have appeared here.
   - You can review this part after class.
2. Case Study: The Intel Core i7/Linux Memory System
   - Intel Core i7
   - Linux Memory System
3. Memory Mapping

What is address translating doing?



- Finding a "map" between virtual memory & phisical address.
- VPO keeps unchanged when translated to PPO.
- Important: find a map between VPN & PPN.
- TLB is needed.
- If the TLB missed, MMU fetched the PTE from main memory.

Get Phisical Address

Suppose $n = 14$, $m = 12$, $P = 64$, $E = 4$(for TLB).

	TLBT						TLBI							
	0x03						0x03							
Bit position	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VA = 0x03d4	0	0	0	0	1	1	1	1	0	1	0	1	0	0
	VPN								VPO					
	0x0f								0x14					

Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	03	–	0	09	0D	1	00	–	0	07	02	1
1	03	2D	1	02	–	0	04	–	0	0A	–	0
2	02	–	0	08	–	0	06	–	0	03	–	0
3	07	–	0	03	0D	1	0A	34	1	02	–	0

(a) TLB: 4 sets, 16 entries, 4-way set associative

	CT						CI				CO	
	0x0d						0x05				0x0	
Bit position	11	10	9	8	7	6	5	4	3	2	1	0
PA = 0x354	0	0	1	1	0	1	0	1	0	1	0	0
	PPN						PPO					
	0x0d						0x14					

Fetch Data from Cache

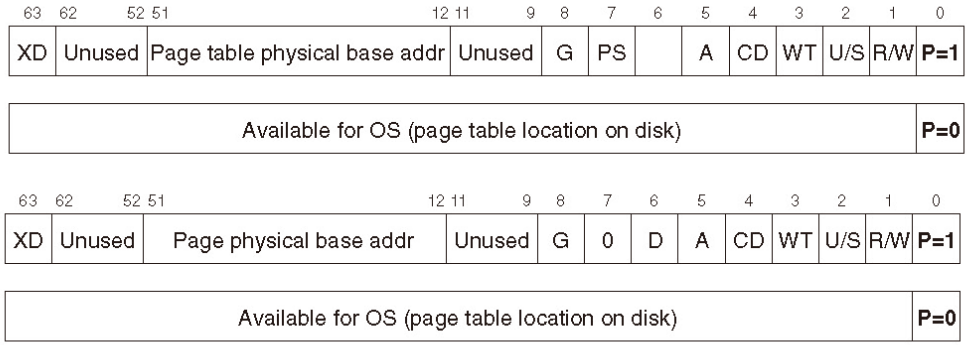
Suppose $E = 16$, $B = 4$, direct-mapped.
(Recall what we learned in chapter 6.)

Idx	Tag	Valid	Blk 0	Blk 1	Blk 2	Blk 3
0	19	1	99	11	23	11
1	15	0	—	—	—	—
2	1B	1	00	02	04	08
3	36	0	—	—	—	—
4	32	1	43	6D	8F	09
5	0D	1	36	72	F0	1D
6	31	0	—	—	—	—
7	16	1	11	C2	DF	03
8	24	1	3A	00	51	89
9	2D	0	—	—	—	—
A	2D	1	93	15	DA	3B
B	0B	0	—	—	—	—
C	12	0	—	—	—	—
D	16	1	04	96	34	15
E	13	1	83	77	1B	D3
F	14	0	—	—	—	—

	CT						CI				CO	
	0x0d						0x05				0x0	
Bit position	11	10	9	8	7	6	5	4	3	2	1	0
PA = 0x354	0	0	1	1	0	1	0	1	0	1	0	0
	PPN						PPO					
	0x0d						0x14					

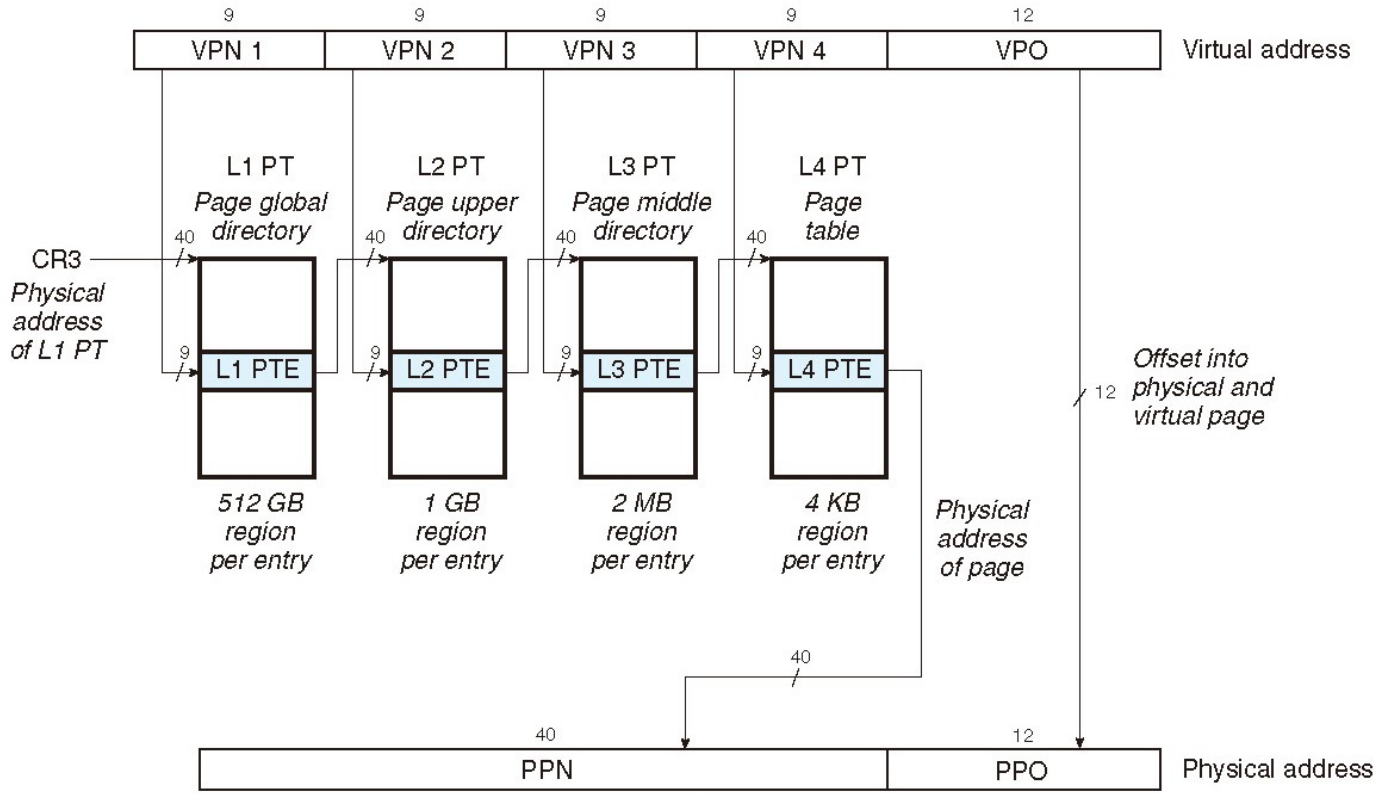
Format of Page Table Entries

物理页表要求4KB对齐

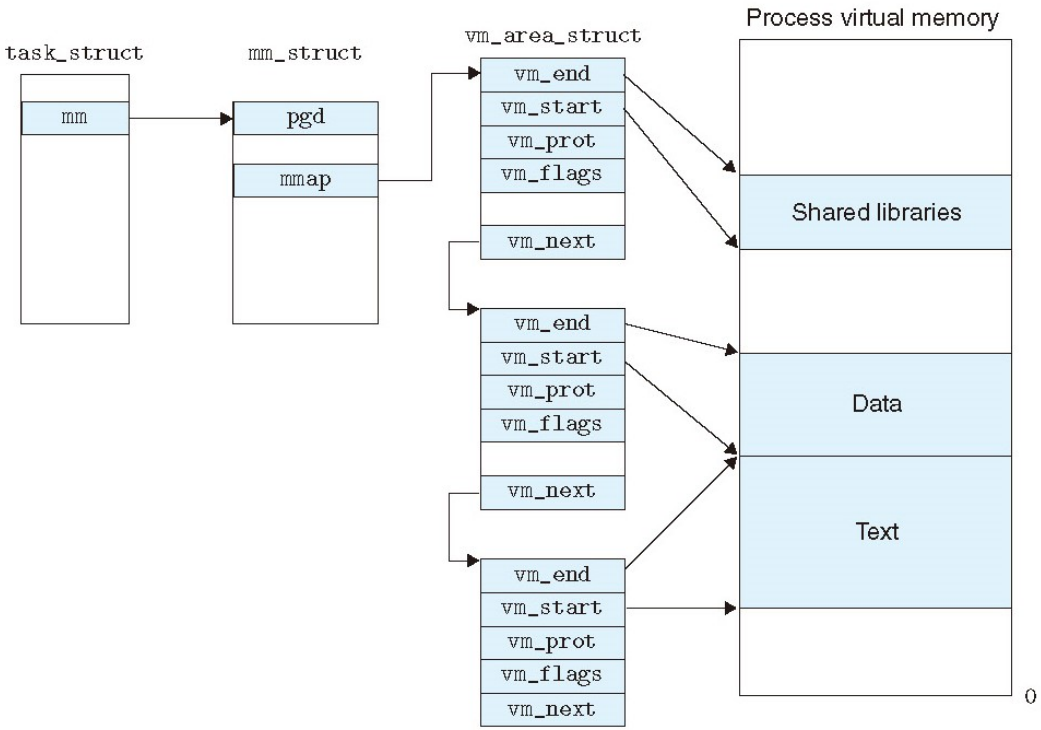


Field	Description
P	Child page present in physical memory (1) or not (0).
R/W	Read-only or read/write access permission for child page.
U/S	User or supervisor mode (kernel mode) access permission for child page.
WT	Write-through or write-back cache policy for the child page.
CD	Cache disabled or enabled.
A	Reference bit (set by MMU on reads and writes, cleared by software).
D	Dirty bit (set by MMU on writes, cleared by software).
G	Global page (don't evict from TLB on task switch).
Base addr	40 most significant bits of physical base address of child page.
XD	Disable or enable instruction fetches from the child page.
PS	Page size either 4 KB or 4 MB (defined for level 1 PTEs only).

Translate VA into PA



Area

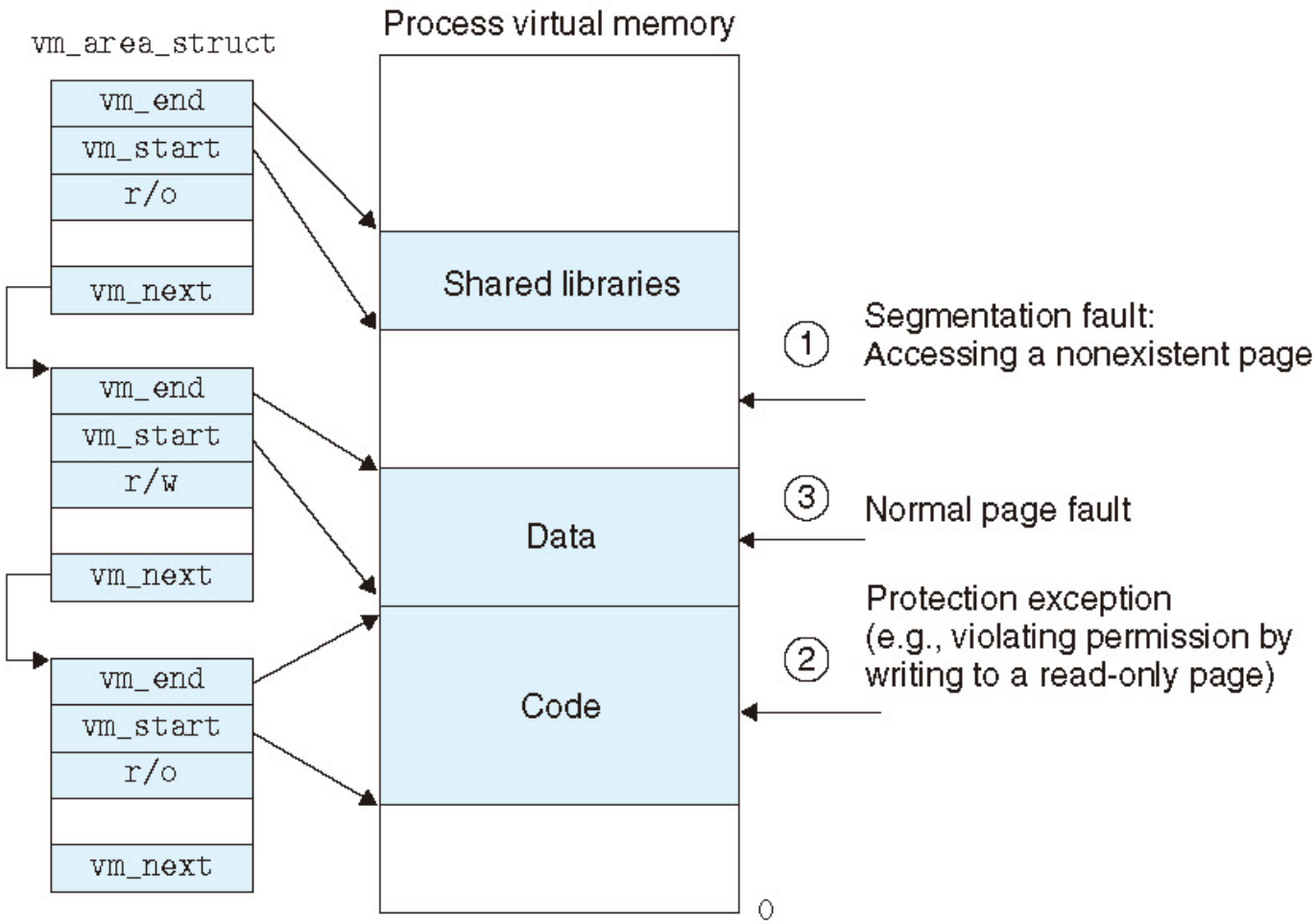


1. pgd: 指向第一级页表的基址
2. mmap: 指向 vm_area_structs 的一个链表
3. vm_prot: 这个区域包含的所有页的读写权限
4. vm_flags: 共享/私有

Page Fault Exception Handling

1. 虚拟地址 A 合法吗?
   - 假如不合法, 触发段错误(segmentational fault)
2. 内存访问合法吗?
   - 假如不合法, 触发保护异常(protection exception)
3. 合法地址, 合法访存
   - 选择一个牺牲页, 假如已经修改过的话将其交换出去
   - 换进来一个新页 & 更新页表
   - 重新执行当前指令

Page Fault Exception Handling



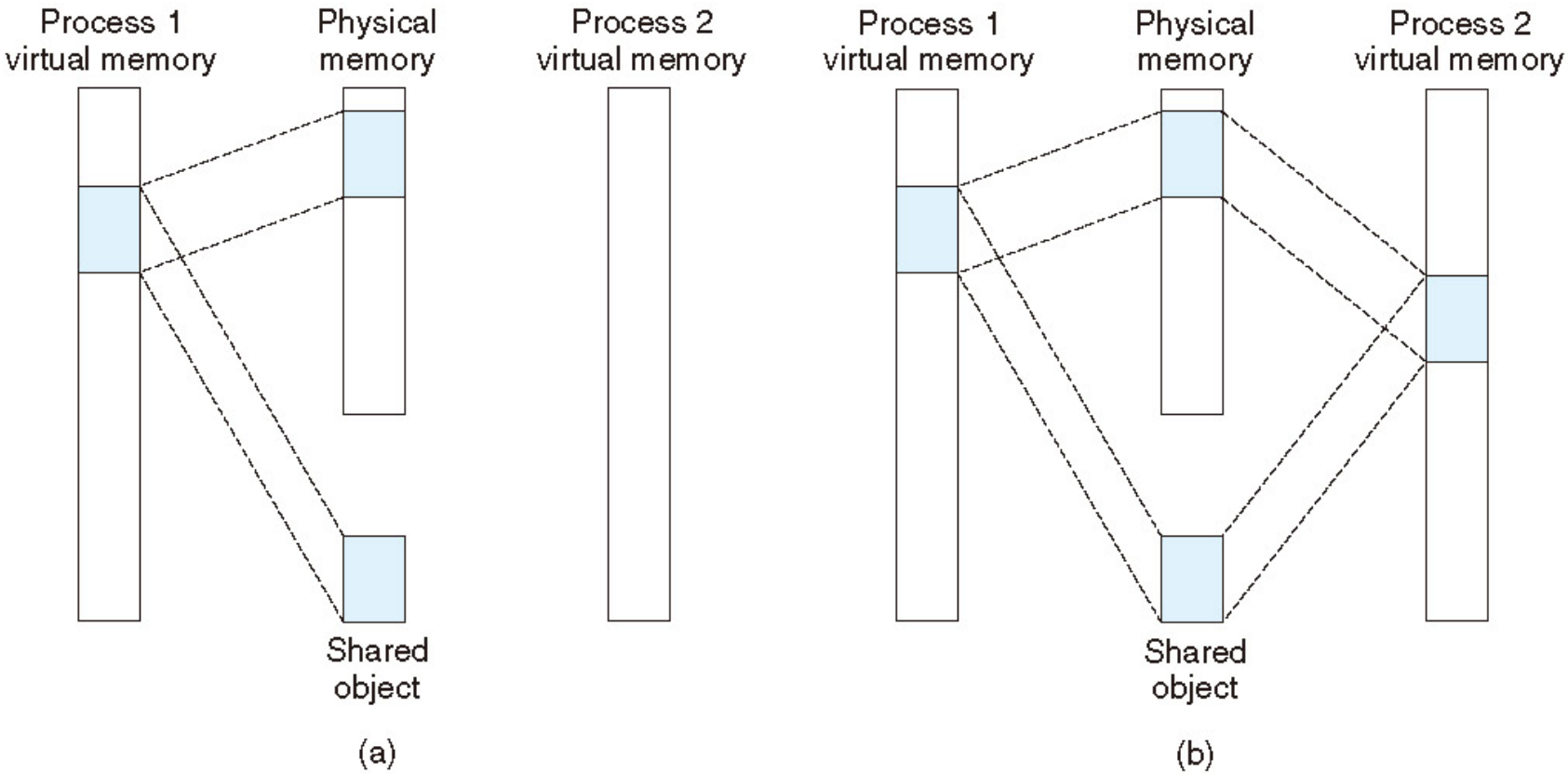
Mapping

区域可以映射到两种类型的对象:

- Linux文件系统中的普通文件
  - 普通磁盘文件中的连续部分
  - 没有实际交换进入物理内存直至CPU第一次引用
- 匿名文件
  - 由内核创建
  - 第一次引用会分配一个全0的物理页
  - 被写入后与其他文件一样

Shared Objects

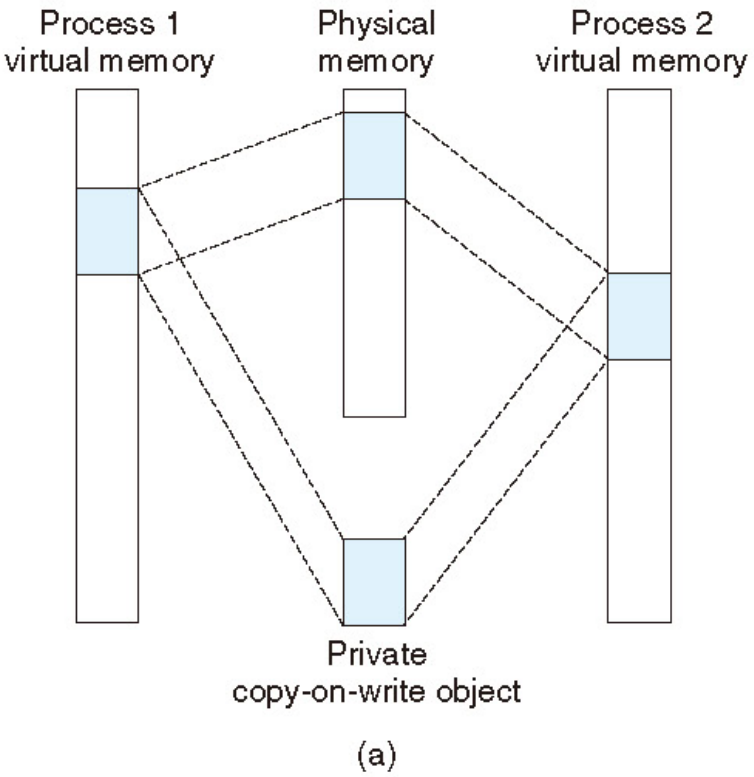
- 进程对这一区域的写操作，对于那些也把这个共享对象映射到它的虚拟内存的进程而言也是可见的
- 也会反映在磁盘上的原始对象上

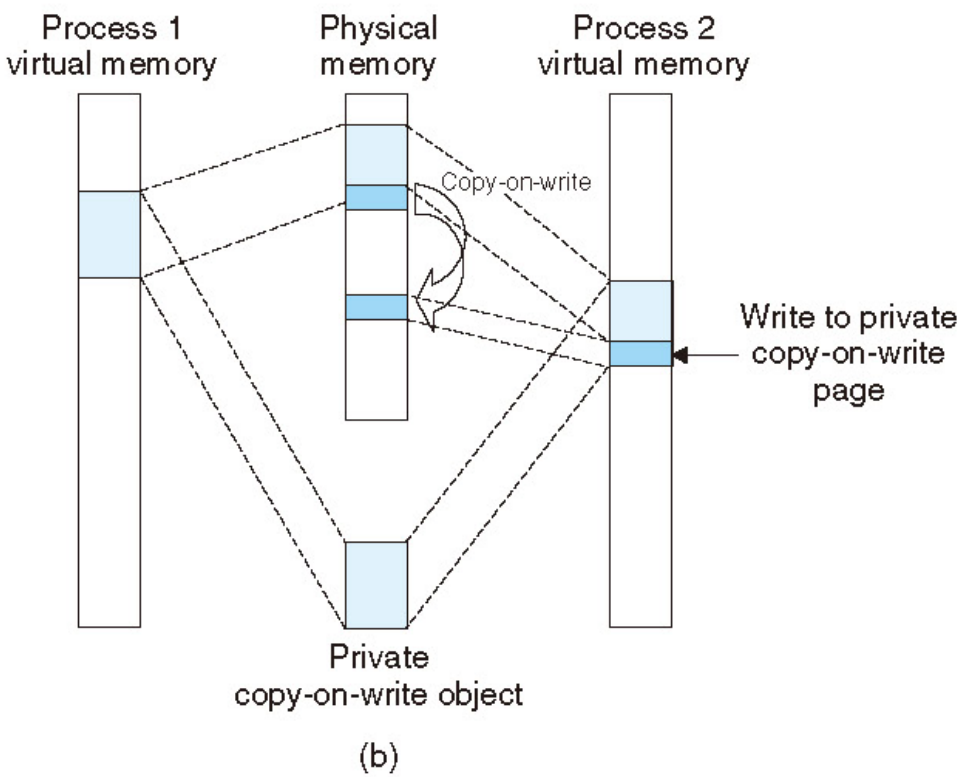


Private Copy-on-Write Object

- 一个映射到私有对象的区域所做的改变对其他进程不可见
- 不会反映到磁盘上的原始对象上
- 只读，写时复制
- 尝试写会导致保护异常
- 在物理内存中创建当前页的复制，更新页表
- 重新执行这一写操作

Private Copy-on-Write Object





The fork and execve Function

fork:
创建 mm_struct, 区域结构, 页表的副本
只读, 写时复制
父子进程中任意进程进行写操作时创建新页面

execve:
删除已存在的用户区域
映射私有区域
映射共享区域
设置PC
下次调度进程时从当前入口点开始执行

mmap Function

```
#include <unistd.h>
#include <sys/mman.h>
void *mmap(void *start, size_t length, int prot,
int flags, int fd, off_t offset);
```

返回指向当前映射区域的指针(假如创建成功)
每一区域的指针对于内核可见(可以通过指针找到区域，也可以通过区域找到对应的指针)

```
#include <unistd.h>
#include <sys/mman.h>
int munmap(void *start, size_t length);
```

删除成功返回0，删除失败返回1