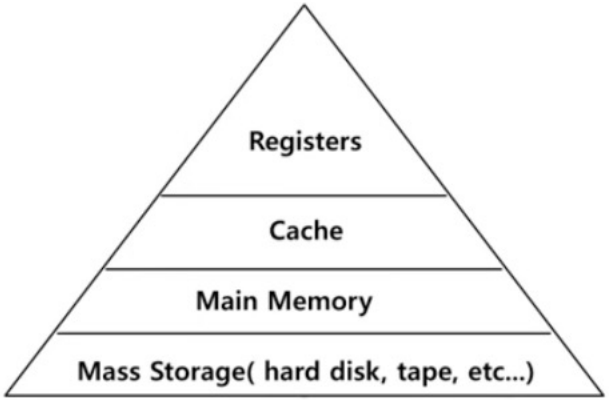
超越物理内存 - 交换机制
（Swapping Mechanism）

超越物理内存：机制

- 需要在**内存层次结构**中增加一个额外的层级
  - 操作系统需要一个地方来存放当前不活跃的地址空间部分
  - 在现代系统中，这一角色通常由**硬盘驱动器**来担任



Memory Hierarchy in modern systems

寄存器：最快、最小，位于CPU内部，用于临时存储。

缓存（Cache）：速度快，容量小，存储频繁访问的数据。

主内存（Main Memory）：即RAM，容量较大，速度适中。

大容量存储（Mass Storage）：如硬盘、磁带，容量大但速度慢。

为进程使用单一巨大地址空间

•一些早期系统需要程序员手动移入或移出内存中的代码或数据，在调用函数或访问数据前

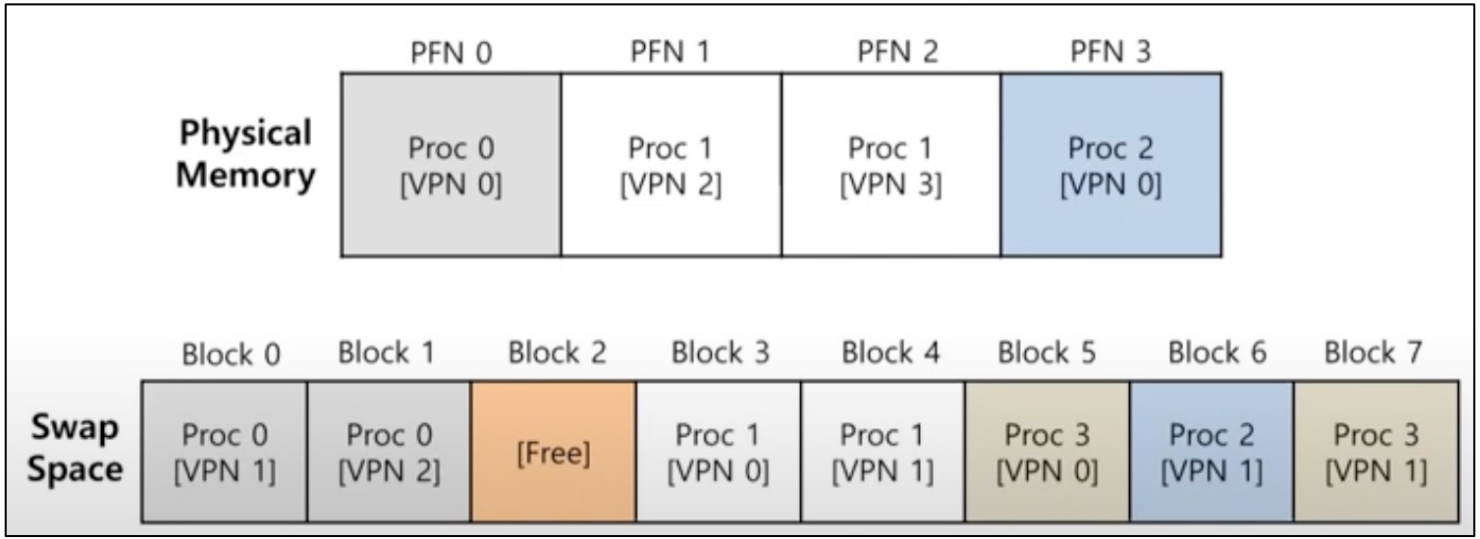
•对程序员来说更简单 —— 不需要手动管理或组织进程使用的内存。

•超越单个进程：

- 添加**交换空间（swap space）**使操作系统能够支持大量虚拟内存的幻觉，从而**支持多个并发运行的进程**，因为早期系统没有足够的主内存。

交换空间（swap space）

- 在磁盘上预留一些空间用于页面来回移动
- 操作系统需要记住交换空间，以**页面大小为单位**



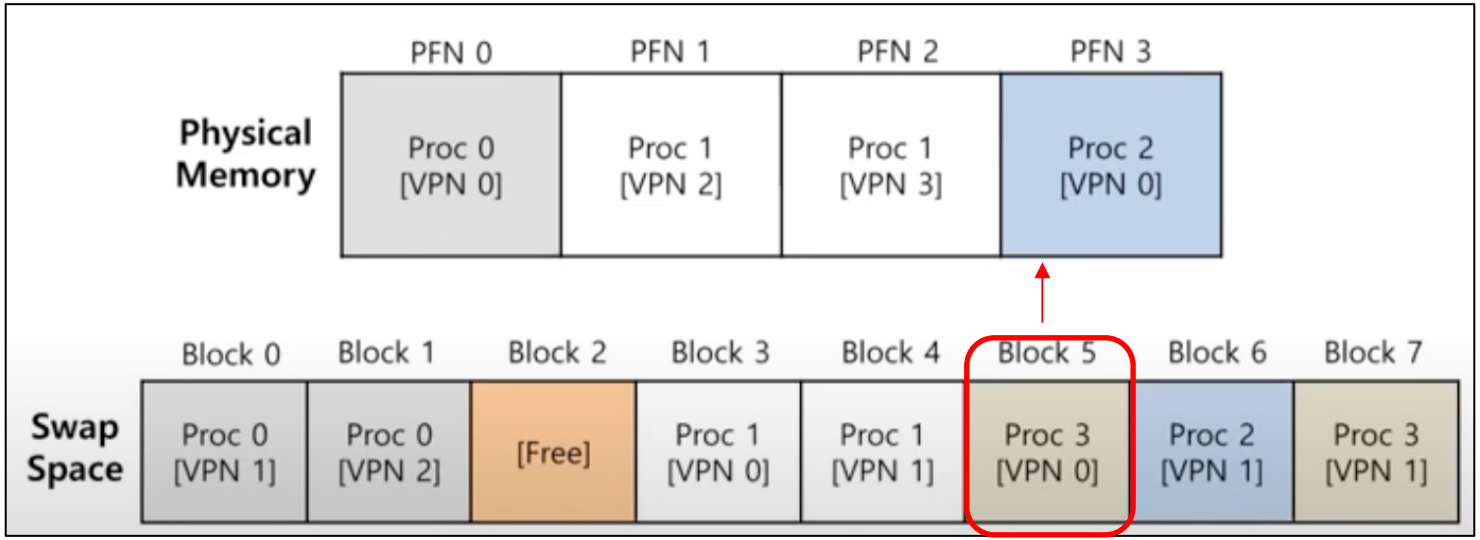
Present Bit（存在位）

- 在系统中添加一些机制，以支持页面在磁盘的交换
  - 当硬件查看页表项（PTE）时，可能会发现页面不在物理内存中

Value	Meaning
1	page is present in physical memory
0	The page is not in memory but rather on disk.

如果内存满了怎么办？

- 操作系统倾向于换出一些页面，为系统即将引入的新页面腾出空间
  - 选择要踢出或替换的页面的过程被称为**页面替换策略**。



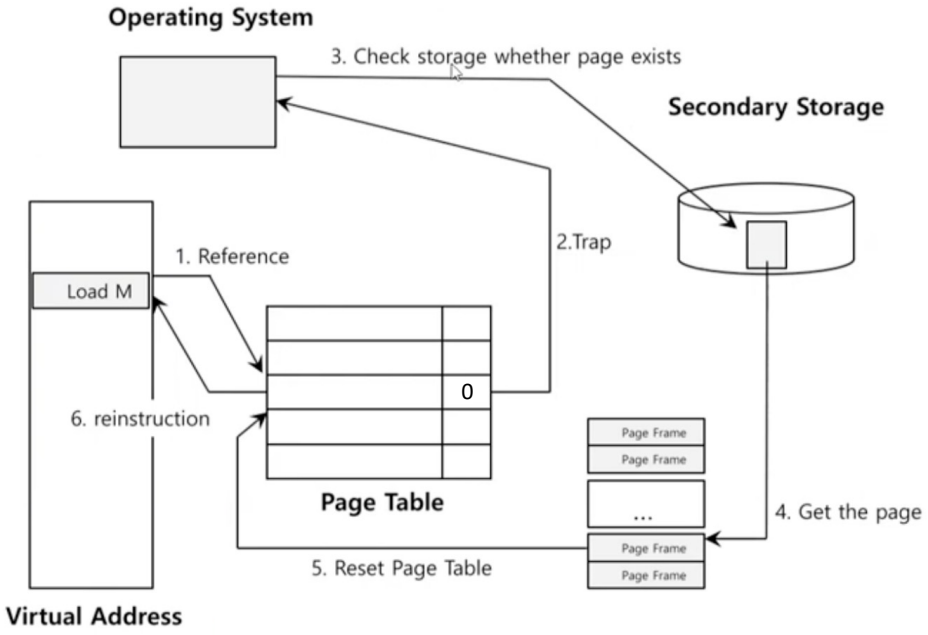
页错误（The Page Fault）

•访问**不在物理内存中**的页面

- 如果一个页面不在内存中且已被换出到磁盘，操作系统需要将该页面换入内存以处理页面错误。

页错误控制流

- PTE 用于数据，例如磁盘地址的物理页帧号 (PFN)



1. **引用**（从虚拟地址到页表）
2. **陷入**（从页表到操作系统）
3. **检查存储中是否存在页面**（从操作系统到二级存储）
4. **获取页面**（从二级存储到页表）
5. **重置页表**（从页表内部更新）
6. **重新执行**（从页表回到虚拟地址）

当操作系统接收到页面错误时，它会查看 PTE 并发出磁盘请求

页错误控制流 – 硬件

```
1:      VPN = (VirtualAddress & VPN_MASK) >> SHIFT
2:      (Success, TlbEntry) = TLB_Lookup(VPN)
3:      if (Success == True){ // TLB Hit
4:      if (CanAccess(TlbEntry.ProtectBits) == True)
5:              Offset = VirtualAddress & OFFSET_MASK
6:              PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
7:              Register = AccessMemory(PhysAddr)
8:      else RaiseException(PROTECTION_FAULT)
```

页错误控制流 – 硬件

```
9:      }else{ // TLB Miss
10:     PTEAddr = PTBR + (VPN * sizeof(PTE))
11:     PTE = AccessMemory(PTEAddr)
12:     if (PTE.Valid == False)
13:             RaiseException(SEGMENTATION_FAULT)
14:     else
15:     if (CanAccess(PTE.ProtectBits) == False)
16:             RaiseException(PROTECTION_FAULT)
17:     else if (PTE.Present == True)
18:     // assuming hardware-managed TLB
19:             TLB_Insert(VPN, PTE.PFN, PTE.ProtectBits)
20:             RetryInstruction()
21:     else if (PTE.Present == False)
22:             RaiseException(PAGE_FAULT)
```

页错误控制流 – 软件

```
1:        PFN = FindFreePhysicalPage()
2:        if (PFN == -1) // no free page found
3:                PFN = EvictPage() // run replacement algorithm
4:                DiskRead(PTE.DiskAddr, pfn) // sleep (waiting for I/O)
5:                PTE.present = True // update page table with present
6:                PTE.PFN = PFN // bit and translation (PFN)
7:                RetryInstruction() // retry instruction
```

- 操作系统必须为即将调入的页面找到一个物理页
- 如果没有这样的页面，则等待**页面替换算法**运行并将一些页面踢出内存

页面替换何时真正发生

- **操作系统等到内存完全满时，才会替换页面以腾出空间给其他页面**
  - 这有点不现实，操作系统有许多理由需要主动保持一小部分内存空闲。

- **交换守护进程（Swap Daemon）/页守护进程（Page Daemon）**
  - 当可用页面少于**低水位线**（Low Watermark pages）时，一个后台线程负责释放内存
  - 该线程会逐出页面，直到可用页面达到**高水位线**（High Watermark pages）

有用的Linux指令

- 交换空间查看：
  - swapon --show --output-all
- 内存使用情况查看：free –h
- 内存统计工具：smem
- 进程交换空间使用分析：
  - for file in /proc/*/status ; do awk '/VmSwap|Name/{printf $2 " " $3}END{ print ""}' $file; done | sort -k 2 -n -r | less
- 监控交换守护进程：top -p `pidof kswapd0`

swapon & free指令

- 交换空间查看：
  - swapon --show --output-all

- 内存使用情况查看：free –h

```
[intro]$ swapon --show --output-all
NAME      TYPE SIZE USED PRIO UUID LABEL
/swap.img file 2.8G 4.8M   -2
[intro]$ free -h
               total        used        free      shared  buff/cache   available
Mem:           3.3Gi       308Mi       3.0Gi       4.1Mi       110Mi       3.0Gi
Swap:          2.8Gi       4.8Mi       2.8Gi
```

smem指令

```
[intro]$ smem
  PID User     Command                         Swap      USS      PSS      RSS
 1188 ty       /usr/bin/dbus-daemon --sess        0      352      849     3928
 1184 ty       tmux                               0      372     1013     3572
 1186 ty       tmux                               0     1224     2022     4628
 1084 ty       -bash                              0     2504     2656     4700
 1187 ty       -bash                             32     2300     2905     5424
 1200 ty       -bash                              0     2440     3039     5564
 2738 ty       /usr/bin/python3 /usr/bin/s        0     8236    10176    14012
```

进程交换空间使用指令

•进程交换空间使用分析：

- for file in /proc/*/status; do awk '/VmSwap|Name/{printf $2 " " $3}END{ print ""}' $file; done | sort -k 2 -n -r | less

•for file in /proc/*/status; do ... done：该部分循环遍历 /proc 目录下的每个进程的 status 文件。

•awk '/VmSwap|Name/{printf $2 " " $3}END{ print ""}'：这部分使用 awk 过滤出每个进程的 VmSwap（交换空间使用量）和 Name（进程名称）字段。

•sort -k 2 -n -r：按照交换空间使用量进行降序排序，-k 2 指定第二列（交换空间）为排序依据，-n 表示数字排序，-r 表示降序。

```
unattended-upgr 1792 kB
(sd-pam) 1152 kB
bash 128 kB
watchdogd
usb-storage
udisksd 0 kB
tmux: server0 kB
tmux: client0 kB
systemd-udevd 0 kB
systemd-timesyn 0 kB
systemd-resolve 0 kB
systemd-network 0 kB
systemd-logind 0 kB
systemd-journal 0 kB
systemd 0 kB
systemd 0 kB
sort 0 kB
scsi_eh_0
rsyslogd 0 kB
rcu_tasks_trace_kthread
rcu_tasks_rude_kthread
rcu_tasks_kthread
rcu_preempt
```

监控交换守护进程指令

•监控交换守护进程：top -p `pidof kswapd0`

```
top - 04:24:50 up 46 min,  2 users,  load average: 0.01, 0.04, 0.09
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.2 sy,  0.0 ni, 99.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3389.4 total,   2268.3 free,    330.5 used,    956.9 buff/cache
MiB Swap:   2910.0 total,   2905.5 free,      4.5 used.   3058.9 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
     51 root      20   0       0      0      0 S   0.0   0.0   0:01.18 kswapd0
```

小结

- 我们介绍了访问超出物理内存大小时的交换机制
  - 介绍了交换空间
  - 存在位（present bit）
- 介绍了页错误的概念和处理流程
- 介绍了一些交换空间相关的Linux指令