抽象-地址空间

邰颖
南京大学
智能科学与技术学院

内存虚拟化

什么是内存虚拟化（**Memory Virtualization**）？

- 操作系统对物理内存进行虚拟化
- 操作系统为每个进程提供**虚拟的内存空间**（illusion memory space）
- 每个进程**看起来像是独占整个内存**（但实际上是共享的）

内存虚拟化

内存虚拟化的优势

- 编程的易用性。不需要直接管理物理内存，可以使用连续的虚拟地址

- 在**时间和空间**上的内存利用效率
  - 时间效率：仅在需要时加载数据，提高性能
  - 空间效率：允许多个进程共享代码/数据，交换到磁盘空间等

- 保证进程和操作系统的隔离
  - 防止其他进程的错误访问

早期系统中的操作系统

- 操作系统只是一个库
- 内存中仅加载一个进程
  - 进程执行至完成
    - **CPU** 利用率和内存效率低
      - 当进程执行 I/O 操作时，**CPU** 处于空闲状态
    - 内存利用率低，其他进程本可以被加载到内存中

多道程序与时分共享

- **多道程序与时分共享系统**：在内存中加载多个进程
  - •短时间内执行一个进程
  - •在内存中的进程之间切换
  - •结果：提高 CPU 利用率和内存效率
- 相比批处理系统，系统交互性更强
- 引入了一个重要的**保护问题**
  - 如果一个用户进程访问了另一个用户进程的数据，或者直接访问了内核怎么办？

3 个进程：共享内存

创建对物理内存的抽象

•操作系统创建物理内存的抽象：

1.进程地址空间：每个进程有独立的虚拟地址空间，包括代码段、数据段、堆和栈。

2.地址映射：内存管理单元（MMU, Memory Management Unit）通过页表（Page Table）负责将虚拟地址转换为物理地址。

3.分页（Paging）：让内存管理更灵活。

4.交换空间（Swap）：在物理内存不足时，将一部分数据换出到磁盘，提高可用性。

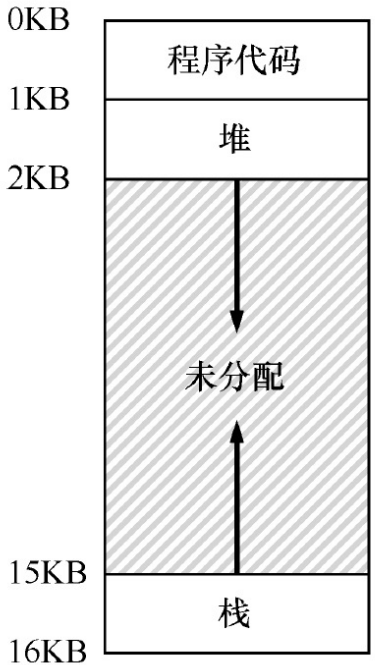
创建对物理内存的抽象（续）

- **操作系统创建物理内存的抽象：**
  - 进程看到的内存空间是连续的，而实际的物理地址可能是分散的。
  - 进程彼此隔离，无法直接访问其他进程的内存。
  - 操作系统可以高效管理和分配内存，提高系统性能和安全性。

抽象：地址空间

•地址空间：运行的程序看到的系统中的内存

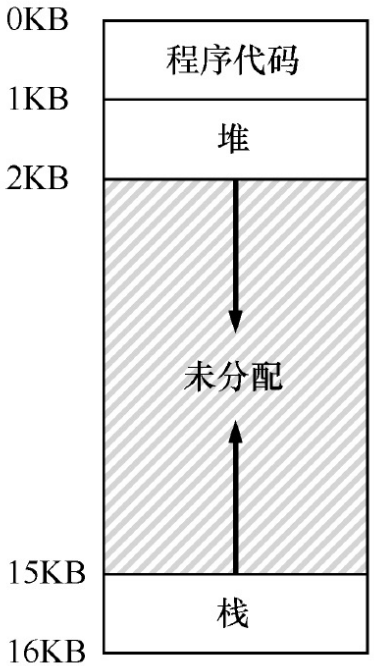
- 地址空间/映射包含运行进程的所有内存信息
- 运行进程拥有**自己的内存状态**：
  - **重要：这可能并不是物理内存中的实际地址！**
- 地址空间通常由**程序代码（Program Code）**、**堆（Heap）**、**栈（Stack）**等部分组成。



地址空间
（16KB）

抽象：地址空间

- 代码区（Code）
  - 存放指令的位置
- 堆区（Heap）
  - 动态分配内存
    - C 语言使用 malloc()
    - 面向对象语言使用 new
- 栈区（Stack）
  - 在函数调用时存储返回地址或返回值
  - 包含局部变量、函数的实际参数



地址空间
（16KB）

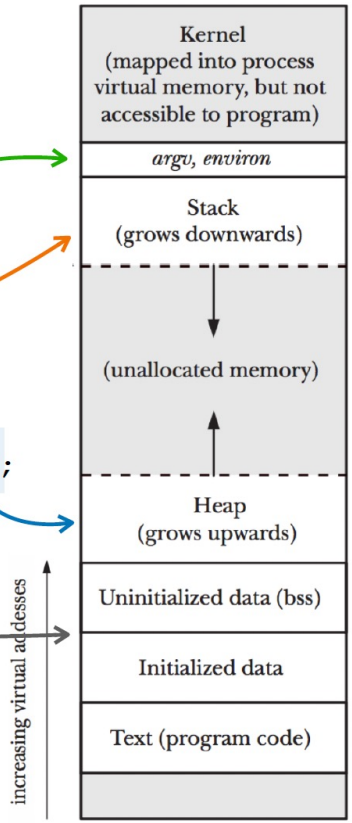
地址空间示例

```
int x;
int y = 15;

int main(int argc, char *argv[])
{
  int *value;
  int i;

  value = (int *)malloc(sizeof(int)*5);
  for (i = 0; i < 5; I++)
    value[i] = I;

  return 0;
}
```



内核空间：内核映射到进程地址空间的区域，但进程无法直接访问

命令行参数、环境变量

栈（向低地址增长）　局部变量、函数调用等

堆（向高地址增长）　动态内存分配

未初始化的数据段
已初始化的数据段　全局变量 静态变量

代码段　程序指令

虚拟地址示例

- 每个正在运行的程序（进程）中的地址都是虚拟的！
  - 操作系统（OS）将 **虚拟地址（virtual address）** 转换为 **物理地址（physical address）**。

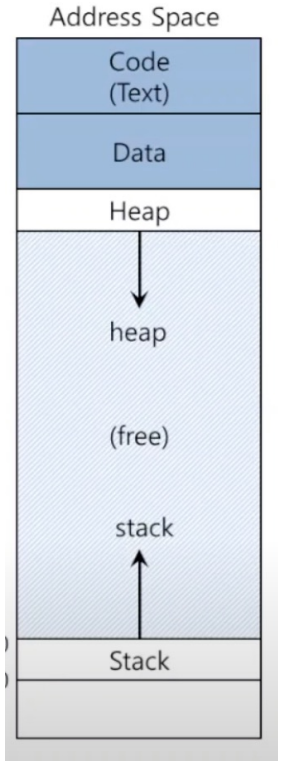
```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    printf("location of code : %p\n", main);
    printf("location of heap : %p\n", malloc(100e6));
    int x = 3;
    printf("location of stack: %p\n", &x);
    return 0;
}
```

A simple program that prints out virtual addresses

虚拟地址示例演示

```
[vm-intro]$ ./va
location of code : 0xb58456940928
location of heap : 0xf1213ae01010
location of stack: 0xffffe450cc34
[vm-intro]$ ./va
location of code : 0xabbd6bc90928
location of heap : 0xf112c31a1010
location of stack: 0xffffebe7f344
[vm-intro]$ ./va
location of code : 0xc7e4401f0928
location of heap : 0xe6a97b1e1010
location of stack: 0xffffdc0cc324
```



虚拟地址示例演示

```
[vm-intro]$ sudo sysctl kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[vm-intro]$ ./va
location of code : 0xaaaaaaaa0928
location of heap : 0xfffff1e91010
location of stack: 0xffffffff034
[vm-intro]$ ./va
location of code : 0xaaaaaaaa0928
location of heap : 0xfffff1e91010
location of stack: 0xffffffff034
[vm-intro]$ ./va
location of code : 0xaaaaaaaa0928
location of heap : 0xfffff1e91010
location of stack: 0xffffffff034
```

虚拟地址示例调试

gdb ./va

b * main

r

info proc mapping

```
process 42738
Mapped address spaces:

          Start Addr           End Addr       Size     Offset  Perms  objfile
      0xaaaaaaaa0000     0xaaaaaaaa1000     0x1000          0x0  r-xp   /mnt/os-ex
amples/ostep-code-master/vm-intro/va
      0xaaaaaaaabf000     0xaaaaaaaac0000     0x1000     0xf000  r--p   /mnt/os-ex
amples/ostep-code-master/vm-intro/va
      0xaaaaaaaac0000     0xaaaaaaaac1000     0x1000    0x10000  rw-p   /mnt/os-ex
amples/ostep-code-master/vm-intro/va
      0xffffff7df0000     0xffffff7f8f000   0x19f000          0x0  r-xp   /usr/lib/a
arch64-linux-gnu/libc.so.6
      0xffffff7f8f000     0xffffff7f9d000     0xe000    0x19f000  ---p   /usr/lib/a
arch64-linux-gnu/libc.so.6
      0xffffff7f9d000     0xffffff7fa0000     0x3000    0x1ad000  r--p   /usr/lib/a
arch64-linux-gnu/libc.so.6
      0xffffff7fa0000     0xffffff7fa2000     0x2000    0x1b0000  rw-p   /usr/lib/a
arch64-linux-gnu/libc.so.6
      0xffffff7fa2000     0xffffff7fae000     0xc000          0x0  rw-p
      0xffffff7fbe000     0xffffff7fe6000    0x28000          0x0  r-xp
arch64-linux-gnu/ld-linux-aarch64.so.1
      0xffffff7ff7000     0xffffff7ff9000     0x2000          0x0  rw-p
      0xffffff7ff9000     0xffffff7ffb000     0x2000          0x0  r--p   [vvar]
--Type <RET> for more, q to quit, c to continue without paging--
```

代码、数据段

共享库代码、数据

虚拟地址示例调试

// 单步执行
n

info proc mapping

```
amples/ostep-code-master/vm-intro/va
      0xaaaaaaac1000     0xaaaaaaae2000    0x21000        0x0  rw-p   [heap]
      0xffffff1e91000    0xffffff7df0000  0x5f5f000        0x0  rw-p
      0xffffff7df0000    0xffffff7f8f000   0x19f000        0x0  r-xp   /us
arch64-linux-gnu/libc.so.6
      0xffffff7f8f000    0xffffff7f9d000    0xe000   0x19f000  ---p   /usr/lib/a
arch64-linux-gnu/libc.so.6
      0xffffff7f9d000    0xffffff7fa0000    0x3000   0x1ad000  r--p   /usr/lib/a
arch64-linux-gnu/libc.so.6
      0xffffff7fa0000    0xffffff7fa2000    0x2000   0x1b0000  rw-p   /usr/lib/a
arch64-linux-gnu/libc.so.6
      0xffffff7fa2000    0xffffff7fae000    0xc000        0x0  rw-p
      0xffffff7fbe000    0xffffff7fe6000   0x28000        0x0  r-xp   /usr/lib/a
arch64-linux-gnu/ld-linux-aarch64.so.1
--Type <RET> for more, q to quit, c to continue without paging--
      0xffffff7ff7000    0xffffff7ff9000    0x2000        0x0  rw-p
      0xffffff7ff9000    0xffffff7ffb000    0x2000        0x0  r--p   [vvar]
      0xffffff7ffb000    0xffffff7ffc000    0x1000        0x0  r-xp   [vdso]
      0xffffff7ffc000    0xffffff7ffe000    0x2000    0x2e000  r--p   /usr/lib/a
arch64-linux-gnu/ld-linux-aarch64.so.1
      0xffffff7ffe000    0xffffff8000000    0x2000    0x30000  rw-p   /us
arch64-linux-gnu/ld-linux-aarch64.so.1
      0xfffffffdf000  0x1000000000000    0x21000        0x0  rw-p   [stack]
```

堆

栈

虚拟内存系统的目标

- **透明性 (Transparency)**
  - 实现过程对进程应该是不可见的
  - 进程应表现得好像它拥有自己独立的物理内存
- **效率 (Efficiency)**
  - 在时间上（不能显著降低进程运行速度）和空间上（内存使用不能有太多开销）都要高效
  - 需要依赖硬件支持（如 TLBs 等）
- **保护与隔离 (Protection and Isolation)**
  - 防止一个用户进程访问其他用户进程或内核

小结

介绍了内存虚拟化

介绍了操作系统如何创建对物理内存的抽象

介绍了进程的地址空间