作者：代兴　邮箱：503268771@qq.com　Vx 公众号：东方架构师

内存布局的组成部分

Linux 保护模式，每个进程有独立的虚拟内存空间。多个进程，虚拟内存空间相互隔离，安全性高。

进程的内存布局，包含多个部分，由操作系统、编译器指定。
内存地址从小到大包括程序区、堆区、内存映射区、栈区、直接映射区、动态映射区、固定映射区等。

程序区
把源代码编译为可执行程序。运行程序时，把程序加载到内存中，CPU 执行内存中的程序指令。
程序区包含代码段、数据段等，位于低地址内存。

堆区
brk 函数、sbrk 函数可以调整堆的大小。
malloc 函数分配动态内存，从堆区申请大块内存。

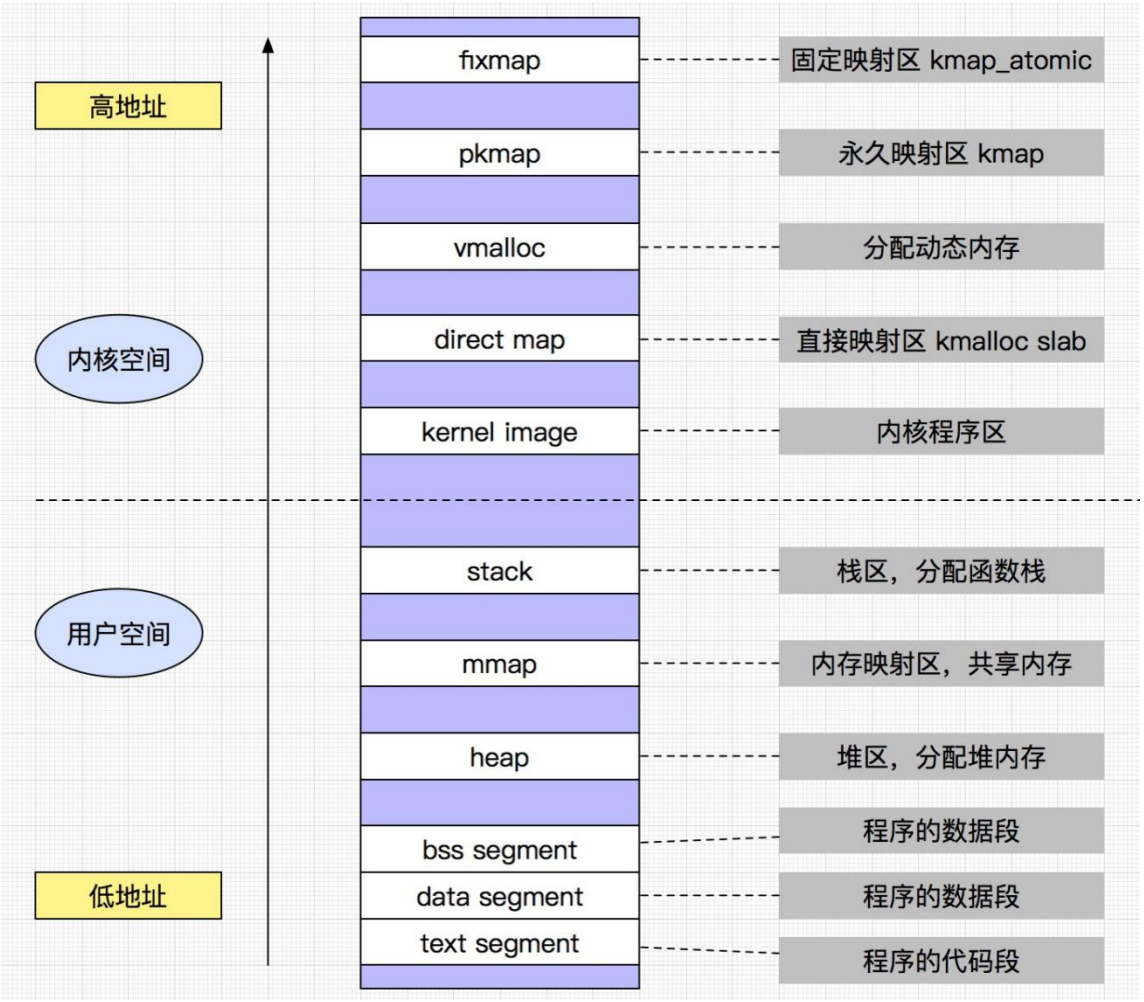
内存映射区
内存映射文件，mmap 函数打通内存系统和文件系统。
有名文件映射，使用真实的文件。比如动态库 so 文件。
匿名文件映射，没有真实的文件。比如多个进程实现共享内存。
malloc 函数分配大块内存时，使用 mmap 函数映射大块内存。

栈区
为线程分配函数栈。地址的增长方向，从高地址向低地址。
函数栈的大小有限制，比如 8MB。如果地址超过大小，则触发栈溢出异常。

直接映射区
连续的一块虚拟内存，连续的一块物理内存，被提前写到页表，线性映射。
虚拟地址和物理地址，可以通过加减 PAGE_OFFSET 相互转换。
kmalloc 函数，slab 分配器，操作直接映射区。

动态映射区
vmalloc 函数分配动态内存。适用于长期映射，分配大小不定的内存。

固定映射区
kmap_atomic 函数临时映射单个物理页 page，返回一个虚拟地址。
比如，写文件时，先通过文件页缓存获得某个位置对应的物理页 page，然后用 kmap_atomic 函数返回一个虚拟地址，然后在这个虚拟地址写数据。



用命令查看进程的内存布局

进程的信息，可以通过目录/proc 查看。

```
[root@localhost proc]# pwd
/proc
[root@localhost proc]# ll
total 0
dr-xr-xr-x.  9 root    root                0 Dec 11 12:34 1
dr-xr-xr-x.  9 root    root                0 Dec 11 12:34 10
dr-xr-xr-x.  9 root    root                0 Dec 11 12:34 11
dr-xr-xr-x.  9 root    root                0 Dec 11 12:34 1104
dr-xr-xr-x.  9 postfix postfix             0 Dec 11 12:34 1109
```

某些文件的名称是数字，这些文件表示单个进程，数字表示进程 ID。
以 1104 为例。

```
[root@localhost proc]# ps aux | grep 1104
root       1104  0.0  0.2  89708  2216 ?        Ss   12:34   0:00 /usr/libexec/postfix/master -w
[root@localhost 1104]# pwd
/proc/1104
[root@localhost 1104]# ls
attr        clear_refs        cpuset    fd        limits      mem           net          oom_score
```

```
personality  schedstat  stack   syscall  wchan
autogroup   cmdline           cwd       fdinfo    loginuid    mountinfo    ns           oom_score_adj
projid_map   sessionid  stat    task
auxv        comm             environ  gid_map  map_files  mounts      numa_maps  pagemap        root
setgroups  statm   timers
cgroup      coredump_filter   exe        io         maps         mountstats   oom_adj     patch_state
sched        smaps      status  uid_map
```

目录/proc/1104 包含很多文件、子目录，表示进程的 CPU、内存、文件、任务、运行统计等。这里重点关注内存信息 maps、smaps。

文件 maps 表示进程的内存区间的简单信息。条目很多，这里展示部分。
第一列为内存地址的范围，从上往下内存地址依次增加。

```
[root@localhost 1104]# cat maps
5644099ee000-564409a15000 r-xp 00000000 fd:00 651154                     /usr/libexec/postfix/master
564409c14000-564409c16000 r--p 00026000 fd:00 651154                     /usr/libexec/postfix/master
564409c16000-564409c17000 rw-p 00028000 fd:00 651154                     /usr/libexec/postfix/master
564409c17000-564409c18000 rw-p 00000000 00:00 0
56440a0de000-56440a0ff000 rw-p 00000000 00:00 0                          [heap]
7f2e1756c000-7f2e17590000 r-xp 00000000 fd:00 24976                      /usr/lib64/libselinux.so.1
7f2e17590000-7f2e1778f000 ---p 00024000 fd:00 24976                      /usr/lib64/libselinux.so.1
7f2e1778f000-7f2e17790000 r--p 00023000 fd:00 24976                      /usr/lib64/libselinux.so.1
7f2e17790000-7f2e17791000 rw-p 00024000 fd:00 24976                      /usr/lib64/libselinux.so.1
7ffe4db07000-7ffe4db28000 rw-p 00000000 00:00 0                          [stack]
```

程序区：
用 ps 查看到进程的程序文件为 `/usr/libexec/postfix/master` ，maps 包含 `/usr/libexec/postfix/master` ，说明程序文件被加载到内存。
程序文件在 maps 对应 3 条记录，权限分别为 r-xp、r--p、rw-p 。代码段、数据段分开，代码段的权限为读执行，数据段分为只读、可读可写。

堆区：
[heap]表示堆内存。

内存映射区：
so 库文件，被加载到内存。和程序文件类似，按照权限分为多个记录。

栈区：
[stack]表示函数栈使用的内存。

文件 smaps 表示进程的内存区间的详细信息。条目很多，这里展示部分。

```
[root@192 16919]# cat smaps
00400000-00401000              r-xp              00000000              08:03              51418619
/root/code/x86-asm/common2/mem/layout
Size:                  4 kB
Rss:                   4 kB
Pss:                   4 kB
Shared_Clean:          0 kB
Shared_Dirty:          0 kB
```

```
Private_Clean:         4 kB
Private_Dirty:         0 kB
Referenced:            4 kB
Anonymous:             0 kB
AnonHugePages:         0 kB
Swap:                  0 kB
KernelPageSize:        4 kB
MMUPageSize:           4 kB
Locked:                0 kB
VmFlags: rd ex mr mp me dw sd
00656000-00677000 rw-p 00000000 00:00 0                                  [heap]
Size:                132 kB
Rss:                   4 kB
Pss:                   4 kB
Shared_Clean:          0 kB
Shared_Dirty:          0 kB
Private_Clean:         0 kB
Private_Dirty:         4 kB
Referenced:            4 kB
Anonymous:             4 kB
AnonHugePages:         0 kB
Swap:                  0 kB
KernelPageSize:        4 kB
MMUPageSize:           4 kB
Locked:                0 kB
VmFlags: rd wr mr mp me ac sd
```

用代码分析进程的内存布局

编写代码： layout.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h> // mmap
#include <fcntl.h>
#include <string.h>
#include <errno.h>

// 全局变量。在程序的数据区
char cat_name[8] = {'J', 'a', 'c', 'k', '\0'};
```

```
// 全局变量。在程序的数据区
int cat_age = 5;

// 函数。在程序的代码区
void cat_run()
```

```
{
    // 栈内变量。 在栈区
    int tmp = 666;
    printf("cat is running . stack param addr = %p \n", &tmp);
}

// 函数。在程序的代码区
int main()
{
    // 调用函数
    cat_run();
    // 栈内变量。 在栈区
    int day1 = 6;
    // 栈内变量。 在栈区
    int day2 = 7;
    // malloc 函数分配的内存。在 heap 区、mmap 区。
    void *malloc_ptr = malloc(666);
    // mmap 函数分配的内存。在 mmap 区。
    void *mmap_ptr = mmap(NULL, 500, PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANON, -1, 0);

    // printf("error %d %s \n", errno, strerror(errno));

    // 函数。在程序的代码区
    printf("函数 cat_run 的地址  =  %p \n", &cat_run);
    printf("函数 main 的地址     =  %p \n", &main);

    // 全局变量。在程序的数据区
    printf("全局变量 cat_name 的地址  =  %p \n", &cat_name);
    printf("全局变量 cat_age 的地址   =  %p \n", &cat_age);

    // malloc 函数分配的内存。在 heap 区、mmap 区。
    printf("malloc 分配内存的地址  =  %p \n", malloc_ptr);

    // mmap 函数分配的内存。在 mmap 区。
    printf("mmap 分配内存的地址    =  %p \n", mmap_ptr);

    // 栈内变量。 在栈区
    printf("栈内变量 day1 的地址  =  %p \n", &day1);
    printf("栈内变量 day2 的地址  =  %p \n", &day2);

    // 暂停进程。方便查看内存布局。
    sleep(90000);
    return 0;
}
```

编译代码：

```
# 编译为可执行程序
gcc layout.c -o layout
```

```
# 查看 ELF 文件
readelf -a layout > layout.elf.txt
```

运行代码：

```
[root@192 mem]# ./layout
cat is running . stack param addr = 0x7ffe43d5060c
函数 cat_run 的地址  =  0x40060d
函数 main 的地址     =  0x400634
全局变量 cat_name 的地址  =  0x60104c
全局变量 cat_age 的地址   =  0x601054
malloc 分配内存的地址  =  0x227f010
mmap 分配内存的地址    =  0x7f5f14a72000
栈内变量 day1 的地址  =  0x7ffe43d5062c
栈内变量 day2 的地址  =  0x7ffe43d50628
```

查看内存布局：

```
[root@192 mem]# ps aux | grep layout
root      14652  0.0  0.0   4352   356 pts/9    S+   05:45   0:00 ./layout
root      14691  0.0  0.0 112812   980 pts/0    S+   05:46   0:00 grep --color=auto layout
[root@192 mem]# cat /proc/14652/maps
00400000-00401000              r-xp              00000000              08:03              51418619
/root/code/x86-asm/common2/mem/layout
00600000-00601000              r--p              00000000              08:03              51418619
/root/code/x86-asm/common2/mem/layout
00601000-00602000              rw-p              00001000              08:03              51418619
/root/code/x86-asm/common2/mem/layout
0227f000-022a0000 rw-p 00000000 00:00 0                                  [heap]
7f5f14486000-7f5f1464a000 r-xp 00000000 08:03 15928                      /usr/lib64/libc-2.17.so
7f5f1464a000-7f5f14849000 ---p 001c4000 08:03 15928                      /usr/lib64/libc-2.17.so
7f5f14849000-7f5f1484d000 r--p 001c3000 08:03 15928                      /usr/lib64/libc-2.17.so
7f5f1484d000-7f5f1484f000 rw-p 001c7000 08:03 15928                      /usr/lib64/libc-2.17.so
7f5f1484f000-7f5f14854000 rw-p 00000000 00:00 0
7f5f14854000-7f5f14876000 r-xp 00000000 08:03 611075                     /usr/lib64/ld-2.17.so
7f5f14a6b000-7f5f14a6e000 rw-p 00000000 00:00 0
7f5f14a72000-7f5f14a75000 rw-p 00000000 00:00 0
7f5f14a75000-7f5f14a76000 r--p 00021000 08:03 611075                     /usr/lib64/ld-2.17.so
7f5f14a76000-7f5f14a77000 rw-p 00022000 08:03 611075                     /usr/lib64/ld-2.17.so
7f5f14a77000-7f5f14a78000 rw-p 00000000 00:00 0
7ffe43d32000-7ffe43d53000 rw-p 00000000 00:00 0                          [stack]
7ffe43dfe000-7ffe43e00000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

查看 ELF 文件：

查看 layout.elf.txt ，重点看某些符号的信息。

```
Symbol table '.symtab' contains 69 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
    57: 0000000000601054     4 OBJECT  GLOBAL DEFAULT   24 cat_age
    58: 000000000060104c     8 OBJECT  GLOBAL DEFAULT   24 cat_name
    64: 0000000000400634   272 FUNC    GLOBAL DEFAULT   13 main
```

```
    65: 000000000040060d    39 FUNC    GLOBAL DEFAULT   13 cat_run
```

分析结果：
查看 ELF 文件 layout.elf.txt，查看符号 cat_age、cat_name、main、cat_run 的 value，即为内存地址。
这些符号的地址，和运行程序输出的地址一致。比如，符号 cat_run 的地址为 000000000040060d，输出的地址为 0x40060d。高位的 0 可以省去。

函数 cat_run 的地址 = 0x40060d，在程序的代码区 00400000-00401000 。
全局变量 cat_name 的地址 = 0x60104c，在程序的数据区 00601000-00602000 。
malloc 分配内存的地址 = 0x227f010，在堆区 0227f000-022a0000 。
mmap 分配内存的地址 = 0x7f5f14a72000，在内存映射区。同时，很多 so 文件也在内存映射区。
栈内变量 day1 的地址 = 0x7ffe43d5062c，在栈区 7ffe43d32000-7ffe43d53000 。

默认情况，内存布局的顺序和代码符号的顺序一样。编译器优化，可能导致顺序发送改变。
代码中，cat_name 在 cat_age 的前面，内存地址 000000000060104c 在 0000000000601054 的前面。
代码中，cat_run 在 main 的前面，内存地址 000000000040060d 在 0000000000400634 的前面。

函数栈，增长方向为高地址向低地址，栈内变量的顺序和代码符号的顺序反向。
代码中，栈内变量 day1 在 day2 的前面，内存地址 0x7ffe43d5062c 在 0x7ffe43d50628 的后面。