



多线程编程(4)

说证玩^Mwww.yomocode.com

麦当劳喜欢您来,喜欢您再来



扫描关注 Linux阅码场



多线程编程(4) WWW.Yomocode.com

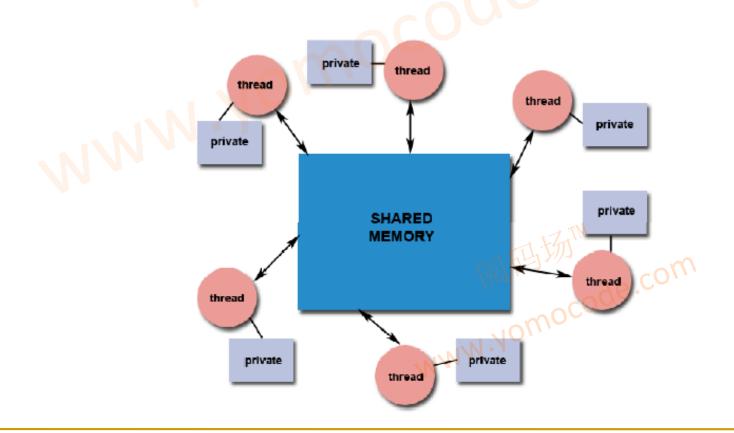
多线程与栈

4.1 线程的栈

4.2 guard page

线程栈的双重特性

栈:线程独有,保存其运行状态和局部自动变量的。栈在线程开始的时候初始化,每个线程的栈互相独立(但内存属于整个进程),因此,栈是thread safe的。



栈的guard page



POSIX库API

#include <pthread.h>

int pthread_attr_setguardsize(pthread_attr_t *attr, size_t guardsize);
int pthread_attr_getguardsize(const pthread_attr_t *attr, size_t
*guardsize);

WWW.Yomocode.com

有意思的strace 的结果

mmap(NULL, 20480, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7fbfcd14c000 mprotect(0x7fbfcd14d000, 16384, PROT_READ|PROT_WRITE) = 0

mmap(NULL, 20480, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7fbfcd147000 mprotect(0x7fbfcd148000, 16384, PROT_READ|PROT_WRITE) = 0

WWW.yomocode.com

pthread_create创建的线程 内存情况



内核提供的针对GROWSDOWN栈的guard page

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=3 20b2b8de12698082609ebbc1a17165727f4c893

```
+ * This is like a special single-page "expand downwards()"
+ * except we must first make sure that 'address-PAGE_SIZE'
    doesn't hit another vma.
+ * The "find vma()" will do the right thing even if we wrap
+static inline int check_stack_guard_page(struct_vm_area_struct_*vma, unsigned_long_address)
        address &= PAGE MASK;
        if ((vma->vm flags & VM GROWSDOWN) && address == vma->vm start) {
                address -= PAGE SIZE;
                if (find_vma(vma->vm_mm, address) != vma)
                        return -ENOMEM;
                expand stack(vma, address);
        return 0:
  * We enter with non-exclusive mmap_sem (to exclude vma changes,
  * but allow concurrent faults), and pte mapped but not yet locked.
  * We return with mmap sem still held, but pte unmapped and unlocked.
@@ -2772, 6 +2792, 9 @@ static int do_anonymous_page(struct mm struct *mm, struct vm_area_struct *vma,
        spinlock t *ptl;
        pte_t entry;
        if (check_stack_guard_page(vma, address) < 0)
                return VM_FAULT_SIGBUS;
        if (!(flags & FAULT FLAG WRITE)) {
                entry = pte mkspecial(pfn pte(my zero pfn(address),
                                                vma->vm page prot));
```

内核的一些改进

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=1be7107fbe18eed3e319a6c3e83c78254b693acb



mm: larger stack guard gap, between vmas

Stack guard page is a useful feature to reduce a risk of stack smashing into a different mapping. We have been using a single page gap which is sufficient to prevent having stack adjacent to a different mapping. But this seems to be insufficient in the light of the stack usage in userspace. E.g. glibc uses as large as 64kB alloca() in many commonly used functions. Others use constructs liks gid_t buffer[NGROUPS_MAX] which is 256kB or stack strings with MAX_ARG_STRLEN.

This will become especially dangerous for suid binaries and the default no limit for the stack size limit because those applications can be tricked to consume a large portion of the stack and a single glibc call could jump over the guard page. These attacks are not theoretical, unfortunatelly.

Make those attacks less probable by increasing the stack guard gap to 1MB (on systems with 4k pages; but make it depend on the page size because systems with larger base pages might cap stack allocations in the PAGE_SIZE units) which should cover larger allocal() and VLA stack allocations. It is obviously not a full fix because the problem is somehow inherent, but it should reduce attack space a lot.

One could argue that the gap size should be configurable from userspace, but that can be done later when somebody finds that the new IMB is wrong for some special case applications. For now, add a kernel command line option (stack_guard_gap) to specify the stack gap size (in page units).

Implementation wise, first delete all the old code for stack guard page: because although we could get away with accounting one extra page in a stack vma, accounting a larger gap can break userspace — case in point, a program run with "ulimit —S —v 20000" failed when the 1MB gap was counted for RLIMIT_AS; similar problems could come with RLIMIT_MLOCK and strict non-overcommit mode.

Instead of keeping gap inside the stack vma, maintain the stack guard gap as a gap between vmas: using vm start gap() in place of vm start.

谢谢!N.Yomocode.com

www.yomocode.com





阅码场出品