PB173 – Ovladače jádra – Linux
IX.

Jiri Slaby

ITI, Fakulta Informatiky

29. 11. 2012

Paměť jinak I.

LDD3 kap. 15 (zastaralá)

- I. (dnes): Mapování paměti jádra (mmap)
- II. (příště): Přímý přístup do paměti (DMA)

Mapování paměti jádra

Předání dat do/z procesu

- Známe: `read`, `write`, `ioctl`, ...
- U všeho nutné kopírování dat
  - `copy_{from,to}_user` apod.

Mapování paměti

- Namapování stránek do procesu
- Proces používá kus stejné paměti jako jádro
- Syscall: `mmap`

mmap

- V userspace volání `mmap`
  - `void *mmap(void *addr, size_t len, int prot, int flags, int fd, off_t off)`
- V jádře jeden člen v `struct file_operations`
  - `int mmap(struct file *filp, struct vm_area_struct *vma)`
  - Parametry jsou v `struct vm_area_struct`

Úkol

Alokace pomocí `mmap` (uživatelský prostor)

1. Anonymní paměť
   - `void *mmap(void *addr, size_t len, int prot, int flags, int fd, off_t off)`
   - `len` ... 20M
   - `prot` ... `PROT_READ` a `PROT_WRITE`
   - `flags` ... `MAP_PRIVATE` a `MAP_ANONYMOUS`
   - `fd` ... -1
2. Mapování /dev/zero
   - `flags` ... `MAP_PRIVATE`
   - `fd` ... deskriptor otevřeného /dev/zero

Parametry `mmap`

Prototyp (znovu):

```
void *mmap(void *addr, size_t len, int prot, int flags, int fd,
off_t off)
```

- Jsou v jádře uloženy v `struct vm_area_struct`

```
struct vm_area_struct {
  unsigned long vm_start; /* addr or random when addr is NULL */
  unsigned long vm_end;   /* vm_end = vm_start+len */
  unsigned long vm_pgoff; /* vm_pgoff = off/PAGE_SIZE */
  unsigned long vm_flags; /* vm_flags = encoded(flags|prot) */
  pgprot_t vm_page_prot;  /* only for remap_* functions */
  ...
  const struct vm_operations_struct *vm_ops; /* later ... */
  void *vm_private_data;
}
```

Je třeba namapovat stránky mezi `vm_start` a `vm_end`.
Ale také ověřit privilegia (čtení, zápis, spuštění).

Základní mmap funkce

API

- **linux/mm.h**, struct vm_area_struct
- __get_free_page/pages ⇒ remap_pfn_range
- vmalloc_user ⇒ remap_vmalloc_range

```
int my_init(void)
{
  mem = __get_free_pages(GFP_KERNEL, 2);
  /* mem = vmalloc_user(PAGE_SIZE); */
}
...
int my_mmap(struct file *filp, struct vm_area_struct *vma)
{
  if ((vma->vm_flags & (VM_WRITE | VM_READ)) != VM_READ)
    return -EINVAL;
  return remap_pfn_range(vma, vma->vm_start, virt_to_phys(mem) >>
      PAGE_SHIFT, 4 * PAGE_SIZE, vma->vm_page_prot);
  /* return remap_vmalloc_range(vma, mem, 0); */
}
```

Úkol

Přemapování prostorů

1. Alokovat 2stránku (vmalloc) a 2stránku (page alokátory)
2. Zapsat na všechny 4 stránky libovolný, ale různý řetězec
3. Přemapovat stránky v `mmap`
   - První dvojice RO, druhá R/W
   - $0 \leq$ `pgoff` $< 2 \Rightarrow$ jedno mapování
   - $2 \leq$ `pgoff` $< 4 \Rightarrow$ druhé mapování
4. Z userspace 2$\times$ `mmap` s off 0 a 2*4096
   - Již hotovo v pb173/09

API znovu

- **`linux/mm.h`**, `struct vm_area_struct`
- `int mmap(struct file *filp, struct vm_area_struct *vma)`
- `__get_free_page*` $\Rightarrow$ `remap_pfn_range`
- `vmalloc_user` $\Rightarrow$ `remap_vmalloc_range`

`mmap` po stránkách

Přemapování roztroušených stránek

- Přes `remap_pfn_range` obtížně
- Při výpadcích stránek se mapují tyto stránky jednotlivě
  - Každý ovladač má „page fault handler”
  - Stará jádra: `nopage`
  - Nová jádra: `fault`
- V `mmap`/`nopage`/`fault` je třeba zkontrolovat rozsahy a velikosti
  - Před tím to dělaly `remap_*_range` funkce
- `vma->vm_ops`
  - Struktura ukazující na háčky (včetně `nopage`/`fault`)
- `vma->vm_private_data`
  - Pro naše potřeby
  - K předání informací z `mmap` do `vm_ops`

mmap po stránkách (ponovu)

Nové API

```
struct vm_operations_struct {
  void (*open)(struct vm_area_struct *vma);
  void (*close)(struct vm_area_struct *vma);
  int (*fault)(struct vm_area_struct *vma, struct vm_fault *vmf);
}
int my_fault(struct vm_area_struct *vma, struct vm_fault *vmf)
{ /* my_data == vma->vm_private_data; */
  unsigned long offset = vmf->pgoff << PAGE_SHIFT;
  struct page *page;
  page = my_find_page(offset);
  if (!page)
    return VM_FAULT_SIGBUS;
  get_page(page);
  vmf->page = page;
  return 0;
}
int my_mmap(struct file *filp, struct vm_area_struct *vma)
{ /* don't forget to check ranges */
  vma->vm_ops = &my_vm_ops;
  vma->vm_private_data = my_data;
  return 0;
}
```

mmap po stránkách (postaru)

Staré API

```
struct vm_operations_struct {
  ...
  struct page (*nopage)(struct vm_area_struct *vma, unsigned long
      address, int *type)
}
struct page *my_nopage(struct vm_area_struct *vma, unsigned long
    address, int *type)
{ /* my_data = vma->vm_private_data; */
  unsigned long offset = vma->pgoff << PAGE_SHIFT;
  struct page *page;

  offset += address - vma->vm_start;
  page = my_find_page(offset);
  if (!page)
    return NOPAGE_SIGBUS;
  get_page(page);

  if (type)
    *type = VM_FAULT_MINOR;

  return page;
}
```

Úkol

Mapování roztroušených stránek (domácí)

1. Předchozí příklad rozšiřte
2. Místo alokací 2stránek alokovat 4 samostatné stránky
3. Přemapovat stránky v `mmap`
   - Změna `remap_pfn_range` na `vma->vm_ops->nopage`
4. Userspace program stejný (funkčnost navenek stejná)

Potřebné podkroky

- Definice `int fault(struct vm_area_struct *vma, struct vm_fault *vmf);`
  - Popř. staré `struct page *nopage(struct vm_area_struct *vma, unsigned long address, int *type)`
- Kontrola rozsahů v `mmap` (`end-start < 2*PAGE_SIZE` apod.)
- Definice `vm_ops` a přiřazení do `vma->vm_ops`