

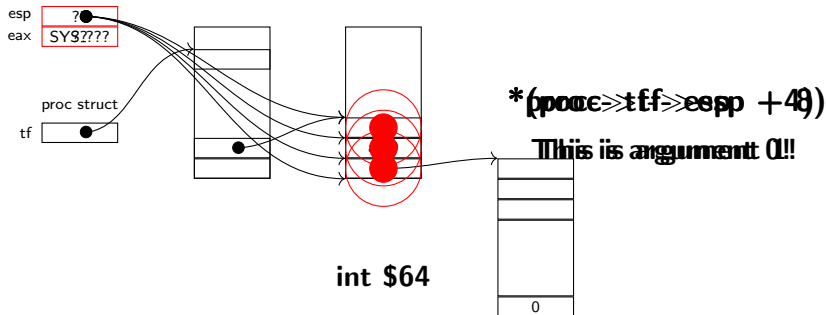
xv6©-rev10  
(Copyright Frans Kaashoek, Robert Morris, and Russ Cox.)  
exec syscall I

Carmi Merimovich

Tel-Aviv Academic College

January 10, 2017

## sys\_exec arguments



## sys\_exec

```
6526 int sys_exec(void) {  
    char *path, *argv[MAXARG];  
    int i;  
    uint uargv, uarg;  
    if (argstr(0,&path)<0 || argint(1,(int*)&uargv)<0)  
        return -1;  
    memset(argv, 0, sizeof(argv));  
    for (i=0; ; i++) {  
        if (i >= NELEM(argv)) return -1;  
        if (fetchint(uargv+4*i,(int*)&uarg)<0) return -1;  
        if (uarg == 0) {  
            argv[i] = 0;  
            break;  
        }  
        if (fetchstr(uarg, &argv[i]) < 0) return -1;  
    }  
    return exec(path, argv);  
}
```

## exec

```
exec(char *elf, char *argv[]);
```

There are four steps in **exec**'s implementation:

1. ELF file loading.
2. Allocating user stack and guard page.
3. Passing `argv[]` from the old address space to the new one.
4. Switching address spaces and fixing `trapframe`.

exec: step 1.

- We hold the knowledge to implement steps 2–4 above.
- What knowledge is crucially missing in order to implement step 1?
  - How to read from files.
  - The ELF.

## Reading from file in kernel mode

## Kernel file system interface for reading file

For now we use this kpi, just as user mode programmers use cluelessly the **open/read/close** api

```
void begin_op(void);  
struct inode *namei(char *name);  
void                ilock(struct inode *ip);  
  
int                readi(struct inode *ip,  
                        char *buf, uint len,  
                        uint pos);  
  
void                iunlockput(struct inode *ip);  
void end_op(void);
```



## Open/Read loop/Close

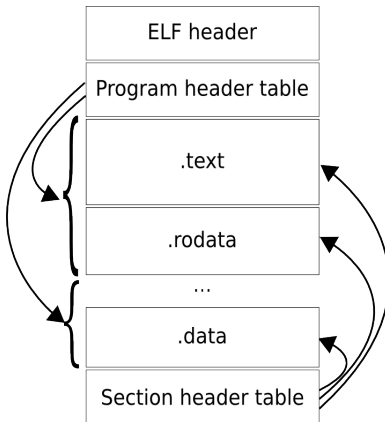
```
begin_op();  
ip = namei(filename);  
  
ilock(ip);  
for (...) {  
    error = readi(ip, buf, bufsiz, pos);  
    if (error <= 0) ...  
:  
}  
  
iunlockput(ip);  
end_op();
```

# (static) Executable and Linkable Format (ELF)

## ELF components

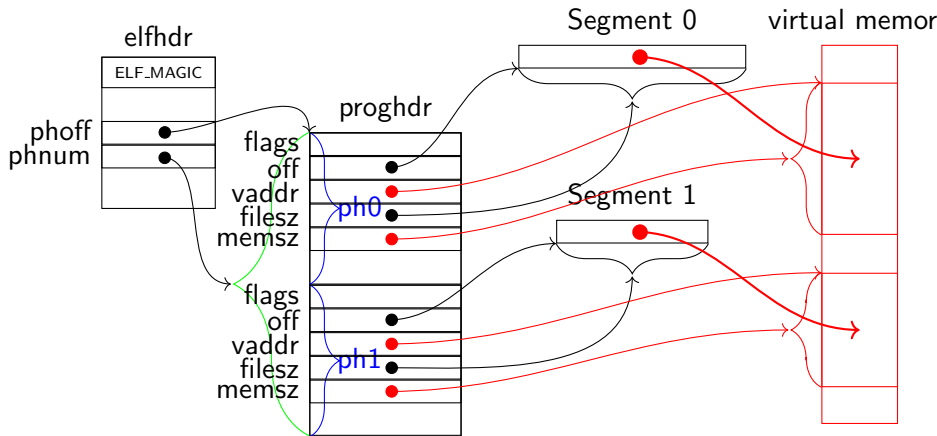
1. ELF header. Must begin at byte zero of the file.
2. PROGHDR vector.
3. Program segments.

## ELF file, very abstract



By Suruea - Own work, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=2922605>

## ELF, more detailed



## ELFHDR

```
955 struct elfhdr {  
    uint magic; // must equal ELF_MAGIC  
    uchar elf[12];  
    ushort type;  
    ushort machine;  
    uint version;  
    uint entry; // Entry point  
    uint phoff; // (File) Location of PROGHDR vectors  
    uint shoff;  
    uint flags; // flag  
    ushort ehsize;  
    ushort phentsize;  
    ushort phnum; // Length of PROGHDR vector  
    ushort shentsize;  
    ushort shnum;  
    ushort shstrndx;  
};
```

## ELFHDR fields we are interested in

- **magic**: Should be `ELF_MAGIC` (0x464C457F).
- **entry**: Virtual address the program is starting at.
- **phoff**: File offset the Program segments Headers vector begins at.
- **phnum**: Number of elements in the Program segments Headers vector.

## For each segment there is PROGHDR

```
974 struct proghdr {  
    uint type; // Only PROG_LOAD matters to us  
    uint off; // Section location in file  
    uint vaddr; // Virtual address of section  
    uint paddr; // Physical address of section  
    uint filesz; // Section size in file  
    uint memsz; // Section size in memory  
    uint flags;  
    uint align;  
};
```



# ELF file loading

1. Create a new address space.
2. For each segment with a `PROG_LOAD` type do the following:
  - 2.1 If necessary, allocate memory and add mapping rules so that address `vaddr` up to `vaddr + memsz - 1` will be legal in the new address space.
  - 2.2 Read the file from position `off` to `off + filesz - 1` into memory addresses `vaddr` up to `vaddr + memsz - 1`.
    - Observe: The new address space is not active. Hence we have to read page after page.

## allocuvm(): Mission 2.1 above

```
1927 allocuvm(pde_t *pgdir, uint oldsz, uint newsz) {
    char *mem;
    uint a;
    if (newsz >= KERNBASE) return 0;
    if (newsz < oldsz) return oldsz;
    a = PGROUNDUP(oldsz);
    for (; a < newsz; a += PGSIZE) {
        mem = kalloc();
        if (mem == 0) {
            deallocuvm(pgdir, newsz, oldsz);
            return 0;
        }
        memset(mem, 0, PGSIZE);
        mappages(pgdir, a, PGSIZE, v2p(mem), PTE_W|PTE_U);
    }
    return newsz;
}
```

## loadvm: Mission 2.2 above

```
1903 loadvm(pde_t *pgdir, char *addr,
        struct inode *ip, uint offset, uint sz) {
    uint i, pa, n;
    pte_t *pte;
    if ((uint) addr % PGSIZE != 0) panic("loadvm: _addr
    for (i = 0; i < sz; i += PGSIZE) {
        if ((pte = walkpgdir(pgdir, addr+i, 0)) == 0) pani
        pa = PTE_ADDR(*pte);
        if (sz - i < PGSIZE)
            n = sz - i;
        else
            n = PGSIZE;
        if (readi(ip, p2v(pa), offset+i, n) != n) return -
    }
    return 0;
}
```

## loaduvm vs. readi

```
loaduvm(pde_t *pgdir, char *addr,  
        struct inode *ip, uint offset, uint sz);
```

```
readi(struct inode *ip, char *buf, uint len, uint pos)
```

- **loaduvm** is not tied at all to ELF loading!
- It is a generalized **readi**.
- It allows reading into non-active virtual address spaces!

## exec: ELFHDR loading

6610

```
int exec(char *path, char **argv) {  
    struct elfhdr elf;  
    struct inode *ip;  
    begin_op()  
    if ((ip = namei(path)) == 0) {end_op();  
        return -1;}  
    ilock(ip);  
    pde_t *pgdir pgdir = 0;  
  
    if (readi(ip, (char*)&elf, 0, sizeof(elf)) !=  
        sizeof(elf))  
        goto bad;  
    if (elf.magic != ELF_MAGIC) goto bad;  
  
    if ((pgdir = setupkvm()) == 0) goto bad;  
    sz=0;
```

## exec: Program segments loading

6642

```
struct proghdr ph;  
for (i=0, off=elf.phoff; i < elf.phnum; i++,  
      off += sizeof(ph)) {  
    if (readi(ip, (char*)&ph, off, sizeof(ph)) !=  
        sizeof(ph)) goto bad;  
    if (ph.type != ELF_PROG_LOAD) continue;  
    if (ph.memsz < ph.filesz) goto bad;  
    if (ph.vaddr + ph.memsz < ph.vaddr) goto bad;  
    if ((sz = allocuvm(pgdir, sz,  
                      ph.vaddr + ph.memsz)) == 0) goto bad;  
    if (ph.vaddr % PGSIZE != 0) goto bad;  
    if (loaduvm(pgdir, (char*)ph.vaddr, ip,  
                ph.off, ph.filesz) < 0) goto bad;  
}  
iunlockput(ip);  
end_op(); ip = 0;
```