xv6©-rev10

(Copyright Frans Kaashoek, Robert Morris, and Russ Cox.)

syscall arguments

Carmi Merimovich

Tel-Aviv Academic College

January 5, 2017

Fetching from user mode routines

Safe fetching from user mode using address:

```
int fetchint(int addr, int *i);
int fetchstr(int addr, char *s);
```

Safe fetching from user mode using argument number:

```
int argint(int n, int *i);
int argptr(int n, void *p, int size);
int argstr(int n, char *s);
```

Fetching a byte

- Assume the definition **uint addrs** is given.
- How do we fetch, in C, the byte in address **addrs**?

Fetching a byte

- Assume the definition **uint addrs** is given.
- How do we fetch, in C, the byte in address **addrs**?
- **char b = *(char *)addrs;**

Fetching a byte

- Assume the definition **uint addrs** is given.
- How do we fetch, in C, the byte in address **addrs**?
- **char b = *(char *)addrs;**
- Assume we are in the kernel, and **addrs** was supplied by user mode.
- What is the problem?

Fetching a byte

- Assume the definition **uint addrs** is given.
- How do we fetch, in C, the byte in address **addrs**?
- **char b = *(char *)addrs;**
- Assume we are in the kernel, and **addrs** was supplied by user mode.
- What is the problem?
    - **addrs** is not trustable.

Fetching a byte

- Assume the definition **uint addrs** is given.
- How do we fetch, in C, the byte in address **addrs**?
- **char b = *(char *)addrs;**
- Assume we are in the kernel, and **addrs** was supplied by user mode.
- What is the problem?
    - **addrs** is not trustable.
- What do we do?

Fetching a byte

- Assume the definition **uint addrs** is given.
- How do we fetch, in C, the byte in address **addrs**?
- **char b = *(char *)addrs;**
- Assume we are in the kernel, and **addrs** was supplied by user mode.
- What is the problem?
    - **addrs** is not trustable.
- What do we do?

```
if (addrs < proc->sz)
 b = *(char *)addrs;
else
//
// handle the error
//
```

fetchbyte

```
int fetchbyte(uint addrs, char *cp) {
 if (addrs >= proc->sz)
  return (-1);
 *cp = *(char *)addrs;
 return (0);
}
```

fetchbyte

```
int fetchbyte(uint addrs, char *cp) {
 if (addrs >= proc->sz)
  return (-1);
 *cp = *(char *)addrs;
 return (0);
}
```

- What changes should be done in order to get fetchshort?

fetchshort

```
int fetchshort(uint addrs, short *sp) {
 if (addrs >= proc->sz || addrs+1 >= proc->sz)
  return (-1);
 *sp = *(short *)addrs;
 return (0);
}
```

fetchshort

```
int fetchshort(uint addrs, short *sp) {
 if (addrs >= proc->sz || addrs+1 >= proc->sz)
  return (-1);
 *sp = *(short *)addrs;
 return (0);
}
```

- Do we really need both comparisons above?

fetchshort

```
int fetchshort(uint addrs, short *sp) {
 if (addrs >= proc->sz || addrs+1 >= proc->sz)
  return (-1);
 *sp = *(short *)addrs;
 return (0);
}
```

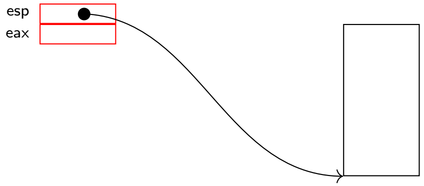
- Do we really need both comparisons above?
    - Yes.

Fetching **long** from an untrusted user address

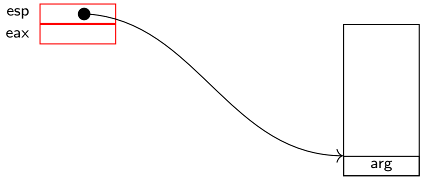
- Fetching a **long** from address **addr** means fetching four bytes from addresses **addr**, **addr+1**, **addr+2**, and **addr+3**.
- All these addresses should all be below **proc->sz**.
- The **fetchint** routine returns −1 if the above is not correct.
- If it is correct, the content of the **long** is put into ***ip**., and the functions returns **0**.

```
int fetchint(uint addr, int *ip) {
 if (addr >= myproc()->sz || addr+4 > myproc()->sz)
  return -1;
 *ip = *(int*)(addr);
 return 0;
}
```

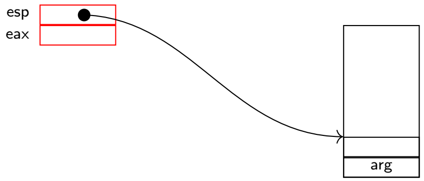
One argument syscall



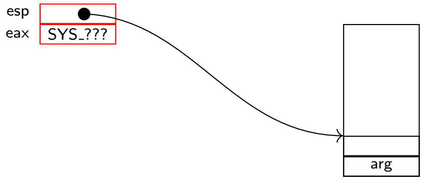
One argument syscall



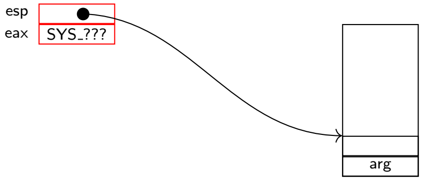
One argument syscall



One argument syscall

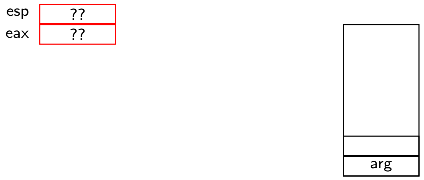


One argument syscall

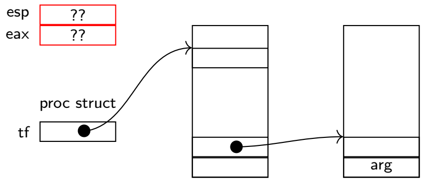


int $64

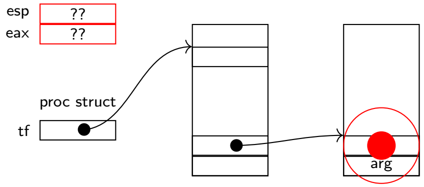
One argument syscall



One argument syscall

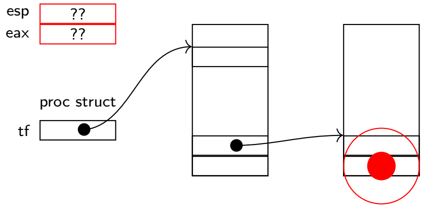


One argument syscall



***myproc()->tf->esp**

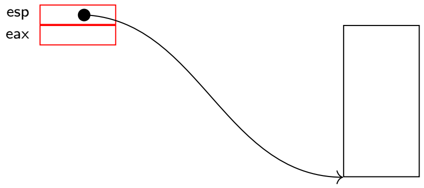
One argument syscall



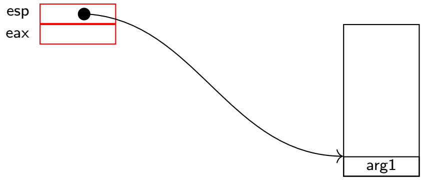
***(myproc()->tf->esp +4)**

This is the argument!

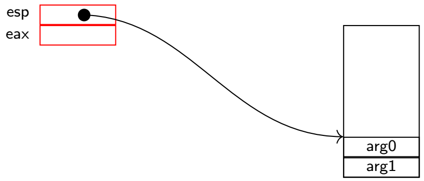
Two arguments syscall



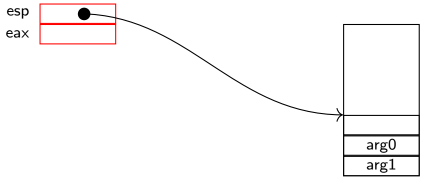
Two arguments syscall



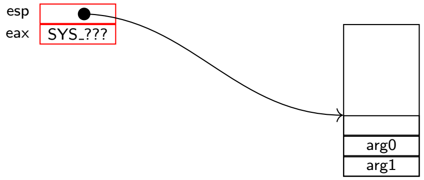
Two arguments syscall



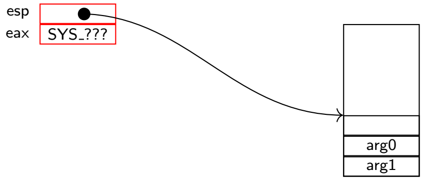
Two arguments syscall



Two arguments syscall

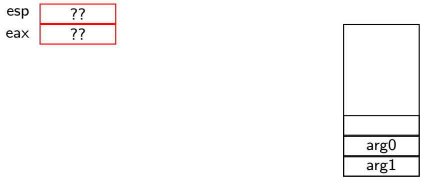


Two arguments syscall

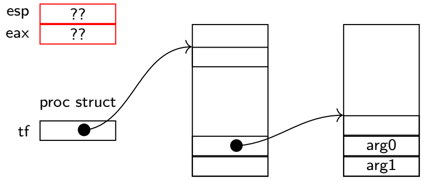


int $64

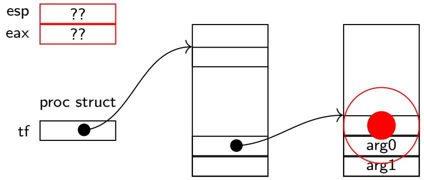
Two arguments syscall



Two arguments syscall

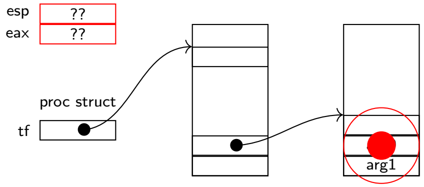


Two arguments syscall



***myproc()->tf->esp**

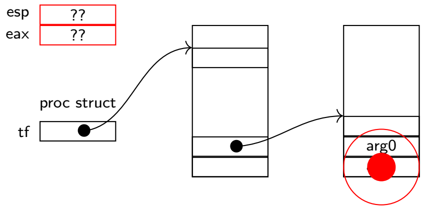
Two arguments syscall



***(myproc()->tf->esp +4)**

This is argument 0!

Two arguments syscall



***(myproc()->tf->esp + 8)**

This is argument 1!

system call args in kernel mode

- The kernel stack replaced the user stack.
- Hence the arguemtns are NOT on the kernel stack.
- However, the user stack address was saved on the `trapframe`.
- Thus in the `syscall` routine we have:
  - arg 0 address is `myproc()->tf->esp + 4`.
  - arg 1 address is `proc->tf->esp + 8`.
  - ⋮
  - arg n address is `proc->tf->esp + 4 + 4` × `n`.
- `proc->tf->esp` was really set by the user, hence untrusted.
- Any computation derived from `proc->tf->esp` is also untrusted.

Fetching the **n**-th integer argument

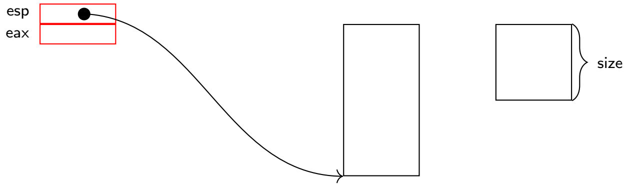
```
int argint(int n, int *ip) {
 return fetchint(myproc()->tf->esp + 4 + 4*n, ip);
}
```

Another form for **argint**

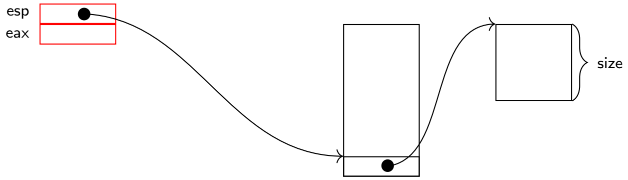
```
struct stackform {
 int ignore;
 int arg[1]; //new compilers allow arg[]
};

int argint(int n, int *ip) {
 return fetchint(&proc->tf->esp->arg[n], ip);
}
```

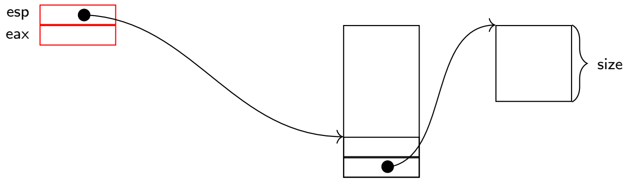
Buffer argument



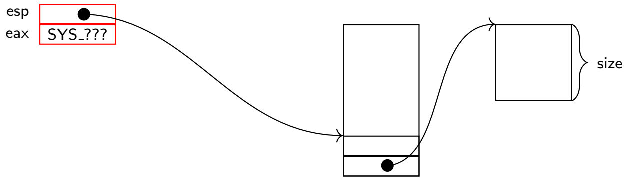
Buffer argument



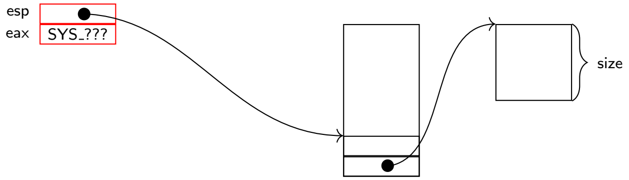
Buffer argument



Buffer argument



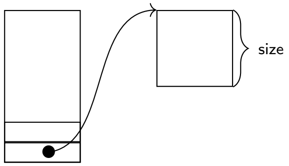
Buffer argument



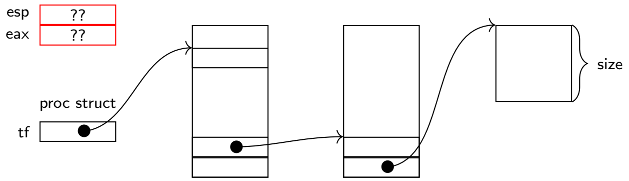
int $64

Buffer argument

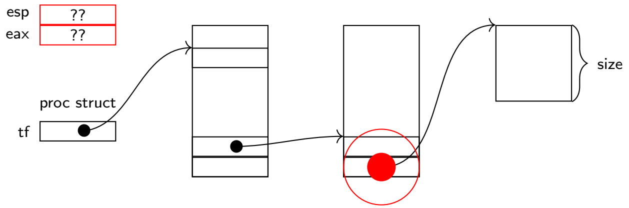
esp	??
eax	??



Buffer argument



Buffer argument



addr=*(proc->tf->esp +4)

***(addr+i)**

Buffer argument

```
int argptr(int n, char **pp, int size) {
 int addr;

 if (argint(n, &addr) < 0)
  return -1;
  if ((uint)addr >= proc->sz ||
            (uint)addr+size > proc->sz)
  return -1;
 *pp = (char*)i;
 return 0;
}
```

String argument

```
3320 int argstr(int n, char **pp) {
      int addr;
      if (argint(n, &addr) < 0)
       return -1;
      return fetchstr(addr, pp);
     }

3278 int fetchstr(uint addr, char **pp) {
      char *s, *ep;

      if (addr >= proc->sz) return -1;
      *pp = (char*)addr;
      ep = (char*)proc->sz;
      for (s = *pp; s < ep; s++)
       if (*s == 0)  return s - *pp;
      return -1;
     }
```