Operating Systems

Lab. Class

Week 2

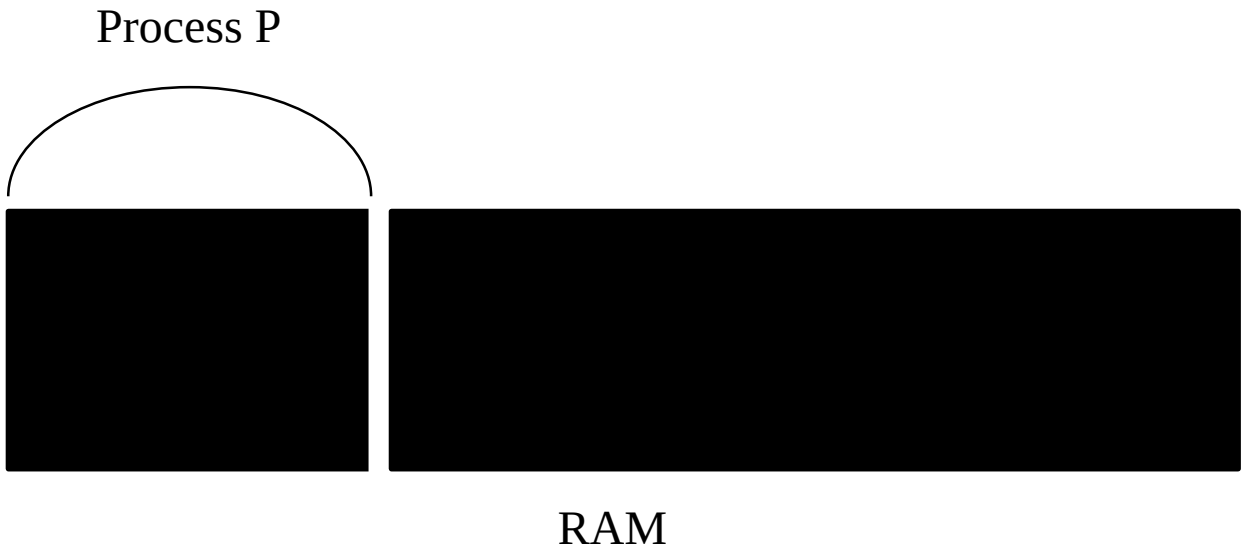
Project Plan

- 5 projects
  0. Install xv6
  1. System call
  2. Scheduling
  3. Virtual memory
  4. Concurrency
  5. File system

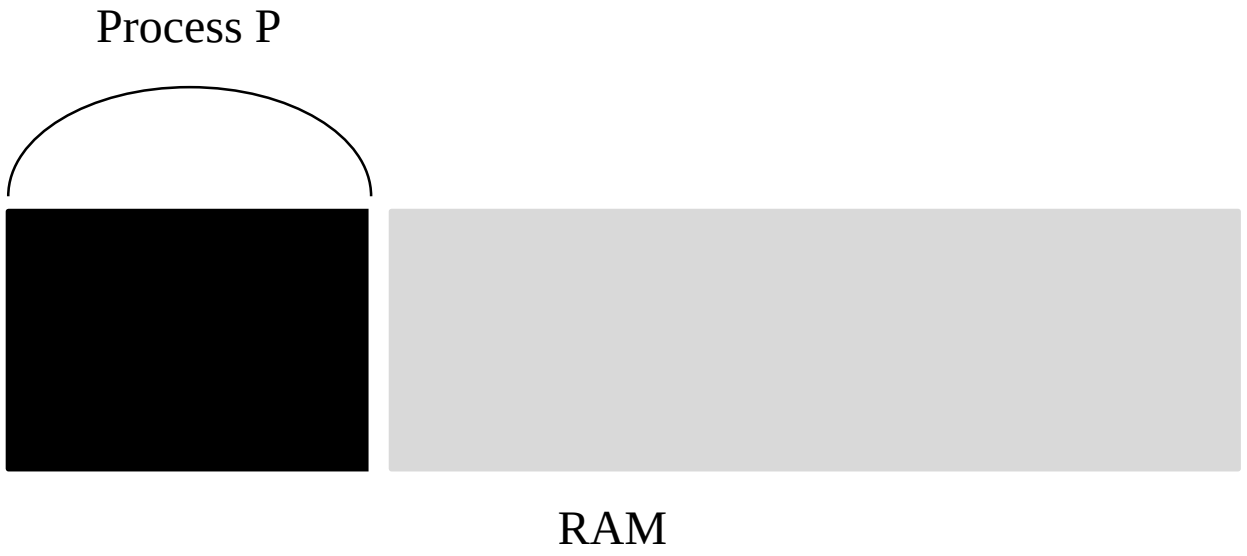
- Individual projects

Trap Handling Process

- Intel architecture

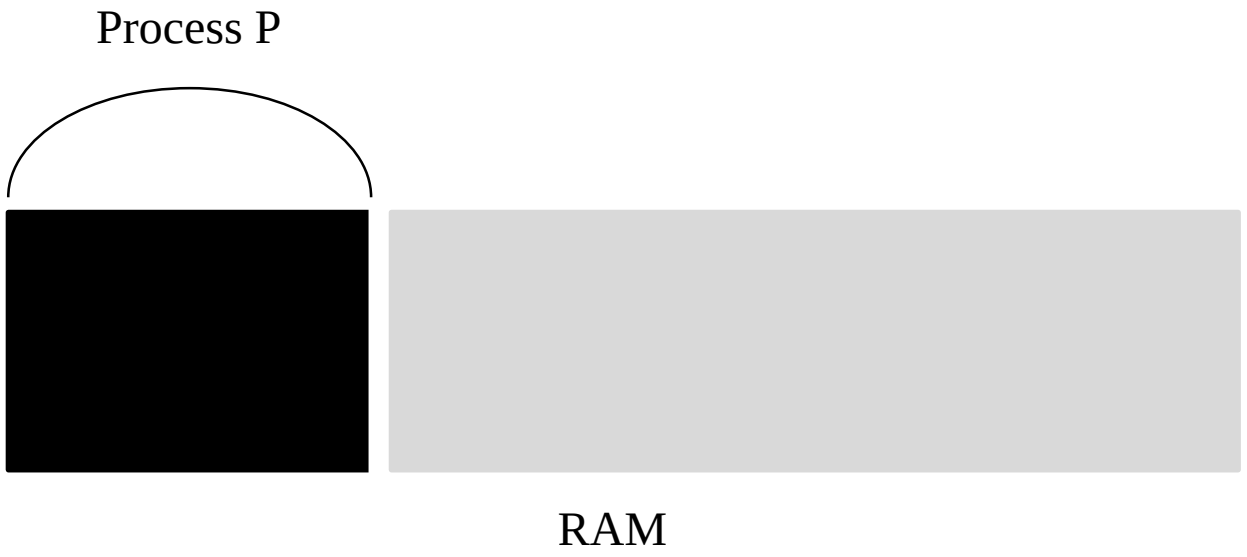


Trap Handling Process (Cont'd)



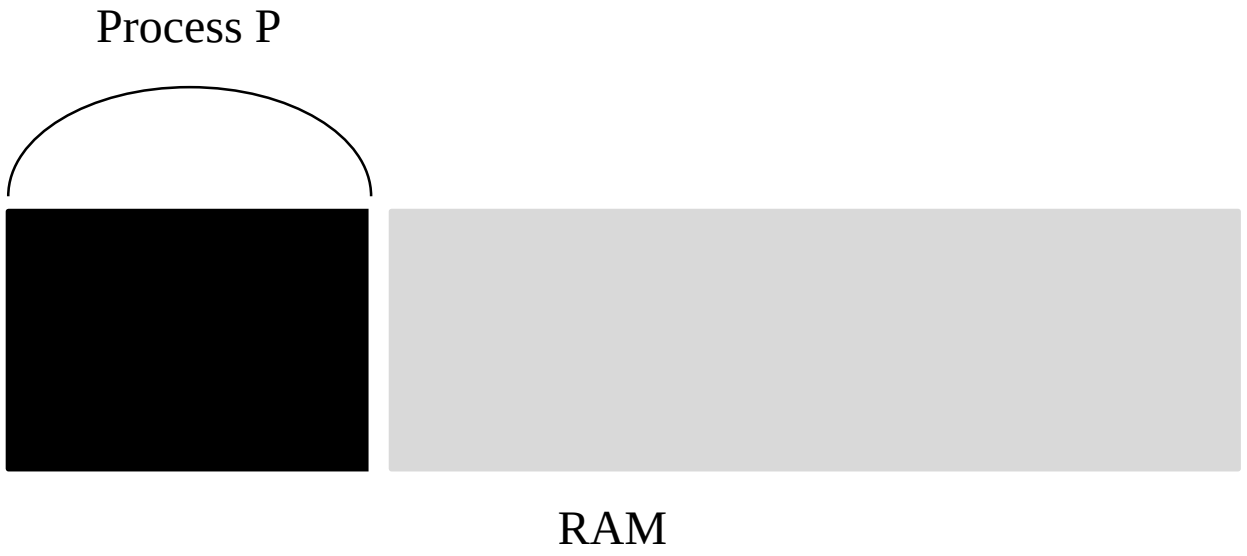
P can only see its own memory because of **user mode**
(other areas, including kernel, are hidden)

Trap Handling Process (Cont'd)



P wants to call kill()

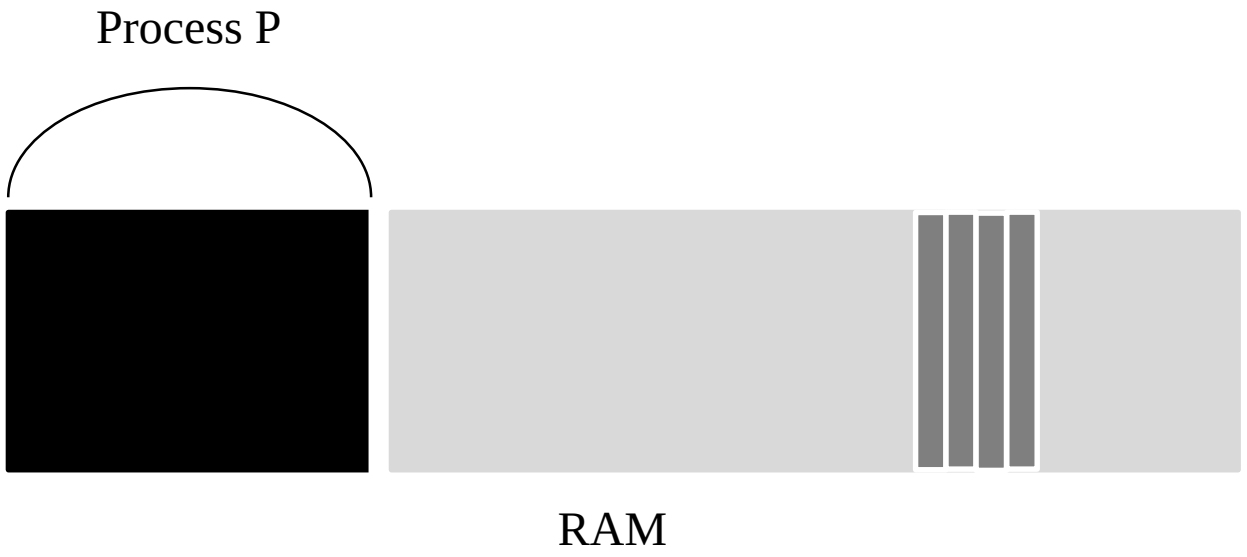
Trap Handling Process (Cont'd)




```
movl $6, %eax;    int $64
```

Trap Handling Process (Cont'd)

```
static int (*syscalls[])(void)    (syscall.c)
```

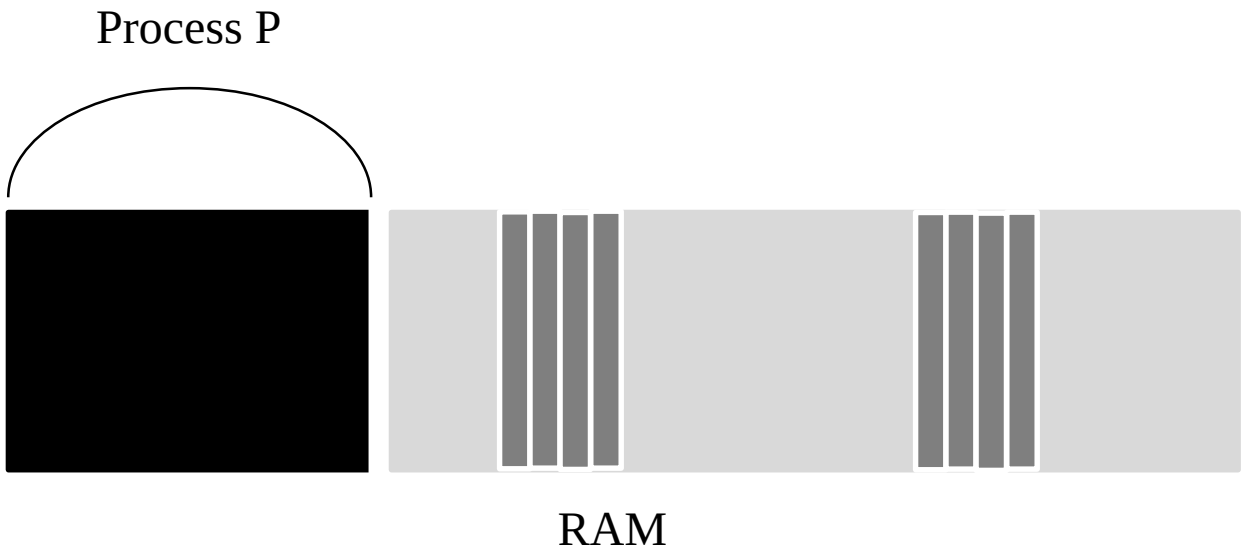



```
movl $6, %eax;    int $64
```

syscall-table index

Trap Handling Process (Cont'd)

```
struct gatedesc idt[256]   (trap.c)
```

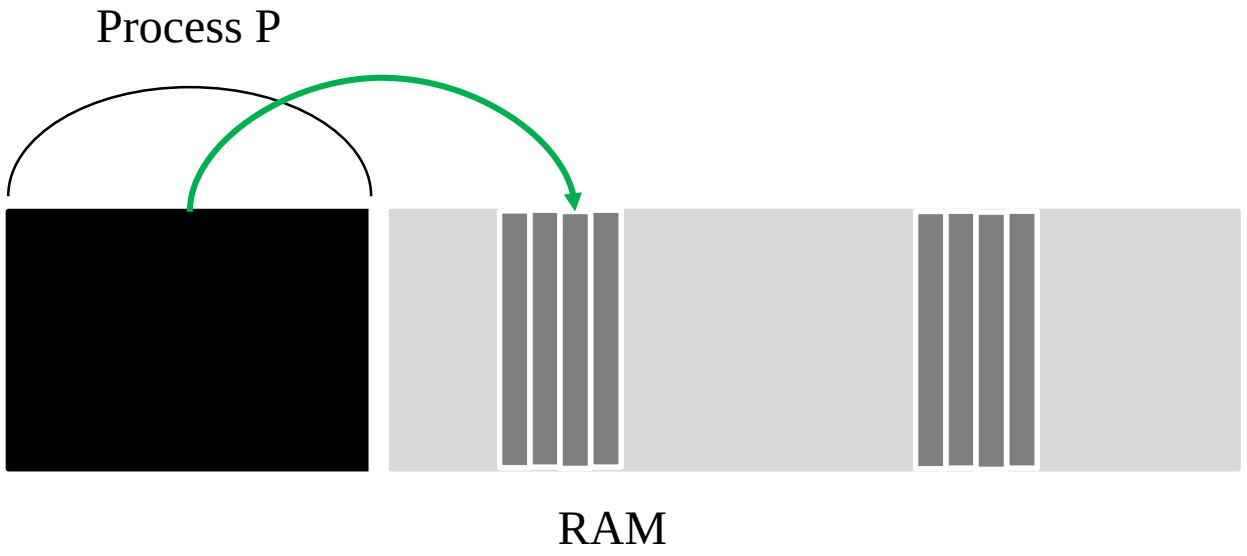



```
movl $6, %eax;     int $64
```

syscall-table index

trap-table index

Trap Handling Process (Cont'd)



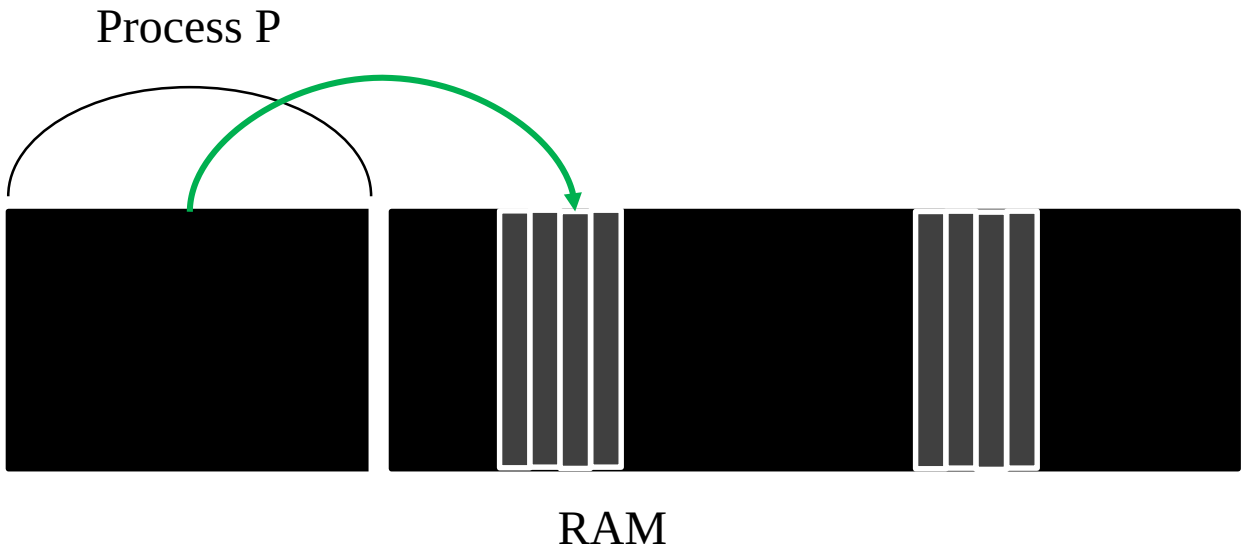

```
movl $6, %eax;     int $64
```

syscall-table index

trap-table index

Trap Handling Process (Cont'd)

Kernel mode: we can do anything!



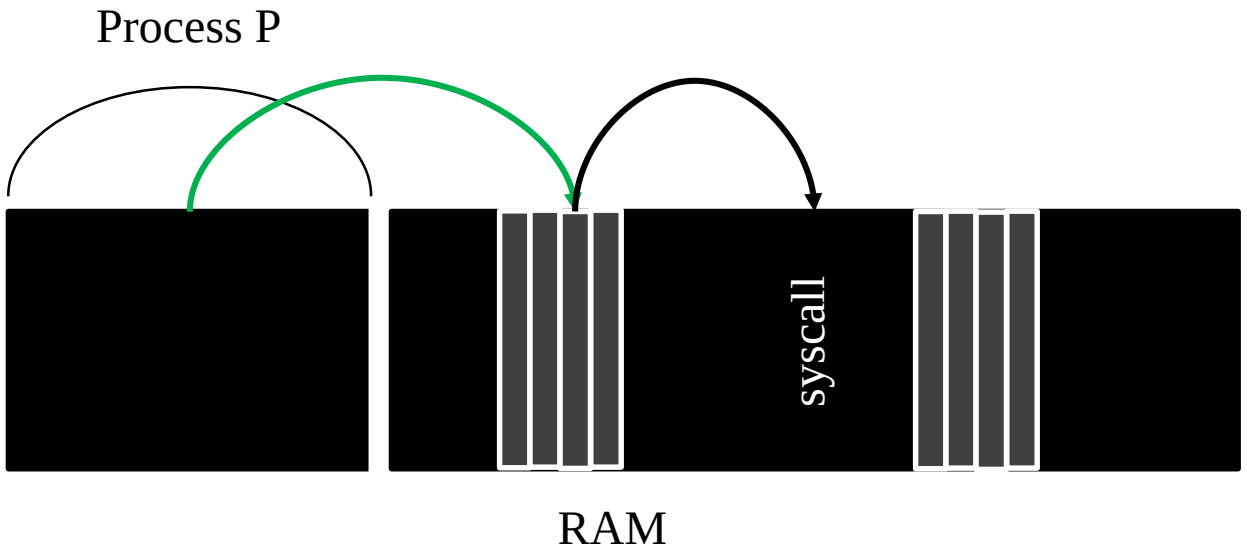

```
movl $6, %eax;
```

syscall-table index

```
int $64
```

trap-table index

Trap Handling Process (Cont'd)

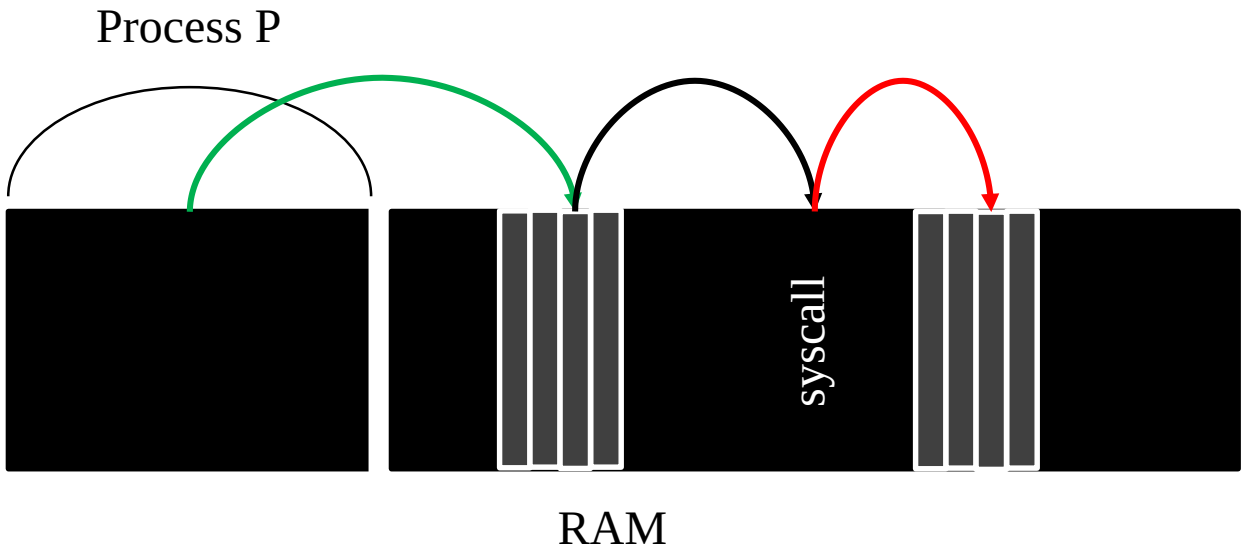



```
movl $6, %eax;     int $64
```

syscall-table index

trap-table index

Trap Handling Process (Cont'd)



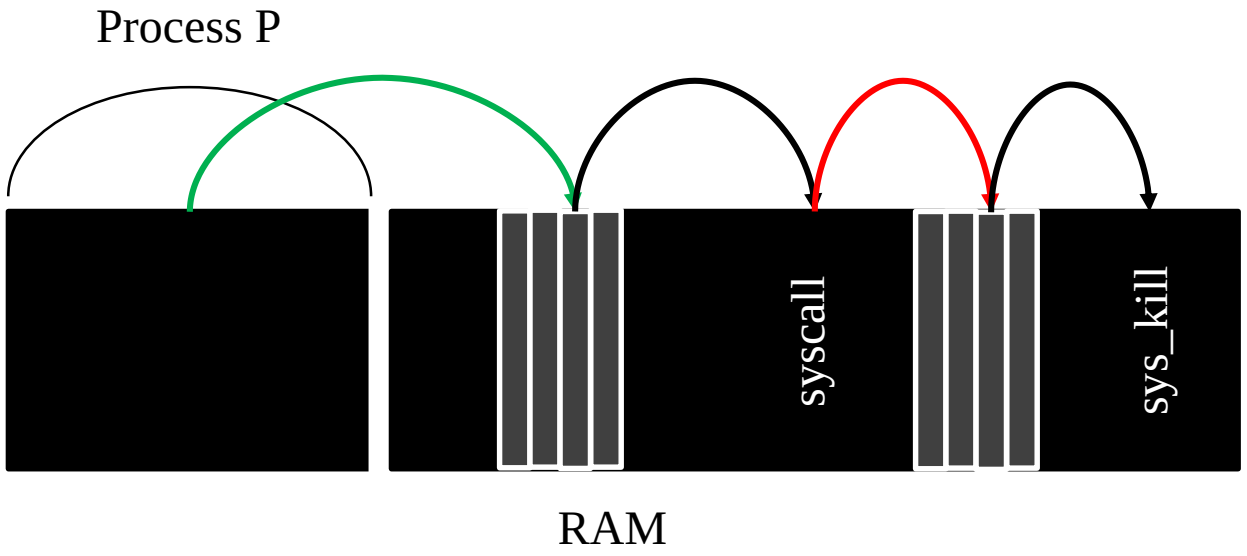

```
movl $6, %eax;
```

syscall-table index

```
int $64
```

trap-table index

Trap Handling Process (Cont'd)




```
movl $6, %eax;
```

syscall-table index

```
int $64
```

trap-table index

Trap Handling Process on Xv6

- Example: `kill` system call

```
 1 #include "types.h"
 2 #include "stat.h"
 3 #include "user.h"
 4
 5 int
 6 main(int argc, char **argv)
 7 {
 8   int i;
 9
10   if(argc < 2){
11     printf(2, "usage: kill pid...\n");
12     exit();
13   }
14   for(i=1; i<argc; i++)
15     kill(atoi(argv[i]));
16   exit();
17 }
```

Trap Handling Process on Xv6 (Cont'd)

- user.h

```
 4 // system calls
 5 int fork(void);
 6 int exit(void) __attribute__((noreturn));
 7 int wait(void);
 8 int pipe(int*);
 9 int write(int, void*, int);
10 int read(int, void*, int);
11 int close(int);
12 int kill(int);
13 int exec(char*, char**);
14 int open(char*, int);
15 int mknod(char*, short, short);
16 int unlink(char*);
17 int fstat(int fd, struct stat*);
18 int link(char*, char*);
19 int mkdir(char*);
20 int chdir(char*);
21 int dup(int);
22 int getpid(void);
23 char* sbrk(int);
24 int sleep(int);
25 int uptime(void);
```

Trap Handling Process on Xv6 (Cont'd)

- usys.S

```
11 SYSCALL(fork)
12 SYSCALL(exit)
13 SYSCALL(wait)
14 SYSCALL(pipe)
15 SYSCALL(read)
16 SYSCALL(write)
17 SYSCALL(close)
18 SYSCALL(kill)
19 SYSCALL(exec)
20 SYSCALL(open)
21 SYSCALL(mknod)
22 SYSCALL(unlink)
23 SYSCALL(fstat)
24 SYSCALL(link)
25 SYSCALL(mkdir)
26 SYSCALL(chdir)
27 SYSCALL(dup)
28 SYSCALL(getpid)
29 SYSCALL(sbrk)
30 SYSCALL(sleep)
31 SYSCALL(uptime)
```

```
1 #include "syscall.h"
2 #include "traps.h"
3
4 #define SYSCALL(name) \
5   .globl name; \
6   name: \
7     movl $SYS_ ## name, %eax; \
8     int $T_SYSCALL; \
9     ret
```

Trap Handling Process on Xv6 (Cont'd)

- syscall.h

```
 1 // System call numbers
 2 #define SYS_fork    1
 3 #define SYS_exit    2
 4 #define SYS_wait    3
 5 #define SYS_pipe    4
 6 #define SYS_read    5
 7 #define SYS_kill    6
 8 #define SYS_exec    7
 9 #define SYS_fstat   8
10 #define SYS_chdir   9
11 #define SYS_dup    10
12 #define SYS_getpid 11
13 #define SYS_sbrk   12
14 #define SYS_sleep  13
15 #define SYS_uptime 14
16 #define SYS_open   15
17 #define SYS_write  16
18 #define SYS_mknod  17
19 #define SYS_unlink 18
20 #define SYS_link   19
21 #define SYS_mkdir  20
22 #define SYS_close  21
```

Trap Handling Process on Xv6 (Cont'd)

- traps.h

```
25 // These are arbitrarily chosen, but with care not to overlap
26 // processor defined exceptions or interrupt vectors.
27 #define T_SYSCALL       64      // system call
28 #define T_DEFAULT      500      // catchall
```

Trap Handling Process on Xv6 (Cont'd)

- trap.c

```
11 // Interrupt descriptor table (shared by all CPUs).
12 struct gatedesc idt[256];
13 extern uint vectors[];  // in vectors.S: array of 256 entry pointers
14 struct spinlock tickslock;
15 uint ticks;
16
17 void
18 tvinit(void)
19 {
20   int i;
21
22   for(i = 0; i < 256; i++)
23     SETGATE(idt[i], 0, SEG_KCODE<<3, vectors[i], 0);
24   SETGATE(idt[T_SYSCALL], 1, SEG_KCODE<<3, vectors[T_SYSCALL], DPL_USER);
25
26   initlock(&tickslock, "time");
27 }
```

Trap Handling Process on Xv6 (Cont'd)

- vectors.S

```
 1 # generated by vectors.pl - do not edit
 2 # handlers
 3 .globl alltraps
 4 .globl vector0
 5 vector0:
 6   pushl $0
 7   pushl $0
 8   jmp alltraps
 9 .globl vector1
10 vector1:
11   pushl $0
12   pushl $1
13   jmp alltraps
14 .globl vector2
```

```
1278 # vector table
1279 .data
1280 .globl vectors
1281 vectors:
1282   .long vector0
1283   .long vector1
1284   .long vector2
1285   .long vector3
1286   .long vector4
1287   .long vector5
1288   .long vector6
1289   .long vector7
1290   .long vector8
1291   .long vector9
1292   .long vector10
```

Trap Handling Process on Xv6 (Cont'd)

- trapasm.S

```
 3   # vectors.S sends all traps here.
 4 .globl alltraps
 5 alltraps:
 6   # Build trap frame.
 7   pushl %ds
 8   pushl %es
 9   pushl %fs
10   pushl %gs
11   pushal
12
13   # Set up data and per-cpu segments.
14   movw $(SEG_KDATA<<3), %ax
15   movw %ax, %ds
16   movw %ax, %es
17   movw $(SEG_KCPU<<3), %ax
18   movw %ax, %fs
19   movw %ax, %gs
20
21   # Call trap(tf), where tf=%esp
22   pushl %esp
23   call trap
24   addl $4, %esp
```

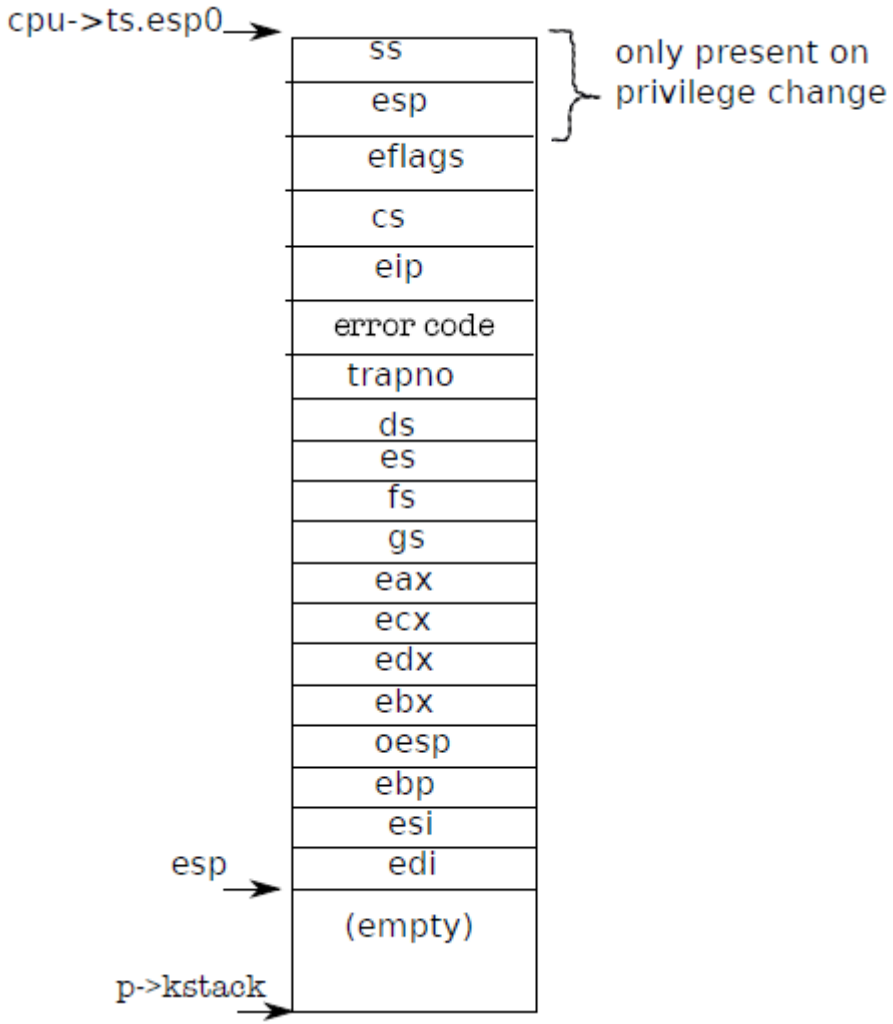


Figure 3-2. The trapframe on the kernel stack

Trap Handling Process on Xv6 (Cont'd)

- trap.c

```
36 void
37 trap(struct trapframe *tf)
38 {
39   if(tf->trapno == T_SYSCALL){
40     if(proc->killed)
41       exit();
42     proc->tf = tf;
43     syscall();
44     if(proc->killed)
45       exit();
46     return;
47   }
```

Trap Handling Process on Xv6 (Cont'd)

- syscall.c

```
102 static int (*syscalls[])(void) = {
103 [SYS_fork]    sys_fork,
104 [SYS_exit]    sys_exit,
105 [SYS_wait]    sys_wait,
106 [SYS_pipe]    sys_pipe,
107 [SYS_read]    sys_read,
108 [SYS_kill]    sys_kill
```

```
 84 extern int sys_exit(void);
 85 extern int sys_fork(void);
 86 extern int sys_fstat(void);
 87 extern int sys_getpid(void);
 88 extern int sys_kill(void);
 89 extern int sys_link(void);
```

```
126 void
127 syscall(void)
128 {
129   int num;
130
131   num = proc->tf->eax;
132   if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
133     proc->tf->eax = syscalls[num]();
134   } else {
135     cprintf("%d %s: unknown sys call %d\n",
136             proc->pid, proc->name, num);
137     proc->tf->eax = -1;
138   }
139 }
```

Trap Handling Process on Xv6 (Cont'd)

- sysproc.c

```
29 int
30 sys_kill(void)
31 {
32   int pid;
33
34   if(argint(0, &pid) < 0)
35     return -1;
36   return kill(pid);
37 }
```

- proc.c

```
411 int
412 kill(int pid)
413 {
414   struct proc *p;
415
416   acquire(&ptable.lock);
417   for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
418     if(p->pid == pid){
419       p->killed = 1;
```