xv6ⓒ-rev10
(Copyright Frans Kaashoek, Robert Morris, and Russ Cox.)

Scheduler

Carmi Merimovich

Tel-Aviv Academic College

November 20, 2017

Context

```
kinit1(end, P2V(4*1024*1024)); // phys page alloca
kvmalloc();      // kernel page table
  ⋮
seginit();       // set up segments
  ⋮
pinit();         // process table
  ⋮

  ⋮
kinit2(P2V(4*1024*1024), P2V(PHYSTOP)); // must co
userinit();      // first user process
mpmain();
```

Auxiliary context

- The primary processor begins its C code in **main()**.
- The auxiliary processors begins their C code in **mpenter()**.
- The state on entering either **main()** or **mpenter()** is the same.
- There is a separate stack of each processor.

```
static void mpenter(void) {
 switchkvm();
 seginit();
 lapicinit();
 mpmain();
}
```

mycpu()

```
2436 struct cpu* mycpu(void) {
      int apicid, i;

      if (readeflags()&FL_IF)
       panic("mycpu␣called␣with␣interrupts␣enabled\n");

      apicid = lapicid();
      for (i = 0; i < ncpu; ++i) {
       if (cpus[i].apicid == apicid)
        return &cpus[i];
      }
      panic("unknown␣apicid\n");
     }
```

mpmain()

```
static void mpmain(void) {
 cprintf("cpu%d: starting %d\n", cpuid(), cpuid());
 idtinit(); // load idt register
 xchg(&(mycpu()->started), 1); // tell startothers()
 scheduler(); // start running processes
}
```

myproc()

```
struct proc *myproc(void) {
  struct cpu *c;
  struct proc *p;
  pushcli();
  c = mycpu();
  p = c->proc;
  popcli();
  return p;
}
```

scheduler

```
2758 void scheduler(void) {
 struct proc *p;
 struct cpu *c = mycpu();
 c->proc = 0;
 for(;;) { sti();
  acquire(&ptable.lock);
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
   if (p->state != RUNNABLE) continue;

   c->proc = p;
   switchuvm(p);
   p->state = RUNNING;
   swtch(&c->scheduler, p->context);
   switchkvm();

   c->proc = 0;
  }
  release(&ptable.lock);
 }
```

scheduler() operation

- For each proc struct p with state RUNNABLE the following is executed:
  - **c**$-$>**proc = p**;
  - **switchuvm()**.
  - **swtch()**.
  - **switchkvm()**.
  - **c**$-$>**proc=NULL**.

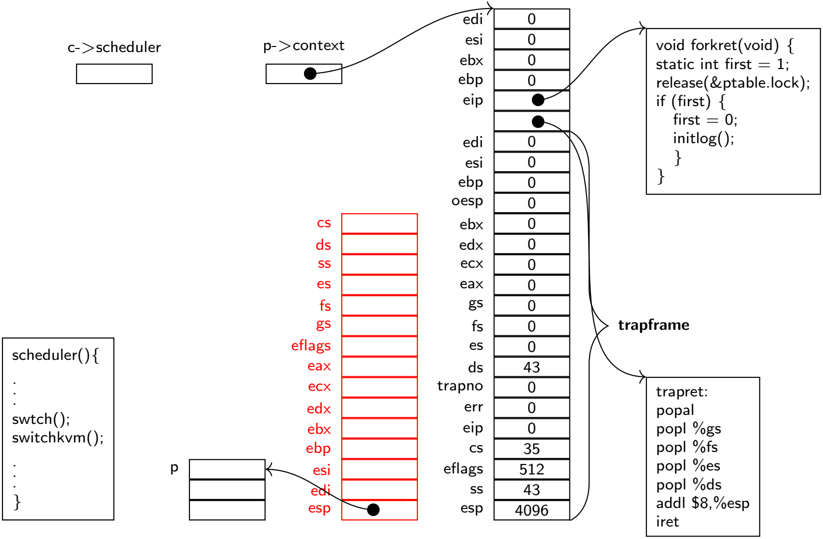
swtch()

```
.globl swtch
swtch:
 movl 4(%esp), %eax
 movl 8(%esp), %edx

 pushl %ebp
 pushl %ebx
 pushl %esi
 pushl %edi

 movl %esp, (%eax)
 movl %edx, %esp
```

```
  popl %edi
  popl %esi
  popl %ebx
  popl %ebp
  ret
```

The `eip` field of `context` is generated by the instruction calling `swtch`.

swtch() operation



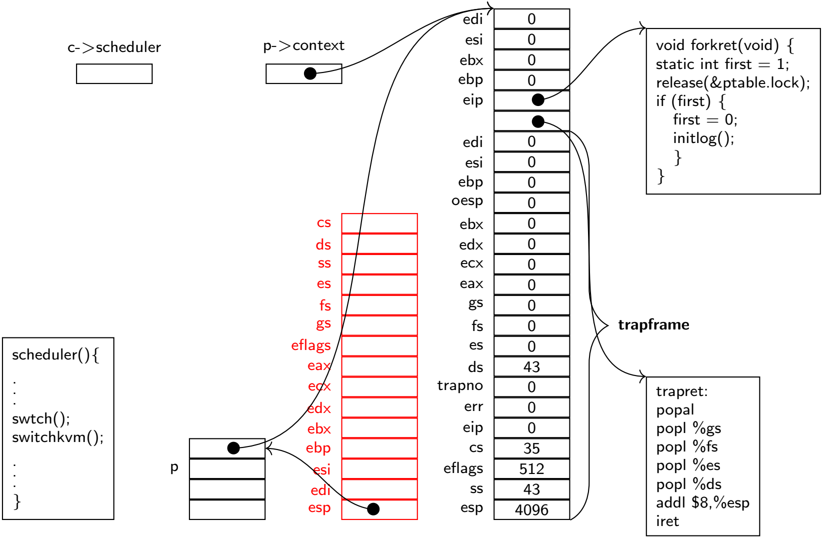
edi	0
esi	0
ebx	0
ebp	0
eip	●
	●
edi	0
esi	0
ebp	0
oesp	0
ebx	0
edx	0
ecx	0
eax	0
gs	0
fs	0
es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

swtch() operation



```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

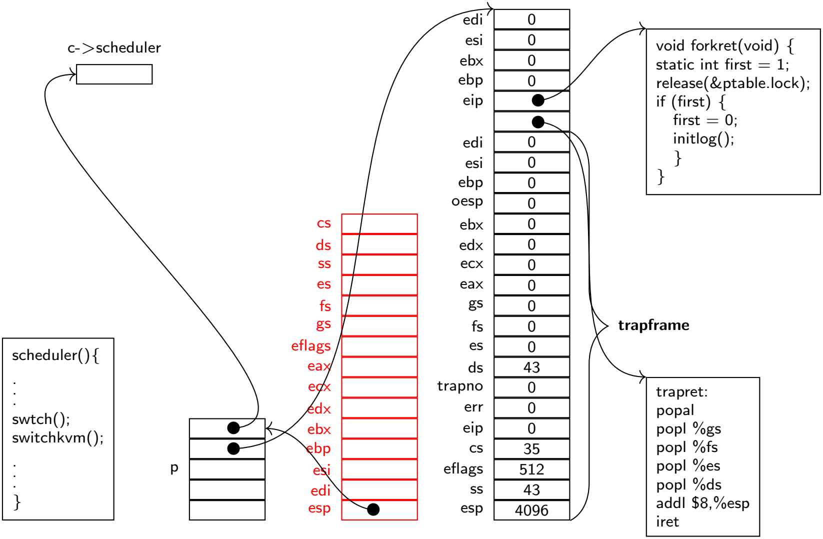
cs	
ds	
ss	
es	
fs	
gs	
eflags	
eax	
ecx	
edx	
ebx	
ebp	
esi	
edi	
esp	●

edi	0
esi	0
ebx	0
ebp	0
eip	●
	●
edi	0
esi	0
ebp	0
oesp	0
ebx	0
edx	0
ecx	0
eax	0
gs	0
fs	0
es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation



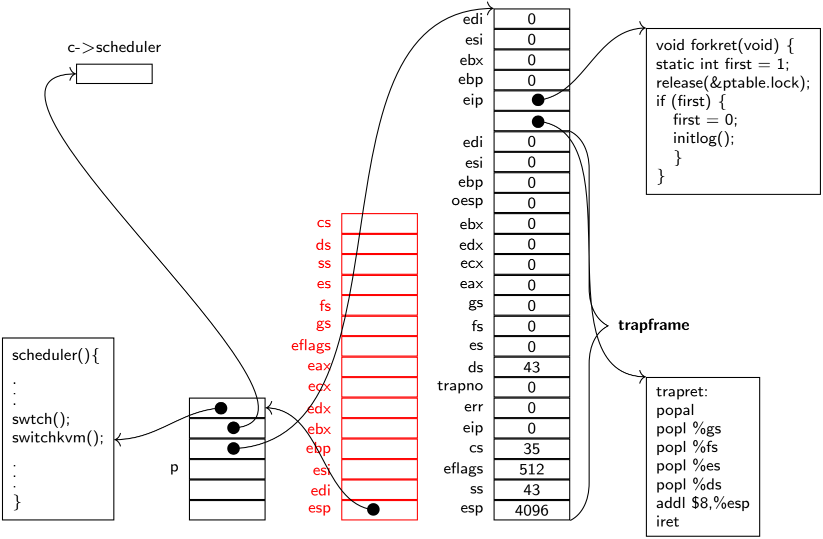
```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

edi	0
esi	0
ebx	0
ebp	0
eip	●
	●
edi	0
esi	0
ebp	0
oesp	0
ebx	0
edx	0
ecx	0
eax	0
gs	0
fs	0
es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation



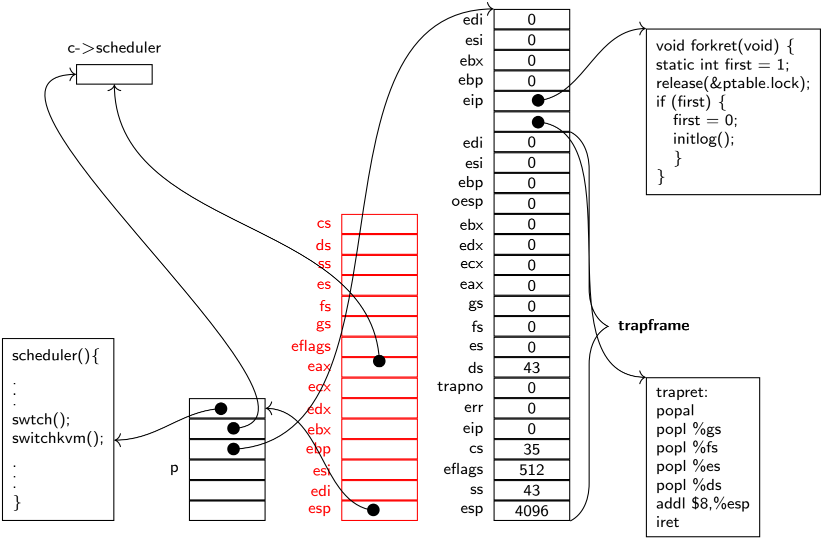
```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

edi	0
esi	0
ebx	0
ebp	0
eip	●
	●
edi	0
esi	0
ebp	0
oesp	0
ebx	0
edx	0
ecx	0
eax	0
gs	0
fs	0
es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation



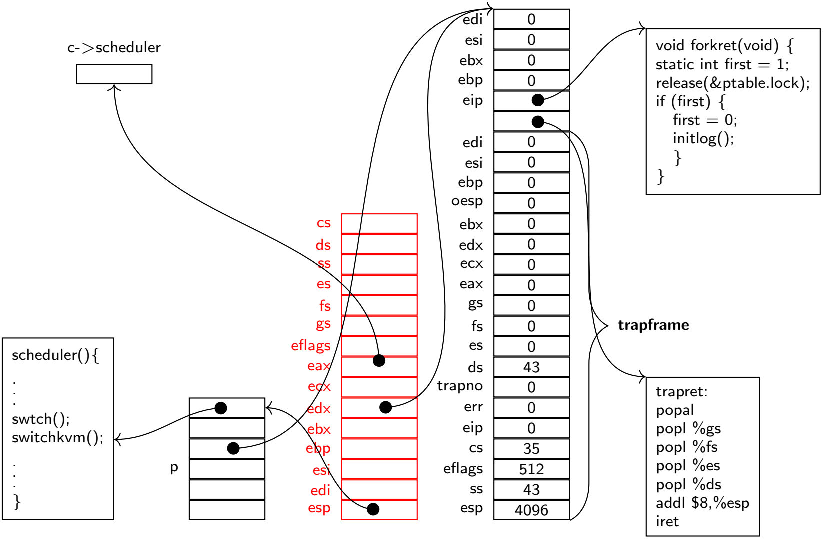
```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

edi	0
esi	0
ebx	0
ebp	0
eip	●
	●
edi	0
esi	0
ebp	0
oesp	0
ebx	0
edx	0
ecx	0
eax	0
gs	0
fs	0
es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation



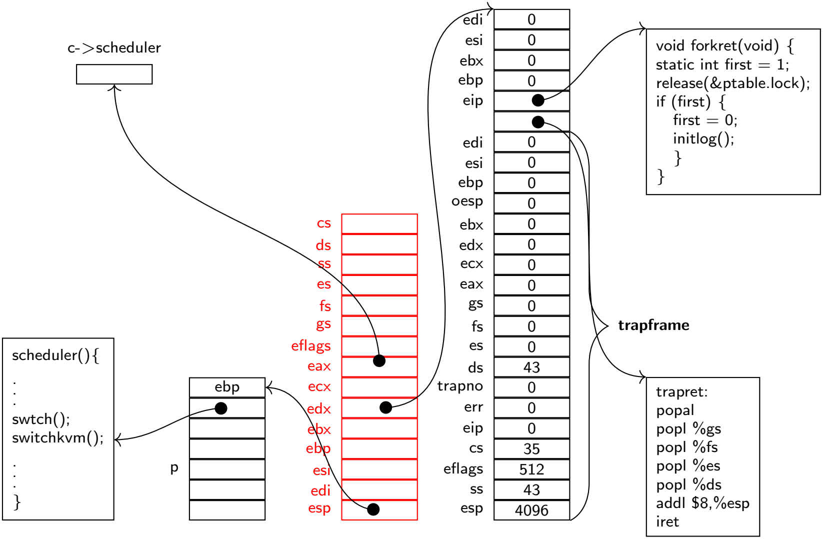
```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

edi	0
esi	0
ebx	0
ebp	0
eip	
edi	0
esi	0
ebp	0
oesp	0
ebx	0
edx	0
ecx	0
eax	0
gs	0
fs	0
es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation



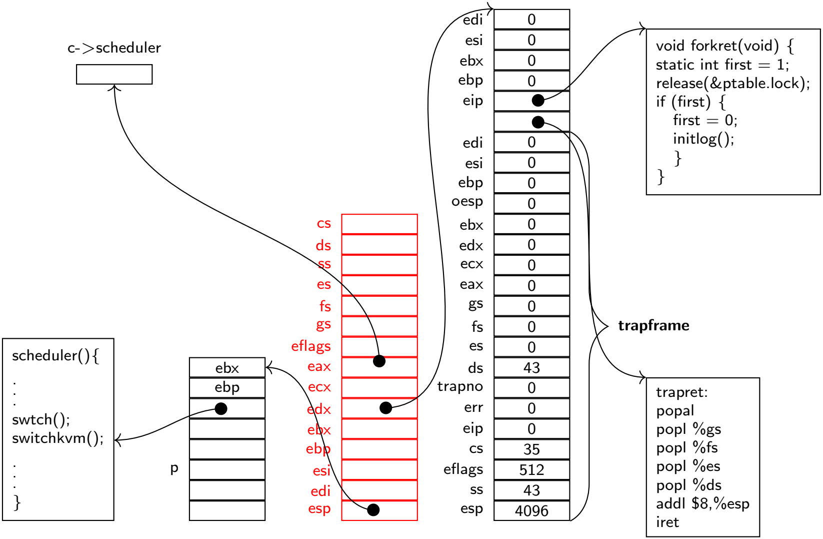
```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

edi	0
esi	0
ebx	0
ebp	0
eip	●
	●
edi	0
esi	0
ebp	0
oesp	0
ebx	0
edx	0
ecx	0
eax	0
gs	0
fs	0
es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation



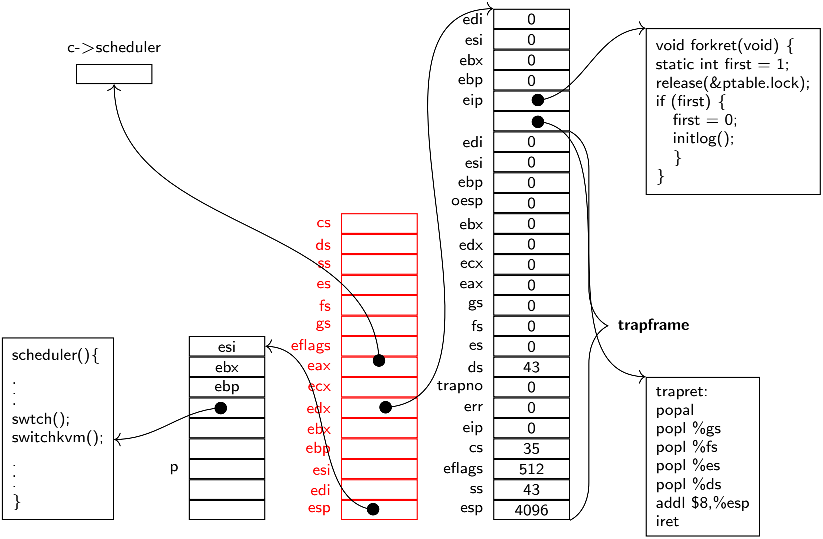
```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

edi	0
esi	0
ebx	0
ebp	0
eip	●
	●
edi	0
esi	0
ebp	0
oesp	0
ebx	0
edx	0
ecx	0
eax	0
gs	0
fs	0
es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation

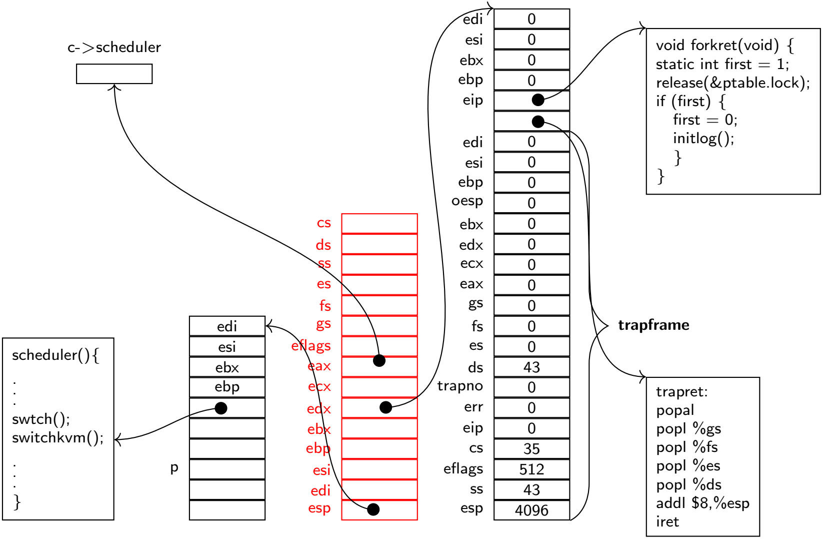


```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation

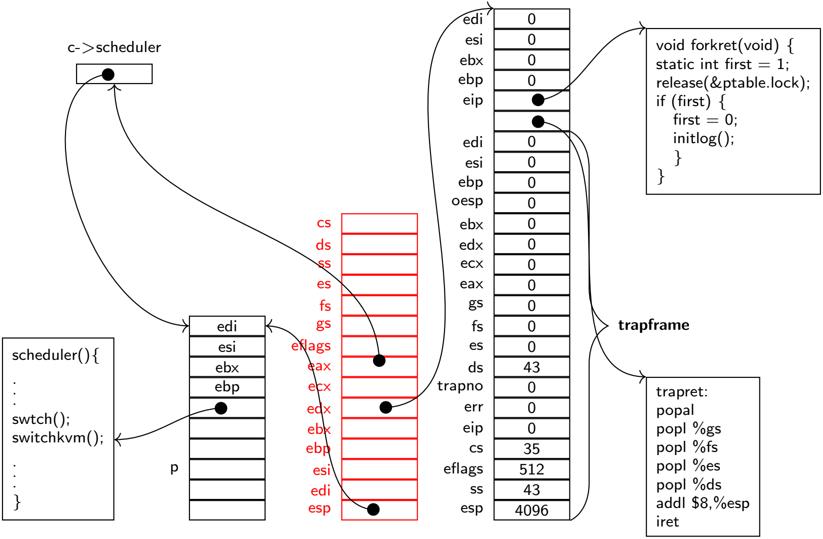


```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation



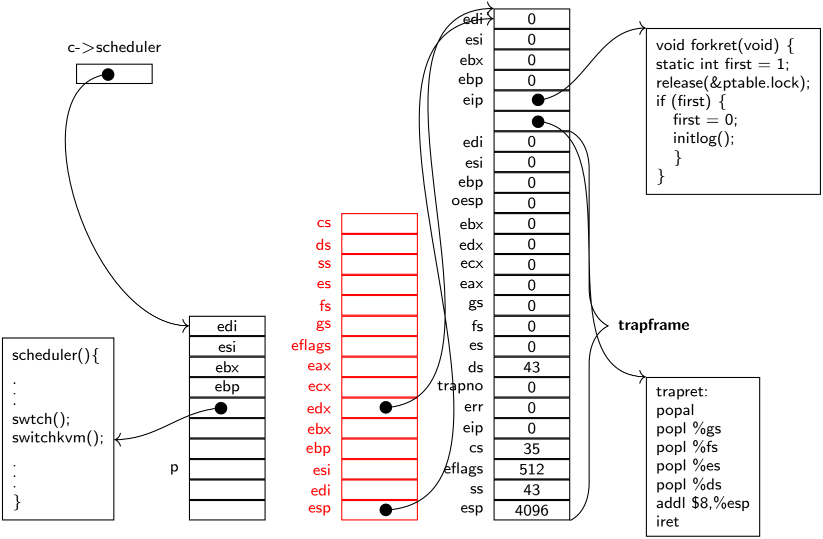
```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

edi	0
esi	0
ebx	0
ebp	0
eip	●
	●
edi	0
esi	0
ebp	0
oesp	0
ebx	0
edx	0
ecx	0
eax	0
gs	0
fs	0
es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation



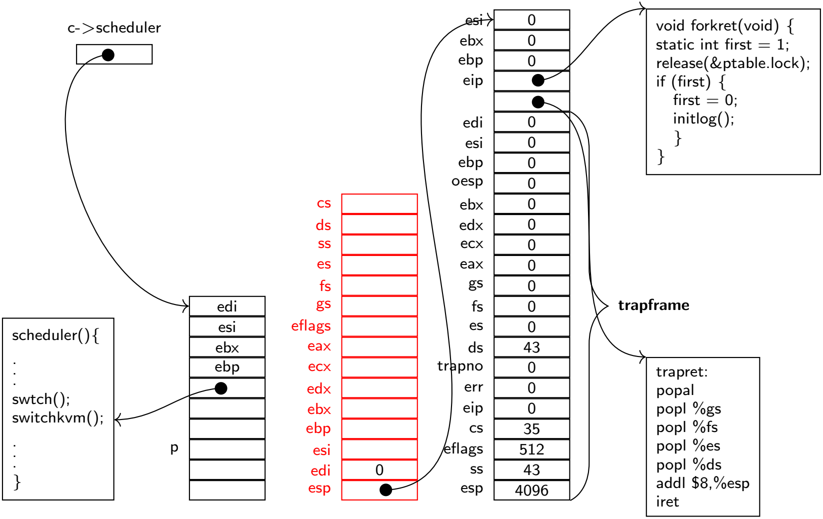
```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

edi	0
esi	0
ebx	0
ebp	0
eip	●
	●
edi	0
esi	0
ebp	0
oesp	0
ebx	0
edx	0
ecx	0
eax	0
gs	0
fs	0
es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation



```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

	edi
	esi
	ebx
	ebp
	●
p	

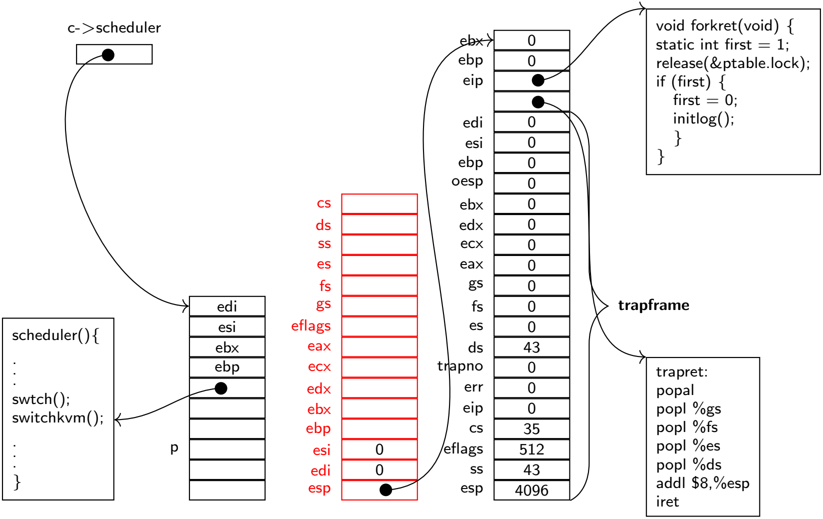
cs	
ds	
ss	
es	
fs	
gs	
eflags	
eax	
ecx	
edx	
ebx	
ebp	
esi	
edi	0
esp	●

esi	0
ebx	0
ebp	0
eip	●
	●
edi	0
esi	0
ebp	0
oesp	0
ebx	0
edx	0
ecx	0
eax	0
gs	0
fs	0
es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation

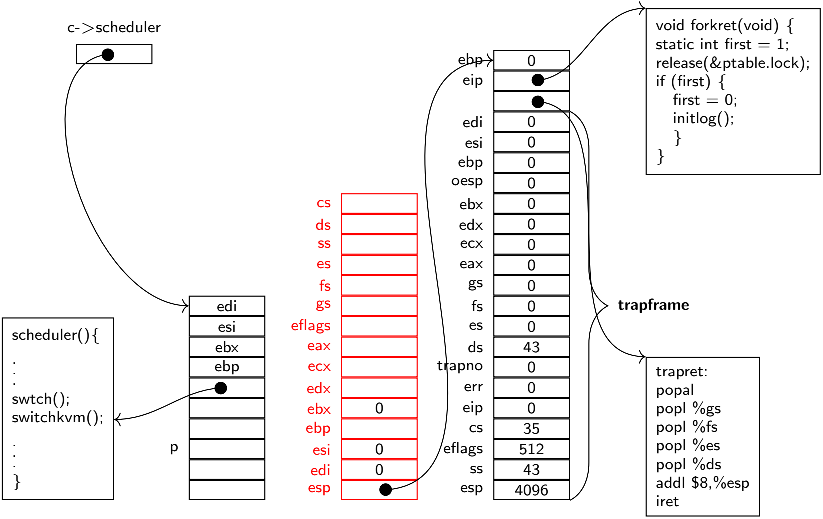


```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation



```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

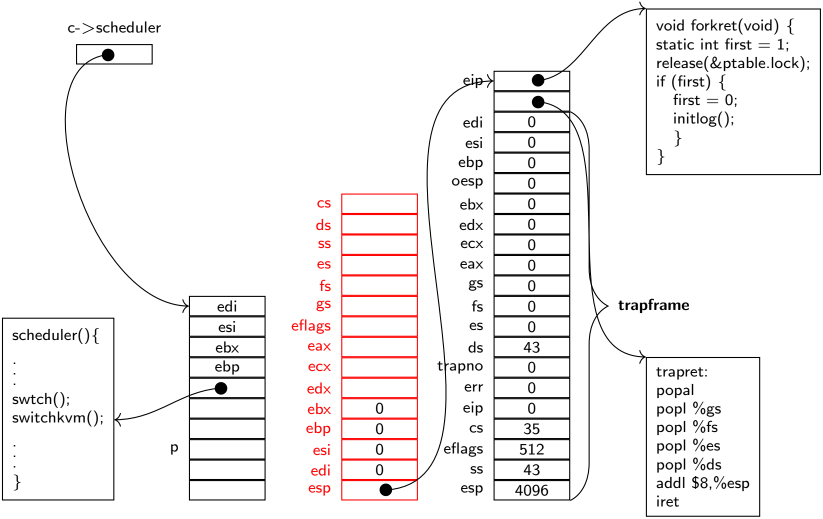
cs	
ds	
ss	
es	
fs	
gs	
eflags	
eax	
ecx	
edx	
ebx	0
ebp	
esi	0
edi	0
esp	●

ebp	0
eip	●
	●
edi	0
esi	0
ebp	0
oesp	0
ebx	0
edx	0
ecx	0
eax	0
gs	0
fs	0
es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation



```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

	edi
	esi
	ebx
	ebp
	●
p	

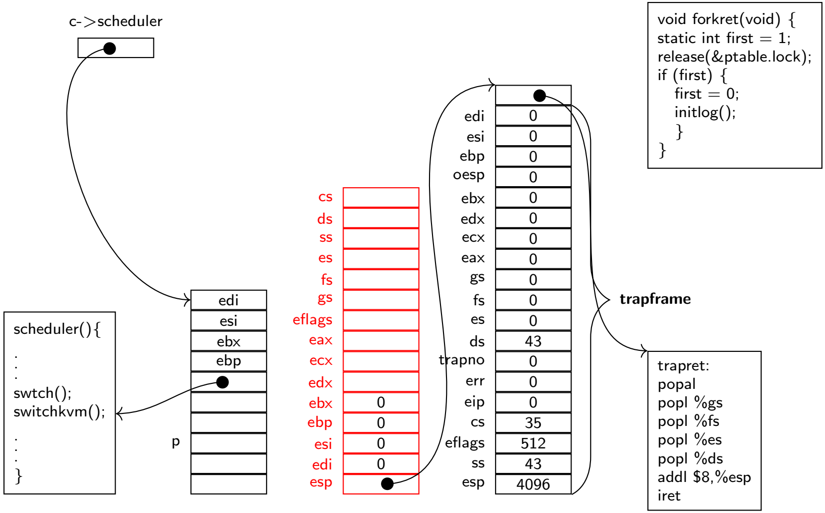
cs	
ds	
ss	
es	
fs	
gs	
eflags	
eax	
ecx	
edx	
ebx	0
ebp	0
esi	0
edi	0
esp	●

eip	●
	●
edi	0
esi	0
ebp	0
oesp	0
ebx	0
edx	0
ecx	0
eax	0
gs	0
fs	0
es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation



```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

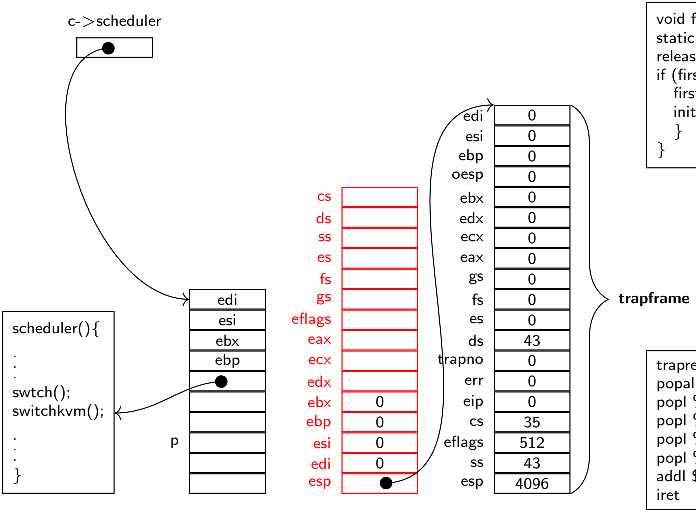
cs	
ds	
ss	
es	
fs	
gs	
eflags	
eax	
ecx	
edx	
ebx	0
ebp	0
esi	0
edi	0
esp	●

	●
edi	0
esi	0
ebp	0
oesp	0
ebx	0
edx	0
ecx	0
eax	0
gs	0
fs	0
es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation



c->scheduler

```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

edi
esi
ebx
ebp
●
p

cs	
ds	
ss	
es	
fs	
gs	
eflags	
eax	
ecx	
edx	
ebx	0
ebp	0
esi	0
edi	0
esp	●

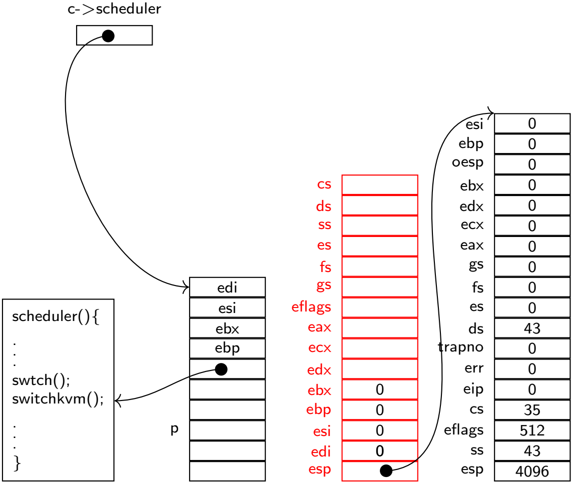
edi	0
esi	0
ebp	0
oesp	0
ebx	0
edx	0
ecx	0
eax	0
gs	0
fs	0
es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

trapframe

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation



```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

edi
esi
ebx
ebp
●

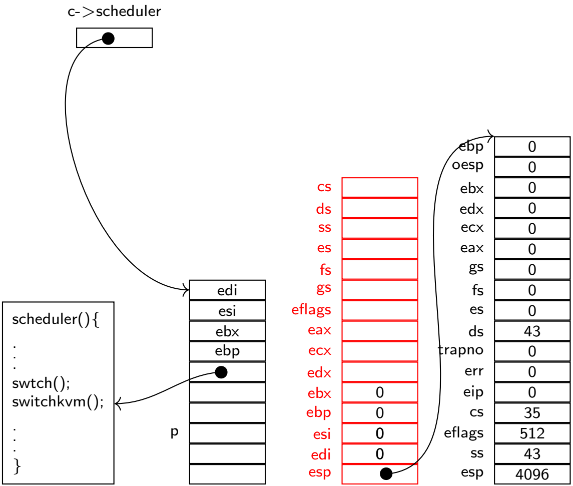
cs	
ds	
ss	
es	
fs	
gs	
eflags	
eax	
ecx	
edx	
ebx	0
ebp	0
esi	0
edi	0
esp	●

esi	0
ebp	0
oesp	0
ebx	0
edx	0
ecx	0
eax	0
gs	0
fs	0
es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation



c->scheduler

```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

edi
esi
ebx
ebp
●
p

cs	
ds	
ss	
es	
fs	
gs	
eflags	
eax	
ecx	
edx	
ebx	0
ebp	0
esi	0
edi	0
esp	●

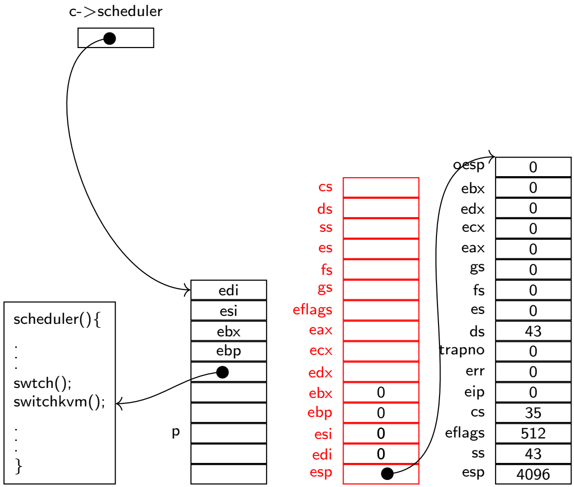
ebp	0
oesp	0
ebx	0
edx	0
ecx	0
eax	0
gs	0
fs	0
es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```



c->scheduler

```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

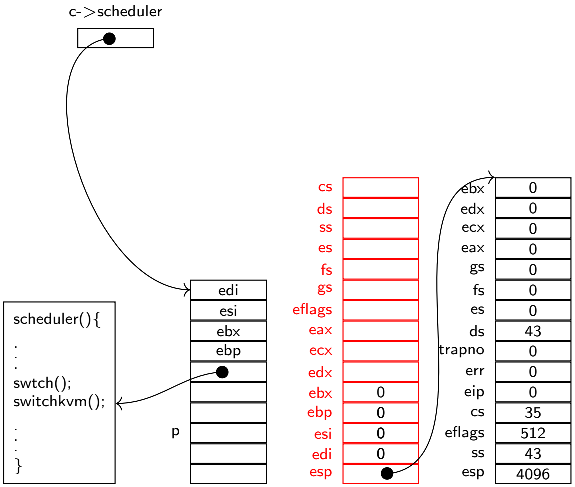
edi
esi
ebx
ebp
●
p

cs	
ds	
ss	
es	
fs	
gs	
eflags	
eax	
ecx	
edx	
ebx	0
ebp	0
esi	0
edi	0
esp	●

oesp	0
ebx	0
edx	0
ecx	0
eax	0
gs	0
fs	0
es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation



```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

	edi
	esi
	ebx
	ebp
	●
p	

cs	
ds	
ss	
es	
fs	
gs	
eflags	
eax	
ecx	
edx	
ebx	0
ebp	0
esi	0
edi	0
esp	●

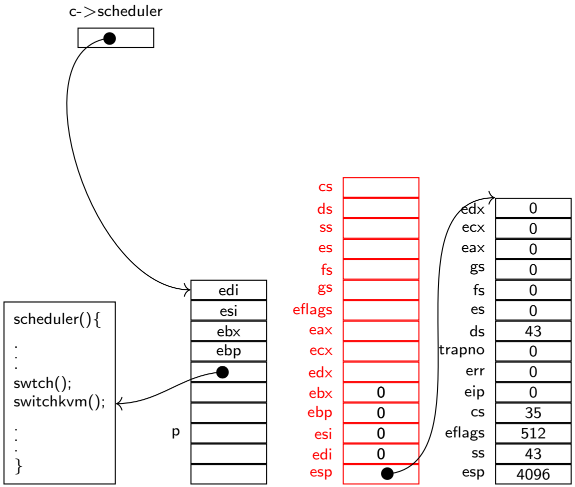
ebx	0
edx	0
ecx	0
eax	0
gs	0
fs	0
es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```



```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

edi
esi
ebx
ebp
●

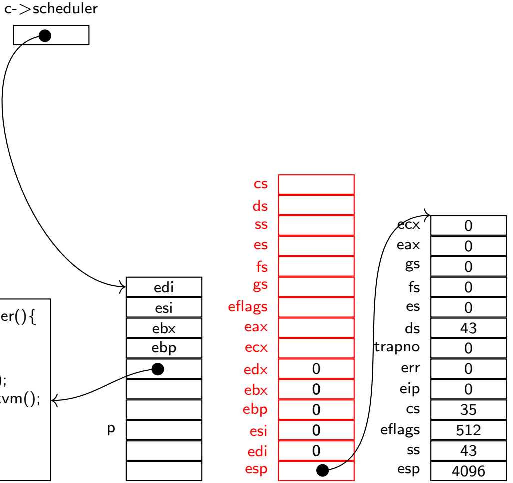
cs	
ds	
ss	
es	
fs	
gs	
eflags	
eax	
ecx	
edx	
ebx	0
ebp	0
esi	0
edi	0
esp	●

edx	0
ecx	0
eax	0
gs	0
fs	0
es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

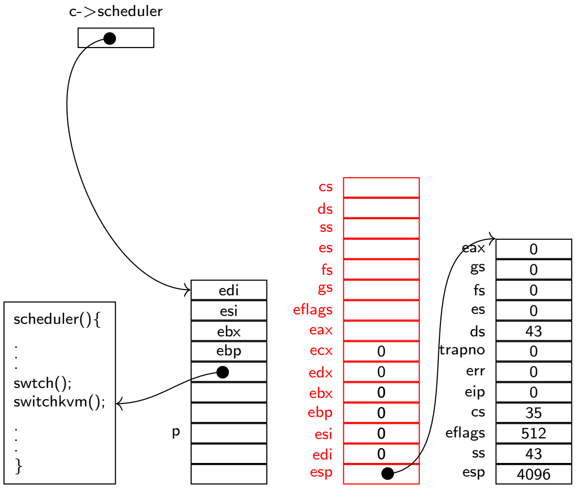


```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```



```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

edi
esi
ebx
ebp
●

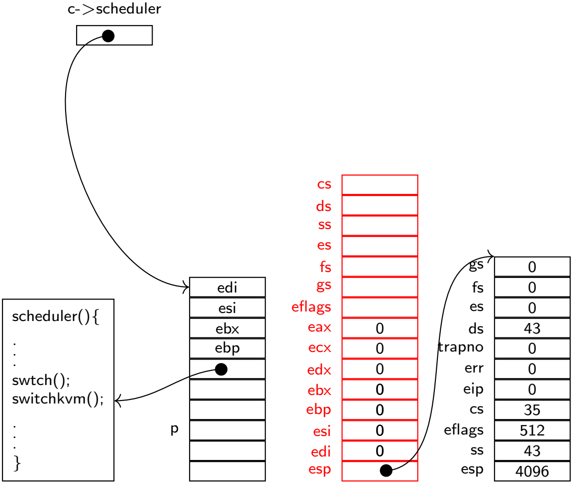
cs	
ds	
ss	
es	
fs	
gs	
eflags	
eax	
ecx	0
edx	0
ebx	0
ebp	0
esi	0
edi	0
esp	●

eax	0
gs	0
fs	0
es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```



```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

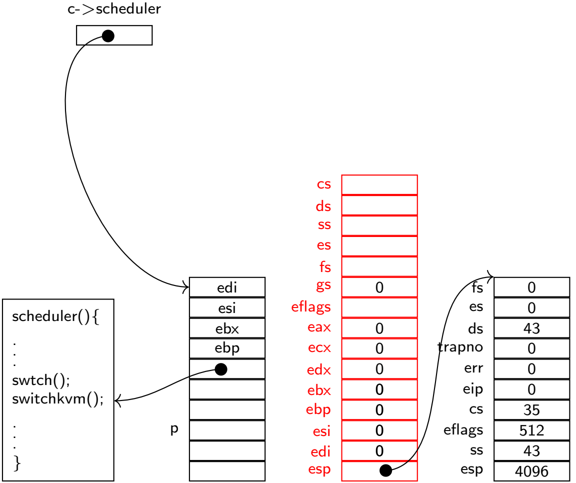
cs	
ds	
ss	
es	
fs	
gs	
eflags	
eax	0
ecx	0
edx	0
ebx	0
ebp	0
esi	0
edi	0
esp	●

gs	0
fs	0
es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```



```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

edi
esi
ebx
ebp
●

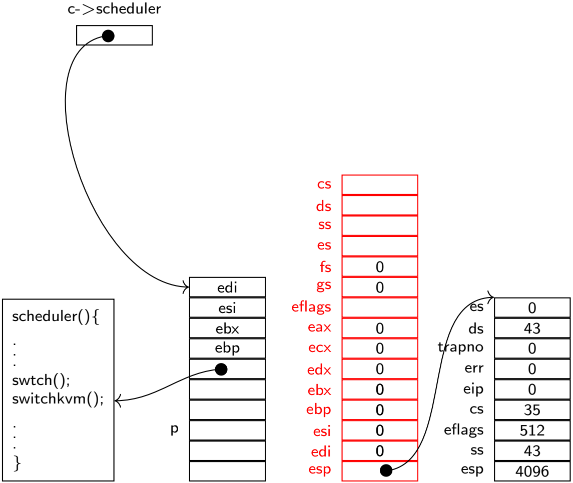
cs	
ds	
ss	
es	
fs	
gs	0
eflags	
eax	0
ecx	0
edx	0
ebx	0
ebp	0
esi	0
edi	0
esp	●

fs	0
es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```



```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

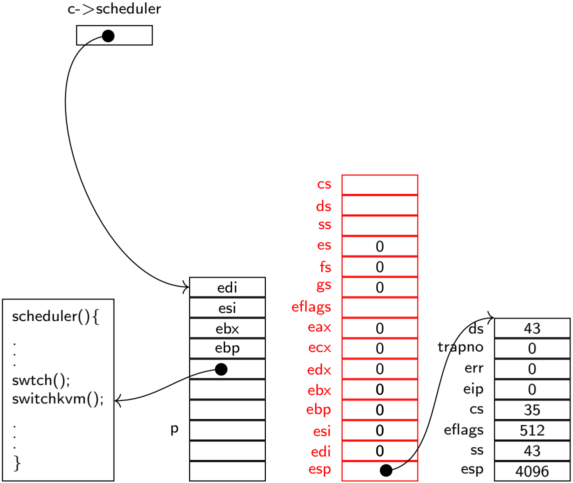
edi
esi
ebx
ebp
●

cs	
ds	
ss	
es	
fs	0
gs	0
eflags	
eax	0
ecx	0
edx	0
ebx	0
ebp	0
esi	0
edi	0
esp	●

es	0
ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation



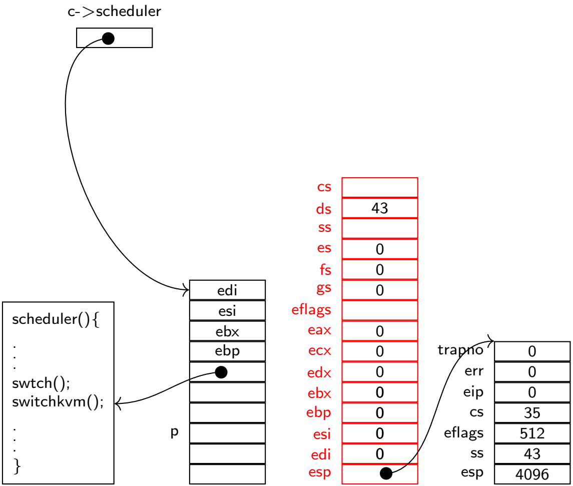
cs	
ds	
ss	
es	0
fs	0
gs	0
eflags	
eax	0
ecx	0
edx	0
ebx	0
ebp	0
esi	0
edi	0
esp	●

ds	43
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation



```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

cs	
ds	43
ss	
es	0
fs	0
gs	0
eflags	
eax	0
ecx	0
edx	0
ebx	0
ebp	0
esi	0
edi	0
esp	●

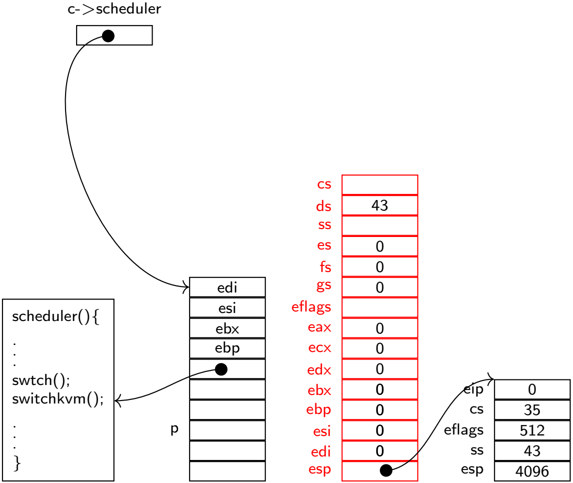
trapno	0
err	0
eip	0
cs	35
eflags	512
ss	43
esp	4096

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

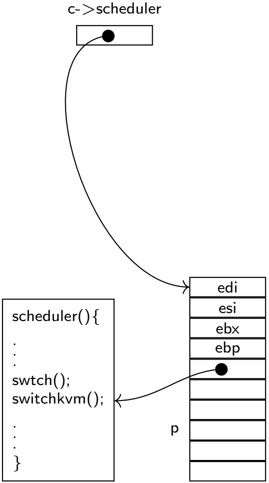


```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

swtch() operation

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```



```
scheduler(){
.
.
.
swtch();
switchkvm();
.
.
.
}
```

cs	35
ds	43
ss	43
es	0
fs	0
gs	0
eflags	512
eax	0
ecx	0
edx	0
ebx	0
ebp	0
esi	0
edi	0
esp	4096

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

Context switch

- Calling **swtch()**:
  - Creates a **context** structure on the current stack.
  - Stores the **cotext** structure address created in the first argument.
  - Load the **context** structure pointed to by the second argument.
- We are switching KERNEL contexts.
- User mode context of a process is loaded by the kernel side of the process.

Kernel context seems too small

- Where are **eax**, **ecx**, **edx**????

Kernel context seems too small

- Where are **eax**, **ecx**, **edx**????
  - The **gcc** calling conventions deals with them!

Kernel context seems too small

- Where are **eax**, **ecx**, **edx**????
  - The **gcc** calling conventions deals with them!
- Where are **cs**, **ds**, and **ss**????

Kernel context seems too small

- Where are **eax**, **ecx**, **edx**????
  - The **gcc** calling conventions deals with them!
- Where are **cs**, **ds**, and **ss**????
  - The base and limit fields are identical across all kernel sides.
  - So, they need to be loaded only on aech kernel entering.

Kernel context seems too small

- Where are **eax**, **ecx**, **edx**????
  - The **gcc** calling conventions deals with them!
- Where are **cs**, **ds**, and **ss**????
  - The base and limit fields are identical across all kernel sides.
  - So, they need to be loaded only on aech kernel entering.
- Where is **gdtr**????

Kernel context seems too small

- Where are **eax**, **ecx**, **edx**????
  - The **gcc** calling conventions deals with them!
- Where are **cs**, **ds**, and **ss**????
  - The base and limit fields are identical across all kernel sides.
  - So, they need to be loaded only on aech kernel entering.
- Where is **gdtr**????
  - The address and size fields are different across processors.
  - The base and limit MUST NOT change between kernel sides on the same CPU.
  - Since **gdtr** is privileged, it needs to be loaded ONLY on kernel initialization.

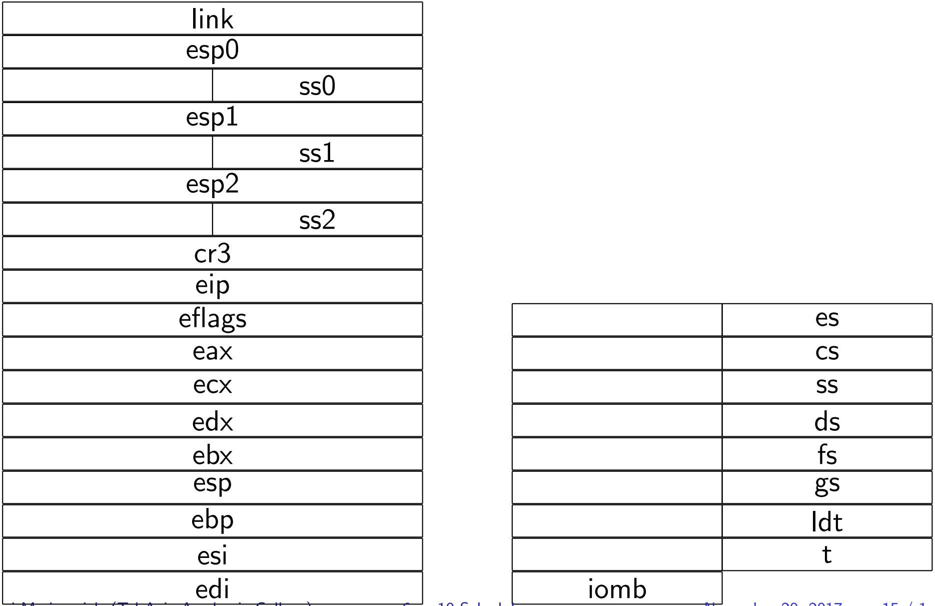
If switching to user mode is expected:

- The `tr` register should contain the index of a TSS descriptor.
- The `TSS` descriptor should point to a `taskstate` structure.
- The `ss0` and `esp0` fields should point to a valid kernel stack top.
- $\boxed{\text{The above is ESSENTIAL for proper interrupt service in user mode.}}$

switchuvm

```
void switchuvm(struct proc *p) {
 pushcli();
 mycpu()->gdt[SEG_TSS] = SEG16(STS_T32A,
                               &mycpu()->ts,
                               sizeof(mycpu()->ts)-1, 0)
 mycpu()->gdt[SEG_TSS].s = 0;
 mycpu()->ts.ss0 = SEG_KDATA<<3;
 mycpu()->ts.esp0 = (uint)p->kstack + KSTACKSIZE;
 mycpu()->ts.iomb = (ushort) 0xFFFF;
 ltr(SEG_TSS << 3);
 if (p->pgdir == 0)
  panic("switchuvm: no pgdir");
 lcr3(v2p(p->pgdir)); // switch to new address space
 popcli();
}
```

taskstate (hardware structure)



link

esp0

ss0

esp1

ss1

esp2

ss2

cr3

eip

eflags

eax

ecx

edx

ebx

esp

ebp

esi

edi

es

cs

ss

ds

fs

gs

ldt

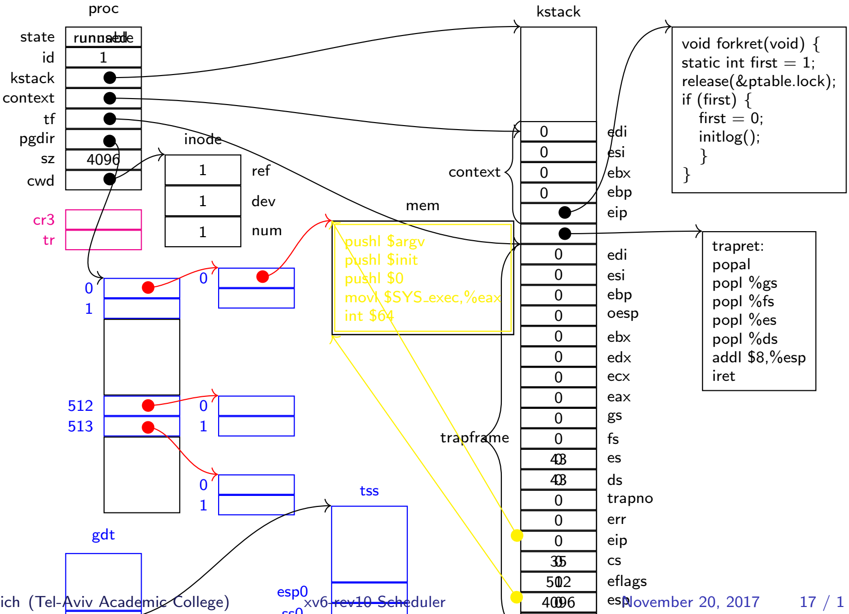
t

iomb

taskstate in C

```
struct taskstate
{
 uint link;
 uint esp0;
 ushort ss0;
 ushort padding1;
 uint *esp1;
 ushort ss1;
 ushort padding2;
 uint *esp2;
 ushort ss2;
 ushort padding3;
 void *cr3;
 uint *eip;
 uint eflags;
 uint eax;
 uint ecx;
 uint edx;
 uint ebx;
 uint *esp;
 uint *ebp;
 uint esi;
 uint edi;
 ushort es;
 ushort padding4;
 ushort cs;
 ushort padding5;
 ushort ss;
 ushort padding6;
 ushort ds;
 ushort padding7;
 ushort fs;
 ushort padding8;
 ushort gs;
 ushort padding9;
 ushort ldt;
 ushort padding10;
 ushort t;
 ushort iomb;
};
```

switchuvm

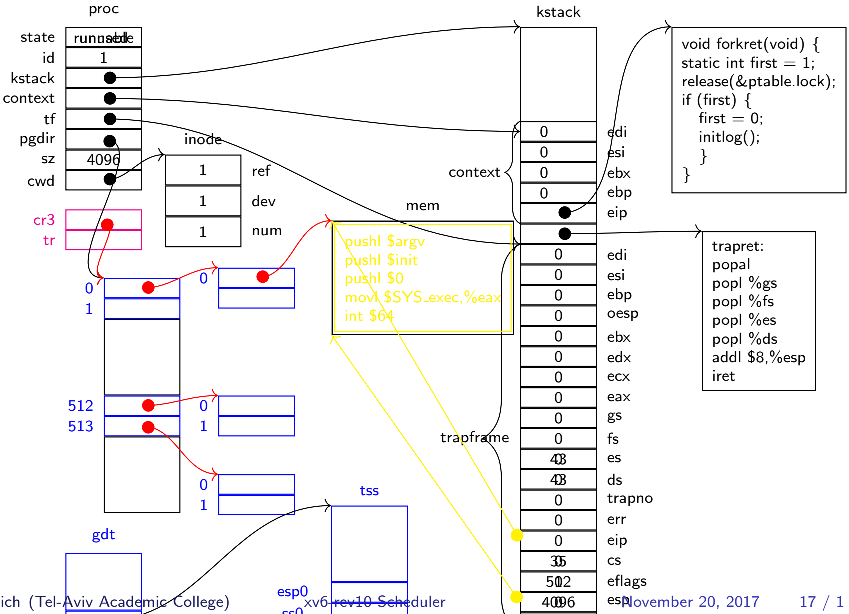


```
pushl $argv
pushl $init
pushl $0
movl $SYS_exec,%eax
int $64
```

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

switchuvm

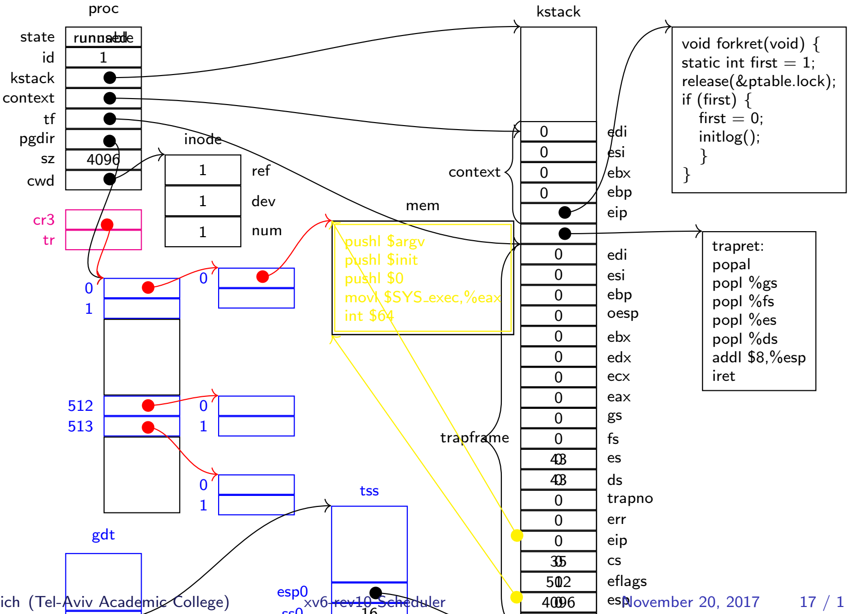


```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

```
pushl $argv
pushl $init
pushl $0
movl $SYS_exec,%eax
int $64
```

switchuvm

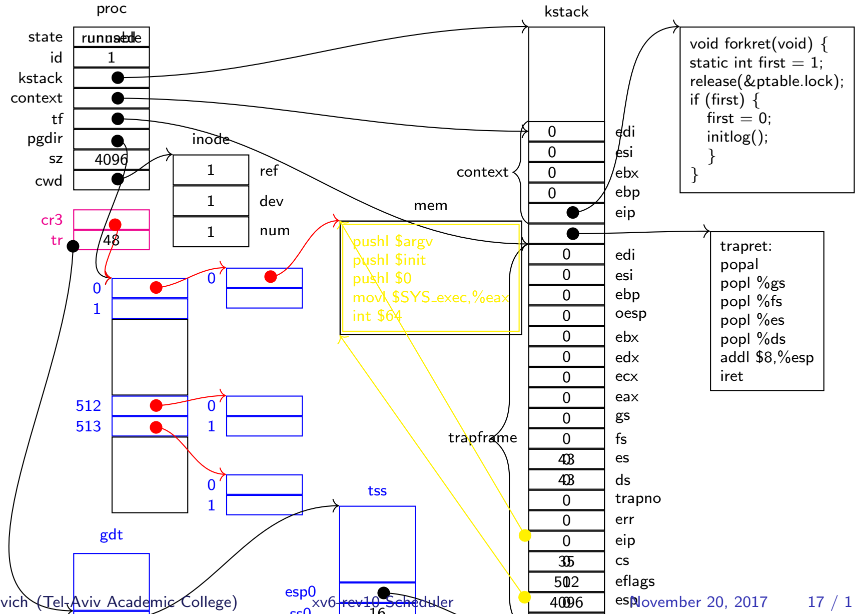


```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

```
pushl $argv
pushl $init
pushl $0
movl $SYS_exec,%eax
int $64
```

switchuvm



```
pushl $argv
pushl $init
pushl $0
movl $SYS_exec,%eax
int $64
```

```
void forkret(void) {
static int first = 1;
release(&ptable.lock);
if (first) {
   first = 0;
   initlog();
   }
}
```

```
trapret:
popal
popl %gs
popl %fs
popl %es
popl %ds
addl $8,%esp
iret
```

co-routines

- The scheduler switches to a process by using:

```
    swtch(&c->scheduler, p->context);
```

- A process leaves the cpu by returning to the scheduler using:

```
    swtch(&p->context, mycpu()->scheduler);
```

- We have here co-routines.

Event driven kernel

- At this point there is no more LINEAR EXECUTION of the kernel.
- The kernel as of now is EVENT DRIVEN.