

xv6©-rev10
(Copyright Frans Kaashoek, Robert Morris, and Russ Cox.)
First Process Creation, II

Carmi Merimovich

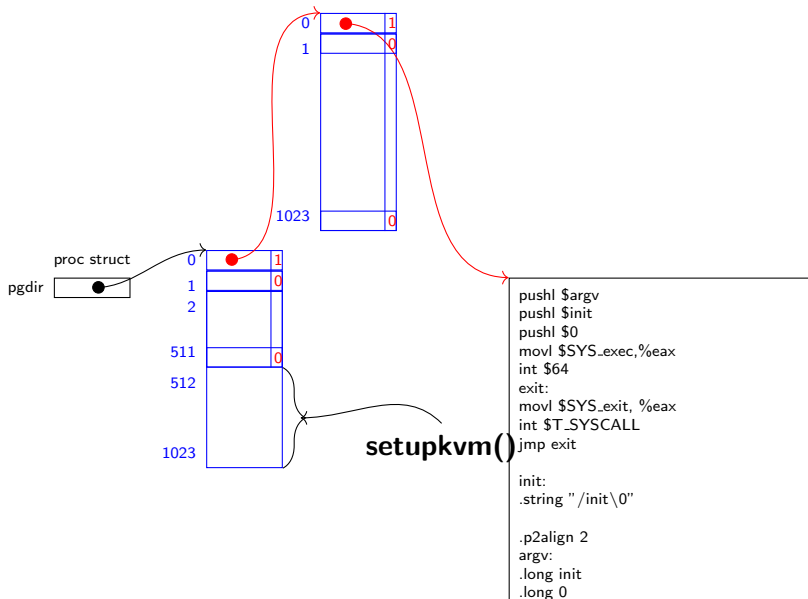
Tel-Aviv Academic College

November 20, 2017

Context

```
kinit1(end, P2V(4*1024*1024)); // phys page allocation
kvmalloc(); // kernel page table
:
segininit(); // set up segments
:
pinit(); // process table
:
:
kinit2(P2V(4*1024*1024), P2V(PHYSTOP)); // must copy
userinit(); // first user process
mpmain();
```

Context in `userinit()`



The Road to User Mode

Initial user mode state for first process

eax	0
ebx	0
ecx	0
edx	0
ebp	0
esi	0
edi	0
esp	4096
eip	0

don't care

cs	27
ds	35
ss	35
es	0
fs	0
gs	0

```
0  pushl $argv
    pushl $init
    pushl $0
    movl $SYS_exec,%eax
    int $64
    exit:
    movl $SYS_exit, %eax
    int $T_SYSCALL
    jmp exit

    init:
    .string "/init\0"

    .p2align 2
    argv:
    .long init
    .long 0

4095
```

The naive approach for the scheduler

```
movw $35,%ax
movw %ax,ds
movw %ax,ss # Interrupts use the stack. We just ruined
movl $4096,%esp
```

```
movl $0,%eax # Much better to move eax to other regs
movl $0,%ecx
movl $0,%edx
movl $0,%ebx
movl $0,%ebp
movl $0,%esi
movl $0,%edi
```

```
movw %ax,%es
movw %ax,%fs
movw %ax,%gs
```

```
ljmp $27,$0
```

The problem

It is not possible to find a reasonable order to load:

- cs, eip.
- ss, esp.
- eflags (Containing the Interrupt flag).

What the hardware should provide us with?

- The **iret** instruction fits the bill.

iret instruction in kernel mode

Implied stack operands:

	31		9		2	1	0
esp							
ss	unused			gdt index		0	11
eflags							
cs	unused			gdt index		0	11
esp → eip							

The operands are loaded into the appropriate registers and POPped of the stack.

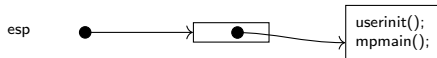
Working Approach for the scheduler!!

```
pushl    $4096
pushl    $35
pushl    $512
pushl    $27
pushl    $0
```

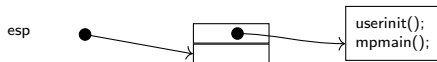
```
iret
```

- **We are not the scheduler!**

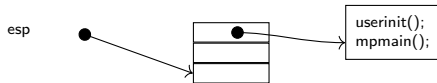
Current stack



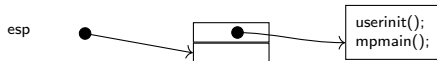
Current stack



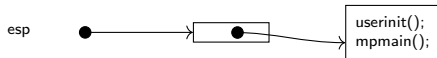
Current stack



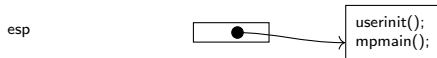
Current stack



Current stack




Current stack

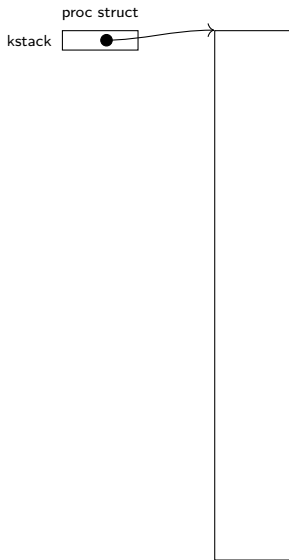


What do we do??

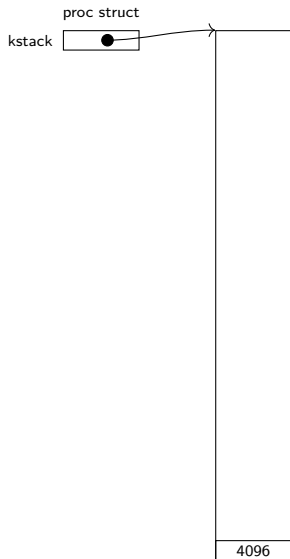
Build kernel stack for the first process

proc struct
kstack 

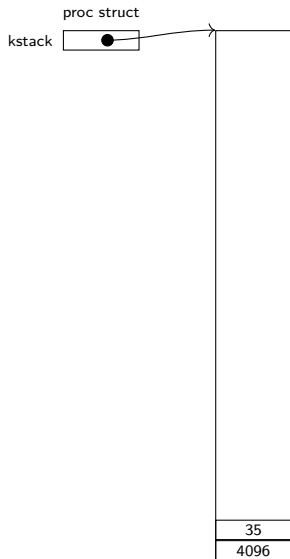
Build kernel stack for the first process



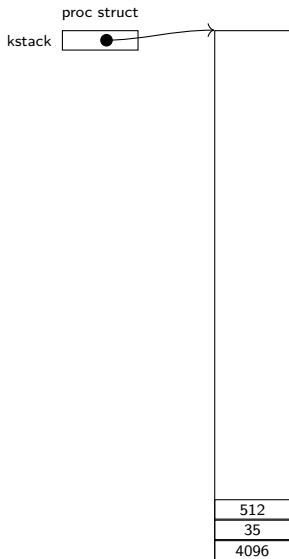
Build kernel stack for the first process



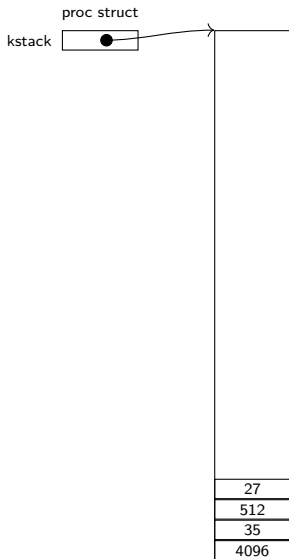
Build kernel stack for the first process



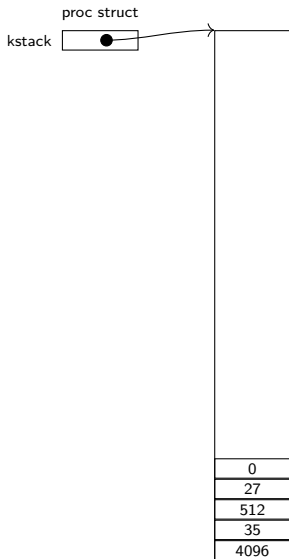
Build kernel stack for the first process



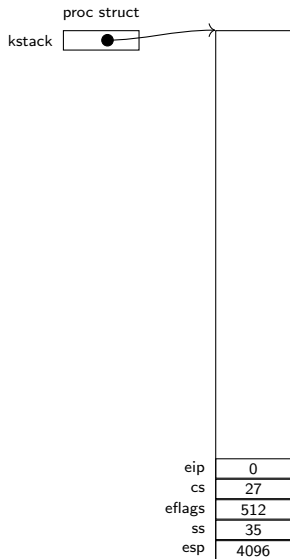
Build kernel stack for the first process



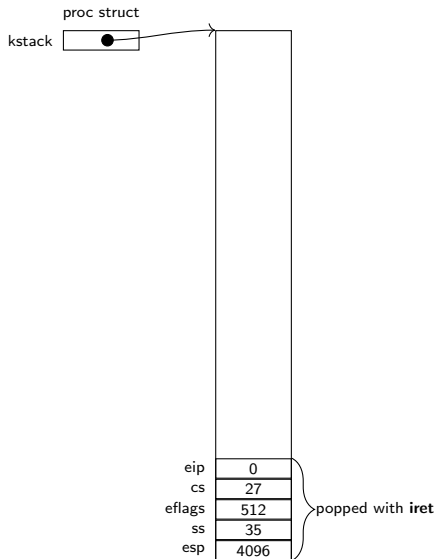
Build kernel stack for the first process



Build kernel stack for the first process



Build kernel stack for the first process

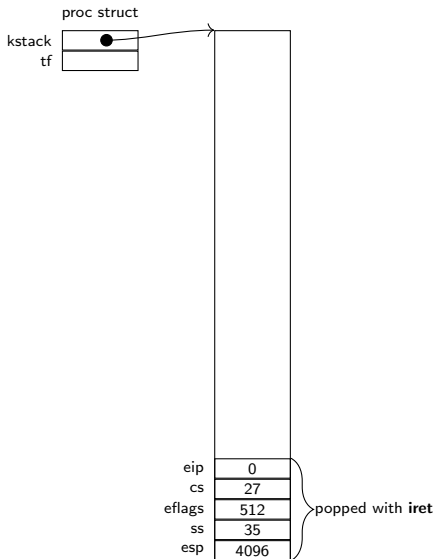


- We expect the scheduler to execute **iret** with its **esp** pointing to the **eip** field.
- We began the discussion with more registers.
- What about those other registers?
- The scheduler should load them with the prescribed values.
 - Ugly. Registers are being setup in two different places.

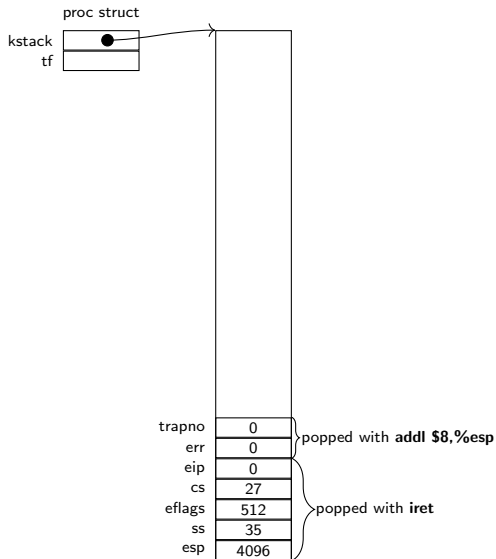
OR

- We will put registers values on stack for all the registers!

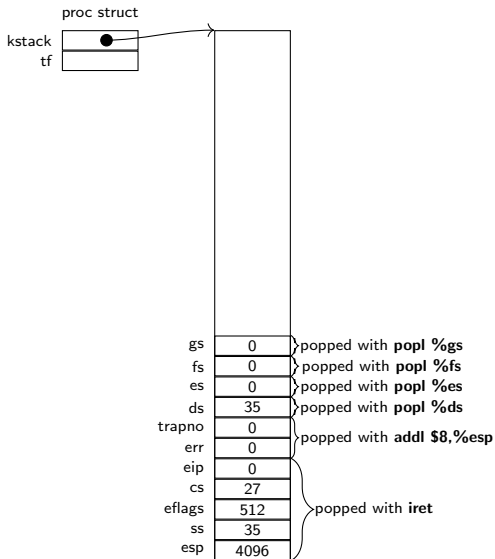
Build kernel stack for first process



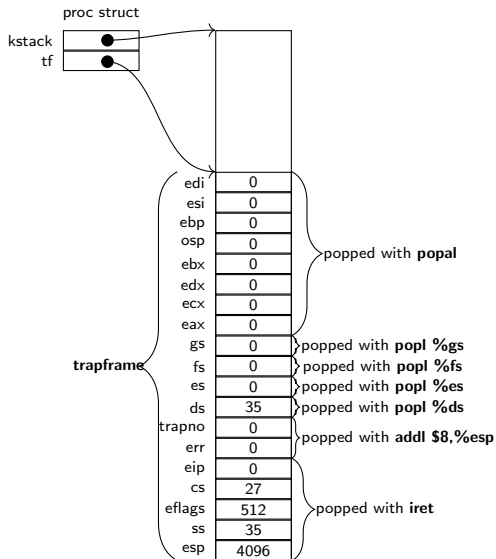
Build kernel stack for first process



Build kernel stack for first process



Build kernel stack for first process



trapframe structure

602

```
struct trapframe {  
    uint edi;  
    uint esi;  
    uint ebp;  
    uint oesp;  
    uint ebx;  
    uint edx;  
    uint ecx;  
    uint eax;  
  
    ushort gs;  
    ushort padding1;  
    ushort fs;  
    ushort padding2;
```

```
    ushort es;  
    ushort padding3;  
    ushort ds;  
    ushort padding4;  
    uint trapno;  
  
    uint err;  
    uint eip;  
    ushort cs;  
    ushort padding5;  
    uint eflags;  
  
    uint esp;  
    ushort ss;  
    ushort padding6;
```

}

Building trapframe example

```
sp      = p->kstack + KSTACKSIZE;  
sp      -= sizeof *p->tf;  
p->tf = (struct trapframe*)sp;  
memset(p->tf, 0, sizeof *p->tf);
```

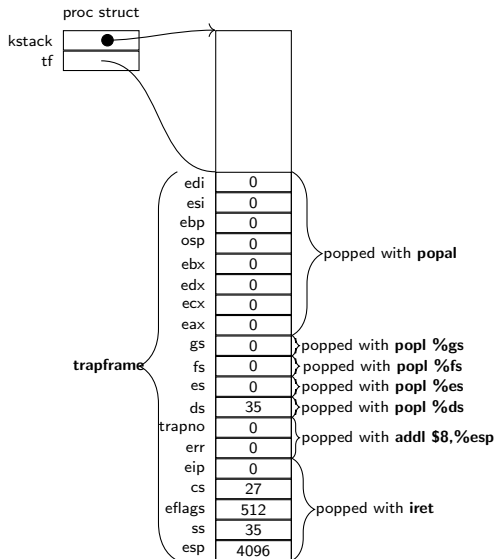
```
p->tf->cs = (SEG_UCODE << 3) | DPL_USER;  
p->tf->ds = (SEG_UDATA << 3) | DPL_USER;  
p->tf->es = p->tf->ds;  
p->tf->ss = p->tf->ds;  
p->tf->eflags = FL_IF;  
p->tf->esp = PGSIZE;  
p->tf->eip = 0; // beginning of initcode.S
```

- So, the code the scheduler is supposed to execute is:

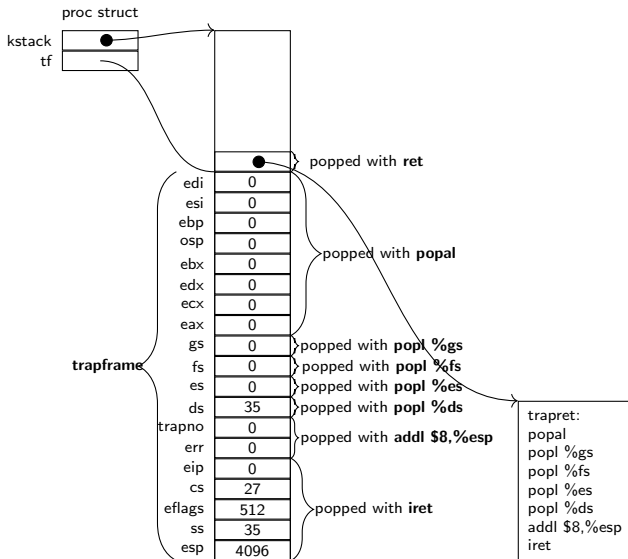
```
popal  
popl %gs  
popl %fs  
popl %es  
popl %ds  
addl $8,%esp  
iret
```

- For the sake of generality, we leave the address of this code on the stack.

Build kernel stack for first process



Build kernel stack for first process



trapret pointer setting

```
sp -= 4;  
*sp = trapret;
```

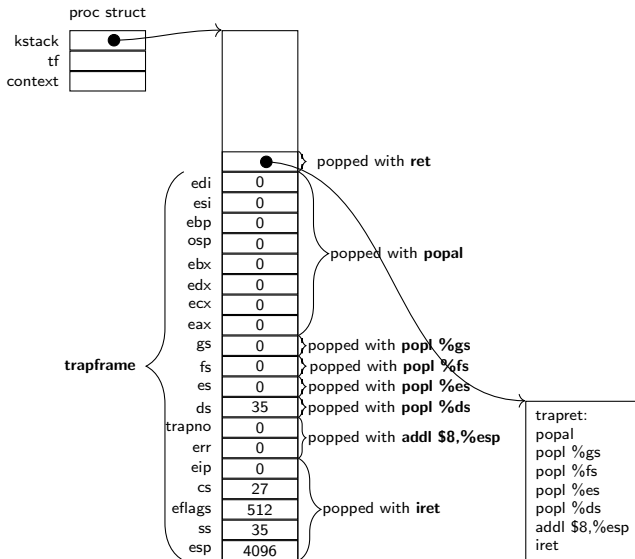
A word of caution

- The **real** reason behind the **trapframe** construction is the interrupt servicing code.
- Wait for the real reasoning until there.

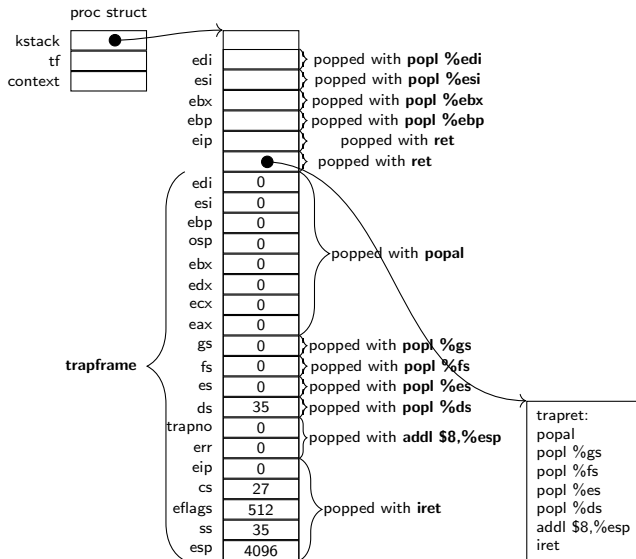
The **context** struct

- The **xv6** machinery is more general than the above.
- The scheduler has a context of its own.
- Using just **trapframe** will cause the scheduler to loose context.
- So, a **context** struct is pushed on the stack in order to preserve context.

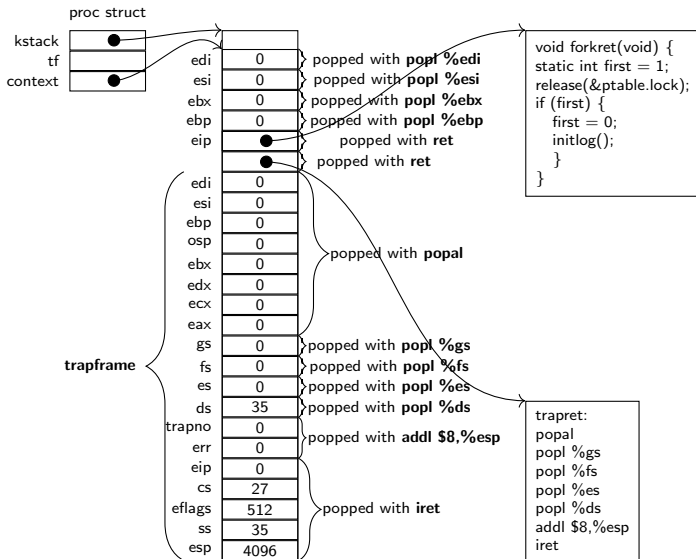
Build kernel stack for the first process



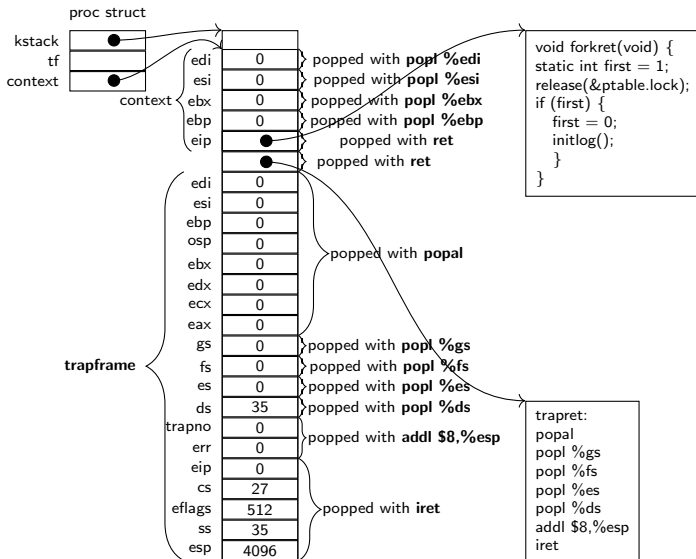
Build kernel stack for the first process



Build kernel stack for the first process



Build kernel stack for the first process



context structure

```
2623 struct context {  
    uint edi;  
    uint esi;  
    uint ebx;  
    uint ebp;  
    uint eip;  
};
```

Bulding the **context** structure

```
sp -= sizeof *p->context;  
p->context = sp;  
memset(p->context, 0, sizeof(*p->context));  
p->context->eip = forkret;
```

The whole first process creation

proc	
state	unused
id	
kstack	
context	
tf	
pgdir	
sz	
cwd	

```
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
```

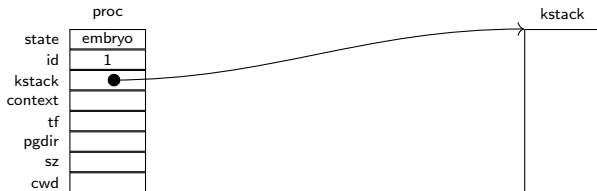
The whole first process creation

proc	
state	embryo
id	1
kstack	
context	
tf	
pgdir	
sz	
cwd	

```
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
```

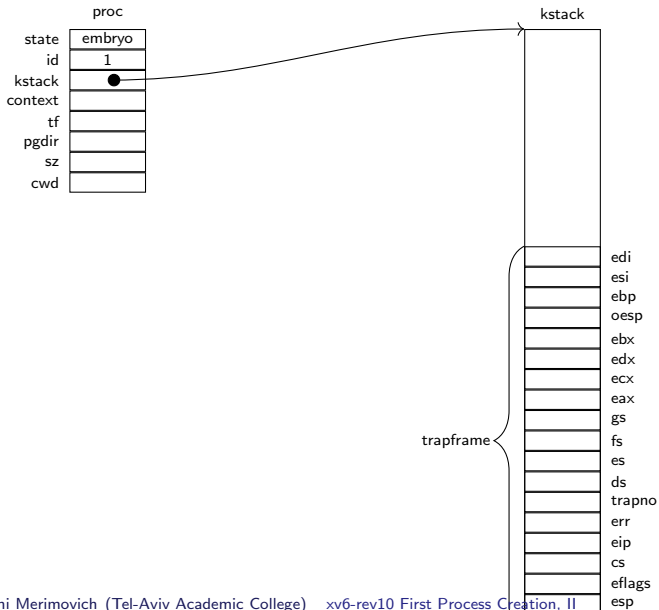
The whole first process creation



```
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
```

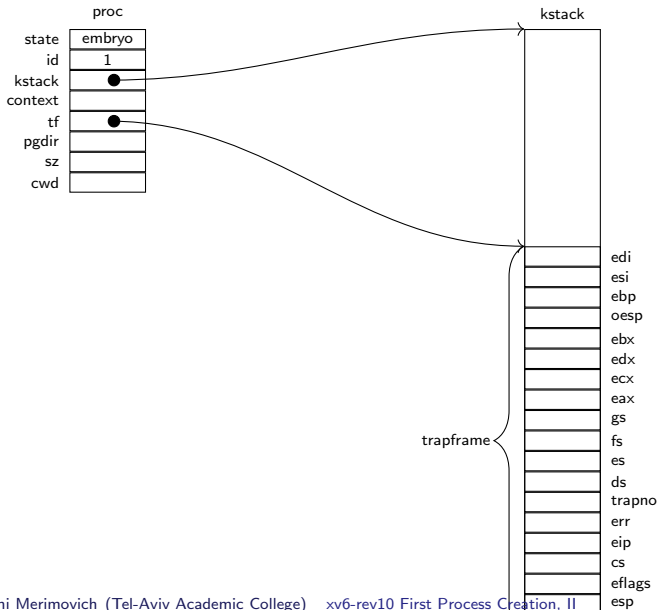
The whole first process creation



```
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
```

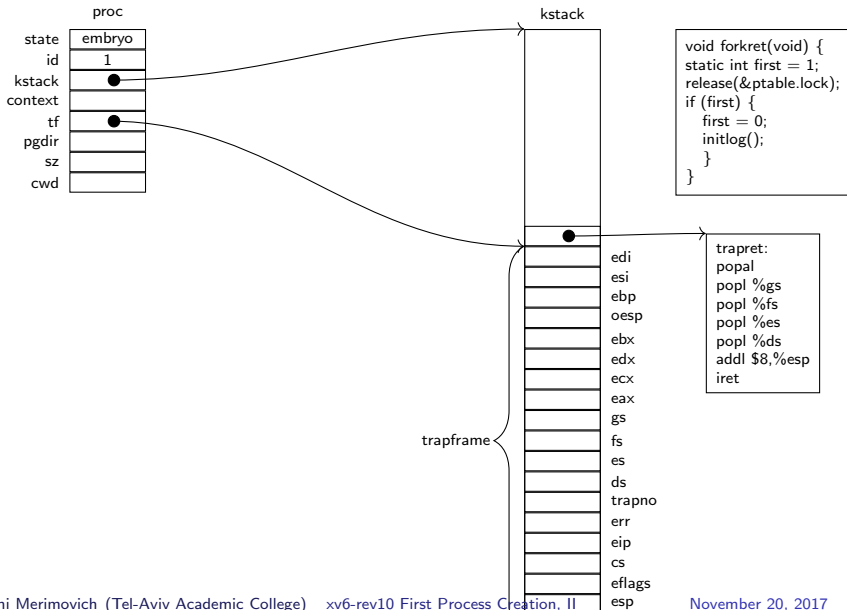
The whole first process creation



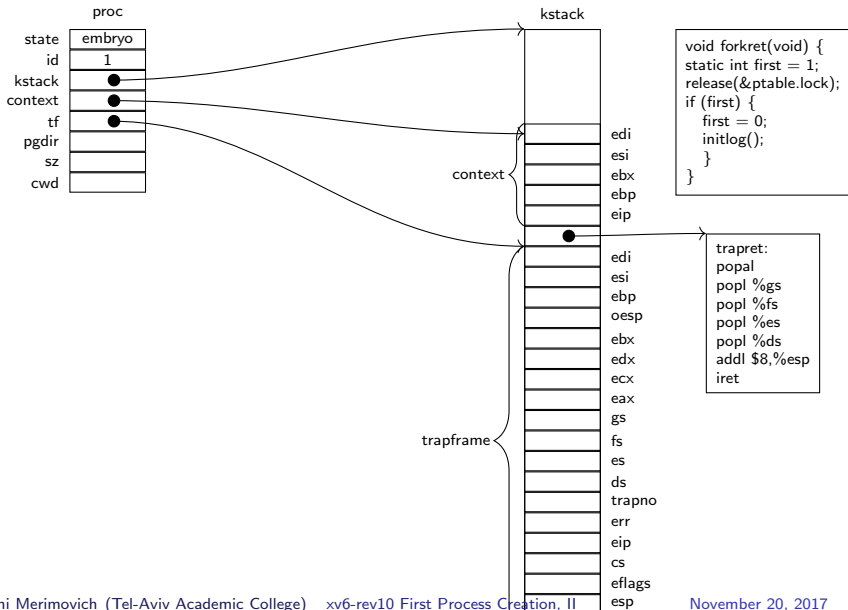
```
void forkret(void) {
    static int first = 1;
    release(&ptable.lock);
    if (first) {
        first = 0;
        initlog();
    }
}
```

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

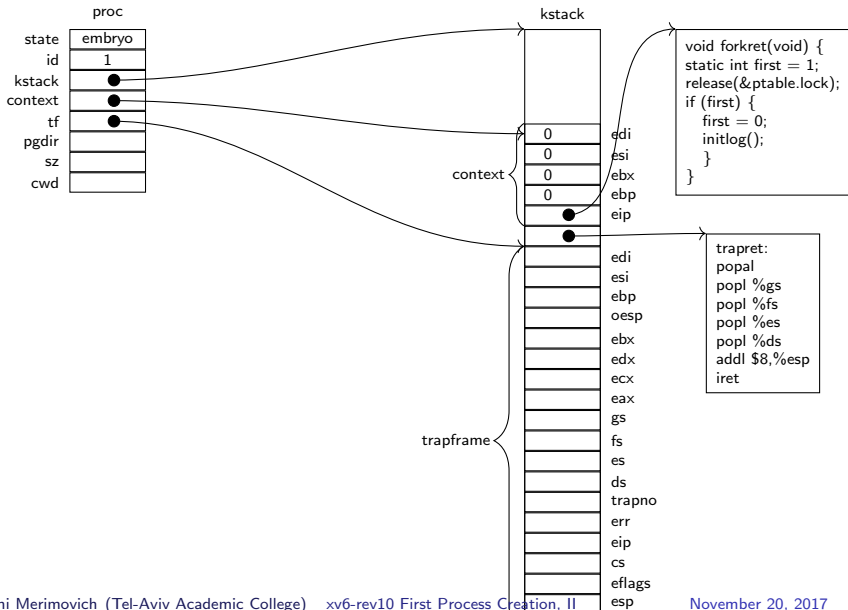

The whole first process creation



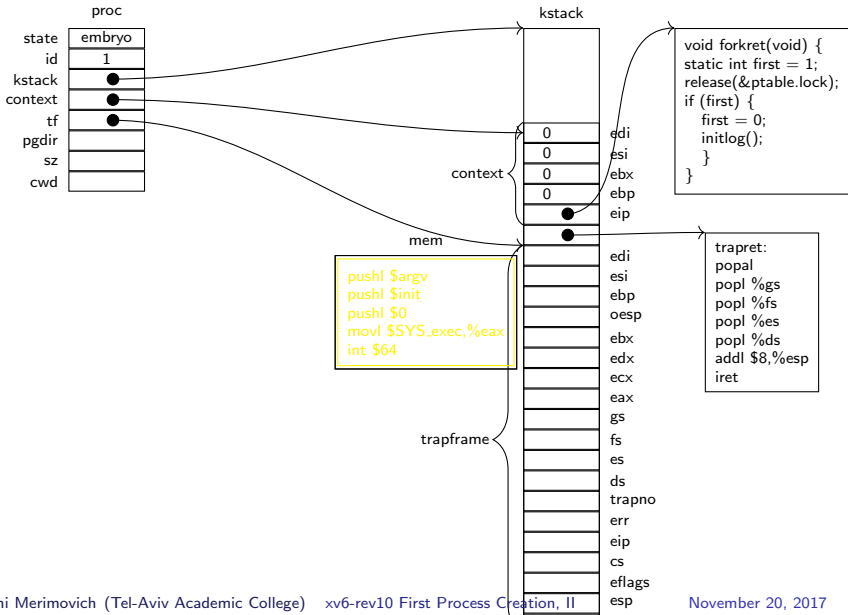
The whole first process creation



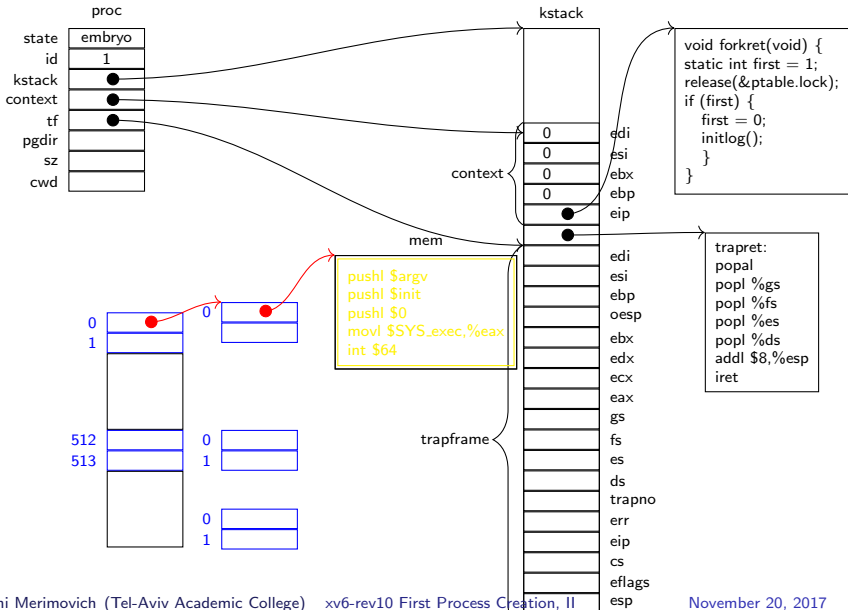
The whole first process creation



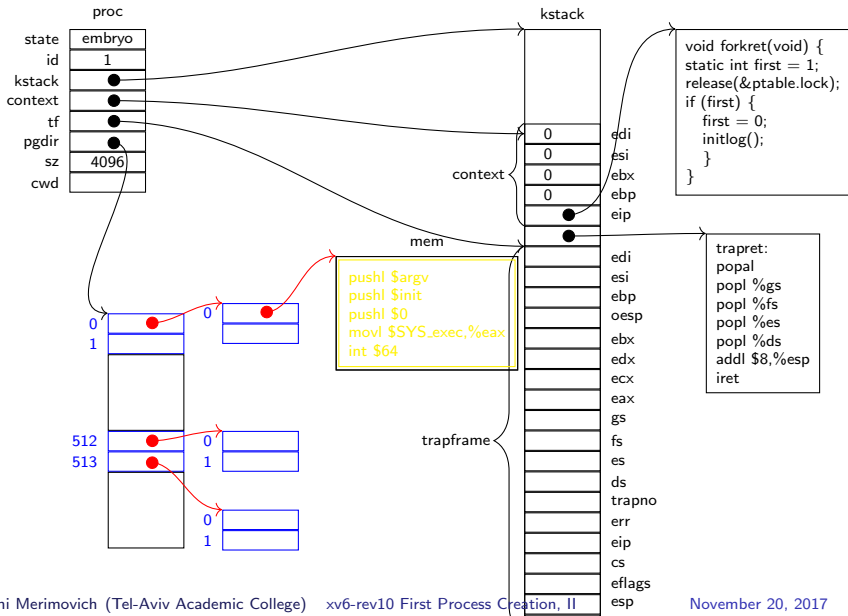
The whole first process creation



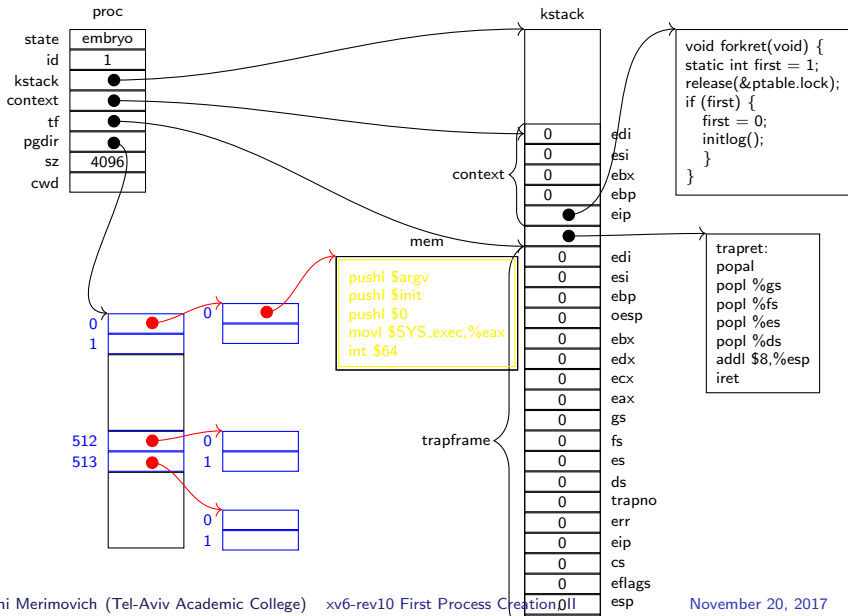
The whole first process creation



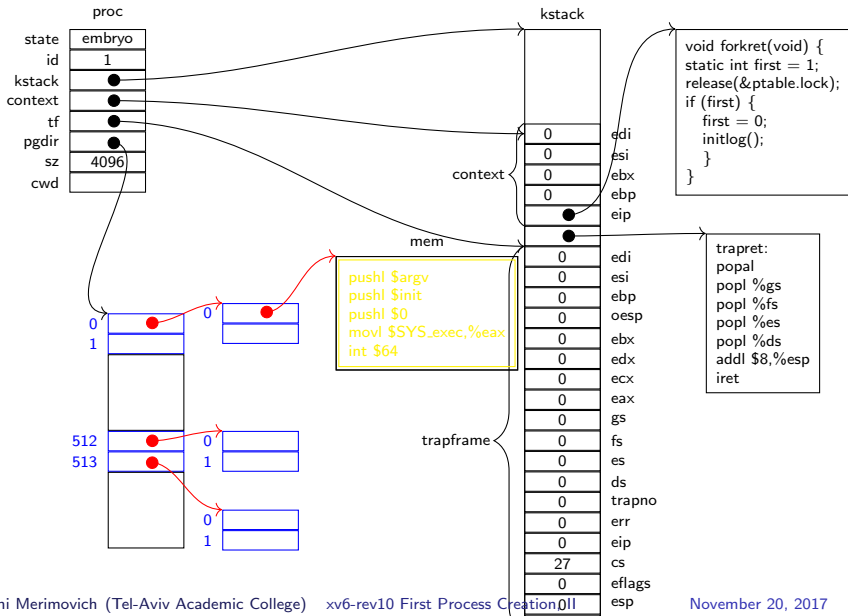
The whole first process creation



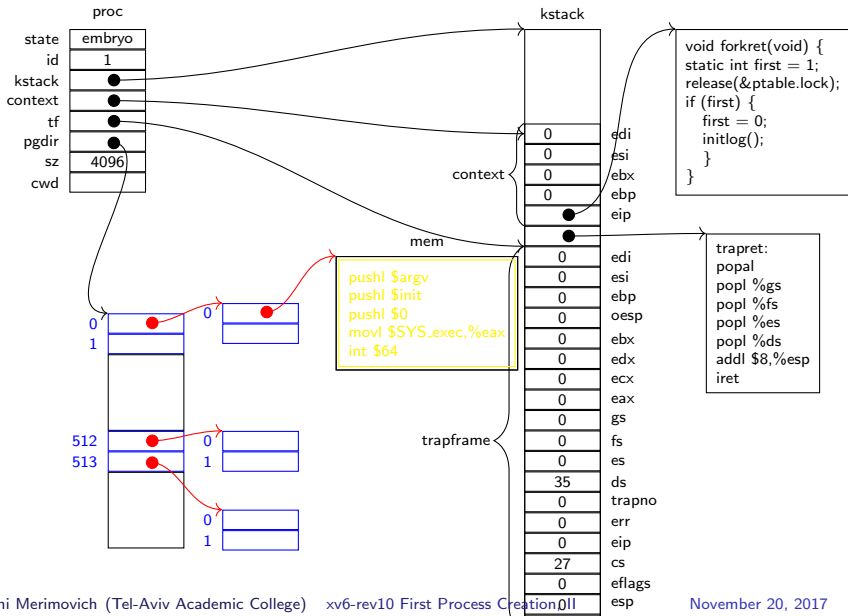
The whole first process creation



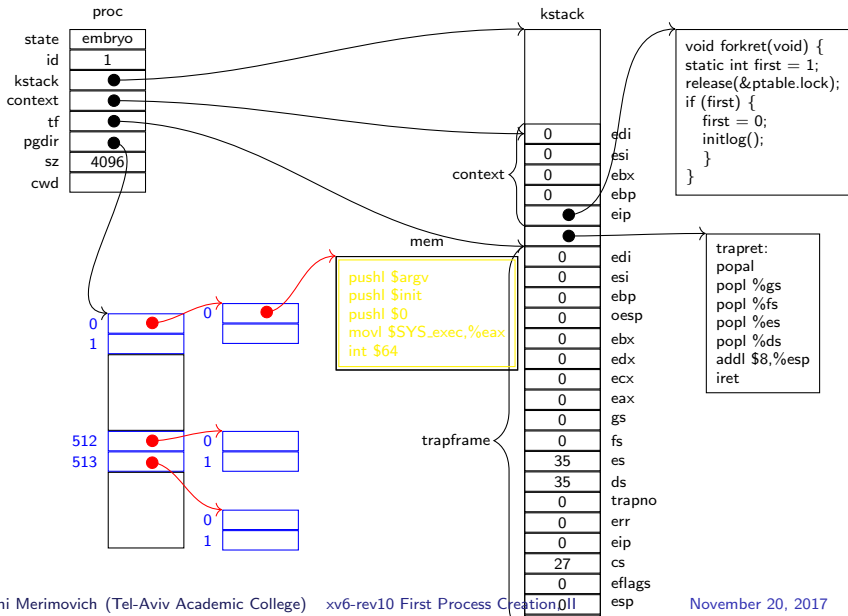
The whole first process creation



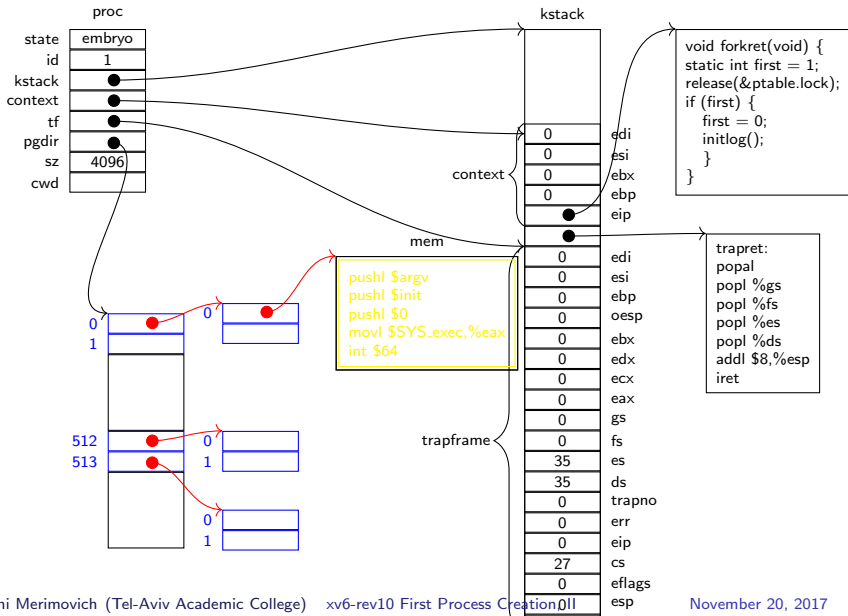
The whole first process creation



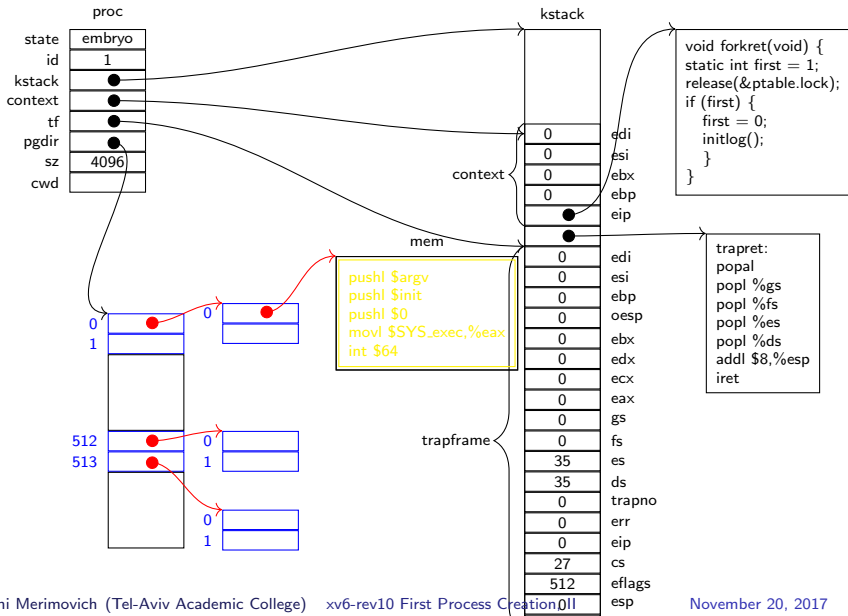
The whole first process creation



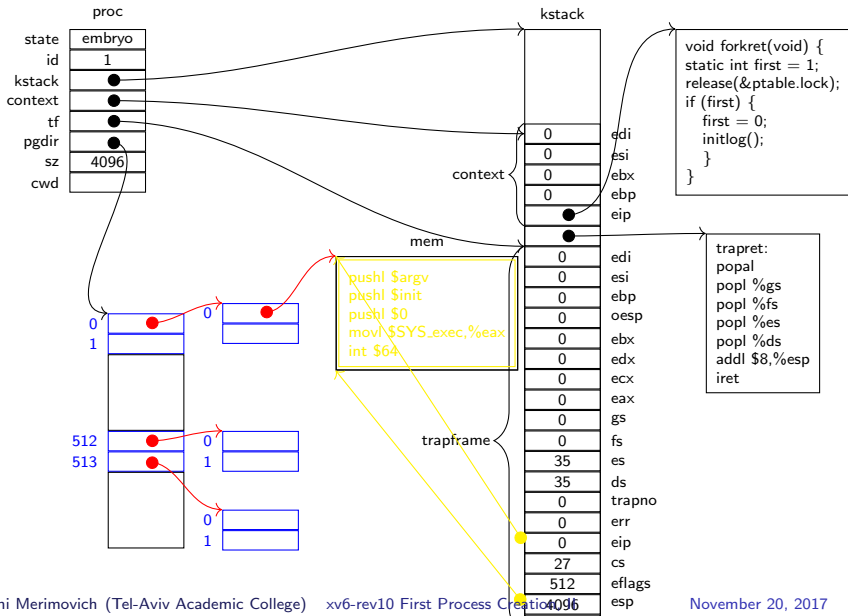
The whole first process creation



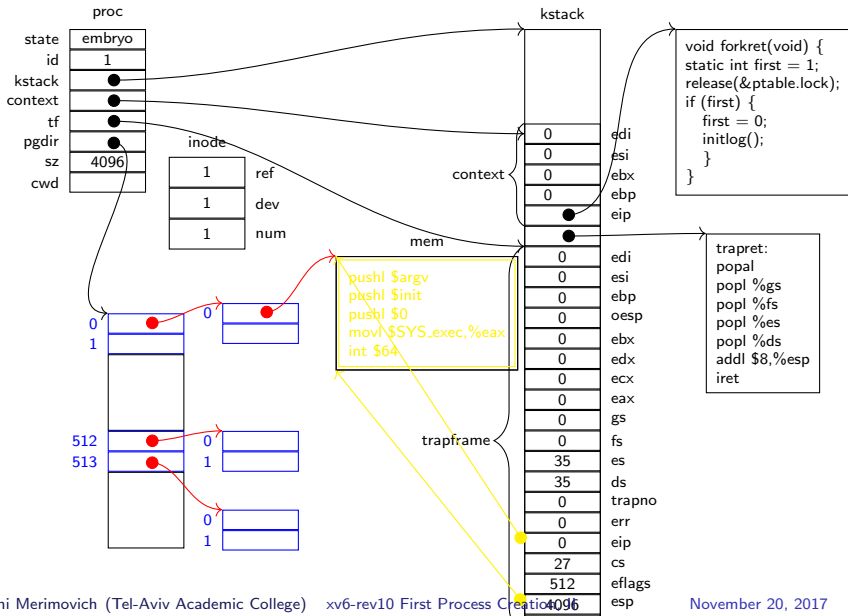
The whole first process creation



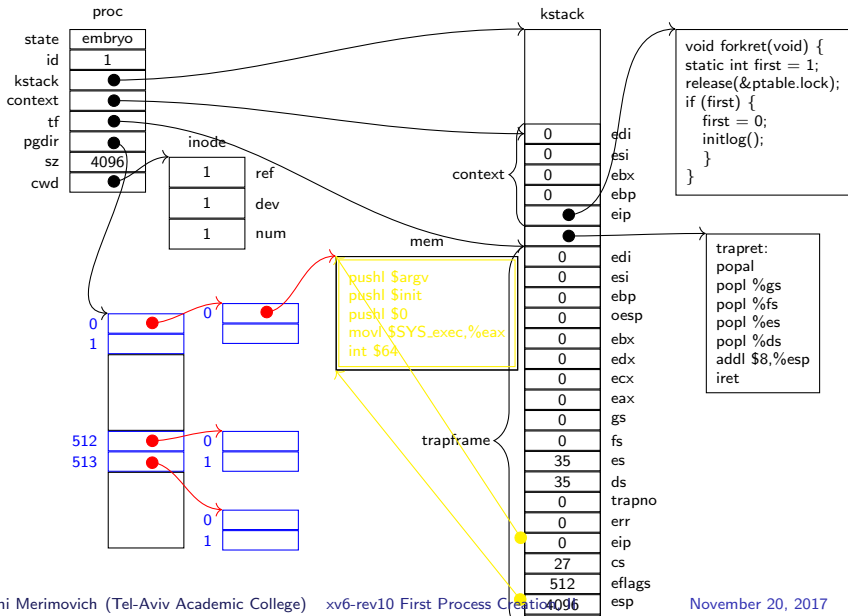
The whole first process creation



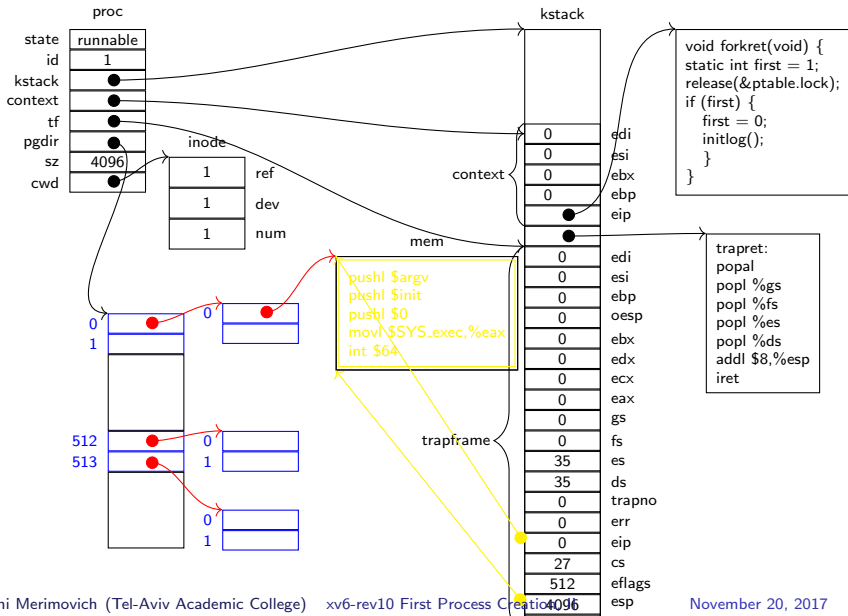
The whole first process creation



The whole first process creation



The whole first process creation



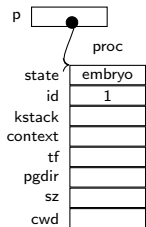
userinit()

- **allocproc()**: Allocate **proc** structure.
 - Allocate process kernel stack.
 - Leave space for a **trapframe** structure.
 - Build **context** structure on the stack.
- **setupkvm**: Create page table.
- **inituvm**: User space:
 - Allocate one page.
 - Copy user code to the allocated page.
 - Modify page table to use the allocated page.
- Set the **trapframe** so the user process will be able to run.

allocproc(): (1) Finding unused proc structure

```
2473 static struct proc *allocproc(void) {  
    struct proc *p;  
    char *sp;  
  
    acquire(&ptable.lock);  
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)  
        if (p->state == UNUSED)  
            goto found;  
    release(&ptable.lock);  
    return 0;  
  
found:  
    p->state = EMBRYO;  
    p->pid = nextpid++;  
    release(&ptable.lock);
```

allocproc(): (1) Operation



```
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
```

allocproc: (2) Initialize process kernel stack

2494

```
if ((p->kstack = kalloc()) == 0) {  
    p->state = UNUSED;  
    return 0;  
}  
sp      = p->kstack + KSTACKSIZE;  
sp      -= sizeof *p->tf;  
p->tf = (struct trapframe*)sp;  
sp      -= 4;  
*(uint*)sp = (uint)trapret;  
sp      -= sizeof *p->context;  
p->context = (struct context*)sp;  
memset(p->context, 0, sizeof *p->context);  
p->context->eip = (uint)forkret;  
return p;  
}
```

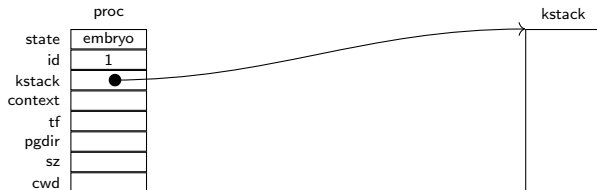
allocproc: (2) Operation

proc	
state	embryo
id	1
kstack	
context	
tf	
pgdir	
sz	
cwd	

```
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
```

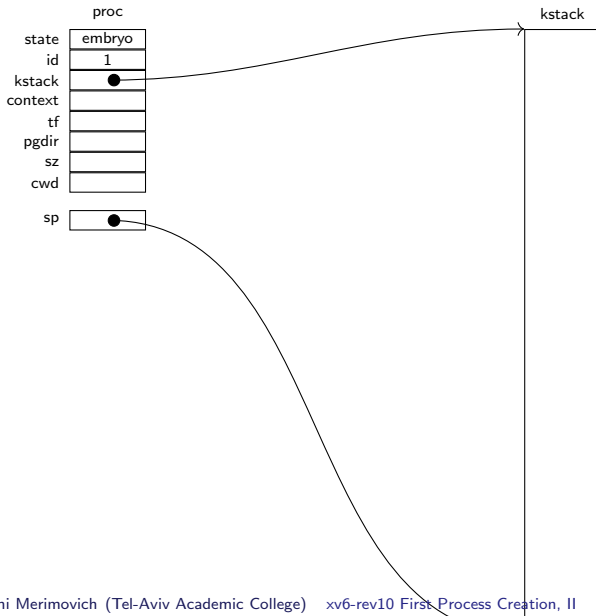
allocproc: (2) Operation



```
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
```

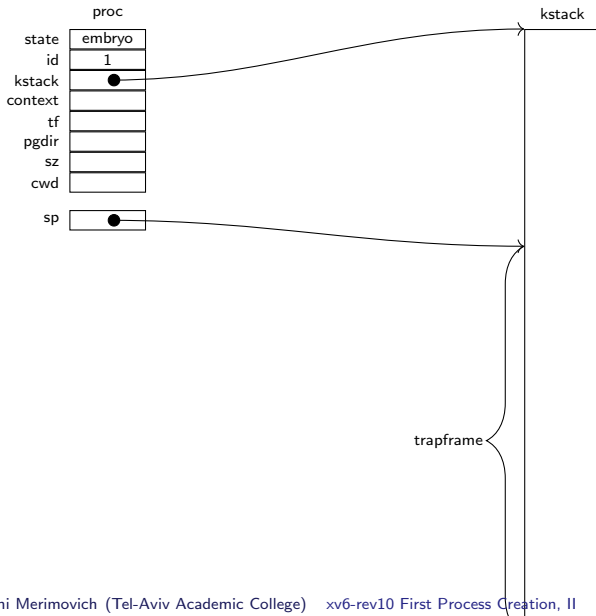
allocproc: (2) Operation



```
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
```

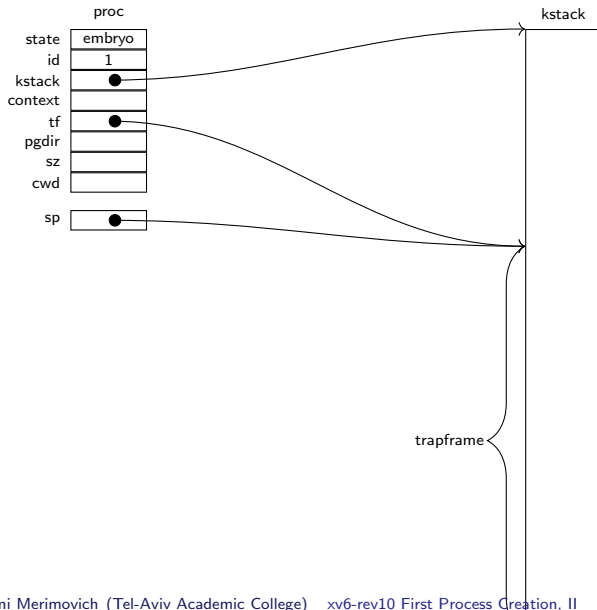
allocproc: (2) Operation



```
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
```

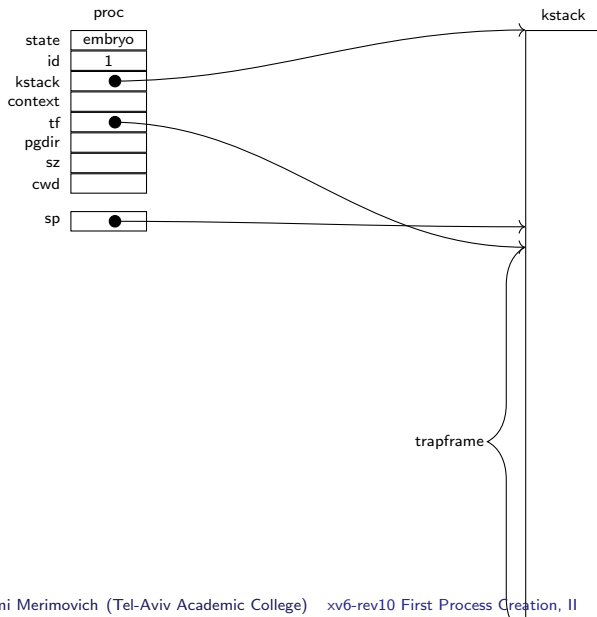

allocproc: (2) Operation



```
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
```

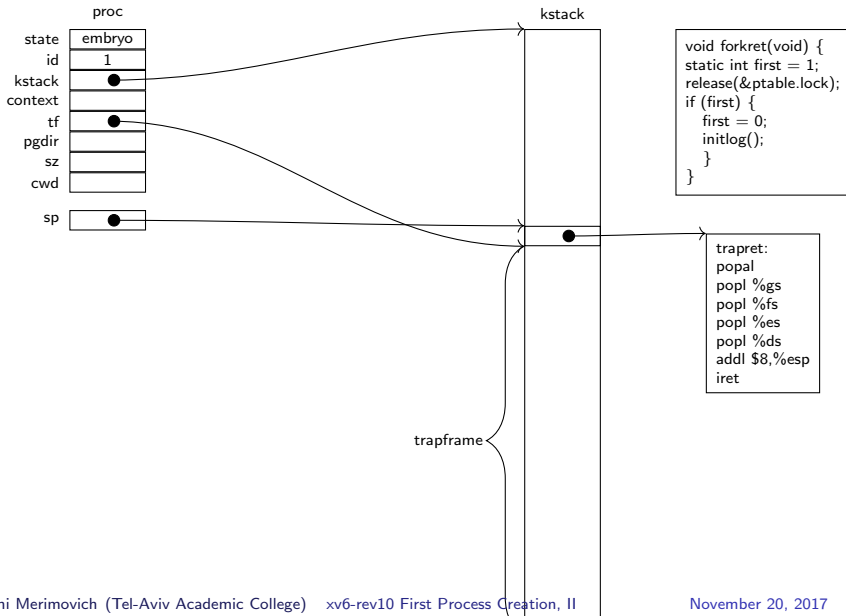
allocproc: (2) Operation



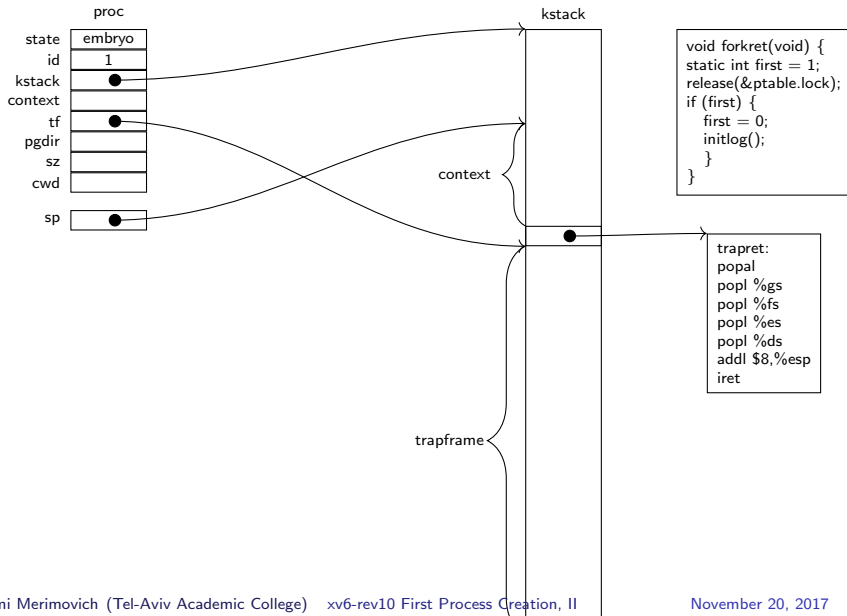
```
void forkret(void) {  
    static int first = 1;  
    release(&ptable.lock);  
    if (first) {  
        first = 0;  
        initlog();  
    }  
}
```

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

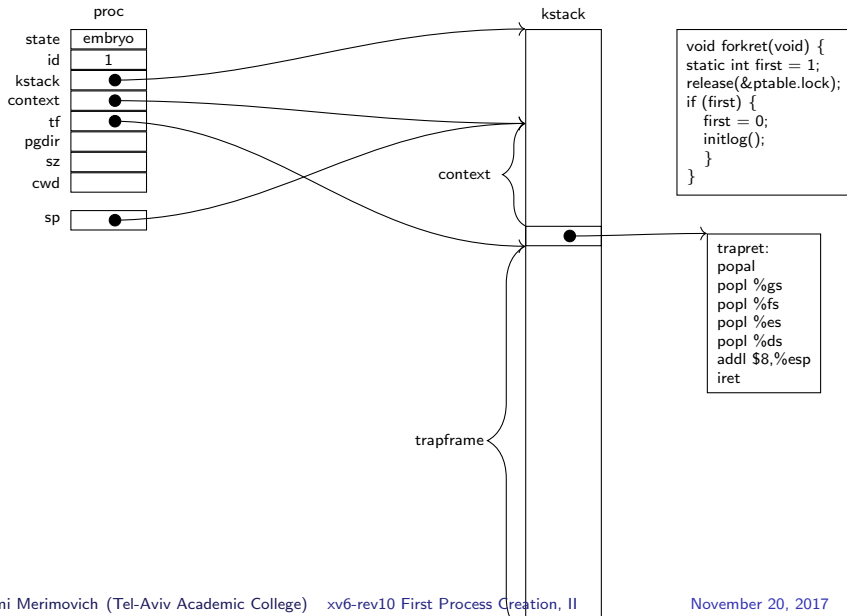
allocproc: (2) Operation



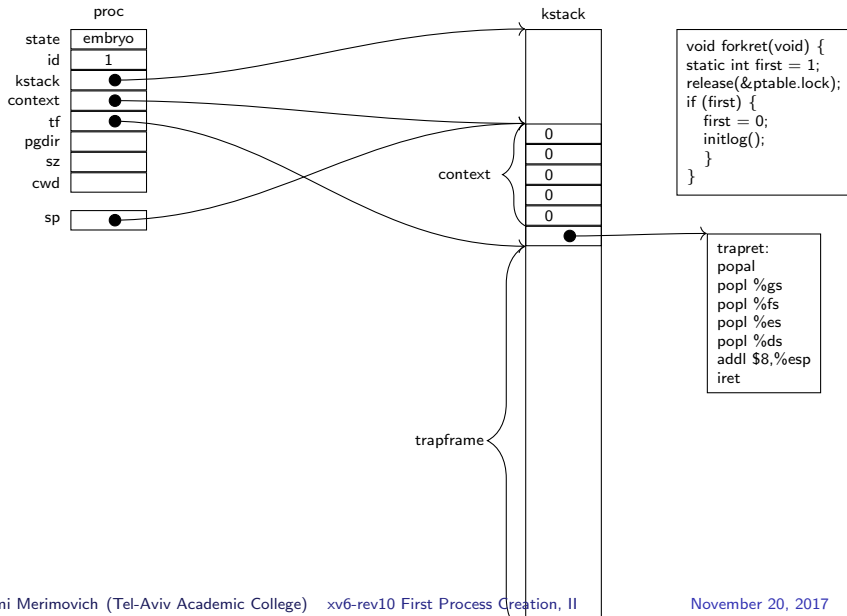
allocproc: (2) Operation



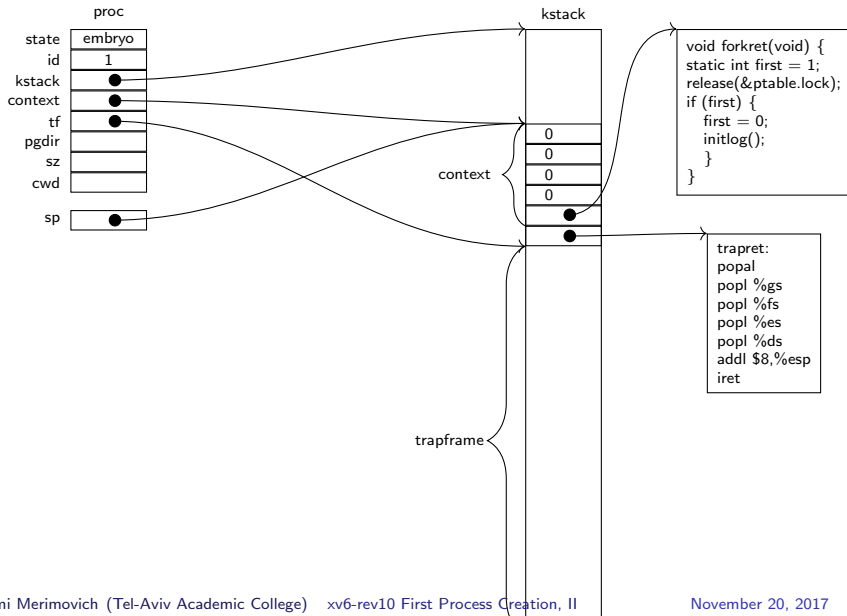
allocproc: (2) Operation



allocproc: (2) Operation



allocproc: (2) Operation



inituvm

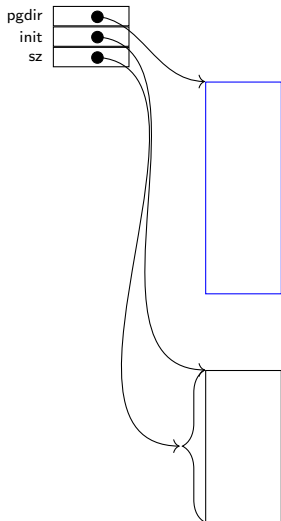
```
1886 inituvm(pde_t *pgdir, char *init, uint sz)
{
    char *mem;

    if (sz >= PGSIZE)
        panic("inituvm: more than a page");
    mem = kalloc();
    memset(mem, 0, PGSIZE);
    mappages(pgdir, 0, PGSIZE, v2p(mem), PTE_W|PTE_U);
    memmove(mem, init, sz);
}
```

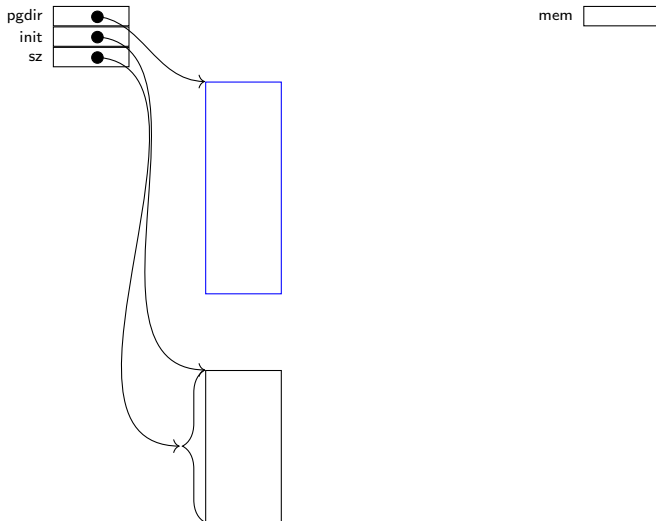

initvm() operation

pgdir	
init	
sz	

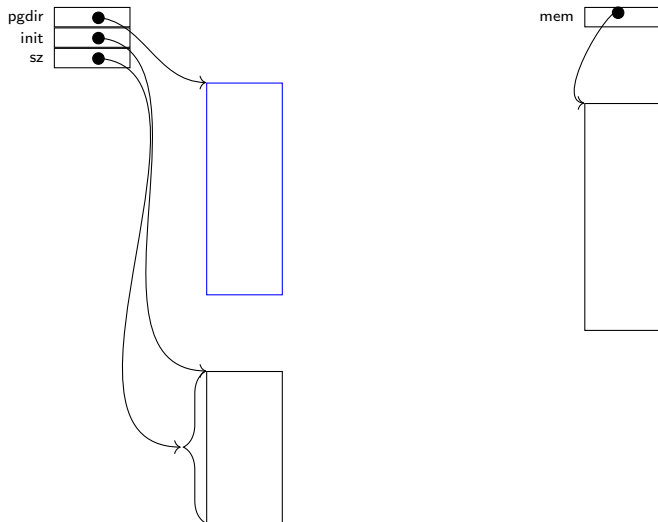
initvm() operation



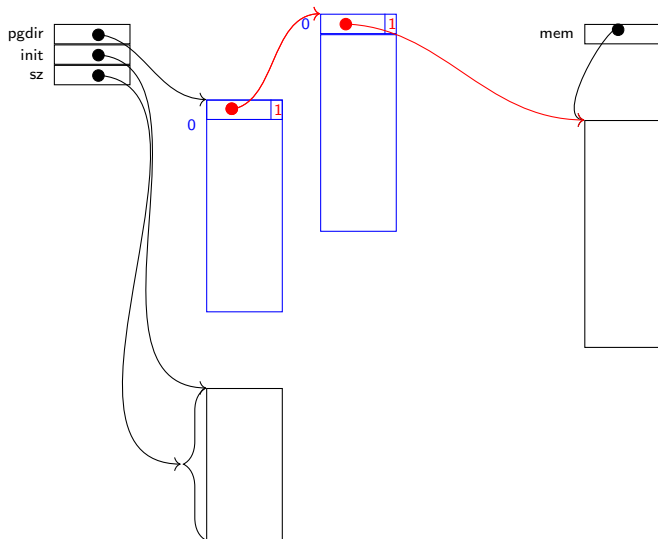
inituvm() operation



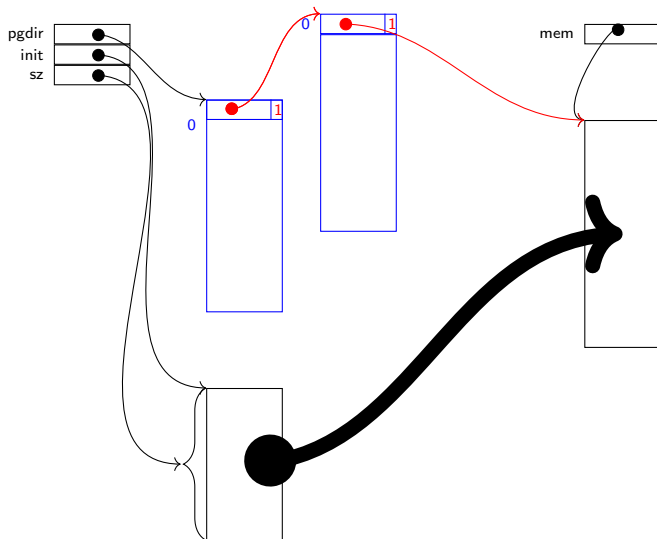
initvm() operation



initvm() operation



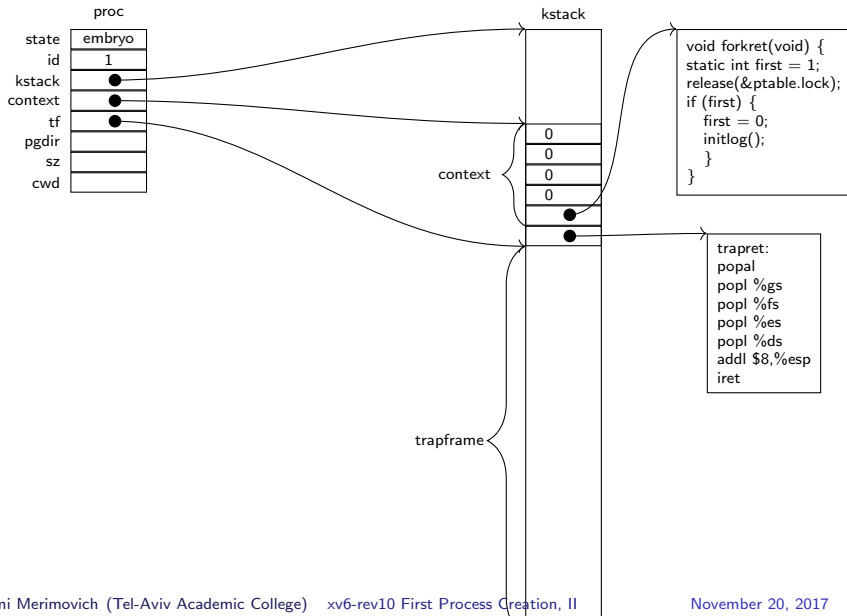
initvm() operation



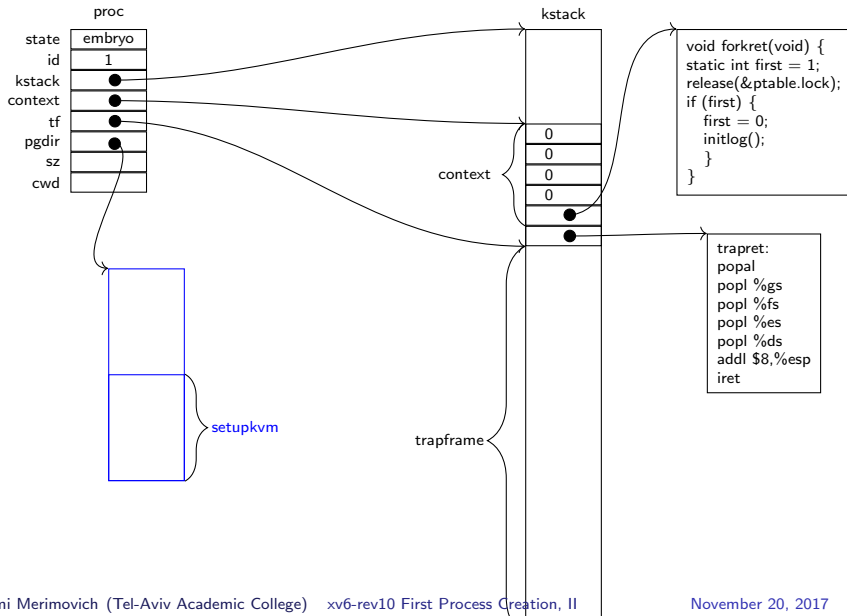
userinit(): (1) Invoking allocproc & initvm

```
2520 userinit(void) {  
    struct proc *p;  
    extern char _binary_initcode_start[],  
                _binary_initcode_size[];  
  
    p = allocproc();  
    initproc = p;  
    if ((p->pgdir = setupkvm()) == 0)  
        panic("userinit: out of memory");  
    initvm(p->pgdir, _binary_initcode_start,  
            (int) _binary_initcode_size);  
    p->sz = PGSIZE;
```

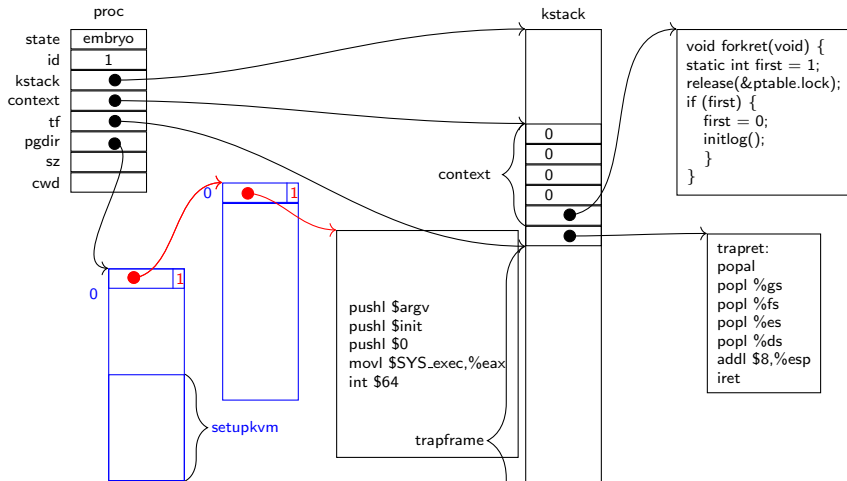
userinit(): (1) Operation



userinit(): (1) Operation



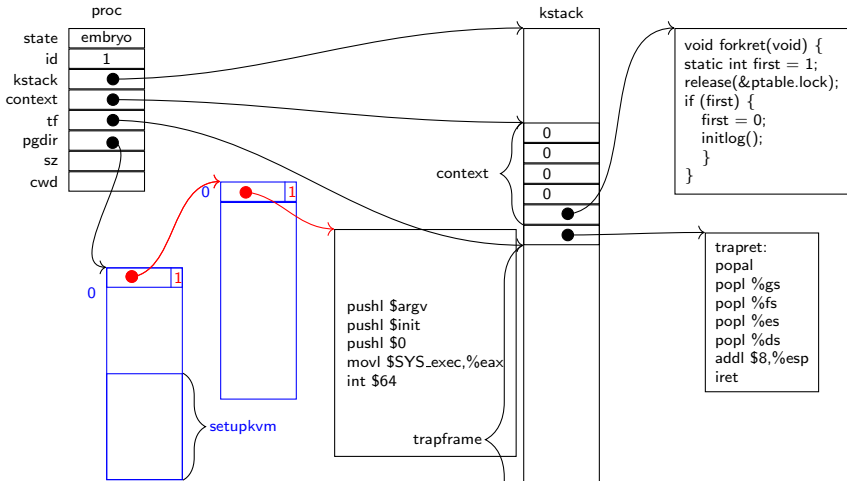
userinit(): (1) Operation



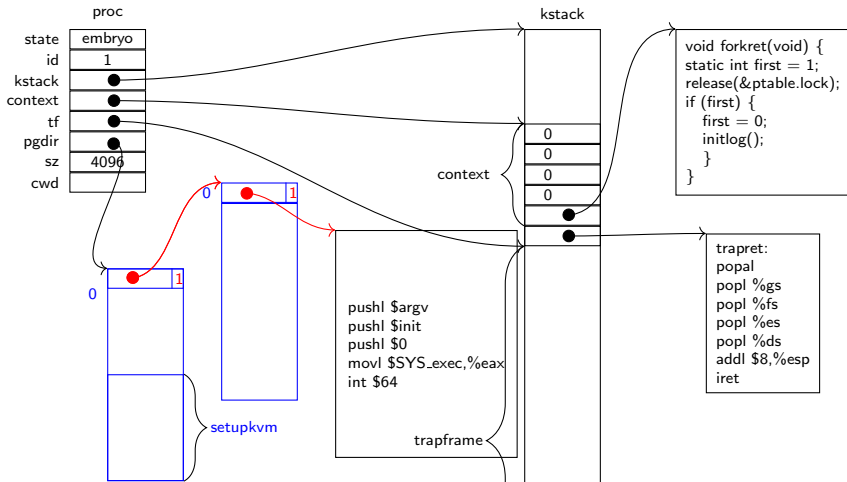
userinit(): (2) trapframe initialization

```
2532  memset(p->tf, 0, sizeof(*p->tf));  
    p->tf->cs = (SEG_UCODE << 3) | DPL_USER;  
    p->tf->ds = (SEG_UDATA << 3) | DPL_USER;  
    p->tf->es = p->tf->ds;  
    p->tf->ss = p->tf->ds;  
    p->tf->eflags = FL_IF;  
    p->tf->esp = PGSIZE;  
    p->tf->eip = 0; // beginning of initcode.S  
    p->cwd = namei("/");  
    p->state = RUNNABLE;  
}
```

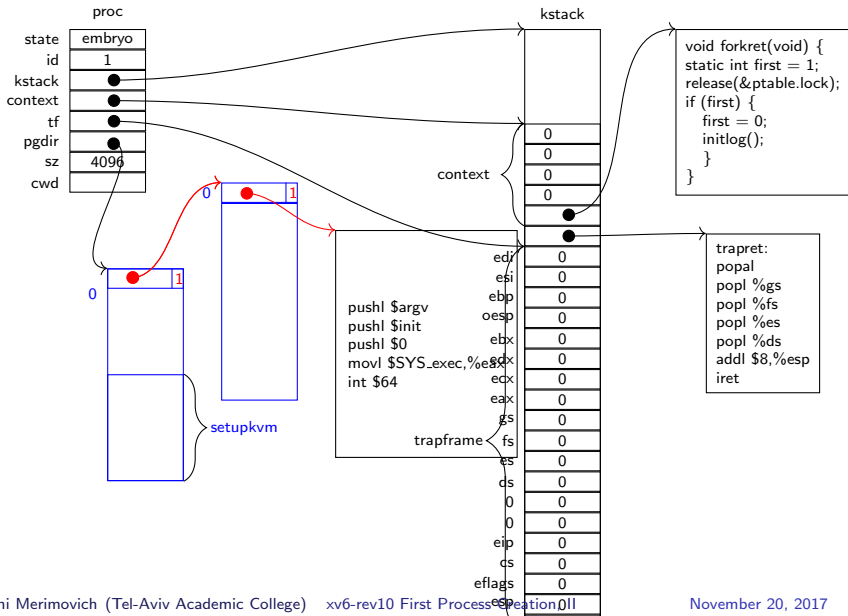
userinit(): (2) Operation



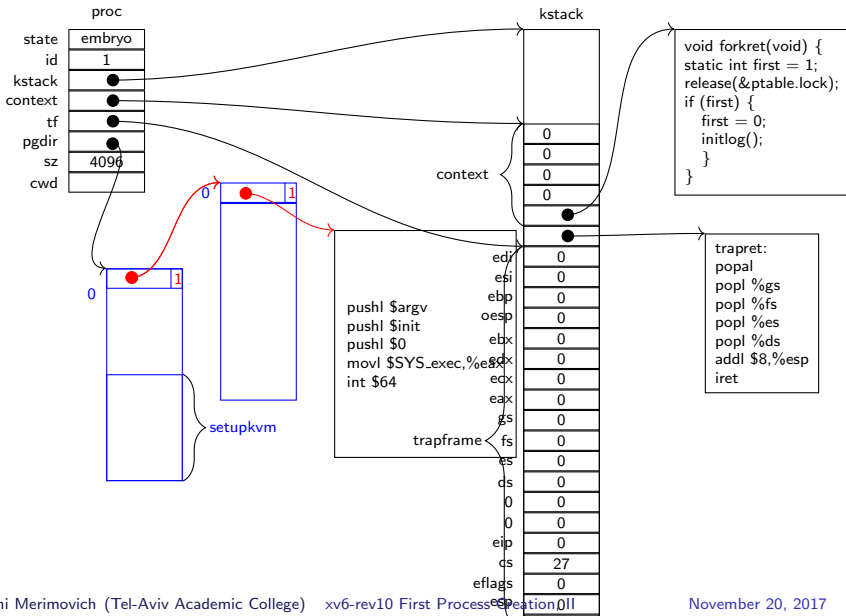
userinit(): (2) Operation



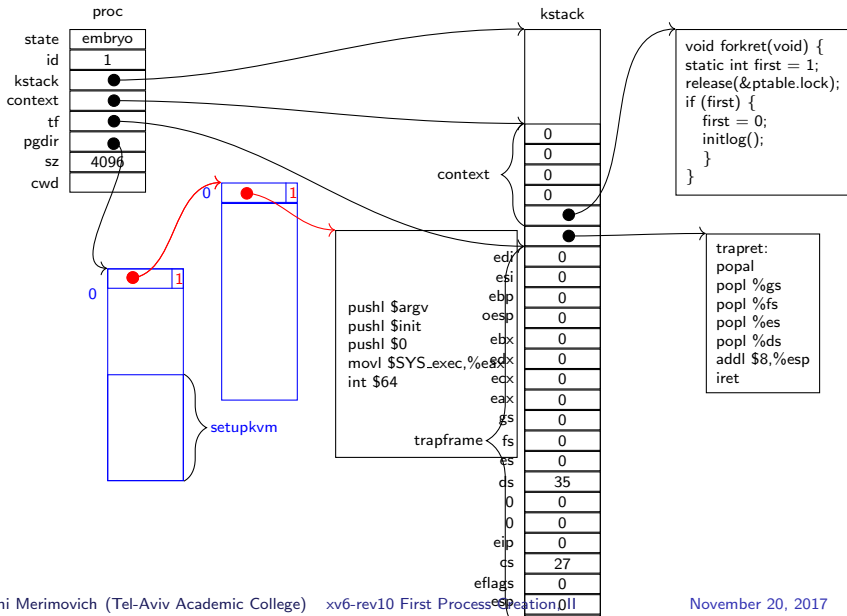
userinit(): (2) Operation



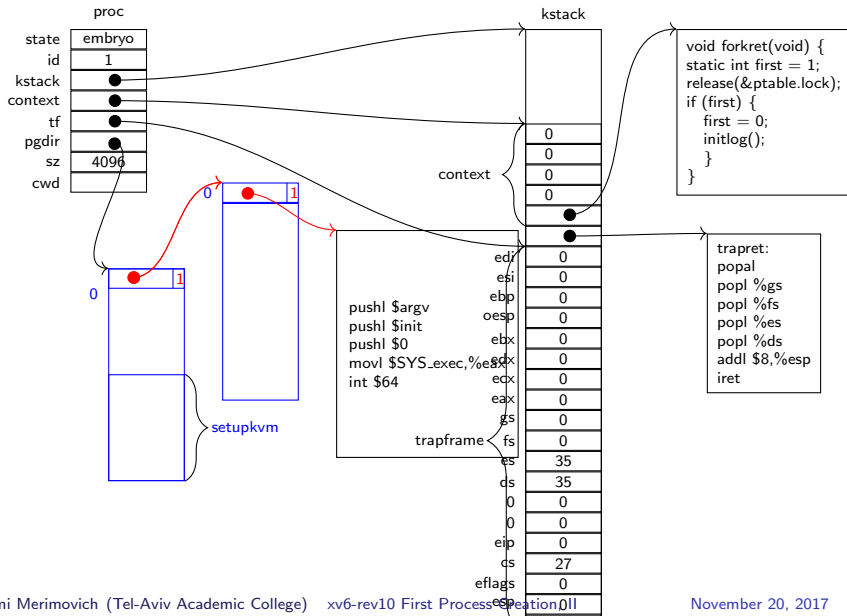
userinit(): (2) Operation



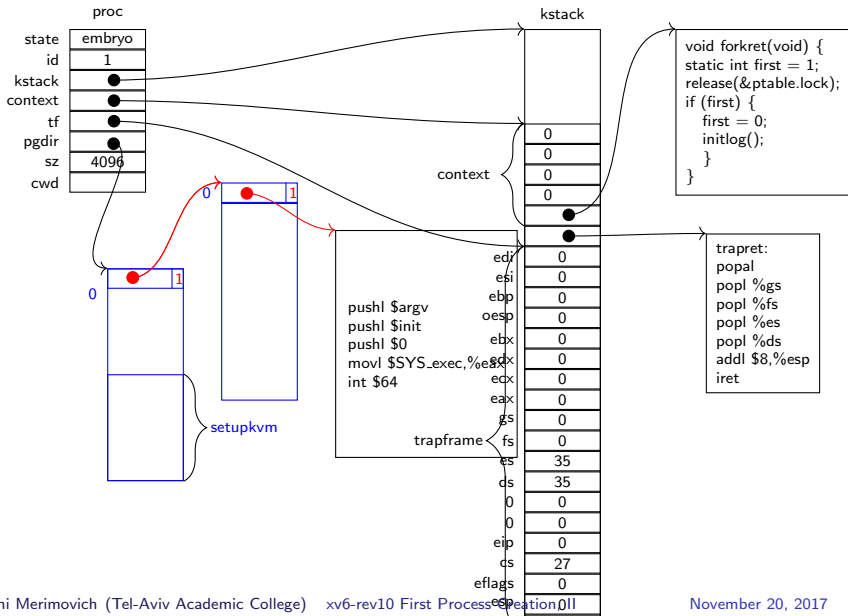
userinit(): (2) Operation



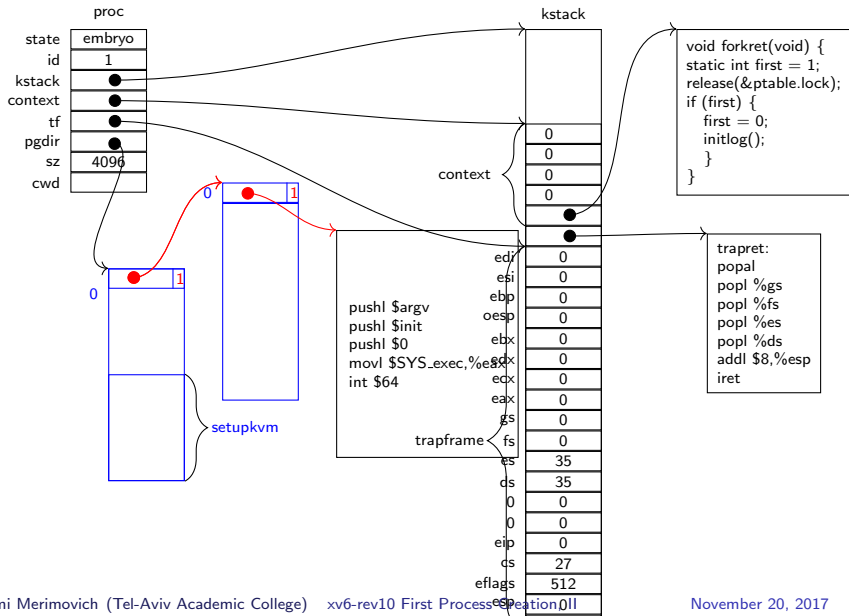
userinit(): (2) Operation



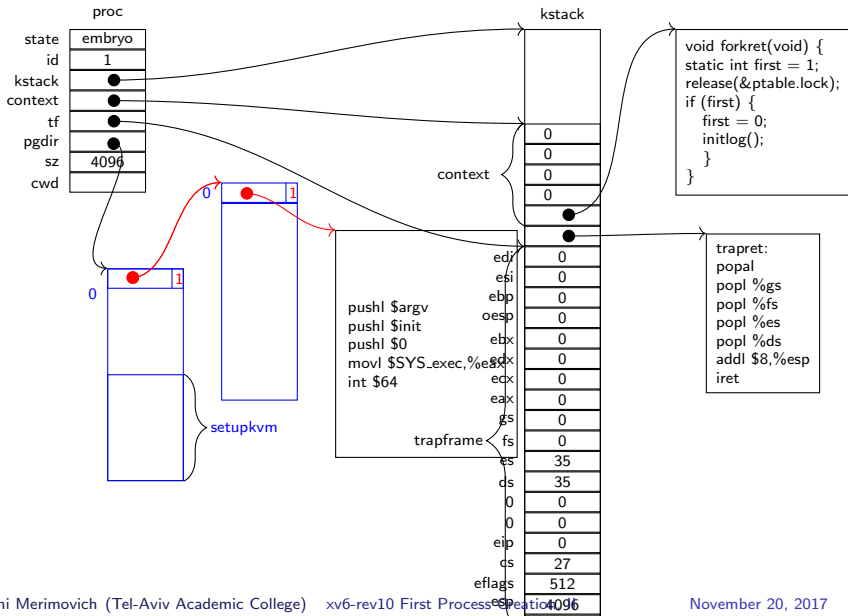
userinit(): (2) Operation



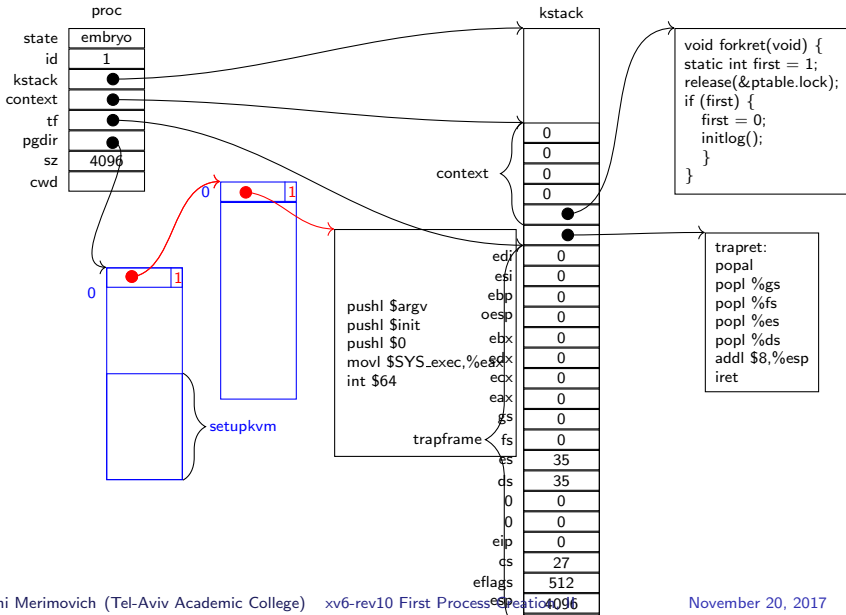
userinit(): (2) Operation



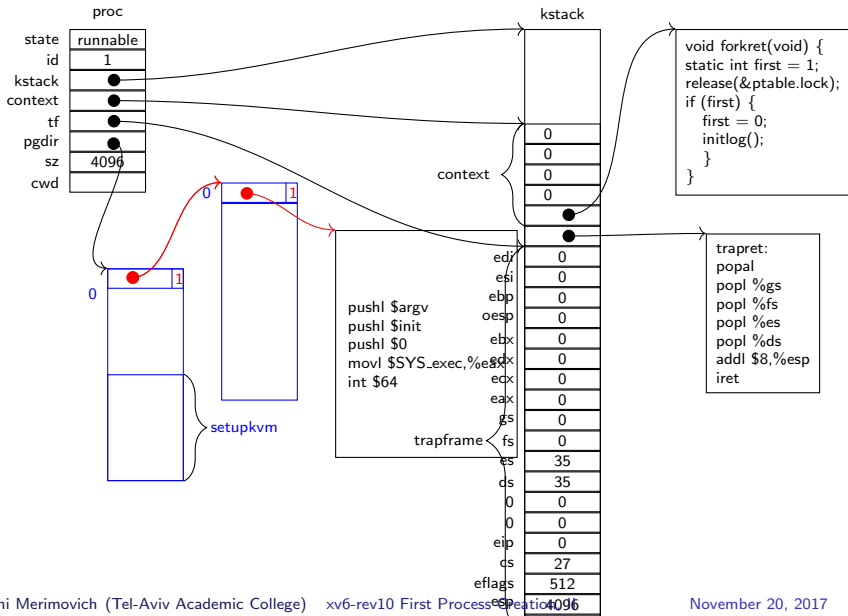
userinit(): (2) Operation



userinit(): (2) Operation



userinit(): (2) Operation



trapret

```
3324 trapret :  
    popal  
    popl %gs  
    popl %fs  
    popl %es  
    popl %ds  
    addl $0x8, %esp # trapno and errcode  
    iret
```

- What is the registers state after the `iret` instruction?

Registers after iret

eax {	0	} eax,ecx,edx,ebx,ebp,esi,edi	
edi {	0		
esp {	4096		
eip {	0		
cs {	SEG_UCODE	011	
ds {	SEG_UDATA	011	
ss {	SEG_UDATA	011	
es {	0		
fs {	0		
gs {	0		

Informing the CPU about safe stack

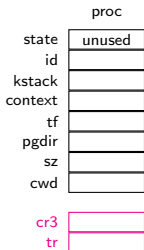
The PUSHAL/POPAL instructions

```
pushl %eax  
pushl %ecx  
pushl %edx  
pushl %ebx  
pushl %esp !!!  
pushl %ebp  
pushl %esi  
pushl %edi
```

```
popl %edi  
popl %esi  
popl %ebp  
popl %esp  
popl %ebx  
popl %edx  
popl %ecx  
popl %eax
```

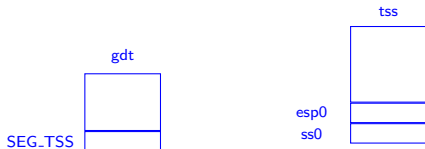
What now?

- How the kernel will return to run?
 - Only by interrupt service routine.
- Where is the interrupt service routine located?
 - Later topic,
- What happens to the stack when interrupt is delivered??
 - GOOD POINT.
 - Before going to user mode we set stack address for interrupts.



```
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
```

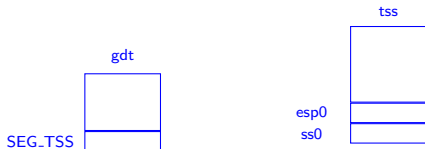


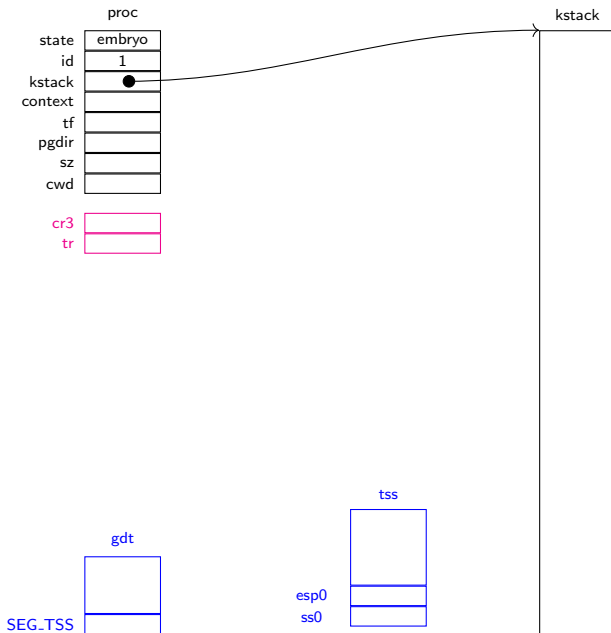
	proc
state	embryo
id	1
kstack	
context	
tf	
pgdir	
sz	
cwd	

cr3	
tr	

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

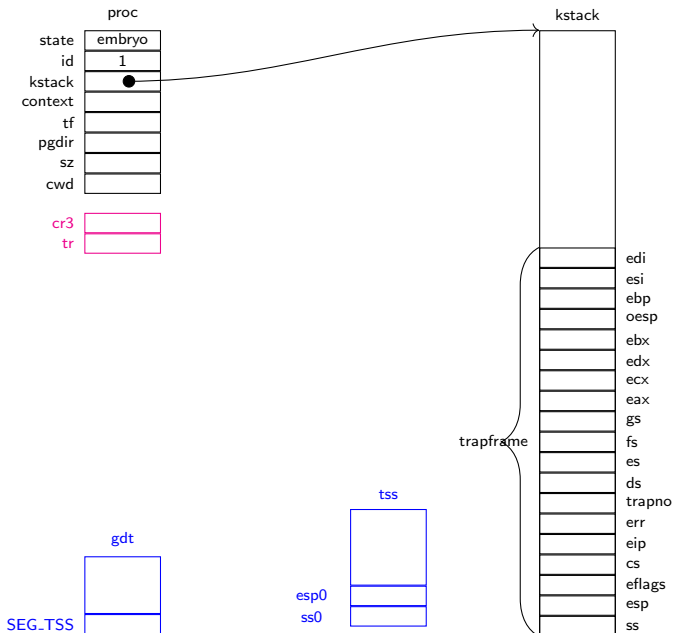
```
trapret:
    popal
    popl %gs
    popl %fs
    popl %es
    popl %ds
    addl $8,%esp
    iret
```





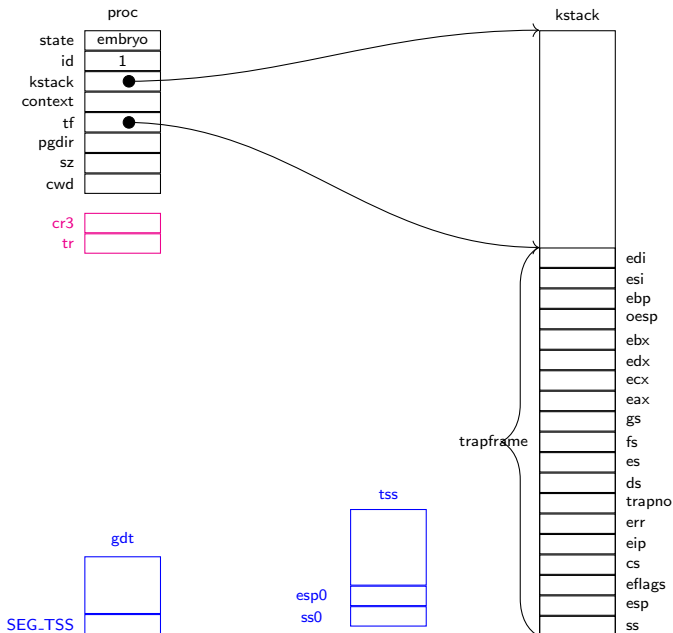
```
void forkret(void) {
    static int first = 1;
    release(&ptable.lock);
    if (first) {
        first = 0;
        initlog();
    }
}
```

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



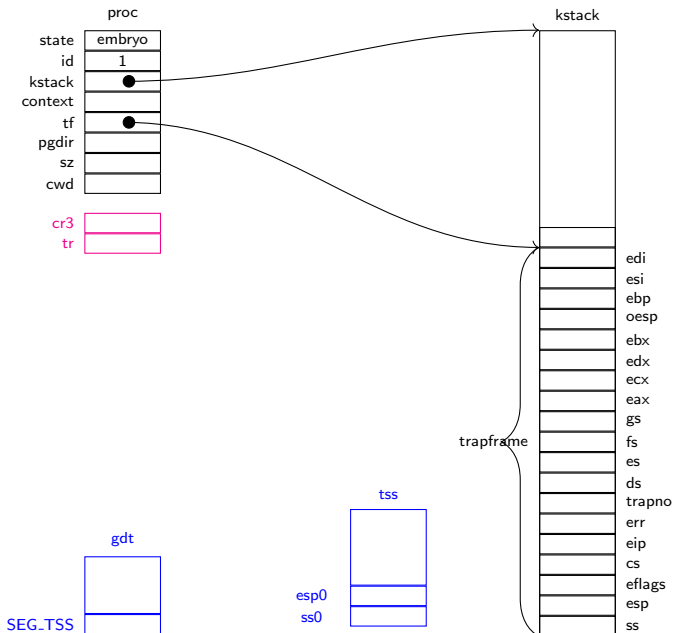
```
void forkret(void) {
    static int first = 1;
    release(&ptable.lock);
    if (first) {
        first = 0;
        initlog();
    }
}
```

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



```
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
```

```
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
```

