



MKC - Exercise 3

Nils Asmussen

2021-07-01



- Create new Execution Contexts (threads)
- Manage ECs in a (double linked ring) list
- Switch between them (cooperatively)

- Hands-on
 - User-level threading
 - 1st “real” system call: `create_ec`
 - 2nd system call: `yield`

```
$ git clone
```

```
https://os.inf.tu-dresden.de/repo/git/mkc.git
```

```
$ git checkout exercise3
```

```
# build it
```

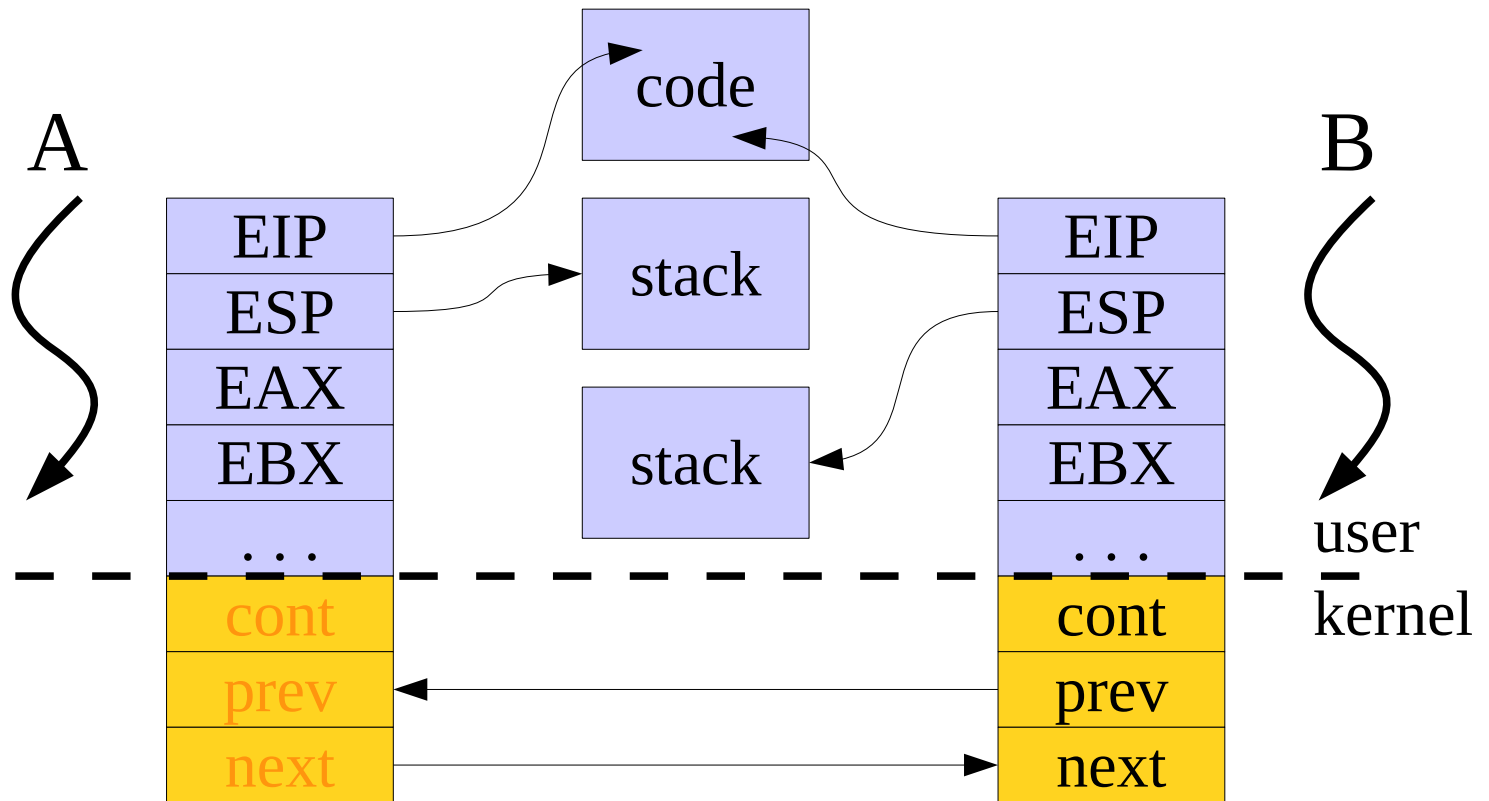
```
$ make
```

```
# run it
```

```
$ make run
```

- Very very simple scheduler
 - No priorities, no time budgets
 - Cooperative multithreading
 - Single address space, uniprocessor
- Kernel: kern/include/ec.h
 - Registers (state)
 - Continuation (where to continue execution)
 - Management information (e.g. ***prev**, ***next**)
- User: user/src/user.cc
 - Code (instruction pointer)
 - Most likely a Stack (stack pointer)

What is a Thread/EC?



- Thread function: no parameter, nothing to return, but needs a stack
- Where to get the new stack from? `malloc()` → not available (so far)
- Put it statically in data segment or on local stack of the currently running thread:
`char new_stack[64];`
- Stack grows downwards, thus ESP should point to the end: **`new_stack + sizeof(new_stack)`**

- Write a new thread function in `user/src/user.cc`
 - Simple function doing nothing but spinning
 - Later it shall call **`sys_yield()`**, thus switching to the next thread
- New bindings for to-be-written syscalls:
 - **`sys_create_ec`** (2 arguments):
 - Creates a shining new Execution Context
 - EIP of new EC (thread function's address)
 - ESP to be used - we need a user stack per EC
 - **`sys_yield`** (no argument)
 - Simply switches to the next thread

- Organize ECs in a ring list
 - add **prev** and **next** pointer (kern/include/ec.h)
 - Private **enqueue()** function, adding **this** to the tail of the list (kern/src/ec.cc)
 - Special case when creating very first EC, **Ec::current** is not yet set, watch out!
- Add a new system call
 - Two parameters (instruction and stack pointer)
 - **Ec::sys_regs()** and kern/include/regs.h
 - Create **new EC**, add it to the list, and sysexit
 - Verbose printf, newly created EC, its EIP/ESP, maybe even the whole list of ECs

- Switch from currently running EC (**Ec::current**) to next one (**current->next**)
 - Every EC has a continuation - the function to execute whenever becoming ready (again)
 - The currently running thread shall continue with **ret_user_sysexit**, thus set **cont** accordingly
 - Switch to **current->next** via **make_current()**
- Create more threads in user application, printf whenever they yield: **EC:%p → EC:%p**