barkhausen institut

Microkernel Construction

Exercise 3: Threads

Nils Asmussen

2026-05-21

Roadmap

- 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`

Get the Code

```shell
$ git clone https://github.com/Nils-TUD/MKC
$ git checkout exercise3
```

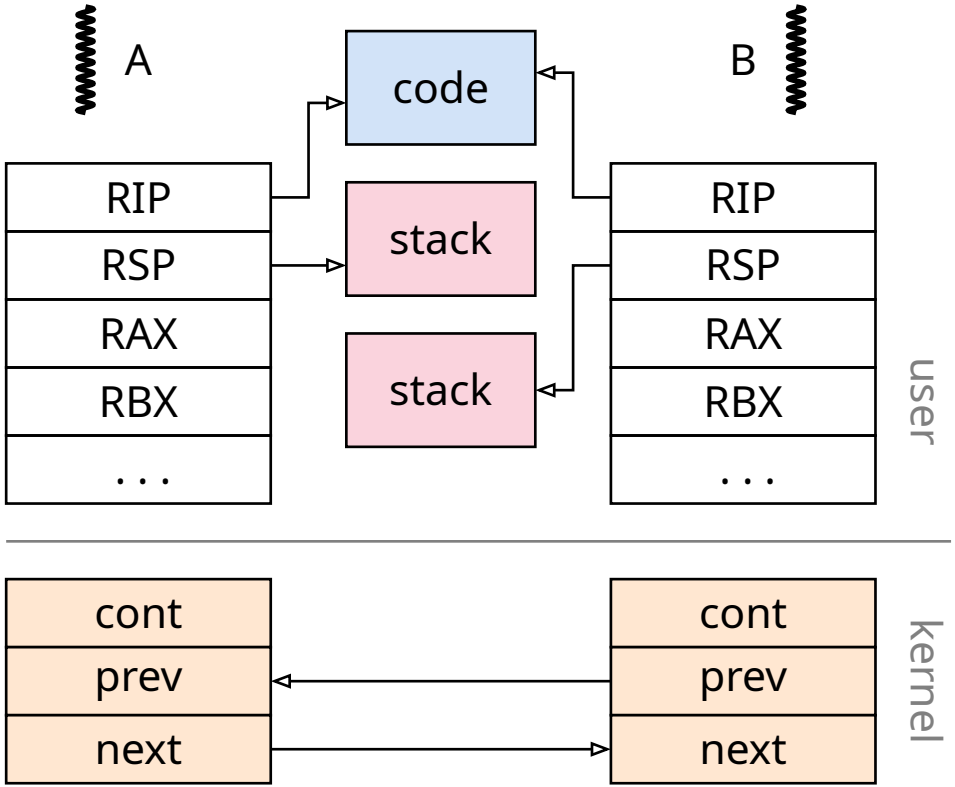
```shell
# build it
$ make
```

```shell
# run it
$ make run
```

Scheduling

- 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?



New User Level Thread

- Thread function: no parameter, nothing to return, but needs a stack
- Where to get the new stack from?
  - `malloc()` → not available (so far)
  - Statically in data segment
  - On local stack of the currently running thread:

    ```
    char new_stack[64];
    ```
- Stack grows downwards, thus `RSP` should point to the end:
  - `new_stack + sizeof(new_stack)`

Task 0: Minimal Thread User Code

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

Task 1: `sys_create_ec`

- 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 still NULL, 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 with `RIP`/`RSP` (whole list?)

Task 2: `sys_yield`

- Switch from `Ec::current` to next one (`current->next`)
  - Every EC has a continuation (where to continue on resume)
  - Current thread shall continue with `ret_user_sysexit`
  - Switch to `current->next` via `make_current()`
- Create more threads in user application
  - printf whenever they yield: `EC:%p -> EC:%p`