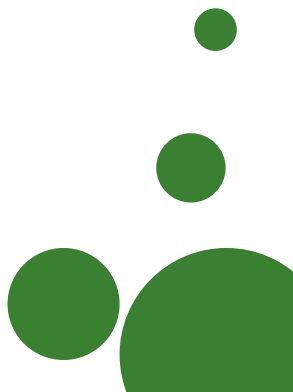




NSWI200 Operating Systems

Lubomír Bulej, Viktor Fuglík, Vojtěch Horký

2026-10-01



- teachers
 - Lubomír Bulej
 - Viktor Fuglík
 - Vojtěch Horký
- agenda
 - lab contents
 - first milestones (assignment)

- labs for discussion are preferred
 - please, do not be afraid to ask :-)

- labs for discussion are preferred
 - please, do not be afraid to ask :-)
- GitLab forum
 - we do not watch your repositories

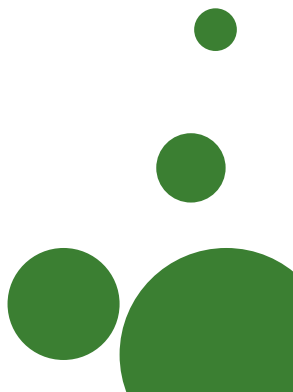
- labs for discussion are preferred
 - please, do not be afraid to ask :-)
- GitLab forum
 - we do not watch your repositories
- e-mail only for private matters
 - on anything else we will not respond over e-mail

- labs for discussion are preferred
 - please, do not be afraid to ask :-)
- GitLab forum
 - we do not watch your repositories
- e-mail only for private matters
 - on anything else we will not respond over e-mail
- exceptional situations
 - be proactive and contact us as soon as possible (where possible)

- labs for discussion are preferred
 - please, do not be afraid to ask :-)
- GitLab forum
 - we do not watch your repositories
- e-mail only for private matters
 - on anything else we will not respond over e-mail
- exceptional situations
 - be proactive and contact us as soon as possible (where possible)
- submission via repositories on GitLab

- labs for discussion are preferred
 - please, do not be afraid to ask :-)
- GitLab forum
 - we do not watch your repositories
- e-mail only for private matters
 - on anything else we will not respond over e-mail
- exceptional situations
 - be proactive and contact us as soon as possible (where possible)
- submission via repositories on GitLab
- set up **notifications** on Forum and upstream repositories

Understanding the concepts work better by implementing them rather than letting someone else implement it for you.



Milestones

M00: C language refresher



M00: C language refresher



- write your own printf

M00: C language refresher



- write your own `printf`
- learn how to use `list_t` linked list type

M00: C language refresher



- write your own `printf`
- learn how to use `list_t` linked list type
- runs natively on your own machine (no cross-compilation)

- automated tests from us for all milestones

- automated tests from us for all milestones
- part of GitLab CI but runnable locally as well
 - NSWI177 materials are still available in case they are needed ;-)

- automated tests from us for all milestones
- part of GitLab CI but runnable locally as well
 - NSWI177 materials are still available in case they are needed ;-)
- feel free to
 - investigate how they work
 - copy and extend them to your own needs

- automated tests from us for all milestones
- part of GitLab CI but runnable locally as well
 - NSWI177 materials are still available in case they are needed ;-)
- feel free to
 - investigate how they work
 - copy and extend them to your own needs
- but do not change their logic!
 - adding your own logging prints is okay, though

- first lines of your kernel

- first lines of your kernel
- few lines of assembly from us, otherwise *tabula rasa*

- first lines of your kernel
- few lines of assembly from us, otherwise *tabula rasa*
- move your `printf` to kernel code

- no standard library
 - you need to write everything yourself
 - you will get linked lists from us
 - nothing more complex will be needed

- no standard library
 - you need to write everything yourself
 - you will get linked lists from us
 - nothing more complex will be needed
- cross-compilation
 - build code on architecture A for architecture B
 - packages for Fedora, Ubuntu and Docker images from us

- no standard library
 - you need to write everything yourself
 - you will get linked lists from us
 - nothing more complex will be needed
- cross-compilation
 - build code on architecture A for architecture B
 - packages for Fedora, Ubuntu and Docker images from us
- MSIM simulator
 - deterministic simulation

- no standard library
 - you need to write everything yourself
 - you will get linked lists from us
 - nothing more complex will be needed
- cross-compilation
 - build code on architecture A for architecture B
 - packages for Fedora, Ubuntu and Docker images from us
- MSIM simulator
 - deterministic simulation
- keep to the provided C style and formatting

M02 and onward: work in teams

M02 and onward: work in teams

- teams of 3

M02 and onward: work in teams

- teams of 3
- cooperation on a kernel code

- teams of 3
- cooperation on a kernel code
- learn how to develop and manage team work

- Git is your friend

- Git is your friend
- Git branches are cool
 - systematic naming helps a lot too

- Git is your friend
- Git branches are cool
 - systematic naming helps a lot too
- merge requests can be the central place for discussion

- Git is your friend
- Git branches are cool
 - systematic naming helps a lot too
- merge requests can be the central place for discussion
- issues help to distribute your work

- Git is your friend
- Git branches are cool
 - systematic naming helps a lot too
- merge requests can be the central place for discussion
- issues help to distribute your work
- commit early, commit often

- Git is your friend
- Git branches are cool
 - systematic naming helps a lot too
- merge requests can be the central place for discussion
- issues help to distribute your work
- commit early, commit often
- Forum can be used to find your team members

- combine your printk into one kernel

- combine your printk into one kernel
- add kcalloc and kfree

- combine your printk into one kernel
- add kmalloc and kfree
- again: no standard library ...

- combine your printk into one kernel
- add kcalloc and kfree
- again: no standard library ...
- ... but whole RAM is yours
 - 2MB ought to be enough for everybody :-)

- threads
 - illusion of virtual CPU private to the running code of a single function
 - separate code, stack and register context

- threads
 - illusion of virtual CPU private to the running code of a single function
 - separate code, stack and register context
- threads must yield the CPU

- threads
 - illusion of virtual CPU private to the running code of a single function
 - separate code, stack and register context
- threads must yield the CPU
- assembly for register switching from us

- allocate physical frames

- allocate physical frames
- preparing for paging and virtual memory in M06

- allocate physical frames
- preparing for paging and virtual memory in M06
- split ownership of RAM (frames vs heap)

M05: Preemptive scheduler



- better illusion of virtual CPU: threads do not need to yield

M05: Preemptive scheduler

- better illusion of virtual CPU: threads do not need to yield
- also support for generic exception handling

M06: Virtual Memory Support



- illusion for programs that whole RAM is theirs

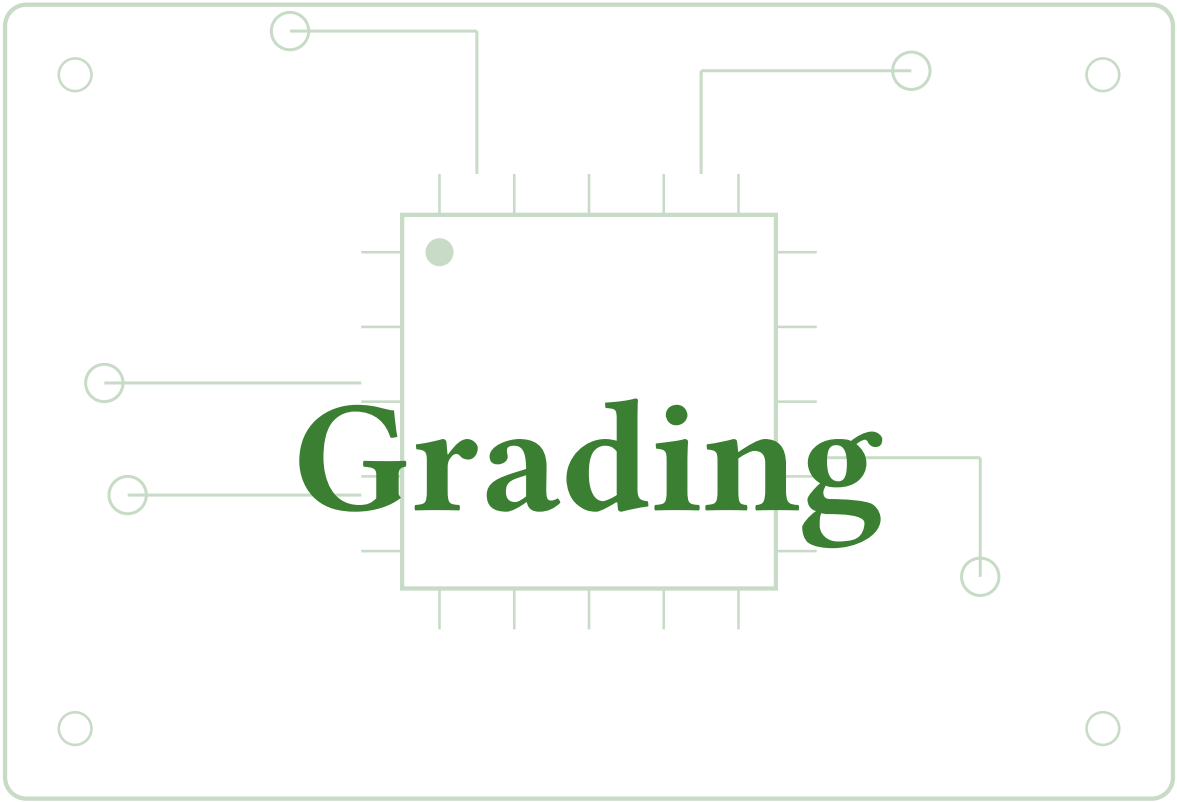
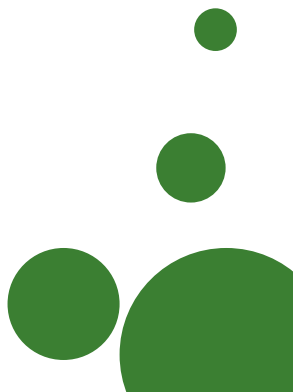
M06: Virtual Memory Support



- illusion for programs that whole RAM is theirs
- needed for M07

- illusion for programs that whole RAM is theirs
- needed for M07
- support for page tables and memory translation
 - CPU works with virtual addresses and translates them to physical for actual access to RAM

- final step: run end user applications on your kernel
 - fully separated from each other
 - illusion of their own CPU and memory
 - kernel handles privileged operations for them



Grading

To pass the course ...

... complete all milestones before mid January (baseline tests must work).

For a better grade

- answer quizzes on lectures (negative points otherwise)
- implement optional extensions before mid January
- implement milestone baselines within (soft) deadlines

Details on course website

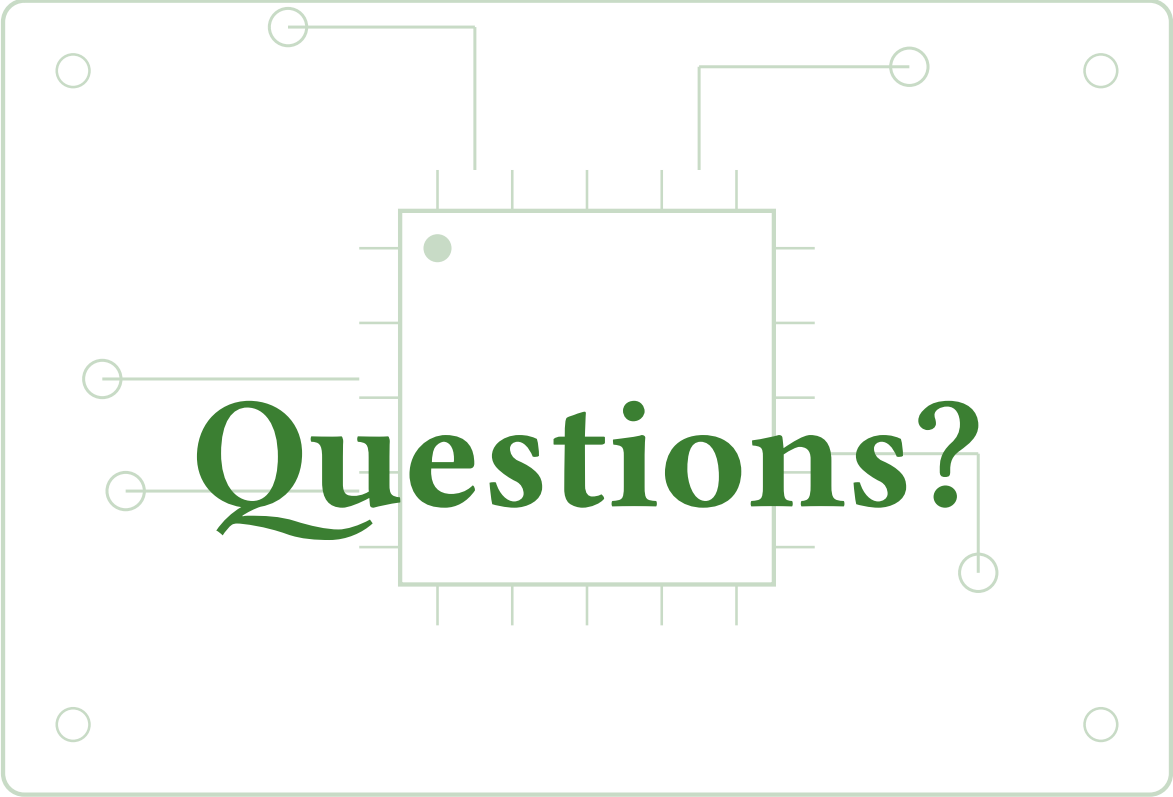
- answer your questions

- answer your questions
- answer your questions

- answer your questions
- answer your questions
- answer your questions

- answer your questions
- answer your questions
- answer your questions
- discuss the milestones

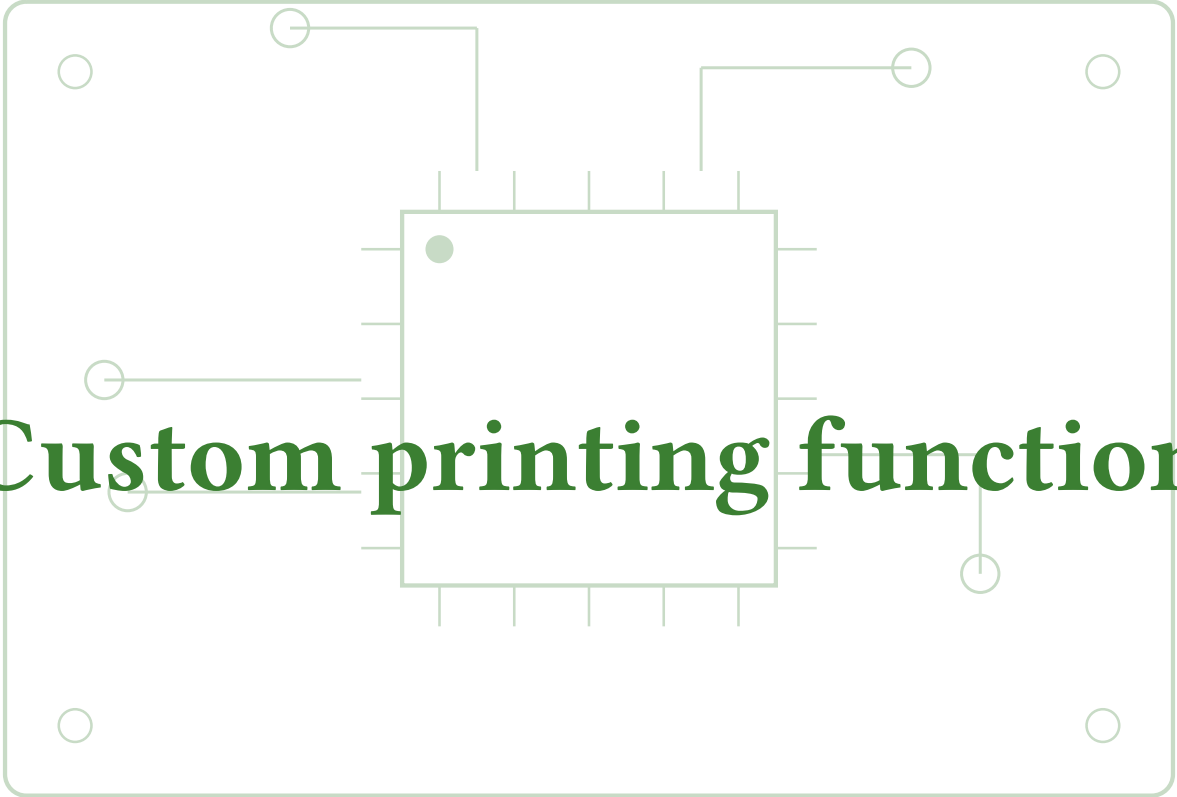
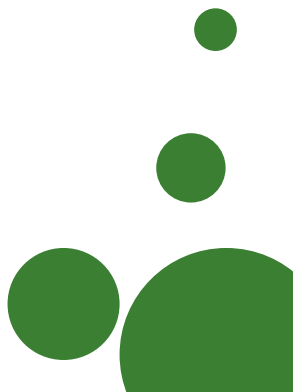
- answer your questions
- answer your questions
- answer your questions
- discuss the milestones
- debug your code (BYOD)



Questions?



- `my_printf`
- using `list_t` and `link_t`



Custom printing function

my_printf(const char* format, ...)



Print the format string, replacing % directives with actual arguments.

```
my_printf("The %s is %d.\n", "answer", 6*9 - 12); // The answer is 42
```

Supported directives:

- characters (char) via %c
- signed integers (int) in decimal via %d
- unsigned integers (unsigned int) in hexadecimal via %x and %X
- zero-terminated strings (char *) via %s
- pointers (void *) in hexadecimal via %p

my_printf(const char* format, ...) II

No need to implement width, precision, length modifiers and other flags.

Extension:

- %pL for printing lists (`list_t`)
- %pT to print `struct timespec *`
- %pB to print byte buffer

Variadic arguments

```
static void print_strings(const char* header, ...) {
    puts(header);

    va_list args;
    va_start(args, header);
    const char *argument = va_arg(args, const char *);
    while (argument != NULL) {
        printf(" - '%s'\n", argument);
        argument = va_arg(args, const char *);
    }
    va_end(args);
}

print_arguments("Start of alphabet", "a", "b", "c", NULL);
print_arguments("No arguments", NULL);
print_arguments("Bad usage", "a", "b", NULL, "never printed", NULL);
```



Why linked lists (and why these)?

Why linked lists (and why these)?

Nothing more complex will be needed or required (this is OS course, not ADS).

Why linked lists (and why these)?

Nothing more complex will be needed or required (this is OS course, not ADS).

Our lists will have maximum of tens (or very low hundreds) of items.

Why linked lists (and why these)?

Nothing more complex will be needed or required (this is OS course, not ADS).

Our lists will have maximum of tens (or very low hundreds) of items.

Do not implement them yourself. Seriously.

Why linked lists (and why these)?

Nothing more complex will be needed or required (this is OS course, not ADS).

Our lists will have maximum of tens (or very low hundreds) of items.

Do not implement them yourself. Seriously.

Switch to more complex data structure only when:

- you have plenty of time
- you have measured it will improve the performance

Why linked lists (and why these)?

Nothing more complex will be needed or required (this is OS course, not ADS).

Our lists will have maximum of tens (or very low hundreds) of items.

Do not implement them yourself. Seriously.

Switch to more complex data structure only when:

- you have plenty of time
- you have measured it will improve the performance

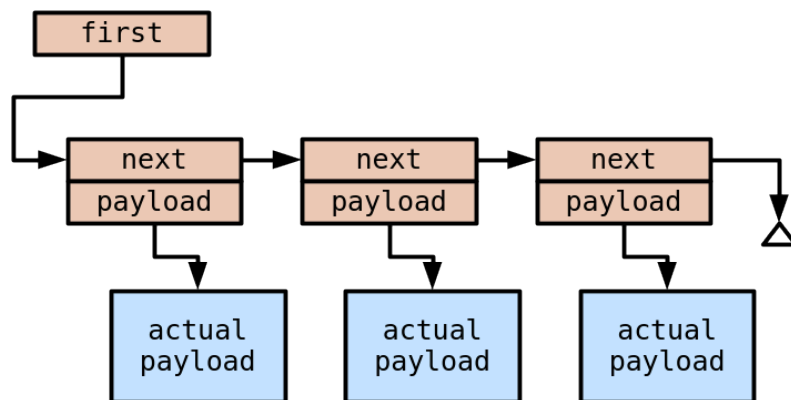
And even then: think twice (and reuse the code from Linux or HelenOS, please).

Typical linked list

```
list_t list;  
list_init(&list);
```

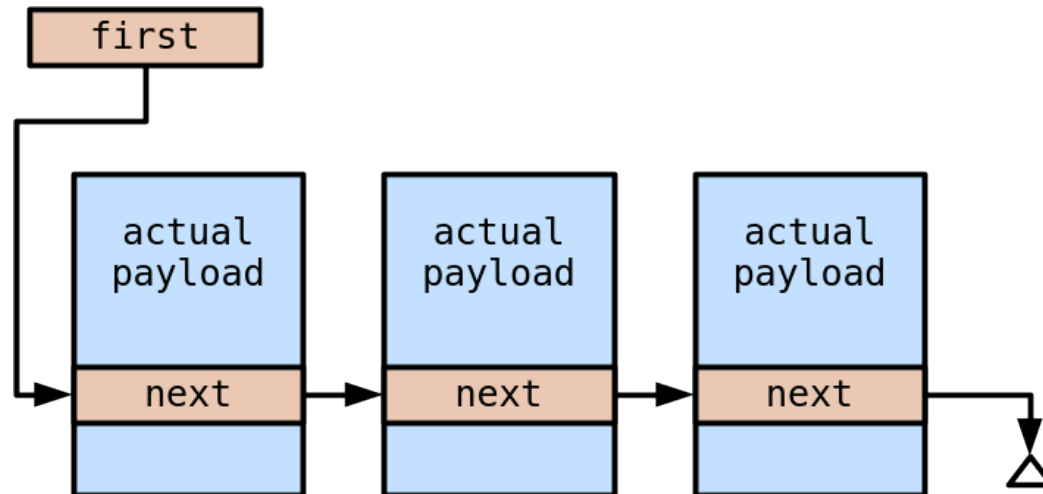
```
payload_t * payload = create_payload();  
payload->value = 42;  
list_item_t * item = create_list_item();  
item->data = payload;
```

```
list_append(&list, item);
```



Embedded kernel-style linked list

```
list_t list;  
list_init(&list);  
  
payload_t * payload = create_payload();  
payload->value = 42;  
link_init(&payload->link);  
  
list_append(&list, &payload->link);
```



Embedded kernel-style linked list



```
list_foreach(list, payload_t, link, it) {  
    printk("Payload %d\n", it->value);  
}
```

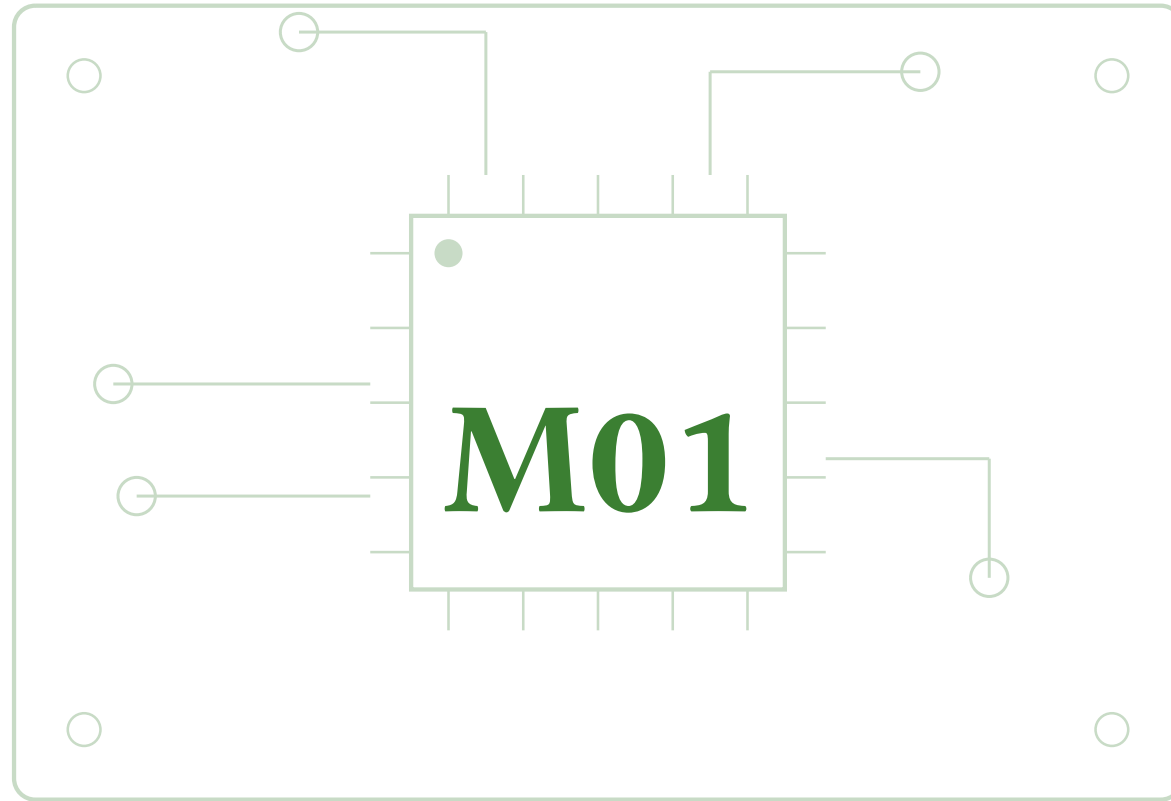
...

```
link_t * first_link = list_pop(&list);  
payload_t * payload = list_item(first_link, payload_t, link);
```

Access to your repositories

Access to Forum

Understanding M00



- transfer `my_printf` to kernel and some othe bits

- transfer `my_printf` to kernel and some other bits
- no standard library

- transfer `my_printf` to kernel and some other bits
- no standard library
- development workflow
 - `./configure.py [--kernel-test=(printf/int|...)]`
 - `make`
 - `msim`
 - repeat or `make distclean` for a different test

- transfer `my_printf` to kernel and some other bits
- no standard library
- development workflow
 - `./configure.py [--kernel-test=(printf/int|...)]`
 - `make`
 - `msim`
 - repeat or `make distclean` for a different test
- running suite `./tools/tester.py suite suites/*.txt`

- transfer `my_printf` to kernel and some other bits
- no standard library
- development workflow
 - `./configure.py [--kernel-test=(printf/int|...)]`
 - `make`
 - `msim`
 - repeat or `make distclean` for a different test
- running suite `./tools/tester.py suite suites/*.txt`
- configuration, cross-compilation and running