To begin with, boot (or reactivate) the Ulix Devel VM and run the command in the shell update-ulix.sh out. This will download the files you need to edit the current
len practice exercises.

8. Paging / Literate Programming

In the folder tutorial03 / In your home directory you will find a version of the Ulix kernel that (as shown in the lecture) supports paging. It lies as a Literate Program (ulix.nw) in front.

a) Change to the folder and extract from the Literate Program ulix.nw With make the source code files ulix.c and start.asm; these are then also automatically translated. Then start the kernel with make run. The system performs some storage management tests and then stops.

Also take a look at the PDF file ulix.pdf (that you post changes to ulix.nw With make pdf can regenerate). You will find the description of the previous one here Codes for frame and page management and thus also the detailed variant of a sample solution to exercise 7 from the last exercise sheet.

It is not necessary to read the rather long document line by line, but you should skim through it and compare the way the code is divided into different code chunks with your own solution.

b) In the lecture we have the Page Table Descriptors and the Page Descriptors as structures (struct) defined and already discussed that alternatively an interpretation as unsigned int is conceivable. In this exercise you will rearrange the Literate Program so that it works with these simple integer types. Work in a copy (so that the original files are kept); to do this, create in the home directory with

```
cp -r tutorial03 tutorial03-copy
```

a copy of the whole directory and make the changes in tutorial03-copy / in front. A few tips on how to do it:

(i) First, change the type declarations for page_table_desc and page_desc in

```
typedef unsigned int page_table_desc; typedef unsigned
int page_desc;
```

(ii) To the guys page_directory and page_table you can do without it altogether and instead use individual directories or tables as

```
page_table_desc pd [1024] __attribute__ ((aligned (4096))); page_desc
                                   pt [1024] __attribute__ ((aligned (4096)));
```

declare. This will give you access to the entry in the future n from pt about pt [n] instead of pt. pds [n] to. With pt (without index) there is a pointer (of type unsigned int *) available at the beginning of the table. To get a single descriptor with functions like fill_ page_desc () or fill_page_table_desc () call pass a pointer to it

pt [n], so & pt [n].

(iii) After the changes, all functions that work with these types will no longer work; So you have to make adjustments each time. To z. B. in fill_page_desc () one
To fill the page descriptor with content, you can first enter the address value frame_addr and its lowest twelve bits 0 put:

```
tmpvalue = frame_addr & 0b11111111111111111111000000000000;
```

or.

```
tmpvalue = frame_addr & 0xFFFFF000;
```

(In the hexadecimal notation you combine four bits in one hex number.) You must then add the bits to be set. You could define flag constants based on the bit positions, something like this:

```
# define FLAG_PRESENT 1 << 0          // Bit 0: present // Bit 5:
# define FLAG_ACCESSED 1 << 5         accessed
```

etc. Then you can use an "or" operation (|) with z. B.

```
tmpvalue = tmpvalue | FLAG_ACCESSED;
```

set the respective bit. When everything is ready, write the value with * desc = tmpvalue; one. (You have to adapt the function prototype and later add a pointer to unsigned int to hand over.)

(iv) To later extract the address from a descriptor, set the lowest twelve bits again (as above) 0. However, to extract a single flag you can use the
Descriptor with FLAG_ * "unden" (&) and the result to == 0 or! = 0 testing:

```
if ((descriptor & FLAG_ACCESSED) == 0) {/ * flag is not set * /}
```

c) Test that the old and new programs work identically. You can do this in both
Programs the function hexdump () use an address range as a hex dump.
gives. Use the start and end addresses of the page table or the page directory as arguments, e.g. B. in the form

```
hexdump ((unsigned int) current_pd, (unsigned int) current_pd + 4096);
```

When the program is called, the outputs of hexdump () in files output.txt (in the respective folder), you can then compare them later:

```
cd ~ / tutorial03 /;                make; make run> output.txt
cd ~ / tutorial03-copy /; make; make run> output.txt cd ~ /; diff ~ / tutorial03 * /
output.txt
```

You should be at diff- Call received no output: Then the two files are identical. (The hexdump Calls have to *to* the

changes to the tables take place.)

d) Check with make pdf, that you will still be out of the Literate Program after your changes
Create a PDF file - if that doesn't work, go to troubleshooting.