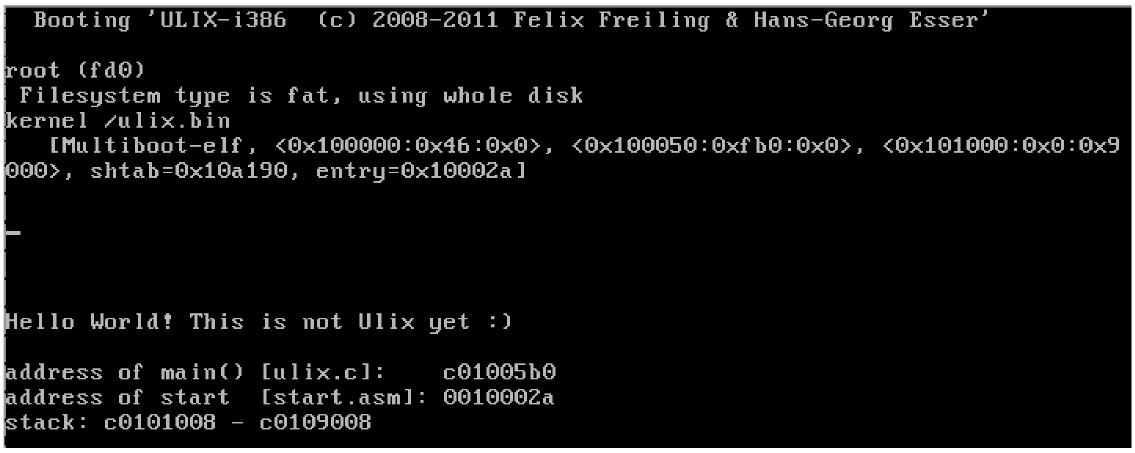
To begin with, boot (or reactivate) the Ulix Devel VM and run the command in the shell update-ulix.sh out. This will download the files you need to edit the current
len practice exercises.

5. Segmentation

In the folder tutorial01 / in your home directory you will find an early version of the Ulix kernel that only initializes the segment descriptors.

a) Read the source code files ulix.c and start.asm. Then translate the source text
With make and start the kernel with make run. You should get the following output:



```
 Booting 'ULIX-i386  (c) 2008-2011 Felix Freiling & Hans-Georg Esser'

root (fd0)
 Filesystem type is fat, using whole disk
kernel /ulix.bin
   [Multiboot-elf, <0x100000:0x46:0x0>, <0x100050:0xfb0:0x0>, <0x101000:0x0:0x9
000>, shtab=0x10a190, entry=0x10002a]


_



Hello World! This is not Ulix yet :)

address of main() [ulix.c]:    c01005b0
address of start  [start.asm]: 0010002a
stack: c0101008 - c0109008
```

b) Obviously the kernel performs the C function Main() out. How does the system get from the as-
sembler code (from label begin) into the C function?

c) The file ulix.dump contains a listing of the generated assembler code. You can find all labels from the assembler file here start.asm, but also the function names from the file
ulix.c. Find the labels in the file start, higher half and Main and look at
which memory addresses the associated code was linked. (The memory addresses are in each case on the far left in hexadecimal notation without the leading " 0x ".) Can you use assembler
Code to recognize the activation of the trick GDT? It is made by a "long jump" (jmp) triggered when the jump address is a logical address of the form Segment: address specified
is. Also, find the labels stack_first_address and stack_last_address and
compare the addresses displayed for this with the output in the VM (last line); they should match.

d) The kernel uses the function printf (), to output text. This function is in the file printf.c implemented, but the code there ultimately uses the function
kputch () (kernel put character), which again in ulix.c is defined. The printf () - Implementation is only concerned with processing the printf- typical format parameters, e.g.% s for strings or% d for numbers. We are interested in how kputch ().

Try to understand how the function kputch () Writing characters on the screen
can. If necessary, google for “0xb8000 video” to find information. As a solution, a short explanation of the use of pointers: With the commands

char * mem; mem = (char *) 0x1234; * mem = 'a';

can you use the byte ' a '(ASCII value: 0x61) in the address 0x1234 of the memory.

e) Why used in kputch () the following line

```
screen = (char *) 0xc0000000 + 0xb8000 + posy * 160 + posx * 2;
```

the factors 160 and 2, and why will 0xc0000000 added?

f) Check with the command objdump -h ulix.bin, which memory areas the three sections . setup, .text and . bss use. (The other sections. comment, .stab and . stabstr you can ignore them.) Compare the values with the information given by the GRUB boot manager when loading the kernel in the line [Multiboot eleven, ...] outputs.

6. Paging

The folder tutorial02 / in your home directory contains the next variant of the Ulix kernel: this time with paging.

a) Read the source code files ulix.c and start.asm and locate the Clause 5 presented code snippets. (ulix.c contains additional code that you did not see in the lecture.)

b) Translate the code with make and start the Mini-Ulix with make run.

c) In ulix.c has the function kputch () changed a little. Here is now the following Code block:

```
if (paging_ready)
     screen = (char *) 0xb8000 + posy * 160 + posx * 2; else

     screen = (char *) 0xc0000000 + 0xb8000 + posy * 160 + posx * 2;
```

This is where the variable paging_ready evaluated that initially false is and after the initialization of the paging true is set. In this case, when calculating the address, the addition of 0xc0000000 away (see Exercise 5d). Why does it work

7. Literate programming

a) Convert the two files ulix.c and start.asm (out tutorial02 /) into a literary man Program called tutorial02.nw. The documentation that you add can consist of key words, and you can use the slide contents as a guide.

b) Test that you have restored the original (or similar, even if successfully compilable) can extract code files.

c) You can also generate a LaTeX file and a PDF file from it. Send them to me nw- file (the Literate Program) and the pdf- Email the file to (→ hgesser@cs.fau.de), I will then give you feedback on the implementation. (This part is voluntary, but recommended.)