Name ______________________ Matriculation no. ______________ Tutorial no. ________

Operating Systems 2013/14 Assignment 1

Prof. Dr. Frank Bellosa
Dipl.-Inform. Marius Hillenbrand
Dipl.-Inform. Marc Rittinghaus

Submission Deadline: Monday, November 18th, 2013 – 9:30 a.m.

A new assignment will be published roughly every two weeks, right after the last one was due. It must be handed in before its submission deadline.
Please print out the pages containing **T-Questions** and answer them on your printout. Clearly mark every page with your name, matriculation number and **tutorial number**. Simply put it in the mailbox in the basement of building 50.34 (Info-Neubau).
P-Questions are programming assignments. Download the provided tarball from the VAB and make sure to use the included templates and Makefiles. Do not fiddle with the compiler flags. Submission instructions can be found in the introductory section below.
In this assignment you will get familiar with the C programming language, particularly with pointers, bit manipulations, and the representation of functions and local variables on the stack.
Any assignment handed in after its deadline will be ignored!

hf & gl :-)

T-Question 1.1: Basics

a. Shifting a given integer value 12 bits to the right is equivalent with a division by factor: **1 T-pt**

____

b. How does multiprogramming increase the utilization of resources? **1 T-pt**

____

____

____

c. How does timesharing extend multiprogramming to provide for interactive computing by several users? **1 T-pt**

____

____

____

Name	Matriculation no.	Tutorial no.

T-Question 1.2: The User-Kernel Boundary

a. Are the following statements true or false? (correctly marked: 0.5P, not marked: 0P, incorrectly marked: -0.5P) **2 T-pt**

true	false	
☐	☐	The trap instruction should be privileged.
☐	☐	Turning off interrupts should be privileged.
☐	☐	A system call is an invocation that crosses protection domains.
☐	☐	System call parameters may be passed via the kernel stack.
☐	☐	System call parameters may be passed in registers.
☐	☐	Devices signal the end of DMA operations with exceptions.
☐	☐	System calls are synchronous to code.
☐	☐	Interrupts are synchronous to code.

b. Why must the kernel carefully check system call parameters? **1 T-pt**

T-Question 1.3: The C Programming Language

a. What is the size (in bytes) of a pointer on a typical laptop today? **1 T-pt**

b. Give an expression that calculates the number of elements in the following array. Use `sizeof`. **1 T-pt**

```
uint16_t array[17];
```

c. Transform the given statements to equivalents that use square brackets. **2 T-pt**

```
double a = *(array + 17); // array declared as double array[10];
int16_t b = *(ptr + 4); // ptr declared as void *;
```

Name ______________________ Matriculation no. ______________ Tutorial no. ________

d. You see several assignment statements. Write the resulting values of `a` into the right column. **3 T-pt**

Assignment (`int a;`)	Value assigned to `a`
`a = 1 && 2;`	
`a = 1 & 2;`	
`a = 2 || 0;`	
`a = 2 | 0;`	
`a = ! 0;`	
`a = 1==0;`	

e. Look up PRId64 & Co. in `inttypes.h`. What would you put in as ??? to complete the code below in a platform-independent way? **1 T-pt**

```
uint16_t i = 23;
printf( "i␣in␣hex␣is␣???", i );
```

f. Briefly explain what the declared variables a to d are.

- `char* a, b;`
- `float (*c)();`
- `int *d[101];`

2 T-pt

About the Programming Assignments

The following introductory words outline our expectations of your work and the requirements your solutions have to fit.

Write Readable Code

In your programming assignments, you are expected to write well-documented, readable code. There are a variety of reasons to strive for clear and readable code: Code that is understandable to others is a requirement for any real-world programmer, not to mention the fact that, after enough time, you will be in the shoes of one of the others when attempting to understand what you wrote in the past. Finally, clear, concise, well-commented code makes it easier to grade your assignment! (This is especially important if you cannot get the assignment running. If you cannot figure out what is going on, how do you expect us to do it?)
There is no single right way to organize and document your code. It is not our intent to dictate a particular coding style for this class. The best way to learn about writing readable code is to read other people's code.
Here are some general tips for writing better code:

- Split large functions. If a function spans multiple pages, it is probably too long.
- Group related items together, whether they are variable declarations, lines of code, or functions.
- Use descriptive names for variables and procedures. Be consistent with this throughout the program.
- Comments should describe the programmer's intent, not the actual mechanics of the code. A comment which says "Find a free disk block" is much more informative than one that says "Find first non-zero element of array".

Write Compilable And Executable Code

Obscure code is bad, but uncompilable code is even worse. Your solution has to compile successfully or you will not get points. To increase your coding awareness, we expect you to use the GNU C compiler with some restrictions on warning-behavior as written in the makefiles. Do not change these flags!
The Fedora OS as installed in the ATIS pool will be the reference platform for controversial cases. If you are unable to write a fully working solution, at least make sure that your partial solution does compile, even though it might not produce the correct result. Document your intents and problems as comments in the source file to give your tutor a head start in understanding your code.

Groups

We assume that you will complete the assignment on your own. Please feel free to discuss your solutions with your colleagues, but do not share code. A plagiarism detection system will be in effect this semester. We also check against last years' submissions whenever possible!

Templates And Stubs

You will find templates for all programming assignments in the VAB. Untar them with the command `tar -xzf asstXX.tar.gz`. The archives contain a directory for each individual task, wherein you can find several files:

<task name>.h A header file defining the function prototypes as listed in the assignment's description. You should not modify this.

<task name>.c Put your solution in here.

main.c Contains the entry point in the resulting program. While we provide trivial test cases, you should write your own test code here. Note: This file will be replaced by your tutor and thus not regarded as part of your solution.

Makefile Call `make` to build your sources.

These templates should ease your work as well as ours, so don't change anything unless explicitly allowed.

Assignment Submission

First of all, remove all the object files and binaries (run `make distclean`), but keep the files needed to compile your solution. Before submission, you should guarantee a well-formed directory structure using the bash script `prep4submission.sh` provided alongside the templates. Make sure that the resulting gzipped tarball is called "asst1_<matriculation no>.tar.gz" and its directory structure matches the following pattern: "asst1/<matriculation no>/".
Please email the resulting file to <**os-praxis-2013@ira.uka.de**> as well as the tutor that was assigned to you via webinscribe before the deadline has passed, otherwise your submission will not be graded. **Include your matriculation number and assignment number in the subject line and make sure that your name is included in your from line.**

P-Question 1.1: More Hexadecimal

You may only modify the files `hex.c` and `main.c`.

a. Write a function that converts from a single hexadecimal digit in a char to an integer. You may use neither a library function nor a lookup table for this task. Return -1 if the parameter is not a valid hex digit **1 P-pt**

```
int hexDigitToInt(char hexDigit);
```

b. Write a function that converts from a hexadecimal string to an integer. Reuse your function from above, but do not use any library function. Handle both strings starting directly with a hex digit and strings starting with 0x. You may assume that the resulting integers fit into an int. Return -1 if the given string is not a valid hex number. **2 P-pt**

```
int hexToInt(char* hexString);
```

P-Question 1.2: Pointers

You may only modify `pointer.c` and `main.c`.

a. Write a function that returns the rounded-towards-zero, arithmetic mean over all values in an array. The array is passed to the function via a pointer to the first element of the array and is bounded by the parameter `size` (`size > 0`). **1 P-pt**

```
int average ( int *arrayPointer, unsigned int size );
```

P-Question 1.3: PowerPC Processor Status Word

You may only modify `msr.c` and `main.c`.

You already know that handling low-level control registers of the hardware is part of an operating system's regular operation. For example, the OS needs to make sure that user applications run in non-privileged mode, whereas a system call, exception, or interrupt is handled by the OS in privileged mode.

In this assignment, you will write functions for modifying (a simplified version of) the PowerPC machine state register (MSR). The MSR is the most important control register of a PowerPC processor and controls whether the CPU is executing in privileged or unprivileged mode, amongst others. We provide documentation of its layout in msr.h. Build your functions from explicit bit manipulation operations (<<, >>, ~, ^, &, and |).

a. Write a function that determines from the MSR whether the CPU is running in 64-bit mode. It should return 1 in that case, and 0 otherwise. **1 P-pt**

```
int is64BitMode ( uint32_t *MSR );
```

b. Write two functions that modify the MSR to enter or leave privileged mode. **1 P-pt**

```
void enterPrivilegedMode ( uint32_t *MSR );
void leavePrivilegedMode ( uint32_t * MSR );
```

c. Sometimes it is necessary to stop a CPU from handling interrupt requests from the HW. Write a function `enableInts` that sets the interrupt enable flag in an MSR and a function `disableInts` that disables the interrupt enable flag in an MSR. **1 P-pt**

```
void enableInts ( uint32_t *MSR );
void disableInts ( uint32_t *MSR );
```

d. Write a function `assembleMSR` that prepares an MSR value and sets the individual fields as specified by the parameters. **2 P-pt**

```
uint32_t assembleMSR ( uint8_t compMode, uint8_t intEnable,
uint8_t problemState, uint8_t fpEnabled);
```

P-Question 1.4: Pushing Bits Around

You may only modify `bits.c` and `main.c`.

a. Given a large array `A` (e.g., 1 MB), write three functions `getN`, `setN` and `clrN` that return, set or clear the n'th bit (not byte!) in the array respectively. The first bit (means 0'th bit) should be the most significant bit of the first byte so the physical mapping is contiguous in memory. You can find an example program that demonstrates how bitshifts, functions, and such work and further gives you an example of indentation/comment style (sudoku.c) in the VAB alongside this assignment. **3 P-pt**

```
int getN ( uint8_t *A, unsigned int n );
void setN ( uint8_t *A, unsigned int n );
void clrN ( uint8_t *A, unsigned int n );
```

b. Write a function that rotates the bits of a 64 bit integer `n` bits to the right. Bits rotated "out" of the integer shall be rotated "in" on the other side. Keep in mind that `n` may be negative which shall rotate to the left. **2 P-pt**

```
void rot ( uint64_t *i, int n );
```

P-Question 1.5: Stack Magic

a. Calling a function builds a new "frame" on the stack, saving the state of the calling function (caller). It is also used to return results of the called function (callee) back to the caller.

   We provided an assembler version of the following code (`stack.s`) compiled with `gcc -O0 -S` on a 32-bit x86-machine. Use that code instead of own compilations to avoid irritations of differing platforms. Nevertheless, we encourage you to comprehend the compilation on your computer and watch potential differences. You can find a short description of how to read assembler code at [1].

   Annotate each line of the assembler code with a comment that explains why it is executed (e.g., write "save old stackpointer" rather than "push esp on stack").

   [1] `http://os.ibds.kit.edu/downloads/lehre_ss2010_mkc_ia32condensed.pdf` **4 P-pt**

```
#include <stdint.h>
uint64_t multiply ( uint32_t x, uint32_t y )
{
    uint64_t res;
    res = x * y;
    return res;
}
```

```
int main ()
{
    uint32_t a = 3, b = 5, z;
    z = multiply( a, b );
    return 0;
}
```

Total:
16 T-pt
18 P-pt