University of Karlsruhe
System Architecture Group
Gerd Liefländer

Teaching Assistant:
Philipp Kupferschmied

System Architecture 2008/09
Assignment 1

In addition to the slides of the first lecture(s), also have a look at the slides "History of Operating Systems" and "Hardware (P)Review", which are both available for download on the course website.

The quality of the tutorials depends on you and your active participation. The tutorials are given in German. You should be able to discuss the problems and evaluate their solutions by yourselves, so please prepare carefully. We have asked our tutors to give only small hints rather than complete solutions.

Question 1.1: Function Calls and Stacks

Let us study stacks and procedures in C.

1. Discuss the following code fragment:

```
double foo(int *p) {
    int x;
    double y;
    x = *p;
    /* do something useful */
    return (y);
}

double bar(void) {
    double d;
    int i = 42;
    d = foo(&i);
    return (d);
}
```

   Try to visualize the stack contents before, during, and after the execution of `foo`. An `int` is 4 bytes and a `double` is 8 bytes long. Assume a 4-byte aligned, downwards growing pre-decrement stack and the existence of a stack-frame pointer (Stapelschachtelzeiger). All values are passed via the stack between calling and called function. All local entities within a function are addressed relative to the frame pointer.

2. Some authors claim that parameter passing via registers or main memory is either too special or too difficult. Discuss the pros and cons of these kinds of parameter passing.

3. What kind of *operating system components* are implemented as functions?

4. Characterize as briefly and precisely as possible the difference between a function (i.e., a closed subroutine) and a macro (i.e., an open subroutine). With regard to the callers, outline the consequences of these two different implementations.

Question 1.2: Famous System Architecture Bugs

1. Use the web to find the *real reasons* behind the famous mars pathfinder problem.

2. Use the web to find *other bad examples of system engineering* (not yet mentioned in the lecture).

Question 1.3: Terms and Definitions

1. What are orthogonal design parameters? Can you find examples for design parameters that are non-orthogonal?
2. Explain the difference between *policy* and *mechanism*.
3. Enumerate the major tasks of an operating system.

Question 1.4: Orthogonal Design Parameters

1. Assume an OS for a PC. Try to find at least *ten different software system components*. Explain as short as possible their functionality. Try to use commonly used OS terms.
2. Take one of these subsystems and try to discuss its *orthogonal design space* (orthogonal design parameters).
3. Try to find another ten hardware components with each one related to the above ten system components. Take one of these hardware components and try to discuss their orthogonal design parameters.

Question 1.5: Polling I/O

Suppose you have a CPU running at 400 MHz. One polling operation costs 400 cycles. Calculate the CPU overhead with polling for the following devices:

1. Mouse with a polling rate of 3 000 Hz.
2. Floppy disk, data transfer of 16 bit per poll at 50 kB/s.
3. Hard disk, data transfer of 32 bit per poll at 8 MB/s.

Remark: For the disk devices you need to choose a polling rate so that no data will be lost.
Discuss the range of an appropriate polling-rate for the mouse device.

Question 1.6: Paper Review

Read the paper [1]. This paper will be discussed in the tutorials. Some points that might be worth being discussed:

- What characterizes *ambitious systems*, and in howfar can these characteristics lead to errors?
- Where does the complexity of developing ambitious systems come from?
- What went wrong with CTSS?
- What possibilities exist to tackle the complexity problem?

There will be no "solutions" published for this question!

References

[1] Fernando J. Corbató. On building systems that will fail. *Commun. ACM*, 34(9):72–81, 1991.