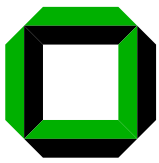

University of Karlsruhe
System Architecture Group
Gerd Liefländer

Teaching Assistant:
Philipp Kupferschmied

System Architecture 2008/09
Assignment 3

Question 3.1: Crucial OS Terms and their Relations

Relate the following terms to each other: thread, address, address space, process, task, single-programming system, multi-programming system, single-user system, multi-user system.

Question 3.2: Single/Multi Programming/User Systems

1. Discuss the advantages of multi-programming systems over single-programming systems.
2. What are typical combinations of single/multi-programming and single/multi-user systems?
3. Describe the concept of a virtual machine.

Question 3.3: Privileged Software

1. Many systems distinguish between non-privileged and privileged software. Why? Enumerate typical examples of both types.
2. What are typical examples of privileged instructions? Why are they privileged?
3. Consider a typical PC operating system (e.g., Windows XP or Linux). Why would a clever system architect try to implement some of the OS functionality outside the OS kernel? Which OS functions would you consider prime candidates?

Question 3.4: Syscalls: The User/Kernel Boundary

1. Enumerate three popular POSIX system calls that allow to deal with activities.
2. System calls enable the transition from user level to kernel level. Why do we have to be very careful when designing this transition?
3. Some systems try to enhance kernel protection by introducing an extra kernel stack instead of using the application's stack whilst performing a system call. Does this technique really improve kernel protection? Explain!

Question 3.5: System Structures

1. Compare systems based on a monolithic kernel with microkernel-based "multi-server" operating systems. What are the respective strengths and weaknesses?
2. Assume an application wants to execute a `read` system call that reads some bytes from a file into a buffer in memory. Compare the overhead caused by kernel entries/exits and address space switches of this system call when it is implemented
   - as part of the API of a monolithic kernel
   - as a function of a file server running on top of a microkernel.