University of Karlsruhe
System Architecture Group
Gerd Liefländer

Teaching Assistant:
Philipp Kupferschmied

System Architecture 2008/09
Assignment 5

The lecture on **Wednesday** will take place in the *Tulla* lecture hall for the rest of the semester.

Question 5.1: Switching Threads

1. Thread switching for both PULTs and KLTs can be implemented by preserving the previous register state on the stack, exchanging the stack pointers, and loading the new state from the new stack. In GCC for IA32, this may look as follows:

```
tcb_t *current;

#define SWITCH_TO(next) do { \
    asm volatile ( \
    "pushal"                    /* push all registers onto the stack */ \
    "movl %esp, %[old_esp]"     /* store old stack pointer in prev TCB */ \
    "movl %[new_esp], %esp"     /* load new stack pointer from next TCB */ \
    "popal"                     /* pop all registers from the stack */ \
    : [old_esp] "=m"(current->esp)  \
    : [new_esp] "m" ((next)->esp) \
    ); \
    current = (next); \
} while (0)
```

   In the lecture, there was always a single central *function* to provide this code, which was called when needed:

```
void threadSwitch(tcb_t *next) {
    SWITCH_TO(next);
}
```

   What would happen if you used the macro `SWITCH_TO` directly in various places (e.g., also in `altThreadSwitch`) instead of calling `threadSwitch`?

```
void altThreadSwitch(tcb_t *next) {
    int i;
    SWITCH_TO(next);
    for (i=0; i != 10; i++) { foo(3); }
}
```

2. Is it really necessary to have one kernel stack per KLT, or is is one kernel stack for all threads sufficient?

Question 5.2: Scheduling Basics

- What kind of hardware support is required for an operating system that implements a *non-cooperative* scheduling policy?
- How is control transfered to the operating system when an interrupt occurs?

- When a kernel level thread performs a blocking system call, the operating system has to select another thread to run. Why might it be beneficial to prefer threads in the same address space? What problem can arise in that case?
- Consider a process that wants to read from a file on disk by means of a blocking `read` system call. Outline the sequence of OS invocations and scheduling decisions, from the invocation of the system call until it completes.

Question 5.3: Task and Thread States

- Consider a system that distinguishes the thread states *blocked. ready*, and *running.* Enumerate the events that lead to transitions between these states.
- Consider a multithreaded application that either uses KLTs or PULTs. Which of the following combinations of task state and thread state can occur?
  1. task state: *ready*, KLT state: *running*
  2. task state: *blocked*, KLT state: *ready*
  3. task state: *ready*, PULT state: *running*
  4. task state: *blocked*, PULT state: *ready*
- Is it really necessary to store the state of a a task in addition to the state of each KLT that belongs to the task?

Question 5.4: Single and Multi-Threading

Assume you are working for a software development company that also has a web-server in its product portfolio. One day, a customer who uses this webserver complains that three out of four CPUs of his new server machine are idle when the server software runs. By having a look at the source code, you find out that the server software doesn't make use of threads. It is your job to improve this situation.

- First of all, you have to decide what kind of threads to use. Assume your customer uses an operating system that offers kernel-level threads, but that you have also found a fast and leightweight PULT library. Which solution do you prefer?
- After you decided what thread model to use, you remember what you have learned in the system architecutre lecture: It is not trivial to make a single-threaded application multi-threaded, as some constructs that worked well in the single-threaded case will cause problems in a multithreaded scenario. Enumerate problems that can arise.
- Finally, you have to think about how a web-server can benefit from multiple threads (or if it can benefit at all). Discuss different implementation possibilities (e.g. how many threads are created, what work do the threads do, how long do they live ...).