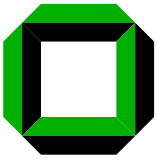

University of Karlsruhe
System Architecture Group
Gerd Liefländer

Teaching Assistant:
Philipp Kupferschmied

System Architecture 2008/09
Assignment 4

Question 4.1: Processes in Unix

1. What does the `fork` system call do?
2. *Copy-on-write* can be used to reduce the overhead of forking a process. How does this technique work?
3. Assume you have to write a shell that can be used to launch arbitrary other programs. Is the `fork` system call sufficient for that purpose?
4. Compare `fork` with the `CreateProcess` system call of the Windows API. In howfar do these two system calls differ? What are strengths and weaknesses of each approach?

Question 4.2: Thread Models

1. Compare the three thread models: pure user-level threads, pure kernel-level threads (not to be confused with kernel mode threads!), and hybrid threads. Point out advantages and limitations of each thread model.
2. Upon entry to the kernel, some systems switch to a *kernel stack*. Recall the benefits of this model as opposed to using the user-level stack also during kernel operations.
3. How can such a kernel stack be located? Whence can we obtain the address of the kernel stack to be used subsequently?
4. What is the difference between a kernel-level thread and a kernel-mode thread? What is the motivation for and purpose of the latter?

Question 4.3: Value of PULTs

Discuss the following statement: "Jobs are either I/O-bound or compute-bound. In neither case would user-level threads be a win. Why would one go for pure user-level threads at all?"

Question 4.4: Threads

1. How can concurrent activities interact/communicate in a (local) system?
2. Which types of events can trigger a KLT switch?
3. Which types of events can trigger a PULT switch?

Question 4.5: TCBs

1. Discuss what information must be contained in a Thread Control Block. Which additional (optional) information might also be contained in a TCB?

2. Thread control blocks (TCBs) in kernel space or in a user-level thread library can be arranged (a) in an array, (b) in a linked list, (c) a tree-structure, or (d) in a combination of these.

   (a) Which data structure is more appropriate for the following frequent operations:
   - create new thread/allocate TCB
   - retrieve TCB by thread ID (e.g., for IPC)
   - find next runnable thread
   - kill a thread and all of its children

   (b) Which structure is suitable for embedded devices?

   (c) Which is best for typical desktop systems?