Department of Computer Science and Engineering
Chalmers

Exam in Unix Internals EDA203/DIT681

DAY: 2012-05-22 TIME: 14.00-18.00 PLACE: V building
==========

Examiner: Arne Dahlberg

Questions during exam: Arne Dahlberg, 772 1705

Solutions: No solutions will be posted

Grading Policy: EDA203 3: 30-38, 4: 39-47, 5: 48-60
DIT681 G: 30-47, VG: 48-60

Aids during the exam:

- McKusick, Neville-Neil: The design and implementation of the FreeBSD operating system.
- An English language dictionary.

Instructions:

- Start answering each assignment on a new page; number the pages and use only one side of each sheet of paper.
- Justify all answers. Lack of justification can lead to loss of credit even if the answer might be correct.
- No references to the text book is allowed and copying of text from the book is not allowed.
- If you make any assumptions in answering any item, do not forget to clearly state what you assume.
- Write clearly! If I cannot read your solution, I will assume that it is wrong.

Problem 1 (10p)

a. Describe how the ULE scheduler works. (6p)

b. Describe how the load balancing in the ULE scheduler works. (4p)

Problem 2 (10p)

a. The page replacement algorithm in FreeBSD is an approximation of *least actively used*. This algorithm uses a reference counter in every page frame. Describe how this reference counter is initialized and updated. (2p)

b. How is it possible that it can be more than one referenced bit set for a page frame at the same time. (2p)

c. In most cases the buffer cache in FreeBSD use page frames to store the data blocks, but there are some exceptions. Explain why page frames are not always used. (2p)

d. When the *bread()* routine is used to look up a buffer in the buffer cache, the returned buffer is also mapped into kernel virtual memory. What is the reason that the returned buffer is mapped into kernel virtual memory? (2p)

e. How can the kernel determine if a file block is already present in the buffer cache? (2p)

Problem 3 (10p)

a. FreeBSD has two types of mutex operations. One type that sleeps and one type that spins. Describe how *mtx_lock_spin()* and *mtx_unlock_spin()* works. (3p)

b. Explain the problem that could happen on a SMP (Symmetric Multi Processor) if a process holding a spin lock was allowed to be interrupted. (2p)

c. Explain why spin locks cannot be used on a single processor machine. (2p)

d. Describe the method used by FreeBSD to prevent deadlock when more than one data structure need to be locked. (3p)

Problem 4 (10p)

a. NFS may have problems with nonidempotent operations. Give an example of such a problem. How is the problem with nonidempotent operations solved in FreeBSD? (2p)

b. NFS implemented according to version 2 protocol may have very bad write performance. How is this problem solved in the FreeBSD implementation of NFS? (2p)

c. In NFS version 3, UDP may be replaced by TCP as the transport protocol. Give at least two reasons for this change. (2p)

d. When running an NFS server, an extra server process called *portmap* is needed. What is the reason that the *portmap* server is needed. (2p)

e. It is usually a goal that a distributed file system shall have the same semantics as a local file system. In NFS there is a clear deviation from this concerning the position of the super user. Explain the difference and give a reason for it. (2p)

Problem 5 (10p)

a. What is meant with an association when talking about network communication? (1p)

b. Describe how the implementation is done to allow for the correct association to be found when a packet is received. Also explain which data structures that are used. (4p)

c. The TCP protocol want to avoid that packets sent by TCP need to be fragmented at the IP level. Describe the method used in TCP to ensure that no IP level fragmentation is needed. (5p)

Problem 6 (10p)

a. Assume that a UDP packet arrives on an Ethernet. Describe the handling of this packet from it is received by the Ethernet interface till it is queued at the UDP socket. (Description of handling at link, network and transport layer is needed) (8p)

b. How do the ethernet input routine know that the packet in part a. shall be delivered to the *ip_input()* routine (and no other input routine)? (2p)