Exam in Operating Systems 2011-05-28

Inga hjälpmedel!

Examinator: Jonas Skeppstedt, tel 0767 888 124

30 out of 60p are needed to pass the exam.

1. (10p) Three reasons the kernel of an operating system starts execution are:

   (a) (3p) Exceptions

   (b) (3p) Interrupts

   (c) (3p) System calls

   Explain what they mean and what happens when they occur. (1p) For failed system calls, how can the kernel make sure the process' variable `errno` gets the proper error code despite the kernel cannot know the virtual address of that variable?

   Answer: *All three make the processor state switch to supervisor mode (unless it not already was in supervisor mode, of course).*

   - *Exceptions occur in the pipeline/MMU.*
   - *Interrupts are signals from an external device which interrupt the current thread of execution.*
   - *System calls switch to the kernel through the use of a special instruction in order to switch to supervisor mode and enter the kernel.*

   The kernel can put the error code in the normal return value register, and then instead of doing a normal return to the library function which executed the system call instruction, it can return to instructions which writes the error code in the `errno` *variable and then the* -1 *value in the return value register to indicate a failed system call. Since this code is part of the process' address space, it knows the address of* `errno`*.*

2. (10p) UNIX users make frequent use of pipes. What are pipes used for and why would the shell need to create it — instead of the processes actually using the pipe? What is meant by ”closing” a pipe and why is it important that the shell does that — what will happen otherwise?

   Answer: *See slides for what pipes are. The reason the shell and not the processes create the pipes is that they should not know anything about this but simply read from standard input and write to standard output.*

Closing the pipe means closing the file descriptor representing the pipe. This has the effect that the number of processes which have the pipe open (for either reading or writing depending on which descriptor was closed) decremented by one. Not doing so results in the kernel thinking that the remaining process with the pipe opened for writing might write to it in the future — and this results in the reading process never getting the end-of-file.

3. (20p) Virtual memory.

   (a) (1p) For how many years have there been commercial machines with virtual memory? Hint: the correct answer is a multiple of ten.

   Answer: *50 years.*

   (b) (1p) Which company made the machine in the previous question? Hint: it's not IBM.

   Answer: *Burroughs*

   (c) (4p) Explain what a page table is and how it can be implemented.

   Answer: *A page table contains mappings from virtual to physical pages and information about whether the page was recently accessed, whether it has been modified after it was copied to RAM, whether it is readonly, whether the physical page number refers to RAM or is a number identifying the page on a swap device. The page table can be, and often is, implemented as a multi-level table where the virtual page number is divided into e.g. three part. Each part is used to index an array. The leftmost (most significant bits), or high, part index a first array. Each indexed array element contains a pointer to a new array which is indexed by the next part of the virtual page number. Due to locality of references, this approach saves memory since not the complete address space needs page table entries but only used pages.*

   (d) (4p) Explain what a TLB is.

   Answer: *To avoid having to look in the page table for a virtual to physical translation, two hardware caches are used, one for each of instruction and data accesses. These caches are called translation lookaside buffers, or TLBs, and contain similar information as the page table entry. The TLBs are usually fully associative, and managed either directly by hardware or by the kernel.*

   (e) (4p) Explain why there is a need for **two** TLB's in a modern processor.

   Answer: *Too increase performance there are two TLB's as explained above.*

(f) (6p) Explain what happens when a process wants to read data which is located only on the swap. You need to explain additional parts of a virtual memory system than those mentioned in the question!

Answer:

- *The process requests a translation from a TLB which is not found. Either hardware or the kernel will check the page table for the translation.*
- *A page fault is detected and now, if not already involved, the kernel must fix the situation. The kernel takes a RAM page from a list checks which process is using it. If the page was modified it must be written to the swap device. To find where on the swap device it should be written, the core map entry of this physical page is inspected.*
- *When the previous owner's page table entry has been updated the new page page table is set to the owner of this page in the core map. Where the page was stored in swap can also be written in the core map entry.*
- *Then the page is fetch from swap and the page table entry is updated.*
- *The translation can be put in the TLB entry and the instruction can be re-executed.*

4. (15p) File systems.

   (a) (5p) How can it be that file reads are more frequent than file writes but that disk writes are more frequent than disk reads?

   Answer: *The caching of files in memory usually hit but the modified files must be written to disk so the reads are often avoided while writes can only be avoided if exactly the same block is overwritten before it reached the disk.*

   (b) (5p) How are modern file systems designed to exploit this fact in order to improve the disk write performance? Give at least one example of such a file system.

   Answer: *For instance EXT3 makes the writes in a journal which avoids time consuming moving of the disk write head. After data has been written to the journal, it is copied to its correct locations in the file system, but this copying is not time-critical.*

   (c) (5p) Two ways to reduce the number of system calls for accessing a file are to

   - memory map the file using `mmap`, or
   - use the C library call `setvbuf`

   For which file access patterns is each most suitable and why? That is, which file access patterns are suitable for `mmap` and which are suitable for `setvbuf`? (they might be the same...)

   Answer: *Normal buffering, possibly with a larger buffer set by* `setvbuf` *works well for sequential accesses, and* `mmap` *can additionally be useful for more irregular accesses since it can eliminate the need for using the* `lseek` *system call. By mapping a file to virtual memory, instead of* `lseek` *calls, simple pointer arithmetic is sufficient.*

5. (5p) What are UNIX signals and why is it important to return from a signal handler via the kernel? What happens if the signal handler never returns — either by invoking a function to perform the rest of the program's work or by invoking longjmp? Hint: the effect of both is the same.

   Answer: *UNIX signals are a way for the kernel to inform a process of an event. As a response to being sent a signal, a function, called a signal handler, is executed. During the time the process executes the signal handler, that signal is blocked. If the signal handler never returns that signal will remain blocked. Therefore signal handlers return not to the program but to the kernel which can unblock the signal and then return to the program.*