UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

OPERATING SYSTEMS

Tuesday 1st May 2012

14:30 to 16:30

Year 3 Courses

Convener: K. Kalorkoti
External Examiners: K. Eder, A. Frisch, J. Gurd

INSTRUCTIONS TO CANDIDATES

Answer any TWO questions.

All questions carry equal weight.

CALCULATORS MAY NOT BE USED IN THIS EXAMINATION

1. (a) Give an outline of the structure of the traditional Unix filesystem. You may write in either prose or bullet-point form. *[8 marks]*

In the remainder of this question, we will consider some issues in the design of a *distributed file system*, using as an example the AFS deployed on DICE. AFS has the same general layout as Unix; in particular it uses a Unix-style directory hierarchy.

In answering questions, you may wish to draw on your knowledge from any part of the course.

AFS is a *server–client* filesystem: files live on a dedicated server machine, and are accessed by other client machines (known as *Virtue*). When a user process on a client accesses a file, an AFS client process called *Venus* on the client machine intercepts the request and handles it by talking to the remote AFS server (*Vice*).

(b) i. Explain briefly what happens when a process calls `open()` on a normal Unix file. *[2 marks]*

ii. Discuss how you would expect the `open()` of an AFS file to differ. *[2 marks]*

Obviously, fetching a file from a remote server is much slower than reading a file from disk. Therefore Virtue maintains a local cache of all the data requested from Vice.

(c) There are two main possibilities for caching. Either Virtue could maintain a cache of blocks within files, like the disk i/o subsystem's internal cache; or it could cache entire files at once, for example by copying the remote file over to a local copy inside the `open()` call.

Suggest one or two arguments in favour of each approach. *[4 marks]*

(d) AFS chooses the second approach, caching the entire file at `open()` time. It transfers the modified file back to the Vice server at `close()` time.

Suppose two separate Virtue clients are accessing the same Vice file, and both modify it. Assuming there is no locking at the client level, what do you think the correct result is when the second client does `close()`? Does your opinion have any implications for the design of the Vice server process? *[4 marks]*

Virtue does not usually discard the cached copy when the user process terminates; it keeps it for the next use on that client. Therefore, Virtue needs to know when the Vice file is modified by another client, so it can refresh the cache.

(e) Using your knowledge of other caches in operating systems, discuss possible strategies for ensuring that Virtue knows about modifications. Evaluate their relative advantages. *[5 marks]*

2. (a) Outline the basic structure of a modern virtual memory system. [*6 marks*]

Consider the following (highly simplified) data structures, presented in a C-style notation.

```
/* ascb - control block representing a userspace address space */
struct ascb {
  short asid; /* address space identifier */
  int lock; /* lock word for mutex on this ascb */
  struct page1table *page1; /* pointer to first-level page table */
};

/* first level page table - contains list of second-level page tables */
struct page1table {
  struct pte1 table[4096]; /* array of 1st level page table entries */
}

/* entry in a first level page table */
struct pte1 {
  bool valid; /* true if this entry points to a 2nd level table */
  bool ondisk; /* true if the pointed-to 2nd level table is itself
                  paged out */
  struct page2table *page2; /* pointer to a 2nd level table */
}

/* second level page table */
struct page2table {
  struct pte2 table[4096]; /* array of 2nd level page table entries */
}

/* entry in a second level page table */
struct pte2 {
  bool valid; /* true if this entry points to frame or slot */
  bool ondisk; /* true if the pointed-to 2nd level table is itself
                  paged out */
  void *frame; /* address of real memory frame for this page, or
                 location on disk */
}
```

QUESTION CONTINUES ON NEXT PAGE

QUESTION CONTINUED FROM PREVIOUS PAGE

(b) Write pseudo-code using these data structures that shows what the hardware memory management unit does in order to translate a virtual memory address into a real memory address, assuming a 32-bit memory architecture and a standard paging system. Assume also that the global variable `pagetable` contains a pointer to the currently active first level page table. If you need to use bitwise operations that you do not know how to write in C, use your own notation and explain it, or use English. Ignore the existence of a TLB. [*8 marks*]

(c) Most operating systems allow the creation of memory that is shared between address spaces. What fields would you expect to add to the above data structures in order to allow this, and how would they be used? [*6 marks*]

(d) The ASCB record includes a lock field to allow mutex on the address space data structures. Discuss which virtual memory related operations should obtain the lock (either shared or exclusively). [*5 marks*]

3. (a) A *semaphore* is a commonly used abstraction for mutual exclusion. Explain how a basic integer semaphore is used, specifying the methods it provides to its users. [*4 marks*]

   (b) Show how to implement an integer semaphore using a low-level mutual exclusion primitive such as test-and-set. Use either clear English or pseudo-code. [*6 marks*]

   (c) The *multiple-readers–single-writer (MRSW) lock* is another common abstraction, used in many operating systems. Explain how it is used, pointing out its differences from a semaphore. [*4 marks*]

   (d) Show how to emulate an MRSW lock with one or more semaphores, explaining any additional data structures you need. [*6 marks*]

   (e) Some programming languages, such as Java, have a mutual exclusion primitive specified as part of the language. If you were designing such a language, which mutex primitive would you choose to include, and why? [*5 marks*]