UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

OPERATING SYSTEMS

Thursday 28th April 2011

09:30 to 11:30

Year 3 Courses

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

INSTRUCTIONS TO CANDIDATES

Answer any TWO questions.

All questions carry equal weight.

CALCULATORS MAY NOT BE USED IN THIS EXAMINATION

1. In this question, pseudocode is presented in C style, which you should also use in your answers. Recall that in C `&vble` means 'the address of the variable `vble`' and that `*addr` means 'the contents of the memory location at address `addr`'. Answers are assessed only on the logic of the code, not on syntactic details.

   (a) State the criteria to be satisfied by solutions to the mutual exclusion problem. [*5 marks*]

   (b) Recall that a `TestAndSet` function operates on a word of memory at address `addr` as follows: it reads `*addr` (the contents of `addr`), sets `*addr` to 1, and returns the previous value, all in one atomic operation. Show how to use `TestAndSet` to protect a section of critical code shared between several processes. [*3 marks*]

   Many hardware architectures have an alternative atomic instruction called `CompareAndSwap`. Its behaviour is described by the following C pseudo-code:

```
bool CompareAndSwap(int *addr, int oldval, int newval) {
  if ( *addr == oldval ) {
    *addr = newval;
    return TRUE;
  } else {
    return FALSE;
  }
}
```

   all of which happens atomically as seen by other processors.

   (c) Show how to protect a section of critical code using `CompareAndSwap`. [*3 marks*]

QUESTION CONTINUES ON NEXT PAGE

QUESTION CONTINUED FROM PREVIOUS PAGE

Now consider a standard linked list of integers:

```
typedef struct {
  int data;  /* data stored in this node */
  node *next; /* pointer to the next node in the list */
} node;

typedef node *list;
```

Let `mylist` be a global, shared, variable containing a list. We can insert a new data element `d` at the front of the list by `insert(&mylist,d)`, where `insert` is the following standard procedure:

```
void insert(list *list_ptr, int d) {
  node *newnode_ptr = malloc(sizeof(node));
  newnode_ptr->data = d;
  newnode_ptr->next = *list_ptr;
  *list_ptr = newnode_ptr;
}
```

(d) Show, by considering two processes each calling `insert` on the same list, that this code is not thread-safe in multiprocessing systems. [*3 marks*]

(e) Using `CompareAndSwap`, modify the `insert` code so that it is thread-safe, **without** introducing any other *shared* variable. Assume that addresses are actually integers, so that pointers can be used as values in `CompareAndSwap`. (You need not write casts.) [*5 marks*]

(f) It has been claimed that because of the high overheads involved in traditional locking mechanisms, and the great frequency of access to shared data structures, operating systems should be designed so as to use, wherever possible, data structures that can be atomically manipulated without external locks, as in the previous example.

Discuss this claim briefly, giving your opinion on its validity, and indicate how you would determine more objectively whether the claim has a basis in reality. [*6 marks*]

2. (a) What is a *Process Control Block* (PCB)? What information is typically stored in it? [*8 marks*]

A *signal* is a facility provided by most operating systems, and is the user-level equivalent of an interrupt. In the Unix version, a signal of a particular type is sent to a process by calling the system call `kill(process-id,signal-type)`. A process can set up a *handler* function by calling `signal(signal-type,handler)`. From the user's point of view, when a process receives a signal, normal execution is paused and the handler is called; when the handler finishes, normal execution resumes.

(b) Suggest how the `signal` system call might be implemented. Describe any additional fields you would expect in the PCB to support the signal mechanism. [*5 marks*]

(c) Describe what you would expect to happen inside the OS when a process A calls `kill` on another process B. [*6 marks*]

(d) There are two common situations where design decisions have to be made in the implementation of signals. For each of the following, say what you think the issues would be:

i. A process receives a signal while it is blocking inside a system call. [*3 marks*]

ii. A process receives a signal while it is executing a signal handler. [*3 marks*]

3. (a) Describe the implementation of the traditional Unix filesystem. [*8 marks*]

   (b) Some modern Unix filesystems support file-system internal compression. That is, a file can be flagged so that within the filesystem, the file's data is compressed, although reading and writing to the file appears as normal to users of the filesystem. What issues would you expect to have to be addressed in adding compression to a traditional Unix filesystem? [*5 marks*]

   (c) The filesystems discussed in the course are all implemented within the kernel. However, it can be argued that filesystems should be implemented in userspace, not in the kernel.

      i. How could you implement a filesystem in userspace? What would be running the code? [*4 marks*]

      ii. Give one argument in favour of userspace filesystems, and one argument against. [*4 marks*]

   (d) Modern Linux distributions ship with a facility that allows even unprivileged users to implement a filesystem laid over the 'real' filesystem – that is, the user's filesystem is 'mounted' at some existing real directory, and references to paths underneath that are dealt with by the user code. What security and integrity restrictions would you expect this facility to enforce? [*4 marks*]