UNIVERSITY OF EDINBURGH

FACULTY OF SCIENCE AND ENGINEERING

M.Sc./DIPLOMA IN ARTIFICIAL INTELLIGENCE
M.Sc./DIPLOMA IN COGNITIVE SCIENCE
M.Sc./DIPLOMA IN COMPUTER SCIENCE
M.Sc./DIPLOMA IN INFORMATICS

OPERATING SYSTEMS : SOLUTION NOTES

Friday 0 2002

??:?? to ??:??

Examiners:
N. Chater (External)
N. Jennings (External)
E. Hancock (External)
K. Hawick (External)
R.N. Ibbett (Convener)
J. Oberlander (Co-Convener)

INSTRUCTIONS TO CANDIDATES

Answer any TWO questions.

All questions carry equal weight.

1. (a) i. Mutual exclusion is ensured. By setting c1=0, process 1 forces process 2 to wait before entering its critical region.

ii. It is too drastic. If both processes set their c variables to 0 before executing the while statement, they will both wait indefinitely thereafter.

(b) Peterson's solution:

```
shared variable c1 =1, c2 = 1, turn = 1;

process 1:                         process 2:
  while (TRUE) {                     while (TRUE) {
    c1 = 0;                            c2 = 0;
    turn = 2;                          turn = 1;
    while (c2==0 && turn==2){}         while (c1==0 && turn==1){}
    CRITICAL SECTION 1                 CRITICAL SECTION 2
    c1 = 1;                            c2 = 1;
  }                                  }
```

Mutual exclusion is still assured. The while clause tests if the other process is either in or wanting to go into its critical section by checking the other's c variable. A loop is now not possible since the turn variable cannot be set to both 1 and 2 simultaneously and it is not changed whilst both processes are executing their while clauses.

A formal proof technique involves creating an invariant logical expression for each process which expresses the state of the variables when it is executing the while-clause line and when it is not. This can be used to demonstrate that each process cannot falsify the other's invariant and to demonstrate mutual exclusivity by simplification of the logical expression expressing the condition of non-exclusion to the value false. The guarantee of eventual forward progress can also be proven in the same way.

(c) Examples :

```
test-and-set(lock) { /* IBM, M68xxx,... */
  temp=lock; lock=1; return temp;
}

while (TRUE) {
  while (test-and-set(lock)==1) { }
  CRITICAL SECTION
  lock = 0;
}


swap(a, b) { /* Sparc, ... */
```

```
temp=a; a=b; b=temp;
}

while (TRUE) {
  key = 1;
  while (key==1) swap(lock, key);
  CRITICAL SECTION
  lock = 0;
}
```

Also Increment-and-test/Test-and-decrement (ICL), Atomic Load/Store (Sparc), etc.

Advantages:
Applicable to any number of processes on single or multi-processor;
Simple and easy to verify;
Can support multiple critical sections, each with own control variable.

Disadvantages:
Busy-waiting is used;
Deadlock possible where a higher priority process gets the CPU whilst in its busy-wait loop and another process is in its critical section but not getting any CPU.
Starvation possible : when a process leaves a critical section and more than one process is waiting, selection of the next process might be arbitrary and hence a process could be starved indefinitely.

2. (a) i. Process : program in execution in its own virtual machine etc.
   ii. Thread : sub-unit of a process with its own thread of execution, stack etc.
   iii. Interrupt : interruption in the execution flow of a process caused by some external asynchronous event such as I/O.
   iv. Exception : interruption in the execution flow of a process caused by some event synchronous with the execution such as divide by zero, memory access violation etc.
   v. Signal : change in execution flow to some specific handler in a process to deal with a message or status information sent by the operating system, for instance in response to a memory access violation.

(b) To monitor page usage using 'mprotect', the initial part of the process incorporating the application must set up a signal handler function which will be called when a memory access violation exception occurs (it happens to be SIGSEGV). 'mprotect' should be called to set the access permissions to each virtual page used by the application to 'none'. When the application starts to run it will cause an exception for each page it uses. The job of the signal handler is to record or print out the address of the page which caused the violation and perhaps its access type, read, write or execute, and then to call 'mprotect' to allow that page to be accessed in future without causing any more exceptions. The handler then exits and execution returns to the application where the access will be re-attempted, and this time succeed.

(c) To monitor page usage in every successive time-slice, the same signal handler can be used but some means of resetting the page access permissions back to 'none' at the start of every time-slice, must be incorporated. This could be done by creating a separate thread which calls the 'sleep' system call to wake it up again at the start of the next time-slice. (In practise, the actual Linux 'sleep' system call works only in units of a second which is too long a time-slice to be very useful; an alternative can, however, be fudged up using a 'poll' system call to a non-existent device with a time-out which is allowed to be set in msecs.) Upon reawakening, it will call 'mprotect' to remove all the access permissions for the application' pages, thereby causing a whole new set of access violation exceptions which can be monitored as before. It might also be wise to set this thread's priority high so that it always runs immediately its sleep finishes.

3. (a) Segmentation allows :
   i. segments for overlays to deal with lack of physical memory
   ii. relocation of code and data addresses can be made segment local
   iii. protection of access e.g. read, write, execute segments
   iv. efficiency of memory usage : segments can be moved around memory to minimise external fragmentation losses
   v. modularity : a segment can correspond to a module of program
   vi. sharability : modules in segments can be shared between processes

   (b) Pentium segmentation : implicit selection of segment registers depending on type of access used in the instruction; code, data, stack, extra stack segments etc. Each segment register indexes to one of two descriptor tables, global and local, which contain the segment base addresses, lengths and protection status of the segment. The address generated by the instruction is used as an offset within the selected segment.

   The Pentium segmentation system in its protected mode provides a comprehensive set of facilities. Segments are large and are mostly selected by context i.e. the program segment and the stack segment. Address overflow from one segment to another is precluded. Separate process and system spaces are convenient. Several levels of privilege are available. Multitasking can be dealt with using its Task State segment.

   (c) The two lowest bits of the CS register form the 'Current Privilege Level' (CPL). This is normally the same as the privilege level of the current code segment. CPL changes when control passes to a code segment with a different privilege level.

   The Descriptor Privilege Level (DPL) is taken from the entry of a segment in the segment table. Instructions may only cause a segment register to be loaded (to allow access to that segment) if DPL is greater than or equal to CPL i.e. the lower the CPL value, the greater the privilege.

   There is also a 'Requestor Privilege Level' (RPL) which can represent the level of privilege of the procedure which created the segment selector. When RPL is greater than or equal to CPL, this value overrides the value of CPL for accesses using that selector. This is used to guard against a less privileged program requesting a more privileged program to access protected data.

   (d) A segment table entry now points to a Page Table whose entries in turn point to Page Frames (of fixed size) in memory. The virtual address is further partitioned into a segment no., a page-within-segment number, and a within-page displacement. The page-within-segment no. is used to index into the appropriate page table. Each page table entry has a 'Presence' bit which indicates whether or not that page of VM is in physical memory. It also has 'Accessed' and 'Dirty' bits.

The benefits of paging include the ability to implement a 'Demand Paging' strategy, removal of external fragmentation, ability to share pages between processes etc.