2004 Operating Systems Foundations SOLN Paper 11 Q6 JMB

2(i) Runtime can multiplex one OS-A process between user threads
Each user thread must have its own stack to preserve its state and a control block with stack pointer and instruction pointer (resume address).
Synchronisation between user threads (as defined in whatever prog. lang) must be supported. Basically WAIT and SIGNAL, causing threads potentially to be blocked & unblocked.
Students have seen a semaphore manager.
(8) A scheduler must decide which thread to run when the current thread blocks - perhaps done through semaphore management.
Note that a single thread of control executes the runtime and all user threads.

(ii) • Can't respond to external events (I/O - Interrupt) by switching threads
(3) • Can't exploit a multiprocessor.
• A blocking system call by any thread blocks the whole process.

(iii) Concurrency control can be simple - a thread only gives up
(1) control on explicit WAIT (or EXIT).

b(i) Differences in OS-B's runtime from that above.
• runtime must create a kernel thread for each user thread (by means of a system call which returns a thread ID) and must kill any thread that terminates (again by syscall)
• if a user thread's call of WAIT to the runtime causes that thread to block, the WAIT routine must make a system call to block the corresponding kernel thread
(6) • .. ditto .. SIGNAL must call UNBLOCK.
• no scheduling/policy is needed since OS-B schedules kernel threads.

(ii) • depends whether OS-B does pre-emptive scheduling
(2) • can exploit a multiprocessor
• threads can block independently of each other

(iii) A user thread can be preempted at any point of its
(1) execution - fully general concurrency control is needed. Clock
eg need a semaphore to protect every shared data structure.
(But app. programmer may do this anyway for OS-A).

(20)

Question is on Part 3 of the course.