**Instructions**  There are 4 questions. *Complete only 3 of them.* You have 90 minutes. The exam is open papers and notes. State your assumptions and show your work. Be concise.
1 Priority inversion

Lampson and Redell suggested three solutions to priority inversion:

1. Do nothing

2. Raise priority of thread in monitor to the priority of the highest waiting thread

3. Raise priority of thread in monitor to maximum system priority

   a) Compare the effectiveness of these solutions (e.g., how closely system behavior tracks priority) and the efficiency of these solutions (e.g., how much overhead they entail). Be specific – the statement “approach A has lower overhead than approach B” will get no credit unless you explain why this statement is true – what specific actions does B have to take that A doesn’t?
b) Lampson and Redell argue that simple approaches are appropriate for a Mesa-like environment where each machine is used by a single user and all programs are written in Mesa by world-class programmers. State arguments for and against using option 1 and for and against using option 3 in a Mesa-like environment.
2 User-level control of thread scheduling

Both Exokernel and Scheduler Activations use visible revocation to allow a user-level scheduler to control which threads run when there are fewer processors than threads. But, the abstraction is slightly different.

In Exokernel, on a timer interrupt “a register is saved in the ‘interrupt save area’ [of the process], the exception program counter is loaded, and [Exokernel] jumps to a user-specified interrupt handling code with interrupts re-enabled. The application’s handlers are responsible for general-purpose context switching.” Instead, in Scheduler Activations, the kernel suspends two threads \( t_1 \) and \( t_2 \) and calls `has been preempted(stateT1, stateT2)` in the process with \( t_1 \) and \( t_2 \)’s processor state as arguments.

(a) In both systems, describe in detail what happens when two user-level threads \( U_1 \) and \( U_2 \) are running on two processors \( P_1 \) and \( P_2 \) and a timer interrupt on \( P_1 \) occurs which causes the kernel to de-schedule a thread on one of the processors to allow a different process to run one of its threads \( V_1 \). Describe what happens beginning with the interrupt and ending with the state where \( V_1 \) and \( U_2 \) are running on \( P_1 \) and \( P_2 \).
(b) How many times do processors switch from user-mode to kernel-mode in each arrangement?

(c) How many times are the processor register set copied between processor and memory in each arrangement?

(d) How much data is copied from kernel memory to user memory in each arrangement?
3 Network performance

Below is the graph for the following benchmark of Active Message performance on the Intel Paragon:

for (BurstSize = 1; BurstSize <= 128; BurstSize = BurstSize * 2) {
    Start Timer
    repeat BurstSize times
        Send small message (and receive any queued replies)
        Poll for replies for Delay microseconds
    Stop Timer
    Handle remaining replies

(a) Use the data in this graph to estimate machine communication performance – $L$ round-trip short-message latency, $o_s$ send overhead, $o_r$ receive overhead, and $g$ bottleneck short-message end-to-end throughput.

(b) Suppose that at one point in a program one node needs to send a short message to 3 other nodes on the machine and that each node that receives this message replies with a short message to the originating node. Estimate the minimum time from when the node begins sending its messages until it receives its last reply. Show your work (please first write your formula in terms of $o_s$, etc. before plugging in numbers).

(c) Suppose that at one point in a program one node needs to send a short message to 1023 other nodes on the machine and that each node that receives this message replies with a short message to the originating node. Estimate the minimum time from when the node begins sending its messages until it receives its last reply. Show your work.
4 Memory management

Suppose we re-implement Multix shared segments/dynamic linking but use a 64-bit virtual address space and ensure that if a shared segment Y maps to virtual address X in process A, it can only map to the same virtual address X in process B.

1. Describe how to maintain such an invariant when dynamically linking ("making known") a shared segment into a process. (Be sure your solution handles swapping where segment Y is moved from one physical address Y1 to another physical address Y2 while being accessible at virtual address X the whole time.)

2. Using this invariant, describe how to simplify dynamic linking compared to Multix. Explain how the shared code would (a) read the shared variable at offset x in the shared segment X, (b) call procedure at offset y in the shared segment Y, and (c) read the per-process variable at offset z in private segment Z (assume that all references to per-process private variables must be passed to shared code as parameters.)