Operating Systems **CS355** Spring 2007

Lab 6: Adding functionality to HERACLOS

In this lab, we will upgrade the low-level hardware and write new operating system components for the HERA processor. The files will be available in `Shared_Project_6`.

For this lab, I expect the hardware to be much simpler than the software, and we will undertake this project if I can get the hardware modifications ready in time. You will then choose one software project; at the end of exam period, we will meet for each student to present their project. The projects have some degree of flexibility, and could be suitable for one person or a group of two.

HARDWARE. (I'll do these, unless someone else wants to do them too.)

- Modify the HERA processor so that it can transfer data to and from the shadow PC.
- Add a clock line that interrupts HERA every so often (say, every 1000 instructions).
- Add a "process number" register and simple memory controller so that each of 14 threads can use addresses 0000...0fff to address a distinct set of physical addresses that map to subsets of physical memory 1000...efff. Physical addresses 0000...0fff are reserved for the O.S., and physical addresses f000...ffff are shared among all threads. When in "kernel mode", all memory operations involve physical, not logical, addresses. Instruction addresses are treated as physical addresses (to ROM) in both kernel and user modes. This will require some modifications to the location of the buffer for user I/O, which we should undertake as a group.

SOFTWARE. (You should choose **ONE** of the following and discuss it with me before beginning)

- Create a process table, **thread switch** interrupt routine, and thread start and end operations. On interrupt, you will need to store the current state information in the process table, select a new process to run, load the process number into the memory controller, and resume that process. You do not have to change process states (except between `RUNNING` and `READY`), but your table should be flexible enough to allow other states (see "semaphores" below) not run any process in any other state (so that someone could implement, say, semaphores).
- Create a simple **semaphore** data structure in the kernel and write semaphore acquire and release system calls. These will need to change process states in the process table, so you will need to coordinate with who ever is doing that. Modify the input and output routines so that processes use a semaphore to ensure exclusive access to the terminal, and ensure that a process waiting for terminal input is never selected by the scheduler.
- Create system calls for **shared memory**. You will need to allocate and free regions of memory, and possibly also gain access to a segment created by another thread. Try to minimize the amount of work done in kernel mode while still ensuring that the timing of the thread switch interrupt can't cause problems.
- Write a multi-threaded HERA program that uses the features described above to do something that shows them off.
- **As an alterative**, you may ignore the HERA system altogether, and read about the implementation of one of the topics from the last weeks of class (process scheduling, virtual memory, file systems and disk access) or a topic we did not cover in detail (e.g., special concerns of operating systems running on multiprocessors). You should **write a description** contrasting the implementations on at least two real systems, and explaining what each does beyond the extremely simplified system described above, and explain these things to the class.