The Basic Idea: The existing Minotaur system programs (`sysprogs` in `stdfile`) should run just fine on the baseline Minotaur and as well on your a2 version of Minotaur. For a2, we are **not** interested in determining if the new foreground/background feedback queue has better performance than simple round-robin in Minotaur. You will have to write one or more new system programs that use the `YIELD` system call (no parameters), to test that feature; see the description of the Generic System Call facility in Section 11 of the document *Minotaur External Documentation*.

Some Mechanism Details: The foreground time slice, `foregTS`, is equal to the current `timeSlice`. The background time slice, `backgTS`, is equal to `2 * foregTS`. Condition A is defined by: A process with an average CPU time used per dispatching assignment that is less than the system-wide average, where the system-wide average is taken over all currently active, ready processes. The periodic interval for initiating a scan of the background queue is `timeSlice / 2` .

Some Rationale: New processes deserve to have their first CPU burst at higher priority. Processes that have `YIELD`ed a processor are good citizens and should be rewarded with a higher priority on their next burst. Processes that satisfy Condition A generally have computed flat out, with little delay for events or I/O, and should be favoured with a higher priority. Beyond that, do not attempt to ascribe any penetrating insight to the problem statement; its details basically came from here and there.

Some Issues: Concurrency is paramount. How well you handle it will largely determine how well you do on the assignment. Testing is important. Pending specific details from the TA, you should design and document a testing strategy that helps to verify the correctness of your implementation.

Details of Details: Check the newsgroup for details of coding requirements, testing requirements, handin requirements, and requirement requirements (some details stated here may change during the course of the assignment).

CSC 468F/2204F

OPERATING SYSTEMS

FALL 1997
UNIVERSITY OF TORONTO

Assignment 2 (a2)

due: electronically by 12 noon, Friday, November 7
hard-copy by the start of the tutorial that day, 1:10 p.m.

Purpose: To awaken the beast within you.

General Comments: Keep in touch with your team members to ensure that they are still alive (`ping` them if you have not talked recently); the drop date happens while this assignment is "current" and that has been known to cause people to do strange things. Submit your hard-copy handin inside some secure outer enclosure, clearly labelled with team member names and student numbers and with your (single) tutorial room. Ensure that the equality sheet has been signed and included.

Specific Comments: The tentative office hours for this assignment are from 2 to 3 p.m. on Friday, October 31 (scary!) and from 5 to 6 p.m. on Monday, November 3 (rooms TBA). Silent running begins at 12 noon on Wednesday, November 5. Note that the midterm occurs while this assignment is "current" and takes up a tutorial slot in week 7; ensure that you allocate enough time to finish this assignment.

Become familiar with `Kernel/Process`, `Hardware/CPU`, and their associates.

Overview: In Section 4.5, Process Scheduling, of the document *The Minotaur Operating System*, it is stated: "The process scheduler simply implements a round-robin policy, with a single global FIFO queue of ready processes." Your task is to "simply" design and implement a two-level (foreground and background) feedback queue for processor scheduling in Minotaur. Processes in the foreground queue have higher priority than those in the background queue; within a queue level, processes are served round-robin. The following types of processes are naturally found on the foreground queue:

- kernel processes (see `ProcStruct.Priority` in `Kernel/Process`)
- user processes newly initiated (see `NewProcess` in `Kernel/Process.b`)
- user processes that have just `YIELD`ed the processor (see below)

Kernel processes are inserted at the head of the foreground queue; user processes are inserted at the tail. The background queue contains ready user processes not found on the foreground queue; insertion here occurs at the tail. Periodically, the background queue is scanned and each process that satisfies Condition A is moved to the foreground queue for its next burst.