# Programming project 1 — warm-up exercise, handed out 12 September

There is no due-date; you are not required to do this exercise. However, I do recommend working on it, as you will be able to reuse most of the code you write here for the actual programming project.

You are to simulate the workings of an operating system with a round-robin process scheduler.

**Processes.** The simulation should contain some number N (four or five may be reasonable) of processes. Each process has a certain execution "profile", i.e., a pattern of computation work with system calls interspersed. A profile is given as a sequence of the following actions:

- work (Y) — do some CPU work for Y units of time
- system-call-open-file (F) — system call to open file F
- system-call-close-file (F) — system call to close file F
- system-call-quick — an unspecified quick system call

You can play with processes that are finite sequences, or processes that repeat the same sequence forever in a loop, or with randomly-generated sequences.

**System calls.** For each system call you should simulate the mechanism of "trapping" to the operating system, such as saving the PC and the PSW. System-call-quick takes one unit of time and immediately returns to the user process that invoked it. On the other hand, system-call-open-file and system-call-close-file take one unit of time but then incur a delay of Z units of time for disk access. Therefore, the operating system gives the CPU to another process.

**Scheduler.** A simple round-robin scheduler, with quantum size X units of time.

**What should the simulator report?** Each system call in the simulator should print a line, such as "File F opened".

**Possible experiments.** Construct some processes, set X=10 and Z=100, and measure CPU utilization. What fraction of CPU time is spent in actual work, what fraction in system calls, and what fraction is idle? Try to construct different process mixes that will result in different CPU utilization, e.g., 10%, 50%, 90%.

**Simulator structure.** An additional goal of this assignment is to gain experience in writing code using concurrent cooperating processes under Unix. Therefore, the simulator itself is to be written as several processes. The processes will communicate either using shared memory or using sockets.

You are free to structure the processes as you wish, but here are two suggestions. You may map each of the simulated processes onto a Unix process, and have the operating system functions in an extra Unix process. Alternatively, you may work on a fine-grain simulation, in which system components that in reality are asynchronous are represented by separate Unix processes (CPU, timer, memory, disk, etc.); you will actually represent processes as code in a simulated memory, and will simulate the execution of instructions in the CPU.

Do not consider this sheet as a full specification of the project—it's deliberately leaving out details that you should fill in, and to a certain extent you can choose how much detail to provide in the simulation.