# CS377: Operating Systems

# Lab 1: Simulator

**Assignment**

For this and subsequent labs, you will be developing a ***trace-driven simulator*** for an operating system. Rather then implement a real operating system, which can be both time-consuming and error-prone, most researcher use *simulators* to see if their ideas are worth implementing. A *trace* is a representation of program activity, often gathered automatically from real programs. We will be using traces to "drive" our simulator, that is, as input to the simulator.

For this assignment you will implement the core of the simulator – the "main loop" that is going to interpret the traces and make appropriate calls to other parts of the system – the CPU scheduler, the virtual memory manager, and the disk scheduler. We're giving you APIs and implementations for all three of those, as well as some axillary classes to read and parse trace files. Your program would have to maintain a notion of time, and output total time elapsed when it finishes interpreting all the traces, along with statistics from other modules (by calling their **report**() method).

**Trace format**

Our traces will be text files. Each trace file represents one process, and each line is one trace record, containing the following information:

<repetitions> <operation-type> <long integer>

*repetitions* = repetitions of this operation (= one fewer than # of times it's executed)

*operation-type* = string indicating what this trace record corresponds to:

| | |
|---|---|
| R, r | Read from memory |
| W, w | Write to memory |
| I, i | Execute instruction |
| < | File read |
| > | File write |
| S | Sleep |
| ( | Open file |
| ) | Close file |

For read, write, and instruction execution, the rest of the line is a decimal number corresponding to the page number of the virtual address that is read/written/executed. For sleep, the rest of the line is an integer corresponding to the sleep time; for file I/O, it's the disk track number on which the operation is being performed.

**Implementation**

You are given the following classes:

- **AbstractCPUScheduler** – defines API for CPU schedulers
- **AbstractVMManager** – defines API for memory managers
- **AbstractDiskScheduler** – defines API for disk schedulers
- **SimpleVMManager** – implementation of memory manager

- **FIFOCPUScheduler** – implementation of CPU scheduler
- **FIFODiskScheduler** – implementation of disk scheduler
- **ProcessTrace** – represents a trace
- **InstructionBlock** – represents 1 line of a trace
- **DiskRequest** – defines format of requests to disk
- **Wakeable** (interface) – interface for objects that can be awakened from sleeping on an event.

**ASSIGNMENT: (a)** You are to write a main class called Simulator, and whatever utility classes you may need to implement it. Your program should take the following command line arguments: the name of the file which lists the CPU scheduler class name, memory manager class name and disk scheduler class name (in that order), and the names of trace files to be run. Your program should output statistics collected by different modules (by means of a **report**() method) and total time elapsed (number of instruction executed) since the beginning of the simulations. Since your program will be graded automatically, you should make sure your program DOES NOT output anything else.

We're also giving you *part* of the Simulator class. LoadDynamicClasses is an example function that loads classes listed in the file, name of which is passed as a parameter. Modify it if you need to and use it to load your instances of CPU scheduler, memory manager and disk scheduler (FIFOCPUScheduler, SimpleVMManager and FIFODiskScheduler for this assignment). Also make use of the QUANTUM and DISK_ACCESS_TIME constants in your main loop to determine when to switch contexts and to determine when disk requests complete (you may assume that 1 disk request is completed each DISK_ACCESS_TIME units of time).

**(b)** You also have to write and submit artificial test cases to demonstrate that your program performs correctly.

**Logistics**

You will be working in randomly-assigned groups of two. You have to maintain all your code in Subversion repository in the Edlab – WE WILL BE CHECKING YOUR PROJECTS OUT OF THAT REPOSITORY TO GRADE THEM! We highly suggest that you use Java 1.5 and Eclipse 3.1 for developing your simulators. Eclipse also has nice plug-ins to work with Subversion repositories.

```
public class Simulator {

	private static final int QUANTUM = 100;

	private static final int DISK_ACCESS_TIME = 1000;
	/**
	 * Loads CPU scheduler, memory manager and disk scheduler classes
	 * whoes names are listed in a file
	 * @param filename Files where class names are listed
	 */
	private void loadDynamicClasses(String filename) {
		//check file existence.
		//Can't run without this file, so halt if the file not there
		File f = new File(filename);
		if (!f.exists()) {
			System.out.println("File " + filename + "does not exist!");
			System.exit(1);
		}

		String CPUSchedulerName = null;
		String VMManagerName = null;
		String DiskSchedulerName = null;

		//try reading class names from the file
		//can't work without class names, so halt on failure
		try {
			BufferedReader reader = new BufferedReader(new FileReader(f));
		    CPUSchedulerName = reader.readLine();
			VMManagerName = reader.readLine();
			DiskSchedulerName = reader.readLine();
			reader.close();
		}
		catch (IOException e) {
			System.out.println("Cannot read file " + filename + ".");
			System.exit(1);
		}

		//try loading classes
		//can't proceed without them, so halt on failure
		ClassLoader cl = ClassLoader.getSystemClassLoader();
		try {
			Class c = cl.loadClass(CPUSchedulerName);
			cpuScheduler = (AbstractCPUScheduler)c.newInstance();
			c = cl.loadClass(VMManagerName);
			memoryManager = (AbstractVMManager)c.newInstance();
			c = cl.loadClass(DiskSchedulerName);
			diskScheduler = (AbstractDiskScheduler)c.newInstance();
		}
		catch (Exception e) {
			System.out.println(e.getMessage());
			System.exit(1);
		}
	}

	public static void main(String[] args) {
	...
	}
}
```