# ICSI412 Operating Systems – Project 1 – Scheduler Beginnings

## Overview

We are going to build an operating system simulator throughout this course. We will get started by working on the scheduler. The scheduler is the program that keeps track of the user processes and schedules them to run.

## How do we simulate an operating system in Java?

In Java, once we call a function, it runs until completion. We don’t really want that, though, when we simulate an operating system. We will have to do a little extra work in our userland programs to make our simulation work. Specifically, we must break our userland program into pieces. Each piece is the work done between times when we talk to the operating system. For example, in C, we might do this:

int fd = open(“myfile”,O\_CREAT);

for (int i=0;i<100;i++)

someArray[i] = i;

write(fd,someArray,100);

close(fd);

We will have to break that into three different pieces. One for the open, one for the loop and the write and one for the close. You will see this in upcoming assignments.

## Supplied Classes

**OSInterface** – this is the interface that we will work with throughout the semester. This defines the interface between the user processes (userland) and the operating system. For now, it will contain only two methods:

int CreateProcess(UserlandProcess myNewProcess);

This is called by user programs to create a new process. You may remember fork() from a previous class – that is similar but it “clones” the current process. We are following a different methodology. CreateProcess returns a PID (process id) – a number that increments once per process. The first process will have PID 1, the second process PID 2, etc. The numbers are not reclaimed; there will never be another PID 1.

boolean DeleteProcess(int processId);

This is called by user programs to delete a process based on the PID. It returns true if it found and deleted the process and false otherwise.

**UserlandProcess** – this class represents work that we want to do from userland. You will create subclasses that implement the single abstract method.

public abstract RunResult run(void);

This method will be called by your operating system (acting as the kernel) to do user work.

**RunResult** – this class represents the result of running the userland code – how long did it take? Did it run until the system timer stopped it? Or is it giving back time voluntarily.

public boolean ranToTimeout;  
 public int millisecondsUsed;  
For this assignment, these values are not used, but you should fill them in for the next assignment. Values for milliseconds should be less than 100.

**Task 1** – Create HelloWorldProcess and GoodbyeWorldProcess

Create two new classes which extend UserlandProcess. Their run() method should print “Hello World”/”Goodbye World” and return a RunResult that you made.

**Task 2** - Create KernelandProcess

From userland, we call “CreateProcess”. What should the kernel do? It needs to keep track of the list of processes. You might be inclined to make an ArrayList in the kernel of UserlandProcess objects. The problem with that is safety and security. We are going to (in future assignments) store things like permissions and security information. We can’t do that in an object that userland has access to.

We will make a new object, KernelandProcess which holds a reference to the UserlandProcess. It will hold another piece of information – the PID described in CreateProcess.

**Task 3** – Create the Scheduler

For this project, we will be making a very basic scheduler – a round robin. You should implement CreateProcess and DeleteProcess in the scheduler as well as a void run() function that actually runs the tasks. CreateProcess and DeleteProcess should manage a list of KernelandProcesses.

Run() should loop over the process list and call run() on each. And it should do that in an infinite loop.

**Task 4** – Create Startup

We have created a lot of functionality, but no way to start our simulator.

Create a class called startup. Give it a standard Java main:

public static void main (string[] args)

In that main, create a scheduler. Use it to create processes for HelloWorld and GoodbyeWorld. Then call Run() on the scheduler to get it started.

***You must submit buildable .java files for credit.***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rubric | Poor | OK | Good | Great |
| HelloWorld | None (0) | Prints the message (5) |  | Prints the message and returns a reasonable RunResult (15) |
| GoodbyeWorld | None (0) | Prints the message (5) |  | Prints the message and returns a reasonable RunResult (15) |
| KernelandProcess | None (0) |  |  | Holds UserlandProcess and PID (5) |
| BasicScheduler CreateProcess | None (0) | Generates a PID (5) | Generates a PID and a KernelandProcess (10) | Generates a PID, KernelandProcess and manages the list (15) |
| BasicScheduler DeleteProcess | None (0) |  | Finds the process by PID (10) | Finds the process by PID and removes it (15) |
| BasicScheduler Run loop | None (0) |  | Runs all processes (10) | Runs all processes in an infinite loop (15) |
| Startup Process | None (0) |  |  | Creates a scheduler, the two userland processes and calls the run loop (10) |
| Free Points |  |  |  | 10 |