

Marking Requirements

To facilitate the marking of your simulator, we ask you to do the following.

- (i) Provide a 'readme' that explains the features of your design and implementation, and how to run it (so that the marker can find their way around).
- (ii) Ensure that your program simulates time in a specific manner.
- (iii) Ensure that your program can handle config and psuedo-program files in subdirectories.
- (iv) Instrument your code so that a trace of execution can be made.

(ii) Time costs

- When the kernel handles a system call or an interrupt, it costs **SC** units of time.

At the end of a system call, your simulator should advance kernel time (and thus system time) by **SC** units. (The actual value for **SC** is hard coded as 2.)

- If handling a system call or interrupts involves a context switch (i.e. a process is put onto or removed from the CPU), then this costs a further **CS** units of time. (The actual value is provided as a command line argument.)

After the context switch your simulator should advance kernel time by **CS** units.

- When a new process is put on the CPU, a time slice timeout must be scheduled. Assuming **TS** units for the time slice, The timeout should be set to $(System\ Time + SC + CS + TS)$ i.e. the cost of call and context switch does not eat into the time slice.

(iii) Test data in sub directories

Ensure that your program can deal with config files in sub directories e.g.

```
java Simulator TestDirOne/config.txt 100 2
```

Also, ensure that your program can deal with config files that use relative pathnames for programs e.g.

```
# Config for test one.
I/O 1 DISK
PROGRAM 0 TestDirOne/progOne.dat
PROGRAM 5 TestDirOne/progTwo.dat
```


(iv) Code instrumentation

For marking purposes we need to see what your simulator is doing when it runs. We want output such as the following:

```
Time: 0000000000 SysCall(EXECVE, "TestOne/program1.prg")
Time: 0000000000 Context Switch({Idle},{1, TestOne/program1.prg}).
Time: 0000000034 SysCall(TERMINATE_PROCESS)
Time: 0000000034 Context Switch({1, TestOne/program1.prg},{Idle}).
```

```
System time: 38
Kernel time: 8
User time: 30
```

Idle time: 0
Context switches: 2
CPU utilization: 78.95

The output consists of a trace followed by the simulation results. The trace should detail all calls to the kernel syscall method and interrupt method, and to the CPU context switch method. Output from the context switch method should detail the system time at entry, the outgoing process, and the [INCOMING](#)  process.

We have constructed methods that you can add to your kernel to trace syscalls and interrupts:

```
private void sysCallTrace(int number, Object... varargs) {
    String details=null;
    switch (number) {
        case MAKE_DEVICE:
            details=String.format("MAKE_DEVICE, %s,\"%s\"", varargs[0], varargs[1]);
            break;
        case EXECVE:
            details=String.format("EXECVE, \"%s\"", varargs[0]);
            break;
        case IO_REQUEST:
            details=String.format("IO_REQUEST, %s, %s", varargs[0], varargs[1]);
            break;
        case TERMINATE_PROCESS:
            details="TERMINATE_PROCESS";
            break;
        default:
            details="ERROR_UNKNOWN_NUMBER";
    }
    System.out.printf("Time: %010d SysCall(%s)\n", sysTimer.getSystemTime(), details);
}
```

```
private void interruptTrace(int interruptType, Object... varargs) {
    String details = null;
    switch (interruptType) {
        case TIME_OUT:
            details=String.format("TIME_OUT, %s", varargs[0]);
            break;
        case WAKE_UP:
            details=String.format("WAKE_UP, %s, %s", varargs[0], varargs[1]);
            break;
        default:
            details="ERROR_UNKNOWN_NUMBER";
    }
    System.out.printf("Time: %010d Interrupt(%s)\n", sysTimer.getSystemTime(), details);
}
```

You will need to add a toString method to your implementation of ProcessControlBlock:

```
public String toString() {
    return String.format("{%d, %s}", this.getPID(), this.getProgramName());
}
```