Name: _____

IB Computer Science 2
Signals worksheet

Download `Timer.zip` to the folder where you usually write activity code, and extract it. To make macOS trust the library, run: `xattr -d com.apple.quarantine libTimer.dylib`

External control flow

Most programs execute from top to bottom, potentially skipping the body of a conditional, repeating the body of a loop, or jumping in response to a method call. All such internal control flow structures determine the next instruction based on the one that is currently executing.

The one exception we have seen is the event-driven programming paradigm that underlies graphical user interfaces. The user can click or touch elements at any time, causing methods known as event handlers to run and handle the asynchronous input. Processors support a similar mechanism known as **interrupts**, where hardware devices place voltage on a special signal wire that causes the processor to immediately jump to a special method called an *interrupt service routine*. After handling the asynchronous hardware event, this code instructs the processor to jump back to the location of the instruction that was executing before.

The interrupt service routines are part of the operating system. It does not let processes install their own, or they could take over a CPU core and prevent other processes from running on it. However, the OS provides analogous externally-imposed control flow called **signals**, which allow another process or the OS to make a process jump to a method called a *signal handler*.

Examining how software signals affect processes will provide a close approximation of how hardware interrupts affect the operating system.

1. Examine the `SpecialKeys.java` program. Run it a few times, trying the suggested key combinations. Which of the keys changes the program's control flow? Circle them.

   Ctrl+D Ctrl+C Ctrl+Z

2. The two keys you circled are the only ones that generated signals. Each signal has an associated default action. We have been using one of these key combinations since early on; which one, and what is its signal's default action?

3. The other key's signal stops the process, pausing its execution. Try running the `fg` command, then typing some text on the next line and hitting enter. The `fg` command sent yet another signal to the process; what effect does this signal have?

The `man signal` command opens a manual page that lists the available signals and their default actions. Here is an excerpt showing some relevant signals, including SIGINT (generated by Ctrl+C), SIGTSTP (generated by Ctrl+Z), and SIGCONT (sent by the `fg` command):

```
      Name         Default Action         Description
…
2     SIGINT       terminate process      interrupt program
…
14    SIGALRM      terminate process      real-time timer expired
…
18    SIGTSTP      stop process           stop signal from keyboard
19    SIGCONT      discard signal         continue after stop
…
24    SIGXCPU      terminate process      cpu time limit exceeded
```

(If you opened the manual page and are struggling to close it, use the Q key.)

4. Use the above table to check your answers to the preceding questions.

Polling

5. Examine `BusyLoop.java`, which loops forever. Run the command `ulimit -St 5` to set a CPU time limit. Then run `java BusyLoop.java`. What happens and why?

6. Look back at `SpecialKeys.java`. Predict how the CPU time limit will affect it.

7. Run `java SpecialKeys.java` in the same terminal so it is subject to the time limit. Hold down a key but do not press enter. What happens and why?

8. Split the terminal and run `top` in the new one. Then rerun `BusyLoop`. What percentage of the time does it spend occupying its CPU core when it is running?

9. Hit the Q key to exit. Identify at least two disadvantages of a program like `BusyLoop`.

It would be very inefficient if collecting input required **polling**, or constantly checking, whether the user had entered another character. Instead, whenever a program requests a line of user input, the operating system deschedules it, freeing its CPU core for use by other processes. The OS resumes the paused process only once a full line of input is available. This is why the `SpecialKeys` program barely consumed any CPU time.

Active notification

Both interrupts and signals are active notification mechanisms that inform code as soon as an asynchronous event occurs. Hence, they can be used to rewrite code to eliminate polling.

10. Read `PollingClock.java` and predict how the time limit will affect it. Run it to check.

11. Examine `SignalClock.java`'s `main()` method. The only difference from `PollingClock`'s is the new `subscribeToSignal()` call. Based on the table and your findings on the previous page, predict what the program will do.

12. Without modifying the code, run `java SignalClock.java`. Was your hypothesis correct? If not, why does the program behave this way?

13. Uncomment the commented-out lines and rerun it using `java --add-exports java.base/jdk.internal.misc=ALL-UNNAMED SignalClock.java`. What bug does the program now exhibit and why?

14. Remove the loop in `main()` and change the `System.out.print()` to a `System.console().readLine()`, so that the program does not exit immediately. How does the CPU time limit affect the program, and what does this tell you?

15. How do you think the operating system listens for keyboard input without polling?

16. How do you think the operating system preempts a process to schedule another one?

If you have extra time…

17. Thinking about `main()`'s behavior while `handleSignal()` is running, which of the following terms accurately describe(s) signal handling:

    concurrency parallelism

18. Explain your answer. It may help to review the explanation of how signals and interrupts work on the first page.

Run `java --add-exports java.base/jdk.internal.misc=ALL-UNNAMED BusyLoop.java` and make sure the CPU time limit is still being enforced. If you have switched terminals and it is not, run `ulimit -St 5` and retest.

19. Predict what would happen if the program installed a signal handler for SIGXCPU.

20. Test your hypothesis, adding the new code before the loop so it actually runs.