These state diagrams assume that a process can only reach the "Terminated" state from the "Running" state. How could a process in the "Runnable" or "Waiting" state transition to the "Terminated" state?
User could kill process before the process has called `exit()`!
Ex: Control-C a process
Ex: `kill -9 1831`
User could kill process before the process has called exit()!
Ex: Control-C a process
Ex: kill -9 1831
User could kill process before the process has called exit()!

Ex: Control-C a process
Ex: kill -9 1831

Generates SIGINT
Generates SIGKILL

Keyboard

“kill -9 1831”

Terminal + shell

SIGKILL

Foreground process
Define a simple C function that, when invoked, will eventually cause a stack overflow. Then describe how the stack overflow might lead to data corruption of heap objects.

```c
unsigned int factorial(unsigned int n){
    if(n == 1){
        return 1;
    }else{
        return n * factorial(n-1);
    }
}

factorial(6); //Works as expected.
factorial(0); //Disaster strikes!
```
Define a simple C function that, when invoked, will eventually cause a stack overflow. Then describe how the stack overflow might lead to data corruption of heap objects.

```c
unsigned int factorial(unsigned int n){
    if(n == 1){
        return 1;
    }else{
        return n * factorial(n-1);
    }
}
factorial(6); //Works as expected.
factorial(0); //Disaster strikes!
```

On a 32-bit machine, 0-1 = 4294967295

The FML number Integer underflow!
Case study: Linux kernel

Per-user-thread kernel stack
(system calls, exceptions)

thread_info

Stack start (high addr)

Stack end (low addr)

Per-cpu kernel stack
(hardware interrupts)

4 KB

4 KB

2/2/2015 CS161 Spring 2016