Concurrent Programming (Part 1)

Robert M. Dondero, Ph.D.
Princeton University

Objectives

- We will cover:
  - What a **process** is
  - How to **fork** and **wait** for processes
  - What a **thread** is
  - How to **spawn** and **join** threads

Agenda

- **Concurrency**
- Process-level concurrency
- Thread-level concurrency

Concurrency

- To implement concurrency...
- **Option 1**: **Process**-level concurrency
  - Multiple processes run concurrently
- **Option 2**: **Thread**-level concurrency
  - Multiple threads run concurrently within the same process

Concurrency

- **COS 217**
  - (Sometimes) covers **process**-level concurrency
    - As implemented in **C** via fork() and wait()
  - Does not cover **thread**-level concurrency
- **COS 333**
  - Covers **processes**-level concurrency
    - As implemented in **Python**
  - Covers **thread**-level concurrency

Agenda

- Concurrency
- **Process-level concurrency**
- Thread-level concurrency

Process Concurrency

- ***Program***
  - Executable code
- ***Process***
  - An instance of a program in execution
  - Each process has its own distinct **context**

Process Concurrency

- ***Context*** consists of:
  - Process id
  - Address space: TEXT, RODATA, DATA, BSS, HEAP, STACK
  - Processor state: general purpose registers, flags register, instruction pointer register, etc.

Process Concurrency

- Process-level concurrency
  - Process P1 ***forks*** child process P2
  - P1 and P2 run ***concurrently***
    - >1 processor available on computer => P1 and P2 run in ***parallel***
    - 1 processor available on computer => OS ***context switches*** between P1 and P2
      - OS “gives the processor” to P1
      - OS “gives the processor” to P2
      - …

Process Concurrency

- Example:
- On a Linux system…
  - Upon login, process running the `ssh` program forks a child process running the `bash` program
  - Process running the `ssh` program and process running the `bash` program run **concurrently**

Process Concurrency

- Example:
- On a Linux system…
  - Upon issuing a `ls` command at the `bash` prompt, process running the `bash` program forks a child process running the `ls` program
  - Process running the `ssh` program, process running the `bash` program, and process running the `ls` program run **concurrently**

Process Concurrency

- See **forking.py**

```
$ python forking.py
parent process termi
blue
blue
blue
blue
blue
blue process termina
red
red
red
red
red
red process terminat
$
```

```
$ python forking.py
parent process termi
red
red
red
red
red
red process terminat
blue
blue
blue
blue
blue
blue process termina
$
```

```
$ python forking.py
blue
parent process terminated
blue
blue
blue
blue
blue process terminated
red
red
red
red
red
red process terminated
$
```

Process Concurrency

- **Fact:**
  - A parent process should **wait for** its child processes to exit; in other words…
  - A parent process should **reap** its child processes that have exited
- **Definition**:
  - A ***zombie process*** is a process that has exited but has not been **waited for** (**reaped**) by its parent process

- Zombie processes needlessly clutter the operating system's data structures

Process Concurrency

- **Problem:**
  - forking.py creates zombie child processes
- **Solution:**
  - Define parent process to wait for (reap) its child processes…

Process Concurrency

- See **waiting.py**

```
$ python waiting.py
blue
blue
blue
blue
blue
blue process terminated
red
red
red
red
red
red process terminated
parent process terminated
$
```

```
$ python waiting.py
red
red
red
red
red
red process terminated
blue
blue
blue
blue
blue
blue process terminated
parent process terminated
$
```

Agenda

- Concurrency
- Process-level concurrency
- **Thread-level concurrency**

Thread Concurrency

- ***Thread***
  - A flow of control within a process
  - A process contains one or more threads
  - Within a process, all threads execute **concurrently**

Thread Concurrency

- **Thread-level** concurrency
  - Within P1, thread T1 ***spawns*** child thread T2
  - T1 and T2 run ***concurrently***
    - >1 processors available on computer => T1 and T2 run in ***parallel*** *
    - 1 processor available on computer => OS ***context switches*** between T1 and T2
  - (Relatively) **inexpensive** context switching

* In principle, but not in Python

Thread Concurrency

- Example…
- In a web browser
  - When you request a page…
  - Browser **spawns** a child thread
  - Child thread performs networking
  - Parent thread remains responsive to user input
  - Parent thread and child thread run **concurrently**

Thread Concurrency

- Example…
- In Java
  - At interpreter startup…
  - Interpreter **spawns** main thread and garbage collector (GC) thread
  - Main thread runs user code
  - GC thread reclaims garbage created by main thread (and other threads)
  - Main thread and GC thread run **concurrently**

Thread Concurrency

- Generalizing…
- The “main” thread runs at process startup
  - Other threads may run at process startup too
- The main thread can spawn other threads
- Note terminology:
  - One process **forks** another
  - One thread **spawns** another

Thread Concurrency

- See **spawning.py**

```
$ python spawning.py
blue
blue
blue
blue
blue
blue thread terminated
red
red
red
red
red
red thread terminated
main thread terminated
$
```

```
$ python spawning.py
blue
blue
blue
blue
red
red
red
red
red
red thread terminated
blue
main thread terminated
blue thread terminated
$
```

Thread Concurrency

- To compose a thread:
  - Define a subclass of `threading.Thread`
    - Override `run()` method
  - Instantiate an object of that class
- To spawn a thread:
  - Call object's `start()` method
    - `start()` does setup, calls `run()`
  - **Don't call `run()` directly!!!**

Thread Concurrency

- Main thread can ***join*** a child thread
  - Main thread can block until child thread terminates
- Note terminology
  - A parent **process** can **fork** and then **wait** for a child process
  - A parent **thread** can **spawn** and then **join** a child thread

Thread Concurrency

- See **joining.py**

```
$ python joining.py
blue
blue
blue
blue
blue
blue thread terminated
red
red
red
red
red
red thread terminated
main thread terminated
$
```

```
$ python joining.py
blue
blue
blue
red
red
red
red
red
red thread terminated
blue
blue
blue thread terminated
main thread terminated
$
```

Summary

- We have covered:
  - What a **process** is
  - How to **fork** and **wait** for processes
  - What a **thread** is
  - How to **spawn** and **join** threads