Programming with Concurrent Threads (Part 1)

Robert M. Dondero, Ph.D.
Princeton University

Objectives

- We will cover:
  - What a **thread** is
  - How to **spawn** and **join** threads
  - **Race conditions**

Agenda

- **Thread-level concurrency**
- Thread basics
- Process object sharing
- Thread object sharing
- Race conditions

Thread Concurrency

- To implement concurrency...
- **Option 1**: Process-level concurrency
  - Last lecture
- **Option 2**: Thread-level concurrency
  - This lecture

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

- Recall...
- **Process-level** concurrency
  - Process P1 ***forks*** process P2
  - P1 and P2 run ***concurrently***
    - >1 CPU available => P1 and P2 run in ***parallel***
    - 1 CPU available => OS ***context switches*** between P1 and P2
  - (Relatively) **expensive** context switching

Thread Concurrency

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

* But see the description of the **Python GIL** at the end of this lecture sequence

Thread Concurrency

- For this lecture, assume only 1 CPU is available

Thread Concurrency: Examples

- 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: Examples

- In Java
  - At interpreter startup…
  - Interpreter **spawns** main thread and garbage collector (GC) thread
  - Main thread executes 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

Agenda

- Thread-level concurrency
- **Thread basics**
- Process object sharing
- Thread object sharing
- Race conditions

Thread Basics: Spawning

- See **spawning.py**
  - The job:
    - Parent thread spawns “blue” child thread
    - Parent thread spawns “red” child thread
  - Three threads run concurrently
  - Context switches between machine lang instructions

13

Thread Basics: Spawning

[see slide]

Code notes:

To spawn a thread:
Define a subclass of `Thread`
Override `run()` method
Create an object of that class

To begin execution of a thread:
Call object’s `start()` method
`start()` calls `run()`
Don’t call `run()` directly

Thread Basics: Spawning

- See **spawning.py** (cont.)

```
$ 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 Basics: Spawning

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

15

Thread Basics: Spawning

[see slide]

Code notes:

To spawn a thread:
Define a subclass of `Thread`
Override `run()` method
Create an object of that class

To begin execution of a thread:
Call object's `start()` method
`start()` calls `run()`
Don't call `run()` directly

Thread Basics: Joining

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

Thread Basics: Joining

- See **joining.py**
  - The job:
    - Same as spawning.py, except…
    - Parent thread **joins** its child threads
  - Parent thread is *blocked* until both child threads exit

17

Thread Basics: Joining

[see slide]

Code notes:

`thread.join()`

Blocks current thread until `thread` terminates

Thread Basics: Joining

- See **joining.py** (cont.)

```
$ 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
$
```

18

Thread Basics: Joining

[see slide]

With multiple processes, a parent process should wait for (reap) its child threads

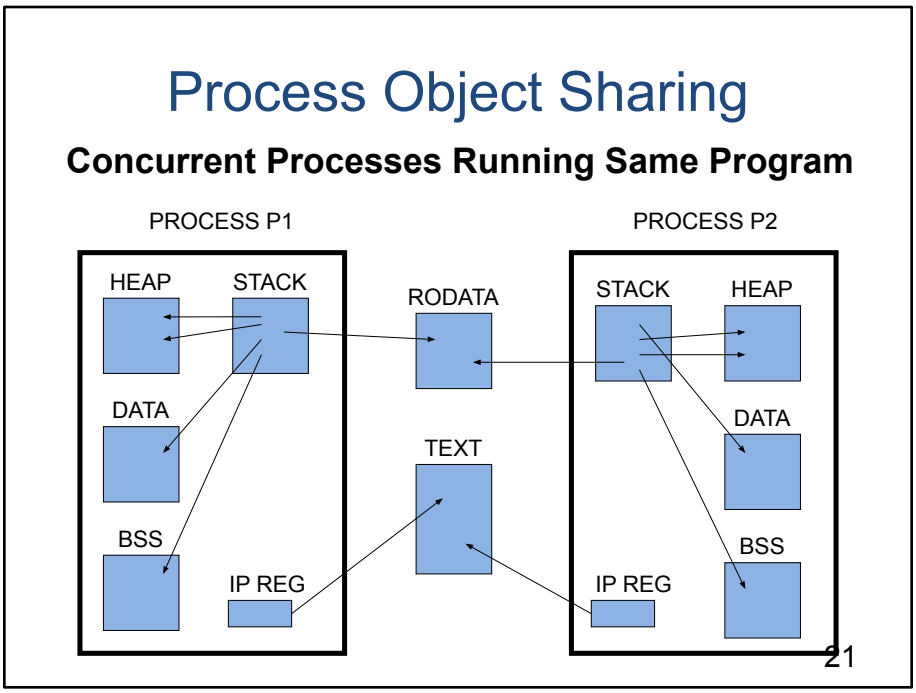
With multiple threads, a parent thread need not join its child threads

Agenda

- Thread-level concurrency
- Thread basics
- **Process object sharing**
- Thread object sharing
- Race conditions

Process Object Sharing

- **Process-level** concurrency
  - P1 and P2 **do not share** objects
    - P1 and P2 have (initially identical but) distinct memory address spaces



Thread Concurrency

[see slide]

Describe small to big

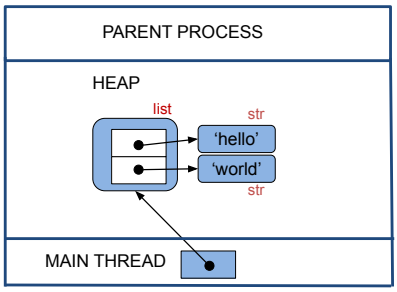
RODATA and TEXT sections are read-only
Shared via virtual memory

Process Object Sharing

- See **processsharing.py**
  - The job:
    - Artificial
    - What does it write?
      - Option A: `['hello', 'world']`
      - Option B: `['hello', 'COS333']`

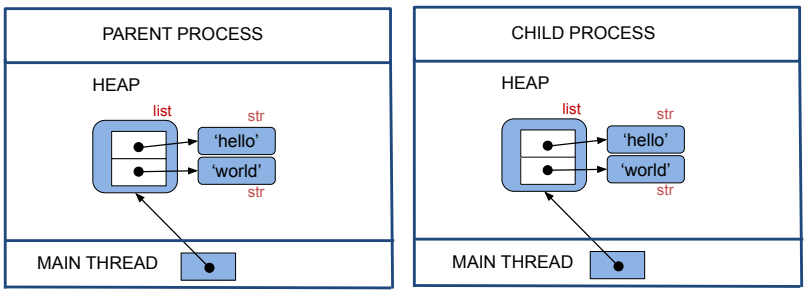
Process Object Sharing

Step 1: Parent process runs



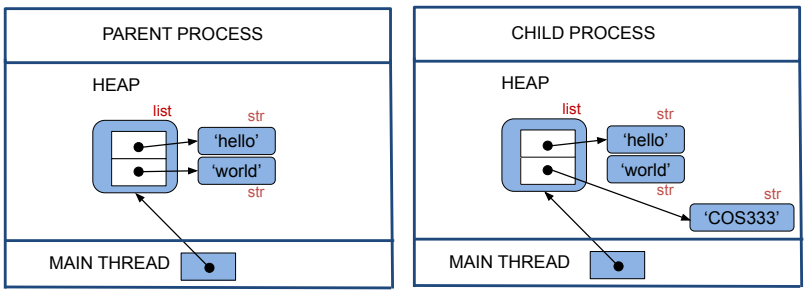
Process Object Sharing

Step 2: Parent process forks child process



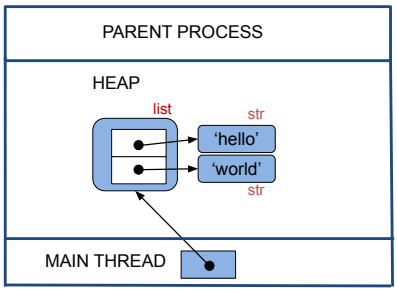
Process Object Sharing

Step 3: Parent process waits; child process runs



Process Object Sharing

Step 4: Child process terminates; parent process runs



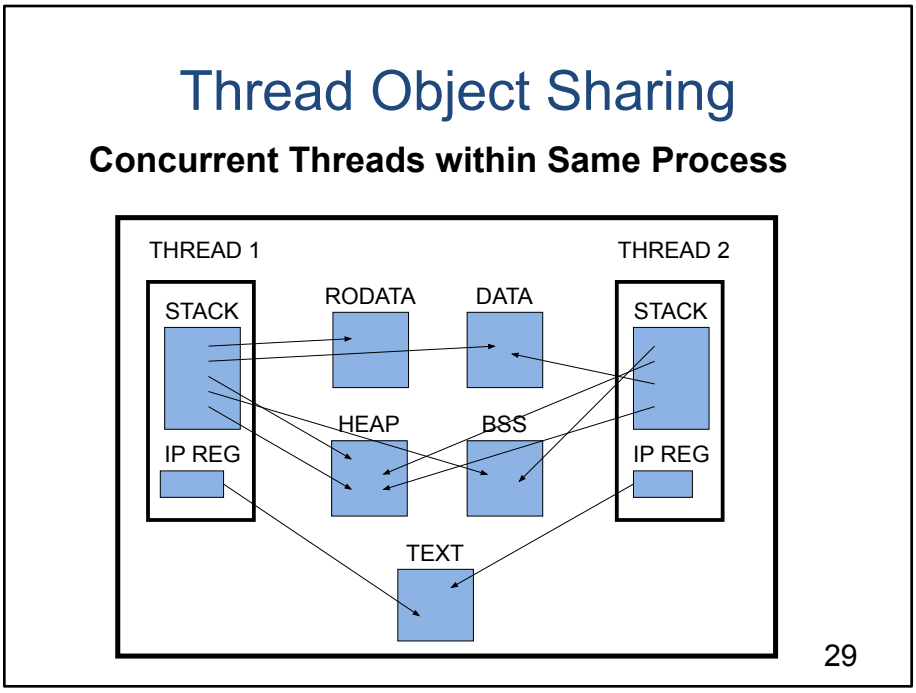
Writes `['hello', 'world']`

Agenda

- Thread-level concurrency
- Thread basics
- Process object sharing
- **Thread object sharing**
- Race conditions

Thread Object Sharing

- **Thread-level** concurrency
  - T1 and T2 **share** objects
    - T1 and T2 have distinct **STACK** sections
    - T1 and T2 share the RODATA, DATA, BSS, and **HEAP** sections



Thread Concurrency

[see slide]

Describe small to big

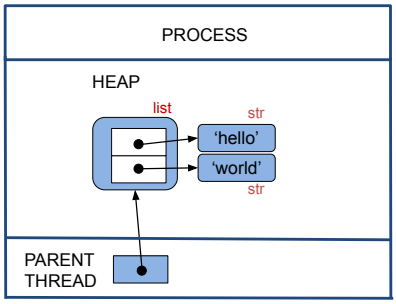
Note especially that objects on the heap are shared

Thread Object Sharing

- See **threadsharing.py**
  - The job:
    - Artificial
    - What does it write?
      - Option A: `['hello', 'world']`
      - Option B: `['hello', 'COS333']`

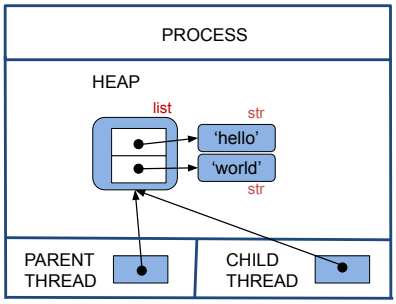
Thread Object Sharing

Step 1: Parent thread runs



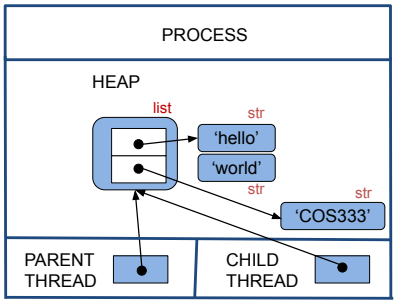
Thread Object Sharing

Step 2: Parent thread spawns child thread



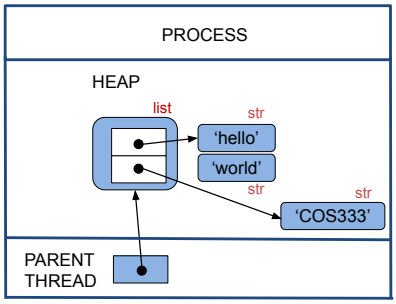
Thread Object Sharing

Step 3: Parent thread joins; child thread runs



Thread Object Sharing

Step 4: Child thread terminates; parent thread runs



Writes `['hello', 'COS333']`

Agenda

- Thread-level concurrency
- Thread basics
- Process object sharing
- Thread object sharing
- **Race conditions**

Race Conditions

- **Problem**:
  - Threads can share objects
  - Danger if multiple threads update/access the same object concurrently
  - ***Race condition***
    - Outcome depends upon thread scheduling

Race Conditions: Example

- See **race.py**
  - The job:
    - Bank account initial balance: 0
    - Deposit 1, 10 times
    - Withdraw 2, 5 times
    - Bank account final balance: 0?

37

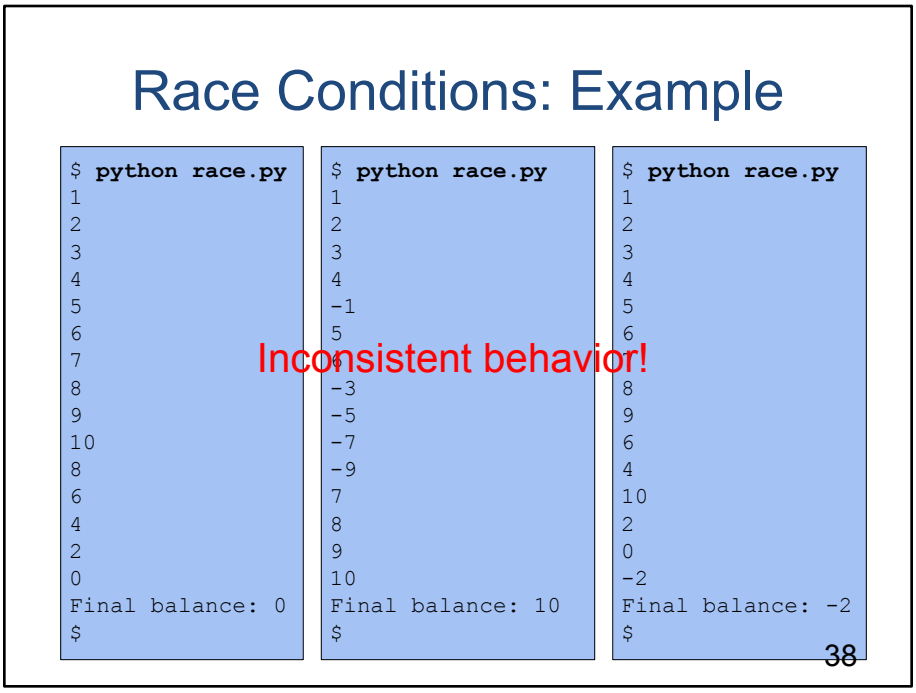
Race Conditions: Example

[see slide]

Code notes

Study code statically

Do you see a problem?



Race Conditions: Example

[see slide]

Code notes

- Study code dynamically
- Note the problem

Race Conditions: Locking

- **Observation**:
  - While a thread is executing `deposit()` or `withdraw()` on a particular `bankAcct` object…
  - No other thread should be able to execute `deposit()` or `withdraw()` on that `bankAcct` object
- **Solution**: ***Locking***
  - Continued in next lecture

Summary

- We have covered:
  - What a **thread** is
  - How to **spawn** and **join** threads
  - **Race conditions**