

Concurrent Programming (Part 2)

Copyright © 2025 by
Robert M. Dondero, Ph.D.
Princeton University

Objectives

- We will cover:
 - Concurrent processes vs. concurrent threads
 - Race conditions
 - Preventing race conditions
 - Thread safety

Agenda

- **Process vs. thread concurrency**
- Race conditions
- Preventing race conditions
- Thread safety

Process vs. Thread Concurrency

- Difference #1
 - **Process**-level concurrency
 - Multiple **processes** run concurrently
 - Parent process **forks** (and **waits** for) a child process
 - **Thread**-level concurrency
 - Multiple **threads** run concurrently within the same process
 - Within a process, parent thread **spawns** (and **joins**) a child thread

Process vs. Thread Concurrency

- Difference #2
 - **Process**-level concurrency
 - Forking & context switching are relatively **slow**
 - **Thread**-level concurrency
 - Spawning & context switching are relatively **fast**

Process vs. Thread Concurrency

- Difference #3
 - **Process**-level concurrency
 - Concurrent processes **do not share objects**
 - **Thread**-level concurrency
 - Concurrent threads **do share objects**
- Elaboration...

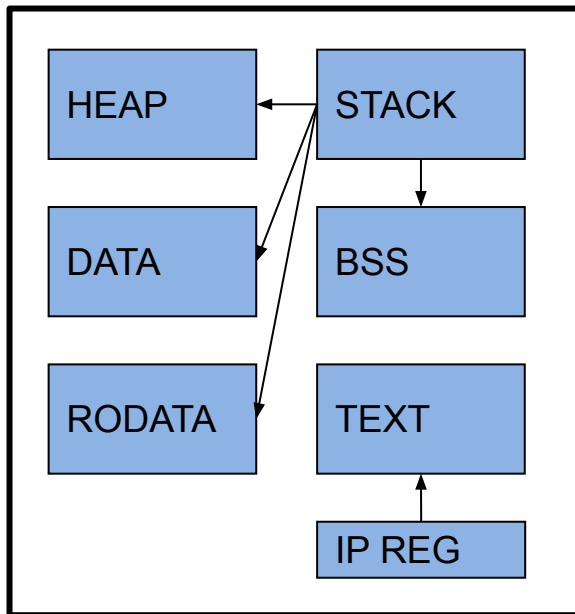
Process vs. Thread Concurrency

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

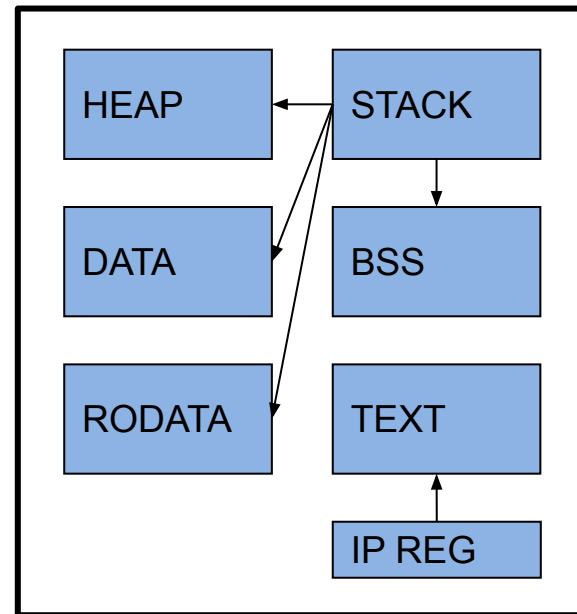
Process vs. Thread Concurrency

Concurrent Processes: Conceptually

PROCESS P1

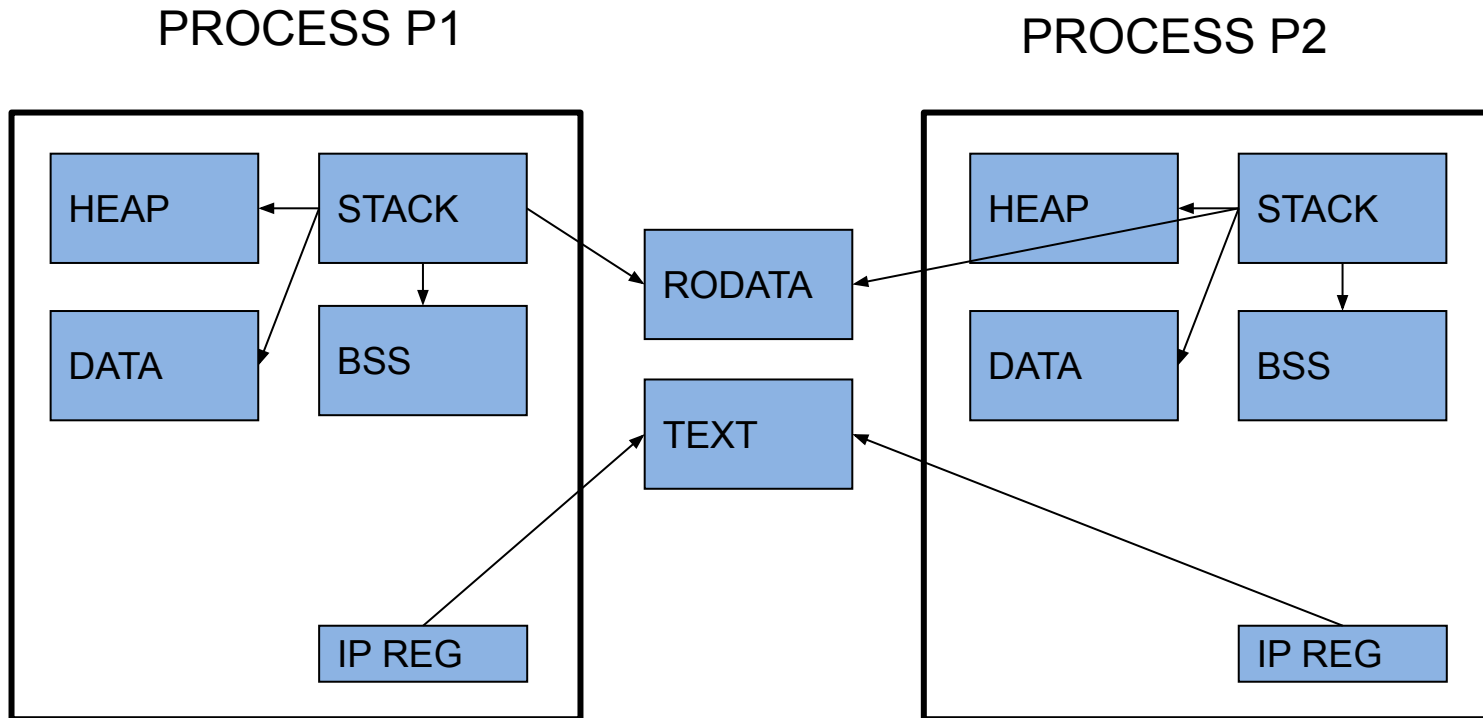


PROCESS P2



Process vs. Thread Concurrency

Concurrent Processes: In Reality



Process vs. Thread Concurrency

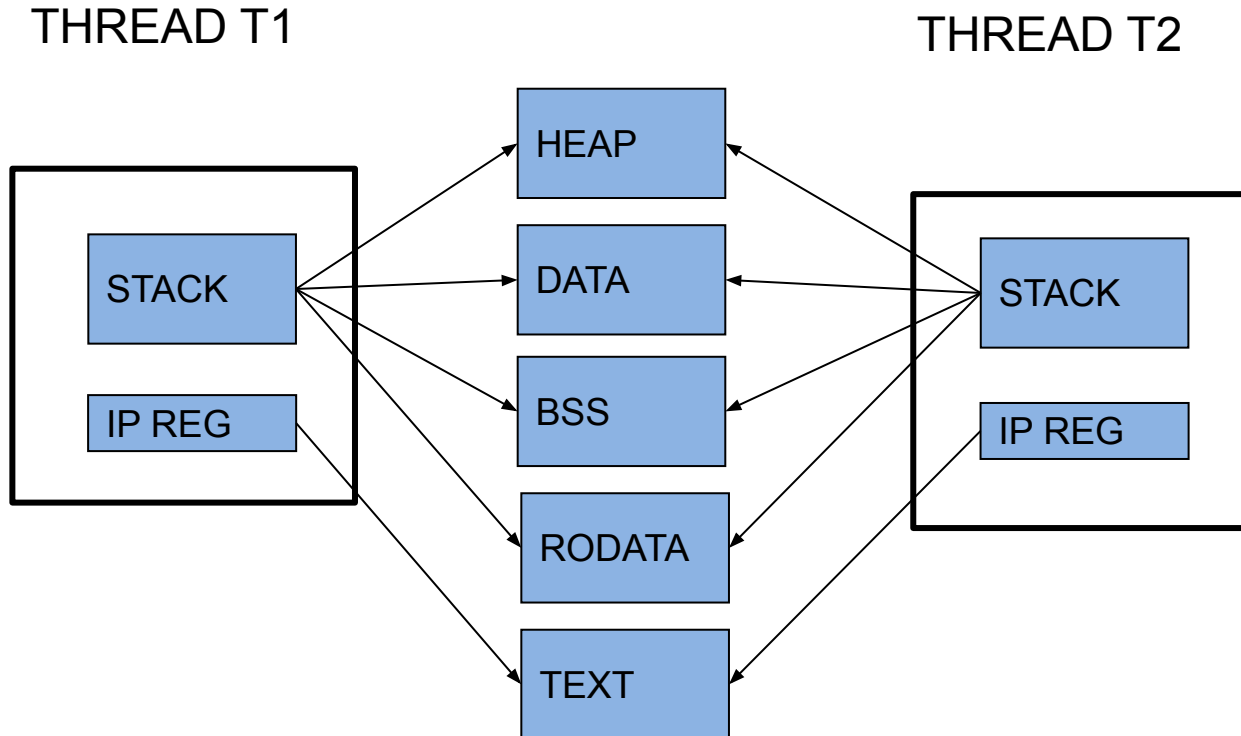
- See **processsharing.py**

Process vs. Thread Concurrency

- **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

Process vs. Thread Concurrency

Concurrent Threads



Process vs. Thread Concurrency

- See **threadsharing.py**

Agenda

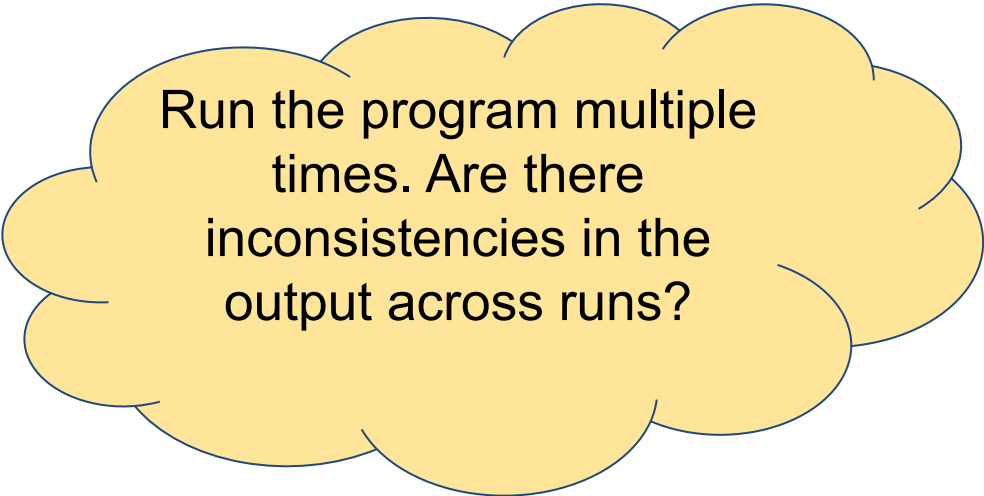
- Process vs. thread concurrency
- **Race conditions**
- Preventing race conditions
- Thread safety

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

- See **race.py**



Run the program multiple times. Are there inconsistencies in the output across runs?

Race Conditions

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

```
$ python race.py
1
2
3
4
5
6
7
8
9
10
8
6
4
2
0
Final balance: 0
$
```

```
$ python race.py
1
2
3
4
-1
5
6
-3
-5
-7
-9
7
8
9
10
Final balance: 10
$
```

```
$ python race.py
1
2
3
4
5
6
7
8
9
6
4
10
2
0
-2
Final balance: -2
$
```

Race Conditions

- **Note:**
 - Use of shared `BankAcct` object by multiple threads causes unpredictable behavior
 - `race.py` contains a **race condition**

Agenda

- Process vs. thread concurrency
- Race conditions
- **Preventing race conditions**
- Thread safety

Preventing Race Conditions

- **Observation:**

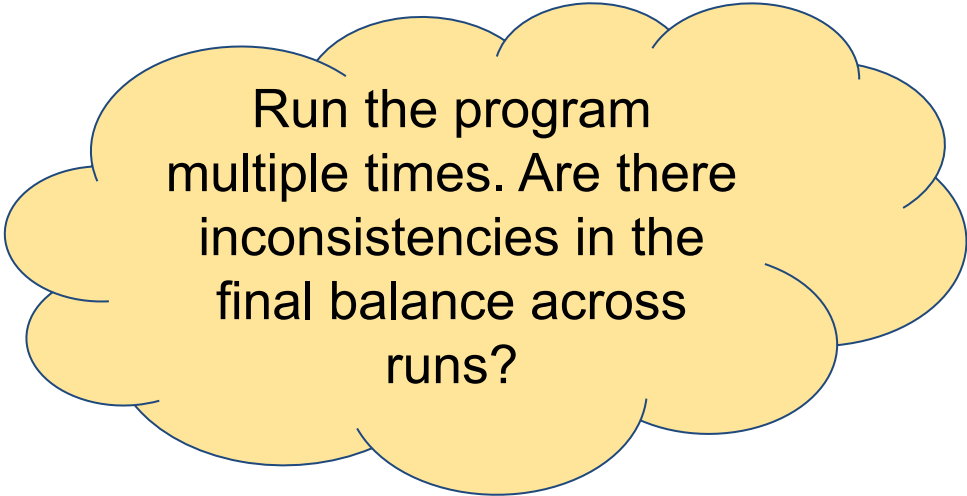
- While a thread is executing `deposit()` or `withdraw()` on a particular `bank_acct` object...
- No other thread should be able to execute `deposit()` or `withdraw()` on that `bank_acct` object

Preventing Race Conditions

- **Solution:** *Locking*
 - Each object has an associated **lock**
 - Current thread **acquires** lock on X
 - Other threads cannot acquire lock on X until current thread **releases** lock on X
 - (Adds lots of overhead)

Preventing Race Conditions

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



Run the program multiple times. Are there inconsistencies in the final balance across runs?

Preventing Race Conditions

- See locking.py (cont.)

```
$ python locking.py
```

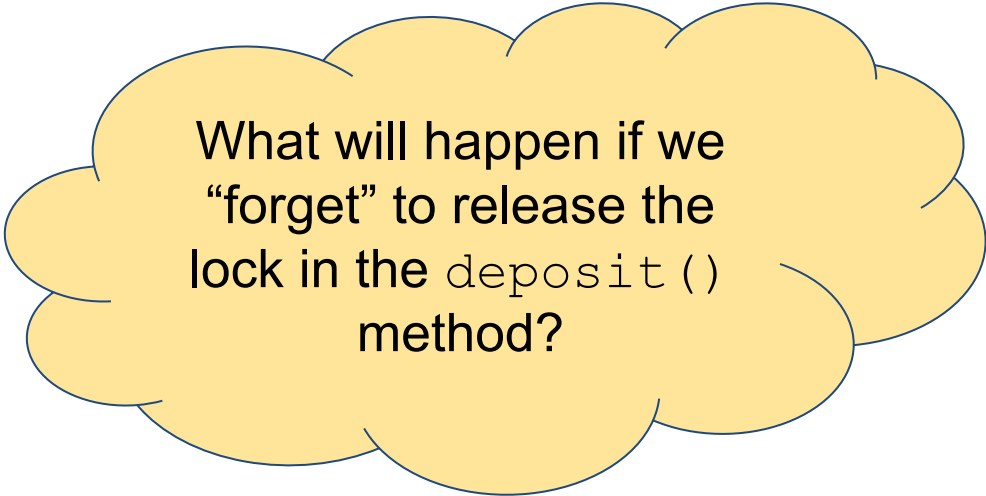
```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
8  
6  
4  
2  
0  
Final balance: 0  
$
```

```
$ python locking.py
```

```
1  
2  
3  
1  
-1  
-3  
-5  
-7  
-6  
-5  
-4  
-3  
-2  
-1  
0  
Final balance: 0  
$
```

Preventing Race Conditions

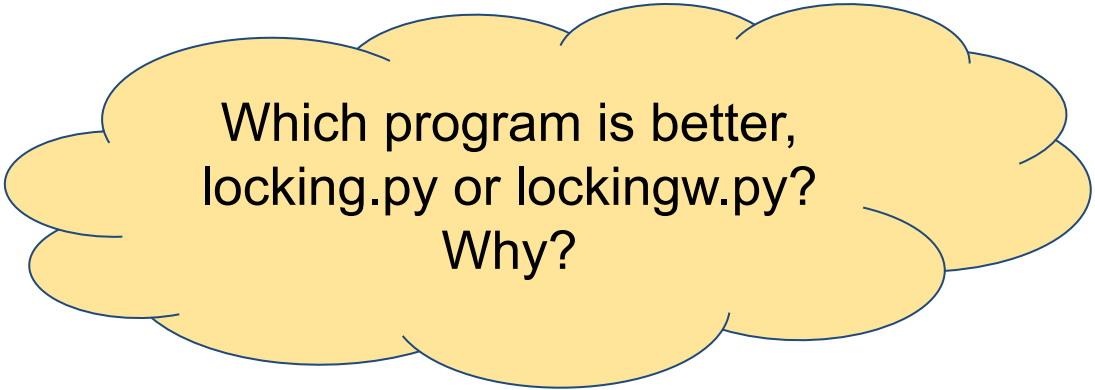
- See **locking.py** (cont.)



What will happen if we
“forget” to release the
lock in the `deposit()`
method?

Preventing Race Conditions

- See **lockingw.py**
 - Uses `with` statement



Which program is better,
locking.py or lockingw.py?
Why?

Agenda

- Process vs. thread concurrency
- Race conditions
- Preventing race conditions
- **Thread safety**

Thread Safety

- Recall `locking.py`
 - A context switch can occur between any 2 **machine lang** instructions
 - Implications:
 - The `get_balance()` method should be protected by locking
 - The `_balance` field should be private
 - But cannot be

Thread Safety

- *Thread safety*
 - Oversimplification...
 - An object is **thread-safe** if all of its methods are “locked” & all of its fields are private

Thread Safety

- **Java**

- Methods can be locked (`synchronized`)
- Fields **can** be private
- Objects can be thread-safe

- **Python**

- Methods can be locked
- Fields **cannot** be private
- Any object that has fields cannot be thread-safe

Aside: Other Languages

	Python	Java	C
Concurrency via multiple processes?	Yes, via standard library	Yes, via standard library	Yes, via standard library
Concurrency via multiple threads?	Yes, via standard library	Yes, via language and standard library *	Yes, via <code>pthread</code> library

* See Appendix 1

Lecture Summary

- In this lecture we covered:
 - Concurrent processes vs. concurrent threads
 - Race conditions
 - Preventing race conditions
 - Thread safety
- See also:
 - **Appendix 1:** Multithreaded programming in Java

Appendix 1:

Multi-Threaded Programming in Java

Multithreaded Pgmming in Java

- See **Spawning.java**
- See **Joining.java**
- See **Race.java**
- See **Locking.java**