CS 340

Daemons, Zombies, and Deadlocks

(The spooky side of CS)

Learning Goals

- Improve understanding of threads and other related vocabulary
- Understanding thread safety concerns
- Be able to analyze multithreading situations

Plan for Today

- Review
- Thread Safety
    - Deadlocks
    - Think-Pair-Share situations

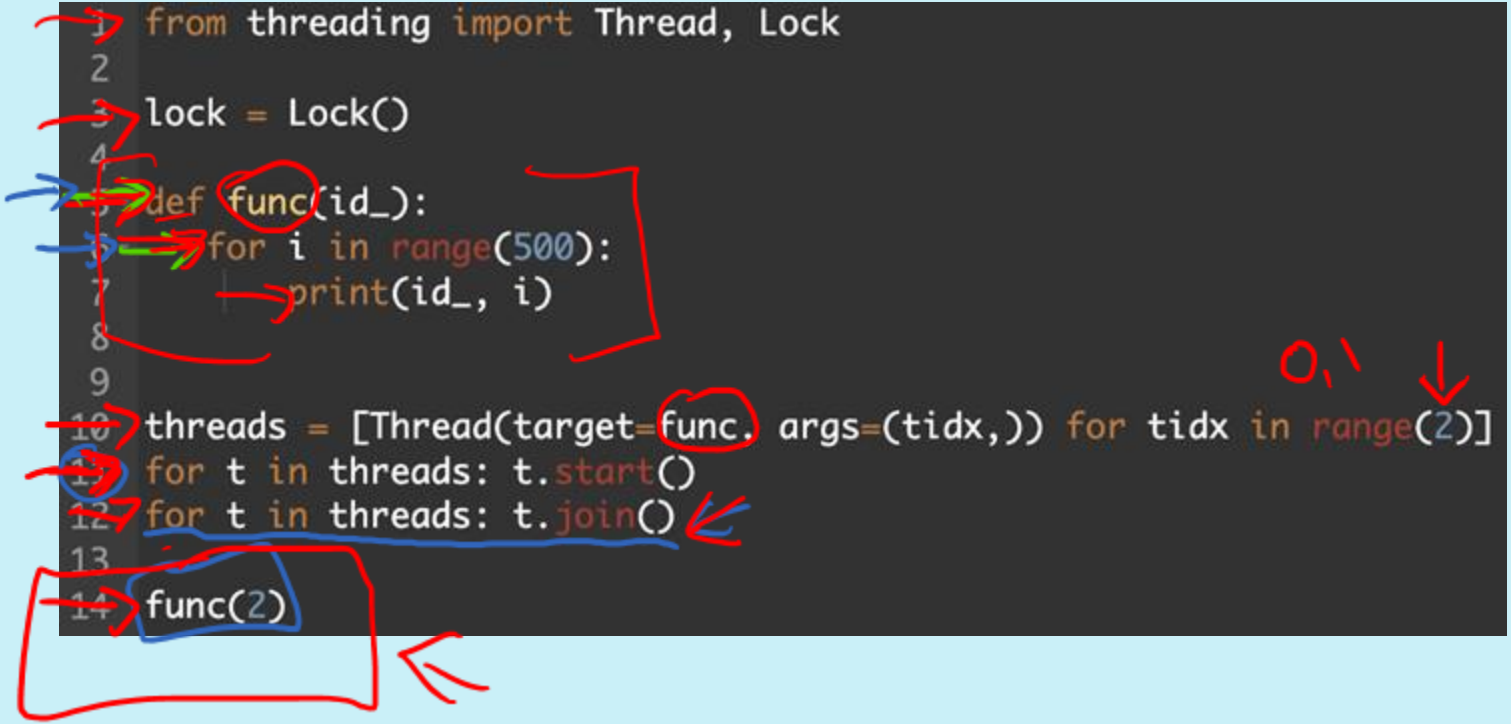
Practice with Clickers (0.1% extra credit)

Threads Vocab I

Thread - Same VM, separate PC and registers

Concurrency - Two things both have started and neither have finished

Parallel - Two things making forward progress at the same time


from threading import Thread, Lock
lock = Lock()
def func(id_):
for i in range(500):
print(id_, i)
0,1
threads = [Thread(target=func, args=(tidx,)) for tidx in range(2)]
for t in threads: t.start()
for t in threads: t.join()
func(2)

How many threads could be running concurrently? (max)

```
1  from threading import Thread, Lock
2
3  lock = Lock()
4  count = 0
5  def func(id_):
6      global count
7      print(id_, "running")
8      with lock:
9          print(id_, "has lock")
10         for i in range(count):
11             print(id_, i)
12         count += 1
13
14
15 threads = [Thread(target=func, args=(tidx,)) for tidx in range(3)]
16 for t in threads: t.start()
17 func(3)
18 for t in threads: t.join()
19 thr = Thread(target=func, args=(4,))
20 thr.start()
21
```

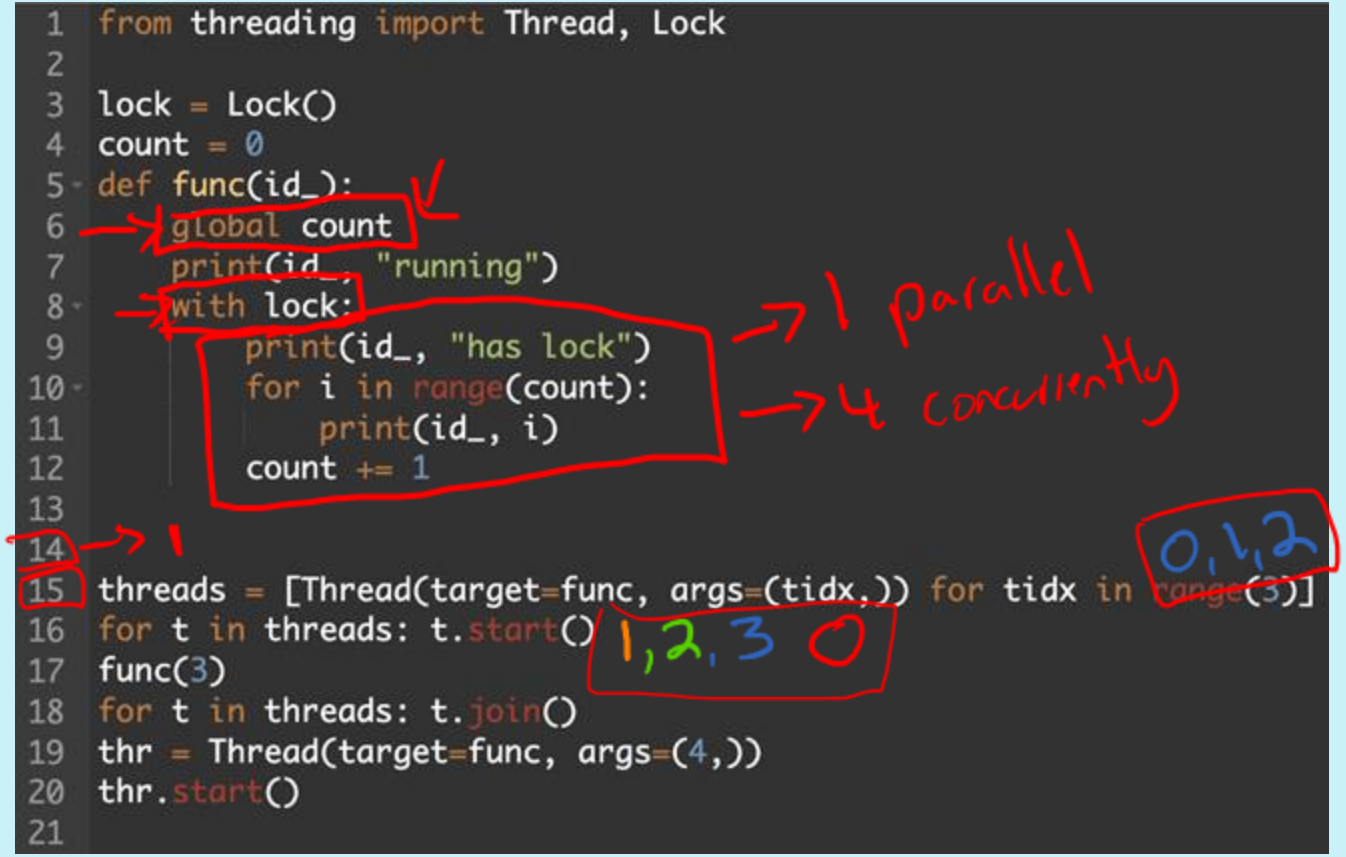
clicker.cs.illinois.edu

Q1

~Code~
340

How many threads could be running in parallel? (max)

```
1  from threading import Thread, Lock
2
3  lock = Lock()
4  count = 0
5  def func(id_):
6      global count
7      print(id_, "running")
8      with lock:
9          print(id_, "has lock")
10         for i in range(count):
11             print(id_, i)
12         count += 1
13
14
15 threads = [Thread(target=func, args=(tidx,)) for tidx in range(3)]
16 for t in threads: t.start()
17 func(3)
18 for t in threads: t.join()
19 thr = Thread(target=func, args=(4,))
20 thr.start()
21
```



clicker.cs.illinois.edu

Q2

~Code~
340

Threads Vocab II

Daemon - A thread that isn't tied to the execution of the main program.

Zombie - A thread that is done running but is still holding onto resources

Race Condition - threads "race" to modify and access variables they share.

My code does different things everytime I run it! Which of the following could be the issue?

clicker.cs.illinois.edu

Q3

~Code~
340

A) Multiple threads are all trying to access a local function variable at the same time.
B) Multiple Daemon threads are all trying to access a local function variable at the same time.
C) The program keeps creating zombie threads instead of releasing them.
D) **All** the code my threads run are put into critical areas through locks.
E) None of the above.

My code stops making forward progress sometimes. Which of the following could be the issue?

clicker.cs.illinois.edu

Q4

~Code~
340

A) A race condition with threads.
B) An infinite loop.
C) The program keeps creating zombie threads instead of releasing them.
D) None of the above.

Thread Safety

A system that cannot make forward progress

Deadlock - The system does not change

Livelock - The system changes but is stuck in a loop

Starvation - A part of the system never gets any resources


ARK
NOVA

Allie's Algorithm

1. get AN
2. get MW
3. play
4. return MW
5. return AN

Jordan's Algorithm

1. get MW
2. get AN
3. play
4. return AN
5. return MW

Thread Safety

A system that cannot make forward progress

Deadlock - The system does not change

Livelock - The system changes but is stuck in a loop

Starvation - A part of the system never gets any resources

Coffman Conditions

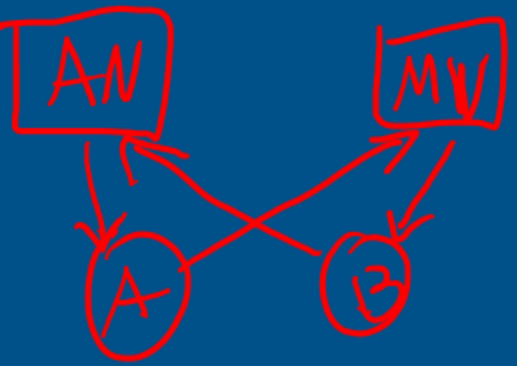
If there is a deadlock then these conditions are true

- Circular Wait
  - Mutual Exclusion
  - Hold and Wait
  - No preemption

only 1 resource

once obtained Keep it

nothing can force you to give up




ARK
NOVA

Thread Safety

A system that cannot make forward progress

Deadlock - The system does not change

Livelock - The system changes but is stuck in a loop

Starvation - A part of the system never gets any resources


ARK
NOVA

Thread Safety

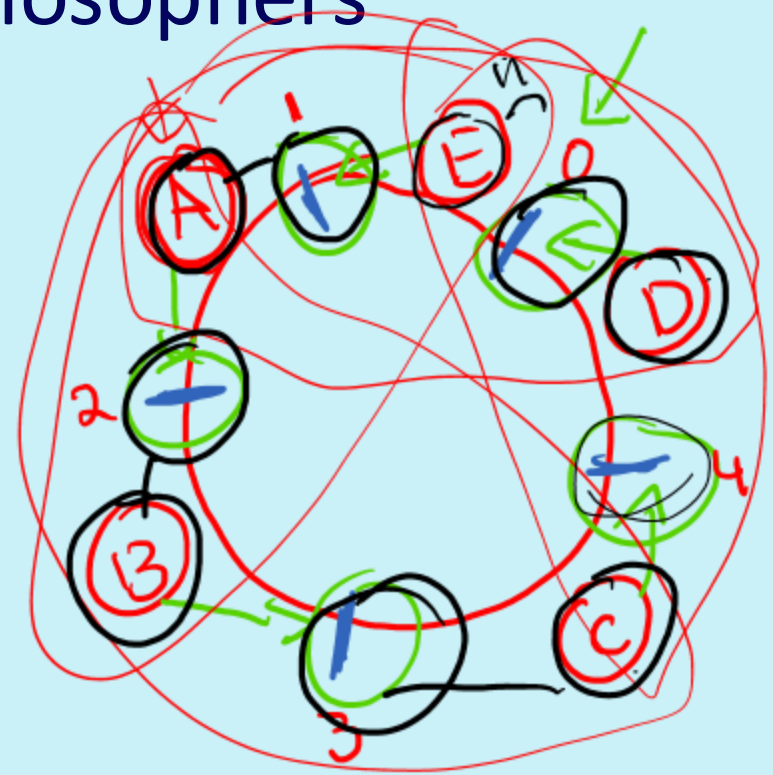
A system that cannot make forward progress

Deadlock - The system does not change

Livelock - The system changes but is stuck in a loop

Starvation - A part of the system never gets any resources

Dining Philosophers



1. grab lowest
2. grab right
3. eat
4. put down
5. think

Dining Philosophers

Speaker Example

Suppose you attend an event where students get to share their ideas on the main quad. The event has two tools for speakers: a microphone to help speakers be heard and a stand to help speakers be seen; each can be used by only one speaker at a time. Every speaker either wants to be seen or heard, and some want to be both seen and heard.

The event organizers are considering several policies to managing speakers, listed below.

They think a policy is **bad** if it could create deadlock, with no speaker able to speak.

They think a policy is **so-so** if it isn't bad but might make people use a tool they don't want to use.

They think a policy is **good** if it ensures that everyone can speak with just the tool that they want to use.

They **don't care** if several people are talking at once, or if speakers get ineterrupted and have to finish their remarks later, or order people speak in.

Speaker Example

clicker.cs.illinois.edu

Q5

~Code~
340

The event organizers are considering several policies to managing speakers, listed below.

They think a policy is **bad** if it could create deadlock, with no speaker able to speak.

They think a policy is **so so** if it isn't bad but might make people use a tool they don't want to use.

They think a policy is **good** if it ensures that everyone can speak with just the tool that they want to use.

They **don't care** if several people are talking at once, or if speakers get ineterrupted and have to finish their remarks later, or order people speak in.

People who want the microphone and the stand must always take the microphone first.

Speaker Example

clicker.cs.illinois.edu

Q6

~Code~
340

The event organizers are considering several policies to managing speakers, listed below.

They think a policy is **bad** if it could create deadlock, with no speaker able to speak.

They think a policy is **so-so** if it isn't bad but might make people use a tool they don't want to use.

They think a policy is **good** if it ensures that everyone can speak with just the tool that they want to use.

They **don't care** if several people are talking at once, or if speakers get ineterrupted and have to finish their remarks later, or order people speak in.

If the stand is unoccupied and you want the stand, take the stand; if ~~your~~ the microphone is unheld and you want the microphone, take the microphone.

Speaker Example

clicker.cs.illinois.edu

Q7

~Code~
340

The event organizers are considering several policies to managing speakers, listed below.

They think a policy is **bad** if it could create deadlock, with no speaker able to speak.

They think a policy is ~~so-so~~ if it isn't bad but might make people use a tool they don't want to use.

They think a policy is **good** if it ensures that everyone can speak with just the tool that they want to use.

They **don't care** if several people are talking at once, or if speakers get ineterrupted and have to finish their remarks later, or order people speak in.

Anyone can take any unoccupied tool at any time. If you have one tool and wait for the other for a full minute, you have to release the tool you're holding and can't pick it up again until someone else has used it.

Speaker Example

The event organizers are considering several policies to managing speakers, listed below.

They think a policy is **bad** if it could create deadlock, with no speaker able to speak.

They think a policy is **so-so** if it isn't bad but might make people use a tool they don't want to use.

They think a policy is **good** if it ensures that everyone can speak with just the tool that they want to use.

They **don't care** if several people are talking at once, or if speakers get ineterrupted and have to finish their remarks later, or order people speak in.

clicker.cs.illinois.edu

Q8

~Code~
340

Anyone can take any unoccupied tool at any time. Additionally, if you have one tool and want the other tool too, you can take it from whoever has it if your UIN is a larger number than theirs.

preemption

Speaker Example

clicker.cs.illinois.edu

Q9

~Code~
340

The event organizers are considering several policies to managing speakers, listed below.

They think a policy is **bad** if it could create deadlock, with no speaker able to speak.

They think a policy is **so-so** if it isn't bad but might make people use a tool they don't want to use.

They think a policy is **good** if it ensures that everyone can speak with just the tool that they want to use.

They **don't care** if several people are talking at once, or if speakers get ineterrupted and have to finish their remarks later, or order people speak in.

You can only take the stand if you already have the micrphone.

Speaker Example

clicker.cs.illinois.edu

Q10

~Code~
340

The event organizers are considering several policies to managing speakers, listed below.

They think a policy is **bad** if it could create deadlock, with no speaker able to speak.

They think a policy is **so-so** if it isn't bad but might make people use a tool they don't want to use.

They think a policy is **good** if it ensures that everyone can speak with just the tool that they want to use.

They **don't care** if several people are talking at once, or if speakers get ineterrupted and have to finish their remarks later, or order people speak in.

There's an MC who assigns speaker order. The next speaker in line takes the tools they want and then speaks.

What to do if in a deadlock?

A system that cannot make forward progress

1. Analyze your threading logic outside of the code to determine the location and situation.
    a. Remove a Coffman condition
2. Turn it off and back on again and hope for the best
3. Remove all threading

What is coming up

- Finish HW (due 12:30pm Thursday [after spring break])
- Work on MP 6 (due 11:59pm Tuesday [after spring break])
- Read website text for more details and information
- Have a relaxing spring break!