

Locks and Synchronization

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require

synchronization
code for restaurant simulation

1 Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

2 Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

3 Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

MCS 275 Lecture 31
Programming Tools and File Management
Jan Verschelde, 31 March 2010

Locks and Synchronization

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

1 Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

2 Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

3 Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Threads for Simulation

simulating arrival of customers

Customers arriving at a fastfood restaurant:

- 1 wait for their order to be ready;
- 2 spend some time eating and then leave.

We use *threads* to implement the customers.

Inheriting from the class `Thread`,
the data object attributes are:

- 1 the name of the customer;
- 2 waiting and eating time.

The names are entered when threads are born.

Waiting and eating times are generated via

`random.randint()` in the `__init__`.

Threads for Simulation

simulating arrival of customers

Customers arriving at a fastfood restaurant:

- 1 wait for their order to be ready;
- 2 spend some time eating and then leave.

We use *threads* to implement the customers.

Inheriting from the class `Thread`,
the data object attributes are:

- 1 the name of the customer;
- 2 waiting and eating time.

The names are entered when threads are born.

Waiting and eating times are generated via

`random.randint()` in the `__init__`.

Threads for Simulation

simulating arrival of customers

Customers arriving at a fastfood restaurant:

- 1 wait for their order to be ready;
- 2 spend some time eating and then leave.

We use *threads* to implement the customers.

Inheriting from the class `Thread`,
the data object attributes are:

- 1 the name of the customer;
- 2 waiting and eating time.

The names are entered when threads are born.

Waiting and eating times are generated via

`random.randint()` in the `__init__`.

Threads for Simulation

simulating arrival of customers

Customers arriving at a fastfood restaurant:

- 1 wait for their order to be ready;
- 2 spend some time eating and then leave.

We use *threads* to implement the customers.

Inheriting from the class `Thread`,
the data object attributes are:

- 1 the name of the customer;
- 2 waiting and eating time.

The names are entered when threads are born.

Waiting and eating times are generated via

`random.randint()` in the `__init__`.

Threads for Simulation

simulating arrival of customers

Customers arriving at a fastfood restaurant:

- 1 wait for their order to be ready;
- 2 spend some time eating and then leave.

We use *threads* to implement the customers.

Inheriting from the class `Thread`,
the data object attributes are:

- 1 the name of the customer;
- 2 waiting and eating time.

The names are entered when threads are born.

Waiting and eating times are generated via

`random.randint()` in the `__init__`.

Threads for Simulation

simulating arrival of customers

Customers arriving at a fastfood restaurant:

- 1 wait for their order to be ready;
- 2 spend some time eating and then leave.

We use *threads* to implement the customers.

Inheriting from the class `Thread`,
the data object attributes are:

- 1 the name of the customer;
- 2 waiting and eating time.

The names are entered when threads are born.

Waiting and eating times are generated via

`random.randint()` in the `__init__`.

Threads for Simulation

simulating arrival of customers

Customers arriving at a fastfood restaurant:

- 1 wait for their order to be ready;
- 2 spend some time eating and then leave.

We use *threads* to implement the customers.

Inheriting from the class `Thread`,
the data object attributes are:

- 1 the name of the customer;
- 2 waiting and eating time.

The names are entered when threads are born.

Waiting and eating times are generated via

`random.randint()` in the `__init__`.

Threads for Simulation

simulating arrival of customers

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Customers arriving at a fastfood restaurant:

- 1 wait for their order to be ready;
- 2 spend some time eating and then leave.

We use *threads* to implement the customers.

Inheriting from the class `Thread`, the data object attributes are:

- 1 the name of the customer;
- 2 waiting and eating time.

The names are entered when threads are born.

Waiting and eating times are generated via `random.randint()` in the `__init__`.

Threads for Simulation

simulating arrival of customers

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Customers arriving at a fastfood restaurant:

- 1 wait for their order to be ready;
- 2 spend some time eating and then leave.

We use *threads* to implement the customers.

Inheriting from the class `Thread`, the data object attributes are:

- 1 the name of the customer;
- 2 waiting and eating time.

The names are entered when threads are born.

Waiting and eating times are generated via

`random.randint()` in the `__init__`.

Running the Simulation

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

```
$ python fastfood.py
give #customers : 3
    name of customer 0 : A
    name of customer 1 : B
    name of customer 2 : C
starting the simulation
A is waiting for 1 time units
B is waiting for 1 time units
C is waiting for 4 time units
simulation has started
A waited 1 time units
B waited 1 time units
B ate for 2 time units
C waited 4 time units
A ate for 6 time units
C ate for 5 time units
$
```

Running the Simulation

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

```
$ python fastfood.py
give #customers : 3
    name of customer 0 : A
    name of customer 1 : B
    name of customer 2 : C
starting the simulation
A is waiting for 1 time units
B is waiting for 1 time units
C is waiting for 4 time units
simulation has started
A waited 1 time units
B waited 1 time units
B ate for 2 time units
C waited 4 time units
A ate for 6 time units
C ate for 5 time units
$
```

Running the Simulation

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require

synchronization
code for restaurant simulation

```
$ python fastfood.py
give #customers : 3
    name of customer 0 : A
    name of customer 1 : B
    name of customer 2 : C
starting the simulation
A is waiting for 1 time units
B is waiting for 1 time units
C is waiting for 4 time units
simulation has started
A waited 1 time units
B waited 1 time units
B ate for 2 time units
C waited 4 time units
A ate for 6 time units
C ate for 5 time units
$
```

31 Mar 2010

Running the Simulation

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require

synchronization
code for restaurant simulation

```
$ python fastfood.py
give #customers : 3
    name of customer 0 : A
    name of customer 1 : B
    name of customer 2 : C
starting the simulation
A is waiting for 1 time units
B is waiting for 1 time units
C is waiting for 4 time units
simulation has started
A waited 1 time units
B waited 1 time units
B ate for 2 time units
C waited 4 time units
A ate for 6 time units
C ate for 5 time units
$
```

Running the Simulation

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require

synchronization
code for restaurant simulation

```
$ python fastfood.py
give #customers : 3
    name of customer 0 : A
    name of customer 1 : B
    name of customer 2 : C
starting the simulation
A is waiting for 1 time units
B is waiting for 1 time units
C is waiting for 4 time units
simulation has started
A waited 1 time units
B waited 1 time units
B ate for 2 time units
C waited 4 time units
A ate for 6 time units
C ate for 5 time units
$
```

Structure of the Script

fastfood.py

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script taketurns.py

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

```

from threading import *
class Customer(Thread):
    """
    Customer orders food and eats.
    """
    def __init__(self,t):
        """
        Initializes customer with name t,
        generates waiting and eating time.
        """
    def run(self):
        """
        Customer waits for order and eats.
        writes three messages.
        """
def main():
    "defines and starts threads"

```

Structure of the Script

fastfood.py

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script taketurns.py

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

```

from threading import *
class Customer(Thread):
    """
    Customer orders food and eats.
    """
    def __init__(self,t):
        """
        Initializes customer with name t,
        generates waiting and eating time.
        """
    def run(self):
        """
        Customer waits for order and eats.
        writes three messages.
        """
def main():
    "defines and starts threads"

```

Structure of the Script

fastfood.py

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script taketurns.py

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

```

from threading import *
class Customer(Thread):
    """
    Customer orders food and eats.
    """
    def __init__(self,t):
        """
        Initializes customer with name t,
        generates waiting and eating time.
        """
    def run(self):
        """
        Customer waits for order and eats.
        writes three messages.
        """

def main():
    "defines and starts threads"

```

Structure of the Script

fastfood.py

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script taketurns.py

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

```

from threading import *
class Customer(Thread):
    """
    Customer orders food and eats.
    """
    def __init__(self,t):
        """
        Initializes customer with name t,
        generates waiting and eating time.
        """
    def run(self):
        """
        Customer waits for order and eats.
        writes three messages.
        """
def main():
    "defines and starts threads"

```

Locks and Synchronization

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

1 Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

2 Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

3 Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

The Function main()

in the script `fastfood.py`

```
def main():
    "defines and starts threads"
    n = input('give #customers : ')
    T = []
    for i in range(0,n):
        s = 'name of customer %d : ' % i
        name = raw_input(' ' + s)
        T.append(Customer(name))
    print "starting the simulation"
    for t in T: t.start()
    print "simulation has started"

if __name__ == "__main__": main()
```

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one

Waiter

multiple threads
require
synchronization

code for restaurant
simulation

The Function main()

in the script `fastfood.py`

```
def main():
    "defines and starts threads"
    n = input('give #customers : ')
    T = []
    for i in range(0,n):
        s = 'name of customer %d : ' % i
        name = raw_input(' ' + s)
        T.append(Customer(name))
    print "starting the simulation"
    for t in T: t.start()
    print "simulation has started"

if __name__ == "__main__": main()
```

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

The Function main()

in the script `fastfood.py`

```
def main():
    "defines and starts threads"
    n = input('give #customers : ')
    T = []
    for i in range(0,n):
        s = 'name of customer %d : ' % i
        name = raw_input(' ' + s)
        T.append(Customer(name))
    print "starting the simulation"
    for t in T: t.start()
    print "simulation has started"

if __name__ == "__main__": main()
```

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

The Constructor in Customer

code for `__init__` in class `Customer`

```
import random

def __init__(self,t):
    """
    Initializes thread with name t,
    generates waiting and eating time.
    """
    Thread.__init__(self,name=t)
    self.wait = random.randint(1,6)
    self.eat = random.randint(1,6)
```

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

The Constructor in Customer

code for `__init__` in class `Customer`

```
import random

def __init__(self,t):
    """
    Initializes thread with name t,
    generates waiting and eating time.
    """
    Thread.__init__(self,name=t)
    self.wait = random.randint(1,6)
    self.eat = random.randint(1,6)
```

Overriding the `run` in `Customer(Thread)`

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require

synchronization
code for restaurant simulation

```
import time

def run(self):
    """
    Thread waits for order and eats.
    writes three messages.
    """
    t = self.getName()
    print t + ' is waiting for %d time units' \
          % self.wait
    time.sleep(self.wait)
    print t + ' waited %d time units' \
          % self.wait
    time.sleep(self.eat)
    print t + ' ate for %d time units' \
          % self.eat
```

Overriding the `run` in `Customer(Thread)`

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

```
import time

def run(self):
    """
    Thread waits for order and eats.
    writes three messages.
    """
    t = self.getName()
    print t + ' is waiting for %d time units' \
          % self.wait
    time.sleep(self.wait)
    print t + ' waited %d time units' \
          % self.wait
    time.sleep(self.eat)
    print t + ' ate for %d time units' \
          % self.eat
```

Overriding the `run` in `Customer(Thread)`

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require

synchronization

code for restaurant simulation

```
import time

def run(self):
    """
    Thread waits for order and eats.
    writes three messages.
    """
    t = self.getName()
    print t + ' is waiting for %d time units' \
          % self.wait
    time.sleep(self.wait)
    print t + ' waited %d time units' \
          % self.wait
    time.sleep(self.eat)
    print t + ' ate for %d time units' \
          % self.eat
```

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

Evaluation: Benefits and Limitations

of the first script `fastfood.py`

Benefits of using threads:

- 1 object oriented design separates stages
→ three stages: born, started, active
- 2 customers seen as independently acting
→ behavior is defined locally

At least two limitations:

- 1 role of server(s) not represented
→ scheduling of order taking ignored
- 2 hard to get statistics
→ all data are gone with thread ends

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require

synchronization
code for restaurant
simulation

Evaluation: Benefits and Limitations

of the first script `fastfood.py`

Benefits of using threads:

- 1 object oriented design separates stages
→ three stages: born, started, active
- 2 customers seen as independently acting
→ behavior is defined locally

At least two limitations:

- 1 role of server(s) not represented
→ scheduling of order taking ignored
- 2 hard to get statistics
→ all data are gone with thread ends

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require

synchronization
code for restaurant
simulation

Evaluation: Benefits and Limitations

of the first script `fastfood.py`

Benefits of using threads:

- 1 object oriented design separates stages
→ three stages: born, started, active
- 2 customers seen as independently acting
→ behavior is defined locally

At least two limitations:

- 1 role of server(s) not represented
→ scheduling of order taking ignored
- 2 hard to get statistics
→ all data are gone with thread ends

Evaluation: Benefits and Limitations

of the first script `fastfood.py`

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require

synchronization
code for restaurant simulation

Benefits of using threads:

- 1 object oriented design separates stages
→ three stages: born, started, active
- 2 customers seen as independently acting
→ behavior is defined locally

At least two limitations:

- 1 role of server(s) not represented
→ scheduling of order taking ignored
- 2 hard to get statistics
→ all data are gone with thread ends

Evaluation: Benefits and Limitations

of the first script `fastfood.py`

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require

synchronization
code for restaurant simulation

Benefits of using threads:

- 1 object oriented design separates stages
→ three stages: born, started, active
- 2 customers seen as independently acting
→ behavior is defined locally

At least two limitations:

- 1 role of server(s) not represented
→ scheduling of order taking ignored
- 2 hard to get statistics
→ all data are gone with thread ends

Locks and Synchronization

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

1 Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

2 Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

3 Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

Two Players taking Turns

as concurrently running threads

Consider two players, taking turns:

first player thinks, makes a move

second player thinks, makes a move

first player thinks, makes a move

second player thinks, makes a move, etc...

Instead of one main program regulating turns,
we want:

- 1 two threads independently running
- 2 checking and changing a shared variable

→ main program plays no role anymore,
the two players are in control of the game

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

Two Players taking Turns

as concurrently running threads

Consider two players, taking turns:

first player thinks, makes a move

second player thinks, makes a move

first player thinks, makes a move

second player thinks, makes a move, etc...

Instead of one main program regulating turns,
we want:

- 1 two threads independently running
- 2 checking and changing a shared variable

→ main program plays no role anymore,
the two players are in control of the game

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

Two Players taking Turns

as concurrently running threads

Consider two players, taking turns:

first player thinks, makes a move

second player thinks, makes a move

first player thinks, makes a move

second player thinks, makes a move, etc...

Instead of one main program regulating turns,
we want:

- 1 two threads independently running
- 2 checking and changing a shared variable

→ main program plays no role anymore,
the two players are in control of the game

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

Two Players taking Turns

as concurrently running threads

Consider two players, taking turns:

first player thinks, makes a move

second player thinks, makes a move

first player thinks, makes a move

second player thinks, makes a move, etc...

Instead of one main program regulating turns,
we want:

- 1 two threads independently running
- 2 checking and changing a shared variable

→ main program plays no role anymore,
the two players are in control of the game

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

Two Players taking Turns

as concurrently running threads

Consider two players, taking turns:
first player thinks, makes a move
second player thinks, makes a move
first player thinks, makes a move
second player thinks, makes a move, etc...

Instead of one main program regulating turns,
we want:

- 1 two threads independently running
- 2 checking and changing a shared variable

→ main program plays no role anymore,
the two players are in control of the game

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

Two Players taking Turns

as concurrently running threads

Consider two players, taking turns:

first player thinks, makes a move

second player thinks, makes a move

first player thinks, makes a move

second player thinks, makes a move, etc...

Instead of one main program regulating turns,
we want:

- 1 two threads independently running
- 2 checking and changing a shared variable

→ main program plays no role anymore,
the two players are in control of the game

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

Two Players taking Turns

as concurrently running threads

Consider two players, taking turns:

first player thinks, makes a move

second player thinks, makes a move

first player thinks, makes a move

second player thinks, makes a move, etc...

Instead of one main program regulating turns,
we want:

- 1 two threads independently running
- 2 checking and changing a shared variable

→ main program plays no role anymore,
the two players are in control of the game

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

Two Players taking Turns

as concurrently running threads

Consider two players, taking turns:

first player thinks, makes a move

second player thinks, makes a move

first player thinks, makes a move

second player thinks, makes a move, etc...

Instead of one main program regulating turns,
we want:

- 1 two threads independently running
- 2 checking and changing a shared variable

→ main program plays no role anymore,
the two players are in control of the game

Running script `taketurns.py`

Threads for Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players taking Turns

implemented as two
concurrently running
threads

code for the script
`taketurns.py`

Restaurant with one

Waiter

multiple threads
require

synchronization
code for restaurant
simulation

```
$ python taketurns.py
player 0 born
player 1 born
starting the game
game has started
0 checks value 3
1 checks value 3
1 thinks 7 time units
0 checks value 3
1 changes value to 2
0 checks value 2
0 thinks 2 time units
0 changes value to 1
1 checks value 1
1 thinks 8 time units
0 checks value 1
1 changes value to 0
0 checks value 0
1 checks value 0
```

Running script `taketurns.py`

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require

synchronization

code for restaurant simulation

```
$ python taketurns.py
player 0 born
player 1 born
starting the game
game has started
0 checks value 3
1 checks value 3
1 thinks 7 time units
0 checks value 3
1 changes value to 2
0 checks value 2
0 thinks 2 time units
0 changes value to 1
1 checks value 1
1 thinks 8 time units
0 checks value 1
1 changes value to 0
0 checks value 0
1 checks value 0
```

Running script `taketurns.py`

Threads for Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players taking Turns

implemented as two
concurrently running
threads

code for the script
`taketurns.py`

Restaurant with one

Waiter

multiple threads
require
synchronization

code for restaurant
simulation

```
$ python taketurns.py
player 0 born
player 1 born
starting the game
game has started
0 checks value 3
1 checks value 3
1 thinks 7 time units
0 checks value 3
1 changes value to 2
0 checks value 2
0 thinks 2 time units
0 changes value to 1
1 checks value 1
1 thinks 8 time units
0 checks value 1
1 changes value to 0
0 checks value 0
1 checks value 0
```

Running script `taketurns.py`

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

```
$ python taketurns.py
player 0 born
player 1 born
starting the game
game has started
0 checks value 3
1 checks value 3
1 thinks 7 time units
0 checks value 3
1 changes value to 2
0 checks value 2
0 thinks 2 time units
0 changes value to 1
1 checks value 1
1 thinks 8 time units
0 checks value 1
1 changes value to 0
0 checks value 0
1 checks value 0
```

Running script `taketurns.py`

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

```
$ python taketurns.py
player 0 born
player 1 born
starting the game
game has started
0 checks value 3
1 checks value 3
1 thinks 7 time units
0 checks value 3
1 changes value to 2
0 checks value 2
0 thinks 2 time units
0 changes value to 1
1 checks value 1
1 thinks 8 time units
0 checks value 1
1 changes value to 0
0 checks value 0
1 checks value 0
```

Running script `taketurns.py`

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

```
$ python taketurns.py
player 0 born
player 1 born
starting the game
game has started
0 checks value 3
1 checks value 3
1 thinks 7 time units
0 checks value 3
1 changes value to 2
0 checks value 2
0 thinks 2 time units
0 changes value to 1
1 checks value 1
1 thinks 8 time units
0 checks value 1
1 changes value to 0
0 checks value 0
1 checks value 0
```

Running script `taketurns.py`

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require

synchronization

code for restaurant simulation

```
$ python taketurns.py
player 0 born
player 1 born
starting the game
game has started
0 checks value 3
1 checks value 3
1 thinks 7 time units
0 checks value 3
1 changes value to 2
0 checks value 2
0 thinks 2 time units
0 changes value to 1
1 checks value 1
1 thinks 8 time units
0 checks value 1
1 changes value to 0
0 checks value 0
1 checks value 0
```

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

Algorithm for Threads in taketurns.py

After making move, the players enter
a *busy-waiting loop*:

```
while True:
    time.sleep(5)
    if value % 2 == 'player name': break
```

The sleep time of 5 units is necessary to prevent one thread from absorbing all CPU cycles.

The *shared* `value` determines the total number of moves the two players make.

After each move, `value` is decreased by one.

Threads for
Simulationsimulating arrival of
customersthe code for the
simulation with
threadsTwo Players
taking Turnsimplemented as two
concurrently running
threadscode for the script
taketurns.pyRestaurant
with one
Waitermultiple threads
require
synchronizationcode for restaurant
simulation

Algorithm for Threads in taketurns.py

After making move, the players enter
a *busy-waiting loop*:

```
while True:
    time.sleep(5)
    if value % 2 == 'player name': break
```

The sleep time of 5 units is necessary to prevent one thread
from absorbing all CPU cycles.

The *shared* `value` determines the total number of moves
the two players make.

After each move, `value` is decreased by one.

Algorithm for Threads in taketurns.py

Threads for Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant with one Waiter

multiple threads
require
synchronization

code for restaurant
simulation

After making move, the players enter
a *busy-waiting loop*:

```
while True:
    time.sleep(5)
    if value % 2 == 'player name': break
```

The sleep time of 5 units is necessary to prevent one thread from absorbing all CPU cycles.

The *shared* `value` determines the total number of moves the two players make.

After each move, `value` is decreased by one.

Locks and Synchronization

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script
`taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

1 Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

2 Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

3 Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Code for the Function `main()`

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

```
def main():
    """
    Defines two players '0' and '1'
    and starts the game.
    """
    shared = []
    p1 = Player('0',shared)
    p2 = Player('1',shared)
    print 'starting the game'
    shared.append(3)
    p1.start()
    p2.start()
    print 'game has started'
```

Shared Variable

how threads communicate with each other

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

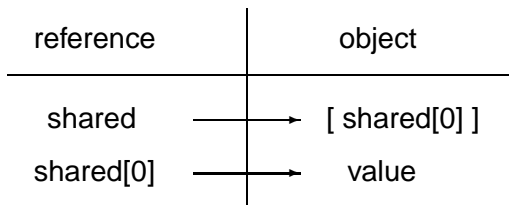
code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Relations between references and objects:



With `shared = []`, we pass only the reference when we create `p1 = Player('0', shared)`.

The value in `shared` (i.e.: `shared[0]`) is set later. We pass the reference, not the values to the data object attributes of the threads.

Shared Variable

how threads communicate with each other

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

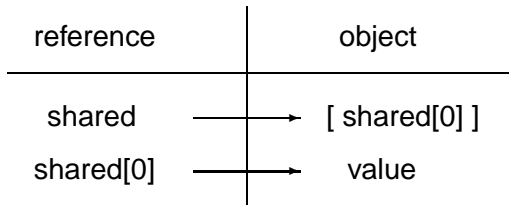
code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Relations between references and objects:



With `shared = []`, we pass only the reference when we create `p1 = Player('0', shared)`.

The value in `shared` (i.e.: `shared[0]`) is set later. We pass the reference, not the values to the data object attributes of the threads.

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
takeTurns.py

Restaurant
with one

Waiter

multiple threads
require
synchronization

code for restaurant
simulation

The Class Player

```

from threading import *
import time
import random

class Player(Thread):
    """
    Player waits turn and makes move.
    """
    def __init__(self,t,v):
        """
        Initializes customer with name t,
        generates waiting and eating time.
        """
        Thread.__init__(self,name=t)
        print 'player ' + t + ' born'
        self.sv = v

```

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one

Waiter

multiple threads
require
synchronization

code for restaurant
simulation

The Class Player

```

from threading import *
import time
import random

class Player(Thread):
    """
    Player waits turn and makes move.
    """
    def __init__(self,t,v):
        """
        Initializes customer with name t,
        generates waiting and eating time.
        """
        Thread.__init__(self,name=t)
        print 'player ' + t + ' born'
        self.sv = v
  
```

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one

Waiter

multiple threads
require
synchronization

code for restaurant
simulation

The Class Player

```

from threading import *
import time
import random

class Player(Thread):
    """
    Player waits turn and makes move.
    """
    def __init__(self,t,v):
        """
        Initializes customer with name t,
        generates waiting and eating time.
        """
        Thread.__init__(self,name=t)
        print 'player ' + t + ' born'
        self.sv = v

```

Defining run of Player

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require

synchronization
code for restaurant simulation

```
def run(self):
    """
    Player checks value every 5 time units.
    If parity matches, value is decreased.
    The game is over if value == 0.
    """
    p = self.getName()
    n = int(p)
    while True:
        while True:
            time.sleep(5)
            v = self.sv[0]
            print p + ' checks value %d ' % v
            if v <= 0 or v % 2 == n: break
```

→ inner loop is busy waiting

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

```

if v <= 0: break # game over
r = random.randint(1,10)
s = p + ' thinks %d' % r
print s + ' time units'
time.sleep(r)

v = self.sv.pop(0) - 1
print p + ' changes value to %d ' % v
self.sv.append(v)

```

Changing the shared variable in

```

v = self.sv.pop(0) - 1
self.sv.append(v)

```

is a *critical section* of the program.

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

```

if v <= 0: break # game over
r = random.randint(1,10)
s = p + ' thinks %d' % r
print s + ' time units'
time.sleep(r)
v = self.sv.pop(0) - 1
print p + ' changes value to %d ' % v
self.sv.append(v)

```

Changing the shared variable in

```

v = self.sv.pop(0) - 1
self.sv.append(v)

```

is a *critical section* of the program.

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

```

if v <= 0: break # game over
r = random.randint(1,10)
s = p + ' thinks %d' % r
print s + ' time units'
time.sleep(r)
v = self.sv.pop(0) - 1
print p + ' changes value to %d ' % v
self.sv.append(v)

```

Changing the shared variable in

```

v = self.sv.pop(0) - 1
self.sv.append(v)

```

is a *critical section* of the program.

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
takeTurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

Thread Safety

Code is ***thread safe*** if its simultaneous execution by multiple threads is correct.

Only one thread can change shared data.

Once only concern for operating system programmer ...

Some illustrations of thread safety concerns:

- 1 shared bank account
- 2 do not block intersection
- 3 access same mailbox from multiple computers
- 4 threads access same address space in memory

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

Thread Safety

Code is ***thread safe*** if its simultaneous execution by multiple threads is correct.

Only one thread can change shared data.

Once only concern for operating system programmer ...

Some illustrations of thread safety concerns:

- 1 shared bank account
- 2 do not block intersection
- 3 access same mailbox from multiple computers
- 4 threads access same address space in memory

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
takeTurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

Thread Safety

Code is ***thread safe*** if its simultaneous execution by multiple threads is correct.

Only one thread can change shared data.

Once only concern for operating system programmer ...

Some illustrations of thread safety concerns:

- 1 shared bank account
- 2 do not block intersection
- 3 access same mailbox from multiple computers
- 4 threads access same address space in memory

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

Thread Safety

Code is ***thread safe*** if its simultaneous execution by multiple threads is correct.

Only one thread can change shared data.

Once only concern for operating system programmer ...

Some illustrations of thread safety concerns:

- 1 shared bank account
- 2 do not block intersection
- 3 access same mailbox from multiple computers
- 4 threads access same address space in memory

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

Thread Safety

Code is ***thread safe*** if its simultaneous execution by multiple threads is correct.

Only one thread can change shared data.

Once only concern for operating system programmer ...

Some illustrations of thread safety concerns:

- 1 shared bank account
- 2 do not block intersection
- 3 access same mailbox from multiple computers
- 4 threads access same address space in memory

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

Thread Safety

Code is ***thread safe*** if its simultaneous execution by multiple threads is correct.

Only one thread can change shared data.

Once only concern for operating system programmer ...

Some illustrations of thread safety concerns:

- 1 shared bank account
- 2 do not block intersection
- 3 access same mailbox from multiple computers
- 4 threads access same address space in memory

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

Thread Safety

Code is ***thread safe*** if its simultaneous execution by multiple threads is correct.

Only one thread can change shared data.

Once only concern for operating system programmer ...

Some illustrations of thread safety concerns:

- 1 shared bank account
- 2 do not block intersection
- 3 access same mailbox from multiple computers
- 4 threads access same address space in memory

Using Locks in Critical Section

making programs thread safe

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Recall ***critical section*** in `taketurns.py`:

```
v = self.sv.pop(0) - 1
self.sv.append(v)
```

To ensure that ***only one*** thread executes code in critical section, we use a ***lock***:

```
import thread
lock = thread.allocate_lock()

lock.acquire()
# code in critical section
lock.release()
```

Using Locks in Critical Section

making programs thread safe

Threads for Simulation

simulating arrival of customers
the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads
code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization
code for restaurant simulation

Recall ***critical section*** in `taketurns.py`:

```
v = self.sv.pop(0) - 1
self.sv.append(v)
```

To ensure that ***only one*** thread executes code in critical section, we use a ***lock***:

```
import thread
lock = thread.allocate_lock()

lock.acquire()
# code in critical section
lock.release()
```

Using Locks in Critical Section

making programs thread safe

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Recall ***critical section*** in `taketurns.py`:

```
v = self.sv.pop(0) - 1
self.sv.append(v)
```

To ensure that ***only one*** thread executes code in critical section, we use a ***lock***:

```
import thread
lock = thread.allocate_lock()

lock.acquire()
# code in critical section
lock.release()
```

Using Locks in Critical Section

making programs thread safe

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Recall ***critical section*** in `taketurns.py`:

```
v = self.sv.pop(0) - 1
self.sv.append(v)
```

To ensure that ***only one*** thread executes code in critical section, we use a ***lock***:

```
import thread
lock = thread.allocate_lock()

lock.acquire()
# code in critical section
lock.release()
```

Using Locks in Critical Section

making programs thread safe

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Recall *critical section* in `taketurns.py`:

```
v = self.sv.pop(0) - 1
self.sv.append(v)
```

To ensure that *only one* thread executes code in critical section, we use a *lock*:

```
import thread
lock = thread.allocate_lock()

lock.acquire()
# code in critical section
lock.release()
```

Locks and Synchronization

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

1 Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

2 Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

3 Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

31 Mar 2010

Restaurant with one Waiter

various stages and states

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Customers pass through various stages:

- | | | | |
|---|---|-------------------|---|
| → | 0 | enters restaurant | |
| | 1 | waits for table | ← |
| → | 2 | orders food | |
| | 3 | waits for food | ← |
| → | 4 | eats the food | |
| | 5 | waits for bill | ← |
| → | 6 | pays and leaves | |

Game of taking turns:

- waiter waits for action of customer
- ← customer waits for action of waiter

31 Mar 2010

Restaurant with one Waiter

various stages and states

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Customers pass through various stages:

- | | | | |
|---|---|-------------------|---|
| → | 0 | enters restaurant | |
| | 1 | waits for table | ← |
| → | 2 | orders food | |
| | 3 | waits for food | ← |
| → | 4 | eats the food | |
| | 5 | waits for bill | ← |
| → | 6 | pays and leaves | |

Game of taking turns:

- waiter waits for action of customer
- ← customer waits for action of waiter

Restaurant with one Waiter

various stages and states

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Customers pass through various stages:

- | | | | |
|---|---|-------------------|---|
| → | 0 | enters restaurant | |
| | 1 | waits for table | ← |
| → | 2 | orders food | |
| | 3 | waits for food | ← |
| → | 4 | eats the food | |
| | 5 | waits for bill | ← |
| → | 6 | pays and leaves | |

Game of taking turns:

- waiter waits for action of customer
- ← customer waits for action of waiter

Restaurant with one Waiter

various stages and states

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Customers pass through various stages:

- | | | | |
|---|---|-------------------|---|
| → | 0 | enters restaurant | |
| | 1 | waits for table | ← |
| → | 2 | orders food | |
| | 3 | waits for food | ← |
| → | 4 | eats the food | |
| | 5 | waits for bill | ← |
| → | 6 | pays and leaves | |

Game of taking turns:

- waiter waits for action of customer
- ← customer waits for action of waiter

Restaurant with one Waiter

various stages and states

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Customers pass through various stages:

- | | | | |
|---|---|-------------------|---|
| → | 0 | enters restaurant | |
| | 1 | waits for table | ← |
| → | 2 | orders food | |
| | 3 | waits for food | ← |
| → | 4 | eats the food | |
| | 5 | waits for bill | ← |
| → | 6 | pays and leaves | |

Game of taking turns:

- waiter waits for action of customer
- ← customer waits for action of waiter

31 Mar 2010

Restaurant with one Waiter

various stages and states

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Customers pass through various stages:

- | | | | |
|---|---|-------------------|---|
| → | 0 | enters restaurant | |
| | 1 | waits for table | ← |
| → | 2 | orders food | |
| | 3 | waits for food | ← |
| → | 4 | eats the food | |
| | 5 | waits for bill | ← |
| → | 6 | pays and leaves | |

Game of taking turns:

- waiter waits for action of customer
- ← customer waits for action of waiter

Restaurant with one Waiter

various stages and states

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Customers pass through various stages:

- | | | | |
|---|---|-------------------|---|
| → | 0 | enters restaurant | |
| | 1 | waits for table | ← |
| → | 2 | orders food | |
| | 3 | waits for food | ← |
| → | 4 | eats the food | |
| | 5 | waits for bill | ← |
| → | 6 | pays and leaves | |

Game of taking turns:

- waiter waits for action of customer
- ← customer waits for action of waiter

Restaurant with one Waiter

various stages and states

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Customers pass through various stages:

- | | | | |
|---|---|-------------------|---|
| → | 0 | enters restaurant | |
| | 1 | waits for table | ← |
| → | 2 | orders food | |
| | 3 | waits for food | ← |
| → | 4 | eats the food | |
| | 5 | waits for bill | ← |
| → | 6 | pays and leaves | |

Game of taking turns:

- waiter waits for action of customer
- ← customer waits for action of waiter

31 Mar 2010

Restaurant with one Waiter

various stages and states

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Customers pass through various stages:

- | | | | |
|---|---|-------------------|---|
| → | 0 | enters restaurant | |
| | 1 | waits for table | ← |
| → | 2 | orders food | |
| | 3 | waits for food | ← |
| → | 4 | eats the food | |
| | 5 | waits for bill | ← |
| → | 6 | pays and leaves | |

Game of taking turns:

- waiter waits for action of customer
- ← customer waits for action of waiter

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Simulation Algorithm

object oriented multithreading

Object data attribute for customer is its state.

The waiter knows the state of every customer.

Moving from one state to the next:

- 1 even states: customer does it
- 2 odd states: action of waiter needed

One class `Customer` and one class `Waiter`.

→ all threads run autonomously

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Simulation Algorithm

object oriented multithreading

Object data attribute for customer is its state.

The waiter knows the state of every customer.

Moving from one state to the next:

- 1 even states: customer does it
- 2 odd states: action of waiter needed

One class `Customer` and one class `Waiter`.

→ all threads run autonomously

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Simulation Algorithm

object oriented multithreading

Object data attribute for customer is its state.

The waiter knows the state of every customer.

Moving from one state to the next:

- 1 even states: customer does it
- 2 odd states: action of waiter needed

One class `Customer` and one class `Waiter`.

→ all threads run autonomously

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Simulation Algorithm

object oriented multithreading

Object data attribute for customer is its state.

The waiter knows the state of every customer.

Moving from one state to the next:

- 1 even states: customer does it
- 2 odd states: action of waiter needed

One class `Customer` and one class `Waiter`.

→ all threads run autonomously

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Simulation Algorithm

object oriented multithreading

Object data attribute for customer is its state.

The waiter knows the state of every customer.

Moving from one state to the next:

- 1 even states: customer does it
- 2 odd states: action of waiter needed

One class Customer and one class Waiter.

→ all threads run autonomously

31 Mar 2010

Threads for
Simulationsimulating arrival of
customersthe code for the
simulation with
threadsTwo Players
taking Turnsimplemented as two
concurrently running
threadscode for the script
taketurns.pyRestaurant
with one

Waiter

multiple threads
require
synchronizationcode for restaurant
simulationRunning the Simulation
restaurant.py

```

give #customers : 3
simulation starts...
1 waits for table
2 waits for table
2 orders food
0 waits for table
1 orders food
2 waits for food
0 orders food
1 waits for food
0 waits for food
1 eats food
1 waits for bill
2 eats food
0 eats food
2 waits for bill
1 pays and leaves
0 waits for bill
2 pays and leaves
0 pays and leaves

```

31 Mar 2010

Running the Simulation

restaurant.py

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script taketurns.py

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

```

give #customers : 3
simulation starts...
1 waits for table
2 waits for table
2 orders food
0 waits for table
1 orders food
2 waits for food
0 orders food
1 waits for food
0 waits for food
1 eats food
1 waits for bill
2 eats food
0 eats food
2 waits for bill
1 pays and leaves
0 waits for bill
2 pays and leaves
0 pays and leaves
  
```

31 Mar 2010

Threads for
Simulationsimulating arrival of
customersthe code for the
simulation with
threadsTwo Players
taking Turnsimplemented as two
concurrently running
threadscode for the script
taketurns.pyRestaurant
with one

Waiter

multiple threads
require

synchronization

code for restaurant
simulationRunning the Simulation
restaurant.py

```

give #customers : 3
simulation starts...
1 waits for table...
2 waits for table
2 orders food
0 waits for table
1 orders food
2 waits for food
0 orders food
1 waits for food
0 waits for food
1 eats food
1 waits for bill
2 eats food
0 eats food
2 waits for bill
1 pays and leaves
0 waits for bill
2 pays and leaves
0 pays and leaves

```

31 Mar 2010

Threads for
Simulationsimulating arrival of
customersthe code for the
simulation with
threadsTwo Players
taking Turnsimplemented as two
concurrently running
threadscode for the script
taketurns.pyRestaurant
with one

Waiter

multiple threads
require
synchronizationcode for restaurant
simulationRunning the Simulation
restaurant.py

```

give #customers : 3
simulation starts...
1 waits for table
2 waits for table
2 orders food
0 waits for table
1 orders food
2 waits for food
0 orders food
1 waits for food
0 waits for food
1 eats food
1 waits for bill
2 eats food
0 eats food
2 waits for bill
1 pays and leaves
0 waits for bill
2 pays and leaves
0 pays and leaves

```

31 Mar 2010

Threads for
Simulationsimulating arrival of
customersthe code for the
simulation with
threadsTwo Players
taking Turnsimplemented as two
concurrently running
threadscode for the script
taketurns.pyRestaurant
with one

Waiter

multiple threads
require
synchronizationcode for restaurant
simulationRunning the Simulation
restaurant.py

```

give #customers : 3
simulation starts...
1 waits for table
2 waits for table
2 orders food
0 waits for table
1 orders food
2 waits for food
0 orders food
1 waits for food
0 waits for food
1 eats food
1 waits for bill
2 eats food
0 eats food
2 waits for bill
1 pays and leaves
0 waits for bill
2 pays and leaves
0 pays and leaves

```

31 Mar 2010

Running the Simulation

restaurant.py

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script taketurns.py

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

```

give #customers : 3
simulation starts...
1 waits for table
2 waits for table
2 orders food
0 waits for table
1 orders food
2 waits for food
0 orders food
1 waits for food
0 waits for food
1 eats food
1 waits for bill
2 eats food
0 eats food
2 waits for bill
1 pays and leaves
0 waits for bill
2 pays and leaves
0 pays and leaves
  
```

31 Mar 2010

Running the Simulation

restaurant.py

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script taketurns.py

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

```

give #customers : 3
simulation starts...
1 waits for table...
2 waits for table
2 orders food
0 waits for table
1 orders food
2 waits for food
0 orders food
1 waits for food
0 waits for food
1 eats food
1 waits for bill
2 eats food
0 eats food
2 waits for bill
1 pays and leaves
0 waits for bill
2 pays and leaves
0 pays and leaves

```

Locks and Synchronization

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

1 Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

2 Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

3 Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

Code for Function main()

in script restaurant.py

```
def main():
    """
    Simulation of restaurant with one server.
    """
    Shared = []
    s = Waiter('w', Shared)
    n = input('give #customers : ')
    T = []
    for i in range(0,n):
        T.append(Customer(str(i), Shared))
    for t in T: Shared.append(0)
    s.start()
    for t in T: t.start()
    print "simulation starts..."
```

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

Code for Function main()

in script restaurant.py

```
def main():
    """
    Simulation of restaurant with one server.
    """
    Shared = []
    s = Waiter('w', Shared)
    n = input('give #customers : ')
    T = []
    for i in range(0, n):
        T.append(Customer(str(i), Shared))
    for t in T: Shared.append(0)
    s.start()
    for t in T: t.start()
    print "simulation starts..."
```

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

Code for Function main()

in script restaurant.py

```
def main():
    """
    Simulation of restaurant with one server.
    """
    Shared = []
    s = Waiter('w', Shared)
    n = input('give #customers : ')
    T = []
    for i in range(0,n):
        T.append(Customer(str(i), Shared))
    for t in T: Shared.append(0)
    s.start()
    for t in T: t.start()
    print "simulation starts..."
```

Code for Class Customer

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

```
class Customer(Thread):
    """
    Customer enters, waits for table,
    orders food, waits for food, eats,
    waits for bill and then leaves.
    """
    def __init__(self,t,S):
        """
        Initializes customer with name t,
        and sets initial state to zero.
        """
        Thread.__init__(self,name=t)
        self.state = 0
        self.shs = S
```

Move and Wait

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script taketurns.py

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

```
def move(self):
    """
    For even stages, moves up after delay.
    """
    n = self.getName()
    r = random.randint(1,10)
    time.sleep(r)
    self.state = self.state + 1
    self.shs[int(n)] = self.state

def wait(self):
    """
    Needs waiter for odd stages.
    """
    n = int(self.getName())
    while True:
        time.sleep(5)
        if self.state < self.shs[n]: break
    self.state = self.state + 1
```

Move and Wait

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script taketurns.py

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

```
def move(self):
    """
    For even stages, moves up after delay.
    """
    n = self.getName()
    r = random.randint(1,10)
    time.sleep(r)
    self.state = self.state + 1
    self.shs[int(n)] = self.state

def wait(self):
    """
    Needs waiter for odd stages.
    """
    n = int(self.getName())
    while True:
        time.sleep(5)
        if self.state < self.shs[n]: break
    self.state = self.state + 1
```

Showing the Stages

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

```
def show(self):
    """
    Returns string representation of state.
    """
    if self.state == 0:
        return ' enters restaurant'
    elif self.state == 1:
        return ' waits for table'
    elif self.state == 2:
        return ' orders food'
    elif self.state == 3:
        return ' waits for food'
    elif self.state == 4:
        return ' eats food'
    elif self.state == 5:
        return ' waits for bill'
    else:
        return ' pays and leaves'
```

The run of the Customer

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

```
def run(self):
    """
    Customer passes through stages.
    """
    n = self.getName()
    while self.state < 6:
        if self.state % 2 == 0:
            self.move()
        else:
            self.wait()
    print n + self.show()
```

Code for Class Waiter

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one

Waiter

multiple threads
require
synchronization

code for restaurant
simulation

```
class Waiter(Thread):
    """
    Waiter checks the state list and
    advances odd states to next level.
    """
    def __init__(self,t,S):
        """
        Initializes waiter with name t,
        and stores shared state list S.
        """
        Thread.__init__(self,name=t)
        self.shs = S
```

The run of the Waiter

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

```
def run(self):
    """
    Waiter advances odd state of customers,
    while otherwise busy waiting.
    """
    while True:
        time.sleep(5)
        done = True
        for i in range(0, len(self.shs)):
            if self.shs[i] % 2 == 1:
                time.sleep(random.randint(1, 3))
                self.shs[i] = self.shs[i] + 1
            if self.shs[i] < 6: done = False
        if done: break
```

The run of the Waiter

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

```
def run(self):
    """
    Waiter advances odd state of customers,
    while otherwise busy waiting.
    """
    while True:
        time.sleep(5)
        done = True
        for i in range(0, len(self.shs)):
            if self.shs[i] % 2 == 1:
                time.sleep(random.randint(1, 3))
                self.shs[i] = self.shs[i] + 1
            if self.shs[i] < 6: done = False
        if done: break
```

The run of the Waiter

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

```
def run(self):
    """
    Waiter advances odd state of customers,
    while otherwise busy waiting.
    """
    while True:
        time.sleep(5)
        done = True
        for i in range(0, len(self.shs)):
            if self.shs[i] % 2 == 1:
                time.sleep(random.randint(1, 3))
                self.shs[i] = self.shs[i] + 1
            if self.shs[i] < 6: done = False
        if done: break
```

Threads for
Simulation

simulating arrival of
customers

the code for the
simulation with
threads

Two Players
taking Turns

implemented as two
concurrently running
threads

code for the script
taketurns.py

Restaurant
with one
Waiter

multiple threads
require
synchronization

code for restaurant
simulation

Discussion of Restaurant Simulation

extensions to basic multithreaded model

Multithreaded programming models customers and server as autonomous agents. What happens is event driven, with the events triggered by individual actors. There is no all controlling, all powerful `main()`.

Many extensions needed:

- 1 customized distributions instead of `random.randint()`
- 2 better data encapsulation of shared state list:
 - 1 separate class to control access
 - 2 customer can only access its own state
 - 3 for thread safety locks are needed
- 3 taking turns is too rigid, allow for interrupts, e.g.: customer asking for refill or more drinks

Threads for
Simulationsimulating arrival of
customersthe code for the
simulation with
threadsTwo Players
taking Turnsimplemented as two
concurrently running
threadscode for the script
taketurns.pyRestaurant
with one
Waitermultiple threads
require
synchronizationcode for restaurant
simulation

Discussion of Restaurant Simulation

extensions to basic multithreaded model

Multithreaded programming models customers and server as autonomous agents. What happens is event driven, with the events triggered by individual actors. There is no all controlling, all powerful `main()`.

Many extensions needed:

- 1 customized distributions instead of `random.randint()`
- 2 better data encapsulation of shared state list:
 - 1 separate class to control access
 - 2 customer can only access its own state
 - 3 for thread safety locks are needed
- 3 taking turns is too rigid, allow for interrupts, e.g.: customer asking for refill or more drinks

Threads for
Simulationsimulating arrival of
customersthe code for the
simulation with
threadsTwo Players
taking Turnsimplemented as two
concurrently running
threadscode for the script
taketurns.pyRestaurant
with one
Waitermultiple threads
require
synchronizationcode for restaurant
simulation

Discussion of Restaurant Simulation

extensions to basic multithreaded model

Multithreaded programming models customers and server as autonomous agents. What happens is event driven, with the events triggered by individual actors. There is no all controlling, all powerful `main()`.

Many extensions needed:

- 1 customized distributions instead of `random.randint()`
- 2 better data encapsulation of shared state list:
 - 1 separate class to control access
 - 2 customer can only access its own state
 - 3 for thread safety locks are needed
- 3 taking turns is too rigid, allow for interrupts, e.g.: customer asking for refill or more drinks

Discussion of Restaurant Simulation

extensions to basic multithreaded model

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Multithreaded programming models customers and server as autonomous agents. What happens is event driven, with the events triggered by individual actors. There is no all controlling, all powerful `main()`.

Many extensions needed:

- 1 customized distributions instead of `random.randint()`
- 2 better data encapsulation of shared state list:
 - 1 separate class to control access
 - 2 customer can only access its own state
 - 3 for thread safety locks are needed
- 3 taking turns is too rigid, allow for interrupts, e.g.: customer asking for refill or more drinks

Discussion of Restaurant Simulation

extensions to basic multithreaded model

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Multithreaded programming models customers and server as autonomous agents. What happens is event driven, with the events triggered by individual actors. There is no all controlling, all powerful `main()`.

Many extensions needed:

- 1 customized distributions instead of `random.randint()`
- 2 better data encapsulation of shared state list:
 - 1 separate class to control access
 - 2 customer can only access its own state
 - 3 for thread safety locks are needed
- 3 taking turns is too rigid, allow for interrupts, e.g.: customer asking for refill or more drinks

Discussion of Restaurant Simulation

extensions to basic multithreaded model

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Multithreaded programming models customers and server as autonomous agents. What happens is event driven, with the events triggered by individual actors. There is no all controlling, all powerful `main()`.

Many extensions needed:

- 1 customized distributions instead of `random.randint()`
- 2 better data encapsulation of shared state list:
 - 1 separate class to control access
 - 2 customer can only access its own state
 - 3 for thread safety locks are needed
- 3 taking turns is too rigid, allow for interrupts, e.g.: customer asking for refill or more drinks

Discussion of Restaurant Simulation

extensions to basic multithreaded model

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for restaurant simulation

Multithreaded programming models customers and server as autonomous agents. What happens is event driven, with the events triggered by individual actors. There is no all controlling, all powerful `main()`.

Many extensions needed:

- 1 customized distributions instead of `random.randint()`
- 2 better data encapsulation of shared state list:
 - 1 separate class to control access
 - 2 customer can only access its own state
 - 3 for thread safety locks are needed
- 3 taking turns is too rigid, allow for interrupts, e.g.: customer asking for refill or more drinks

Summary + Assignments

Threads for Simulation

simulating arrival of customers

the code for the simulation with threads

Two Players taking Turns

implemented as two concurrently running threads

code for the script `taketurns.py`

Restaurant with one Waiter

multiple threads require synchronization

code for `restaurant simulation`

We covered more in chapter 13 in *Making Use of Python*.

Assignments:

- 1 Extend `taketurns.py`: the shared value is the data object attribute of a class, imported by both players.
- 2 Use a class to represent the shared state list in `restaurant.py`.
- 3 Describe how `restaurant.py` should be modified to model a restaurant with multiple waiters.
- 4 Develop a multithreaded model to simulate how elevators run like in SEO: 12 floors, 4 elevators. Start thinking in the small: one elevator serving 3 floors and design for change.