Threads

Daniel Bosk[1]

Department of Information and Communication Systems (ICS),
Mid Sweden University, Sundsvall.

thread.tex 231 2018-02-05 08:55:44Z jimahl

Overview

1. Threads
   - What is a thread?
   - Benefits
   - Issues
2. Implementation of threads
   - Types of threads
   - The different models
3. Signals
   - What are signals?
4. Playing with Threads in Python
   - With race-condition
   - Without race-condition

Literature

This lecture covers threads. It gives an overview of Chapter 4 “Multithreaded Programming” in [SGG13a]

Overview

1. Threads
   - What is a thread?
   - Benefits
   - Issues
2. Implementation of threads
   - Types of threads
   - The different models
3. Signals
   - What are signals?
4. Playing with Threads in Python
   - With race-condition
   - Without race-condition

Mittuniversitetet
MID SWEDEN UNIVERSITY

What is a thread?

- The process is what the operating system consider the smallest entity of execution.
- This contains the program code, the variables (data), etc.
- A process traditionally has only one *thread* of execution. Such a process is called a heavy-weight process.
- Now, however, we'll extend this with more threads of execution.

What is a thread?

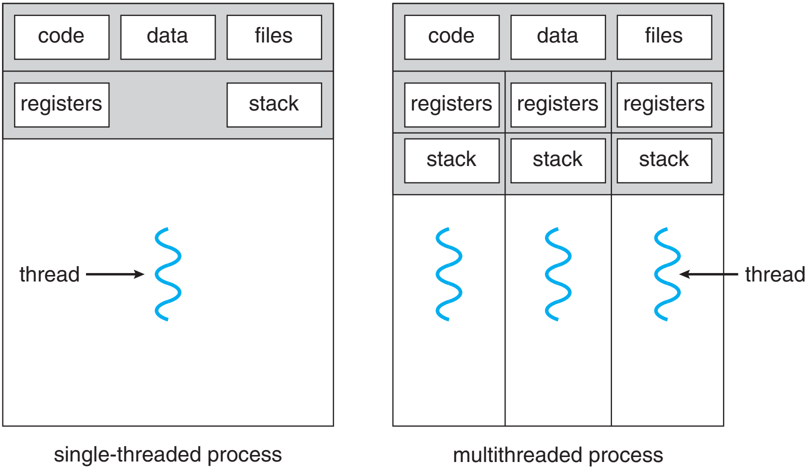

Figure: Single-threaded vs. multithreaded process. Image: [SGG13b].

What is a thread?

- Multiple threads of execution means we can do many things simultaneously in a process.
- Have to be careful though, you never know who is changing something in the process at a given time.

Benefits

- Responsiveness
- Resource sharing
- Economy (context switching, process creation)
- Scalability

Benefits

Scalability

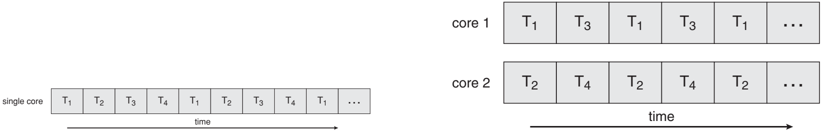

Figure: A single-core and multi-core execution of four threads. Image: [SGG13b].

Issues

- What happens with fork(2) and exec(2)? The calling thread only, or all threads?
- exec(2) usually replaces all threads.
- fork(2) can be different; in some cases only the calling thread is reasonable to fork, in other cases all of them.
- Programming becomes more complex, introducing many potential bugs.

Overview

1 Threads
- What is a thread?
- Benefits
- Issues

2 Implementation of threads
- Types of threads
- The different models

3 Signals
- What are signals?

4 Playing with Threads in Python
- With race-condition
- Without race-condition

Types of threads

User thread These execute in user-mode, and are invisible to the kernel.

Kernel thread These are part of the kernel. They do not necessarily execute all code in kernel-mode though.

Types of threads

- User threads are mapped to kernel threads.
- This mapping can be done in several ways:
  - One-to-one,
  - Many-to-one,
  - Many-to-many.

The different models

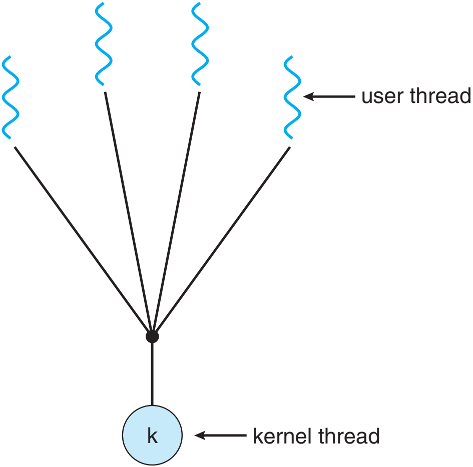

Figure: The many-to-one model. Image: [SGG13b].

The different models

- Can achieve responsiveness, but is not scalable.
- Bad in that if one thread blocks, e.g. through a system call, the whole process blocks, and thus all threads block.
- Can be combined with the one-to-one model though, then threads can be partitioned.

The different models

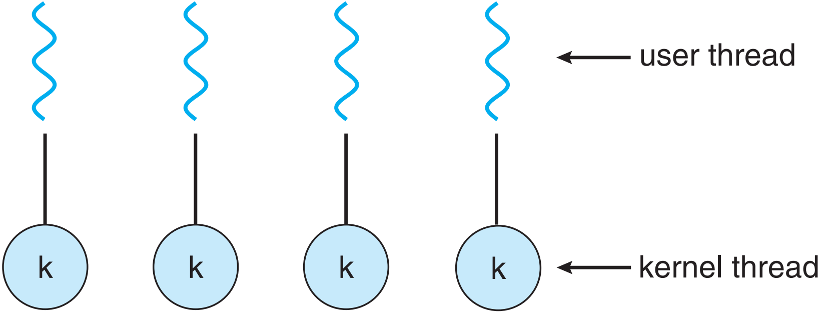

Figure: The one-to-one model. Image: [SGG13b].

The different models

- Good in that it's the OS handling all scheduling.
- Bad in that it requires as many kernel-threads as user-threads.

The different models

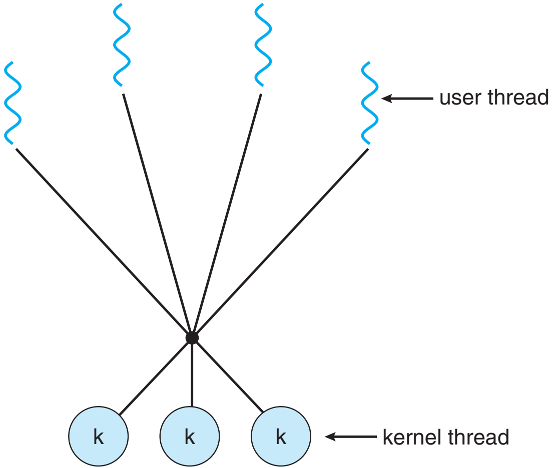

Figure: The many-to-many model. Image: [SGG13b].

The different models

- The many-to-many model requires a light-weight process (LWP).
- To the process this appears as a processor on which it can schedule its threads.
- I.e. the thread library does the scheduling of its threads, as is the case in many-to-one also.
- The kernel schedules all threads in the one-to-one model.

The different models

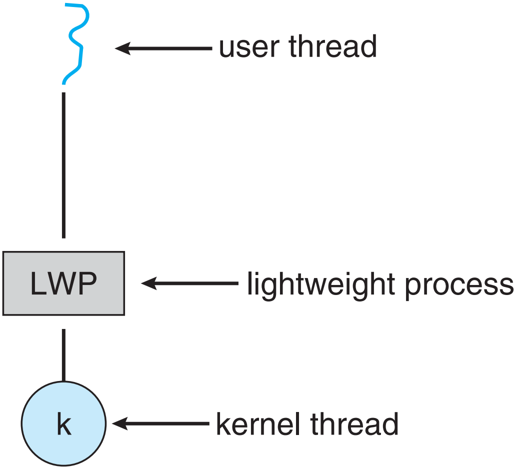

Figure: Structure including an LWP. Image: [SGG13b].

The different models

- The many-to-many model has the advantage that it has none of the problems of the other models.
- It can have an arbitrarily large amount of threads.
- If one thread blocks, i.e. one LWP is blocked, just schedule another LWP.

The different models

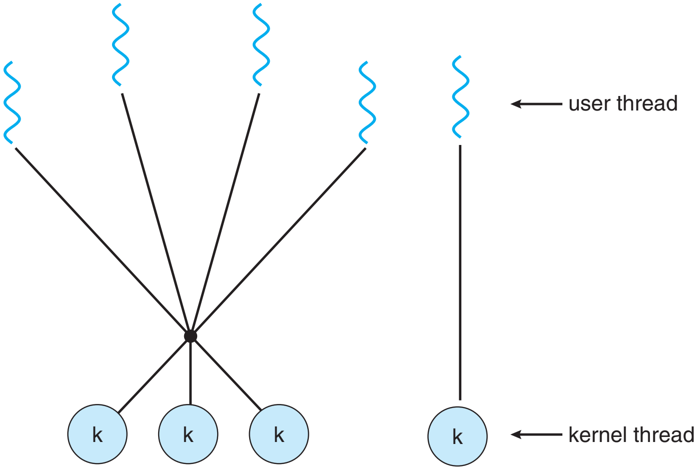

Figure: The two-level model. Image: [SGG13b].

Overview

1 Threads
- What is a thread?
- Benefits
- Issues

2 Implementation of threads
- Types of threads
- The different models

3 Signals
- What are signals?

4 Playing with Threads in Python
- With race-condition
- Without race-condition

What are signals?

- A signal is like an interrupt for a process.
- The OS sends the process a signal.
- The process must stop its current work to handle the signal, then it may return to previous work.
- Examples include “Division by zero”, “Illegal memory access” and hitting Ctrl+C keyboard combination.

What are signals?

1. Process executes division by zero.
2. CPU generates interrupt, calls OS interrupt handler.
3. OS interrupt handler notes the process currently using the CPU.
4. OS generates as signal to the process.
5. Process's signal handler executes.

What are signals?

- With a multithreaded process, which thread should handle a signal from the OS?
- This can usually be specified by the programmer.

Overview

1 Threads
- What is a thread?
- Benefits
- Issues

2 Implementation of threads
- Types of threads
- The different models

3 Signals
- What are signals?

4 Playing with Threads in Python
- With race-condition
- Without race-condition

With race-condition I

```
#!/usr/bin/env python3

# Author:   Daniel Bosk <daniel.bosk@miun.se>
# Date:     15 May 2012

import sys, threading, time

# function to be run in each separate thread
def test(thread_id, delay):
  for i in range(2):
    for j in range(5):
      # critical section
      # print values of i and j
      print( str(thread_id) + ":␣i=" + str(i) +
         ",␣j=" + str(j) )
      # remainder section
      # sleep for delay seconds
```

With race-condition II

```
      time.sleep( float(delay) )

def main():
  # default to using two threads.  this can be overridden by passing a number
  # as argument from the command line.
  # usage: race.py <nthreads>
  nthreads = 2
  if len(sys.argv) > 1:
    nthreads = int( sys.argv[1] )

  threads = []
  for n in range( nthreads ):
    # create a thread which runs the test-function above, documentation:
    # http://docs.python.org/library/threading.html?
```

With race-condition III

```
    t = threading.Thread( target=test,
        args=("thread"+str(n), 1+float(n)/10) )
    t.start()
    threads.append( t )

  print( "waiting␣..." )
  # wait for all threads to finish
  for t in threads:
    t.join()
  print( "done" )

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

Without race-condition I

```
#!/usr/bin/env python3

# Author:  Daniel Bosk <daniel.bosk@miun.se>
# Date:    15 May 2012

import sys, threading, time

# function to be run in each separate thread
def test(lock, thread_id, delay):
  for i in range(2):
    for j in range(5):
      # entry section, wait for lock
      lock.acquire()
      # critical section
      # print values of i and j
      print( str(thread_id) + ":␣i=" + str(i) +
          ",␣j=" + str(j) )
```

Without race-condition II

```
      # exit section, release lock
      lock.release()
      # remainder section
      # sleep for delay seconds
      time.sleep( float(delay) )

def main():
  # prepare a lock for stdout, to synchronise output
  stdout = threading.Lock()

  # default to using two threads.  this can be overridden by passing a number
  # as argument from the command line.
  # usage: ./norace.py <nthreads>
  nthreads = 2
  if len(sys.argv) > 1:
```

Without race-condition III

```
    nthreads = int( sys.argv[1] )

  threads = []
  for n in range( nthreads ):
    # create a thread which runs the test-function above, documentation:
    # http://docs.python.org/library/threading.html?
    t = threading.Thread( target=test,
        args=(stdout, "thread"+str(n), 1+float(n)/10) )
    t.start()
    threads.append( t )

  print( "waiting␣..." )
  # wait for all threads to finish
  for t in threads:
```

Without race-condition IV

```
    t.join()
  print( "done" )

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

Referenser I

Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts*. 9th ed. International Student Version. Hoboken, N.J.: John Wiley & Sons Inc, 2013.

Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts*. 9th ed. Hoboken, N.J.: John Wiley & Sons Inc, 2013.