CSE2016 Programming Methodology

Week 11: Thread Programming

Instructor: Jinyoung Han (jinyounghan@hanyang.ac.kr)

HANYANG UNIVERSITY

Contents

HANYANG UNIVERSITY

Today's Schedule

01. Process and Thread

Process

- Process
  - A process is an instance of program execution
  - E.g., if you open up two browser windows, then you have two processes, even though they are running the same program

- The operating system maintains management information about a process in a process control block (PCB)
- Modern operating systems allow a process to be divided into multiple threads of execution, which share all process management information except for information directly related to execution

01. Process and Thread

HANYANG UNIVERSITY

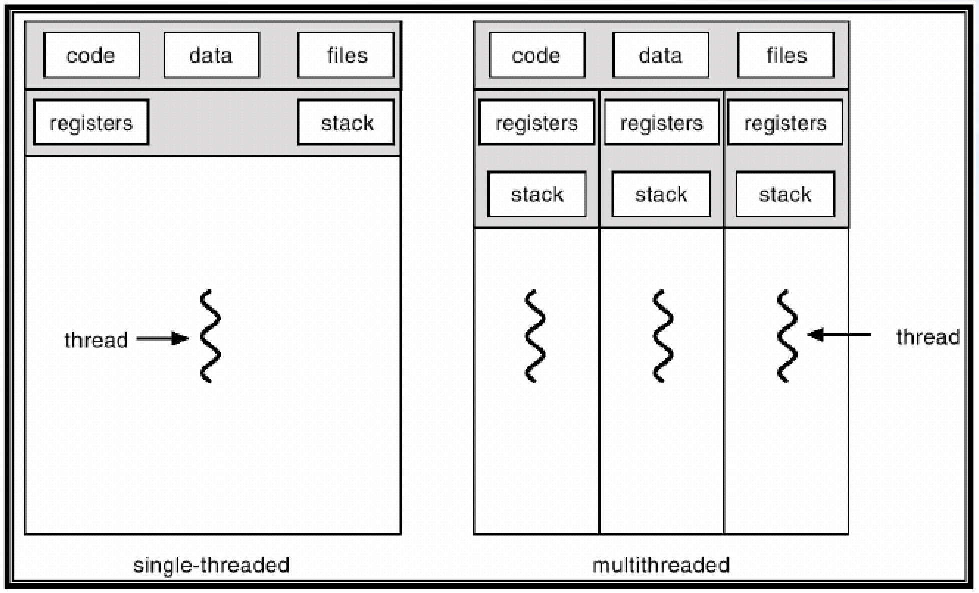
Thread

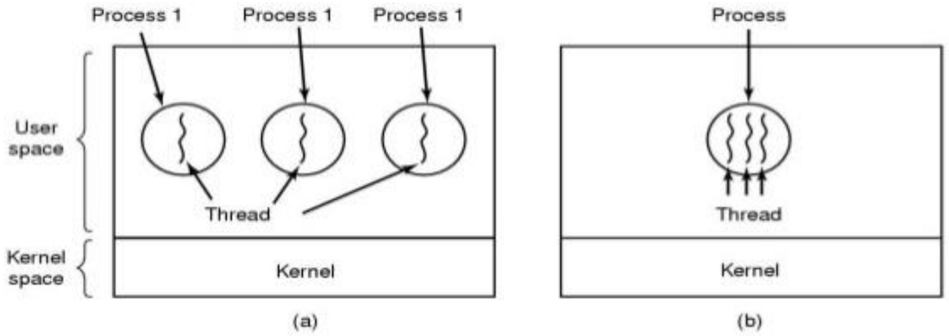
- Most modern operating systems support **threads**: multiple execution streams within a single process
  - Lightweight
  - Threads share process state such as memory, open files, etc.
  - Each thread has a separate stack for procedure calls (in shared memory)
  - Thread is unit of sequential execution

- Why support threads?
  - Concurrent execution on multiprocessors
  - Manage I/O more efficiently: some threads wait for I/O while others compute
  - Most common usage for threads: large server applications

01. Process and Thread

Thread





02. How to Create Threads in Java?

Thread Creation

- (1) Extending java.lang.Thread
  - Overriding the run() method

```
Class Mythread extends Thread
{
            public void run()
            {
                        …
            }
}

Mythread t1 = new Mythread();
t1.start();
```

02. How to Create Threads in Java?

Thread Creation

- (2) Implementing Runnable interface
  - "Thread" actually implements Runnable interface

```
Class Yourthread implements Runnable
{
        public void run()
        {
                ...
        }
}

Yourthread u1 = new Yourthread();
Thread t1 = new Thread(u1);
t1.start();
```

02. How to Create Threads in Java?

Thread Methods

- start(); // starting the thread
- run(); // what to execute
- sleep(); // stop to CPU usage
- wait(); // for synchronization among threads, lock is released and thread is waiting
- notify(); //restarting a thread
- getName(); // returning the thread name
- setPriority(); // giving a priority

03. A Simple Example

Two Threads

- Two threads with different sleep times

```
public class SimpleThread extends Thread
{
    int delay;

    public SimpleThread(String str, int _delay)
    {
        super(str);
        delay = _delay;
        System.out.println(this.getName() + " has started,
        the delay time is " + delay);
    }
```

03. A Simple Example

Two Threads

```
public void run()
{
    for (int i = 0; i < 10; i++)
    {
        System.out.println("[" + i + "] " + this.getName() + " is 
    running..");
    }

    try
    {
        SimpleThread.sleep(delay); // sleep is the static method
    }
    catch(InterruptedException e)
    {
        System.out.println(e.getMessage());
    }
    System.out.println(this.getName() + " is finished.");
}
```

03. A Simple Example

Two Threads

```
public static void main (String[] args)
{
    SimpleThread simple1 = new SimpleThread("thread 1", 50);
    simple1.start();
    SimpleThread simple2 = new SimpleThread("thread 2", 100);
    simple2.start();
}
```

```
thread 1 has started, the delay time is 50
thread 2 has started, the delay time is 100
[0] thread 1 is running..
[1] thread 1 is running..
[2] thread 1 is running..
[3] thread 1 is running..
[4] thread 1 is running..
[5] thread 1 is running..
[0] thread 2 is running..
[6] thread 1 is running..
[1] thread 2 is running..
[7] thread 1 is running..
[2] thread 2 is running..
[8] thread 1 is running..
[3] thread 2 is running..
[9] thread 1 is running..
[4] thread 2 is running..
[5] thread 2 is running..
[6] thread 2 is running..
[7] thread 2 is running..
[8] thread 2 is running..
[9] thread 2 is running..
thread 1 is finished.
thread 2 is finished.
```

04. Thread Scheduling

setPriority()

- We can assign a priority for each thread
  - Using setPriority()
  - MIN_PRIORITY(1) ~ MAX_PRIORITY(10)
  - Default: NORM_PRIORITY(5)

- getPriority()
  - Getting the priority value of a given thread

04. Thread Scheduling

An Example

```
public class SimpleThread2 extends Thread
{
    public static void main (String[] args)
    {
        SimpleThread2 simple1 = new SimpleThread2("thread 1");
        SimpleThread2 simple2 = new SimpleThread2("thread 2");
        SimpleThread2 simple3 = new SimpleThread2("thread 3");
        SimpleThread2 simple4 = new SimpleThread2("thread 4");
        simple1.setPriority(10);

        System.out.println("thread priorities: thread 1 (" +
        simple1.getPriority() + "), thread 2 (" + simple2.getPriority()
        + "), thread 3 (" + simple3.getPriority() + "), thread 4(" +
        simple4.getPriority() + ").");
        simple1.start();
        simple2.start();
        simple3.start();
        simple4.start();
    }
```

04. Thread Scheduling

An Example

```java
public SimpleThread2(String str)
{
    super(str);
    System.out.println(this.getName() + " has started.");
}

public void run()
{
    for (int i = 0; i < 5; i++)
    {
        System.out.println("[" + i + "] " + this.getName() + " is 
        running..");
    }
    System.out.println(this.getName() + " is finished.");
}
```

04. Thread Scheduling

An Example

```
thread 1 has started.
thread 2 has started.
thread 3 has started.
thread 4 has started.
thread priorities: thread 1 (10), thread 2 (5), thread 3 (5), thread 4(5).
[0] thread 1 is running..
[1] thread 1 is running..
[2] thread 1 is running..
[3] thread 1 is running..
[4] thread 1 is running..
thread 1 is finished.
[0] thread 2 is running..
[1] thread 2 is running..
[2] thread 2 is running..
[3] thread 2 is running..
[0] thread 3 is running..
[4] thread 2 is running..
thread 2 is finished.
[1] thread 3 is running..
[2] thread 3 is running..
[3] thread 3 is running..
[4] thread 3 is running..
thread 3 is finished.
[0] thread 4 is running..
[1] thread 4 is running..
[2] thread 4 is running..
[3] thread 4 is running..
[4] thread 4 is running..
thread 4 is finished.
```

What if the starting sequence is changed as follows?

```
simple2.start();
simple3.start();
simple4.start();
simple1.start();
```

Synchronization

- So far, each thread is executed independently
- What if multiple threads share the resource or memory?
  - E.g., two threads try to deposit and withdraw from the same account
- We need to consider "**<u>synchronization</u>**"
  - E.g., semaphore, etc.
  - Java supports the synchronization
    - Synchronized method
    - Synchronized statement

05. Synchronization

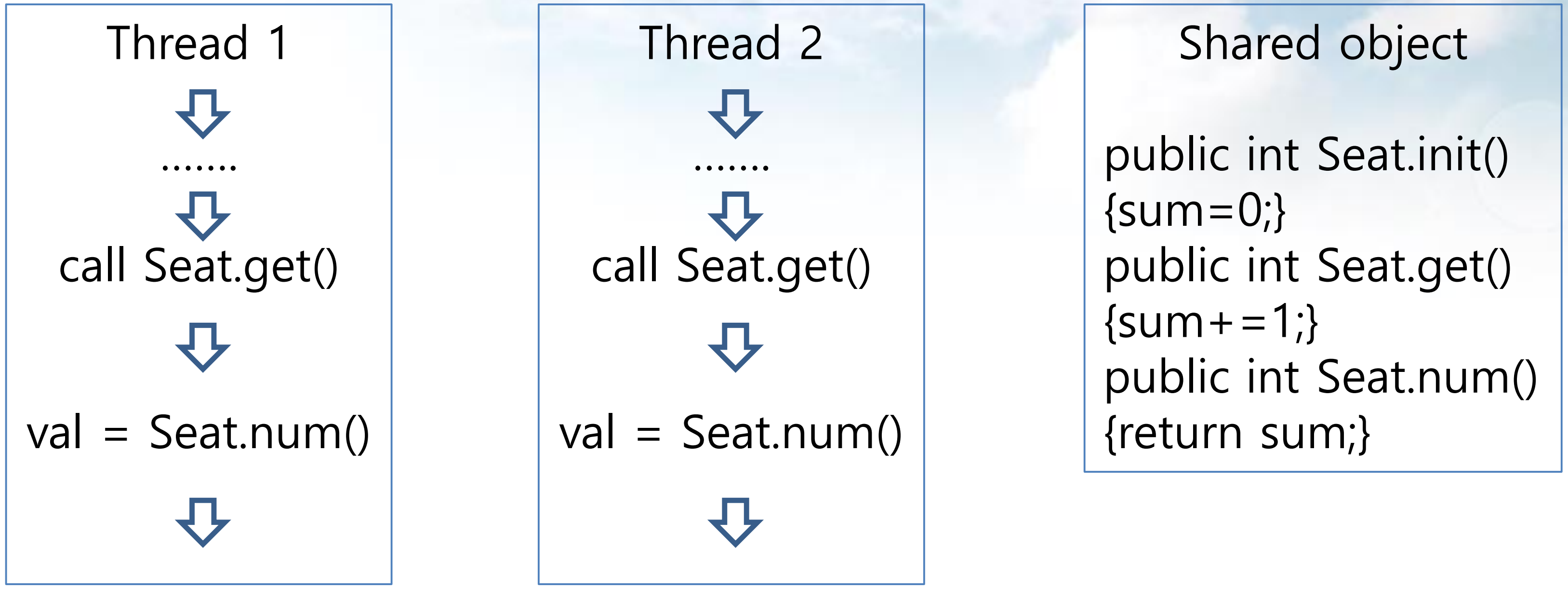
Synchronized Method

- For a method that needs synchronization,
  - Using the "synchronized" keyword
  - Lock operations can be used
    - Only a thread who locks the object can use it

- An example
  - Two threads access the shared object "seat"
  - Each thread tries to get a seat number

05. Synchronization

Seat Example



Thread 1

```
⇩
.......
⇩
call Seat.get()
⇩
val = Seat.num()
⇩
```

Thread 2

```
⇩
.......
⇩
call Seat.get()
⇩
val = Seat.num()
⇩
```

Shared object

```
public int Seat.init()
{sum=0;}
public int Seat.get()
{sum+=1;}
public int Seat.num()
{return sum;}
```

05. Synchronization

Seat Example

```
public class Seat
{
     static int sum;
     public void init(){sum = 0;}
     public void createOne(){sum += 1;}
     public int getNum(){return sum;}

     public void createAssign(String mssg)
     {
          System.out.println(mssg + " is excuted.");
          this.createOne();
          try
          {
                    Thread.sleep(10);
          }
          catch(InterruptedException e)
          {
                    System.out.println(" Interrupted, error = "+ e.getMessage());
          }
          System.out.println(mssg + " : " + this.getNum());
     }
}
```

05. Synchronization

Seat Example

```
public class SomeThread extends Thread
{
    Seat aSeat;
    String mssg;
    public SomeThread(Seat s1, String s)
    {
        aSeat = s1;
        mssg = s;
    }
    public void run()
    {
        aSeat.createAssign(mssg);
    }
}
```

```
public static void main (String[]
args)
{
    Seat seat = new Seat();
    seat.init();

    SomeThread t1 = new
    SomeThread(seat, "thread 1");
    t1.start();
    SomeThread t2 = new
    SomeThread(seat, "thread 2");
    t2.start();
}
```

```
thread 1 is executed.
thread 2 is executed.
thread 2 : 2
thread 1 : 2
```

05. Synchronization

Seat Example

- Synchronized method

```
public synchronized void createAssign(String mssg)
```

```
thread 1 is executed.
thread 1 : 1
thread 2 is executed.
thread 2 : 2
```

05. Synchronization

Seat Example

- What if the shared object is not designed for multi-threads?
- We can consider the “synchronization block”
    - Synchronized (lockObject) {….}

```
synchronized(aSeat)
{
        aSeat.createAssign(mssg);
}
```

```
thread 1 is executed.
thread 1 : 1
thread 2 is executed.
thread 2 : 2
```

06. Deadlock

Deadlock

- Deadlock
  - Two or more threads are waiting for using the shared resource infinitely
  - Deadlock detection is very difficulty
  - Deadlock avoidance is needed

07. Runnable

Runnable

- Java allows extending only one class
  - So, inheriting Thread + other Class is not possible
  - We can implement Runnable interface
  - “Thread” is actually a class that implements Runnable

- A class implementing Runnable connects a Thread class
  - Giving the created object to a Thread constructor

07. Runnable

An Example

```java
public class Run implements Runnable
{
    private String threadName;

    public Run(String str)
    {
        threadName = str;
    }

    public void run()
    {
        System.out.println(threadName + " runnable.");
    }
}
```

07. Runnable

An Example

```
public class RunTest
{
    public static void main (String[] args)
    {
        Run p = new Run("thread 1");
        Run q = new Run("trhead 2");

        Thread t1 = new Thread(p);
        Thread t2 = new Thread(q);
        t1.start();
        t2.start();
    }
}
```

Summary

Summary

- Process and Thread
- How to Create Threads in Java?
- A Simple Example
- Thread Scheduling
- Synchronization
- Deadlock
- Runnable

Thanks

Week 11: Thread Programming
Instructor: Jinyoung Han (jinyounghan@hanyang.ac.kr)