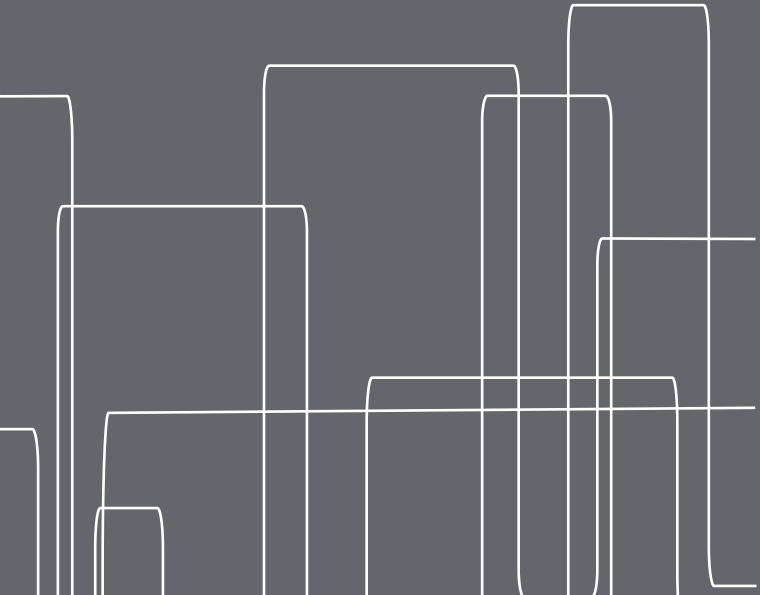
Objektorienterad Programkonstruktion

Föreläsning 11

6 dec 2016



Föreläsningen 13/12

- Halvtidsrepetition
- Maila frågor som ni vill att jag tar upp!
- ccs@kth.se

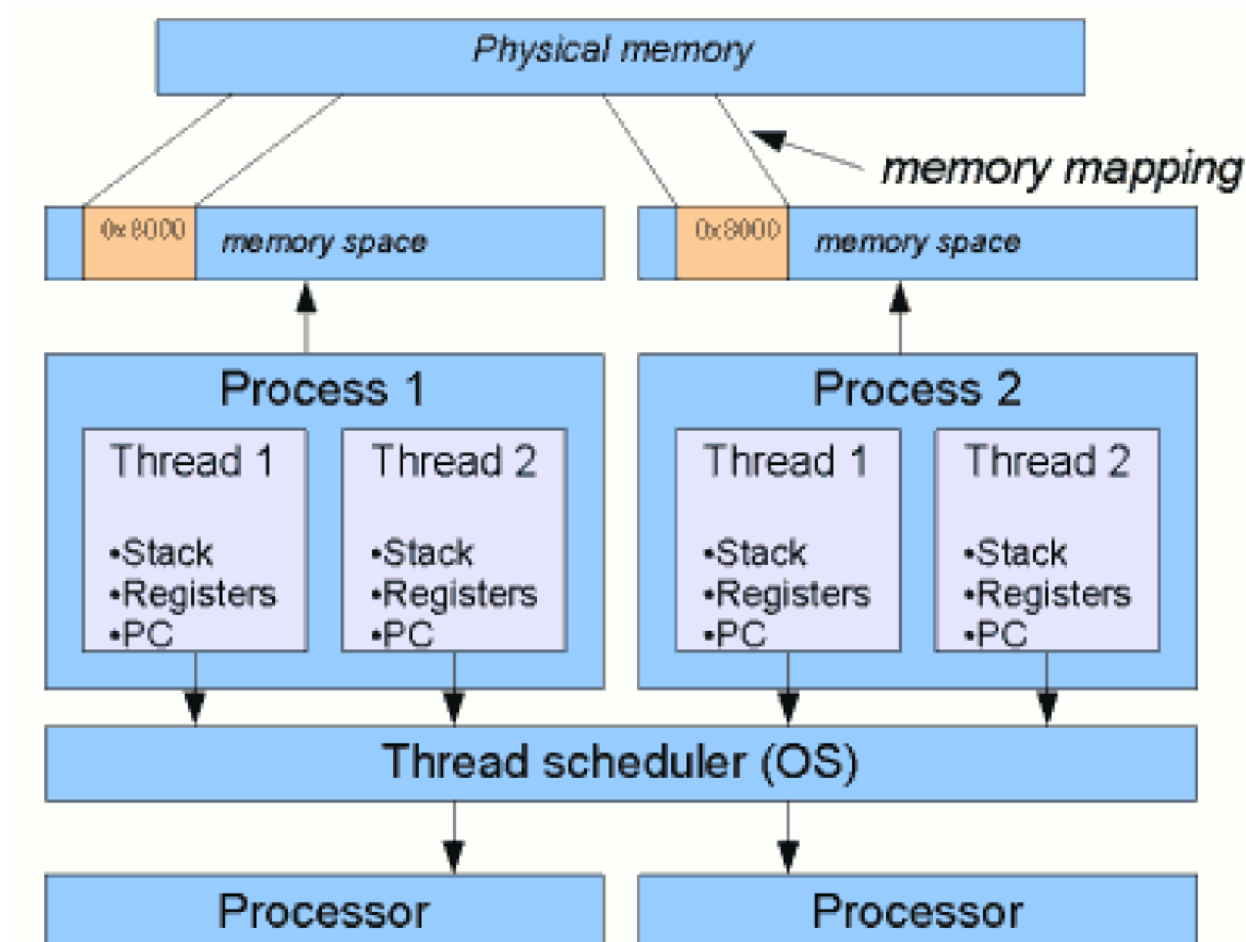
Processer

- Vad vi i dagligt tal menar när vi pratar om ett program som kör
- En process har referenser till en mängd reserverat minne
- En process har referenser till systemresurser, som t.ex filer, sockets, mm
- Ett operativsystem har (minst) en scheduler som ansvarar för att tilldela olika processer tid i processorn för att exekvera sina instruktioner
- Processer har olika prioritet, utifrån vilken turordningen i schedulern bestäms
- Det går att kommunicera mellan processer, men det är ungefär lika bökigt inom en dator som mellan två olika datorer
- I vanliga fall körs javamotorn som en process, men det går att skapa flera ny processer inifrån ett javaprogram

Trådar

- En process är i sin tur uppdelad i trådar (minst en)
- Trådarna kan exekvera parallellt med varandra
- Varje tråd kan schemaläggas separat av schedulern, utifrån en given prioritetsordning. Vid samma prioritet får trådarna turas om, hur ofta de byts ut varierar med schedulern (och OS:et)
- Alla trådar kan ha tillgång till allt minne och filpekare som hör till processen.
- Det delade minnet gör att en context switch ofta går fortare mellan trådar än mellan processer
- Det delade minnet gör att det är lätt att kommunicera mellan olika trådar i samma process
- Java skapas trådar med klassen **Thread** och gränssnittet **Runnable**

Processer och trådar



Thread

- I Java skapas trådar ur klassen `Thread`
- Trådobjektet innehåller kod som körs parallellt med den tråd som startade den
- Koden som ska köras i en tråd finns definierad i dess `run()`-metod
- Trådklassen definierar bl.a
  - `start()`, sätter igång tråden
  - `sleep(int ms)` låter tråden vänta tillfälligt
  - `setPriority(int newPriority)` sätter prioriteten, har nästan ingen effekt!
  - `join(int ms)` väntar på att tråden ska bli klar

Runnable

- Ett alternativ till att ärva från klassen `Thread` är att skapa en egen klass som implementerar `Runnable`
- Då måste ens klass implementera metoden `run()`
- Ett trådobjekt kan skapas med en `Runnable` som argument till konstruktorn, då körs `run()` från Runnableobjektet när `start()` anropas på tråden
- Fördelen med att ärva från Thread är att det ofta blir mindre kod att skriva
- Fördelen med att implementera en egen Runnable-klass är att man kan ärva från en helt annan klass, om man behöver specialiserad funktionalitet
- Thread implementerar Runnable

Thread-exempel

```
public class HelloThread extends Thread {
    public void run() {
        System.out.println("Hello from a thread!");
    }
}

public class ThreadDemo{
    public static void main(String[] args) {
        HelloThread hello = new HelloThread();
        hello.start();
        System.out.println("Hello from main!");
    }
}
```

Runnable

```
public class HelloRunnable implements Runnable {
    public void run() {
        System.out.println("Hello from a thread!");
    }
}

public class RunnableDemo{
    public static void main(String[] args) {
        Thread hello = (new Thread(new HelloRunnable()));
        hello.start();
        System.out.println("Hello from main!");
    }
}
```

Exempel: en multitrådad server

- En server kan med fördel skapa en egen tråd för varje extern förfrågan
- En huvudloop lyssnar efter anslutningar, och startar en ny tråd varje gång det kommer en anslutning utifrån
- De nya trådarna kan ha en konstruktor som tar en `Socket` som argument

Multitrådad server i Java

```
try {
    serverSocket = new ServerSocket(1025);
} catch (IOException e) {
    System.out.println("listen failed on port: 1025");
}

while(!done){
    Socket sock = null;
    try {
        sock = serverSocket.accept();
    } catch (IOException e) {
        System.out.println("Accept failed: 1025");
    }
    Thread thr = new Thread(new Handler(sock));
    thr.start(),
}
```

Threadpool

- **Concurrency pattern** (ej GoF)
- Skapa ett antal trådar en gång för alla
- Uppgifter som skall utföras av en tråd läggs i en uppgiftskö (Task Queue)
- Så fort en tråd blir färdig med sin uppgift tilldelas den nästa uppgift ur kön
- Begränsar antalet trådar som kan skapas
- Minskar overhead för att skapa nya trådar
- Lämplig när det finns en väldigt stor mängd (små) uppgifter som ska utföras
- Ett fåtal trådar kan göra att de första uppgifter som slutförs blir klara fortare än om man har många trådar

Threadpool i Java

- Java definierar gränssnittet `Executor`, som liksom `Thread` kan ta en `Runnable` och köra den
- Det är **inte** definierat när/hur en `Executor` kör en `Runnable`
- I stället för

```
(new Thread(myRunnable)).start();
```

- kan vi anropa

```
myExecutor.execute(myRunnable);
```

- Ett sätt att skapa en ExecutorService som består av en Thread Pool är med fabriksmetoden

```
newFixedThreadPool(int poolSize)
```

- som finns i

```
java.util.concurrent.Executors
```

Threadpool i Java

```
public class NetworkService implements Runnable{
    private ServerSocket serverSocket;
    private ExecutorService pool;

    public NetworkService(int port, int poolSize) throws IOException {
        serverSocket = new ServerSocket(port);
        pool = Executors.newFixedThreadPool(poolSize);
    }

    public void run() {
        try {
            while (true) {
                pool.execute(new Handler(serverSocket.accept()));
            }
        } catch (IOException ex) {
            pool.shutdown();
        }
    }
}
```

Handler

```
class Handler implements Runnable {
    private final Socket socket;

    Handler(Socket socket) {
        this.socket = socket;
    }

    public void run() {
        // read and service request on socket
    }
}
```