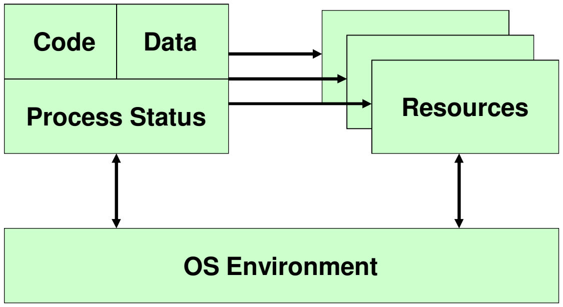
Op. Sys. Processes & Threads Lecture

CSCI 4011

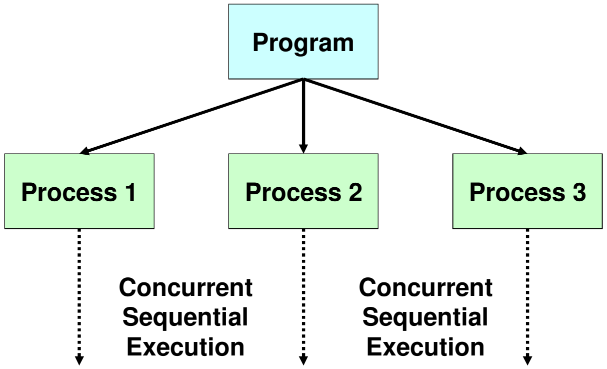
Operating Systems

The View of a Process

Processes

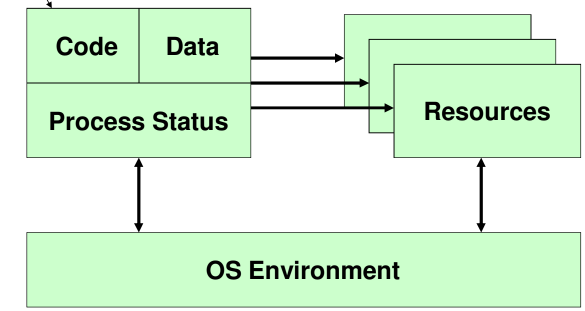


Processes

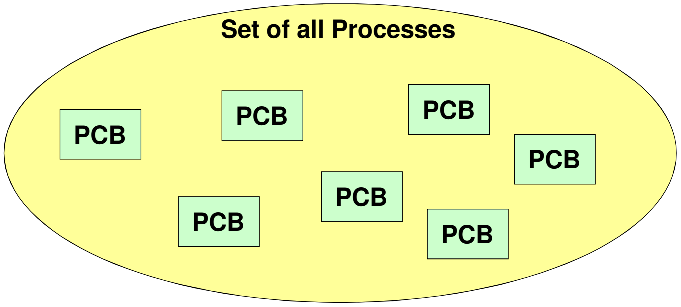

Program
Process 1
Process 2
Process 3
Concurrent Sequential Execution
Concurrent Sequential Execution

PCB

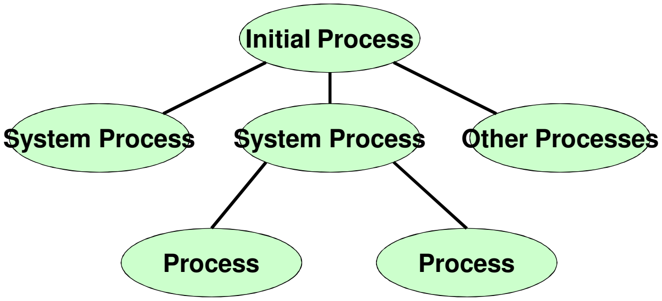
Process Control Block



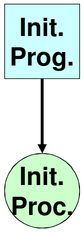
Process Control Block



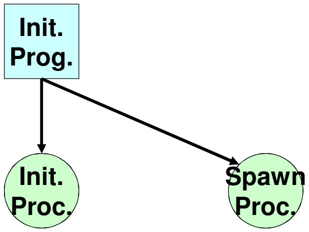
Creating Processes



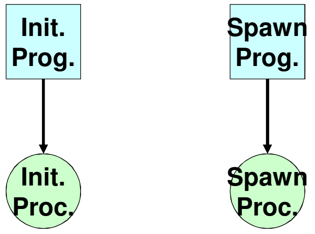
Creating Processes



Creating Processes



Creating Processes



C/C++ Process Programming

- **`#include <stdio.h>`**
- **`#include <sys/wait.h>`**
- **`execl("program", NULL)`**
- **`fork()`**
- **`wait()`**
- **`sleep()`**

C/C++ Process Programming

- **`#include <stdio.h>`**
  - ".h" extension represents a header file
  - # is called as "preprocessor“ command to include the header file when compiled
  - "stdio" is a header file that contains standard functions related to input and output, such as "printf", "scanf”, etc.

C/C++ Process Programming

- **`#include <sys/wait.h>`**
  - "stdio" is a header file that contains standard functions that wait for a process to change state.

C/C++ Process Programming

- **`execl("program", NULL)`**
  - A system call that launches a new process, replacing the current one.
  - The program executed by this new process is defined by the "program" argument.

C/C++ Process Programming

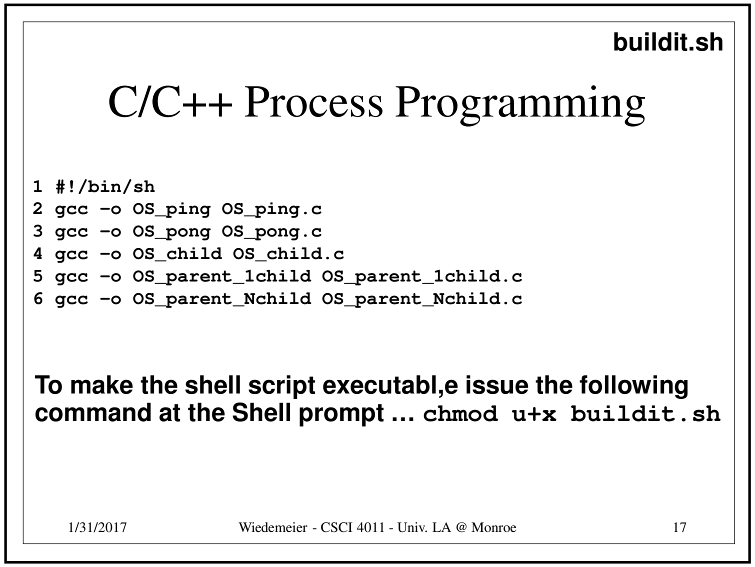
- **`fork()`**
  - A system call that is used to create a new child process.
  - Takes no arguments and returns a process ID.

C/C++ Process Programming

- **`wait()`**
  - A system call that blocks the calling process until one of it's child processes exits or a signal is received.
  - Takes the address of an integer variable and returns the process ID of the completed process.
  - `wait(NULL)` waits until any child process terminates.

C/C++ Process Programming

- **`sleep(#)`**
  - A system call that suspends the execution of the process for # seconds.
  - Takes an integer number representing seconds and returns nothing.

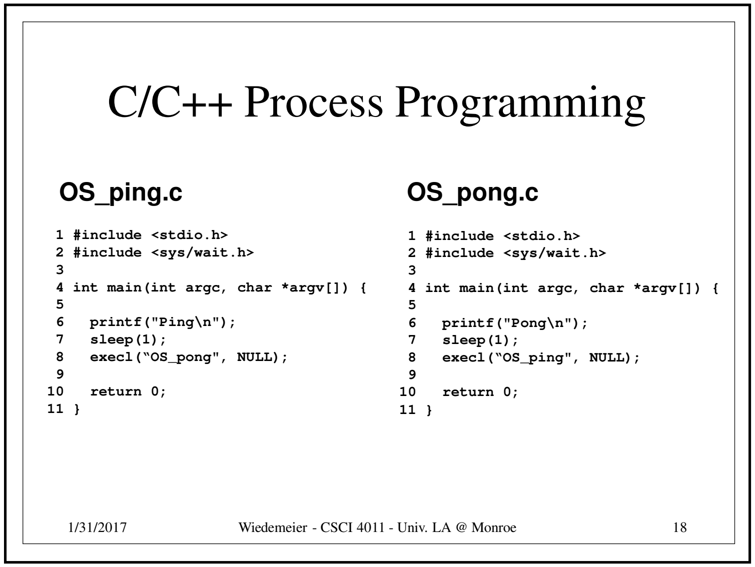


buildit.sh

C/C++ Process Programming

```
#!/bin/sh
gcc -o OS_ping OS_ping.c
gcc -o OS_pong OS_pong.c
gcc -o OS_child OS_child.c
gcc -o OS_parent_1child OS_parent_1child.c
gcc -o OS_parent_Nchild OS_parent_Nchild.c
```

To make the shell script executabl,e issue the following command at the Shell prompt … `chmod u+x buildit.sh`



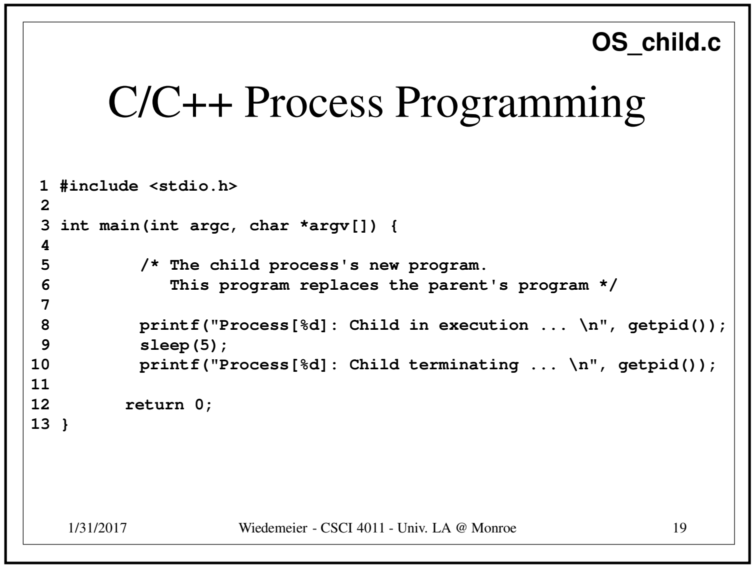
C/C++ Process Programming

OS_ping.c

```
#include <stdio.h>
#include <sys/wait.h>

int main(int argc, char *argv[]) {

  printf("Ping\n");
  sleep(1);
  execl(“OS_pong", NULL);

  return 0;
}
```

OS_pong.c

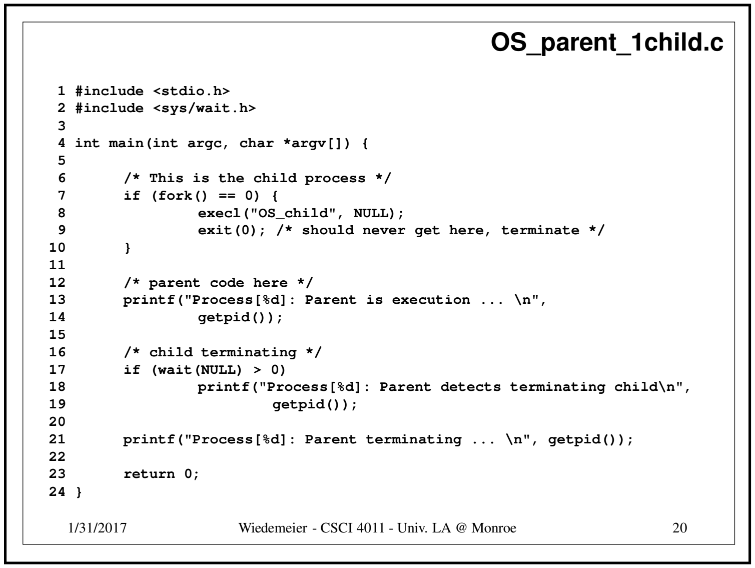
```
#include <stdio.h>
#include <sys/wait.h>

int main(int argc, char *argv[]) {

  printf("Pong\n");
  sleep(1);
  execl(“OS_ping", NULL);

  return 0;
}
```



OS_child.c

C/C++ Process Programming

```
#include <stdio.h>

int main(int argc, char *argv[]) {

        /* The child process's new program.
           This program replaces the parent's program */

        printf("Process[%d]: Child in execution ... \n", getpid());
        sleep(5);
        printf("Process[%d]: Child terminating ... \n", getpid());

      return 0;
}
```



OS_parent_1child.c

```
#include <stdio.h>
#include <sys/wait.h>

int main(int argc, char *argv[]) {

      /* This is the child process */
      if (fork() == 0) {
              execl("OS_child", NULL);
              exit(0); /* should never get here, terminate */
      }

      /* parent code here */
      printf("Process[%d]: Parent is execution ... \n",
              getpid());

      /* child terminating */
      if (wait(NULL) > 0)
              printf("Process[%d]: Parent detects terminating child\n",
                      getpid());

      printf("Process[%d]: Parent terminating ... \n", getpid());

      return 0;
}
```

OS_parent_Nchild.c

```c
#include <stdio.h>
#include <sys/wait.h>

int main(int argc, char *argv[]) {

        const int SIZE = 3;
        int i;

        for (i=0; i<SIZE; i++) {
                /* This is the child process */
                if (fork() == 0) {
                        execl("OS_child", NULL);
                        exit(0); /* should never get here, terminate */
                }
        }

        /* parent code here */
        printf("Process[%d]: Parent is execution ... \n", getpid());

        for (i=0; i<SIZE; i++) {
                /* child terminating */
                wait(NULL);
                printf("Process[%d]: Parent detects terminating child\n",
                        getpid());
        }

        printf("Process[%d]: Parent terminating ... \n",
                getpid());

        return 0;
}
```

Java Process Programming

- `import java.lang.*;`
- `import java.io.*;`
- `Class ProcessBuilder`

buildit.sh

Java Process Programming

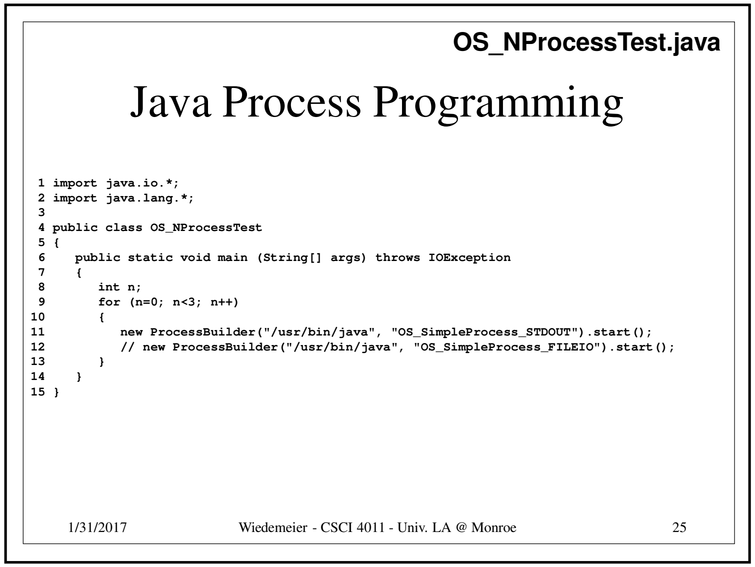
```sh
#!/bin/sh
javac OS_SimpleProcess_STDOUT.java
javac OS_SimpleProcess_FILEIO.java
javac OS_NProcessTest.java
```

To make the shell script executable issue the following command at the Shell prompt ... `chmod u+x buildit.sh`

OS_SimpleProcess_STDOUT.java

Java Process Programming

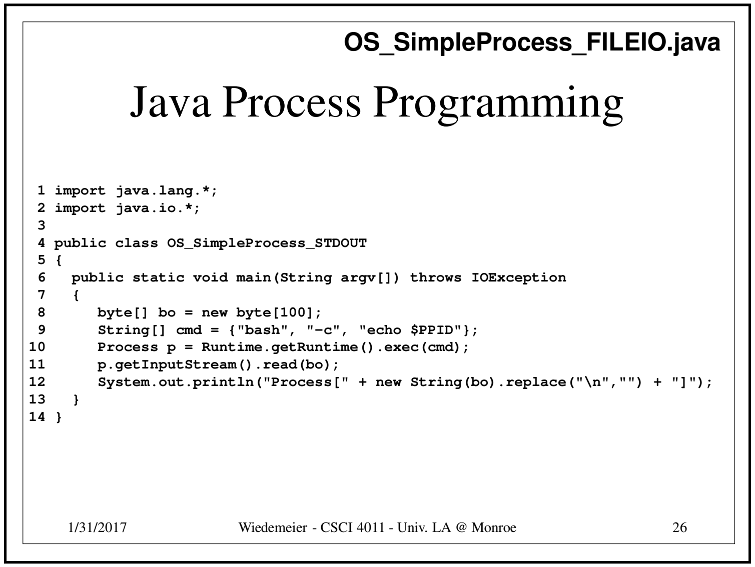
```java
import java.lang.*;
import java.io.*;

public class OS_SimpleProcess_STDOUT
{
  public static void main(String argv[]) throws IOException
  {
     byte[] bo = new byte[100];
     String[] cmd = {"bash", "-c", "echo $PPID"};
     Process p = Runtime.getRuntime().exec(cmd);
     p.getInputStream().read(bo);
     System.out.println("Process[" + new String(bo).replace("\n","") + "]");
  }
}
```



OS_NProcessTest.java

Java Process Programming

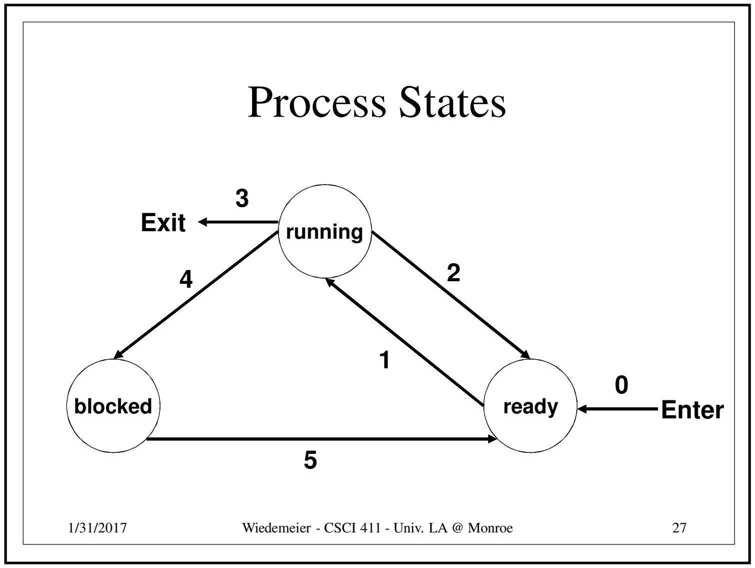
```
import java.io.*;
import java.lang.*;

public class OS_NProcessTest
{
   public static void main (String[] args) throws IOException
   {
      int n;
      for (n=0; n<3; n++)
      {
         new ProcessBuilder("/usr/bin/java", "OS_SimpleProcess_STDOUT").start();
         // new ProcessBuilder("/usr/bin/java", "OS_SimpleProcess_FILEIO").start();
      }
   }
}
```



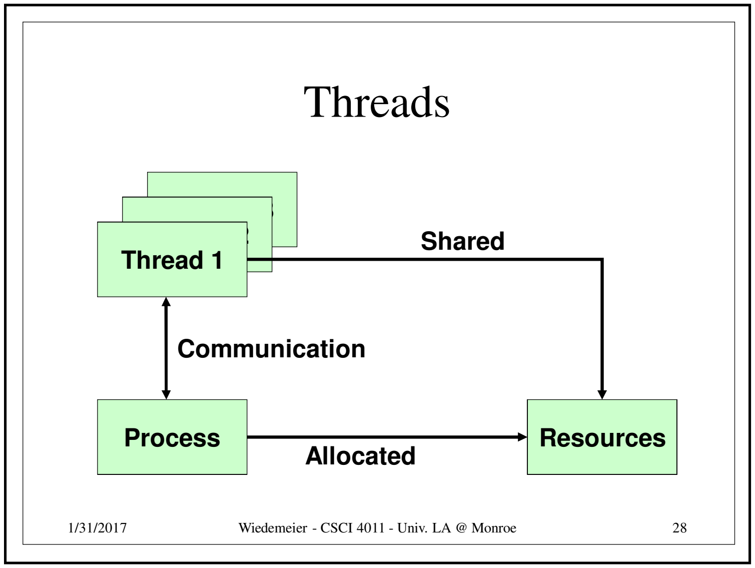
OS_SimpleProcess_FILEIO.java

Java Process Programming

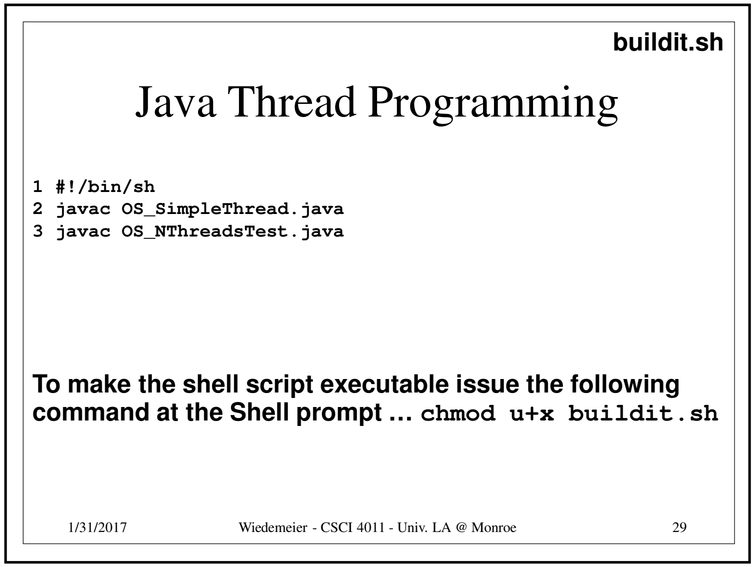
```
import java.lang.*;
import java.io.*;

public class OS_SimpleProcess_STDOUT
{
  public static void main(String argv[]) throws IOException
  {
     byte[] bo = new byte[100];
     String[] cmd = {"bash", "-c", "echo $PPID"};
     Process p = Runtime.getRuntime().exec(cmd);
     p.getInputStream().read(bo);
     System.out.println("Process[" + new String(bo).replace("\n","") + "]");
  }
}
```



Process States



Threads

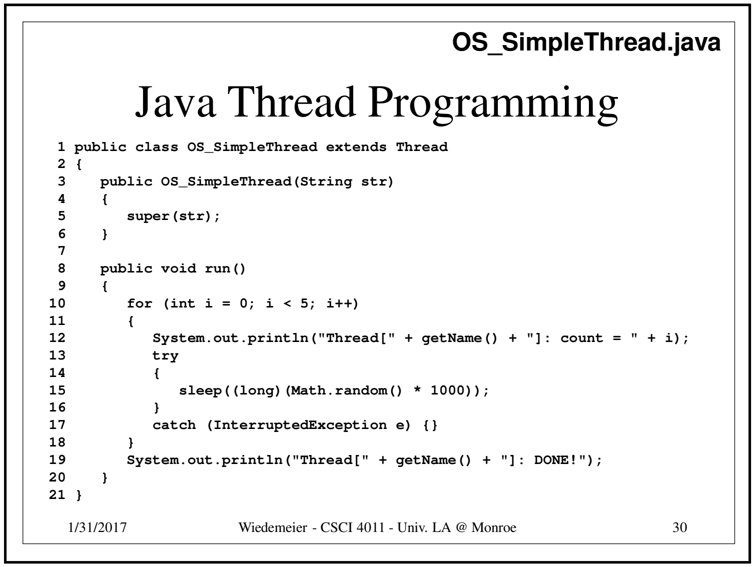


buildit.sh

Java Thread Programming

```
#!/bin/sh
javac OS_SimpleThread.java
javac OS_NThreadsTest.java
```

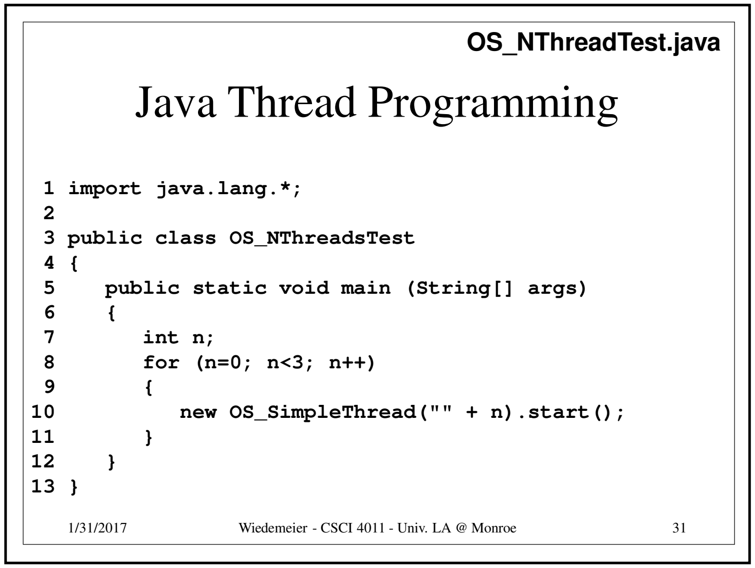
To make the shell script executable issue the following command at the Shell prompt … `chmod u+x buildit.sh`



OS_SimpleThread.java

Java Thread Programming

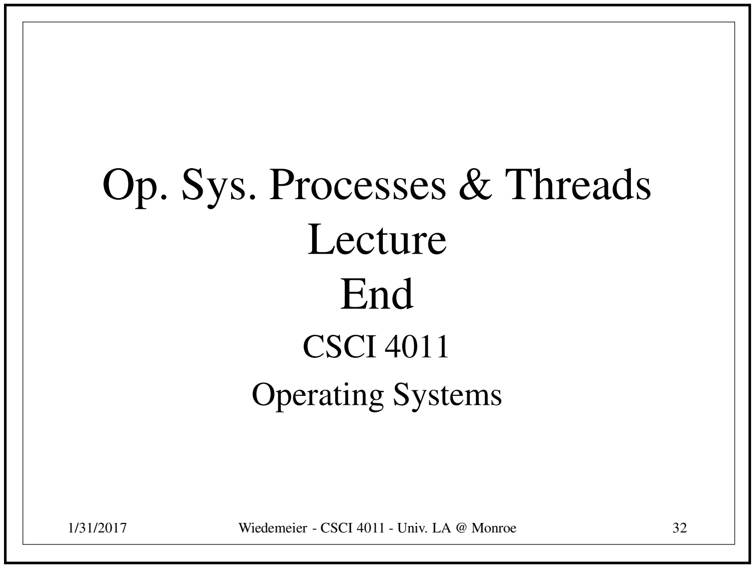
```
public class OS_SimpleThread extends Thread
{
   public OS_SimpleThread(String str)
   {
      super(str);
   }

   public void run()
   {
      for (int i = 0; i < 5; i++)
      {
         System.out.println("Thread[" + getName() + "]: count = " + i);
         try
         {
            sleep((long)(Math.random() * 1000));
         }
         catch (InterruptedException e) {}
      }
      System.out.println("Thread[" + getName() + "]: DONE!");
   }
}
```



OS_NThreadTest.java

Java Thread Programming

```
import java.lang.*;

public class OS_NThreadsTest
{
   public static void main (String[] args)
   {
      int n;
      for (n=0; n<3; n++)
      {
         new OS_SimpleThread("" + n).start();
      }
   }
}
```



Op. Sys. Processes & Threads Lecture End

CSCI 4011

Operating Systems