

# **KV Betriebssysteme**

## **Threads (II)**

## **KV Betriebssysteme**

### **Einheit 9: Threads (II)**

**Roland A. Eggersberger**

Inst. f. Informationsverarbeitung und Mikroprozessortechnik (FIM)  
Johannes Kepler Univ. Linz, Altenbergerstr. 69, A-4040 Linz, Austria

## **Inhalt**

Thread-Methoden  
Producer/Consumer  
Prozesse  
Beispiele

# KV Betriebssysteme

## Threads (II)

### Thread-Methoden (I)

- Threaderzeugung (I)

Indirekt möglich, über das **Runnable**-Interface

```
public interface Runnable {  
    public abstract void run();  
}
```

Auch hier ist **run** zu überschreiben

### Thread-Methoden (II)

- Threaderzeugung (II)

```
class RunnableOne implements Runnable {  
    public void run() {  
        :  
    }  
}
```

Ein Thread wird dann über das **Runnable**-Objekt angelegt

# KV Betriebssysteme

## Threads (II)

### Thread-Methoden (III)

- Threaderzeugung (III)

```
static public void main(String[] args) {  
    Runnable r = new RunnableOne();  
    Thread t = new ThreadOne(r);  
    t.start();  
}
```

Leere Folie

# KV Betriebssysteme

## Threads (II)

### Thread-Methoden (IV)

- Interrupts (I)

Für jeden Thread ist ein Interrupt-Flag verfügbar

Überprüft wird es mit der Klassenmethode  
`interrupted`

### Thread-Methoden (V)

- Interrupts (II)

Ein Thread kann auch das Interrupt-Flag eines anderen Threads prüfen (`isInterrupted`)

# KV Betriebssysteme

## Threads (II)

### Thread-Methoden (VI)

- Interrupts (III)

Mit `interrupt` werden Threads in ihrem Zustand beeinflusst

Ein wartender oder schlafender Thread wird geweckt bzw. das entsprechende Interrupt-Flag gesetzt

### Producer/Consumer (I)

- Ein ständig wiederkehrendes Problem ist der Austausch von Daten zwischen verschiedenen Threads.
- In vielen Fällen ist eine zweiseitige Kommunikation nicht nötig, kann aber problemlos aus zwei unidirektionalen Kommunikationskanälen hergestellt werden.

## KV Betriebssysteme

### Threads (II)

#### Producer/Consumer (II)

- Sowohl der Produzent wie auch der Konsument können den Buffer übernehmen, oder dieser kann ein besonderes Objekt sein.
- Jedes Objekt sollte nur eine Synchronisationsaufgabe besitzen (Also nicht zugleich Buffer sein und synchronized Methoden auch noch für andere Zwecke verwenden).

#### Producer/Consumer (III)

- Man bedenke immer, dass unter Umständen der Buffer nicht mehr leer wird oder keine weiteren Werte produziert werden.
- In diesem Fall darf kein ewiges `wait` erfolgen.

# KV Betriebssysteme

## Threads (II)

### Prozesse (I)

- Anwendung

Nicht nur Threads können gestartet werden,  
sondern auch vollständige Prozesse

Verwendet wird die Klasse Runtime

Gestartet wird ein Prozess mit der Methode `exec`

### Prozesse (II)

- Aufruf

```
Process proc =  
    Runtime.getRuntime().exec(cmd);
```

Dabei müssen aber für die Kommunikation mit dem  
Prozess spezielle Vorkehrungen getroffen werden

# KV Betriebssysteme

## Threads (II)

### Prozesse (III)

- Ein-/Ausgabe

Streams sind eine geeignete Möglichkeit, mit dem Prozess zu kommunizieren

```
proc.getInputStream()  
proc.getOutputStream()  
proc.getErrorStream()
```

Leere Folie

# **KV Betriebssysteme**

## **Threads (II)**

Leere Folie

Leere Folie

# KV Betriebssysteme

## Threads (II)

### Beispiele

Mutual Exclusion

Buffer

System Exec

### Mutual Exclusion

```
public class Mutex {  
    private Thread owner = null;  
    private int wait_count = 0;  
    public synchronized boolean lock(int millis)  
        throws InterruptedException {  
        if(owner == Thread.currentThread()) return true;  
        while(owner != null) {  
            try {  
                wait_count++;  
                wait(millis);  
            }  
            finally {  
                wait_count--;  
            }  
        }  
    }  
}
```

# KV Betriebssysteme

## Threads (II)

```
        if(millis != 0 && owner != null) return false;
    }
    owner = Thread.currentThread();
    return true;
}
public synchronized boolean lock()
    throws InterruptedException {
    return lock(0);
}
public synchronized void unlock() {
    if(owner != Thread.currentThread())
        throw new RuntimeException("Cannot unlock: Thread is not
                                      Mutex owner");
    owner = null;
    if(wait_count > 0)
        notify();
}
}
```

```
import java.io.PrintWriter;

public class MutexTest extends Thread {
    static PrintWriter out = new PrintWriter(System.out, true);
    private static Mutex mtx = new Mutex();
    private String message;
    public MutexTest (String name, String msg) {
        super(name); message = msg;
    }
    public synchronized void run() {
        try{
            mtx.lock();
        }
        catch(InterruptedException e) {
            out.println(e);
        }
        out.println("Achtung (" + getName() + "): " );
        Thread.yield();
```

# KV Betriebssysteme

## Threads (II)

```
        out.println(message + "!");
        mtx.unlock();
    }
    static public void main(String args[]) {
        out.println("Programmstart\n\n");
        MutexTest t1 = new MutexTest("Thread 1", "Message 1");
        MutexTest t2 = new MutexTest("Thread 2", "Message 2");
        MutexTest t3 = new MutexTest("Thread 3", "Message 3");
        t1.start(); t2.start(); t3.start();
        try{
            t1.join(); t2.join(); t3.join();
        }
        catch(InterruptedException e) {
            out.println(e);
        }
        out.println("\n\nProgrammende");
    }
}
```

### Ausgabe mit MutEx

Programmstart

Achtung (Thread 1):  
Message 1!  
Achtung (Thread 2):  
Message 2!  
Achtung (Thread 3):  
Message 3!

Programmende  
Application terminated

### Ausgabe ohne MutEx

Programmstart

Achtung (Thread 1):  
Achtung (Thread 2):  
Achtung (Thread 3):  
Message 1!  
Message 2!  
Message 3!

Programmende  
Application terminated

# KV Betriebssysteme

## Threads (II)

### Buffer

```
public class Test {  
    static PrintWriter out = new PrintWriter(System.out, true);  
    static public void main(String args[]) {  
        out.println("Programmstart\n\n");  
        URL url = null;  
        Buffer buf = new Buffer(5); // Einen Buffer erzeugen  
        try { url = new URL("http://www.fim.uni-linz.ac.at"); }  
        catch(MalformedURLException e) {  
            out.println(e); System.exit(1);  
        }  
        // Producer und Consumer erzeugen  
        Producer prod = new Producer(buf, url);  
        Consumer cons = new Consumer(buf);
```

```
        prod.start();  
        cons.start();  
        try {  
            prod.join();  
            cons.join();  
        }  
        catch(InterruptedException e) {  
            out.println(e);  
        }  
        out.println("\n\nProgrammende");  
    }  
}
```

# KV Betriebssysteme

## Threads (II)

```
// Buffer fuer Strings
// Laenge kann bei Erzeugung angegeben werden
public class Buffer extends Thread {
    protected String[] buf;
    protected int last; // Naechster freier Index
    protected int EOF;
    // Wenn EOF >= 0, EOF = Anzahl der String im Buffer bis EOF

    public Buffer(int len) {
        buf = new String[len];
        last = 0; EOF = -1;
    }
    public synchronized void setEOF() {
        EOF = last;
        // Wenn Buffer leer, koennte der Consumer warten
        if(EOF == 0)
            notify();
    }
```

```
public synchronized boolean isEOF() { return EOF == 0; }
public synchronized boolean put(String item) {
    // Einen String in den Buffer stellen
    if(EOF >= 0) // Keine Eingabe nach EOF
        return false;
    // Ist ein Platz frei?
    if(last == buf.length) {
        try { wait(); }
        catch(InterruptedException e) { return false; }
    }
    buf[last++] = item;
    // Geht nur, weil nur ein Producer (sonst
    // koennte damit ein anderer Producer geweckt werden)
    // Mehrere Producer -> Semaphor oder aehnliches verwenden
    notify();
    return true;
}
```

# KV Betriebssysteme

## Threads (II)

```
public synchronized String get() { // Einen String auslesen
    if(EOF == 0) return null; // Nach EOF gibt es nichts mehr
    if(last == 0) // Ist der Buffer leer? {
        try { wait(); }
        catch(InterruptedException e) { return null; }
        if(EOF == 0)
            // Wenn der Consumer gewartet hat (Buffer leer),
            // waehrend der Producer EOF setzte ist dies notwendig
            return null;
    }
    String res = buf[0];
    // Array nach vorne kopieren
    System.arraycopy(buf, 1, buf, 0, --last);
    if(EOF > 0) EOF--; // EOF anpassen wenn noetig
    notify(); // ACHTUNG: Siehe Put
    return res;
}
```

Leere Folie

## KV Betriebssysteme

### Threads (II)

```
public class Producer extends Thread {  
    URL url;  
    Buffer buf;  
    public Producer(Buffer buf, URL url) {  
        this.url = url; this.buf = buf;  
    }  
    public void run() {  
        BufferedReader in = null;  
        String str;  
        // URL oeffnen und Daten holen  
        try {  
            URLConnection uc = url.openConnection();  
            uc.setUseCaches(false);  
            uc.connect();  
            in = new BufferedReader(  
                new InputStreamReader(uc.getInputStream()));  
        }  
    }
```

```
    catch(IOException e){  
        System.out.println("Could not connect to"  
            + url + "\n" + e + "\n");  
        return;  
    }  
    // LeseSchleife  
    try {  
        while((str = in.readLine()) != null) buf.put(str);  
    }  
    catch(IOException e) {  
        System.out.println("Could not read from "  
            + url + "\n" + e + "\n");  
        return;  
    }  
    // EOF signalisieren  
    buf.setEOF();  
}
```

## KV Betriebssysteme

### Threads (II)

```
public class Consumer extends Thread {  
    Buffer buf;  
    public Consumer(Buffer buf) {  
        this.buf=buf;  
    }  
    public void run() {  
        String tagBuffer = ""; // Eigener Buffer wegen  
        // Tags, die auch über Zeilenwechsel gehen koennen  
        String line = buf.get();  
        while(line != null) {  
            String res = new String("");  
            tagBuffer += line + "\n";  
            int lt = tagBuffer.indexOf('<');  
            while(lt >= 0) { // Start eines Tags  
                int gt = tagBuffer.indexOf('>', lt);  
                if(gt >= 0) {  
                    res += tagBuffer.substring(0, lt);  
                }  
                tagBuffer = tagBuffer.substring(gt + 1,  
                                              tagBuffer.length());  
                lt = tagBuffer.indexOf('<');  
                // Ende eines Tags, weiteren in selber Zeile suchen  
            }  
            else lt = -1;  
            // Schleifenende, wenn Tag nicht in dieser Zeile aus  
        }  
        System.out.print(res);  
        line = buf.get();  
    }  
    if(!buf.isEOF())  
        System.out.print("Error during reading from buffer");  
    }  
}
```

# **KV Betriebssysteme**

## **Threads (II)**

Leere Folie

Leere Folie

# KV Betriebssysteme

## Threads (II)

### System Exec

```
public class SystemExec {  
    static PrintWriter cout = new PrintWriter(System.out, true);  
  
    static public void main(String args[]) {  
        String outfile;  
        String infile;  
        String errfile = null;  
        String command;  
        Process proc = null;  
        InputStreamReader out = null;  
        InputStreamReader err = null;  
        OutputStreamWriter in = null;  
        cout.println("Programmstart\n\n");  
    }  
}
```

```
// Parse arguments  
if(args.length < 3) {  
    cout.println("Usage: SystemExec [-e Error-file] Input-  
    file Output-file Programname {Arguments}");  
    System.exit(1);  
}  
int curIndex = 0;  
if(args[curIndex].equals("-e") ||  
    args[curIndex].equals("-E")) {  
    curIndex++;  
    errfile = args[curIndex++];  
}  
if(args.length-curIndex < 3) {  
    cout.println("Usage: SystemExec [-e Error-file] Input-  
    file Output-file Programname {Arguments}");  
    System.exit(1);  
}
```

# KV Betriebssysteme

## Threads (II)

```
infile = args[curIndex++];
outfile = args[curIndex++];
command = args[curIndex++];
// Add up rest of arguments as args for the program
for(int i = curIndex; i < args.length; i++)
    command += " " + args[i];
try {
    // Start program
    proc = Runtime.getRuntime().exec(command);
    // getInputStream returns an InputStream,
    // which is stdout of the command. You can read
    // the output of the program through this stream.
    out = new InputStreamReader(proc.getInputStream());
    // getErrorStream returns an InputStream, which
    // is stderr of the command
    err = new InputStreamReader(proc.getErrorStream());
```

```
// getOutputStream returns an OutputStream, which
// is stdin of the command. What you write
// in here is fed as input to the program.
in = new OutputStreamWriter(proc.getOutputStream());
}
catch(IOException e) {
    cout.println(e);
    System.exit(1);
}
// Start new Threads for handling the streams
Thread inH = new StdinHandler(in, infile);
Thread outH = new StdoutHandler(out, outfile);
Thread errH = new StderrHandler(err, errfile);
outh.start();
errH.start();
inH.start();
```

# KV Betriebssysteme

## Threads (II)

```
// Wait till all are closed
try {
    inH.join();
    outH.join();
    errH.join();
    proc.waitFor();
}
catch(InterruptedException e) {
    cout.println(e);
}
// Print exit value
cout.println("\n>>>>The programme returned the
exit-code " + proc.exitValue() + "<<<<<");
cout.println("\n\nProgrammende");
// Explicitely end Java VM for setting an exit value
System.exit(0);
}
```

Leere Folie

# KV Betriebssysteme

## Threads (II)

```
public class StdinHandler extends Thread {  
    protected OutputStreamWriter stdin;  
    protected String infile;  
    public StdinHandler(OutputStreamWriter os, String file) {  
        if(os == null || file == null)  
            throw new IllegalArgumentException();  
        stdin = os;  
        infile = file;  
    }  
    public void run() {  
        BufferedReader fr = null;  
        try { fr = new BufferedReader(new FileReader(infile)); }  
        catch(IOException e) {  
            // Should notify main to stop all other threads and end  
            System.out.println("Stdin: " + e);  
            return;  
        }  
    }  
}
```

```
try {  
    String inp = fr.readLine();  
    while(inp != null) {  
        inp += "\n";  
        try { stdin.write(inp,0,inp.length()); }  
        catch(IOException ie) {  
            // We print no error here, as it may happen that the  
            // process has already terminated (if not all was  
            // read). No way to find this out.  
            // Just close down everything nicely.  
            fr.close();  
            return;  
        }  
        inp = fr.readLine();  
    }  
}
```

# KV Betriebssysteme

## Threads (II)

```
// We must close stdin to signal our process,  
// that the end of the stream is reached  
try {  
    stdin.flush();  
    stdin.close();  
}  
catch(IOException e) {  
    // See comments above  
}  
fr.close();  
}  
catch(IOException e) {  
    System.out.println("Stdin: " + e);  
}  
}  
}  
}
```

Leere Folie

# **KV Betriebssysteme**

## **Threads (II)**

Leere Folie

Leere Folie