Using the JACK Audio Connection Kit with Java

Jens Gulden, jgulden@cs.tu-berlin.de
What's JJack for?

- Write Java programs that act as interconnectable JACK clients
- Record / generate / transform audio with Java, JACK-compliant
- Possible alternative to internal JavaSound API (`javax.sound.sampled.*`)
Overview

- JACK
- JACK & Java
- JJack direct JACK reflection
- JJack high-level API
- Examples
JACK processing loop

"Compressor"

process()

"Reverb"

process()

"Synth"

process()

JACK server

Capture

Input

Output

Playback

ALSA+Hardware

Linux Audio Conference LAC 2007, March 23 – Jens Gulden, jgulden@cs.tu-berlin.de
JACK processing loop

- Process (Compressor)
- Process (Reverb)
- Process (Synth)

Input

Capture

ALSA

JACK server

Output
JACK processing loop

```
process()  
"Compressor"

JACK server
  Capture   Playback

(process)  (process)
"Reverb"   "Synth"
```

Linux Audio Conference LAC 2007, March 23 – Jens Gulden, jgulden@cs.tu-berlin.de
JACK & Java

"Compressor"

"JavaReverb"

"Synth"

process()

process()

process()

JACK server

Capture

Playback

Linux Audio Conference LAC 2007, March 23 – Jens Gulden, jgulden@cs.tu-berlin.de
JACK & Java

Java Virtual Machine (JVM)

```java
class JavaReverb {
    process(..) {
        ...
    }
}
```

"JavaReverb"

"Compressor"

"Synth"

JACK server

Capture

Playback

Linux Audio Conference LAC 2007, March 23 – Jens Gulden, jgulden@cs.tu-berlin.de
JJack Approach

- Native library acts as JACK client (lib/i386/linux/libjjack.so)
- Bound to class JJackSystem by Java Native Interface (JNI)
- Native call to `process()` is delegated to Java-method `JJackSystem.process( .. )`
JJack – Examples 1

![JJack Example 1](image1)

![Oscilloscope Example](image2)
JJack Architecture

- 2 levels of architecture:
  - **Direct** JACK reflection (simple)
  - **High-level** API
Direct JACK reflection

- Single `process()` method in Java does the job of a JACK client
- Easy to use
- Sufficient for most applications
Direct JACK reflection

• Simple JJack client in Java:

```java
public void process(JJackAudioEvent e) {
    float v = getVolume(); // parameter from gui

    for (int i = 0; i < e.countChannels(); i++) {
        FloatBuffer in = e.getInput(i);
        FloatBuffer out = e.getOutput(i);
        int cap = in.capacity();
        for (int j = 0; j < cap; j++) {
            float a = in.get(j);
            a *= v; // amplify signal
            if (a>1f) a=1f; else if (a<-1f) a=-1f;
            out.put(j, a);
        }
    }
}
```
Direct JACK reflection

• Deploying:

```java
public class JJackTest {

    public static void main(String[] args) {
        // get JACK system's sample rate, initialize
        int sampleRate = JJackSystem.getSampleRate();
        System.out.print("Sample-rate: "+ sampleRate);

        // set single processing client
        MyJJackClient client = new MyJJackClient();

        JJackSystem.setClient(client);

        /* ... */
    }
}
```
High-Level API

- Optional model for audio signal-routing, **internal** to Java
- Object-oriented API for multiple audio-processors
  - e.g. Channel, Port, AudioProducer, AudioConsumer...
- **JavaBeans**-compatible interconnection mechanism
Example: High-Level API
Example: BeanBuilder
Real-World Application

- **Music sequencer** Frinika (GPL, http://www.frinika.com/)
- Entirely written in Java, implements sequencer model `javax.sound.midi.*`
- Audio I/O either via JJack or JavaSound
Example: Frinika
Native Bridge Implementation

- **High performance thanks to** JDK1.4 `java.nio.*` („New I/O“)

- **Native** `*float` memory-blocks get directly mapped onto Java `float[]` arrays (`java.nio.DirectByteBuffer`)

- **No conversion**, even no copying
Native Bridge Implementation

Java Virtual Machine (JVM)

```java
class MyClient {
    public void process(...) {
    ...
    }
}
```

```java
class JJackSystem {
    public void process(JJackAudioEvent e) {
        client.process(e);
    }
}
```

```c
int process(jack_nframes_t nframes, void *arg) {
    ...
    (*jvm)->AttachCurrentThread(jvm, ...);
    ...
    ByteBuffer = (*env)->NewDirectByteBuffer(...);
    ...
    /* callback to Java, JJackSystem.process() */
    /* (*env)->CallStaticVoidMethod(...) */
}
```
Comparing JJack and JavaSound (1/2)

<table>
<thead>
<tr>
<th></th>
<th>JJack</th>
<th>JavaSound</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>interoperability</strong></td>
<td>high, virtual cable-connectivity</td>
<td>low, device I/O oriented</td>
</tr>
<tr>
<td><strong>timing latency</strong></td>
<td>JACK timing, potential low-latency below 5 ms on fast machines</td>
<td>device-dependent, varying among OSs and JDK versions</td>
</tr>
<tr>
<td><strong>realtime-capability</strong></td>
<td>yes (application can run in a user-thread, JACK in a realtime thread)</td>
<td>no (unless the whole JVM runs in a realtime thread)</td>
</tr>
</tbody>
</table>
Comparing JJack and JavaSound (2/2)

<table>
<thead>
<tr>
<th>process architecture</th>
<th>pull-architecture (JACK thread calls <code>process()</code>)</th>
<th>push-architecture (application is responsible for delivering data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>internal data-format</td>
<td>32-bit float (less aliasing, higher performance for interconnection)</td>
<td>16-bit integer (faster processing for simple clients, else conversion)</td>
</tr>
<tr>
<td>number of channels</td>
<td>any</td>
<td>according to hardware driver or software mixer</td>
</tr>
<tr>
<td>operating system</td>
<td>Linux or Mac (JACK required)</td>
<td>Any (JDK &gt;= 1.3)</td>
</tr>
</tbody>
</table>
Future Plans

• Improve current codebase (with Frinika developers)
• Garbage-collection issues?
• JACK-MIDI?
Conclusion

- **JJack enables** Java as a programming language for interconnectable audio clients
- Audio-specific **architecture**
- May provide better **performance** than JavaSound (some systems)
- **Alternative** to JavaSound API
Thank you!

http://jjack.berlios.de/

Jens Gulden, jgulden@cs.tu-berlin.de

The background image in the heading is taken from http://www.jackaudio.org/.
Linux Audio Conference LAC 2007, March 23 – Jens Gulden, jgulden@cs.tu-berlin.de
Appendix: Latency Measurements

Test-Client

Linux 2.5.15-4-realtime, JJack with JACK-period 256, JDK1.5, onboard VIA hardware

Test-Server

alternative configurations:
• Linux 2.6.15-4-realtime /
• Linux 2.6.15-5 /
• Windows XP
• JDK1.4 / JDK1.6
• Onboard-sound / USB-sound
Appendix: Latency Measurements

- Benchmark application: 
  de.gulden.framework.jjack.util.benchmark.AudioBenchmark (current CVS)

- Note: test-client induces latency itself, suitable for comparing setups, but no absolute results
# Appendix: Latency Measurements

<table>
<thead>
<tr>
<th>System</th>
<th>Onboard-sound</th>
<th>JACK period</th>
<th>JDK1.4</th>
<th>JDK1.6</th>
<th>JavaSound JDK1.4</th>
<th>JavaSound JDK1.6</th>
<th>JavaSound Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux 2.6.15-4-</td>
<td>512</td>
<td></td>
<td></td>
<td></td>
<td>333.71</td>
<td></td>
<td>131071</td>
</tr>
<tr>
<td>real-time</td>
<td>256</td>
<td>18.82</td>
<td></td>
<td></td>
<td></td>
<td>88200</td>
<td></td>
</tr>
<tr>
<td></td>
<td>128</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>65536</td>
<td></td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>8.62</td>
<td></td>
<td></td>
<td></td>
<td>32768</td>
<td></td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>4.94</td>
<td></td>
<td></td>
<td>325.18</td>
<td></td>
<td>16384</td>
</tr>
<tr>
<td>Windows XP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>178.67</td>
<td>131071</td>
<td></td>
</tr>
<tr>
<td></td>
<td>512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>128</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>185.79</td>
<td>28.26</td>
<td></td>
</tr>
</tbody>
</table>

The values are each averaged from at least 20 individual measurements.

Linux Audio Conference LAC 2007, March 23 – Jens Gulden, jgulden@cs.tu-berlin.de
# Appendix: Latency Measurements

<table>
<thead>
<tr>
<th>USB-sound</th>
<th>JACK period</th>
<th>JACK JDK1.4</th>
<th>JACK JDK1.6</th>
<th>JavaSound JDK1.4</th>
<th>JavaSound JDK1.6</th>
<th>JavaSound Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux 2.6.15-4-realtime</td>
<td>512</td>
<td>10.74</td>
<td>10.34</td>
<td>372.3</td>
<td>354.34</td>
<td>19.52</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>128</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux 2.6.15-5</td>
<td>512</td>
<td>10.17</td>
<td>9.53</td>
<td>21.17</td>
<td>20.58</td>
<td>131071</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>88200</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>65536</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>32768</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20.64</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16384</td>
</tr>
<tr>
<td>Windows XP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>179.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>31.02</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>65536</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>32768</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>30.71</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16384</td>
</tr>
</tbody>
</table>

The values are each averaged from at least 20 individual measurements.

Linux Audio Conference LAC 2007, March 23 – Jens Gulden, jgulden@cs.tu-berlin.de