Instrumentalmusik und Live-Elektronik

LVA 17.0078

Week 3

- concurrency and shreds
- events
- different interaction frameworks smelt/wekinator

shreds

ChucK gives you an easy way to execute multiple pieces of code at once using shreds. To spork a function (execute it in parallel with the current code), use
spork ~ *function_name*(*param1_value*, *param2_value*,...)

```
fun void play_C() {
     SinOsc s => dac;
     Std.mtof(60) => s.freq;
     1::second => now;
     s =< dac;
}

fun void play_Fsharp() {
     SinOsc s => dac;
     Std.mtof(66) => s.freq;
     1::second => now;
     s =< dac;
}

spork ~play_C();
spork ~play_Fsharp();
1::minute => now;
```

line at the end of the main code ensures that the parent won't die before the child shreds execute.

shreds

- The ‘me’ keyword : refers to the current shred

spork ~ myshred();

me.yield(); //suspends the current shred and lets other shreds that are scheduled(schreduled) for now to run

me.exit(); // exit the current shred

- Using machine.add()

Machine.add(string path); //sporks that shred which is added

Machine.remove(id);

Machine.replace(id, string path);

events

midi, osc, costume

- create an event object
- the event suspends the current shred
- when the event triggered >>> one or more of the shreds run
- triggering may originate from another chucK shred or outside (e.g. midi or osc)

```
event.signal(); // releases the first shred on that queue
event.broadcast(); //releases all shreds queued by that event
```

OSC

OSC Sender

- Decide on a *host* to send the messages to. E.g., "blah.local" if sending to computer named "blah" or "localhost" to send to the same machine that is sending.
- Decide on a port to which the messages will be sent. This is an integer, like 1234.
- Message "address": For each type of message you're sending, decide on a way to identify this type of message, formatted like a web URL e.g., "conductor/downbeat/beat1" or "/message1"
- Message contents: decide on whether the message will contain data, which can be 0 or more ints, floats, strings, or any combination of them.

OSC

OSC Sender Code

Create an OscSend object:

```
OscSend xmit;
```

Set the host and port of this object:

```
xmit.setHost("localhost", 1234);
```

For every message, start the message by supplying the address and format of contents, where “f” stands for float, “i” stands for int, and “s” stands for string:

To send a message with no contents:

```
xmit.startMsg("conductor/downbeat");
```

To send a message with one integer:

```
xmit.startMsg("conductor/downbeat, i");
```

To send a message with a float, an int, and another float:

```
xmit.startMsg("conductor/downbeat, f i f");
```

For every piece of information in the contents of each message, add this information to the message:

e.g., to add an int:

```
xmit.addInt(10);
```

OSC

OSC Receiver

- Port: decide what port to listen on. This must be the same as the port the sender is using.
- Message address and format of contents: This must also be the same as what the sender is using; i.e., the same as in the sender's startMsg function.

Code:

```
for each receiver Create an OscRecv object:
    OscRecv orec;
Tell the OscRecv object the port:
    1234 => orec.port;
Tell the OscRecv object to start listening for OSC messages on that port:
    orec.listen();
For each type of message, create an event that will be used to wait on that
type of message, using the same argument list as the sender's startMsg
function:
orec.event("conductor/downbeat, i") @=> OscEvent myDownbeat;
```

OSC

OSC Receiver

To wait on an OSC message that matches the message type used for a particular event e, do

```
e => now;
```

To process the message:

Grab the message out of the queue (mandatory!)

```
e.nextMsg();
```

For every piece of information in the message, get the data out. You must call these functions in order, according to your formatting string.

```
e.getInt() => int i;
e.getFloat() => float f;
e.getString() => string s;
```

tools

Small Musically Expressive Laptop Toolkit

Code examples for keyboard, trackpad, smack sensor, and mic:
http://smelt.cs.princeton.edu

Code examples for joystick, osc, ...
http://chuck.cs.princeton.edu/doc/examples/

download and install wekinator
http://wekinator.cs.princeton.edu

if you are interested in visuals to add to your audio, try processing
http://www.processing.org

have fun with it!

Assignment 3 (10 points)

Due: Wed. April 17 at 3.00 PM

Start with sketching your ideas to create a mini instrument with chucK. Think about the interactions you'd like to have with the instrument as a performer on stage. (try to make it interesting for the audience as well)
Incorporate one of the interactions we talked about in the class (from smelt or osc, ...). Think if your instrument is more interesting in an ensemble setting or for a solo performer and why? Try to make as many abstractions as possible using functions and shreds. Please submit all your chucK files by the deadline per email and be ready to discuss and make a mini performance in the class on April 18th. Have fun with it and enjoy the holidays.