10/2 CS240 - Threads & Events

Announcements

For next class (Tuesday 10/5)

1. Read: Eliminating Receive Livelock in an Interrupt-Driven Kernel
2. Submit answers to reading questions (see course schedule) before class

When saying something in class, please state your first name the first time you say something.

Paper Backgrounds

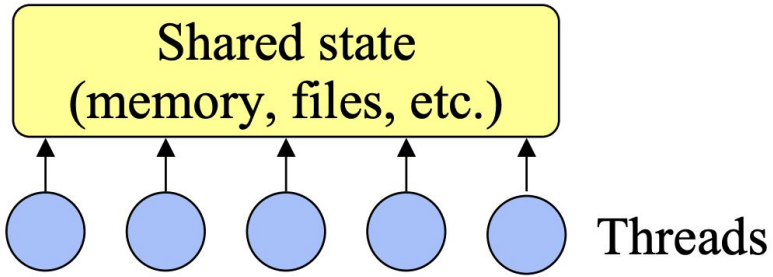
- Why Threads Are A Bad Idea (for most purposes)
  - 1996 Invited conference talk
  - Most machines uniprocessors at the time
- Why Events Are A Bad Idea (for high-concurrency servers)
  - 2003 HotOS paper
  - Former event advocates argue for threads
- Cooperative Task Management without Manual Stack Management
  or, Event-driven Programming is Not the Opposite of Threaded Programming
  - 2002 paper from Microsoft Research
  - Based on experience building event and thread systems

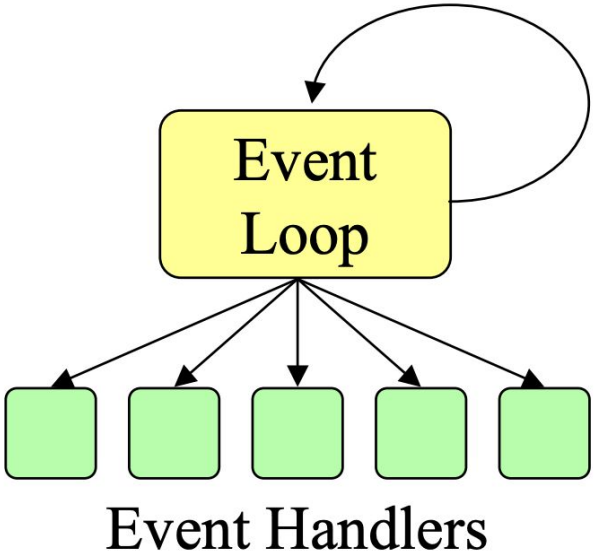
What is the concurrency in this Linux/Mac code?

```
#include <stdio.h>

int main(void) {
    char *str = "Hello, world!\n";
    // Could some other execution happen between these statements?
    printf(str);
    return 0;
}
```

Threads vs Events: Explain these figures





What were the metrics used to evaluate thread and events

Problems with Threads?

Problem with Events?

CS142: Threads versus Events using Callbacks

Threads:

```
request = readRequest(socket);
reply = processRequest(request);
sendReply(socket, reply);
```

Events with callbacks:

```
readRequest(socket, doneRead);

function doneRead(request) {
   processRequest(request, doneProcess);
}
function doneProcess(reply) {
   sendReply(socket, reply, doneSend);
}
function doneSend(errorCode) {
   doneCb(errorCode)
}
```

CS142: Threads versus Events using Callbacks

Threads:

```
request = readRequest(socket);
reply = processRequest(request);
sendReply(socket, reply);
```

Events with callbacks:

```
readRequest(socket, function(request) {
   processRequest(request,
       function (reply) {
         sendReply(socket,reply,doneCb);
       });
});
```

Which one (thread or events) is harder to debug?

Which would is better from a Software Engineering point

- Threads
- Events

Metrics:

Understandability?

System evolution?

Software evolution: routine starts to block

```
function () {

    // access shared data
    routine()
    // access shared data

}
```

According to Ousterhout: When should you use threads?

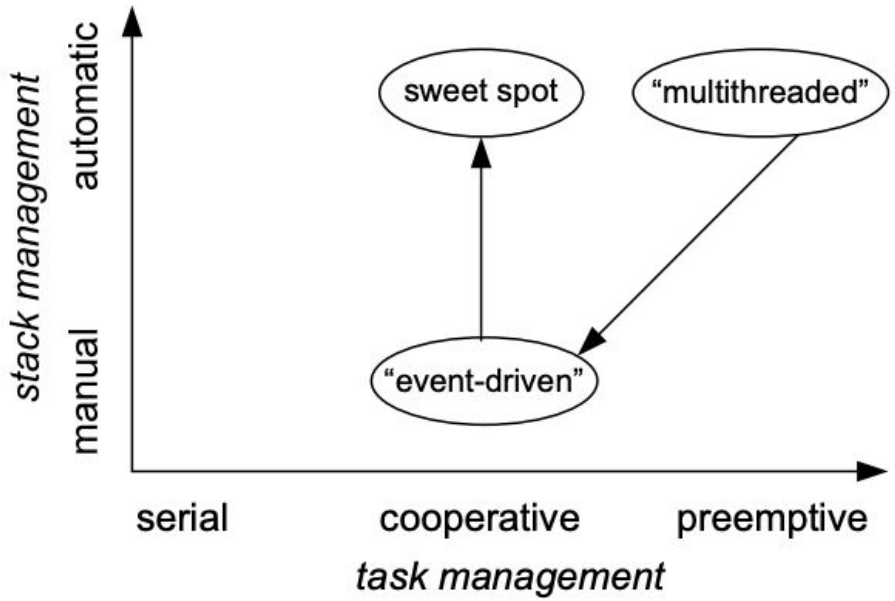
Do the other papers agree with him?

What are user level threads?

Why are they good?

What is hard about having a large number of threads?

Explain axis of this graph



What did you think of the Continuation code?

Section 3.1

What is stack ripping?

What are closures and how are they related to stack ripping?

I/O management

- synchronous
- asynchronous

Data partitioning

- Coloring techniques?

Concurrency in the today's computing environments

- Web - JavaScript - All in on events
  - Both in the browser and server (Node.js)
  - Language has closures and support for continuations (Promises)
- AI - Python - Limited threads
  - Global interpreter lock
  - Multiprocessing library (shared nothing) common
  - Most parallelism is done using threads in a callout module written in C++
- Go language - Lightweight threads (goroutines)
  - Encourage lots of threads but also encourage message passing (channels) to avoid locks, etc.

Closure in JavaScript

```
let globalVar = 1;
function localFunc(argVar) {
 let localVar = 0;
 function embedFunc() {return ++localVar + argVar + globalVar;}
 return embedFunc;
}
let myFunc = localFunc(10);
```

Promises in JavaScript

```
async function doIt(fileName) {
    let file = await ReadFile(fileName);
    let data = await doSomethingOnData(file);
    let moreData = await doSomethingMoreOnData(data);
    return finalizeData(moreData);
}
```

Node.js example using callback functions.

```
net.createServer(function (socket) {
   socket.on('data', function (fileName) {
      fs.readFile(fileName.toString(), function (error, fileData) {
        if (!error) {
           socket.write(fileData);  // Writing a Buffer
        } else {
           socket.write(error.message); // Writing a String
        }
        socket.end();
     });
  });
}).listen(4000);
```

Callback Hell

Callback hell - TJ Holowaychuk's why Node sucks:

1. you may get duplicate callbacks
2. you may not get a callback at all (lost in limbo)
3. you may get out-of-band errors
4. emitters may get multiple “error” events
5. missing “error” events sends everything to hell
6. often unsure what requires “error” handlers
7. “error” handlers are very verbose
8. callbacks suck