Implementing the Event Loop

CS 520
Dept. of Computer Science
Univ. of New Hampshire

Interface

```
handle ← createEventLoop()
         startEventLoop(handle, func, args)
         registerEvent(handle, name, func)
         announceEvent(handle, event, args)
         stopEventLoop(handle)
         cleanupEventLoop(handle)
```

C function pointers

```
void (*func)(void *)
```

$\hookrightarrow$ i.e. event handlers take one argument and return nothing

calling via a function pointer:

```
(*func)(x)

func(x)
```

Event Names

just strings

copy of name will be made
when registering event

createEventLoop()

allocate object to maintain state

initialize queue
- stores functions/arguments to be executed
- initially empty

initialize event registry
- maps event names to event handler functions
- initially empty

initialize mutex & condition variable
- controls access to state
- used to block event loop when queue is empty

return pointer to state as "handle"

startEventLoop(handle, func, arg)

insert func with arg into queue

start the event loop:

```
while(!terminated) {
    dequeue first func/arg on queue
    func(arg)
}
```

startEventLoop will not return until
the event loop is terminated

registerEvent(handle, name, func)

insert name with func into registry

announceEvent(handle, event, args)

```
lookup event in registry
if found, place its handler with the args
        on the end of the queue
if not, report "unhandled event"
```

stopEventLoop(handle)

should cause event loop to terminate
when currently executing event handler
returns

to simplify termination, this function
must be called from a handler

↳ event loop is single threaded
therefore no race condition

cleanupEventLoop(handle)

cleanup memory allocated for state

asynchronous calls

registerEvent and announceEvent could be called by another thread

therefore need to use mutex to protect the state

what if queue is empty?

thread executing event loop should be <u>blocked</u>

$\hookrightarrow$ use condition variable

will be signaled by announceEvent

the user is responsible for allocating
and deallocating any arguments that
are passed to event handlers