Multiprocessor Support for Event-Driven Programs

Nickolai Zeldovich  
(Stanford University)

Speaker: Ching-Her Chang
Outline

- Introduction
- Uniprocessor Event-Driven Design
- Multiprocessor Design
- Implementation
- Evaluation
- Relate work
- Conclusion
Introduction

- event-based programs
- Libasync-smp
event-based programs

- Programs typically using threads or events.
- Event-based programs are structured as a collection of *callback* functions.
- Event-based programs execute callbacks serially.
Libasync-smp

- Libasync-smp supports event-driven programs on multiprocessors.
- Libasync-smp is based on the libasync library.
- The modifications for libasync-smp include:
  1. coordinating access to the shared internal state of a few libasync modules.
  2. support for colors.
  3. scheduling callbacks on multiple CPUs.
Uniprocessor Event-Driven Design

- Libasync
- Event-driven Programming
- Interaction with multiprocessors
Libasync

- `select()` system call
- `wrap()` macro

- Utility modules:
  1. RPC utility module
  2. DNS module
  3. I/O module
Event-driven Programming

```c
main()
{
    int afd = inetsocket(SOCK_STREAM, 80); // listen on TCP port 80
    fdcb(afd, READ, wrap(accept_cb, afd)); // register callback for new connections
    amain(); // start main loop
}
accept_cb(int afd) // called when a new connection arrives
{
    int fd = accept(afd, ...);
    str inBuf(""""); // new ref-counted buffer
    fdcb(fd, READ, wrap(req_cb, fd, inBuf)); // register callback for incoming data
}
```
Event-driven Programming

```c
req_cb(int fd, str inBuf) // called when data arrives
{
    read(fd, buf, ...);
    append input to inBuf;
    if(complete request in inBuf){
        fdcb(fd, READ, NULL); // un-register callback
        parse_request(inBuf, serverName, file); // parse the HTTP request
        tcpconnect(serverName, 80,
                   wrap(connect_cb, fd, file));
        // resolve serverName and connect both are asynchronous
    } else {
        // do nothing; wait for more calls to req_cb()
    }
}
```
Event-driven Programming

connect_cb(int client_fd, str file, int server_fd)
// called when we have connected to the server
{
    // write the request when the socket is ready
    fdcb(server_fd, WRITE,
         wrap (write_cb, file, server_fd));
}
Interaction with multiprocessors

- N-copy approach
Multiprocessor Design

- Coordinating callbacks
- `libasync-smp` API
- Scheduling callbacks
Coordinating callbacks

- *libasync-smp* associates a *color* with each registered callback
- *Color is a 32-bit values*
- split a callback
Coordinating callbacks
libasync-smp API

- cwrap() function
- how to assign colors
  1. use file descriptor number
  2. use the address of data structure
- Cpucb() function
Example

req_cb(int fd, str inBuf) // called when data arrives
{
    read(fd, buf, ...);
    append input to inBuf;
    if(complete request in inBuf){ // un-register callback
        fdcb(fd, READ, NULL); // parse the HTTP request under color fd
        cpucb (cwrap (parse_request_cb, fd, inBuf, (color)fd))
    }
}
Example

else {
    // do nothing; wait for
    // more calls to req_cb()
}
}

parse_req_cb (int fd, str inBuf) // below parsing done w/ color fd
{
    parse_request (inBuf, serverName, file);
    // start connection to server
    tcpconnect (serverName, wrap(connect_cb, fd, file));
}
Scheduling callbacks

- Scheduling callbacks involves two operations:
  1. placing callbacks on a worker thread's queue.
  2. at each thread, deciding which callback to run next.

- A callback is placed on a thread's queue:
  1. call to cpucb()
  2. the *libasync-smp* main loop detected the arrival of an I/O, timer, or signal event for which a callback had been registered.
Scheduling callbacks

- If a worker thread's task queue is empty it attempts to steal work from another thread's queue.

- The scheduler considers:
  1. priority
  2. callback/thread affinity
Scheduling callbacks

- select callback
  calls select() to detect I/O events

The callback queue structure in *libasync-smp*
Implementation

- clone() system call (under Linux)
- rfork() (under FreeBSD)
- thr_create() (under Solaris)

- *libasync-smp* uses spin-locks internally to protect its own data structures
Evaluation

- HTTP server
- SFS server
- Library Optimizations
HTTP server
HTTP server

![Graph showing throughput (MB/s) vs number of CPUs for different server configurations]
HTTP server

![Graph showing throughput (MB/s) vs number of CPUs for different HTTP servers: Apache-MT, Apache-MP, libasync-smp, Flash-AMPED, and N-Copy.](image)
SFS server

![Graph showing throughput vs number of CPUs for different storage systems. The graph compares Libasync-smp and N-Copy.](image)
Table 2: A synthetic benchmark shows improved task processing rates as thread affinity optimizations are added.
Related Work

- AMPED: asymmetric multi-process event-driven
- SEDA: The staged, event-driven architecture