C++

session 10

#10

- Advanced Networking

Advanced Networking

Networking in C/C++ is pretty simple and within few lines of code for a server or a client, you can have 2 nodes communicating with each other.

The example from the previous session were showing how network is working in C and how to implement a trivial architecture. However, those are NOT good practices at all since we can’t have a server communicating with 2 clients at the same time.

Advanced Networking

During this session, we will see what are the differents methods used to implement a server that can deal with (in theory) an unlimited number of clients simultaneously.

Advanced Networking

Every time your write a program, you need to keep that I/O is a source of failure or dead-lock.

Files, Devices, Sockets all of them are sources of I/O and possible failure.

Advanced Networking

When you are working with I/O, you should **NEVER** assume that it will work, you should spend 99% of your time thinking when it is not. Doing this, you will end up having bullet-proof programs.

Advanced Networking

TCP and UDP use similar architectures but we will only focus on TCP since the project #2 and #3 are using it.

Buckle-up and here we start the real stuff.

Advanced Networking

- **synchronous**: you handle one request at a time, each in turn.
  - pros: simple
  - cons: anyone request can hold up all the other requests
- **fork**: you start a new process to handle each request.
  - pros: easy
  - cons: does not scale well, hundreds of connections means hundreds of processes.
  - *fork() is the Unix programmer's hammer. Because it's available, every problem looks like a nail. It's usually overkill*
- **threads**: start a new thread to handle each request.
  - pros: easy, and kinder to the kernel than using fork, since threads usually have much less overhead
  - cons: your machine may not have threads, and threaded programming can get very complicated very fast, with worries about controlling access to shared resources.

Advanced Networking

- Asynchronous:
  - Pros:
    - efficient and elegant
    - scales well - hundreds of connections means only hundreds of socket/state objects, not hundreds of threads or processes.
    - requires no interlocking for access to shared resources. If your database provides no interlocking of its own (as is the case for dbm, dbz, and berkeley db), than the need to serialize access to the database is provided trivially.
    - integrates easily with event-driven window-system programming. GUI programs that use blocking network calls are not very graceful.
  - Cons
    - more complex - you may need to build state machines.
    - requires a fundamentally different approach to programming that can be confusing at first.

Advanced Networking

- Fork
  For instance, Apache by default uses a blocking scheme where the process is forked for every connection. That means every connection needs its own memory space and the sheer amount of context-switching overhead increases more as the number of connections increases. But the benefit is, once a connection is closed the context can be disposed and any/all memory can be easily retrieved.

Advanced Networking

- Fork
  - pid_t fork(void);
  - man 2 fork

Advanced Networking

- Threads
  A multi-threaded approach would be similar in that the overhead of context switching increases with the number of connections but may be more memory efficient in a shared context. The problem with such an approach is it's difficult to manage shared memory in a manner that's safe. The approaches to overcome memory synchronization problems often include their own overhead, for instance locking may freeze the main thread on CPU-intensive loads, and using immutable types adds a lot of unnecessary copying of data.

Advanced Networking

- Threads
  - int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg);
  - man 3 pthread_create

Advanced Networking

- Asynchronous
  Non-blocking would be something like Node.js or nginx. These are especially known for scaling to a much larger number of connections per node under IO-intensive load. Basically, once people hit the upper limit of what thread/process-based servers could handle they started to explore alternative options. This is otherwise known as the C10K problem (ie the ability to handle 10,000 concurrent connections).

Advanced Networking

- Asynchronous
  - select / man 2 select
  - poll / man 2 poll
  - epoll / man 7 epoll
  - kqueue / man 2 kqueue *but BSD/Unix like only*