venkateshcm.com

Reactor Pattern Part 2 - Applications with Non-Blocking I/O

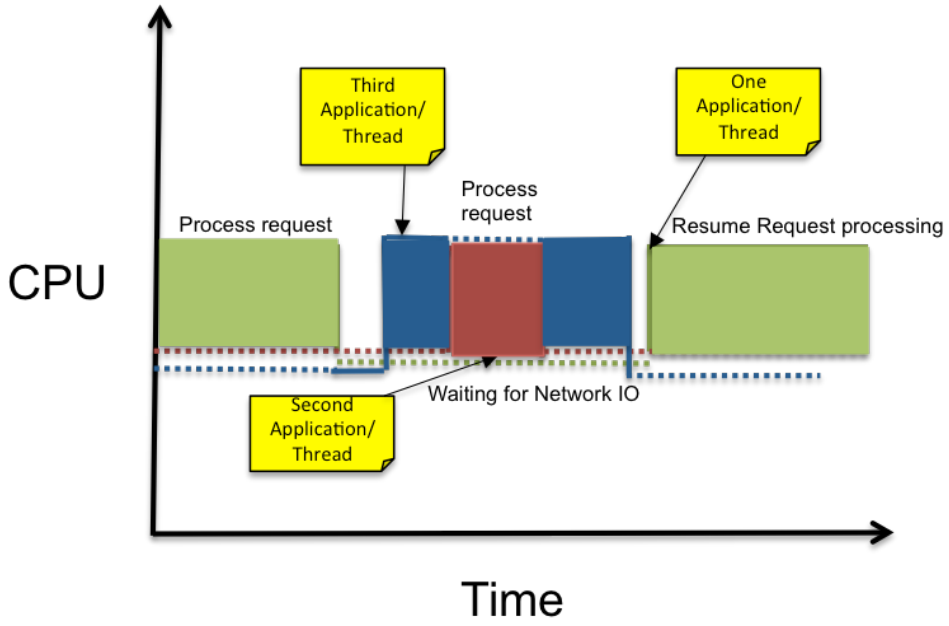
Venkatesh CM

3-4 minutes

In Reactor Pattern Part 1 : Applications with Blocking I/O, I went through issues faced by a single threaded application to scale to handle more requests pre box.

In this blog, we will look at an alternative solution to maximise CPU usage.

The diagram below from the part 1 blog we can notice that applications require CPU in busts and they have wait periods between the processing busts.



If we treat processing busts as events and queue events to be executed on a single thread, we can make sure the thread (CPU) is always occupied and thus giving us maximising CPU usage and avoiding unnecessary context switching.

Consider processing blocks as events without any I/O calls within them which can be executed non-sequentially/asynchronously. The code should be split into execution blocks (events) which can be executed separately with other requests events executed between them.

Event Loop

An event loop is a program loop where a thread waits for events and executes events that occur in a program. For more info on checkout Event Loop

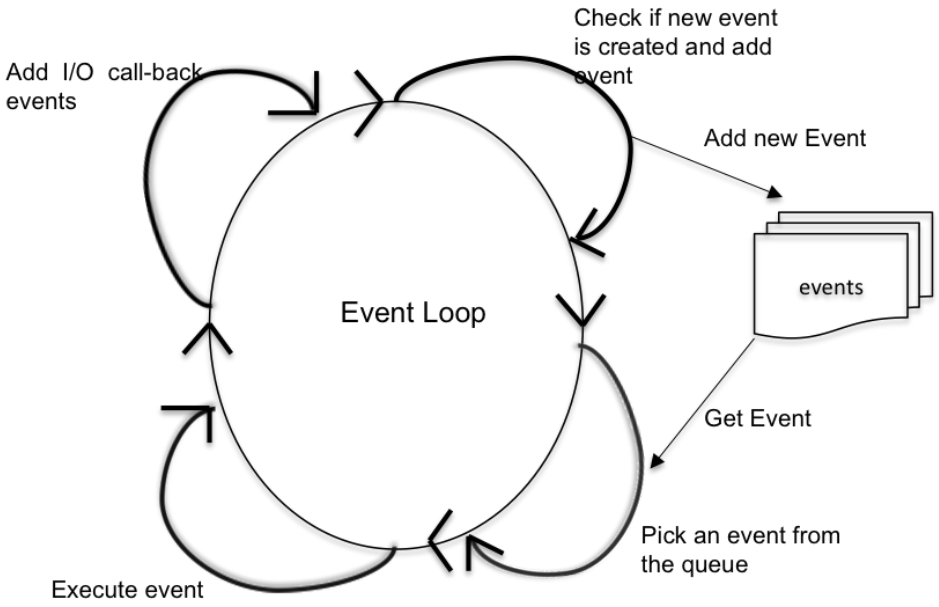
Unix File Descriptors

Following Unix principle of everything is a file, file descriptor is used to detect events when reading/writing to file, network communication, device communication and inter-process communication. System calls “epoll” and “pselect” are used to detect file descriptor state change without blocking application.

The below diagram shows a simplified event loop which goes through four stages in each loop.

- Check if a new event is created (i.e. a web request has come in or a call back event has occurred) and add new event is added to the queue.
- Pick an event from the queue
- Execute event

- Create a call-back event (i.e. database query or network access call-back when the response has arrived. Call-back events are handled by epoll or pselect.



  The solution described above is a simplified version of Reactor Pattern.

  Advantages of Reactor Pattern

- Optimal usage of CPU.
- Can handle more requests with same hardware.
- Can scale above C10K limit

  C10K Problem

  Historically, Reactor Pattern came into prominence after C10K problem and Solution to C10K problem using Non-Blocking I/O succeeded.

  Disadvantages of Reactor Pattern

- Asynchronous event based code base makes it difficult to understand and structure code. Promise library can help.

- Debugging code will be more difficult since stack trace begins from the call-back instead of start of request.

  Some of the disadvantages can be mitigated by using fiber (ruby fibers, node fibers), will cover Fibers in another blog.

  In part 3, we will look at Callback issue in more detail and how promises can help fix callback issues.