Concurrent Programming: *Concurrent Servers: 2. Event based*

Approaches for Writing Concurrent Servers

Allow server to handle multiple clients concurrently

1. Process-based

- Kernel automatically interleaves multiple logical flows
- Each flow has its own private address space

2. Event-based

- Programmer manually interleaves multiple logical flows
- All flows share the same address space
- Uses technique called *I/O multiplexing*.

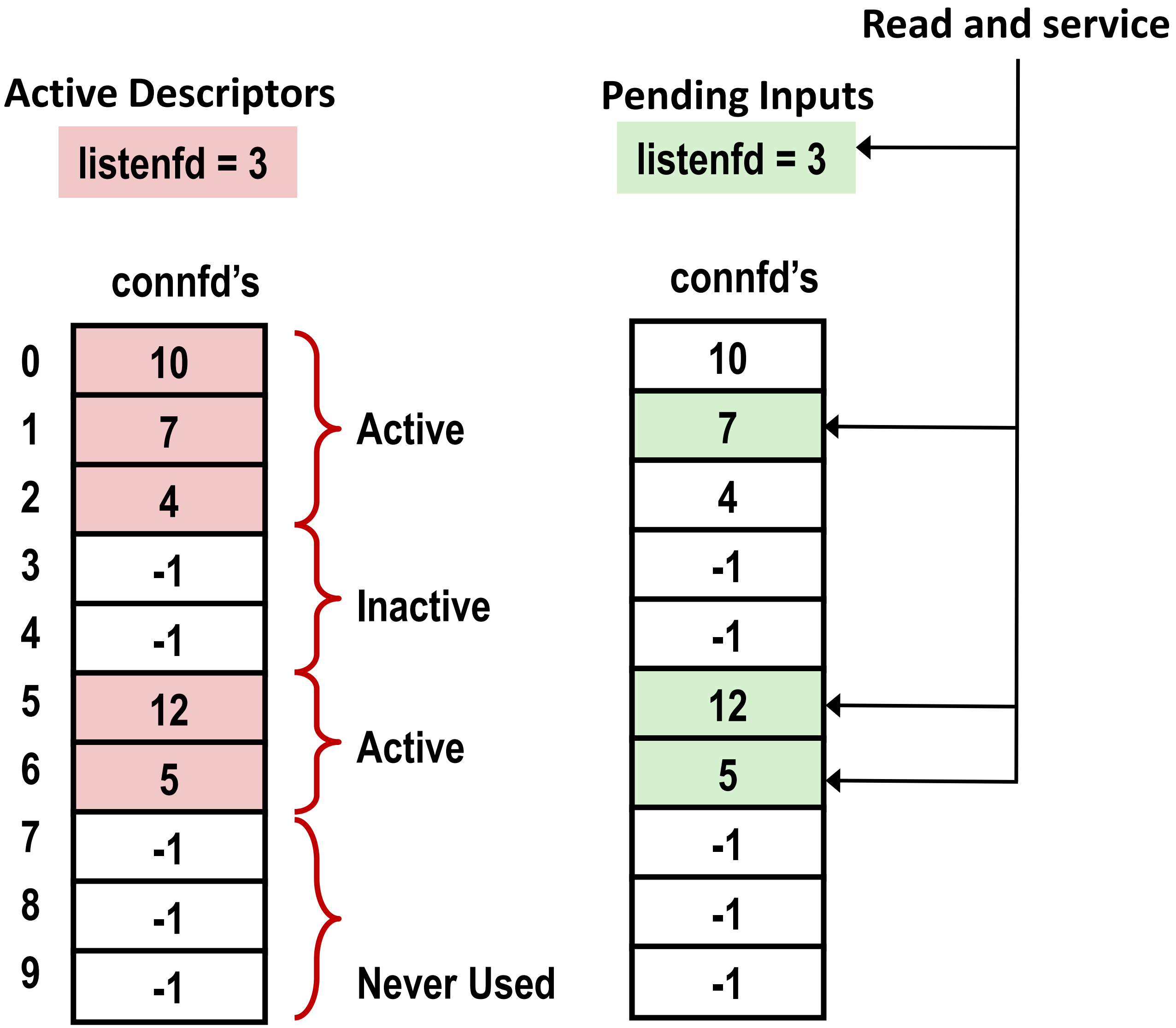
3. Thread-based

- Kernel automatically interleaves multiple logical flows
- Each flow shares the same address space
- Hybrid of of process-based and event-based.

Approach #2: Event-based Servers

- **Server maintains set of active connections**
  - Array of connfd’s
- **Repeat:**
  - Determine which descriptors (connfd’s or listenfd) have pending inputs
    - e.g., using `select, poll` or `epoll*` syscalls
    - arrival of pending input is an *event*
  - If listenfd has input, then `accept` connection
    - and add new connfd to array
  - Service all connfd’s with pending inputs
- **Details for select-based server in book**

I/O Multiplexed Event Processing



Read and service

Active Descriptors

listenfd = 3

Pending Inputs

listenfd = 3

	connfd's		connfd's
0	10	Active	10
1	7	Active	7
2	4	Active	4
3	-1	Inactive	-1
4	-1	Inactive	-1
5	12	Active	12
6	5	Active	5
7	-1	Never Used	-1
8	-1	Never Used	-1
9	-1	Never Used	-1

Pros and Cons of Event-based Servers

- **+ One logical control flow and address space.**
- **+ Can single-step with a debugger.**
- **+ No process or thread control overhead.**
  - Design of choice for high-performance Web servers, e.g., Node.js, nginx, some Apache MPM (C10k Problem)

- **– Significantly more complex to code than process- or thread-based designs.**
- **– Hard to provide fine-grained concurrency**
  - E.g., partial HTTP request headers (one line read per client?)
- **– Hard to take advantage of multi-core**
  - Single thread of control
  - I/O library implements thread support, e.g., libuv (Unicorn Velociraptor)
- **Going further: async I/O using `aio_read`, `aio_write(3)`**