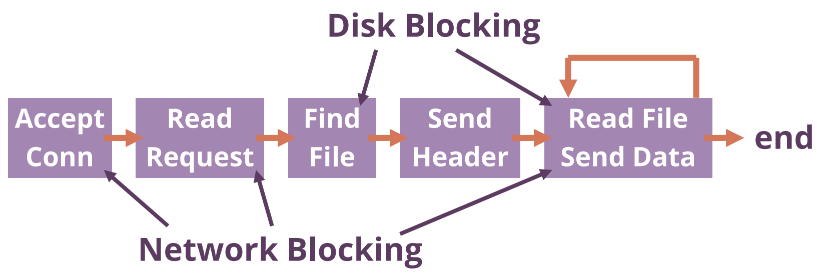
Multi-Threaded Network Programming

MCIT 595

Revisit: Blocking Steps in a Web Server

- `accept`: waiting for new connection;
- `read` a socket waiting for data or close;
- `write` a socket waiting for buffer space;
- I/O `read/write` for disk to finish



Writing High Performance Servers: Major Issues

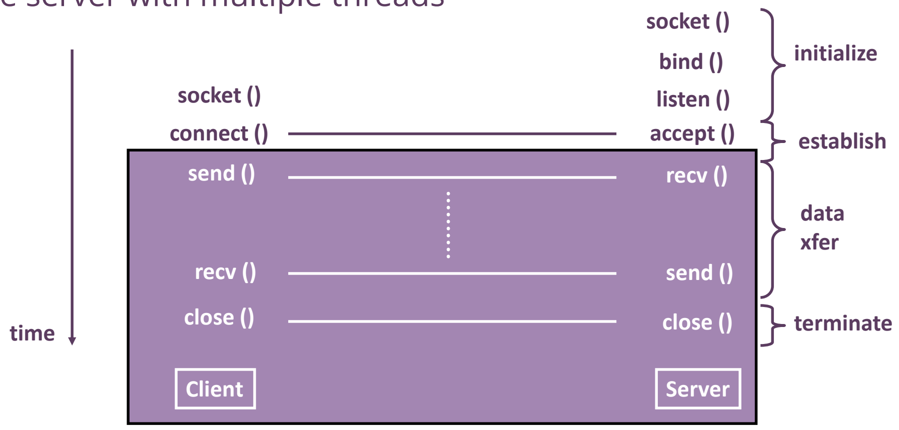
- Many socket/IO operations can cause a process to block:
  - `accept`: waiting for new connection;
  - `read` a socket waiting for data or close;
  - `write` a socket waiting for buffer space;
  - I/O `read/write` for disk to finish
- Thus a crucial perspective of server design is the concurrency design
  - For high performance
  - To avoid denial of service

Two Techniques for Handling Blocking System Calls

- **Solution I:** Multithreading
- **Solution II:** Event-Driven Asynchronous Programming

Revisit: Stream Sockets

- Multiple concurrent clients
- Single server with multiple threads



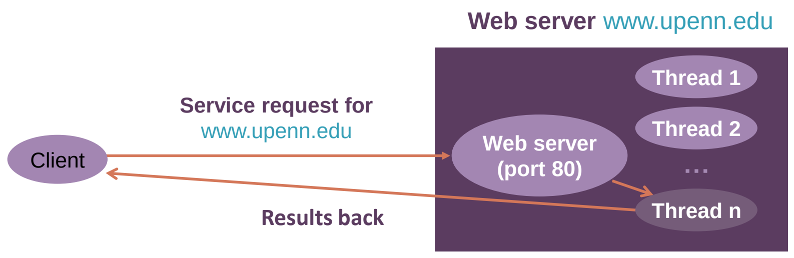
Server Code Snippet

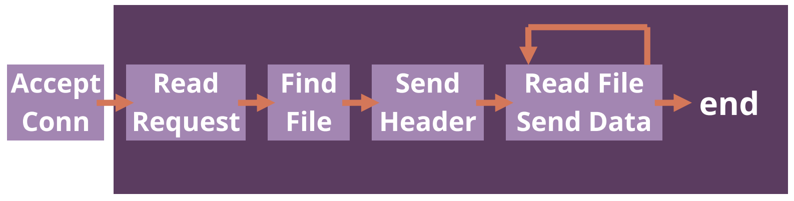
```
/*setup passive open*/
  if((s = socket(PF_INET, SOCK_STREAM, 0)) <0) {
    perror("simplex-talk: socket");
    exit(1);
  }
  if((bind(s, (struct sockaddr*)&sin, sizeof(sin)))<0) {
    perror("simplex-talk: bind");
    exit(1);
  }

  // connections can be pending if many concurrent client requests
  listen(s, MAX_PENDING);

  /* wait for connection, then receive and print text */
  while(1) {
    if((new_s = accept(s, (struct sockaddr *)&sin, &len)) <0){
      perror("simplex-talk: accept");
      exit(1);
    }
    while(len = recv(new_s, buf, sizeof(buf), 0)){
        fputs(buf, stdout);
    }
    close(new_s);
  }
}
```

To execute in a thread

Web Servers: Handling Concurrent Requests

- Using multiple threads so that only the flow processing a particular request is blocked





Executed within a thread

Problems of Multi-Threaded Servers

- High resource usage, context switch overhead, contended locks
- Too many threads → throughput meltdown, response time explosion
  - In practice, bound the number of threads (thread pool)
- Difficulty in reasoning about concurrency
  - Requires mutexes, semaphores, etc to manipulate shared state
  - Deadlocks
- Alternative based on event-driven programming