Máy chủ xử lý tuần tự và máy chủ xử lý đồng thời đa tiến trình

Giáo viên: Nguyễn Hoài Sơn
Bộ môn Mạng và Truyền thông máy tính
Trường Đại học Công nghệ

Nội dung bài học

- Khái niệm các loại máy chủ
- Máy chủ xứ lý tuần tự
- Máy chủ xử lý đồng thời, đa tiến trình, hướng kết nối
- Ví dụ về Máy chủ xử lý đồng thời, đa tiến trình, hướng kết nối

Đặt vấn đề

- Yêu cầu với máy chủ:
  - Đảm bảo chất lượng dịch vụ
    - Đáp ứng đồng thời các yêu cầu của nhiều máy khách cùng một lúc
    - Thời gian đáp ứng yêu cầu của máy khách là nhỏ
  - Chi phí thấp
    - Dung lượng bộ nhớ sử dụng, tài nguyên tính toán
  - Dễ thiết kế, xây dựng và bảo trì

Phân loại máy chủ theo cách thức cung cấp dịch vụ

	Tuần tự	Đồng thời
Không kết nối	Tuần tự không kết nối	Đồng thời không kết nối
Hướng kết nối	Tuần tự hướng kết nối	Đồng thời hướng kết nối

Máy chủ không kết nối

- Máy chủ sử dụng giao thức UDP
  - Không có kết nối giữa máy khách và máy chủ
- Ưu điểm
  - Chỉ cần một socket
  - Thích hợp với kiểu truyền tin quảng bá/quảng phát
  - Người lập trình được tự do lựa chọn cách thức thực thi tính tin cậy trong truyền tin
- Nhược điểm
  - Cung cấp tính tin cậy đòi hỏi chi phí cao với người lập trình

Máy chủ hướng kết nối

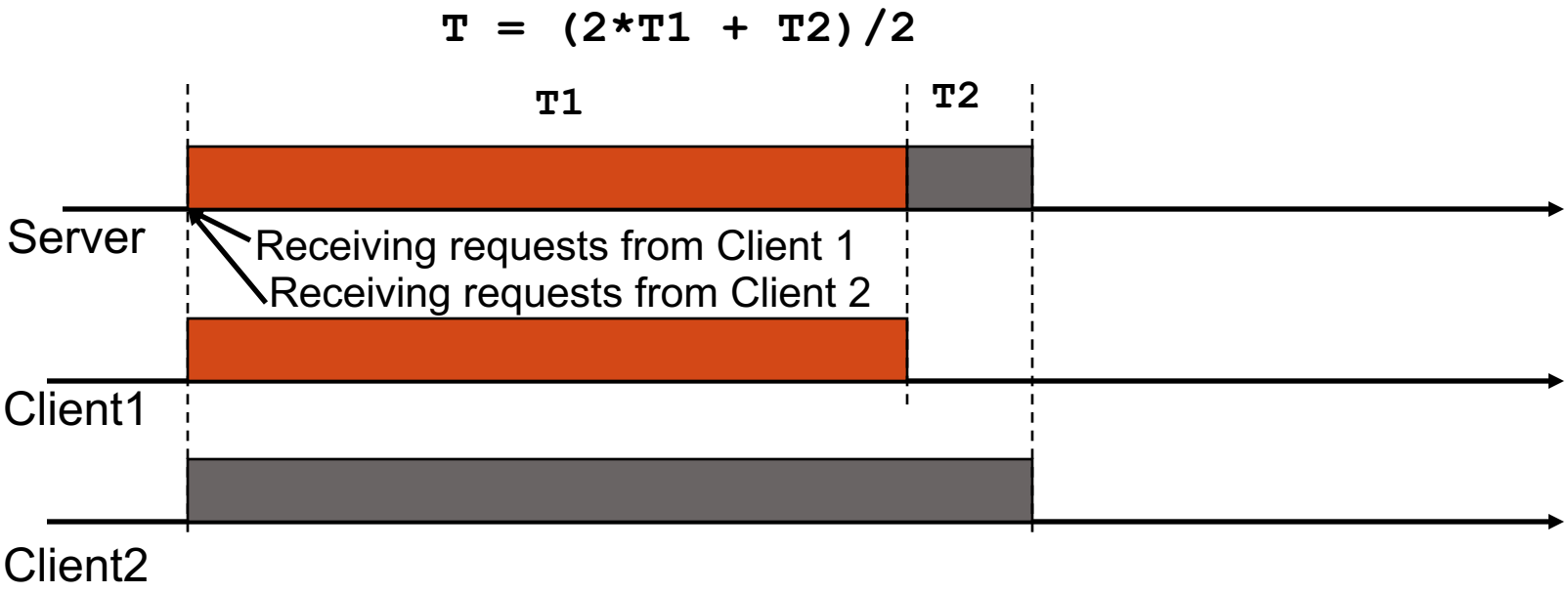
- Máy chủ sử dụng giao thức TCP
  - Sử dụng khái niệm kết nối trong truyền tin
    - Chấp nhận kết nối từ máy khách
    - Tất cả các thông tin được gửi qua kết nối giữa máy chủ và máy khách
    - Đóng kết nối sau khi kết thúc truyền tin
- Ưu điểm
  - Truyền tin tin cậy
- Nhược điểm
  - Chi phí cao: phải tạo socket riêng cho từng kết nối với máy khách

Khái niệm xử lý tuần tự/xử lý đồng thời

- Chương trình xử lý tuần tự
  - Một tiến trình/một luồng điều khiển việc xuất nhập dữ liệu
    - Xử lý tuần tự từng yêu cầu của mỗi máy khách
  - Dễ thực hiện
- Chương trình xử lý đồng thời
  - Nhiều tiến trình/luồng điều khiển việc xuất nhập và xử lý dữ liệu
  - Các tiến trình/luồng thực hiện đồng thời
  - Khó thực hiện

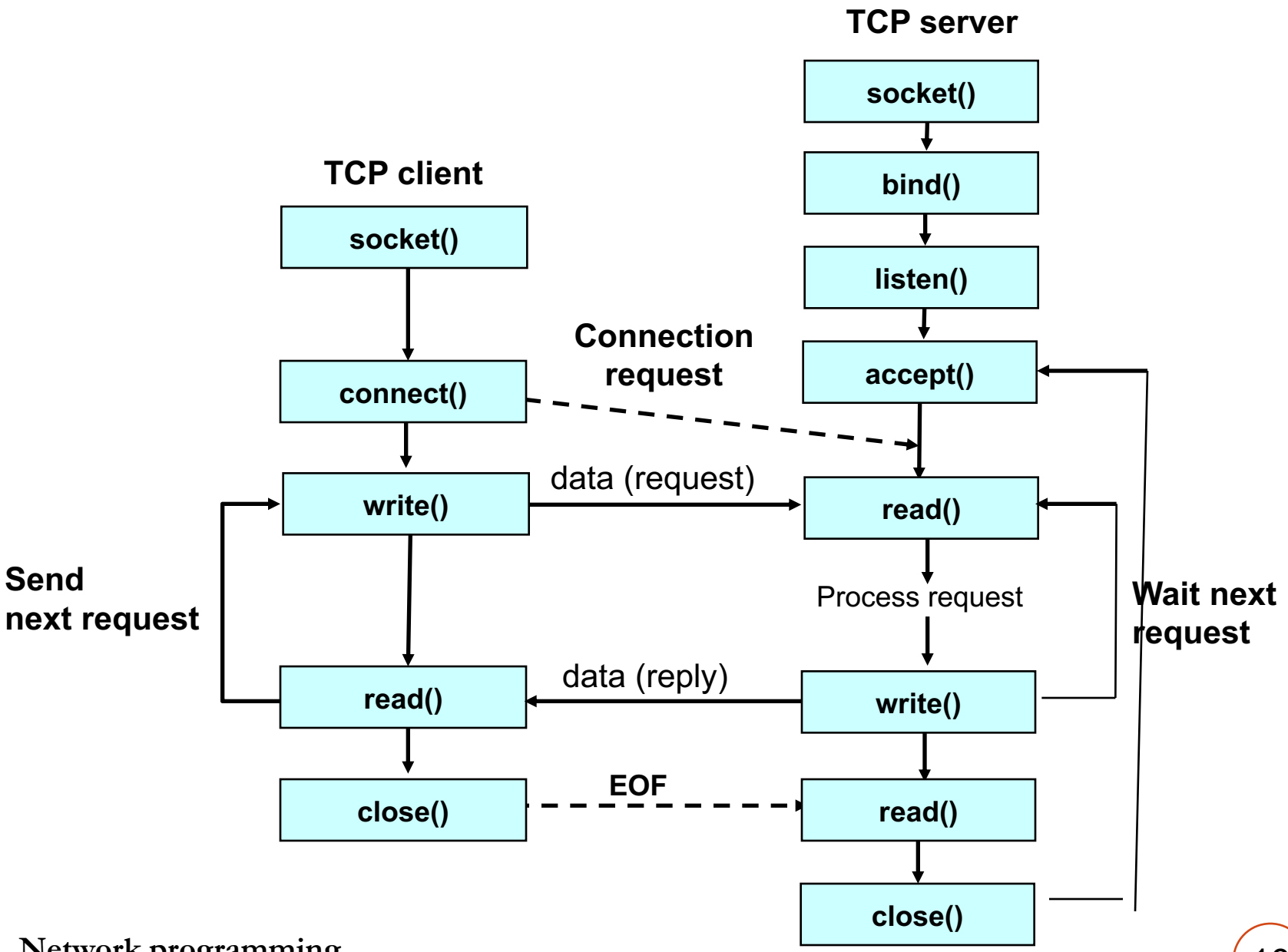
Máy chủ xử lý tuần tự

- Tại mỗi thời điểm, xử lý một yêu cầu
  - Đưa các yêu cầu mới vào một hàng đợi
- Yêu cầu của một máy khách chỉ được xử lý khi yêu cầu trước đó được xử lý xong
- Dễ thực thi nhưng kém hiệu quả
  - Nhất là trường hợp yêu cầu xử lý ít phải chờ yêu cầu xử lý nhiều kết thúc



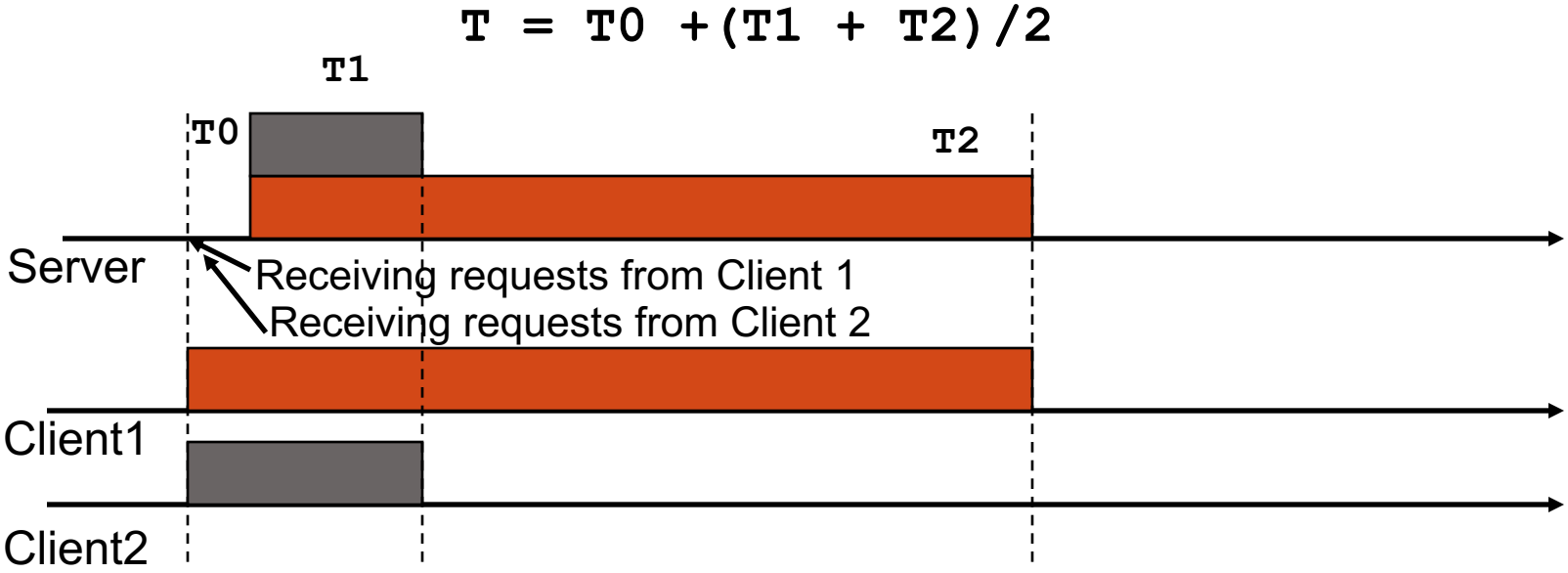
Cách thực hiện máy chủ xử lý tuần tự

- Một máy chủ xử lý tuần tự thực hiện theo các bước đơn giản sau
  - Tạo socket và gán thông tin cổng cho socket
  - Lặp lại
    - Chấp nhận yêu cầu từ máy khách
    - Xử lý yêu cầu
    - Tạo thông báo kết quả và gửi trả lại máy khách
- Trong trường hợp của máy chủ xử lý tuần tự hướng kết nối
  - Tại mỗi thời điểm, máy chủ xử lý từng yêu cầu đến từ một kết nối tới một máy khách


TCP server
socket()
bind()
listen()
accept()
TCP client
socket()
connect()
Connection request
write()
data (request)
read()
Send next request
Process request
Wait next request
read()
data (reply)
write()
close()
EOF
read()
close()

Máy chủ xử lý đồng thời

- Có thể xử lý nhiều yêu cầu đến cùng một lúc
- Máy khách không cần đợi yêu cầu trước đó kết thúc
- Hiệu quả cao hơn máy chủ xử lý tuần tự trong hầu hết các trường hợp nhưng khó thực hiện hơn



Cách thực thi máy chủ xử lý đồng thời

- Đa tiến trình
  - Tạo ra nhiều tiến trình
  - Mỗi tiến trình xử lý một yêu cầu của máy khách
- Đa luồng
  - Tạo ra nhiều luồng
  - Mỗi luồng xử lý một yêu cầu của máy khách
- Đa xuất nhập (I/O multiplex)
  - Đơn tiến trình
  - Xuất nhập dữ liệu tại nhiều socket cùng một lúc
  - Nhận dữ liệu đồng thời từ nhiều kết nối, xử lý tuần tự các yêu cầu

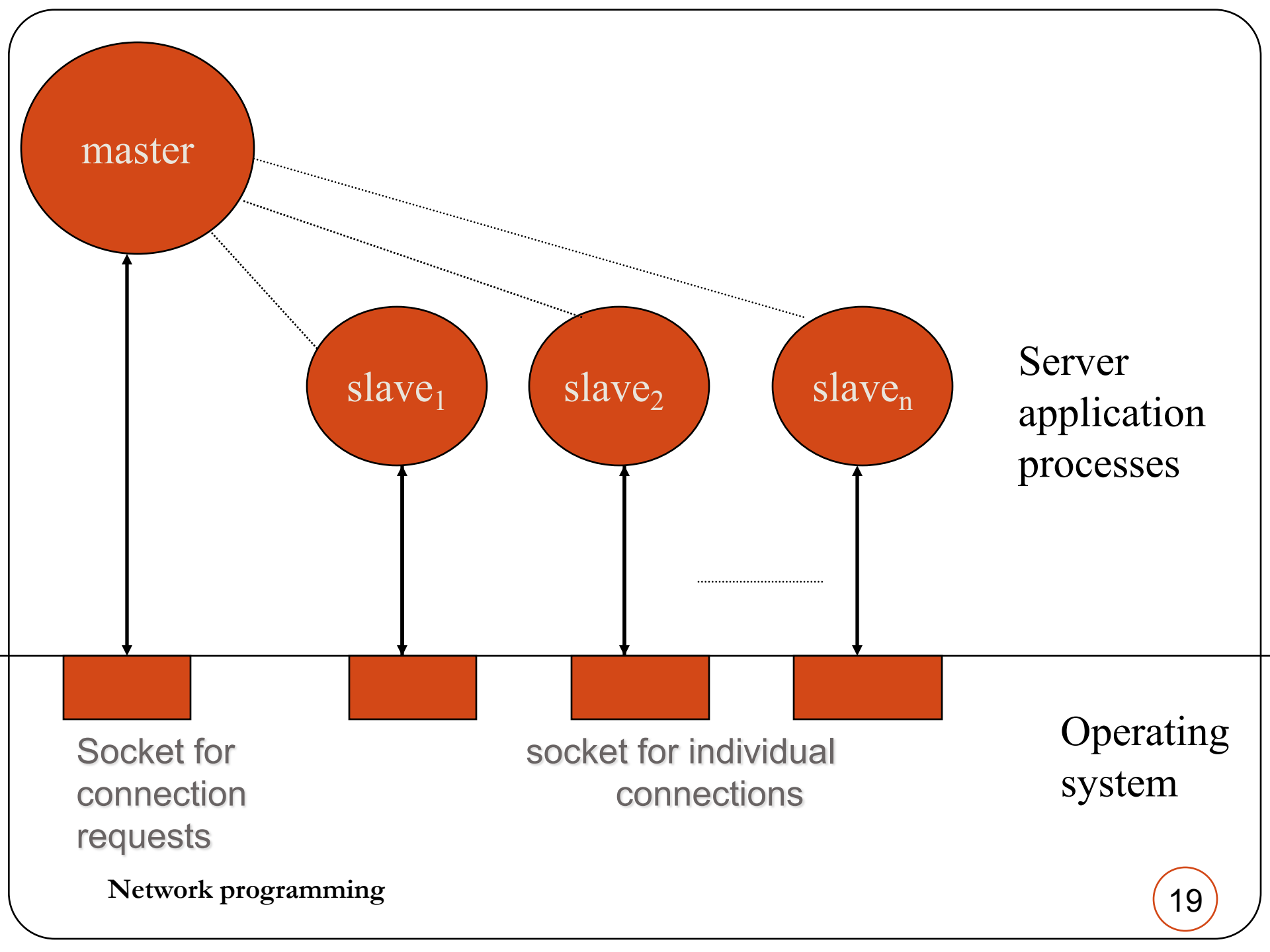
Xử lý đồng thời tốt trong trường hợp nào?

- Trả lời một yêu cầu cần nhiều thời gian xuất/nhập dữ liệu
  - E.g. File transfer, Telnet, Web server,…
- Thời gian xử lý các yêu cầu của máy khách là khác nhau lớn
- Máy chủ chạy trên một máy tính có nhiều bộ vi xử lý

Máy chủ xử lý đồng thời, hướng kết nối, đa tiến trình

Thực thi máy chủ xử lý đồng thời, hướng kết nối đa tiến trình như thế nào?

- Lệnh *accept* dừng thực thi của máy chủ để đợi kết nối đến cổng chờ
- Khi có kết nối từ máy khách, tiến trình mẹ sẽ khởi tạo một tiến trình con để xử lý kết nối
- Tiến trình mẹ sẽ chờ để chấp nhận một kết nối khác


master
slave1
slave2
slaven
Server application processes
Socket for connection requests
socket for individual connections
Operating system

Cách thực thi máy chủ xử lý đồng thời, hướng kết nối, đa tiến trình

- ***Bước 1 tiến trình mẹ:*** Khởi tạo socket, gán thông tin cho socket và chuyển socket sang trạng thái thụ động, chờ kết nối
- ***Bước 2 tiến trình mẹ:*** Lặp lại:
  - Gọi `accept()` chờ kết nối máy khách
  - Khi có kết nối từ máy khách, khởi tạo một tiến trình con để xử lý dữ liệu qua kết nối với máy khách

đồng thời, hướng kết nối, đa tiến trình (2)

- ***Bước 1 tiến trình con:*** Nhận socket kết nối với máy khách
- ***Bước 2 tiến trình con:*** Đọc yêu cầu của máy khách, xử lý yêu cầu và gửi thông báo trả lời cho máy khách
- ***Bước 3 tiến trình con:*** Đóng kết nối và thoát sau khi xử lý xong yêu cầu của máy khách

fork(): Tạo tiến trình mới

```
#include <unistd.h>

pid_t   fork(void);

        Returns: 0 in child, process ID of child in parent, -1 on error
```

- Khởi tạo một tiến trình con chỉ khác tiến trình mẹ ở PID và PPID
- Gọi một lần nhưng trả về 2 lần
  - Trả về trên tiến trình mẹ với giá trị trả về là PID của tiến trình con
  - Trả về trên tiến trình con, với giá trị trả về là 0
- Lấy PID của tiến trình mẹ bằng hàm getppid()

2 cách sử dụng điển hình với fork

- Một tiến trình muốn thực hiện nhiều công việc cùng một lúc
  - Tiến trình sẽ tạo ra một bản copy của nó. Mỗi bản copy sẽ thực hiện một công việc
- Một tiến trình muốn chạy một chương trình khác
  - Tiến trình sẽ tạo ra một bản copy của nó và bản copy đó sẽ chạy chương trình mới

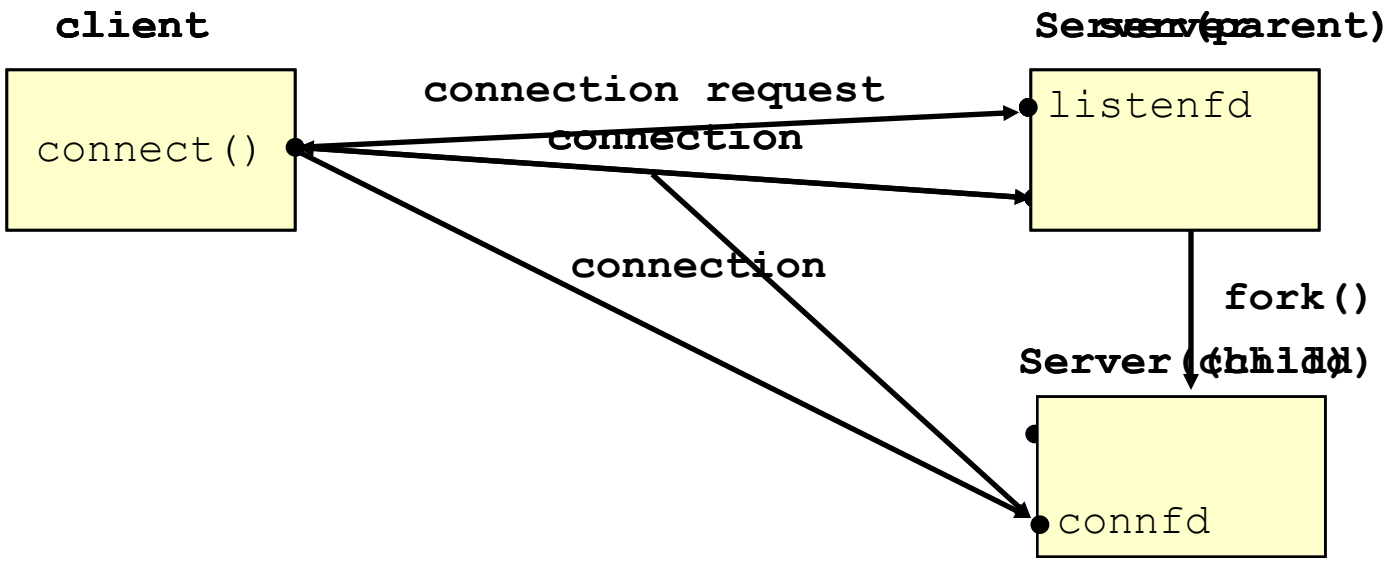
trình

- tcpserv02.c

Chi tiết về xử lý đồng bộ

- Tiến trình mẹ
  - đóng socket kết nối *connfd* của kết nối mới
  - sử dụng socket chờ *listenfd* để tiếp tục chờ kết nối mới
- Tiến trình con
  - đóng socket chờ *istenfd*
  - cung cấp dịch vụ *echo* (str_echo(connfd)) thông qua kết nối mới
    - tiết tục sử dụng socket kết nối *connfd* cho đến khi thoát ra (exit)
  - Tiến trình con phải thoát ra (exit) sau khi đã thực hiện xong dịch vụ
    - Lệnh *Exit* được sử dụng để kết thúc tiến trình
    - Khi tiến trình con kết thúc, hệ thống sẽ tự động đóng các socket kết nối lại

Xử lý đồng bộ với fork()



Vấn đề tiến trình không kết thúc hoàn toàn

- Một tín hiệu (SIGCHLD) sẽ được gửi đến tiến trình mẹ khi một tiến trình con kết thúc
- Tiến trình con sau khi kết thúc sẽ tồn tại ở trạng thái *zombie* cho đến khi tiến trình mẹ thực hiện lệnh gọi *wait3 (wait, waitpid)*
  - linux % ps -o pid,ppid,tty,stat,args,wchan

PID	PPID	TT	STAT	COMMAND	WCHAN
3674	3453	pts/3	S+	./daytime_server	-
3676	3674	pts/3	Z+	[daytime_server]<defunct>	exit

- Tiến trình *zombie* sẽ làm tốn tài nguyên máy tính
  - chiếm chỗ trên bộ nhớ

Giải quyết vấn đề tiến trình không kết thúc hoàn toàn như thế nào?

- Tiến trình mẹ sẽ bắt tín hiệu kết thúc của tiến trình con và gọi hàm xử lý tín hiệu

signal(SIGCHLD, sig_chld)

- Hàm *signal* biểu thị rằng tiến trình mẹ cần gọi hàm ***sig_chld*** mỗi khi nó nhận được tín hiệu SIGCHLD báo hiệu một tiến trình con đã kết thúc
- Hàm ***sig_chld*** gọi hàm *wait3* để hoàn thành việc kết thúc của tiến trình con

```
/* sig_chld - clean up zombie child */
void sig_chld(int signo){
   int status;
   while (wait3(&status, WNOHANG, (struct rusage *) 0) >=
   0)
         /* empty */;
}
```

- Giá trị *status* sau khi hàm wait3 trả về sẽ cho biết trạng thái kết thúc của tiến trình con
- Tùy chọn WNOHANG cho phép tiến trình mẹ không bị dừng thực thi nếu không có tiến trình con nào kết thúc

Cách viết khác của hàm *sig_chld*

```
/* reaper - clean up zombie child */
void sig_chld (int sig)
{
    pid_t  pid;
    int                 status
    while ((pid = waitpid (-1, &status, WNOHANG) )> 0)
            /* empty */;
    return;
}
```

- *waitpid* trả về giá trị là cấu trúc status để có thể kiểm tra thông tin về tiến trình con đã kết thúc
- Tùy chọn –1 để chỉ đợi tiến trình con thứ nhất

Xử lý cuộc gọi hệ thống bị ngắt như thế nào?

- Vấn đề:
  - Tín hiệu SIGCHLD được tạo ra khi tiến trình mẹ đang ngừng thực thi để chờ chấp nhận kết nối mới. Điều gì xảy ra khi hàm xử lý tín hiệu trả về giá trị?
  - Hàm accept sẽ trả về giá trị lỗi là EINTR (interrupted signal call). Tiến trình mẹ cần phải bỏ qua lỗi này.

```
If ((ssock=accept(….)) <0)
    if (errno = EINTR) continue;
    else err_sys("accept error");
```

Ví dụ về máy chủ ECHO xử lý đồng thời, đa tiến trình

- tcpserv03.c