CSc 462/562
Distributed Systems
Inter-process communication

Jianping Pan

(stand-in for Dr. Manning)

IPC [CDK3 Chap 4]

- Internet protocols API [CDK3 4.2]
  - UDP
  - TCP
- Client-server communication [CDK3 4.4]
  - request-reply (example: HTTP)
- Group communication [CDK3 4.5]
- IPC in Unix [CDK3 4.6]

Why IPC

- Inter-process communication
  - data: exchange information
  - control: synchronization

IPC via UDP

IPC via TCP

- Service offered by TCP
  - reliable, in-sequence, stream-like data transfer
- TCP protocol mechanisms
  - connection management
  - flow, error, congestion control
    - sequence number
    - acknowledgment

Client-server example: HTTP

- Server
  - listen on port 80 (and 443 for https)
- Client
  - (user click on *http://host.domain/path/file*)
  - resolve *host.domain* into 1.2.3.4 by DNS
  - use well-known port 80 for *http*
  - open connection to 1.2.3.4:80

HTTP (2)

- Server
  - accept the connection from the client
- Client
  - request object */path/file* (HTTP GET)
- Server
  - process the request
  - return the requested object (200 OK)
  - close connection

HTTP (3)

- Client
  - close connection
  - post-processing

Peer-to-peer

- Why peer-to-peer?
  - – server-client: server may become the bottleneck
  - – distributed server: e.g., CDN
  - – client is also a *server* for other clients
- Peer-to-peer systems
  - – unstructured: e.g., Napster, Gnutella, Skype
  - – structured: e.g., DHT-based

Multicast

- IP multicast
  - – IP multicast address (class D, 224-240.X.X.X)
    - Ethernet multicast address: 01-00-5e-0xxxxxxx-X-X
  - – IGMP: Internet Group Management Protocol
    - receiver-driven
  - – IP multicast routing
    - DVMRP, MOSPF
    - PIM-SM, PIM-DM
  - – M-Bone: multicast backbone (overlay)

Group communication

IPC in Unix

Socket API on Unix/Linux

- int socket(int *domain*, int *type*, int *protocol*);
  - – domain
    - PF_INET (Internet protocol family), and others
  - – type
    - SOCK_STREAM (supported by TCP)
    - SOCK_DGRAM (supported by UDP)
    - and others …
  - – protocol
    - normally implied by socket type

Choose a local identity

- int **bind** (int *sockfd*,
  struct sockaddr *my_addr*,
  socklen_t *addrlen*);
  - – struct sockaddr_in {short int *sin_family*;
    unsigned short int *sin_port*; //16-bit port#
    struct in_addr *sin_addr*; // 32-bit IP address
    unsigned char *sin_zero*[8];};
  - – struct in_addr {unsigned long *s_addr*;};

Wait for incoming connections

- Usually on the server side
- int **listen** (int *s*, int *backlog*);
  - backlog: maximal # of pending connections
- int **accept** (int *s*,
  struct sockaddr **addr*,
  socklen_t **addrlen*);
  - return a new and *accepted* socket fd

Make an outgoing connection

- Usually on the client side
- int **connect** (int *sockfd*,
  const struct sockaddr **serv_addr*,
  socklen_t *addrlen*);

Send data

- int **send** (int *s*,
  const void **msg*,
  size_t *len*,
  int *flags*);
  - int **sendto** (int *s*, const void **msg*, size_t *len*, int *flags*, const struct sockaddr **to*, socklen_t *tolen*);
  - int **sendmsg** (int *s*, const struct msghdr **msg*, int *flags*);
  - ssize_t **write** (int *fd*, const void **buf*, size_t *count*);

Receive data

- int **recv** (int *s*,
  void **buf*,
  size_t *len*,
  int *flags*);
  - int **recvfrom** (int *s*, void **buf*, size_t *len*, int *flags*, struct sockaddr **from*, socklen_t **fromlen*);
  - int **recvmsg** (int *s*, struct msghdr **msg*, int *flags*);
  - ssize_t **read** (int *fd*, void **buf*, size_t *count*);

Close the connection

- int **close** (int *fd*);
- int **shutdown** (int *s*, int *how*);
  - how=0: no receive
  - how=1: no send
  - how=2: no send and no receive

Client/server with Socket API

- Client
  - socket()
  - connect()
  - send()
  - recv()
  - close()
- Server
  - socket()
  - bind()
  - listen()
  - accept()
  - recv()
  - send()
  - close()

Socket API: port reuse

- int **setsockopt** (int *s*,
  int *level*,
  int *optname*,
  const void **optval*,
  socklen_t *optlen*);
  - level: SOL_SOCKET
  - optname: REUSEADDR

Socket API: non-blocking

- int **fcntl** (int *fd*, int *cmd*, long *arg*);
  - cmd: F_SETFL
  - arg: O_NONBLOCK
- int **select** (int n, fd_set **readfds*, fd_set **writefds*, fd_set **exceptfds*, struct timeval **timeout*);
  - FD_SET(int *fd*, fd_set **set*);
  - FD_ISSET(int *fd*, fd_set **set*);