Socket Programming - Concurrent Servers

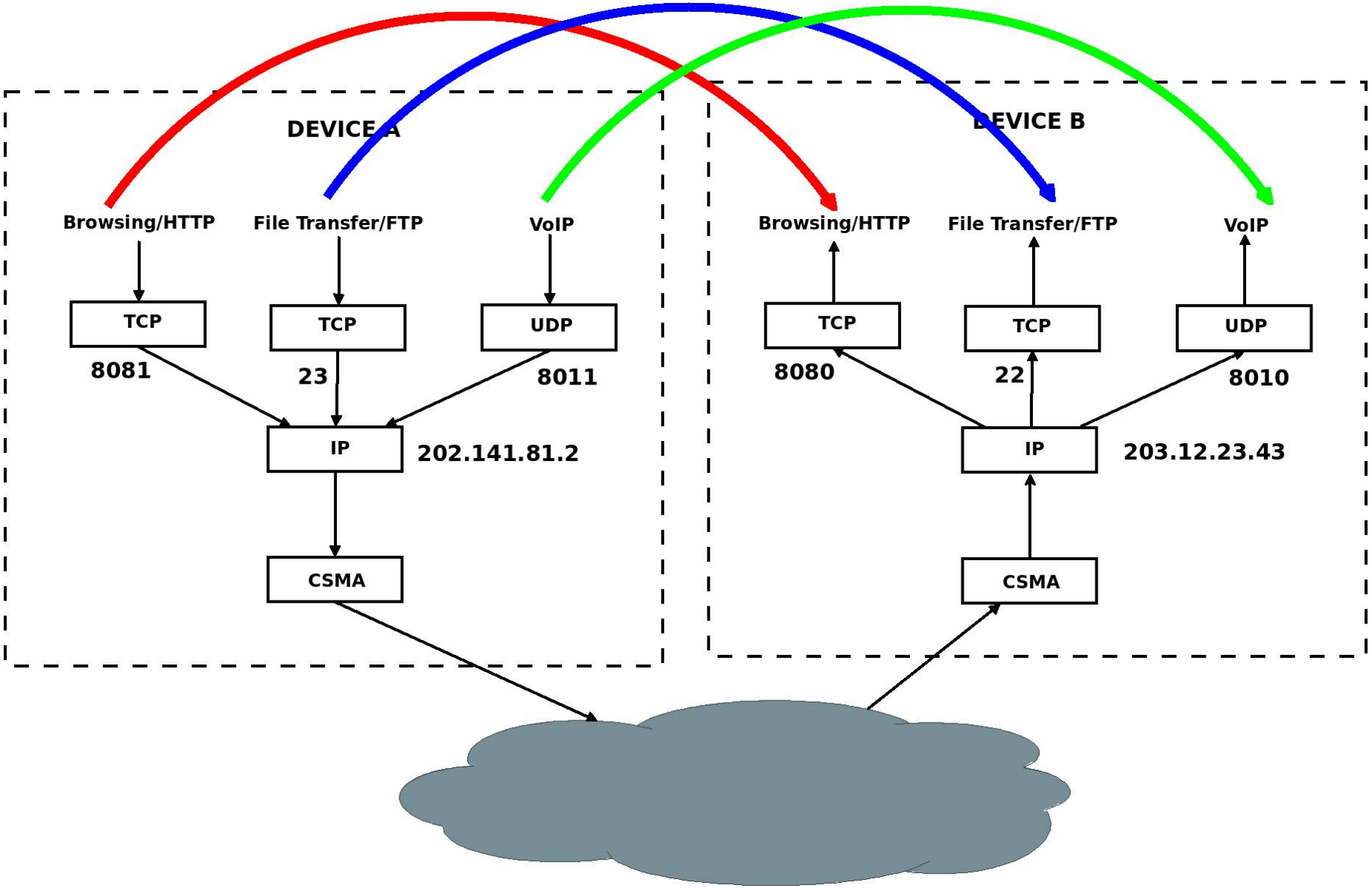
Sandip Chakraborty, K S Rao

Department of Computer Science and Engineering,

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

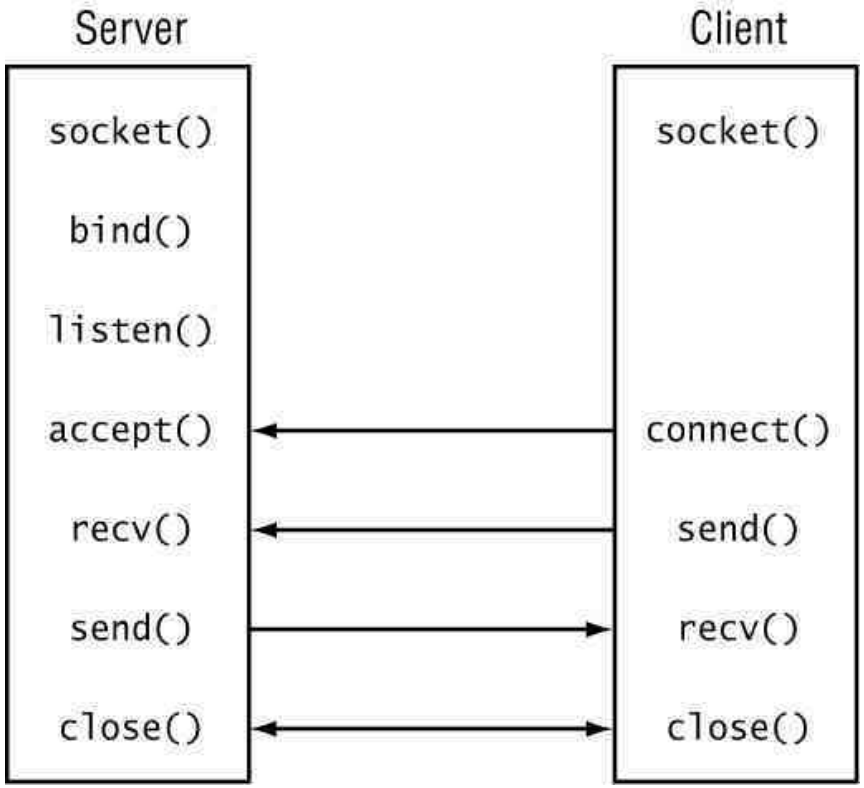
February 1, 2018

What are Sockets?

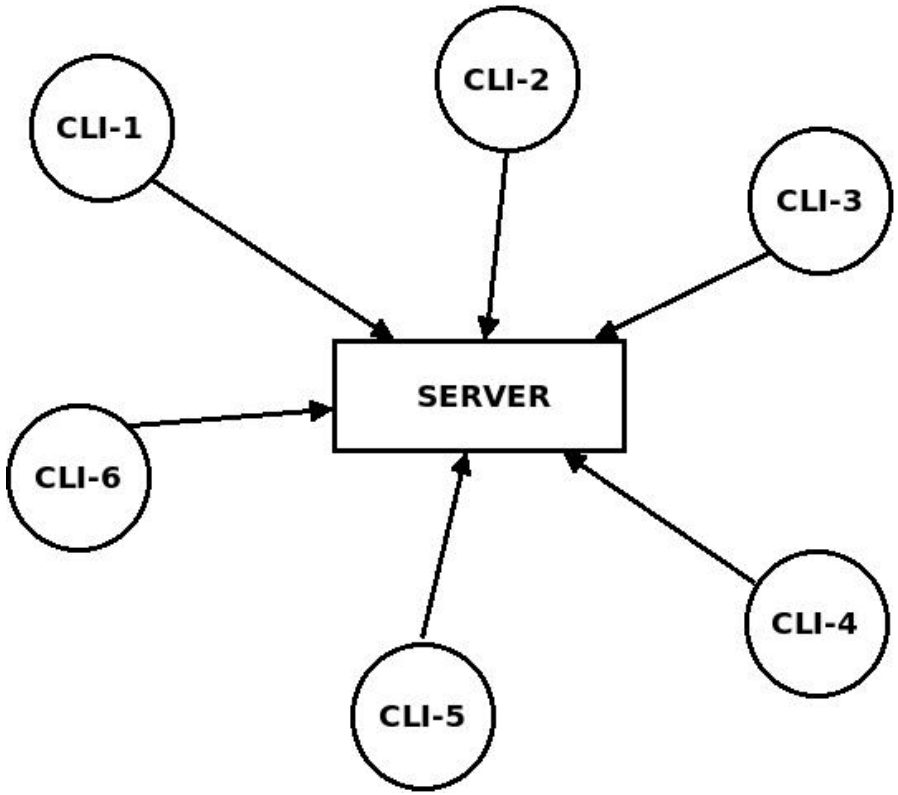


Socket Programming Framework/API

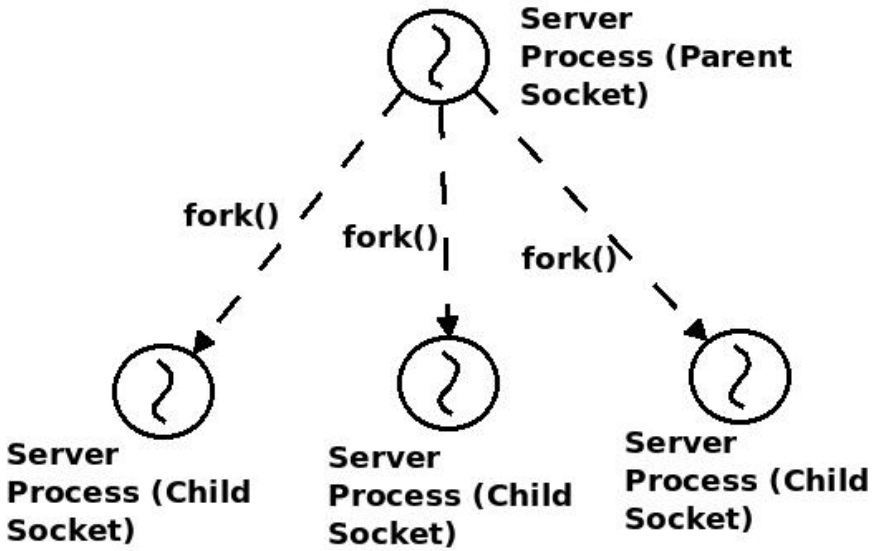
A set of **system calls** to get the service from TCP/IP protocol stack (`net` module in the OS kernel).



Concurrent Servers



Extending the Server Socket for Multiple Connections



Iterative Server

```
/*
 * listen: make this socket ready to accept connection requests
 */
if (listen(parentfd, 5) < 0) /* allow 5 requests to queue up */
  error("ERROR on listen");

/*
 * main loop: wait for a connection request, echo input line,
 * then close connection.
 */
clientlen = sizeof(clientaddr);
while (1) {

  /*
   * accept: wait for a connection request
   */
  childfd = accept(parentfd, (struct sockaddr *) &clientaddr, &clientlen);
  if (childfd < 0)
    error("ERROR on accept");
```

How Iterative Server Works

- The `listen()` call sets a flag that the socket is in listening state and set the maximum number of backlog connections.
- The `accept()` call blocks a listening socket until a new connection comes in the connection queue and it is accepted.
- Once the new connection is accepted, a new socket file descriptor (say `connfd`) is returned, which is used to read and write data to the connected socket.
- All other connections, which come in this duration, are backlogged in the connection queue.
- Once the handling of the current connected socket is done, the next `accept()` call accepts the next incoming connection from the connection queue (if any), or blocks the listening socket until the next connection comes.

Extending Iterative Server to Concurrent Server

- Parallel processing of each incoming sockets, so that the `accept()` call is executed more frequently.

```
newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr,
                           &clilen) ;

if (newsockfd < 0) {
        printf("Accept error\n");
        exit(0);
}

/* Having successfully accepted a client connection, the
   server now forks. The parent closes the new socket
   descriptor and loops back to accept the next connection.
*/
if (fork() == 0) {

        /* This child process will now communicate with the
           client through the send() and recv() system calls.
        */
        close(sockfd);  /* Close the old socket since all
                           communications will be through
                           the new socket.
                        */

        /* We initialize the buffer, copy the message to it,
           and send the message to the client.
        */

        strcpy(buf,"Message from server");
        send(newsockfd, buf, strlen(buf) + 1, 0);

        /* We again initialize the buffer, and receive a
           message from the client.
        */
```

Thank You