Socket Programming

Outline

L2.1 Motivation and overview
L2.2 Socket programming stages
L2.3 Socket programming examples
L2.4 Socket programming assignment
Socket Programming

Motivation and Overview

L2.1 Motivation and overview
L2.2 Socket programming stages
L2.3 Socket programming examples
L2.4 Socket programming assignment

Motivation and Overview

Socket Programming and Applications

• Also called client/server application development
• Introduced in 4.1 BSD in 1982
• Network application implementations
  - standard network application
    • based on RFCs
    • programs conforms rules
    • port numbers should be implemented per RFC
  - proprietary network application
    • they don’t conform RFCs
    • code will not interoperate
    • don’t use standard RFC well-known port numbers (0-1023)
Motivation and Overview

E2E Application Data Flow and Sockets

- Application process send messages to TL via sockets
- Application process is controlled by the developer
- TL (TCP, UDP) is controlled by the OS

Motivation and Overview

Socket Properties

- Socket
  - an interface which application processes can send & receive
  - messages to/from another application process
  - host local
  - created by application
  - controlled by the OS

- Determined by triple <domain, type, protocol>
  - domain - specifies address family (e.g. AF_INET)
  - type - comm. style (e.g. SOCK_STREAM, SOCK_DGRAM)
  - protocol - 0 provides default protocol for AF & socket type
Motivation and Overview

**Sockets and Processes**

- Socket is a method for Inter Process Communication
- Processes are created via client/server programs
- IPC can be done on a single host as well

---

**Socket Programming**

**Socket Programming Stages**

L2.1 Motivation and overview
L2.2 Socket programming stages
L2.3 Socket programming examples
L2.4 Socket programming assignment
1. Developer decides programming language and OS
   - Python, C, Java etc. and UNIX, Linux, MS etc.

2. Developer should decide:
   - to run the application on TCP
     - TCP is connection oriented, reliable byte stream channel
   - to run the application on UDP
     - UDP is connectionless service, best effort, no guarantee
   - skip transport layer to run the application
     - for hop nodes: e.g. ICMP
     - also called raw sockets

3. Developer implements the code

Socket Programming with TCP

- In order to establish connection between client & server
- Server process
  - has to be ready to respond client's requests
  - server has to have a welcoming socket
- Client process
  - creates socket
  - specifies the destination
  - 3-way handshake occurs
    - client invokes server's welcomeSocket accept() method
    - server responds this by creating dedicated connectionSocket
    - TCP establishes pipe between connectionSocket-clientSocket
Socket Programming Stages

**Connection-Oriented Flow Diagram**

- socket()
- bind()
- listen()
- accept()
- wait for connection
- connection establishment
- read()
- process request
- write()

**Socket Programming with UDP**

- Connectionless transport between client & server
  - there is no initial handshaking
  - unlike TCP client can be started first
- Client attaches destination address to each packet
- Client process
  - creates clientSocket of type DatagramSocket
  - in TCP clientSocket is of type Socket
- Server process
  - creates serverSocket of type DatagramSocket
  - there is no welcomeSocket as in TCP

[Stevens-1990]
Socket Programming Stages

Connectionless Flow Diagram

- socket()
- bind()
- recvfrom()

wait for client data

process request

sendto()

server

data (request)

recvfrom()

client

data (reply)

sendto()

[Stevens-1990]

Socket Programming Examples

L2.1 Motivation and overview
L2.2 Socket programming stages
L2.3 Socket programming examples
L2.4 Socket programming assignment
Socket Programming Examples

TCP Client 1

```c
void main(int argc, char *argv[])
{
struct sockaddr_in sad; /* structure to hold server’s IP address */
int clientSocket; /* socket descriptor */
struct hostent *ptrh; /* pointer to a host table entry */
char Sentence[128];
char modifiedSentence[128];

host = argv[1]; port = atoi(argv[2]);
clientSocket = socket(PF_INET, SOCK_STREAM, 0); /* create client socket */
memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
sad.sin_family = AF_INET; /* set family to Internet */
sad.sin_port = htons((u_short)port); /* convert port number to network byte order */
ptrh = gethostbyname(host); /* get host information */
memcpy(&sad.sin_addr, ptrh->h_addr, ptrh->h_length); /* connect socket to specified server */
connect(clientSocket, (struct sockaddr *)&sad, sizeof(sad));

/* get input stream from the user */
gets(Sentence);

/* send sentence to server */
n=write(clientSocket, Sentence, strlen(Sentence)+1);

/* read line from the server and write the modified sentence to screen */
n=read(clientSocket, modifiedSentence, sizeof(modifiedSentence));
printf("FROM SERVER: %s\n",modifiedSentence);

/* close connection */
close(clientSocket);
}
```

TCP Client 2

```c
/* get input stream from the user */
gets(Sentence);

/* send sentence to server */
n=write(clientSocket, Sentence, strlen(Sentence)+1);

/* read line from the server and write the modified sentence to screen */
n=read(clientSocket, modifiedSentence, sizeof(modifiedSentence));
printf("FROM SERVER: %s\n",modifiedSentence);

/* close connection */
close(clientSocket);
```
Socket Programming Examples

TCP Server1

```c
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h> 
#include <sys/socket.h>
#include <netinet/in.h> 
#include <unistd.h>

int main(int argc, char *argv[]) {

  struct sockaddr_in sad; /* structure to hold server’s IP address */
  struct sockaddr_in cad; /* structure to hold client’s IP address */
  int welcomeSocket, connectionSocket; /* socket descriptor */
  struct hostent *ptrh; /* pointer to a host table entry */
  char clientSentence[128];
  char capitalizedSentence[128];
  port = atoi(argv[1]); /* create welcoming socket at port and bind a local address */
  welcomeSocket = socket(PF_INET, SOCK_STREAM, 0);
  memset((char *) &sad, 0, sizeof(sad)); /* clear sockaddr structure */
  sad.sin_family = AF_INET; /* set family to Internet */
  sad.sin_addr.s_addr = INADDR_ANY; /* set the local IP address */
  sad.sin_port = htons((u_short) port); /* set the port number */
  bind(welcomeSocket, (struct sockaddr *)&sad, sizeof(sad));

  listen(welcomeSocket, 10) /* Main server loop – accept and handle requests */
  while(1) {

    /* Wait on welcoming socket for contact by client */
    connectionSocket = accept(welcomeSocket, (struct sockaddr *)&cad, &alen);
    n = read(connectionSocket, clientSentence, sizeof(clientSentence));
    /* capitalize Sentence and store the result in capitalizedSentence */
    /* write out the result to socket */
    n = write(connectionSocket, capitalizedSentence, strlen(capitalizedSentence)+1);
    close(connectionSocket);
  }
}
```

TCP Server2

```c
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h> 
#include <sys/socket.h>
#include <netinet/in.h> 
#include <unistd.h>

int main(int argc, char *argv[]) {

  struct sockaddr_in sad; /* structure to hold server’s IP address */
  struct sockaddr_in cad; /* structure to hold client’s IP address */
  int welcomeSocket, connectionSocket; /* socket descriptor */
  struct hostent *ptrh; /* pointer to a host table entry */
  char clientSentence[128];
  char capitalizedSentence[128];
  port = atoi(argv[1]); /* create welcoming socket at port and bind a local address */
  welcomeSocket = socket(PF_INET, SOCK_STREAM, 0);
  memset((char *) &sad, 0, sizeof(sad)); /* clear sockaddr structure */
  sad.sin_family = AF_INET; /* set family to Internet */
  sad.sin_addr_s_addr = INADDR_ANY; /* set the local IP address */
  sad.sin_port = htons((u_short) port); /* set the port number */
  bind(welcomeSocket, (struct sockaddr *)&sad, sizeof(sad));

  listen(welcomeSocket, 10) /* Main server loop – accept and handle requests */
  while(1) {

    /* Wait on welcoming socket for contact by client */
    connectionSocket = accept(welcomeSocket, (struct sockaddr *)&cad, &alen);
    n = read(connectionSocket, clientSentence, sizeof(clientSentence));
    /* capitalize Sentence and store the result in capitalizedSentence */
    /* write out the result to socket */
    n = write(connectionSocket, capitalizedSentence, strlen(capitalizedSentence)+1);
    close(connectionSocket);
  }
}
```
void main(int argc, char *argv[]) {
    struct sockaddr_in sad; /* structure to hold an IP address */
    int clientSocket; /* socket descriptor */
    struct hostent *ptrh; /* pointer to a host table entry */
    char Sentence[128];
    char modifiedSentence[128];
    host = argv[1]; port = atoi(argv[2]);
    /* Create client socket, NO connection to server */
    clientSocket = socket(PF_INET, SOCK_DGRAM, 0);
    /* determine the server’s address */
    memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
    sad.sin_family = AF_INET; /* set family to Internet */
    sad.sin_port = htons((u_short)port);
    ptrh = gethostbyname(host); /* Convert host name to IP address */
    memcpy(&sad.sin_addr, ptrh->h_addr, ptrh->h_length);
    /* get input stream from user */
    gets(Sentence);
    /* send line to server */
    addr_len = sizeof(struct sockaddr);
    n=sendto(clientSocket, Sentence, strlen(Sentence)+1,
        (struct sockaddr *)&sad, addr_len);
    /* read line from server */
    n=recvfrom(clientSocket, modifiedSentence, sizeof(modifiedSentence),
        (struct sockaddr *)&sad, &addr_len);
    printf("FROM SERVER: %s\n",modifiedSentence);
    /* close connection */
    close(clientSocket);
}
void main(int argc, char *argv[]) {

    struct sockaddr_in sad; /* structure to hold server’s IP address */
    struct sockaddr_in cad; /* structure to hold client’s IP address */
    int serverSocket; /* socket descriptor */
    struct hostent *ptrh; /* pointer to a host table entry */

    char clientSentence[128];
    char capitalizedSentence[128];

    port = atoi(argv[1]);

    /* create welcoming socket at port & bind a local address */
    serverSocket = socket(PF_INET, SOCK_DGRAM, 0);
    memset((char *)&sad, 0, sizeof(sad)); /* clear sockaddr structure */
    sad.sin_family = AF_INET; /* set family to Internet */
    sad.sin_addr.s_addr = INADDR_ANY; /* set the local IP address */
    sad.sin_port = htons((u_short)port); /* set the port number */
    bind(serverSocket, (struct sockaddr *)&sad, sizeof(sad));

    while(1) {

        /* receive messages from clients */
        n=recvfrom(serverSocket, clientSentence, sizeof(clientSentence), 0,
                   (struct sockaddr *)&cad, &addr_len);

        /* capitalize Sentence and store the result in capitalizedSentence */
        for(i=0; i <= strlen(clientSentence); i++) {
            if((clientSentence[i] >= 'a') && (clientSentence[i] <= 'z'))
                capitalizedSentence[i] = clientSentence[i] + ('A' - 'a');
            else
                capitalizedSentence[i] = clientSentence[i];
        }

        /* write out the result to socket */
        n=sendto(connectionSocket, capitalizedSentence, strlen(capitalizedSentence)+1, 0,
                  (struct sockaddr *)&cad, &addr_len);

        close(connectionSocket);
    } /* end of while loop, loop back and wait for another connection client*/
} /* main server loop - accept and handle requests */
Socket Programming Examples

**Python TCP Client**

```python
import socket
TCP_IP = '127.0.0.1'
TCP_PORT = 5005
BUFFER_SIZE = 1024
MESSAGE = "Hello, World!"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))
s.send(MESSAGE)
data = s.recv(BUFFER_SIZE)
s.close()
print "received data:\n", data
```

Socket Programming Examples

**Python TCP Server**

```python
import socket
TCP_IP = '127.0.0.1'
TCP_PORT = 5005
BUFFER_SIZE = 1024

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)
conn, addr = s.accept()
print 'Connection address:', addr
while 1:
    data = conn.recv(BUFFER_SIZE)
    if not data:
        break
    print "received data:\n", data
    conn.send(data)  # echo
c conn.close()
```

05 March 2013  KU EECS 780 – Comm Nets – Socket Prog. Lab  NET-L2-23

05 March 2013  KU EECS 780 – Comm Nets – Socket Prog. Lab  NET-L2-24
Socket Programming Examples

Python UDP Client

```python
import socket
UDP_IP = "127.0.0.1"
UDP_PORT = 5005
MESSAGE = "Hello, World!"

print "UDP target IP:" , UDP_IP
print "UDP target port:" , UDP_PORT
print "message:" , MESSAGE

sock = socket.socket(socket.AF_INET, # Internet
                     socket.SOCK_DGRAM) # UDP
sock.sendto(MESSAGE, (UDP_IP, UDP_PORT))
```

Socket Programming Examples

Python UDP Server

```python
import socket
UDP_IP = "127.0.0.1"
UDP_PORT = 5005

sock = socket.socket(socket.AF_INET, # Internet
                     socket.SOCK_DGRAM) # UDP
sock.bind((UDP_IP, UDP_PORT))

while True:
    data, addr = sock.recvfrom(1024) # buffer size is 1024 bytes
    print "received message:" , data
```
Socket Programming
Lab Report Submission Requirement

L2.1 Motivation and overview
L2.2 Socket programming stages
L2.3 Socket programming examples
L2.4 Socket programming assignment

Socket Programming Assignment
Program Output Requirement

• Create an application that will
  - convert uppercase letters to lowercase
    • e.g. [A...Z] to [a...z]
  - the code will not change any special characters
    • e.g. &*!
  - if the character is in lowercase, the program will not alter it

• Create socket API both for
  - reliable-byte stream
  - datagram services
Socket Programming Assignment

Questions to Answer

• What are the example applications that use TCP and UDP? Give two example for each protocol
• What are the port numbers those applications use?
  – use examples from well-known port number range
• Can you use those port numbers in the application you developed? Why or why not? Explain.
• In the application you developed how many simultaneous client/server connections are possible for each transport protocol? Are the port numbers same for a given protocol for each connection?
  – support you answer with netstat command output

Extra Credit

• For client/server programs, print local and foreign address using functions
  – getsockname()
  – getpeername()
• For client/server programs, can you use telnet as a client? If so, what is the exact syntax?
• What is the largest datagram size can you send and receive using UDP socket? Verify it experimentally
Socket Programming Assignment

Submission Requirements

• Follow instructions on submission requirements page
  - http://www.ittc.ku.edu/~jpgs/courses/homework.html
• You need to submit:
  - client file for reliable byte-stream service
  - server file for reliable byte-stream service
  - client file for datagram service
  - server file for datagram service
  - PDF file for additional questions
• Use C/C++ or Java for your programs
• Send all programs to GTA and cc. Dr. Sterbenz
  - no zip files, no binary, just the source code and pdf file

Socket Programming

Acknowledgements

Some material in these foils comes from the textbook
supplementary materials:
• Kurose & Ross,
  Computer Networking:
  A Top-Down Approach, 5th ed.
  http://wps.aw.com/aw_kurose_network_5

• Stevens,
  UNIX Network Programming
  Prentice Hall, 1990
  http://www.kohala.com/start/unp.html
Socket Programming

Acknowledgements

Some material in these foils comes from the textbook supplementary materials:


• Python.org http://wiki.python.org/moin/TcpCommunication
• Python.org http://wiki.python.org/moin/UdpCommunication

Socket Programming

Additional Reading

Some material in these foils comes from the textbook supplementary materials:

• http://www.cs.utusa.edu/~korkmaz/teaching/cn-resources/programs/capitalize-tcp/
• http://www.cs.utusa.edu/~korkmaz/teaching/cn-resources/programs/capitalize-udp/
• http://gaia.cs.umass.edu/ntu_socket/
• http://beej.us/guide/bgnet/
• http://java.sun.com/docs/books/tutorial/networking/sockets/
• http://www.faqs.org/faqs/unix-faq/socket/
• http://www.iana.org/assignments/port-numbers
• http://docs.freebsd.org/44doc/psd/20.ipctut/paper.pdf
• http://docs.freebsd.org/44doc/psd/21.ipc/paper.pdf