인터넷 프로토콜 5장

데이터 송수신 (2)

목포해양대 해양컴퓨터공학과

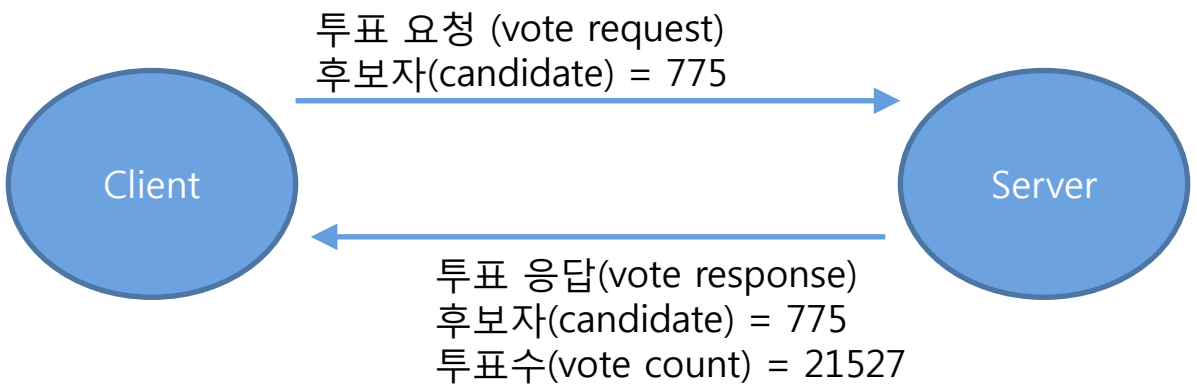
제 5장 데이터의 송수신

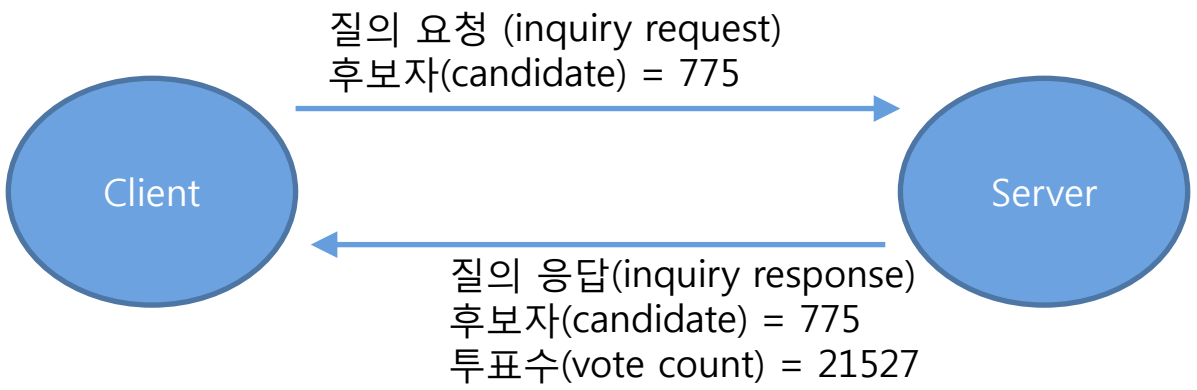
- 5.1 정수 인코딩
- 5.2 메시지 생성, 프레이밍, 그리고 파싱
- 5.3 마무리

5.2 메시지 생성, 프레이밍, 그리고 파싱

- 응용 프로그램 작성시 고려할 사항
  - 트랜스포트 프로토콜은 무엇을 이용할 것인가?
    - UDP
    - TCP
  - TCP를 이용하는 경우 프레이밍 고려
    - 하나의 메시지 크기(경계)를 어떻게 판단할 것인가 ?
    - 방법 1: delimiter-based
    - 방법 2: Explicit length
  - 메시지 인코딩 방식
    - Text-based Message Encoding
    - Binary Message Encoding

Voting 프로토콜


투표 요청 (vote request)
후보자(candidate) = 775
Client
Server
투표 응답(vote response)
후보자(candidate) = 775
투표수(vote count) = 21527


질의 요청 (inquiry request)
후보자(candidate) = 775
Client
Server
질의 응답(inquiry response)
후보자(candidate) = 775
투표수(vote count) = 21527

Voting 프로그램 (1)

- 메시지가 인코딩되는 자세한 내용은 프로그램 로직에 분리하여 숨기는 것이 바람직
  - 본문에서 사용하는 구조체와 전송 형태의 분리
- VoteClientTCP.c
  - 프로그램 본문에서는 구조체를 넘겨주어 Encode 함수 호출
  - 프레임화 전송 시 PutMsg 함수 호출
  - 메시지 수신 시 GetNextMsg 함수 호출
  - 프로그램 본문에서는 Decode 함수를 호출하여 구조체로 변환

Voting 프로그램 (2)

- VoteServerTCP.c
  - 메시지 수신 시 GetNextMsg 함수 호출
  - 프로그램 본문에서는 Decode 함수를 호출하여 구조체로 변환
  - 변환된 구조체를 이용하여 투표 관련 처리
  - 응답 메시지를 생성하기 위한 구조체를 넘겨주어 Encode 함수 호출
  - 프레임화 전송 시 PutMsg 함수 호출

5.2.1 프레이밍

- 구분자 기반(delimiter-based) 방식
  - 텍스트로 인코딩된 메시지에서 주로 사용
  - 메시지 자체 내부에 구분자 패턴이나 기호가 없어야 함
  - 채우기(stuffing) 기술
  - 송신자, 수신자 모두 메시지의 모든 바이트를 검사하여야 함
- 길이 명시(explicit length) 방식
  - 길이를 먼저 보내고, 그 길이만큼의 메시지를 보내는 방식
  - 더 간단하나 메시지 최대 크기에 대한 제한 존재

본문 예제

- DelimFramer.c
  - GetNextMsg( )
  - PutMsg( )
- LengthFramer.c
  - GetNextMsg( )
  - PutMsg( )

5.2.2 텍스트 기반의 메시지 인코딩

- 선로 상의 형식(Wire Format) 정의
- '매직' 문자열
  - 프로토콜 정상 동작 여부를 판단하기 위해 상호 약속한 문자열을 보내고, 이를 확인
  - 네트워크에 들어오는 입력에 대해 어떠한 가정도 하지 말아야 함
    - 비정상적인 메시지가 들어오더라도 이를 처리할 수 있어야 함 (오류 처리)
- 본문 예제

5.2.3 이진 형식의 메시지 인코딩

- 선로 상의 형식(Wire Format) 정의
- ‘매직’ 비트 스트링
- 본문 예제

오류 상황 및 해결 방안

- VoteClientTCP.c에서 수신된 응답에 먼저 송신한 요청 메시지가 남아있어 득표수 확인이 안되는 이상 현상 발생
  - 방법 1
    - GetNextMsg()를 한번 더 호출하여 남아있는 메시지를 제거한 후 응답 확인 가능
    - 근본적인 해결책이 될 수 없음
  - Stream I/O로 Wrap 하는 과정에서 시스템의 오동작이 발생하는 것으로 추정됨
  - 방법 2
    - VoteClientTCP.c와 VoteServerTCP.c에서 Stream I/O로 Wrap 하는 과정을 제거하고, DelimFramer.c와 LengthFramer.c에서 파일이 아닌 소켓에 바로 읽고 쓰는 방식으로 수정
- Makefile을 이용하여 개발을 용이하게 할 필요가 있음

Makefile (1)

```
Common_files = addressUtility.o dieWithMessage.o
LIBS = -lsocket -lnsl
CFLAGS = -xc99

vs_tcp_td : vsTcp.o tcpServerUtility.o voteEncodingText.o delimeterFramer.o $(Common_files)
		cc -o vs_tcp_td $(CFLAGS) $(LIBS) vsTcp.o tcpServerUtility.o voteEncodingText.o delimeterFramer.o $(Common_files)

vc_tcp_td : vcTcp.o tcpClientUtility.o voteEncodingText.o delimeterFramer.o $(Common_files)
		cc -o vc_tcp_td $(CFLAGS) $(LIBS) vcTcp.o tcpClientUtility.o voteEncodingText.o delimeterFramer.o $(Common_files)

vs_tcp_tl : vsTcp.o tcpServerUtility.o voteEncodingText.o lengthFramer.o $(Common_files)
		cc -o vs_tcp_tl $(CFLAGS) $(LIBS) vsTcp.o tcpServerUtility.o voteEncodingText.o lengthFramer.o $(Common_files)

vc_tcp_tl : vcTcp.o tcpClientUtility.o voteEncodingText.o lengthFramer.o $(Common_files)
		cc -o vc_tcp_tl $(CFLAGS) $(LIBS) vcTcp.o tcpClientUtility.o voteEncodingText.o lengthFramer.o $(Common_files)

vs_tcp_bd : vsTcp.o tcpServerUtility.o voteEncodingBin.o delimeterFramer.o $(Common_files)
		cc -o vs_tcp_bd $(CFLAGS) $(LIBS) vsTcp.o tcpServerUtility.o voteEncodingBin.o delimeterFramer.o $(Common_files)

vc_tcp_bd : vcTcp.o tcpClientUtility.o voteEncodingBin.o delimeterFramer.o $(Common_files)
		cc -o vc_tcp_bd $(CFLAGS) $(LIBS) vcTcp.o tcpClientUtility.o voteEncodingBin.o delimeterFramer.o $(Common_files)

vs_tcp_bl : vsTcp.o tcpServerUtility.o voteEncodingBin.o lengthFramer.o $(Common_files)
		cc -o vs_tcp_bl $(CFLAGS) $(LIBS) vsTcp.o tcpServerUtility.o voteEncodingBin.o lengthFramer.o $(Common_files)

vc_tcp_bl : vcTcp.o tcpClientUtility.o voteEncodingBin.o lengthFramer.o $(Common_files)
		cc -o vc_tcp_bl $(CFLAGS) $(LIBS) vcTcp.o tcpClientUtility.o voteEncodingBin.o lengthFramer.o $(Common_files)
```

Makefile (2)

```
clean :
        rm *.o
        rm vs_tcp_* vc_tcp_*

all : vs_tcp_td vc_tcp_td vs_tcp_tl vc_tcp_tl vs_tcp_bd vc_tcp_bd vs_tcp_bl vc_tcp_bl
```