TCP/IP Socket Programming

Chapter 3. 메시지 구성하기

목 차

0. 개요

- 소켓 애플리케이션 작성
  - 하나의 애플리케이션 프로토콜 구현
- 프로그램
  - 다른 프로그램에 정보 제공
  - 다른 프로그램으로부터 제공된 정보 이용
- 송수신자간의 합의 필요
  - 데이터 부호화(data encoded)
  - 송수신 정보의 종류
  - 통신의 연결 및 종료의 방법

0. 개요 (Cont.)

- 소켓을 사용하는 대부분의 프로토콜
  - 데이터 부호화 **(data encoding)**
  - 메시지 포맷 **(format)**
  - 파싱 **(parsing)**

- **TCP/IP 프로토콜**
  - 사용자 데이터의 각 바이트의 조사, 변경 없이 전송
  - 어플리케이션들에 자신의 정보를 제공하기 위해 어떤 식으로 부호화 할 것인지에 대해 유연성 제공
  - 대부분의 어플리케이션 : 필드로 구성된 메시지 이용
  - 포맷, 파싱, 부분의 의미 추출 방법 규정 필요

가 정

- 은행의 예
  - 변수(정수형) deposits, withdrawals
  - 정수형 s
    통신을 위한(TCP)소켓의 식별자를 가지고 있음
  - 크기
    - sizeof(int)는 4 byte, sizeof(short)는 2 byte
  - TCP (send(), recv())를 사용
    - 특별한 조건dl 없는 한 UDP에서 동일하게 적용

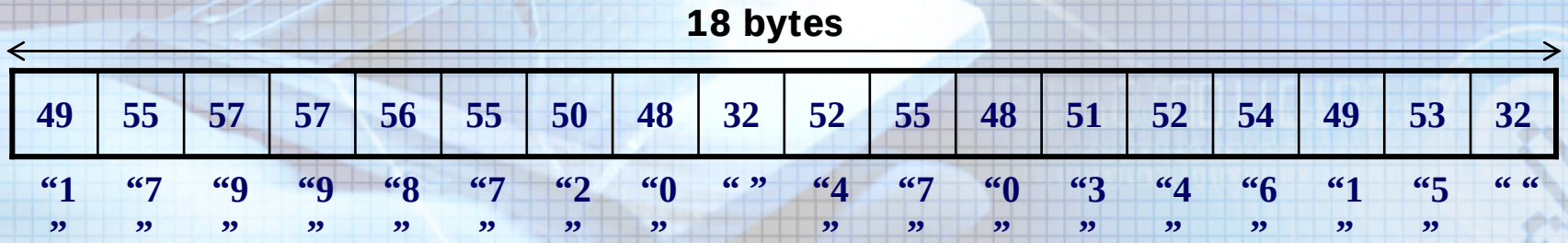
3.1 데이터 부호화 (Encoding Data)

- 출금(withdrawal) 및 예치(deposit) 정보를 표현하는 방법
  - 숫자를 출력하는 십진수(digit)의 스트링 :

    ASCII와 같은 문자 부호화(character encoding)에 따라 결정되는 바이트 순열
  - 장점
    - 숫자를 인쇄, 화면 윈도우에 표시할 때 사용되는 방식과 같음
    - 임의의 큰 수들도 전송 가능

3.1 데이터 부호화 (Encoding Data)

- 두개의 필드 분할방법 (구획자(delimiter) 사용)
  - 0(48)~9(57)이외의 값으로 표현
  - 일반적으로 null 바이트 사용
  - White space 바이트(스페이스 또는 탭)
- 예제
  - 17,998,720 , 47,034,615
  - Delimiter(구분자) : ASCII (32)



3.1 데이터 부호화 (Encoding Data)

- 보내는 프로그램의 구성

```
sprintf (s, msgBuffer, “%d %d ”, deposits, withdreawals);
send (s, msgBuffer, strlen(msgBuffer), 0);
```

- 주의사항
  - Sprintf() 함수 포맷 스트링 끝에 공백 추가
  - msgBuffer는 가능한 가장 큰 결과를 포함 할 수 있는 크기
    예) 9자리 수(부호 포함)일 경우, 23바이트 이상 크기를 가져야함
  - Send() 함수의 strlen()은 메시지 두 필드를 구성하는 바이트만 계산

3.1 데이터 부호화 (Encoding Data)

- 잘못 된 예

```
#define BUFSIZE 132
..
Char msgBuffer [BUFSIZE];
..
..
sprintf (s,  msgBuffer,  “%d %d ”,  deposits,  withdreawals);
send (s,  msgBuffer,  BUFSIZE,  0);
```

수신측 프로그램이 두 번째 스페이스를 넘어서 추가 바이트 수신

3.1 데이터 부호화 (Encoding Data)

- 송신자가 구조체를 사용하는 방법

```
struct {
        int dep;
        int wd;
} msgStruct

msgStruct.dep = deposits;
msgStruct.wd = withdrawals;
Send (s, &msgStruct, sizeof(msgStruct), 0);
```

- 직접 **deposits, withedrawals** 값 **TCP**로 직접 전송 가능

```
Send (s, &deposits, sizeof(deposits), 0);
Send (s, &withdrawals, sizeof(withdrawals), 0);
```

3.2 바이트 순서 (Byte Ordering)

- 가정

  Deposits : 17,998,720

  Withdrawals : 47,034,615

- 바이트 순서
  - msgStruct 코드에 의해 설명
  - Big-Endian : Motorola 68K, sparc

1	18	163	128	2	205	176	247

  - Little-Endian : Intel X86, DEC Alpha

128	163	18	1	247	176	205	2

3.2 바이트 순서 (Byte Ordering)

- 수신과 송신이 반대의 바이트 순서를 따르는 경우
  - 17,998,720 -> -2,136,796,671
  - 47,034,615 -> 139,408,126
- 표준 네트워크 바이트 순서 :: **Big-Endian**

- 2바이트, 4바이트 정수를 원래의 값에서
네트워크 바이트 순서로 변환시켜주는 표준 함수들이 제공

  - htonl( ) , ntohl( )
    - 4 바이트 값을 원래 바이트 순에서 네트워크 바이트 순서로 변환
  - htons( ), ntohs( )
    - 2 바이트 값을 네트워크 순서로 변환 후 반환

3.2 바이트 순서 (Byte Ordering)

```
long int htonl (long int hostLong)      // 4바이트 값 -> 네트워크 바이트 순서 변환
Short int htons (short int hostShort)   // 2바이트 값 -> 네트워크 순서 변환
long int ntohl (long int netLong)       // 네트워크값 -> 4바이트 원래 값으로
Short int ntohs (short int netShort)    // 네트워크값 -> 2바이트 원래 값으로
```

- 여러 바이트의 이진값을 다른 장치로 보낼 때
  - hton * () 함수 필요

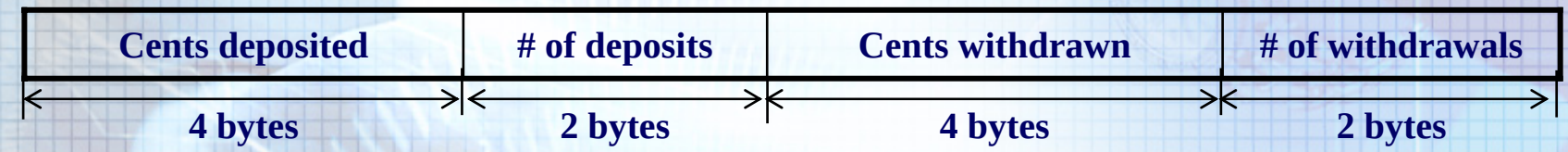
- 송신자를 위한 정확한 코드(4byte 정수형 가정)

```
msgStruct.dep = htonl (deposits);
msgStruct.wd = htonl (withdreawals);
send (s, &msgStruct, sizeof (msgStruct), 0);
```

3.3 정렬(Alignment)과 채워넣기(Padding)

- 정렬(alignment)
  - 여러 개의 서로 다른 크기의 이진-부호화된(binary-encoded) 필드들을 포함할 때 고려사항
- 가정
  - 은행에서 예치 및 출금된 수 뿐 아니라 메시지가 예치 및 출금 트랜잭션(transaction)횟수를 포함
  - 예치, 출금의 트랜잭션 수가 65,536회를 넘지 않음
  - 2바이트의 정수형(unsigned short)로 표현 가능

3.3 정렬(Alignment)과 채워넣기(Padding)



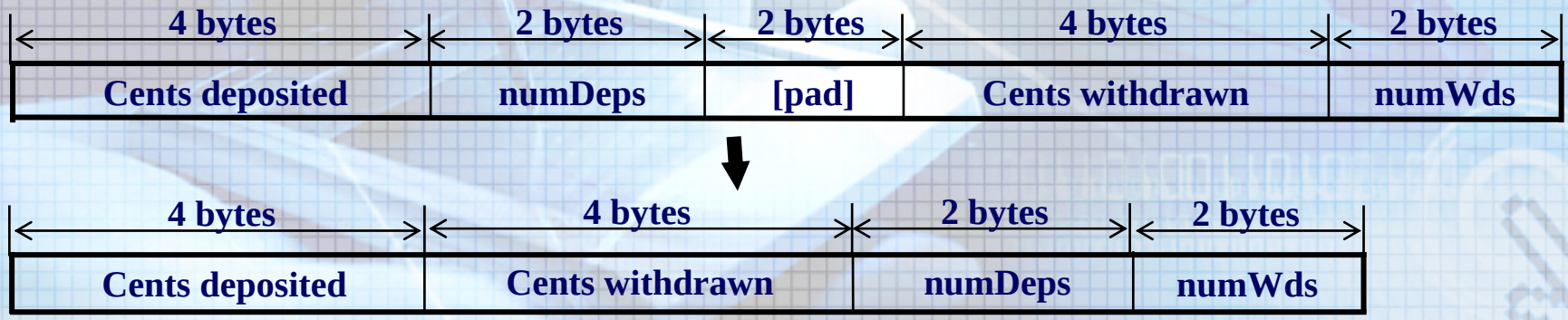

```
Struct {
        int centsDeposited;
        unsigned short short numbDeps;
        int centsWithdrawn;
        unsigned short numWds;
 } msgBuf;

Send (s, &msgBuf, sizeof(msgBuf), 0);
```

- **Padding** 에 의해 **14**바이트 메시지 생성됨

3.3 정렬(Alignment)과 채워넣기(Padding)

- 주요사항
  - 자료 구조들은 최대로 정렬
    - 주소들이 가장 큰 원래 정수 크기로 나누어짐
  - 다른 여러 바이트의 필드들은 크기에 따라 정렬
    - 4바이트 정수 주소
    - 2바이트 정수 주소

```
Struct {
        int centsDeposited;
        unsigned short short numbDeps;
        int centsWithdrawn;
        unsigned short numWds;
} msgBuf;
```



3.4 틀짜기(Framing) 및 파싱(Parsing)

- 틀짜기(Framing)
  - 수신자가 메시지를 파싱 할 수 있도록 정보 포맷
  - 어플리케이션 프로토콜은 메시지 수신자가 전송된 메시지를 모두 받았는지 결정할 수 있게 함
  - 메시지 필드 – 고정 크기, 이미 알려진 크기로 가정
    - 수신자 : 기대하고 있는 크기 만큼 바이트만을 버퍼에 저장
    - 텍스트-스트링(text-string) 표현법
    - TCP소켓을 사용할 때는 어느 정도 수신 후
      - 파싱을 위한 코드 구현 필요

3.4 틀짜기(Framing) 및 파싱(Parsing)

- **ReceiveMessage()**
  - 메시지 수신 후 파싱
  - 길이 가 주어진 한계를 초과하지 않는 완전한 메시지 수신 후 버퍼에 저장한 후에 반환
  - 에러 발생시 에러-탈출 (error-exit)루틴 수행
  - 버퍼에 저장된 메시지 길이 반환

3.4 틀짜기(Framing) 및 파싱(Parsing)

```
# define DELIMCHAR ‘ ‘
# define FILEDSPERMSG 2

int ReceiveMessage (int s, char * buf, int maxLen)  // 한번에 한 바이트 요구,
{
    int received=0;
    int delimCount=0;
    int rv;
    while ((receivd < maxLen) && (delimCount < FIELDSPERMSG))
    {
        rv = recv (s, buf+received, 1, 0);   // recv함수는 수신된 바이트 수를 반환, 오류발생시
    –1 반환
        if ( rv < 0)
            DieWithError (“recv() failed”);
        else if (rv == 0)
             DieWithError (“unexpected end of transmission”);
        if ( * ( buf+received) == DELIMCHAR)
            delimCount += 1;
        received += 1;
    }
```