데이터 송수신 (2)

네트워크 프로그램 설계 5장

제 5장 데이터의 송수신

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

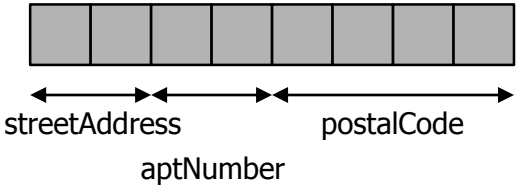
TCP소켓을 스트림으로 포장하기

- TCP 소켓에 복수 바이트 정수값을 인코딩하는 다른 방법
  - 파일 스트림 사용
  - fdopen()을 호출하여 파일 스트림과 소켓식별자를 연관시킨 후 사용
    - FILE *fdopen( )
    - int fclose( )
    - int fflush( )
    - size_t fwrite( )
    - size_t fread( )
- 권장하지는 않음 !

구조체 오버레이 : 정렬과 채우기

- 실제로 구조체가 어떻게 구현되는지 이해 필요

```
struct addressInfo {
    uint16_t    streetAddress;
    int16_t     aptNumber;
    uint32_t    postalCode;
} addrInfo;
```




```
struct integerMessage2 {
    uint8_t     oneByte;
    uint8_t     padding;
    uint16_t    twoBytes;
    uint32_t    fourBytes;
    uint64_t    eightBytes;
};
```

- alignment를 위해 필요한 padding 이해

```
struct integerMessage {
    uint8_t     oneByte;
    uint16_t    twoBytes;
    uint32_t    fourBytes;
    uint64_t    eightBytes;
};
```

- 구조체와 메모리 할당 이해
  - 최대한 정렬
  - 2바이트 이상인 경우 시작주소는 항상 2로 나누어져야 한다

문자열과 텍스트 (1)

- old-fashioned text
  - ASCII
  - C언어는 ASCII의 축소판인 기본 문자셋 (basic character set) 명시 (C99 – ISO646)
- “internationalizable” code
  - C99 extensions
    - wchar_t (wide character) 지원
    - size_t wcstombs( ) (wide character string to multibyte string)
    - size_t mbstowcs( )

문자열과 텍스트 (2)

- “internationalizable” code (계속)
  - C99에서도 인코딩 방식에 대한 명시적 제어권이 제공되지 않음
  - 플랫폼에서 정의한 locale에 따라 정의된 하나의 고정된 문자셋을 이용한다고 가정 (실시간에 locale이 변경되는 경우 결과를 예측할 수 없음!)
  - 서로 다른 문자셋을 혼용하여 이용할 방법이 없음!
- 관련 명령 / 함수(라이브러리)들 (조사하여 정리할 것!)
  - locale
  - wcslen( ), wcslen_l( ), wcscpy( ), …
  - iconv

비트 조작 : 참, 거짓 값의 인코딩 (1)

- 비트맵(bitmaps)
  - 참, 거짓 정보를 인코딩하는 가장 간단한 방법
  - 파일의 접근권한
- C 언어의 bitwise 연산자 이해 필요
  - <<
  - >>
  - |
  - &
  - ~

비트 조작 : 참, 거짓 값의 인코딩 (2)

- 마스크(mask)
  - 하나 혹은 그 이상의 특정 비트가 1로 설정되어 있고, 나머지는 0으로 설정되어 있는 정수값

  - & 연산으로 특정 비트가 1로 설정되어 있는지 여부 판단
  - | 연산으로 특정 비트를 1로 설정
  - 특정 비트를 해제하고자 하는 경우 마스크에 ~연산(비트-보수(bitwise complement)) 한 후 &

응용 과제 1

- 전체 전원스위치 1개, 전등 스위치 5개를 감시 또는 제어하는 프로그램 작성
  - 비트 단위로 처리

  - 필요한 기능
    - 전원 스위치 on/off
    - n 번째 전등 스위치 on/off
    - 현재 스위치 상태 출력
    - 현재의 전등 상태 출력
      - 전원 스위치와 전등 스위치를 같이 고려하여 상태 표시

전송 데이터의 자료형 (1)

- 문자열(String) 전송: 가변 길이 전송
  - 장점
    - 사람이 읽기 쉬움
    - 메시지의 확장이 용이하며 무제한
  - 단점
    - 전송량 대비 전송 내용 비효율, 수신 루틴 비효율, 연산 비효율
  - 상호 협의 할 내용
    - 문자 코드 페이지
      - ASCII, Unicode, UTF
    - 메시지 경계 구분 (프레이밍: framing)
      - 길이 명시 방식: 전송할 문자열의 크기를 고정크기의 자료형에 담아서 전송. 수신자는 크기를 미리 파악하고 정확한 문자열만큼 수신
      - 구분자 방식: 널 문자 혹은 임의의 문자를 메시지의 경계에 삽입. 수신자는 바이트 단위로 읽다가 구분자가 나오면 메시지의 끝으로 확인

전송 데이터의 자료형 (2)

- 문자열(String) 전송
  - 숫자 전송의 예

49	55	57	57	56	55	48	10
'1'	'7'	'9'	'9'	'8'	'7'	'0'	₩n

  - 문자 전송의 예

0	77	0	111	0	109	0	10
	M		o		m		₩n

3	77	111	109

문자열 전송의 예(TCP)

```
char string[strBuffSize];

send(sock, string, strBuffSize, 0)
```

- 주의점 : 버퍼의 크기
  - TCP는 운영체제에 의존적인 TCP 버퍼가 있으며 이 보다 작은 크기로 send()를 호출해야 한다.

전송 데이터의 자료형 (3)

- 정수형(Integer)의 전송
  - 기본 자료형의 단위로 전송
    - 2바이트, 4바이트 단위의 전송
  - 주의 사항
    - 2바이트 이상의 데이터 전송간에는 항상 네트워크 바이트 순서로 전송해야 함
    - 1바이트 교환은 의미가 없음
- Network byte order (Big-Endian)
  - 다중 바이트의 메시지 교환에 필수
  - 호스트 바이트-네트워크 바이트 변환 함수들
    - htonl(), htons(), ntohl(), ntohs()

Little-Endian	0	0	92	246
		23,798		
Big-Endian	246	92	0	0

정수 자료형 전송의 예(TCP)

```
int data;
send(sock, &data, sizeof(data), 0)
```

- ‘short’, ‘long’, ‘double’, ‘char’도 동일한 방식으로 처리