X 윈도우 프로그래밍 기초과정

이만용, geoman@nownuri.net
1996년 3월 19일

목 차

제 1 절 시작하면서

리눅서들은 단지 기성제품을 사용하는데 만족하지 않고, 뜯어고치고 창조하는 일을 좋아하는 굉장한 유저들입니다. 리눅스 커널버전과 슬랙버전 만큼 빨리 변하는 운영체제 소프트웨어가 어디 있겠습니까? 또 이만큼 운영체제의 "기본기"에 충실한 체제가 어디 있겠습니까? 그래서 우리 리눅서들은 어떤 유저들보다도 자부심이 대단합니다.

리눅스여, 만세 만세 만만세!

리눅스를 사용하면서 컴파일을 안해보신 분이라면 아직 리눅스의 진가를 맛보지 못하신 분이라고 할 수 있지요. 리눅스를 사용하면서 자기 스스로 프로그램을 짜보지 못하신 분들은 리눅스의 진짜 진짜 진가를 맛보지 못하는 분이라고 할 수 있습니다. 이제부터는 리눅스의 참맛을 만끽해볼까 합니다.

1.1 강의의 대상

제목에서도 분명하게 나타나는 것처럼, 제가 다루고자 하는 내용은 프로그래밍 일반에 관한 것이 아닙니다. 그것은 제가 다룰 수 있는 한도를 벗어납니다. 이 강좌를 읽고자 하시는 분들은 기본적으로 다음과 같은 사항에 모두 해당되시는 분들이셔야 원할한 학습이 이루어질 것이라고 생각합니다.

- 기본적인 C 프로그래밍을 하실 수 있는 분
- SHELL 스크립트에 대해 아시는 분
- 유닉스 계열의 프로세스/화일처리/시그널 등의 개념을 아시는 분

좀 사양이 높죠? 그래서 그 동안 망설이게 되었습니다. 우선은 리눅스 대중화에 기여할 수 있는 기본적인 것들에 대해서만 강의를 진행할까? 하는 생각도 수없이 많이 해보았습니다. 그러나 리눅스동의 많은 분들께서도 양적변화 뿐 아니라 이젠 질적변화를 서서히 추구해야 한다는 문제의식을 가지신 것 같아서 용기를 내어 해봅니다.

1.2 대략적인 강의 내용 소개

재차 말씀드리지만 강의내용은 리눅스 프로그래밍 일반에 대하여 다루지 않습니다. 저는 X 윈도우 시스템의 GUI 자원을 자신의 프로그램에서 이용하는데 대하여, 즉 X 윈도우용 프로그램의 "인터페이스"에 관한 내용만 다루고자 합니다.

- X 내에서 창의 생성과 소멸, 관리
- X 그래픽
- X Event 처리하기
- 키보드/마우스 등의 입력
- 기타 등등

MS 윈도우즈 같은 GUI 체제에서의 프로그래밍을 해보신 분들은 아실겁니다. 거의 시중에 있는 대부분의 책들이 이러한 내용을 다루고 있지요. 위 내용만 익히신다고 해서 완전한 의미의 X 윈도우용 프로그램을 만드실 수는 없을 겁니다. 하지만 위의 내용을 모르신다면 X 윈도우용 프로그램은 절대로 만드실 수 없습니다. 그 정도로 아주 기초적인 내용만을 담고 있습니다. 더욱 심도깊은 X 윈도우 프로그래밍 강좌는 이번 기초 강좌가 끝난 후에야 가능할 것 같습니다. (그 유명한 위젯 같은 것 등등)

제 2 절 X 프로그래밍 기초 : Xlib

X 윈도우 프로그래밍에 도전하시는 여러분을 환영합니다. 저도 아는 것은 많이 없지만 여러분을 조금이라도 도울 수 있다면, 그리고 저의 강좌를 계기로 더욱 더 많은 수준높은 X 윈도우 프로그래밍 강좌들이 생겨나길 기대하면서 X 윈도우 프로그래밍 기초과정을 시작할까 합니다. 어렵더라도 포기하지 마시고 끝까지 저와 함께 가셔서 당당한 X 윈도우 프로그래머로 우뚝 서십시요. 그리고 리눅스를 빛내줍니다.

2.1 X 윈도우 프로그래밍에 들어가기 앞서

여러분의 선택은 정말로 탁월합니다. X 윈도우 시스템은 MS 윈도우즈 95와 같은 엉터리에다 단명하는 체제와는 전혀 다른 수준의 안정되고 표준적인 시스템입니다. 여러분께서 미래를 생각하신다면, 바로 X 윈도우 프로그래밍을 선택하십시요.

자, 우선 점검해야 할 것들이 있습니다. 여러분의 시스템이 X 프로그래밍을 할 준비가 되어 있는지 알아봐야 하지 않을까요? 이 글을 읽는 분들은 최소한 X 윈도우가 작동하고 계신 선택받은(?) 분들일 것이고, 남은 것은 여러분이 슬랙웨어 설치시 X 프로그래밍을 할 수 있는 헤더 파일 등을 제대로 설치하셨는지 확인해 보시기 바랍니다. /usr/include/X11 ㈁(v)그리고 /usr/X11/lib 에 화일들이 있는지 말입니다.

모두 이상없다고요? 네, 그러면 시작합시다.

2.2 X 윈도우 프로그래밍이란 ?

X 윈도우 프로그램이란 X 윈도우의 '자원(Resource)'을 이용하는 프로그램입니다. X의 자원들은 대표적으로 창(Window), 색상(Color Map), 폰트(font) 등이 있습니다. 이러한 자원들을 X 윈도우 서버(Server)라고 하는 녀석이 전권을 가지고 관리하고 있습니다. 우리의 역할은 이미 정해져 있는 '규칙'에 따라서, 자원을 관리하고 있는 서버에 연결하여, 서버더러 우리가 원하는 자원을 서비스해달라고 요청하는 일입니다. 바로 우리는 서버라는 놈과 어떻게 연결할 것인지, 그리고 어떻게 서버더러 이것 저것 우리가 원하는 것을 달라고 하는지를 배우는 것을 목표로 하게 됩니다.

이러한 프로그램을 클라이언트(Client)라고 하지요. 서버/클라이언트 개념이라고 해서 항상 프로그램을 "무엇이 무엇에 연결하여 원하는 일을 한다"라는 사고방식 하에 모든 프로그램을 짜게 됩니다. 이는 더 나아가 세계

저편에 있는 서버에 우리집의 클라이언트가 연결을 하여 서비스를 받는다는 "네트워크 지향적"이고 미래 지향적인 프로그래밍이 됩니다. 인터넷 시대를 살고 있는 우리에게는 아주 중요한 개념이라고 할 수 있습니다. 여러분께서 저와 함께 X 프로그래밍의 세계를 탐험하시다 보면 아주 자연스럽게 멋진 네트워크 프로그램에 도전하실 수 있을 겁니다.

우리는 단지 정해진 규칙에 따라 서버에 연결하고, 서비스를 받고, 연결을 해제 하는 것을 배우는 것일 뿐이라는 사실을 항상 명심해주세요!

2.3 알아두어야 할 사실

여러분 모두 당장에 실제적인 X 윈도우 프로그래밍을 하고 싶으실 겁니다. 하지만, 이러한 사실들은 알고 들어가셔야 더욱 더 빨리 프로그래밍에 익숙해지실 수 있다고 생각합니다.

cd /usr/X11/lib 해보세요. 그리고 ls ... lib라는 문자로 시작하는 많은 화일들이 있는 것을 확인하실 수 있을 겁니다. 이 녀석들 'ㅡ라이브러리'는 우리를 항상 도와주는 놈들입니다. 이들이 없으면 아무 것도 할 수가 없습니다. 이 녀석들을 여러분께 소개시켜 드리고자 합니다.

- libX11 : 이름에도 드러나듯이 가장 기본적인 라이브러리입니다. 모든 친구들은 이 친구 없이는 무용지물입니다. 가장 중요한, 그리고 가장 본질적인 X 의 기능을 실현해주는 Xlib 입니다.
- libXt : X 툴킷(X Toolkit Intrinsic)이라고 부릅니다. Xlib 위에서 구현되어 있는 아주 유용한 루틴들입니다. 나중에 위젯이라고 하는 X 인터페이스에서 아주 중요한 놈들을 다룰 때 많이 사용합니다. 이것을 자유자재로 사용하실 수 있는 수준이 되시면 X 프로그래밍은 끝난 거나 다름없죠.
- libXaw : Athena Widget 입니다. 앞글자를 따서 aw 입니다. 지금 현재는 거의 상용 모티프(Motif)만이 사용되어 거의 씌이지 않는 것 같습니다. 하지만 X를 개발한 MIT가 만든 것으로서 Free 이므로, 저는 여러분과 미흡하나마 이 친구를 가지고 예쁜 프로그램 인터페이스를 만들어 볼까합니다. 모티프에 관한 것은 서점에 책이 온통 깔려 있으니 그것을 참고하시면 됩니다. 어찌 되었든 알아두시면, 모티프도 금방 사용하실 수 있습니다. 요즘 libXaw3d가 나와있는데, 원판 libXaw가 워낙 초라해서 한계가 있기는 하지만, 제 개인적인 견해는 그래도 설치하시는 것이 더 좋다입니다. 세연이나 고스트스크립트의 버튼이 약간 3차원적으로, 그리고 모티프 냄새를 풍기더군요.
- libXmu : Miscellaneous Utilities의 앞글자들만 따서 mu 입니다. 기타 등등의 기능을 구현해주는 녀석이지만, 없어서는 안될 녀석입니다. 그 기타 등등이라고 하는 것이 하도 많이 쓰여서...
- libXpm : 확장자가 xpm인 화일들 보신 적 있으시죠? 컬러 비트맵 화일 지원 등의 그래픽 지원 라이브러리입니다. 안 중요할 리가 있을까요?

나머지 친구들은 소개시켜 드리기 힘들군요. 아직 때가 되지 않아서... 나중에 소개시켜 드릴 기회가 올 지...

2.4 자, 이제 시작해볼까요?

클라이언트가 제일 먼저 해야 할 일은 무엇일까요? ... **서버에 접속하는 일입니다.**

과연 나는 어떤 서버를 선택할 것인가? 하는 문제를 해결해야 합니다. 보통 도스, 윈도우즈 환경에 익숙하신 분들은 좀 생소한 개념이지요. 하지만 요즘은 인터넷을 많이 하시기 때문에 이해하시기 어렵지는 않을 겁니다. 리눅스 X 윈도우 환경에서는 아주 간단한 문장들로 여러분 옆에 있는 컴퓨터 말고도 세상 저편에 있는 컴퓨터의 모니터에다 하나의 창을 띄우는 일 정도는 너무도 쉽게 할 수 있거든요.

vi, emacs와 같은 에디터를 이용해서 다음과 같은 내용이 들어 있는 화일을 하나 만들어 보세요.

- 예제 1

```
#include <X11/Xlib.h>

main()
{
        Display *d ;

        d = XOpenDisplay ("localhost:0.0") ;

        XCloseDisplay (d) ;
}
```

만약 위의 내용을 ex01.c라는 이름으로 저장하셨다면,

```
cc -o ex01 ex01.c -lX11 해보십시요.
```

컴파일이 잘됩니까? 안된다구요? 왜 그럴까요? 힘든 영어지만 에러 사유를 잘 보시면 아마도 헤더파일을 못찾는다고 그러지 않나요?

그러면 다음과 같이 INCLUDE 디렉토리를 지정해주십시요.

```
cc -o ex01 ex01.c -I/usr/include/X11 -lX11
```

컴파일은 되도 링크가 안되다면 라이브러리 디렉토리도 지정해주세요. -L/usr/X11/lib 를 명령행에 추가시키시면 됩니다.

제대로 컴파일이 이루어졌다면, 실행을 시켜보죠. ex10 엔터!

여러분께서 실망하시는 것을 느낄 수 있습니다. :) 이게 뭐냐?

하지만! 이 프로그램은 X 윈도우 프로그래밍의 거의 모든 것을 보여주는 것이나 다름없는 아주 중요한 프로그램입니다. 저는 그렇게 우기고 싶네요. :)

이 엉터리(?) X 윈도우 프로그램에 대한 설명을 드리고자 합니다.

첫줄 INCLUDE 문장은 XOpenDisplay, XCloseDisplay, Display 변수가 선언되어 있는 헤더화일 Xlib.h를 포함시키는 문장입니다. 다음으로 당연히 main 함수가 정의되어야겠죠?

Display형 포인터 d 를 선언합니다.

2.4.1 Display란?

디스플레이가 무엇인지 잘 아실 겁니다. 여러분께서 바로 눈 앞에 두고 계신 것이 모니터라고 하는 '디스플레이'입니다. 그런데 여기서 Display라고 하는 것은 X 윈도우 시스템에서 약간 확장된 의미로 생각합니다.

정의 : Display란 하나의 서버가 관리하고 있는 1개 이상의 스크린(Screen), 1개의 키보드, 마우스 등등으로 구성되어 있는 집합체이며 워크스테이션(Workstation) 이라고 부르기도 한다.

이러한 디스플레이는 보통 모니터 하나 전체를 나타내는 스크린을 하나 갖는 경우가 보통이지만 특수한 경우 2개 이상의 스크린을 갖는 경우도 있을 수 있습니다. 하나의 컴퓨터에 두 개의 모니터를 쓰는 "듀얼 모니터" 시스템도 종종 볼 수 있습니다. 그 경우 하드웨어가 받쳐준다면, X 시스템은 두 개의 화면을 각각 이용할 수도 있습니다. 이런 프로그램 하나 만들면 정말 재미있겠죠? X 윈도우 시스템에서는 너무도 쉽게 이루어집니다.

2.4.2 XOpenDisplay 함수

자, 이제 우리가 원하는 디스플레이 서버에 접속을 해봅시다.

```
XOpenDisplay ("localhost:0.0");
```

이 문장은 localhost의 0번 디스플레이(즉, 첫번째 디스플레이)의 0번 스크린(즉, 첫번째 스크린)에 접속하라는 뜻입니다. localhost는 바로 여러분 컴퓨터를 지칭하는 호스트명입니다. 이것을 만약 다른 인터넷 호스트 주소로 써준다면, 어떻겠니까? 여러분의 컴퓨터가 네트워크에 물려있고, 상대방 호스트를 사용할 권한이 주어져 있다면, 우리 프로그램은 저 멀리에 있는 호스트 서버와 연결을 하게 됩니다. 아니, 이렇게 간단할 수가!!! 인터넷 프로그램이라는 것이 별 것 아니군요.

localhost:0.0 과 같은 표현식을 눈여겨 보아주십시요. 명령행에서 한 번 set 이라고 쳐보세요. 아마 내용이 많아서 more 를 해주셔야 할 겁니다. 한 번 그 내용을 주루룩 살펴보시면 DISPLAY=:0.0이라는 표현식을 찾으실 수 있을 겁니다. 그 다음으로 HOSTDISPLAY=freeyong:0.0 이런 표현도 보실 수 있을 겁니다. 물론 freeyong과 같은 부분은 여러분마다 다르지요. 여러분께서 지정해주신 이름이 나타날 겁니다. localhost:0.0 을 freeyong:0.0등으로 바꾸어 주셔도 됩니다. 하지만 일부러 아무 의미도 없는 이름을 주어보십시요. 컴파일하고 실행을 시켜 보시면 그 결과를 아실 수 있을 겁니다. 꼭! 확인해보세요. 보통은 표현식 부분에 NULL을 주면 현재 DISPLAY 라는 환경변수의 값을 읽어서 서버로 사용합니다.

2.4.3 XCloseDisplay 함수

접속해서 한 일은 하나도 없지만, 이제 접속을 마쳐야 하지 않을까요? 그 함수가 바로 XCloseDisplay 함수입니다.

오늘은 아주 아주 간단하지만, 또한 아주 아주 근본적인 프로그램을 하나 짜보았습니다. 모든 X 윈도우 프로그램은 기본적으로 예제 1 프로그램과 그 형식이 같습니다. 서버에 연결하고 서비스 받은 후, 접속을 해제합니다. 모든 유용한 작업은 XOpenDisplay 함수와 XCloseDisplay 함수 사이에 적어주시면 됩니다.

이 글을 다 읽으신 후, 꼭 Xlib.h 화일의 내용을 살펴보십시요. 예를 들어 Display 형 변수는 어떻게 선언, 정의되어 있는지 보시면, 정말로 많은 정보를지니고 있는 구조체(struct)라는 것을 확인하실 수 있습니다.

2.5 간단한 창 하나 만들어 봅시다.

앞에서 만든 원초적(?) 프로그램은 단지 서버와 접속하자 마자 접속을 끊어버리는 경우였습니다. 그럼, 이제는 서버에 접속하고 나서 아주 간단한 일을 하나 시켜보도록 합시다. 바로 창 하나를 만들어서 표시해보는 겁니다.

- 예제 2

```
#include <X11/Xlib.h>

main()
{
        Display *d ;
        Window   w, root ;

        d = XOpenDisplay(NULL) ;

        root = XDefaultRootWindow (d);
        w = XCreateSimpleWindow ( d, root, 50, 50, 400, 300,
                                  2, BlackPixel (d,0), WhitePixel(d,0) );
        XMapWindow (d, w);
        XFlush (d);

        getchar();

        XCloseDisplay (d);
}
```

이번 예제에서는 Window 라는 새로운 자료형과 XDefaultRootWindow(), XCreateSimpleWindow(), XMapWindow(), XFlush() 등의 새로운 함수들이 나오는군요. 앗! BlackPixel(), WhitePixel()과 같은 매크로도 보이는군요. 하지만 이 매크로들은 나중에 설명드리기로 하겠습니다. 하루에 너무 많은 것을 배우면 머리가 아프잖아요?

2.5.1 Window 형 변수

우리가 창을 만들고 싶을 때는 창 하나마다 위의 예에서 나온 Window 형 변수 하나가 필요하다고 생각하시면 됩니다. Window 형 변수 하나는 그 창에 대한 모든 정보를 지니고 있는 복잡한 구조체 정도라고 생각하시면 됩니다. 예상할 수 있는 것으로는 창의 크기, 위치, 색상 등이 있습니다.

2.5.2 창 생성, 표시 ...

지금부터 설명드리는 기본적인 절차를 꼭 머리 속에 넣어두세요. 어떤 창을 만들든지 우리는 우선적으로 그 창에 대한 정보를 저장하고 있을 변수 하나를 이미 만들어놓아야 합니다. 즉 Window 형 변수 하나를. 그 다음 서버더러 창을 하나 만들겠다고 서비스 요청을 합니다. (조금 뒤에 설명드리는 함수를 통해) 이변이 없는 한 서버는 창이라는 자원 하나를 서버 측(!!!)에 만들어 놓습니다. 그리고 창 자원에 대한 자원 고유번호(Resource ID)를 아까 클라이언트 쪽에서 만들어 놓은 변수에 반환해줍니다. 이 변수는 바로 클라이언트와 서버 측에 만들어진 한 창에 대한 연결통로 역할을 해줍니다. 무지 중요하죠?

이렇게 서버 측에다 창 자원 하나를 만들어놓고, 그 다음에는 여러분이 원하는 때에 그 창을 화면에 표시해달라고 요청하면 서버가 알아서 그 창을 표시해줍니다. 물론 요청에 따라 화면에서 사라지게 만들 수도 있습니다.

또는 현재 나타내져 있는 창에 대하여 배경색과 테두리색을 바꾼다든지 하는 수정 작업 또한 아까의 Window 형 변수를 통해서 행할 수 있습니다.

우리는 앞으로 모든 자원에 대하여 서버 측에 자원을 만들어 놓고 우리 클라이언트 프로그램에서는 단지 그 자원에 대한 ID만을 변수에다 저장해놓고 있습니다. 그리고 원하는 때에 그 자원을 표시한다든지, 수정한다든지, 또는 없앤다든지 하는 작업을 행합니다. 그리고 나중에 정말 필요가 없어진 자원에 대해서는 서버 측에다 삭제하라고 통고합니다.

제가 이렇게 길게 설명드린 개념을 꼭 숙지해주십시요. 어떻게 보면 우리가 앞으로 하는 일의 거의 대부분을 말씀드린 것이나 다름없습니다.

2.5.3 XCreateSimpleWindow 함수에 대하여...

함수 이름 그대로 해석해보면 "간단한 창을 생성한다"이지 않습니까? 우리가 영어 문화권에 살고 있다면, 이것만큼 식은 죽 먹기가 어디 있겠습니까 만은... 간단한 창을 만드는 함수에 대하여 알아보도록 하지요.

```
함수의 원형(Prototype)

Window XCreateSimpleWindow (  Display*        display,
                              Window          parent,
                              int             x,
                              int             y,
                              unsigned int    width,
                              unsigned int    height,
                              unsigned int    border_width,
                              unsigned long   border_color,
                              unsigned long   background_color );
```

함수 한 번 정말 거창하군요. 앞으로 나오는 함수들 대부분이 거의 이렇다고 생각 하셔도 무방합니다. 정말로 많은 정보를 전달해주어야 하는 것을 보실 수 있을 겁니다. 짐 캐리처럼 숨을 크게 들이쉰 후, 다다다 쉴 새 없

이 얘기해보자면, 1번째 Display 형 포인터는 XOpenDisplay()에서 접속한 디스플레이를 가리키는 포인터이며, 2번째 Window는 현재 만들고자 하는 창의 소속되는 부모 창이며, 3번째, 4번째는 각각 x, y 좌표를 뜻하고, 5번째는 픽셀 단위의 가로폭, 6번째는 픽셀 단위의 세로폭, 그리고 7번째는 테두리의 두께, 마지막으로 8번째와 9번째는 테두리 색과 창의 기본적인 배경색을 카리키는 unsigned long 형 수치값입니다.

보통 하나의 디스플레이에서 대부분의 일을 해치우니까 1번째 인수가 상당히 귀찮을 수도 있지만, 생각해보세요, 여러분의 클라이언트 프로그램이 다중 디스플레이 접속 프로그램이 될 수도 있지 않습니까? 그러면, 여기저기 두 개 이상의 디스플레이를 간단하게 제어할 수 있습니다. 바로 요 첫번째 인수를 가지고 말입니다. 2번째, 부모 창은 무엇인가? — X 윈도우의 모든 창은 항상 어떤 창에 속해 있거나, 어떤 창들을 자기 자식(Child)으로 갖습니다. 아주 근본적으로는 X 윈도우가 뜨자마자 생기는 창은 바로 테두리도 없는 창, 루트(root)창입니다. 바로 이 창에 모든 창들이 속하는 것입니다. 어떤 프로그램이든 처음 창은 바로 루트창의 자식창이 됩니다. 그 창은 다시 자기 자식창들을 얼마든지 가질 수 있지요. 어찌 되었든간에 새로 생성되는 창은 족보(?)를 가져야 합니다. 자기 부모창은 알아야 하지 않겠어요? 여기서 잠깐 XDefaultRootWindow()라는 함수에 대하여 알아봅시다.

```
함수의 원형

Window XDefaultRootWindow ( Display *display );
```

이 녀석의 기능은 Display 형 포인터가 가리키는 서버의 기본(Default) 루트창의 창 ID를 알아다가 ID를 Window 형 변수에다 되돌려 주는 역할입니다.

우리가 만들고자 하는 창은 직접적으로 루트창의 바로 아래 자식창이 되므로, 위에서 나온 XCreateSimpleWindow()의 두번째에서 부모창 ID를 지정해주어야 하기 때문에 필요한 함수입니다.

나머지는 자명하므로 8번째, 9번째 색상 지정 부분에 대한 설명만 드립니다.

X 윈도우에서 색상은 또한 서버가 제공하는 하나의 자원으로서 서버가 관리를 하고 있습니다. 따라서 우리가 생각하는 것 만큼 쉽게 색상을 쓰거나 할 수는 없습니다. 혹시 여러분은 XV 와 같은 그래픽 프로그램을 쓰면서 그래픽 화면의 질이 어떨 때는 다르게 나타나는 것을 경험해보신 적 있습니까? 그 경우엔 XV가 원하는 만큼의 색상 자원을 서버에게서 공급받지 못하기 때문입니다. 어떤 이유에서든...

예에서 드러나듯 그냥 검정과 흰색을 지칭하는 쉬운 숫자를 쓰는 게 아니라, BlackPixel(), WhitePixel()이라는 매크로를 썼습니다. 이번 강좌에서는 색상을 쓰는 과정이 그렇게 간단하지는 않다는 사실만 기억해주시기 바랍니다. 조만간에 색상에 대한 얘기가 이어집니다. 그 때까지만 참아주십시요.

2.5.4 XMapWindow()

이 녀석은 무엇인가 하면, 위에서 서버측에 만들라고 통보한 창을 실제로 나타내라는 지시를 서버에 보내는 녀석입니다. 창의 생성과 표시는 별개의 과정이지요.

```
함수의 원형

XMapWindow ( Display *display, Window w );
```

특정 디스플레이 서버에서 w라는 창을 화면에 실제로 표시합니다. 여기서 영어 Map은 동사로서 "지도로 그리다","배치하다"의 뜻입니다. 제가 다니는 지질과학과에서는 지질도 그리는 것을 매핑(Mapping)한다라고 말합니다.

2.5.5 XFlush()

아니, 이건 또 뭘까요? 플러쉬(flush)는 화장실에서 볼 일을 다 본 후, 변기의 물을 내리는 것을 말합니다. 한꺼번에 물을 좍 흘려보내는 그런 일입니다. X 윈도우에 뭐 그런 일이 필요하냐구요? 거참 이상하군요.

C 프로그램을 공부해보시면, printf와 같은 많은 출력문들이 실행 직후에 출력이 이루어지는 것은 아니라는 사실을 아실 겁니다. 실제는 매번 출력 명령을 받을 때 마다 출력을 하는 것보다는 한꺼번에 버퍼에 모아놓고 때가 되면 단 한 번의 실행으로 효율성을 높이는 기술을 채택합니다. X 윈도우도 또한 그렇습니다. 그 수많은 클라이언트들로부터의 서비스 요청을 그 때 그 때 실행하게 되면 능률이 떨어지게 되므로, 서버는 버퍼를 마련하고 그 버퍼가 차거나 버퍼를 비우라는 명시적인 요구가 있을 때만 실행을 하게 됩니다.

이번 예는 X 윈도우 서버에게는 콧방귀도 뀔 필요가 업을 만큼, 자잘한 요구이기 때문에 그런 명령을 내린다고 해서 버퍼가 찰 리 만무합니다. 그러니, 강제로 '변기 물을 쫙 내려야겠죠?'

```
함수의 원형

XFlush ( Display *display );
```

2.5.6 컴파일

위 예제를 ex02.c라고 저장하셨다면,

```
cc -o ex02 ex02.c -lX11 -I/usr/X11/include -L/usr/X11/lib
```

보통 -I, -L 옵션은 주지 않으셔도 되지만 제대로 헤더화일과 라이브러리를 찾지 못할 때는 명시적으로 주시면 됩니다.

2.5.7 실행

자, 이제 X 터미널 상에서 실행을 해볼까요? 우리가 예상했던 것과는 좀 다를 것입니다. 왜냐구요? 여러분께서 fvwm과 같은 윈도우 관리자를 쓰고 계시면, 우리가 의도하지는 않았지만 그 놈이 우리가 만들고자 하는 창의 모양에 개입하고 들어오기 때문입니다. 이 부분에 대한 설명 또한 다음 번으로 미루기로 하고 단지 창이 만들어졌다는 것만 확인하고 넘어가죠. :)

종료는 실행시킨 터미널 창에서 엔터키를 한 번 눌러주시면 됩니다.

아참! 예제에서 왜 getchar()를 썼는지 한 번 생각해보세요. 그리고, 위에서 엔터 키가 아니라 스페이스바 같은 것을 누르면 왜 종료하지 않는지도 생각해보세요. 아래에서 그 답을 드리겠습니다.

2.6 창 안에 또 창을 만들어 봅시다.

바로 앞에서 아주 간단한 창 하나를 만들어 보았지요. 이번에는 그 창 안에다 다른 작은 창들을 여러 개 만들어 보기로 하겠습니다.

- 예제 3

```
#include <X11/Xlib.h>

main()
{
        Display *d ;
        Window root, p, w1, w2, w3 ;
        unsigned long Black, White ;

        d = XOpenDisplay (NULL);

        /* 주 프로그램 부분 시작 */
```

```
        root = DefaultRootWindow(d);
        Black = BlackPixel(d,0);
        White = WhitePixel(d,0);

        p  = XCreateSimpleWindow (d, root, 100, 100,
                                  600, 400, 2, Black, White );
        w1 = XCreateSimpleWindow (d, p, 50, 50,
                                  200, 150, 2, Black, White );
        w2 = XCreateSimpleWindow (d, p, 200, 100,
                                  200, 150, 2, Black, White );
        w3 = XCreateSimpleWindow (d, p, 350, 200,
                                  200, 150, 2, Black, White );
        XMapWindow(d, p);
        XMapWindow(d, w1);
        XMapWindow(d, w2);
        XMapWindow(d, w3);

        XFlush(d);
        sleep(3);

        XDestroySubwindows(d, p);
        XDestoryWindow(d, p);
        /* 주 프로그램 부분 끝 */

        XCloseDisplay (d);
}
```

이제는 예제 프로그램이 점점 복잡해지는 것 같고, 실제 프로그램 같아지는군요. 하지만 기본 뼈대는 변함이 없습니다! 제가 /* */로 표시한 부분만이 늘었을 뿐, 우리는 항상 1. 서버에 접속, 2. 서버에 요청, 3. 서버와 접속 해제 라는 기본적인 등식을 머리 속에 두고 있어야 합니다.

2.6.1 모든 창은 자식창을 여러 개 가질 수 있다.

모든 창의 부모창은 결국 루트창입니다. 그 창 안에서 모든 자식창들이 생성되고 표시됩니다. 또 그 자식창들은 바로 똑같은 과정을 통해서 자기 자신의 자식창들을 가질 수 있습니다. 마지막 단계의 자식창들도 또한 마찬가지로... 그러한 원리를 이번 예에서 확인하실 수 있습니다. 산아제한 같은 건 필요없겠죠? :)

Window 형 변수 root, p, w1, w2, w3 이렇게 다섯개를 선언해두었습니다. root는 DefaultRootWindow()라는 매크로를 통해서 현재 루트창의 ID를 기억하고 있습니다. 첫번째 XCreateSimpleWindow 함수를 통해서 루트창의 자식창으로서 p 창을 만들었습니다. 두번째 같은 함수를 통해서 이번에는 위에서 만들어 놓은 p 창을 부모창으로 하는 자식창 w1 을 만들었습니다. 같은 과정을 통해서 p 창의 자식창들 w2, w3를 서버 측에 만들어 놓았습니다. 귀찮으시더라도 창생성 함수들에 주어진 XY 좌표, 창 크기를 보시고 창이 어떻게 나타날 것인지를 예측해보십시요.

<< 잠깐 생각중... >>

컴파일 방법은 이전과 동일합니다.

이번 예제에서는 Black, White라는 unsigned long 형 변수 두 개를 선언해놓고는 전번 예제에서도 선보인 적이 있는 BlackPixel(), WhitePixel()이라는 매크로를 써서 각각 검정과 흰색을 나타내는 수치를 저장하고 있습니다. 창 생성함수가 여러 번 쓰였으므로 그 때마다 색상 지정부분에 매크로를 써주기 보다는 변수에 저장해두고 그 변수의 저장값을 사용하고자 했습니다. 별 건 아니죠.

자, 서버측에 만들어 놓은 창들을 나타내 봐야겠죠? :)

XMapWindow 함수를 통해서 디스플레이 d 에 자식창 p 를 표현합니다. 그 다음 3개의 문장도 마찬가지입니

다. 디스플레이 d 에 w1, w2, w3를 표현합니다. w1, w2, w3는 p 창의 자식창들로 등록,생성되어 있기 때문에 p 창 안에 표현됩니다. 그리고, 말씀은 안드렸지만 자식창들의 좌표계는 바로 전 부모창의 좌표계를 기준으로 합니다. 부모창의 왼쪽 윗구석이 자식창들의 원점(0,0)이 되는 것이지요.

2.6.2 자식창들을 한꺼번에 나타내자.

사실 마지막 3개의 문장은 약간 소비적인 문장입니다. 창의 생성과 표시라는 과정은 별개의 과정이므로, 일단 사용을 위해서 몇 개를 생성시켜놓은 후, 그 때 그 때 원하는 자식창들만 표시할 때는 XMapWindow 함수를 써야 하겠지만, 많은 경우 한꺼번에 자식창들을 몽땅 표시하고자 할 때가 많고, 그 경우에는 참 번거롭죠? 그래서 여기서 새로운 함수 하나를 소개시켜 드리고자 합니다.

```
함수의 원형

XMapSubwindows ( Display * d, Window w );
```

디스플레이 d 에서 창 w 의 모든 자식창들을 일괄적으로 화면에 표시합니다. 편리를 위해 제공되는 함수이지요.

앞으로도 많은 함수들이 기능은 거의 같으면서 이렇게 사소하게 다른 여러 가지들로 마련되어 있다는 것을 보실 수 있습니다.

2.6.3 쓸모없는 자원은 없애자.

```
함수의 원형 : XDestroyWindow ( Display *d, Window p );
함수의 원형 : XDestroySubwindows ( Display *d, Window p );
```

창을 파괴해버리라고(Destory) 요청하는 함수입니다. 더 이상 표시할 필요가 없어진 창들은 자원절약을 위해서라도(메모리를 차지하고 있으니까요) 제 때 없애주는 것이 필요합니다. 위에서보면 두 종류의 유사한 함수가 있는데, 첫번째 것은 지정된 하나의 창을 없애는 것이고, 두번째는 그 창이 포함하고 있는 모든 자식창들을 찾아서 없애주는 것입니다. 물론 자식창들 중에서 선별해서 없애려고 하신다면 당연히 하나씩 없애는 함수인 첫번째 것을 쓰실 수 밖에 없습니다.

[바로 전 강의 질문에 대한 답]

키보드 입력 또한 버퍼방식이므로 스페이스바를 몇 번 쳐보았자 실제로 프로그램 에는 전달이 되지 않다가 리턴키를 받으면 일시에 플러쉬됩니다.

[오늘 강의를 마치면서...]

정말 보잘 것 없는 프로그램 예라고 생각하실 지 모르겠지만, 제가 보기에는 X 윈도우 프로그래밍에서 창을 표시할 줄 아신다면 거의 반은 배우신 것이라고 확실히 말씀드릴 수 있습니다. 더욱 고차원적으로 버튼, 대화상자, 메뉴 등의 그래픽 인터페이스도 결국엔 자그마한 창들의 결합일 뿐입니다. 나중에 손쉽게 바로 우리 곁에 있는 Athena Widget 이라고 하는 편리한 인터페이스 보따리를 사용해보게 될 텐데요, 아마도 여러분 스스로 정말 색다른 인터페이스를 만들고 싶다는 생각이 들게 되실 겁니다. 엄청난 인터페이스의 자유! 이것은 X 윈도우가 다른 어떤 GUI 시스템 보다도 개방적임을 증명해보이게 될 겁니다. 기대해주세요.

이 짤막한 예를, 이 예에서 보이는 순서를 꼭 이해하시고 머리 속에 항상 기억해 주십시요. 아주 중요한 패턴 중에 하나입니다.

2.7 이제는 흑백이 아니라 칼라 창을 만들어봅시다.

오늘은 어떻게 색상(Color)을 서버로부터 얻어내고 사용할 수 있는지에 대해서 알아보기로 하겠습니다. 그 동안 계속 미뤄왔던 일이기도 하구요...

지금까지 아무 설명없이 저는 BlackPixel(), WhitePixel()등의 매크로를 예제에서 써왔습니다. 이름이 나타내듯 검정색/백색을 구하여 사용할 수 있었지요. 이제는 이것 말고도 우리가 원하는 색들을 일반적으로 어떻게 사용하는지에 대해서 알아봅니다.

색상 또한 서버가 관리하고 있는 자원의 하나입니다. 따라서 창을 만들어 달라고 서버에 요청하고, 표시하라고 요청하는 작업들과 마찬가지의 과정을 거치게 됩니다.

서버는 컬러맵(Color Map)이라고 하는 것을 가지고 있습니다. 색상을 나타내는 지도 또는 도표를 뜻하지요. 이것이 어떤 의미를 가지는지 잠깐 알아보겠습니다.

2.7.1 컬러맵이란?

컴퓨터가 사용하고 있는 출력장치 중에 비트맵 디스플레이에 속하는 CRT라는 것이 있습니다. 바로 여러분께서 눈 앞에 두고 계신 모니터이지요. 이 모니터의 원리는 여러분 모두 아시다시피 빛의 3원색에 대응하는 3개의 전자총이 모니터 표면에 발라져 있는 RGB 형광물질에 알맞게 비춰짐으로써 우리가 원하는 모든 색을 얻는 것입니다. 우리가 노랑/보라/자주색 등을 원할 때는 빛의 3원색을 알맞게 써서 전자총을 발사하지요. 그런데, 바로 여기서 노랑/보라/자주색 등을 구현할 때 과연 노랑은 RGB 를 각각 어떤 농도로 섞어야 하는가? 보라/자주색은 어떠한가를 저장하고 있는 표가 필요한데요, 바로 그것이 컬러맵입니다.

서버는 인간이 아니라 기계이기 때문에 우리가 yellow라고 말하면 어떤 것인지 전혀 알 방법이 없습니다. 단지 자기가 가지고 있는 컬러맵에서 yellow에 해당되는 것을 찾아서 모니터에게 적당한 양의 RGB 조합을 전달할 뿐입니다. 어떻습니까? 컬러맵이라는 것이 정말 중요한 위치를 차지하고 있지요? 이것이 엉망이 되면 어떻게 될까요? 컴퓨터가 정신못차리고 노랑을 원할 때 분홍을 표시할 지도 모릅니다.

서버는 기본적인 컬러맵을 가지고 있습니다.

2.7.2 원하는 색을 선택하기 위해 거치는 과정

앞으로 색상을 이용하는 어떤 작업도 다음과 같은 과정을 거쳐야 합니다.

1. 기본 컬러맵(Default ColorMap)의 ID를 알아낸다.
2. 위 기본 컬러맵으로부터 원하는 색의 픽셀값(Pixel Value)를 알아낸다.

- 예제 4

```
#include <X11/Xlib.h>

main()
{
        Display *d;
        Window Root, w;
        /* 색상값을 알아낼 때 쓸 변수들 */
        Colormap CMap;
        XColor color, color_exact;
        unsigned long Black, Blue;

        d = XOpenDisplay ( NULL );

        /* 원하는 색상 검정/파랑을 얻는 부분 */
        CMap = XDefaultColormap ( d, 0 );
        XAllocNamedColor ( d, CMap, "black", &color, &color_exact );
        Black = color.pixel;
        XAllocNamedColor ( d, CMap, "blue",  &color, &color_exact );
```

```
        Blue  = color.pixel;

        Root = DefaultRootWindow ( d );
        w = XCreateSimpleWindow ( d, Root, 100, 100, 600, 400,
                                  2, Black, Blue );

        XMapWindow ( d, w );
        XFlush ( d );
        sleep(10);

        XDestroyWindow ( d, w );
        XCloseDisplay ( d );
}
```

위의 예를 살펴보시면, 색상값(unsigned long형 값) 하나 얻는게 그렇게 쉬운 것은 아니라는 사실을 발견하셨을 겁니다.

맨 먼저 해야할 일은 컬러맵의 ID를 서버로부터 알아내는 것입니다. Colormap 형 변수 CMap을 하나 선언해두었습니다. 그리고 나서 XDefaultColormap() 함수를 써서 현재의 서버, 현재의 기본 화면(0번 화면)의 기본 컬러맵 ID를 반환받습니다.

```
함수의 원형

Colormap XDefaultColormap ( Display *d, int screen_no );
```

이제 컬러맵을 알아냈으니, 우리가 원하는 색의 픽셀값을 알아내봅시다. 이 때에는 XColor 형 변수 두 개가 필요합니다. 그리고 XAllocnamedColor 함수를 써서 서버더러 함수에 주어진 컬러맵에서 우리가 원하는 색상이름을 찾아서 그 색상 이름에 해당하는 정보를 XColor형 변수에 저장하도록 요청합니다.

```
함수의 원형

 Status XAllocNamedColor( Display *d,
                          Colormap cmap,
                          _XConst char* color_name,
                          XColor *screen_def_return,
                          XColor *exact_def_return );
```

여기서 나오는 새로운 자료형에 대해서 지금 모두 아실 필요는 없습니다. 아셔야 한다고 생각할 때 꼭 자세히 설명드리겠습니다. 함수의 이름 자체가 상당히 길고 설명적이기 때문에 그 내용을 알아보기 좋습니다. 물론 타이핑하기는 여간 힘든게 아니죠? 이 함수는 이름에서도 드러나듯이 Named Color 즉 이름이 붙어 있는 색상에 대하여, 우리가 이 함수에 주소로 전달해주는 XColor 형 변수 두 개에 알맞는 정보를 할당(Allocation)해줍니다. 하나는 스크린 기본값으로서 하드웨어적으로 가장 근사하게 실현시킬 수 있는 정보가 들어가고, 나머지 하나는 지정한 색의 정확한 정보가 들어간다고 합니다. 우리는 전자만 사용하도록 하겠습니다. 쓸만한 그래픽 프로그램을 만들기 위해서는 아주 정확하게 알아야 하겠지만 지금 제가 하고자 하는 강의는 재빨리 X 프로그래밍 전반에 대하여 훑어보는 수준이니 이해하세요.

으잉? 그런데 이름이 있는 색상이라니... 하실 분들이 계실 것 같군요. 전번 fvwm 강의 중 색상설정 강의 때 showrgb 라는 명령을 기억하십니까? 쉘 상태에서 showrgb 해보십시요. 그러면 현재 서버가 이해할 수 있는 이름붙은 색상들 목록이 RGB 값과 함께 주루룩 나타나게 될 겁니다. 바로 그 목록에 있는 것들만을 말합니다. 우리가 알고 있는 red, yellow, blue, green 등이 바로 그것입니다.

이변이 없는 한, 실패하지 않을테니 사실은 int 형 변수인 Status에는 0이 아닌 True 값이 반환되어 옵니다. 만약에 문제가 발생했다면 False, 0 입니다.

이제는 정보를 지니고 있는 XColor 형 변수를 써서 결국에 우리가 알고 싶었던 값을 알아내봅시다. XColor 형 변수는 Xlib.h 파일에 정의되어 있는 구조체 변수로서 몇 개의 색상에 대한 정보를 지니고 있다고 보시면 됩니다. 우리가 사용할것은 그 중에 unsigned long 형 멤버인 pixel 멤버입니다. 그 값을 Black 이나 Blue 변수에 저장시켜 놓으면 되는 것이지요.

2.7.3 요약

색상은 서버가 관리하는 자원입니다. 이를 사용하기 위해서는 XDefaultColormap 함수를 써서 기본적인 컬러맵 ID를 알아낸 후, XAllocNamedColor 함수를 통해서 특정 이름의 색상에 대한 XColor 정보를 알아냅니다. 그리고 나서 XColor 구조체의 pixel 멤버값을 읽어내시면 됩니다.

그런데, 색상 하나의 픽셀값을 알기 위해서 항상 이렇게 많은 과정을 거쳐야 하다니, 좀 너무한 것 같죠? 그렇다면, 여러분께서 색상이름을 전달해주면, 그 색상에 관한 픽셀값을 반환해주는 사용자 정의 함수를 작성하시면 될 겁니다. 꼭 한 번 만들어보세요. 책을 가지고 계신 분들은 아실 지도 모르겠네요. :)

2.8 만들어 놓은 창 가지고 놀기

오늘은 우리가 이미 만들어 놓은 창들에 대해서 이리저리 마음껏 놀아보겠습니다. 창의 색깔을 바꾼다든지, 창의 크기를 바꾼다든지, 없앴다가 다시 나타나게 한다든지 ... 뭐 이런 놀이를 하려고 합니다.

2.8.1 이미 만들어 놓은 창 색깔 바꾸기

일단 창을 만드실 때 배경색과 테두리색을 결정해놓기는 했지만, 사람이란게 마음이 달라질 때도 있는 것 아니겠어요? 자, 이미 표시되어 있는 창을 그대로 놔둔 채 색깔만 한 번 바꾸어 봅시다. 여러분이 마음에 드는 색을 골라보세요. 바로 전 시간에 말씀드린 색상 정보 알아내기는 기억하고 계시죠?

1. 테두리(Border)색 변화

   ```
   XSetWindowBorder ( Display *d, Window w, unsigned long border_pixel );
   ```

2. 배경(Background)색 변화

   ```
   XSetWindowBackground ( Display *d, Window w, unsigned long background_pixel );
   ```

설명이 필요한가요? 인수로 주어지는 Window 형 변수는 우리가 색을 변화시키고자 하는 대상 창을 나타내는 변수를 쓰면 되고요, 픽셀값은 여러분께서 원하시는 색의 픽셀값을 저번 강의에서처럼 구하셔서 전달하시면 되고...

자, 예제 3 번의 10초 간 지연 함수(sleep) 뒤에다 여러분께서 한번 이 함수들을 이용해서 색상을 바꾸어보십시요. 잊지 마실 것은 XFlush를 해주셔야 한다는 것입니다. XFlush를 해주시고 나서 또 다시 sleep 함수를 쓰셔야 그 결과를 확인하실 수있겠지요?

<< 생각 중... >>

여러분, 좀 어려운가요? 그러면 제가 해본 결과를 보여드리겠습니다.

예제 3 에 추가 sleep(10); 문장 다음부터입니다.

```
XAllocNamedColor( d, CMap, "green", &color, &ExactColor );
XSetWindowBorder( d, w, color.pixel );

XAllocNamedColor( d, CMap, "peachpuff", &color, &ExactColor );
XSetWindowBackground( d, w, color.pixel );

/* ??? */
```

```
XFlush ( d );

sleep(5);
```

등등...

설명을 드리자면, 전반부에서는 green 색의 픽셀값을 알아내서 테두리색을 변화시켰고, 후반부에서는 peachpuff 색으로 배경색을 바꾸라고 지시했습니다. 그리고 꼭 물내리는 것 잊지 마십시요.(flush)

하지만 이렇게 한다고 해서 원하는 결과를 얻을 수 있는 것은 아닙니다. 아직 부족한 것이 하나 있습니다. 그것은 다음과 같은 함수입니다.

3. 창 배경을 지우는 함수, 아니 다시 칠하는 함수!!!

```
XClearWindow ( Display *d, Window w );
```

함수의 이름에서 보이듯, 창을 지우는, 정확히 말해서 창의 배경을 지우는 함수입니다. 더욱 상세히 말씀드리자면, 현재의 배경을 지우고, 현재 창의 정보 중 배경색에 해당하는 색으로 다시 칠해주는 역할을 합니다. 배경을 바꾸시고자 할 때는 한 번 창을 지워주시던가 아니면 그와 똑같은 효과를 갖는 일을 해주시면 됩니다. 그것이 무엇이냐구요? 창을 UnMap 했다가 다시 Map하시면 됩니다.

따라서 제가 위에서 /* ??? */ 라고 한 부분에 다음과 같이 써넣어주십시요.

```
XClearWindow ( d , w );
```

창의 테두리와 배경색을 바꾸는 행위는 창 조작에 있어서 아주 기본적인 행위임은 물론이거니와 아주 중요한 행위이기도 합니다. 여러분이 X 윈도우 프로그램에서 매일 보시는 버튼들을 보세요. 포인터를 가져다 대면 테두리나 그 자체 색이 변하는 것을 보실 수 있습니다. 결국에는 그 버튼도 하나 또는 그 이상의 창으로 이루어져 있는 것에 불과합니다. 메뉴도 그러하고요. X 윈도우는 그야말로 아주 쬐끄만 창부터 시작해서 터미널 창과 같은 큰 창들을 포함하고 있습니다.

2.8.2 창에 대한 여러 정보를 알아냅시다.

많은 분들께서 geometry라는 단어를 보신 적이 있을 겁니다. Geometry라 함은 창에 대한 다음과 같은 정보를 말합니다. 창의 좌표(물론 창의 왼쪽 윗구석 좌표를 말하겠죠?), 폭과 높이, 테두리 두께 등.

사각형 창의 기본 요소들이라고 할 수 있는 것이죠. 창 전반에 대한 정보를 가져다 주는 역할을 하는 함수 하나를 먼저 소개하겠습니다.

```
함수의 원형 : <X11/Xlib.h>에 선언

 Status XGetGeometry ( Display *display,
                       Window drawable_object,
                       Window root_ID,
                       int *x,
                       int *y,
                       unsigned *width,
                       unsigned *height,
                       unsigned *border_width,
                       unsigned *depth
                     )
```

함수에 대한 설명 :

Geometry 정보를 가져오는(Get) 함수입니다. 첫번째 Display 형 포인터는 현재 작업 중인 디스플레이 ID이고, 두번째 Window 형 변수는 바로 우리가 geometry 정보를 알아내고자 하는 대상 창의 ID입니다. 그 다음 변수들은 하나같이 포인터 변수들입니다. 여러분도 C 프로그래밍을 해보셔서 아시겠지만, 어떤 함수든 반환값은 하나 밖에 없습니다. 하지만 그 함수를 통해 여러 개의 반환값을 가지고자 할 때 쓰는 기술이 바로 그 함수에게 주소 지정 방식의 호출(Call by Reference), 즉 포인터로 변수를 전달해주는 방식입니다. 그 함수에서 그 변수의 내용에 알맞는 값을 써주면 되니까요. 그렇습니다. 세번째 변수들부터는 우리가 알고 싶은 정보들을 가지고 올 변수들입니다. 세번째 Window형 변수는 두번째 인수에서 주어진 ID를 갖는 창이 속한 ROOT 창의 ID를 반환합니다. 저로서는 아직도 왜 세번째 인수를 주어야 하는지 그 필요성을 이해할 수는 없지만, 혼동하셔서는 안되는 것은 어떤 창이 속하는 자신의 부모창 ID가 반환되는 것이 아니라 절대적으로 ROOT 창의 ID가 반환된다는 사실입니다. 네번째 인수는 x 좌표, 다섯번째 인수는 y 좌표, 그 다음은 폭, 높이, 테두리 두께입니다. 그리고 마지막은 Depth 즉 깊이를 말하는데, 이것은 현재 창에서 색깔을 구현하는데 있어 몇 비트를 사용하고 있는가를 말해줍니다. 이 숫자가 8이면 8 비트 칼라(8bpp) 즉 256 칼라가 되겠죠? 16 bpp는 65536 칼라, 24 bpp는 16만 7천 칼라(16,777,216)를 말합니다.

자, 웃기는 사실은 위에서 알아낼 수 있는 x, y 좌표는 그 창이 속한 부모창의 좌표계를 기준으로 한다는 사실입니다. 만약 루트창이 A 창을 포함하고, A 창이 다시 B 창을 포함한다고 합시다. B 창에 대하여 XGetGeometry 하시면, 세번째 인수에는 ROOT창의 ID가, x, y 변수에는 A 창의 왼쪽 윗구석을 (0,0)으로 하는 좌표계를 쓰는 상대적인 B 창의 좌표계가 반환되어 옵니다.

덧붙이는 말 한 마디 :

마지막으로 덧붙이고자 하는 사실은 이렇습니다. 여러분께서 Xlib.h 화일에서 XGetGeometry 함수를 찾아서 실제 내용을 보시면 두번째 변수의 자료형은 Window 형이 아니라 실제로는 Drawable 즉 무엇인가를 그릴 수 있는 대상으로 되어 있다는 것을 발견하실 수 있습니다. 그것에 대한 설명은 다시 뒤로 미룹니다. 어찌 되었든 Drawable 형 자료에는 Window 형이 포함되어 있습니다.

2.8.3 이제는 창의 geometry를 바꾸어 봅시다.

창의 geometry를 바꾸는 함수들을 여기에 소개합니다.

```
함수의 원형 : <X11/Xlib.h>에 선언

XMoveWindow ( Display *display, Window w, int x, int y );
XResizeWindow ( Display *display, Window w,
                    unsigned int width, unsigned int height );
XSetWindowBorderWidth ( Display *display, Window w,
                            unsigned int border_width );
```

첫번째 함수는 창의 위치를 주어진 x, y 좌표값에 따라 이동시키는 녀석입니다. 물론 좌표는 부모창 좌표계를 기준으로 합니다. 두번째 함수는 Resize 즉 크기 변화시키는 함수로서 주어진 폭&높이 값에 따라 크기를 변화시킵니다. 세번째는 무슨 일을 하는지 설명해드려야 하나요? 네, 맞습니다. 테두리 폭을 변화시킵니다.

오늘의 모든 내용을 담고 있는 예제를 적어드리고 오늘 강의를 마칠까 합니다. 한 번 여러분들께서 분석해보세요.

- 예제 6

```
/* 창의 색상 변화와 Geometry 변화 */

#include <X11/Xlib.h>
```

```
/* 함수 선언 */
unsigned long UsrColorPixel( Display*, char* );

int main()
{
        Display *d;
        Window w0, w1, w2; /* One TopLevel Window & Two Child Windows */
        unsigned long black_pixel;
        int w_X1, w_Y1, w_X2, w_Y2;    /* 두 자식창의 위치 좌표 */
        unsigned int width, height, I; /* 자식창의 폭 & 높이 */

        /* 서버와 접속하기 그리고 변수들을 초기화 */
        d = XOpenDisplay ( NULL );

        black_pixel = BlackPixel ( d, 0 );
        width = 200; height = 100;
        w_X1 = 10; w_Y1 = 10; w_X2 = width - 10; w_Y2 = height - 10;

        /* 자, 시작해볼까요? */
        printf( "I will make windows.\n" );
        sleep( 2 );


        /* 창 하나 그리고 자식창 2개 생성 */
        w0 = XCreateSimpleWindow ( d, DefaultRootWindow( d ),
                                   100, 100, width*2, height*2, 1,
                                   black_pixel, WhitePixel( d, 0 ) );
        w1 = XCreateSimpleWindow ( d, w0, w_X1, w_Y1, width, height, 1,
                                   black_pixel, UsrColorPixel( d, "magenta" ) );
        w2 = XCreateSimpleWindow ( d, w0, w_X2, w_Y2, width, height, 3,
                                   black_pixel, UsrColorPixel( d, "blue" ) );

        /* 창과 자식창을 화면상에 표시 */
        XMapWindow( d, w0 );
        XMapSubwindows( d, w0 );
        XFlush( d );

        printf( "Unmap & Map.\n" );
        sleep( 3 );



        /* 창 하나를 UNMAP, MAP */
        XUnmapWindow( d, w1 ); XFlush( d );
        sleep( 1 );
        XMapWindow( d, w1 ); XFlush ( d );
        printf( "I will change the color of windows.\n" );
        sleep( 3 );



        /* 창 하나의 색상을 변경 */
        XSetWindowBorder( d, w2, UsrColorPixel( d, "red" ) );
        XSetWindowBackground( d, w2, UsrColorPixel( d, "green" ) );
```

```
XClearWindow( d, w2 );
XFlush( d ); sleep( 1 );
XSetWindowBackground( d, w0, UsrColorPixel( d, "yellow" ) );
XClearWindow( d, w0 );
XFlush( d );
printf( "I will move windows.\n" );
sleep( 3 );



/* 창 하나씩 이동 */
for ( ; w_X1 < width - 10 ; )
{
        XMoveWindow( d, w1, w_X1++, w_Y1 );
        XFlush( d );
}
for ( ;  w_X1 > 10 ; )
{
        XMoveWindow( d, w1, w_X1--, w_Y1 );
        XFlush( d );
}
for ( ;  w_Y2 > 10 ; )
{
        XMoveWindow( d, w2, w_X2, w_Y2-- );
        XFlush( d );
}
for ( ;  w_Y2 < height - 10 ; )
{
        XMoveWindow( d, w2, w_X2, w_Y2++ );
        XFlush( d );
}
printf( "I will change the size of windows.\n" );
sleep( 3 );



/* 창의 크기를 확장/축소 */
XResizeWindow ( d, w1, width + 100, height + 50 );
XMoveResizeWindow ( d, w2, w_X2 + 50, w_Y1 + 20,
                        width - 100, height - 50 );
/* XMoveResize !!! */
XFlush ( d );
printf( "At last, I will change the width of borders.\n" );
sleep( 3 );



/* 창의 테두리 확장 */
for ( I = 1 ; I < 20 ; I++ )
{
        XSetWindowBorderWidth ( d, w2, I );
        XFlush( d );
}
printf( "Jobs done. Merci.\n" );
sleep( 3 );
```

```
        /* 창 파괴 & 서버와의 접속 해제 */
        XUnmapWindow( d, w0 );
        XUnmapSubwindows( d, w0 );
        XDestroySubwindows( d, w0 );
        XDestroyWindow( d, w0 );

        XCloseDisplay( d );


        return 0; /* 성공적으로 프로그램을 수행 */
}


/*

  UsrColorPixel() : 주어진 이름의 색상에 대한 기본 컬러맵의 픽셀값 반환

*/

unsigned long UsrColorPixel( display, name )
Display *display;
char *name;
{
        Colormap cmap;
        XColor c0, c1;

        cmap = DefaultColormap( display, 0 );

        XAllocNamedColor( display, cmap, name, &c0, &c1 );
        /* 여기서 우리는 c1 을 아직 이용하지 않습니다. */

        return ( c0.pixel );
}
```

여기까지가 예제 6 입니다. 마지막에 사용자 정의함수 UsrColorPixel()은 바로 전 시간에 제가 문제로 내드렸던 것에 대한 답 중 하나입니다. 여러분들께서 각자 자신만의 정의함수를 만들어 놓으셨겠지요? 저랑 비교해 보십시요.

이 예제를 보시면서 한 줄 한 줄이 어떤 일을 하게 될 것인지 미리 머리 속에서 그려보시고, 번거로우시더라도 꼭 예제를 자기 손으로 타이핑하시면서(상당한 노가다죠? :)) 함수 하나하나를 익히시기 바랍니다. 모든 함수들이 왜 그렇게 씌여져야 하는가, 왜 그렇게 밖에 만들지 못했는가에 대해서도 생각해주시면 더욱 좋구요. 그 다음, 이 예제 속에는 제가 설명드리지 않은 함수가 있습니다. 그 함수는 여러분께서 힘들이지 않고 그 의미와 사용법을 아실 수 있을 거라고 생각합니다.

여러분도 잘 아실 겁니다. 프로그래밍은 짜증날 정도로 많은 실수들의 연발 속에서 그 실력이 늘어간다는 사실, 그리고 엉뚱한 실수 속에서 중요한 문제들을 파악해 나갈 수 있다는 사실 말입니다. 다음과 같은 문장이 예제에 있죠?

```
XSetWindowBackground( d, w0, UsrColorPixel( d, "yellow" ) );
```

요 문장을 /* */으로 주석문 처리하시거나 지우신 다음 컴파일하고 실행시켜 보세요. 자, 어떤 일이 일어납니까?

X 윈도우 프로그래밍 별 것 아닙니다. 창을 가지고 노는 장난이라고나 할까요?

제 3 절 휴식강의 : 리소스(Resource)

이번에는 Xlib 프로그래밍을 떠나서 잠시 기분전환을 할까 합니다. 계속 똑같은 패턴의 프로그래밍 공부를 하다보면 지치죠? 그래서 재미있는 일 한 가지를 오늘 해보려고 합니다.

X윈도는 사용자에게 무한한 자유를 주는 시스템입니다. 윈도 3.1와 윈 95만 쓰고 살아온 우리들에게는 처음에 상당히 겁먹게 하지만, 알아가면 알아 갈수록 이해하기가 그렇게 어려운 것은 아니며, 한 번 알고 나면 이것만큼 편리한 것이 없습니다. 컴퓨터를 사용한다는 것이 단순반복 작업을 실수없이 빠르게 해낸다는 기본 취지도 있지만, 컬러 디스플레이의 개발 등으로부터 시작된 컴퓨터의 멀티미디어화는 시각적 이미지에 대한 많은 관심을 가지게 해왔습니다. 특히 컴퓨터 그래픽 분야는 현대에 없어서는 안될 중요한 컴퓨터 활용 분야이지요.

X 윈도는 하나로 정의된 인터페이스 제한사항 같은 것이 없습니다. 여러분의 fvwm 윈도관리자를 다른 윈도관리자로 바꾸어주기만 하면 아주 색다른 분위기를 느낄 수 있습니다. 다양성과 함께 분위기는 달리 하면서도 모든 기능은 변함없는 일관성도 가지고 있는 것이 X 윈도입니다.

시작하는 말이 좀 길었죠? 오늘 우리는 자원(Resource)이라고 하는 X 윈도의 중요 한 개념에 대하여 알아보고, 몇 가지 재미있는 실습을 해볼까 합니다. 오늘 실습을 통해 여러분은 X 프로그램을 여러분의 취향에 맞도록 변화시키는 즐거움을 누리실 수 있습니다. 그럼, 시작해보도록 하죠.

3.1 리소스(Resource)란 무엇인가?

앞서도 간략하게 설명드렸지만, 다시금 설명을 드리겠습니다. 리소스라는 개념은 X 윈도 시스템에서 몇 가지 다른 의미로 사용되고 있습니다.

첫번째로 X Server Resource라는 개념인데, 우리가 지금 하고 있는 Xlib 프로그래밍에서도 나왔듯이, 서버가 관리하는 대상으로서의 창(window), 폰트(font), 색상(color), 그래픽 컨텍스트(GC) 등이 바로 그것들입니다. 전술한 바와 같이 클라이언트는 단지 리소스 ID 만을 가지고 이리저리 서버에게 원하는 서비스를 요청할 뿐 입니다.

두번째로 Xlib 보다 한 단계 상위의 Xt(X Toolkit) 프로그래밍을 하다보면 나오는 내용으로서 사용자가 이리저리 셋팅을 할 수 있는 위젯(Widget)의 속성이라고 말할 수 있습니다. 위젯에 관련된 색상, 폰트, 위치, 모양 등이 그것입니다. Xlib 프로그래밍을 어느 정도 끝낸 후 Athena 위젯 Set을 가지고 Xt 프로그래밍을 할 때 본격적으로 다루게 될 내용이지만, 미리 선보이고자 합니다.

X 프로그래머가 되려면 우선 거의 모든 X 유틸리티의 의미와 사용법, 그리고 바로 이 리소스에 대한 연습을 많이 해두어야 합니다.

3.2 리소스 Name과 Class

예를 들어가며 설명하는 것이 좋겠군요. 여러분이 현재 X 윈도에서 이 글을 읽고 계신다고 가정하겠습니다. 자, 명령행이든 fvwm 버튼바에서든 간에 Seyon 이라는 통신 프로그램을 실행시켜 보세요. 하나의 터미널 창이 뜨고, 또 하나 작은 창이 뜹니다. 그 작은 창의 제목은 "Seyon Command Center"입니다. 여기서 다이얼링 등의 여러 가지 세연 관련 명령을 내리는 사령부의 역할을 해낼 수 있습니다. 잘 아시는 사항일 겁니다.

텍스트 문서이다 보니 설명하기가 좀 힘들기는 하지만 되는 만큼 해보죠. 우선 세연 사령부를 자세히 보시면 전체적으로 4개의 영역으로 되어 있다는 것을 알 수 있습니다. 위에서부터 첫번째 영역은 DTR, DSR, RTS, CTS, RNG 등 모뎀 관련 상황을 보여주는 상황창입니다. 두번째 영역은 세연 프로그램이 사용자에게 보내는 메세지를 보여주는 창입니다. 세번째 영역은 사용자 정의 버튼들이 보여지는 영역 입니다. Athena 뭐 이런 문자열들이 보이는 한 줄을 말합니다. 그 다음 마지막으로 About, Help, Set, Dial 등의 버튼들이 배열되어 있는 제일 큰 사각형 영역을 확인 할 수 있습니다. 저한테만 보이나요? 사실 두번째만 제외하고 같은 색상으로 칠해져 있으니 언뜻 파악하기가 쉽지는 않습니다. 하지만 앞으로 분명해질 겁니다.

여기서 작은 버튼들을 오브젝트, 객체(Object)라고 부릅니다. Xt 프로그래밍을 하실 줄 아시는 분들은 금방 이해하시겠죠? 모르시더라도 일단 저의 설명을 끝까지 들어주신다면 이해하기 어렵지는 않을 것입니다. 네 번째 영역의 첫번째 About 이라고 씌여져 있는 버튼 객체를 한 번 살펴 보죠. 이러한 모든 객체는 세연이라는 프로그램을 Muhammad라는 사람이 만들 때, 고유한 이름을 주게 됩니다. 이 About이라고 씌여져 있는 버튼은 프로그램 상에서 about이라는 이름(Name)을 가지고 있습니다. 다음으로 Help 버튼은 help라는 이름을, Set 버튼은 set이라는 이름을, 등등 해서 Shell 버튼은 shellCommand라는 이름을 가지고 있습니다. 뭐 이름은 정확한 규칙이 있는 것이 아니라 Muhammad씨 마음대로 지어준 것입니다. 이것이 바로 버튼 객체의 이름(Name)입니다.

그런데, 자세히 볼 필요도 없이 그 버튼들은 그 위에 씌여있는 글자만 다르고, 위에서 살펴보았듯이 이름만 다르지, 본질적으로 같은 부류에 속한 녀석들이라는 것을 알 수 있습니다. 이 말은 유식하게 그 버튼들 모두 하나의 Class에 속한다고 말합니다. 요 버튼들이 모두 속한 Class는 Athena 위젯 set 중에서 Command 위젯에 해당합니다. Command라고 하는 것이 이 Class의 이름입니다.

헷갈리죠? 세상 모든 것에는 이름이 있기 마련인데... 앞으로 그냥 이름 또는 객체 이름이라고 하는 것은 프로그래머에 의해서 붙여지는 개별적인 이름(Name)을 뜻합니다. 그 다음으로 Class 또는 클래스 명은 그 클래스 전체를 부르는 대표적인 이름입니다. 워낙 헷갈리는 부분이라 X 프로그래머들은 그냥 이름은 소문자로 시작하고 클래스 명은 대문자로 시작하는 관습을 가지고 있습니다. 이 규칙을 무시하는 프로그래머는 없을 겁니다. 자기 멋대로 했다가는 X 프로그래머 세계에서 추방당하고 말 겁니다. :)

3.3 객체의 리소스

위에서 살펴본 버튼 객체는 프로그래머 또는 사용자가 자유롭게 정해줄 수 있는 몇 가지 속성을 갖는데 바로 그것을 객체의 리소스, 그냥 줄여 리소스라고 부릅니다. 앞으로 리소스라고 하면 바로 객체의 성질을 의미하는 리소스를 말합니다.

예를 들어 설명해보죠. 자, 지금 현재 열려 있는 세연을 닫으시고 한텀이나 X 텀 상에서 다음과 같이 입력해봅시다.

```
seyon -modem /dev/modem -xrm "*about.label: ABOUT"
```

-modem 옵션 뒤에 여러분의 모뎀이 달린 포트를 지정하는 화일명을 써주시면 되고 그 다음 제가 설명드리고자 하는 정말로 중요한 명령행 옵션이 여기에 있군요.

중요! 중요! 중요! -xrm 옵션

사용법: -xrm "리소스 정의 문자열"

자, 어떻게 되었습니까? About이라고 씌여있던 것이 ABOUT로 바뀌었죠?

그 다음에는 다음과 같이 해봅시다.

```
seyon -modem /dev/modem -xrm "*about.foreground: red"
```

글자색이 바뀌었지요?

Commmand 위젯 클래스에 속해 있고 이름은 about인 버튼의 라벨(Label)과 전경색을 한 번 바꾸어보았습니다. 이런 식으로 해서 우리는 우리 개성대로 원하는 세연 사령부를 만들어 낼 수 있습니다.

약간 어려워지는군요. 다음과 같이 해봅시다.

```
seyon -modem /dev/modem -xrm "*Command.foreground: Turquoise"
```

결과를 보면 모든 버튼의 전경색이 Turquoise색으로 변한 것을 알 수 있습니다. 클래스 명을 지정하면 그 클래스에 속하는 모든 객체에 대하여 설정하는 것입니다.

-xrm 옵션을 이런 일을 해낼 수 있는 녀석입니다. "리소스 정의 문자열"의 형식에 대하여 간략하게 알아봅시다.

```
"이름.이름.이름: <리소스값>"

      또는

(ApplicationClass|applicationName)*(ResourceClass|resourceName) : value
```

이렇게 알아두어도 일단은 괜찮습니다.

자, 여기서 이름은 클래스명일 수도 있고 객체의 특별한 이름이 될 수도 있습니다. 그리고 이름을 지정하는데 있어 *나 ? 문자와 같은 와일드 카드도 사용가능합니다. 위의 예에서 항상적으로 * 문자를 사용해왔지요. 이름과 이름은 마침표(.)로 분리 합니다. 이러한 이름은 몇 단계고 계속될 수 있습니다. 위에서는 3개만 보였지만 몇 개가 마침표로 분리될 지는 프로그램마다 다릅니다.

위젯이라는 객체들 간에는 위계(hierachy)라고 하는 것이 있습니다. 족보라고 보면 좋습니다. 세연에서는 다음과 같은 관계가 성립합니다. 세연 사령부는 위에서 서술한 일정 영역, 즉 버튼들을 포함할 어떤 영역을 갖습니다. 그 영역은 다시 버튼들을 가지고 있습니다. 버튼을 누르면 또 다른 메뉴가 뜬다든지 하는 것으로 보아 버튼들 각각도 무엇인가를 계속 포함하고 있다는 것을 알 수 있습니다.

이러한 모든 객체들은 세연이라는 전체 프로그램에 속해 있는데, 세연의 프로그램 이름이 seyon이고, 세연의 클래스명은 Seyon입니다. 이건 Muhammad씨가 X 프로그래밍 관습에 따라 붙여준 이름입니다. 세연에 관련된 모든 객체를 지정할 때는 항상 "Seyon." 이런 식으로 시작해야 할 것입니다. 하지만 이런 식으로 모두 써주는 경우는 드물고 * 문자 등을 써서 표현합니다. *는 모든 문자를 대표할 수 있다는 사실을 염두에 두고 이 글을 계속 읽어나가시길 바랍니다. 리소스 또한 리소스 클래스명도 있고 리소스 그냥 이름도 있습니다.

라벨도 바꾸고 색상도 바꾸시려면 -xrm 옵션을 여러 번 주시면 됩니다.

3.4 Xlib의 리소스 매니져

-xrm 옵션이 쓸모가 있기는 하나 그것을 여러번 쓰는데는 한계가 있죠? 중요한 몇 개의 리소스를 바꾸어주려고 할 때 쓰는 정도일 뿐입니다. 이것 말고 다른 메커니즘이 있습니다.

세연은 실행되자마자 /usr/lib/X11/app-defaults 디렉토리에서 자신의 클래스명에 해당하는 화일명 Seyon을 찾아서 읽습니다. 그 화일에는 세연에 대한 중요한 정보들이 들어 있습니다. 세연 사령부의 모든 셋팅이 그 안에 들어있기도 하죠. 그 디렉토리는 루트권한으로 들어가셔서 Seyon 화일을 Write 가능으로 설정한 후 다른 이름으로 잠시 바꾸어 봅시다. 세연은 엉망이 되고 맙니다. 다음으로 세연은 여러분의 .Xdefaults 셋팅에서 세연에 대한 셋팅이 있나 살펴봅니다. 마지막으로 세연은 명령행에서 -xrm 옵션을 읽어냅니다.

주의 :.xinitrc 에 xrdb -merge .Xdefaults 와 같은 문장이 있나 살펴보십시요. 또는 userresources=$HOME/.Xdefaults인가 확인해보십시요.

만약 리소스에 대한 셋팅이 서로 충돌하는 경우에는 명령행이 최우선, 그 다음으로 .Xdefaults, 그리고 나서 어플리케이션 디폴트 Seyon이 뒤따릅니다. 마찬가지로 클래스명에 대한 셋팅에 대하여, 특정 객체에 대한 셋팅이 우선합니다.

```
-xrm "*Command.foreground: red"
-xrm "*about.foregrond: blue"
```

이런 식의 셋팅이 있다면, about 버튼은 Command 클래스에 속하므로 red 속성을 가지는 것이 아니라 더욱 더 특별한 셋팅인 blue를 띠게 됩니다. - 물론 이렇게 간단히 말할 수는 없지만, 더 자세한 것은 관련서적을 참고하시기 바랍니다.

리스스에 대한 적절한 셋팅을 선택해주는 것이 바로 우리에게는 평소 보이지 않게 열심히 일하고 있는 Xlib의 "리소스 매니져"입니다.

자, 루트권한으로 /usr/lib/X11/app-defaults/Seyon을 수정해봅니다. 여기서 우리는 정말로 재미있는 일을 많이 할 수 있습니다.

3.5 어플리케이션 디폴트 화일 다루기

fvwm에서는 #로 시작하는 문장이 주석문이었지만, 여기서는 !로 시작하는 문장이 주석문으로 처리됩니다. fvwm에서 .fvwmrc 화일을 다루듯 찾기 기능을 통해 원하는 문장을 찾아서 수정해봅시다. 전체를 설명드릴 수는 없고 재미있는 것만 골라서 설명드립니다.

```
*.title: Seyon Command Center
```

어떤 클래스, 객체든 간에 title이라고 하는 리소스를 가진 녀석의 값에다 Seyon Command Center라는 문자열을 할당합니다. 이 문자열을 바꾸어주면 세연 사령탑의 타이틀바 제목이 바뀝니다.

```
*.iconName: <아이콘 이름>
```

아이콘 상태에서 나타나는 이름을 지정해줍니다. fvwm의 IconTitle 같은 셋팅으로도 가능하죠.

```
*font: <폰트>
```

*로 시작하니 모든 문자열을 대표하고 끝이 font로 끝나는 모든 리소스 셋팅에 적용됩니다. 따라서 여기서 주어지는 <폰트>는 세연이 기본적으로 사용하게 됩니다.

```
*Cursor: <커서명>
```

기본 지정값은 hand2로 되어 있죠? fvwm 마우스 커서 바꾸기와 마찬가지입니다만, 여기서는 숫자가 아니라 문자열을 사용합니다. /usr/include/X11/cursorfont.h 화일에 정의되어 있습니다. 앞에 붙은 XC_는 빼고 이름을 써줍니다. 검은 손인 hand1으로 한 번 바꿔보시는 것도...

```
*ShapeStyle:   <모양>
```

버튼의 모양을 정의해준다. <모양>에 올 수 있는 값은 /usr/include/X11/Xaw의 Command.h에 정의되어 있습니다. 가능한 값은 Rectangle, RoundedRectangle, Oval, Ellipse 이렇게 네 가지가 있습니다.

한참을 내려간 후...

*Box로 시작하는 부분을 찾아봅시다. 이것은 우리가 위에서 살펴보았던 버튼들을 포함하고 있는 사각형 영역을 말합니다. 기본적으로는 다음과 같은 줄이 없지만 삽입해봅시다.

```
*Box.background:       <색상>
```

자, 단조롭기만 했던 사각형 영역의 색상이 바뀝니다.

```
*ok.lable: OK
*cancel.lable: Cancel
*done.label: Done
*yes.label: Yes
*no.lable: No
*edit.label: Edit
```

이런 식의 줄들을 발견하셨습니까? ok, cancel, done, 등은 버튼 객체의 이름입니다. ok 버튼의 label 리소스가 OK로 되어있죠? 이 부분을 고치면 영어권이 아닌 나라 사용자가 사용하기 편리하겠지요? 프랑스 사람이라면 *yes.label을 Qui로, *no.label을 Non으로 바꿀 겁니다. 독일 사람이라면 *yes.label을 Ya (맞나요?) *no.label을 Nein으로 바꿀 겁니다. 우리는 "예", "아니오"로 하면 됩니다.

*about으로 시작하는 줄들은 about 버튼에 관한 것입니다.

*set으로 시작하는 줄들을 밑에 있죠?

Set 버튼을 누르면 보드, 패리티, 포트 등에 대한 셋팅을 하는 새로운 창이 하나 생깁니다. 그것의 이름이 uBox인 것 같습니다.

```
*set*uBox.Toggle.background: <색상>
*set*uBox.Toggle.foreground: <색상>
```

토글버튼에 해당하는 것은 Strip 8th, CTS/RTS 설정 등입니다. 전경색/배경색은 설정되지 않은 상태의 색상이며, 설정되면 전경색/배경색이 뒤바뀝니다. 나머지 줄은 모두 label에 관한 것이므로 설명드리지 않겠습니다.

```
*transfer*upload*font: <폰트>
```

업로드할 때 화일명을 입력받는 창에서 쓰는 폰트가 너무 작다고 생각하지 않으십니까? <폰트>를 큰 것으로 바꾸어 주시면 됩니다.

*edit*title: File Edit 라는 줄까지 전진합니다. 보통 에디터에 사용되는 폰트가 너무 작죠? *edit*Text*font 를 바꾸어 주시면 됩니다. 기왕에 바꾼 거 위로 올라가면 *help 셋팅이 있는데, *help*Text*font라는 줄을 넣어서 폰트를 바꾸어 봅시다.

```
!
! Message box
!
```

여기까지 가봅시다. 메시지 박스의 배경색을 바꾸어 보죠.

```
*messageBox*background: <색상>
*messageBox*foreground: <색상>

!
! Status box
!
```

모뎀 관련 상황창입니다.

```
*statusBox.background: <색상>
*statusBox.foreground: <색상> 등 등

!
! Quick Keys
!
```

여기에는 사용자 정의 버튼이 들어가는 창입니다.

```
*quickKeyBox.Command.font: <색상>
*quickKeyBox.background: <색상>

*quickKey1.visible: < on 또는 off >
*quickKey2.visible: < on 또는 off >
...
```

단축 버튼창에 쓸 데 없는 버튼들이 표시되어 있습니다. 보통 쓰지도 않고 있지요. on은 그 버튼을 보이게 하고 off는 보이지 않게 합니다.

```
*quickKey1.action: <행동>
...

*quickKey1.label: <문자열>
...
```

자, 위에서 <행동>에 들어갈 수 있는 세연 내부의 함수들은 여러 가지가 있습니다. 여기서 세연의 놀라운 기능이 돋보입니다. 자세한 사항은 세연 메뉴얼을 참고하시고요. 많이 쓰이는 것 하나만 소개합니다. 보통 버튼 하나로 ZMODEM 업로드/다운로드 버튼을 정의해서 쓰는 겁니다.

Transfer 버튼을 누르면 현재 등록되어 있는 화일 전송 정의가 리스트로 나옵니다. 저는 세연의 내부 함수 중 FileTransfer()를 사용하겠습니다.

```
사용 형식 : FileTransfer( <전송 프로토콜 번호> [, <화일리스트> ] )
```

<전송 프로토콜 번호>가 의미하는 것은 위 리스트에 나오는 순서를 말합니다. 맨 처음 것이 1부터 시작합니다. <화일리스트>는 보통 적어주지 않습니다. 화일을 다운받는 경우에는 필요없고, 화일 전송의 경우에는 화일이름을 입력받도록 프로토콜이 잘 정의되어 있을 겁니다. 저는 XRZ 0.5를 받아서 다운프로그램으로 쓰고 있는데요, 그것을 G E T 이라는 이름의 단축버튼으로 정의해서 씁니다. 시간이 된다면 XSZ를 만들고는 싶지만 소스분석만 해놓고 매일 미루기만 하네요.

- 예

```
*quickKey1.visible: on
*quickKey1.lable: G  E  T
*quickKey1.action: FileTransfer(1); Beep();
*quickKey?.font: 8x16
```

참고로 저의 화일 전송 정의 리스트 1 번은 "Xrz3D" "$xrz" No 입니다.

마지막으로 Fun message에 대하여 알아봅시다.

```
*funMessages: \
        "Hi there! Hellooooooooooo!" \
        ...
        "I've fallen and I can't get up!"
```

여러 개의 문자열을 적어주는데 줄을 바꿀 때 \ 문자를 사용한 것에 유의합시다.

3.6 사용자 리소스 화일 .Xdefaults

앞에서 달리 세연의 클래스명 Seyon을 앞에다 붙여주시기 바랍니다. 저의 예를 보여드리는 것으로 끝을 맺겠습니다. 세연 그 자체에 대하여 설명을 한다는 것이 웬만한 분량 가지고는 엄두도 못낼 만큼 많아서 ... 제가 세연까지 다룰 수는 없을 것 같습니다. 다른 파워 리눅서분들께서 해주시면 제게도 도움이 되련만...

화일 내용을 갱신하신 후, 텀 상에서 xrdb -merge .Xdefaults 하시면 X 를 종료하거나 하지 않아도 갱신된 리소스 셋팅을 확인하실 수 있습니다.

```
! --- Seyon ---
Seyon.modems: /dev/modem
Seyon.vt100.background: DarkBlue
Seyon.vt100.foreground: white
```

```
Seyon.*.pointerShape: pencil
Seyon.dialDelay:      0
Seyon.idleGuardInterval:      300
Seyon.autoZmodem:     off
Seyon.metaKeyTranslation:     off
Seyon.captureFile:    cap
Seyon.showFunMessages:        on
Seyon.dialRepeat:     100
Seyon.autoZmodemAction:       $xrz
Seyon.modemHangupString:      ATH0^M
Seyon.idleGuard:      on
Seyon.hangupConfirm:  off
Seyon.rtsctsFlowControl:      on
Seyon.defaultBPS:     38400
Seyon.zmodemAutoDownload:     off
Seyon.zmodemAutoDownloadCommand:      $xrz

! --- 이 부분이 리소스 셋팅을 다루고 있습니다 ---
Seyon*title: Seyon Command Centre
Seyon*iconName: My Seyon
Seyon*Cursor: hand1
Seyon*yes.label: Qui
Seyon*no.label: Non
Seyon*set*uBox.Toggle.background: Pink
Seyon*set*uBox.Toggle.foreground: SteelBlue
Seyon*transfer.label: Up/Dn
Seyon*transfer*upload*title: Which File ?
Seyon*transfer*upload*font: -adobe-courier-medium-r-normal--18-*-*-*-*-*-*-*
Seyon*transfer*upload*dialog.lable: Enter FileName
Seyon*exit.label: Quit
Seyon*messageBox*foreground: purple4
Seyon*messageBox*background: seashell1
Seyon*statusBox.background: grey
Seyon*quickKeyBox.Command.font: 9x15bold
Seyon*quickKeyBox.background: steelblue
Seyon*quickKey1.visible: on
Seyon*quickKey2.visible: on
Seyon*quickKey3.visible: off
Seyon*quickKey4.visible: off
Seyon*quickKey5.visible: off
Seyon*quickKey1.action: FileTransfer(1); Beep();
Seyon*quickKey2.action: FileTransfer(2); Beep();
Seyon*quickKey?.font: 8x16
Seyon*quickKey1.label: G  E  T
Seyon*quickKey2.label: P  U  T
Seyon*Command.background: PeachPuff
Seyon*quickKey1.background: DeepPink
Seyon*dial.background: blue
Seyon*dial.foreground: snow
Seyon*hangup.background: SeaGreen
Seyon*hangup.foreground: snow
Seyon*misc.background: Pink
Seyon*cancel.background: DeepPink
Seyon*dismiss.background: DeepPink
Seyon*exit.background: grey
```

```
Seyon*exit.foreground: black
Seyon*directory*lBox.ok.background: blue
Seyon*directory*lBox.ok.foreground: snow
```

! 참고로 화일 전송 정의 2번은 다운로드로서 "ZDown" "$sz -vv" y 입니다.

X 윈도는 사용자에게 무한한 자유를 주고 있다!

3.7 xterm 의 리소스에 관하여

xterm의 클래스명은 XTerm입니다.

man xterm 하시면 자세하게 리소스 목록이 나옵니다. 그것을 참고하시면 됩니다. 저는 몇 가지만 다루기로 하지요.

```
XTerm*cursorColor: <색상>
```

커서의 색상을 바꿉니다.

```
XTerm*font:    <폰트>
XTerm*pointerShape: <포인터명>
XTerm*scrollBar:       <on/off>        스크롤바를 표시할 것인가?
XTerm*saveLines:       <수치값>        몇 줄을 기억해두고 있을 것인가?
```

하여튼 엄청나게 많은 리소스들이 있습니다.

우리 모두가 많이 쓰는 hanterm은 Xterm을 기반으로 하신 것 아시죠? hanterm의 클래스명은 Hanterm입니다. 한텀에 대하여 리소스 셋팅을 하시려면 XTerm 부분을 Hanterm으로 바꾸어 주시면 됩니다.

3.8 멀티 리소스 셋팅 가지기

다음과 같이 해봅시다.

```
xterm-1.Font: 7x13bold
xterm-2.Font: 6x10
xterm-3.Font: fixed
```

xterm-1, xterm-2, xterm-3은 클래스명이 아니라 그냥 이름입니다. 다음과 같이 xterm을 실행시켜 보면 어떻게 될까요?

```
xterm -name "xterm-1"
xterm -name "xterm-2"
xterm -name "xterm-3"
```

예상하신 결과가 나왔습니까?

3.9 기 타

여러분이 많이 사용하는 CD 플레이어로서 workman이 있습니다. 클래스명은 Workman입니다. 대표적인 리소스는 다음과 같습니다.

```
Workman.initialVolume: <수치>
```

초기 볼륨을 정해준다. <수치>는 0부터 100 사이 초기 볼륨이 맘에 들지 않으면 정해주십시요.

```
Workman.smallButtons:  <논리값(true,false)>
```

작은 버튼을 사용할 것인가?

```
Workman.autoPlay:      < never, normal, always >
```

always를 선택하면 워크맨 실행시 시디 안에 들어있는 음악시디를 연주합니다. 자동으로 CD를 연주해 준다든지 하는 기능은 벌써부터 있어왔는데, 지금에야 새로운 기능이 생긴 것 마냥 떠들어대는 경우를 보면 한심합니다. X 윈도 시작시 항상 그렇게 하시려면 fvwm 초기화 함수에 workman을 실행시켜 주십시요. 물론 autoPlay 리소스를 always로 설정하시고요. 그 다음 Style 옵션으로 StartIconic 하시면 아이콘 상태로 시작하겠죠?

그 다음으로 xcalc를 예로 들어 보죠. 우리 모두의 xcalc는 상당히 칙칙한 계산기에 불과합니다.

클래스명은 XCalc입니다. 버튼 하나 하나에 대하여 색상을 저정할 수도 있습니다.

```
XCalc.ti.Command.background:   <색상>
XCalc.ti.Command.foreground:   <색상>
```

보통의 경우 xcalc는 TI-30 계산기를 에뮬레이트하지만 xcalc 실행옵션에 -rpn을 주면 옆으로 긴 HP-10C 계산기처럼 행동합니다. 이 경우에 위의 ti를 모두 hp로 바꾼 셋팅을 필요합니다.

TI 계산기의 버튼 하나 하나에 대한 셋팅은 XCalc.hp.button4.background 이런식으로 해줍니다. CE/C 버튼이나 AC 버튼은 색상을 바꾸어봐도 좋겠지요? XCalc.ti.button4.background 그리고 button5의 색상을 바꾸십시요.

제 4 절 X 프로그래밍 기초 : 그래픽

아직 창에 대하여 할 이야기들은 많이 남아 있지만, 썰렁한 창만 가지고 이리 저리 장난치는 것은 그만두고 당분간은 GRAPHICS 에 대하여 공부해보도록 하겠습니다.

4.1 X 윈도우 그래픽의 기본 개념

어디에(Where) 그릴 것인가?

뭐, 너무 당연한 것 아니겠어요? 창(Window) 안에다 그림을 그리겠지요. 하지만, 오직 그것만은 아니랍니다. 창 말고도 그릴 수 있는 곳이 있는데, 그것을 Pixmap 이라고 합니다. 중요한 자료형(Variable Type) 중에 Window와 Pixmap 이 두 가지가 있는데, 이 두 개를 합쳐서 자료형 Drawable이라고 합니다. Drawable? 자, 그림을 그리는 것이 가능하다는 뜻이죠? 어디서 본 것도 같죠? 바로 전번 강의 XGetGeometry 함수에서 잠깐 나왔던 녀석입니다. 당분간 계속 창에다만 그릴 것이오니, Pixmap에 대한 설명을 다음으로 미룹니다. 이렇게 미뤄도 되냐구요? 걱정하지 마십시요. Drawable 자료형이라고 되어 있는 것은 Window나 Pixmap 두 가지에 똑같이 적용되므로 새롭게 배우실 필요가 없고 이해하시기도 쉬울 겁니다.

어떤 붓을 가지고? (Graphics Context)

큰 붓, 작은 붓 ... 빨간색, 노란색, 주황색 ... 무엇인가를 그릴 때 자신의 구미에 맞게 다양한 도구를 선택할 수 있어야 하겠죠? X 윈도우 시스템은 Graphics Context 첫글자만 따서 GC라고 하는 아주 특별한 자원(Resource)을 가지고 있습니다. 요 GC라고 하는 것에 과연 어떻게(How) 그릴 것인가? 라는 문제에 대한 정보를 전적으로(!) 가지고 있습니다. 항상 자원은 서버가 독재적으로(!) 관리를 하는 모습을 여기서도 엿볼 수 있군요. :)

어떤 것을 그릴 수 있는가? (그래픽 내용)

그래픽에 기본인 점(Point), 선(Line)을 그릴 수 있습니다. 하나의 점, 하나의 선을 그리라고 서버에게 요청하는 함수들이 당연히 구비되어 있고, 여기에 개발자 편리를 위해서 여러 개의 점, 여러 개의 선을 그리는 함수, 사각형, 원호 등등을 그리도록 요청하는 함수가 또한 준비되어 있습니다. 마지막으로 텍스트 문자를 그려낼 수 있는 함수들이 있어서 원하는 폰트를 가지고 문자를 표현할 수 있습니다. X 윈도우는 GUI 체제이므로 글자 또한 쓰는 게 아니라 다른 도형과 마찬가지로 그리는 것이죠. 참고로 여러분이 슬랙을 설치할 때 PEX 라이브러리를 설치하셨다면 3차원 그래픽을 아주 쉽게 그릴 수 있습니다. 나중에 기회가 되면 약간 설명을 드릴 예정입니다만...

4.1.1 그래픽의 기본 절차

어떤 프로그래밍이든 마찬가지이겠지만, X 윈도우 프로그래밍은 어떤 일을 수행하든지 정해져 있는 몇 단계의 절차를 따라야 합니다. 이러한 절차를 머리 속에 항상 기억하실 수 있는 분들이야말로 X 윈도우 프로그래밍 도사들이 되시는 겁니다.

<기본 절차>

1. 우선 그릴 대상이 미리 준비되어 있어야 하겠죠? 우리는 XCreateSimpleWindow 함수를 통해 창을 만들어 놓으면 됩니다.
2. **중요!!!** 그 다음에는 GC라고 하는 특별한 자원을 서버에게 만들어 달라고 요청합니다. 그리고 우리는 GC의 ID만 서버에게서 반환을 받습니다. 우리가 GC 자체내용을 다 가져올 수는 없습니다. 단지! 그 ID만 가져옵니다.
3. 그리고 싶은 스타일을 정하고 GC의 특정값을 바꾸어 달라고 요청합니다. 예를 들어 선의 굵기, 점선이냐 실선이냐, 색상은 어떤 것으로 하느냐 등등
4. 그리고 싶은 도형을 선택해서 알맞는 함수를 호출하여 점이든 선이든 또는 사각형이든가에 어떤 창에 어떤 GC를 이용해서 그리라고 서버에게 요청합니다.

지금부터는 이 순서 그대로 설명을 해나갈 것입니다. 위 기본절차를 꼭 기억하세요.

4.1.2 우선 GC 를 만들자.

기본 절차 1 에 대해서는 설명드리지 않습니다. 창 만드는 거 지금까지 해왔잖아요?

```
함수의 원형 : <X11/Xlib.h>

GC XCreateGC ( Display *display,
               Drawable drw,
               unsigned long bitmask,
               XGCValues *values
             );
```

bitmask, XGCValues 형 변수 values에 대한 설명만이 필요하겠군요.

- <XGCValues 형 변수 구조체에 대한 설명>

<X11/Xlib.h>에 정의되어 있는 XGCValues구조체엔 다음과 같은 멤버들이 있습니다.

```
-------------------------------------------------------------------
#   멤버명                      비트 매스트              기본값
-------------------------------------------------------------------
```

1	int function	GCFunction	GXCopy
2	unsigned long plane_mask	GCPlaneMask	모두 1
3	unsigned long foreground	GCForeground	0
4	unsigned long background	GCBackground	1
5	int line_width	GCLineWidth	0
6	int line_style	GCLineStyle	LineSolid
7	int cap_style	GCCapStyle	CapButt
8	int join_style	GCJoinStyle	JoinMiter
9	int fill_style	GCFillStyle	FillSolid
10	int fill_rule	GCFillRule	EvenOddRule
11	int arc_mode	GCArcMode	ArcPieSlice
12	Pixmap tile	GCTile	?
13	Pixmap stipple	GCStipple	?
14	int ts_x_origin	GCTileStipXOrigin	0
15	int ts_y_origin	GCTileStipYOrigin	0
16	Font font	GCFont	?
17	int subwindow_mode	GCSubwindowMode	ClipByChildren
18	Bool graphics_exposures	GCGraphicsExposures	True
19	int clip_x_origin	GCClipXOrigin	0
20	int clip_y_origin	GCClipYOrigin	0
21	Pixmap clip_mask	GCClipMask	None
22	int dash_offset	GCDashOffset	0
23	char dashes	GCDashList	4

비트 매스크 매크로와 기본값에서의 매크로는 <X11/Xlib.h> 화일에서 자동으로 포함(#include)하는 <X11/X.h> 화일의 GRAPHICS DEFINITION 부분에 정의되어 있으니 그 부분을 찾아보시기 바랍니다. ? 표한 부분은 아직은 알 필요가 없거나 16번 Font의 경우에는 시스템마다 다른 다른 값이므로 정해지지 않은 값이라는 뜻입니다.

위 도표를 주의깊게 살펴보십시요. 멤버명과 비트 매스크명 사이에는 한 눈에도 척 알아볼 수 있는 명명규칙이 있습니다. 비트 매스명은 항상 GC로 시작하고 멤버명이 소문자인 것과 달리 항상 대문자, 그리고 단어마다 대문자를 씁니다. 멤버명에 있는 대쉬(-) 문자는 없앱니다. 멤버명을 소문자로만 썼기 때문에 단어구별을 위해 대쉬가 들어간 것에 불과하니까요. 이러한 명명 방식은 X 전반에 사용되고 있으니 소홀히 넘기지 마시기 바랍니다.

프로그램 내에서 보통 다음과 같은 과정을 통해 GC를 서버 측에 생성,등록시키고 ID를 입수합니다.

```
Display *display;      /* 디스플레이 서버의 ID */
Window window;         /* 생성될 창의 ID */
GC gc;                 /* 우리가 사용할 붓 */
XGCValues gvalue;      /* 붓의 속성을 지정할 때 그 속성을 저장해둘 변수 */

...

/* 우리가 원하는 그래픽 속성만을 셋팅한다 */
gvalue.line_width = 20;
gvalue.line_style = LineOnOffDash;
gvalue.cap_style = CapRound;
gvalue.join_style = JoinRound;          /* 주의 */

gc = XCreateGC ( display, window, GCLineWidth | GCLineStyle | GCCapStyle,
                 &gvalue );

/* 여기서 눈여겨 볼 것은 GCLineWidth | GCLineStyle | GCCapStyle 부분이다.
   우리가 XGCValues 형 변수 gvalue에 셋팅한 속성의 매스크를 OR 형식으로
```

```
    지정해주어야 한다.                                                      */
```

붓을 처음 만들 때부터 미리 몇 가지 속성을 만들어 놓고 시작하는 전형적인 방식 입니다. 비트 매스크 쓴 방식을 정말로 유심히 살펴보십시요. 그러면 한 가지 사실을 눈치채실 수 있습니다. /* 주의 */라고 한 줄에서 두 줄이 이어질 때 어떤 모양으로 이어질지에 대하여 정해주었지만 GC를 만들 때 비트 매스크에서 GCJoinStyle을 넣어주지 않았기 때문에 속성이 되지 못합니다. 어떻게 보면 당연하죠? 컴퓨터가 무슨 재주로 우리가 무슨 값을 원하는지 알겠습니다. 이러저러한 것을 지정해 두었으니 그것을 셋팅하라. 이런 식으로 자세히 알려주지 않으면 안됩니다.

4.1.3 GC의 스타일 설정하기

매번 GC를 만들 때만 속성을 정할 수 있는 것은 아닙니다. 이미 빨강 물감을 적신 붓이라 하더라도 물에 씻어 내고 파랑 물감을 적실 수도 있는 것 아니겠습니까?

```
함수의 원형 : <X11/Xlib.h>

XChangeGC (    Display *display,
               GC gc,
               unsigned long mask,
               XGCValues *gvalues
          );
```

사용법은 위에서와 같습니다. 붓의 속성을 바꾸고 싶을 때는 XGCValues 형 변수의 멤버에 적절한 값을 셋팅한 후, 이 함수를 부르면 됩니다. 비트매스크를 잘 쓰셔야겠죠?

4.1.4 이제 물감에 적신 붓으로 그려봅시다.

점 그리는 함수

```
함수 원형 : <X11/Xlib.h>

XDrawPoint( Display *display, Drawable d, GC gc, int x, int y );
XDrawPoints ( Display *display, Drawable d, GC gc, XPoint *points,
               int npoints, int mode );

typedef struct {
       short x, y;
} XPoint;
/* 참고 : XSegment, XRectangle, XArc */
```

XDrawPoints 함수는 뒤에 복수형임을 의미하는 s가 붙은 것에 유의합시다. 여러 개의 점을 XPoint 형 구조체 배열에 넣어서 한 번의 함수 부르기로 그리고자 할 때 유용합니다. 모든 스타일에 대한 셋팅은 항상 GC 형 변수가 가리키는 서버의 메모리에 있다는 것을 기억합시다. npoints 변수는 점 배열의 갯수를 지정해줍니다. mode에는 다음 두 가지가 있는데 CoordModeOrigin은 주어진 모든 좌표가 Drawable의 원점에 대한 좌표로 주어졌음을 의미하며, CoorDModePrevious는 최초의 점만 Drawable 원점에 대한 좌표이며 나머지는 바로 직전의 점에 대한 상대 좌표로 표시되어 있음을 나타내줍니다.

직선 그리는 함수

```
함수 원형 : <X11/Xlib.h>
```

```
XDrawLine( Display *display, Drawable d, GC gc, int x1, int y1,
                                             int x2, int y2 );
XDrawLines( Display *display, Drawable d, GC gc, XPoints *points,
                              int npoints, int mode );
XDrawSegments( Display *display, Drawable d, GC gc, XSegment *segments,
                                                     int nsegments );
```

첫번째 함수는 Drawable 내에서 (x1, y1)에서 (x2, y2) 좌표까지 선을 긋는 아주 일반적인 함수입니다. 두번째 함수는 연필 떼지 않고 그리기를 말합니다. XPoint 구조체 배열의 점들이 계속 이어지는 형태로 그려진다. 좌표에 대한 표기는 mode 값으로 합니다. 점 그리는 함수 XDrawPoints에서와 같습니다. 세번째 XDrawSegments는 두번째 함수와 달리 연속된 배열의 점들을 한 쌍 씩 묶어서 선을 여러 개 그린다. (x1,y1)에서 (x2,y2)에 긋고, 다시 (x3,y3)에서 (x4,y4)에 긋는 방식입니다.

선 스타일이 많으므로 Xlib에 대한 레퍼런스 PS 화일 등을 구하는 것이 좋습니다. 책을 사지 않아도 완벽한 설명서가 될 만큼 잘 만들어져 있습니다.

```
참고 함수 : XSetLineAttributes ()
            XSetDash ()
```

직사각형 그리는 함수

```
함수 원형 : <X11/Xlib.h>

XDrawRectangle ( Display *display, Drawable d, GC gc, int x, int y,
                       unsigned int width, unsigned int height );
XDrawRectangles( Display *display, Drawable d, GC gc,
                       XRectangle rectangels[], int nrectangles );
```

나머지 그래픽에 관심이 많으신 분들은 스스로 나머지 함수들을 조사하시기 바랍니다. 이것에 만족하시지 못하는 분들은 3D 그래픽 라이브러리인 PEXlib 에 대한 공부를 하시면 됩니다. 이것에 대한 내용은 Xlib 프로그래밍 수준을 넘어서고 그 자체로도 방대하기 때문에 여러분의 의지에 맡깁니다.

위에서 설명한 함수들을 가지고 다음에는 실습을 해보기로 합시다. 그리고 간략하게 폰트를 가지고 문자열 나타내기를 다루어보겠습니다.

4.2 그래픽의 연습

Xlib 그래픽은 X 프로그래밍의 기초가 되어주는 아주 기본적인 기능입니다. X 윈도우는 온통 그래픽으로 가득 차 있다고 생각하시면 됩니다. 점 하나부터 모든 복잡한 도형은 결국 Xlib 의 함수를 통해서 가능한 것입니다. 제 계획으로는 엄청난 속도로 Xlib 프로그래밍을 마치고 어서 빨리 여러분이 재미있어 할 Xt 프로그래밍으로 넘어가고 싶은데요, 바로 그 곳에서 위젯을 새로 만든다든가 할 때 아주 유용하게 쓰입니다. 저를 따라 오시면 여러분만의 멋진 위젯을 만들 수 있도록 안내해드리겠습니다. 또 모르죠. 여러분 중에서 정말로 상상력이 뛰어나고 멋진 위젯이 나오면 FWF(Free Widget Foundation)의 위젯셋에 포함되어 전세계의 프로그래머들에게 알려질 수도 있습니다. 소스가 공개되어 있지 않는 다른 시스템에서는 불가능한 일 아니겠습니까?

다시 한 번 확인합니다! Xlib 그래픽은 항상 GC 라고 하는 리소스를 가지고 합니다. 그 GC는 그래픽에 관한 여러 정보를 지니고 있는 녀석인데요, 당연히 서버 측에다 GC 생성을 의뢰하고 우리는 그 GC의 ID만을 받아옵니다. 이것은 X 윈도우의 네트워크 지향적 특성에서 나오는 것으로서, 네트워크를 통해서 최소의 정보를 전하면서도 원하는 기능을 얻고자 하기 때문입니다. 예를 들어 매번 red 라는 색상으로 선을 그으라는 100번의 명령이 있다면, red로 그리라고는 명령이 계속 네트워크를 타고 서버로 전해져야 할 것입니다. 여러 가지 주문사항을 미리 등록해놓고 간단히 그 ID만을 전송하는 방식이 훨씬 네트워크의 로드를 줄이는 방법이 되겠죠? 물론 죽어나는 것은 항상 서버입니다. :)

이제부터 그래픽을 연습해봅시다

- 예제 7

```
#include <X11/Xlib.h>

/* 색상 이름에 해당하는 픽셀값 반환하는 함수 */
unsigned long UsrColorPixel( Display*, char* );
/* 요청을 플러쉬하고 지정된 초만큼 지연시키는 함수 */
void    Pause( Display*, int );

static XPoint points[] = {{ 200, 50 },{ 100, 150 },{ 300, 150 },{ 200, 50 }};
static XSegment segs[] = {{ 10, 10, 390, 190 },{ 10, 190, 390, 10 }};
static char dash_list[] = { 20, 5, 10, 5 };

int main()
{
        Display *d;
        Window w; /* One TopLevel Window & Two Child Windows */
        GC gc;
        XGCValues gv;

        /* 서버와 접속하기 그리고 변수들을 초기화 */
        d = XOpenDisplay ( NULL );

        /* 자, 시작해볼까요? */

        w = XCreateSimpleWindow ( d, DefaultRootWindow( d ),
                                  100, 100, 400, 200, 1,
                                  BlackPixel(d,0), WhitePixel(d,0) );
        XMapWindow( d, w );
        Pause( d, 2 );



/*
 *      [1] 점(Point)을 찍어봅니다. XDrawPoint(s) 함수 사용
 */
        gv.foreground = UsrColorPixel( d, "red" );
        gc = XCreateGC( d, w, GCForeground, &gv );
        XDrawPoint ( d, w, gc, 200, 100 );
        Pause( d, 3 );

        gv.foreground = UsrColorPixel( d, "blue" );
        XChangeGC ( d, gc, GCForeground, &gv );
        XDrawPoints ( d, w, gc, points, 3, CoordModeOrigin );
        Pause( d, 3 );

/*
 *      고해상도에서는 뭐 거의 안보일 정도로 RED색의 작은 점 하나,
 *      그리고 나서 BLUE색의 점 3개를 찍고 있습니다. 정말로 집중해서
 *      보서야 할 겁니다. 모니터가 더러우면 먼지와 구별이 안될지도... :)
 */
        XClearWindow( d, w );
        {
            int i, j;
            for ( j = 0; j < 200 ; j++ )
```

```
                for ( i = 0; i < 400 ; i++ )
                {
                    XDrawPoint( d, w, gc, i, j );
                    XFlush ( d );
                }
        }
        Pause( d, 3 );

/*
 *      BLUE 점으로 창 하나 가득히 칠해봅니다. XFlush 함수를 고의로 써서
 *      그 과정을 알아볼 수 있도록 하였습니다. 생각보다 빠르게 찍힐 겁니다.
 *      리눅스는 완전한 의미의 32비트 OS 이니까요.
 */



/*
 *      [2] 선(Line)을 그려봅니다.
 */

        XSetWindowBackground ( d, w, UsrColorPixel( d, "blue" ) );
        gv.foreground = WhitePixel( d, 0 );
        gv.background = UsrColorPixel( d, "red" );
        gv.line_width = 20;
        XChangeGC( d, gc, GCForeground|GCBackground|GCLineWidth, &gv );
        XDrawLine ( d, w, gc, 20, 20, 380, 180 ); /* CapButt */
        Pause ( d, 3 );

        gv.cap_style = CapRound;
        XChangeGC( d, gc, GCCapStyle, &gv );
        XClearWindow( d, w );
        XDrawLine ( d, w, gc, 380, 20, 20, 180 );
        Pause ( d, 3 );

        gv.cap_style = CapProjecting;
        XChangeGC( d, gc, GCCapStyle, &gv );
        XClearWindow( d, w );
        XDrawLine ( d, w, gc, 20, 20, 380, 180 );
        Pause ( d, 3 );

/*
 *      여기까지는 XDrawLine 함수를 사용하였습니다. 캡 스타일(Cap)이 서로
 *      어떻게 다른지 확인해 보십시요.
 */

        XSetLineAttributes ( d, gc, 10, LineDoubleDash, CapRound, JoinRound );
        XSetDashes( d, gc, 0, dash_list, 4 );
        XClearWindow( d, w );
        XDrawSegments ( d, w, gc, segs, 2 );
        Pause ( d, 3 );

        XClearWindow ( d, w );
        XSetLineAttributes ( d, gc, 15, LineOnOffDash, CapButt, JoinBevel );
        XDrawLines( d, w, gc, points, 4, CoordModeOrigin );
        Pause ( d, 3 );
```

```
/*
 *      선끝점 스타일에는 CapButt, CapRound, CapProjecting 3 가지가 있습니다.
 *      선과 선이 만나는 방식에도 세 가지가 있습니다. JoinMiter, JoinRound,
 *      그리고 JoinBevel 이 그것이죠.
 *      중요한 것! 선의 스타일에는 그냥 LineSolid, LineOnOffDash, LineDouble-
 *      Dash 가 있습니다.
 *      XSetLineAttributes(), XSetDashes()는 연구과제입니다.
 */




/*
 *      [3] 이제 2차원으로 가봅시다. 사각형부터 시작!
 */

        XClearWindow( d, w );
        XSetLineAttributes ( d, gc, 5, LineSolid, CapButt, JoinRound );
        XDrawRectangle( d, w, gc, 50 /* X */, 50 /* Y */,
                                300 /* width */, 100 /* height */ );
        Pause( d, 3 );

        {
            XRectangle rect[7];
            int i;

            for ( i = 0; i < 7 ; i ++ )
            {
                rect[i].x = 10 + i * 50; rect[i].y = 20;
                rect[i].width = 40 ; rect[i].height = 150;
            }
            XClearWindow( d, w );
            XSetLineAttributes(d,gc,1,LineDoubleDash,CapProjecting,JoinMiter);
            XDrawRectangles ( d, w, gc, rect, 7 );
            Pause ( d, 3 );
        }

/*
 *      자, XDrawRectangle(s)의 사용법을 아셨나요? 그렇다면 연구과제가 있스
 *      니다. 이들 대신에 XFillRectangle(s)를 써보십시요. 어떤 일이 일어납
 *      니까?
 */




/*
 *      [4] 둥그런 것도 그려봅시다.
 */

        {
            int i, radi;
            XArc arc[2];

            XClearWindow ( d, w );
```

```
            XSetLineAttributes(d,gc,10,LineSolid,CapRound,JoinRound);
            for ( radi = 0 ; radi <= 360 ; radi += 90 )
            {
                XDrawArc ( d, w, gc, 10, 10, 380, 180, 0 * 64, radi * 64 );
                Pause( d, 1 );
            }
            for ( i = 0 ; i < 2 ; i ++ ) {
                arc[i].x = 30 + 200 * i;
                arc[i].y = 30;
                arc[i].width = arc[i].height = 100;
                arc[i].angle1 = 0 * 64;
                arc[i].angle2 = 90 * ( i + 1 ) * 64;
            }
            XClearWindow ( d, w );
            XDrawArcs( d, w, gc, arc, 2 );
            Pause( d, 1 );
            XSetForeground( d, gc, UsrColorPixel(d,"yellow") );
            for ( i = 0 ; i < 2 ; i ++ ) {
                XClearWindow ( d, w );
                XSetArcMode ( d, gc, i == 0 ? ArcPieSlice : ArcChord );
                for ( radi = 0 ; radi <= 360 ; radi += 90 )
                {
                    XFillArc ( d, w, gc, 10, 10, 380, 180, 0 * 64, radi * 64 );
                    Pause( d, 1 );
                }
                XClearWindow( d, w );
                XFillArcs( d, w, gc, arc, 2 );
                Pause( d, 1 );
            }
        }



        /* GC, 창 파괴 & 서버와의 접속 해제 */
        XFreeGC( d, gc );
        XUnmapWindow( d, w );
        XDestroyWindow( d, w );

        XCloseDisplay( d );

        return 0;
}

/*

  UsrColorPixel() : 주어진 이름의 색상에 대한 기본 컬러맵의 픽셀값 반환

*/

unsigned long UsrColorPixel( display, name )
Display *display;
char *name;
{
        Colormap cmap;
        XColor c0, c1;
```

```
        cmap = DefaultColormap( display, 0 );

        XAllocNamedColor( display, cmap, name, &c0, &c1 );

        return ( c0.pixel );
}

void Pause( Display *d, int secs )
{
        XFlush ( d );
        sleep( secs );
}
```

컴파일은 -lX11 옵션을 주셔서 Xlib 와 링크시켜 주시면 됩니다.

최대한 Xlib 그래픽 함수들의 기능을 보이려고 노력했습니다. 하지만 위에서 보여드린 것만으로는 많이 부족합니다. 예를 들어 자유로운 다각형을 그리면서 색상을 칠할 수 있는 XFillPolygon()과 같이 유용한 함수는 설명드리지 않았습니다. 자세한 것은 여러분의 탐구에 달려 있습니다.

다음 번에는 폰트 출력하는 법에 대해서 알아보기로 합시다. 영문자 출력과 당연히 한글 출력에 관한 얘기를 해보도록 하죠. 폰트 강의를 마치면 이제 X 윈도우다운 프로그램을 짤 수 있는 "이벤트"를 다룰 수 있게 되겠군요. Xlib 도 얼마 남지 않았습니다. 힘내세요. :)

제 5 절 X 프로그래밍 기초 : 폰트

Xlib 와 함께 하는 X 프로그래밍이 재미있으신가요? Xlib 은 아주 저수순의 함수 모음입니다. 따라서 이것 자체만으로는 프로그램을 짜내기가 상당히 힘듭니다. 실제 어플 개발에 있어서는 바로 위 상위 수준인 Toolkit 수준에서 이루어지지요. 또는 많은 사람들이 사용하고 있는 실질적인 표준 모티프(Motif)가 있습니다. 하지만 제가 강조하고 싶은 것은 Toolkit과 모티프만으로는 어떤 좋은 프로그램도 만들 수가 없습니다. Xlib 가 저수준이지만 그것은 다시 말하면 강력하다는 말도 됩니다. 이 둘이 결합해야만 성능좋은 프로그램이 나온다고 확신합니다.

아직도 해야할 얘기들이 많지만, 앞으로 남은 것들에 대해서 간략하게 말씀드리고 오늘의 강의를 시작해볼까 합니다. 오늘의 폰트 출력 강의에 이어 가장 중요하다고 생각되는 이벤트 처리문제 (마우스 이벤트, 키보드 이벤트)가 남아 있습니다. X 윈도의 컬러 시스템에 대해서는 Xt (X Toolkit) 프로그래밍에 들어가서 설명드리겠습니다. 너무 오랫동안 Xlib 프로그래밍에 머무르면 흥미를 잃기 쉽기 때문입니다.

5.1 폰트 출력에 대하여 ...

어떠한 텍스트(Text)를 출력하기 위해서는 다음과 같이 3 가지 과정을 거쳐야 합니다.

1. 우선 원하는 폰트를 골라서 서버더러 그 폰트를 적재하라고 요청한다. 그러면 서버는 역시 폰트에 대한 ID만을 돌려준다. 만약 요청한 폰트가 없다면 에러가 발생한다.

2. 폰트도 결국엔 "그리는 것"이다. 따라서 그래픽 컨텍스트 GC가 이미 있어야 한다. 일단 만들어 놓은 GC에다가 폰트 ID를 지정해준다.

3. 이제는 적절한 함수를 통해서 출력한다. 이 때 우리는 2]$IDGC.1XDrawString, 2XDrawString16$.

```
간략한 함수의 사용 도표를 그려보면,

[1] XLoadFont() -->  [2] XSetFont()  -->  [3] XDrawString[16]()
```

5.2 X 윈도우의 폰트 시스템을 알아봅시다.

본인은 X 윈도우 관리자 fvwm 강의를 하면서 폰트설정 부분에서 아주 간략하게 X 윈도우의 폰트체계, XLFD 라는 명명규칙에 대해서 논했다. fvwm 강의에서는 철저하게 유저(user) 입장에서 쉽게 기술하려고 했으나, 이제 프로그래머가 되려고 하시는 여러분에게는 더욱 더 전문적인 지식이 필요하리라 봅니다.

통상 X 윈도우의 폰트는 /usr/lib/X11/fonts 에 저장되어 있습니다. 그 하부 디렉토리를 보시면 100dpi, 75dpi, Speedo, Type1, misc 등의 디렉토리를 찾으실 수 있을 겁니다. 그 디렉토리에 들어가보시면 화일명이 .pcf.Z 로 끝나는 엄청나게 많은 화일들을 보실 수 있습니다. Z 는 compress로 압축되었음을 나타내 주는 것으로서 X 윈도우 시스템은 폰트에 대한 압축을 지원합니다. 여러분도 아시겠지만, 폰트가 차지하는 디스크 용량은 어머어마 합니다. 특히 M$ 윈도우 계열에서 쓰이던 완성형 저장 방식의 폰트들은 화일 하나가 1메가를 넘어가는 것이 허다했지요. 깨끗한 문서에 대한 욕심은 다양한 폰트에 대한 요구로 이어졌구요. 압축 폰트에 대한 지원은 정말로 필요한 것임을 이해하실 수 있을 겁니다. 그 다음 fonts.dir, fonts.alias와 같은 중요한 화일을 만나실 수 있습니다.

fonts.dir

이 화일은 현재 디렉토리에 설치되어 있는 폰트 화일에 대한 XLFD 정보가 들어 있습니다. 새로운 폰트를 설치하고 나서는 mkfontdir 명령을 써서 fonts.dir 화일을 갱신합니다.

fonts.alias

XLFD 식의 명명방식으로 폰트 이름을 써준다는 것은 상당한 고역이 아닐 수 없습니다. 그래서 alias 별명을 사용합니다. 예를 들어서 우리가 가장 많이 사용하고 있는 fixed 라는 이름의 폰트는 각각의 시스템마다 실제로는 다른 폰트를 지칭하고 있을 수도 있습니다. 이러한 alias는 폰트이름을 외우기 좋게 사용자가 정의해서 쓸 수 있도록 해준다는 면도 가지고 있고, 특정 폰트이름이 모든 시스템에 항상 있는 것처럼 할 수도 있다는데 그 의의가 있습니다. 바로 위에서 예를 든 fixed라는 이름의 폰트이 좋은 예입니다. 참고로 fixed 라는 폰트 alias 가 지정되어 있지 않으면 대부분의 경우 X 자체가 뜨다가 말 겁니다. 이 화일은 자동으로 생성되는 것이 아니니 꼭 백업을 해두고 수정하시기 바랍니다.

fonts.scale

여러분이 Speedo나 Type1 디렉토리로 가보시면 fonts.scale이라는 화일을 찾으실 수 있습니다. 폰트 화일명과 그에 대한 XLFD 방식의 표기가 씌여져 있습니다. 이 두 가지 특수한 폰트에 대해서는 조금 있다가 설명드리기로 합니다. 대부분의 경우 fonts.dir 화일과 내용이 같을 겁니다.

5.3 X Logical Font Descriptio : XLFD 에 대하여

X 윈도우 시스템에서는 폰트가 갖고 있는 성질을 지시하기 위하여 14개의 필드로 이루어진 기술방식을 씁니다. 바로 이것을 XLFD라고 하지요. 다음과 같이 이루어져 있습니다.

```
-misc-fixed-medium-r-normal--10-100-75-75-c-60-iso8859-1
```

14개의 필드는 모두 - 문자로 분리되어 있습니다. 각각의 의미에 대해서 알아보도 록 할까요?

제작자 필드 (Foundry Field)

보통 맨 앞에 오는 이 필드는 * 문자를 써서 어떤 곳에서 만들었는지 상관하지 않는 경우가 많습니다. 보통 Adobe 회사가 만든 폰트의 경우에는 adobe라는 문자가 들어가 있지요. 이 밖에도 bitstream, b&h, schumacher, sun, kaist, hanyang, misc 등의 문자열이 들어가 있는 것을 종종 보실 수 있을 겁니다.

패밀리 필드 (Family Field)

이 녀석이 그 폰트의 전체적인 모양을 결정합니다. 우리가 많이 들어본 Helvetica, Times Roman, Courier 등등이 그것입니다. 우리 폰트인 경우에는 myeongjo, gothic 등이 바로 그것이죠.

무게 필드(?) (Weight Field)

medium, bold, demibold 등의 값을 가집니다. 어떤 의미인지 아실 겁니다.

경사도 필드(?) (Slant Field)

활자의 경사에 대한 지시자입니다. r 은 우리가 알고 있는 정상적인 로만체, i는 이탤릭체를 말하며, o 는 무엇일까요? Oblique 라고 해서 이것도 경사문자체인데, 뭐라고 설명드려야 할지... 이것 말고도 ri, ro 등의 값이 있는데 각각 reverse italic, reverse oblique 의 의미를 갖습니다.

폭 필드 (Setwidth Name Field)

보통의 경우 모두 normal로 설정되어 있습니다. 이외에 condensed, narrow 등의 값을 가집니다.

부가적 스타일 필드(?) (Additional Style Field)

세리프(Serif)와 산세리프(Sans Serif)의 차이 같이 부가적 스타일의 차이를 나타내 주는 필드입니다. 세리프는 보통 우리가 보는 로마자 활자로서 I, M 과 같은 글자 위 아래에 있는 가늘고 짧은 선을 말합니다. 뭔지 감이 오십니까? I 자 위 아래에 있는 옆으로 가는 선 보이시죠? 여러분이 한텀을 쓰시고 또한 기본 폰트 (-kaist-*-johab-* ...)를 쓰신다면 지금 당장 확인하실 수 있습니다. 세리프 문자란 바로 그러한 장식을 갖는 문자를 말합니다. 산세리프는 그러한 장식이 없는 문자를 말합니다. Sans 라고 하는 것이 영어 고어로 Without 이라는 의미이기 때문입니다. Sans Serif 또는 Sanserif 라고 합니다. 보통 nil 의 값, 아무 값도 없는 경우가 허다합니다. 여러분이 폰트명을 살펴보시면 항상 6번째 필드가 아무 값도 없이 – 이런 식으로 처리되어 있는 것을 보실 수 있습니다.

활자 크기 필드 (Pixel Size Field)

아주 많이 사용하는 필드로서 활자의 크기를 나타내는 필드입니다. 우리가 가장 많이 사용하는 영역은 아마도 10 20 사이가 아닐까 생각합니다. Scalable 폰트에는 해당사항이 없습니다. 자유롭게 지정해 주십시요. 그 이외의 비트맵 폰트들은 특정 크기의 활자들만이 존재합니다.

필드 (Point Size Field)

단도직입적으로 모릅니다! :)

X 방향 해상도 필드 (X Resolution Field)

Y 방향 해상도 필드 (Y Resolution Field)

각각 X, Y 방향의 해상도를 나타냅니다. 여러분이 75 dpi, 100 dpi 폰트를 설치 하셨다면 그 디렉토리에 가서 확인해보십시요. 75, 100 등의 숫자가 씌여져 있습니다. Scalable에서는 0 입니다.

공간 필드 (Space Field)

영문자에서 활자 I와 M을 생각해 봅시다. 두 문자의 폭이 다르죠? I 자는 홀쭉한 문자이고 M 자는 뚱뚱한 문자인데, 컴퓨터에서는 두 활자를 똑같은 사각형의 영역에 넣어 생각하는 경우가 대부분입니다. 그냥 출력하기 편하니까요. 그러한 일반적인 경우를 m , Monospace 라고 하며, I와 M 자와 같이 폭에 따라 간격을 조화롭게 정렬해주는 것을 p , Proportional 비례문자라고 합니다. 출판물 그리고 수준 높은 에디터의 경우 p 활자를 쓰겠지요? c 는 m 과 동일한 의미입니다. 우리가 쓰고자 하는 활자는 대부분 m 입니다. 계산이 간편하니까요.

평균 폭 필드 (Average Width Field)

글자 그대로 입니다. 평균이라고 한 이유는 위에서 말씀드리는 것에 의하면 쉽게 이해하실 수 있을 겁니다. 한글에는 적용사항 없는 것 같습니다.

등록 필드 (Registry Field)

등록 필드입니다. 여기에 들어가는 값 몇 가지를 소개함으로써 설명을 대신 하겠습니다. adobe, dec, iso646.1991, 우리가 아주 자주 보는 iso8859, johab, johabs, ksc5601.1987 등이 그것입니다. 즉 문자 세트를 의미하는 것 같죠? johab(s)은 조합형 방식의 자소 폰트이며, ksc5601.1987은 완성형 방식의 글자 폰트입니다. 일반적인 영문자는 iso8859입니다.

인코딩 필드 (Encoding Field)

휴, 마지막 필드이군요. 0, 1, 8, irv 등의 값들이 들어가 있습니다. 잘 모르는 필드입니다. 13번째 필드와 연관하여 생각하시면 좋습니다. 보통 iso8859-1, ksc5601.1987-0, iso646.1991-irv 과 같은 끄트머리 글자들을 보실 수 있습니다.

5.4 영문 텍스트 출력 연습

출력 예를 살펴보기로 합시다.

- 예제

```
#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>

int main()
{
       Display         *dpy;
       Window          w;
       Font            f;
       GC              gc;
       XSetWindowAttributes xswa;

       xswa.override_redirect = True;
       dpy = XOpenDisplay ( NULL);
       w   = XCreateSimpleWindow( dpy, RootWindow( dpy, 0 ), 50, 50,
                                      400, 300, 5, BlackPixel( dpy, 0 ),
                                              WhitePixel( dpy, 0 ) );
       XChangeWindowAttributes ( dpy, w, CWOverrideRedirect, &xswa );

       XMapWindow ( dpy, w );

       /* ----------------- 여기서부터가 본격적으로 중요한 부분 ----- */
       gc = XCreateGC( dpy, w, 0L, ( XGCValues * ) NULL );     /* [1] */
       f  = XLoadFont( dpy, "fixed" );                         /* [2] */
       XSetFont ( dpy, gc, f );                                /* [3] */

       /* 폰트 등록이 된 GC 를 가지고 전경색으로 문자열 출력 */
       XDrawString( dpy, w, gc, 100, 130,
                       "Hello, Linuxers! Never Seen :)", 16 ); /* [4] */
       /* ----------------------------------------------------------- */

       XFlush( dpy );
       getchar();

       /* 뒷처리 */
       XUnloadFont( dpy, f );
       XFreeGC( dpy, gc );
       XDestroyWindow( dpy, w );
       XCloseDisplay( dpy );
}
```

컴파일 방법은

```
gcc -o drawstring drawstring.c -L/usr/X11/lib -lX11
```

소스 예제가 좀 좋지 않더라도 이해하시기 바랍니다. :) 이미 전에 설명드린 부분은 빼고 본격적으로 다른 부분만 설명드리겠습니다.

자, 폰트도 결국은 점을 찍어서 창에다 그리는 것이므로 GC 를 필요로 합니다. 우선은 GC 를 하나 만들어야겠죠?

```
gc = XCreateGC( dpy, w, 0L, ( XGCValues* ) NULL );
```

그리고 위에서 강조했던 폰트 적재와 사용의 중간 과정을 머리 속에 떠올리십시요.

폰트 적재 (Font Loading)

```
함수 원형 :

Font XLoadFont( Display *dpy, char *font_name );
```

font_name 문자열은 바로 위에서 길게 설명드렸던 XLFD 식의 완전한 폰트이름 또는 폰트 별명(alias)를 지정해주시면 됩니다. 성공하면 Font 형 변수를 반환합니다. 실패하면 물론 NULL 이겠지요?

위에서는 fixed 라는 별명을 가진 X 윈도우의 가장 기본적인 폰트를 사용하였습니다. 그 폰트가 지정되어 있지 않다면 X 윈도우 자체가 아마 시작하지 않았을것입니다. 여러분이 아시는 폰트 이름 또는 직접 /usr/X11/lib/fonts 디렉토리에 가셔서 각 디렉토리의 fonts.dir 에 나와있는 것들을 시험해보시기 바랍니다.

한 번 멋진 아도비사의 폰트를 사용해볼까요? 그럼 fixed 라고 쓰신 부분을 바꿔 보십시요.

```
-adobe-courier-medium-o-normal--25-180-100-100-m-150-iso8859-1
```

그리고 다시 컴파일...

그런데 만약에 여러분께서 지정해주신 폰트를 X 서버(정확히는 폰트에 관하여는 X 폰트 서버 xfs 의 담당)가 찾지 못할 때, 또는 지원하지 않는 경우 에러가 발생하며 Font 형 반환값에 어떤 값이 올 지는 정확히 모릅니다.

그리고 stderr 에 다음과 같은 메세지가 출력됩니다.

```
X Error of failed request: BadName(named color or font does not exist)
  Major opcode of failed request:  45 (X_OpenFont)
  Serial number of failed request:  9
  Current serial number in output stream:  15
```

폰트 적재와 관한 한 실제 프로그래밍에서 XLoadFont 함수를 쓰지는 않는 것 같습니다. 대신 XLoadQueryFont 라고 하는 함수를 사용하지요. :) 자... 그런데 요건 여러분께 과제로 남겨야 하겠군요. 바로 다음에 설명을 드리겠습니다. 과제는 XLoadQueryFont, 이와 관련한 XFontStruct 의 구조 조사입니다. 맨페이지에 너무도 정확히 나와 있으니깐 걱정하지 마시기 바랍니다.

폰트를 GC 에 등록

```
함수 원형 :

XSetFont( Display *dpy, GC gc, Font font );
```

폰트 ID 값을 GC 정보에 수록하도록 요청합니다. 이렇게 함으로써 앞으로 출력되는 텍스트는 GC 에 지정되어 있는 전경색으로 표시되게 됩니다.

그러고 보니 GC 라고 하는 것은 그래픽에 관한 모든 정보를 지니고 있는 녀석이라고 할 수가 있네요. 다시 한 번 기억합시다! X 윈도우 그래픽에 있어서 중요한 요 GC 를.... :)

실제 텍스트 출력

```
함수 원형 :

XDrawString(display, d, gc, x, y, string, length)
            Display *display;
            Drawable d;
            GC gc;
            int x, y;
            char *string;
            int length;
```

Drawable 은 Window 형 변수나 Pixmap 형 변수을 말하죠? 자, 창 안에서의 좌표값을 주시고 문자열 string 을 넘겨 줍니다. 그리고 마지막으로 잊지 마셔야 할것은 문자열의 길이가 얼마인지를 length 에 저장해서 넣어주셔야 된다는 사실입니다.

여러분, 위 예제 프로그램 소스와 실행키신 화면에 출력된 결과에서 뭔가 이상한 점을 발견하지 않으셨습니까? :>>

예. 그렇습니다. "Hello, Linuxers! Never Seen :)" 라고 문자열을 주긴 했지만 뒷부분 즉, 'Never Seen :)' 이라고 하는 부분은 실제 출력이 되질 않습니다.

뒷처리

세상만사 시작도 중요하고 과정도 중요하지만 유종의 미를 거두는 것도 중요하다고 생각합니다. /* 뒷처리 */라고 주석을 단 부분에서는 XUnloadFont() 을 사용하고 있습니다. 그 뒤의 모든 과정도 마찬가지이지만 프로그램 종료 후 자동으로 이루어지는 과정이긴 합니다. 하지만, 여러분께서 뭔가 상용 프로그램이나 뭔가 그럴듯한 프로그램을 만드시려고 하신다면 꼭 뒷처리를 잘해주십시요.

제 6 절 X 프로그래밍 기초 : 한글폰트

자, 복습해볼까요? X 윈도우에서 하나의 문자 또는 문자열을 출력하기 위해서는 몇 가지 과정을 거쳐야 합니다.

1. 자신이 원하는 폰트명(XLFD 방식이든 별명이든 간에...)을 알고 XLoadFont 함수로 X 윈도우의 폰트 서버(xfs)로부터 Font ID를 받아옵니다.
2. 그 ID 를 가지고 그래픽 콘텍스트(GC)에 등록합니다.
3. 이제 마지막으로 XDrawString 함수를 써서 원하는 창의 원하는 위치에다 문자열을 뿌려줍니다.

아주 간단한 과정이죠? 오늘은 이것을 심화학습하겠습니다. 그리고 한글 출력에 대하여 알아보도록 하죠.

6.1 한글 출력에 관한 몇 가지 이야기

6.1.1 시작하는 지루한 이야기

컴퓨터를 쓰면서 우리를 가장 많이 괴롭히는 문제 중에 하나가 바로 한글 문제 입니다. 잘못된 정치로 인해 우리 한글 코드는 조합형/완성형 이제는 얼마전 공진청에서 만든 거의 유명무실한 KSC5700 코드, M$ 등이 주축이 되어 추진하고 있는 유니코드, M$ 사에서 한국인들을 우롱하는 듯이 버젓이 만들어 놓고 윈도그 95와 윈도그 NT 에서 쓰고 있는 확장완성형(또는 통합완성형, 이름이 어찌 되었든 실체는 같음)이 있습니다. 우리의 한글이 거의 몸살을 앓고 있다고 해도 과언이 아닙니다. 컴퓨터의 초창기(현재의 한컴사 사장 이찬진씨가 그 유명한 아래한글을 만들던 시절 이전)에 각 회사에서 왕성하게 보여주었던 한글화의 열기는 정말로 뜨거웠습니다. 그야 물론 당연히 조합형이었지요. 하지만 정부와 몇몇 관변 정부의 하수인 관변 학자들 덕분에 완성

형이라는 괴물이 탄생하였고, 그 이후 한글화의 열기는 찬물을 뒤집어 썼습니다. 게다가 한글은 어느새 기업이 소프트웨어를 팔아먹기 위한 무기가 되어버리고 말았습니다. 한글화 기술은 기업의 독점적인 라이센스나 다름 없어졌고 M$ 사의 경우에는 더더욱 한글을 볼모로 삼고 있지요.

리눅서들에게 한글화 기술을 가져다 줄 회사는 없습니다! 바로 우리들의 힘으로 이룩해야 합니다. 또한 리눅서 들은 남들이 해주길 바라지도 않습니다. 바로 이제부터 우리 스스로의 힘으로 아름다운 한글을 구현해 나갑시다. 한글을 볼모로 잡고 소비자를 우롱하는 기업들에게 본때를 보여줘야 할 시간이라고 생각합니다.

6.1.2 한글 출력! 무엇이 다른가?

모든 문제는 바로 영문 로마자는 1 바이트로 표현하고도 남는 반면, 한국어, 중국어 일본어와 같은 동양권 언어 들은 2 바이트 이상을 필요로 한다는 것입니다. 따라서 1 바이트 로마자를 출력하는 함수와는 다른 방식의 함 수를 사용하며, 문자열에 대한 약간의 가공과정을 거쳐야 합니다.

- XDrawString 함수 대신에 XDrawString16 이라는 함수 사용
- 보통의 문자열을 그대로 사용할 수 없으며, XChar2b 라는 구조체 자료형을 사용해야 한다. 따라서 보통 문자열 –> XChar2b형 배열로 바꿔주는 지루한 과정이 필요.
- 영문과 한글을 같이 출력해야 하므로 영문은 XDrawString 으로, 한글은 XDrawString16 함수를 따로 씁니다. 그렇다면 우리는 보통의 C 문자열에서 영문은 영문대로, 한글은 한글대로 추출해내는 그런 작업이 필요.

그 외에 폰트를 읽어오는 것, GC에 등록하는 것과 같은 과정은 차이 없습니다. 단지 조건이 하나 있다면 X 폰트서버(xfs, X 윈도우를 시작하면 우리 모르게 떠 있습니다 그러니 걱정하실 필요없음)가 2 바이트 출력을 지원하느냐의 문제인데, 요즘에 2 바 이트 지원하지 않는 X 서버도 있나요? :)

6.1.3 X 윈도우에서 한글 어플을 개발하는 방법

X 윈도우에서 한글 어플리케이션을 개발하는 방법은 크게 2 가지가 있습니다.

- 가장 귀찮은 방법, 그리고 가장 똑똑치 못한 방법, 하지만 거의 대부분의 한글 어플들이 채용하고 있는 방식, 잘만 하면 가장 완벽한 한글 구현 방법 : 바로 각각의 프로그램이 한글 기능을 내장하는 방식입니다. 보통 유명한 영문 프로그램의 소스에서 입력/출력에 관한 부분을 해킹하는 방식에서 많이 씌이고 있습니다.

  예) 한텀, 한Emacs, 한글 fvwm 2.* 등 절대 다수의 프로그램들
- 많은 프로그래머의 공동노력이 필요한 방법, 보통 우리가 윈도그 3.1 과 같은 곳에서 흔히 보는 방법 : 즉! X 윈도우에서의 하나로 통일된 입력/출력 방식을 (Input/Output Method) 만들어 놓고 그 기능을 어플들에서 사용하는 방식. 즉, 한글 기능을 내장하지 않고, 입출력에 대한 기본적인 규칙만 지켜주면 되는 방식입니다. 안타깝게도 상용 X 윈도우에서는 이런 한글 기능을 기술진들에 의해 만들었는지 모르지만 우리가 쓰고 있는 XFree86 에는 아직 이런 기능이 없습니다. 이것은 전문 용어로 말하자면 X 윈도우의 Internationalization 기능 즉 I18N 과 L10N을 이용한다고 말합니다. (참고로 I18N 은 Internationalization 이라는 단어에서 처음 I 와 맨끝 N 사이에 18 개의 영문자가 들어있기 때문에 그렇게 명명한 것이고, L10N은 Localization 이라는 글자에서 유래합니다) 여러분이 IBM AIX 같은 것을 써 보셨는지 모르겠군요. 이 방법은 공통된 어떤 일에 관성을 준다는 잇점이 있지만 자칫 잘못 만들면 거의 엉터리 한글이 되고 맙니다:) 현재 X11 R6 에서는 충분한 개발환경이 주어져 있습니다. 단지...

  예) ?

이외에도 우리가 윈도그에서 주로 사용하였던 한메한글처럼 HanX 라고 하는 것을 이용해서 한글을 볼 수 있도록(출력만) 하는 방식도 있습니다.

우리가 익혀야 할 것은 전자/후자의 개발 모두에 공통되는 가장 기본적인 한글출력 방법입니다. 현재 저는 후자의 개발에 노력 중입니다. 어서 빨리 XFree86 에도 상용 못지 않은 한글 시스템을 탑재하여 프로그램 개발에서 한글 처리에 할애하는 시간과 노력을 아낄 수 있도록 해야한다고 믿습니다.

6.2 한글 출력 과정

6.2.1 무엇이 또 문제인가?

현재 X 윈도우에서 사용되는 한글 폰트의 종류는 크게 3 가지라고 말할 수 있습니다. 1 번째로는 여러분이 애용하시는 한텀 조합형 폰트입니다. 2 번째는 대우폰트, 한양폰트 등의 완성형 폰트입니다. 3 번째는 같은 완성형 폰트이면서도 앞의 두 개와는 아주 사소하게 다른 처리과정을 거쳐야 하는 삼보 폰트가 있습니다.

이렇게 다양한 폰트 형식의 존재는 다양한 폰트 출력 예비작업이 필요함을 의미 합니다.

저는 일단 여기서 제일 간단한 2 번째 경우만 설명드리기로 하겠습니다. 미려한 폰트인 1 번째 한텀 조합형 폰트의 출력을 다른 한 번의 강의를 할애해야 할 만큼 많은 이야기가 필요하기 때문입니다. (사실은 삼보 폰트가 더 쉽지만서도 구할 수가 없으니...)

자, 그러면 여러분이 완성형 방식의 폰트를 가지고 있다고 생각하고 시작하겠습니다

```
함수의 원형 : <X11/Xlib.h> 에 선언

 XDrawString16(display, d, gc, x, y, string, length)
           Display *display;
           Drawable d;
           GC gc;
           int x, y;
           XChar2b *string;
           int length;
```

우리는 한글 폰트를 가지고 한글을 출력하기 위해서 XDrawString 이라는 1 바이트 언어권 함수를 사용하지 않고 끝에 16이라는 숫자가 붙은 함수를 사용하게 됩니다. 그리고 유심히 살펴보면 XDrawString 에서는 char * 형이었던 부분이 여기서는 XChar2b * 형으로 바뀌어 있죠. 우리는 여기서 새로운 자료형에 대해서 또한 알아봐야 합니다. :<

```
typedef struct {          /* normal 16 bit characters are two bytes */
          unsigned char byte1;
          unsigned char byte2;
 } XChar2b;
```

우리 말에 대한 표현은 현재 2 바이트 표현법이 가장 많이 사용되고 있으면 표준적이라고도 말할 수 있을 것 같습니다. 한글 한 글자 "한" 이라는 문자는 영문 2 바이트로 되어 있고 그것과 일대일 대응하는 것이 바로 XChar2b 형 변수입니다. 따라서 우리는 char * 형 배열을 XChar2b * 형 배열로 변환시켜 주기만 하면 됩니다. 근데 우리가 많이 가지고 있는 한양폰트나 대우폰트의 경우, 말이 쉽지 그렇게는 되진 않습니다.

다음과 같은 연산이 이루어져야 합니다. 일단 '한'이라고 하는 2 바이트 한글 한 글자가 있다고 치면...

```
char *Han = "한";
XChar2b HanX;

HanX.byte1 = *(Han)   - 0x80;        /* 주의! 0x80 값을 빼주고 있다 */
HanX.byte2 = *(Han+1) - 0x80;
```

보시다시피 그냥 한글 2 바이트와 XChar2b 형 구조체 멤버 간의 간단한 일대일 대응이 아니라, 0x80 값을 빼주어야 한다는 자잘한 중간 과정이 필요합니다. (참고로 삼보 폰트는 0x80 값은 값을 빼주지 않고 그냥 일대일 대응시킵니다.)

이제 좀 감이 오셨습니까? 알고 나니 굉장히 쉽지 않나요? :)

```
#include <X11/Xlib.h>
#include <stdio.h>

int main()
{
        Display *dpy;
        int screen;
        Window w;
        Font hanFont, engFont;
        GC gc;
        XChar2b HStrBuf[256];
        int n;

        dpy = XOpenDisplay( NULL );

        screen = DefaultScreen( dpy );
        w = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy), 0, 0, 400, 300, 2,
                        BlackPixel( dpy, screen ), WhitePixel( dpy, screen ));
        XMapWindow( dpy, w );
        XFlush(dpy);

        sleep(4);       /* 왜 넣었을까요? */

        gc = XCreateGC ( dpy, w, 0L, (XGCValues *) NULL );
        hanFont = XLoadFont ( dpy,
                        "-daewoo-mincho-*-r-normal--16-*-*-*-c-*-*-*" );

        engFont = XLoadFont ( dpy, "fixed" );

        XSetFont( dpy, gc, hanFont );
        n = convert_ks_to_display("안녕하세요리눅서", HStrBuf, 10);
        XDrawString16(dpy, w, gc, 100, 100, HStrBuf, n);

        XSetFont( dpy, gc, engFont );
        XDrawString(dpy, w, gc, 100, 150, "Welcome to Linux!", 17 );

        XFlush(dpy);
        getchar();
}

/* 한텀 소스에서 인용 */

int convert_ks_to_display(char *str, XChar2b *buf, int n)
{
  int i, j;

  for (i = 0, j = 0; i < n;) {
    buf[j].byte1 = str[i] - 0x80;
    buf[j].byte2 = str[i+1] - 0x80;
```

```
    i += 2, j++;
  }
  return j;
}
```

컴파일 방법은 똑같습니다. 이상 없이 실행되려면 여러분은 /usr/X11R6/lib/fonts 디렉토리의 하부 디렉토리 misc 디렉토리에 대우 폰트가 설치되어 있어야 합니다. 화일명은 hanglm16.pcf.Z 입니다. fonts.dir 화일을 살펴보시기 바랍니다. 현재 가지고 계시는 완성형 방식의 폰트를 아무거나 적어주시면 됩니다. 전에 설명드렸던 XLFD 방식의 표기 또는 자주 쓰는 폰트명에 대하여 fonts.alias 에다 별명을 정해주시고 그 별명을 사용하시든지 마음대로 하시면 됩니다. 참고로 위에서 사용한 영문 폰트명 fixed 는 사실 별명에 해당됩니다. fonts.alias 맨 앞에 정의되어 있습니다.

6.2.2 아직도 남은 문제가 있다!

한글로 출력을 해보시니 어떠세요? 제가 처음으로 한텀 소스를 보고 독학하면서 별것 아니지만서도 한글이 출력되니 참 기분이 좋더군요. 근데 문제는 그렇게 간단하지 않았습니다. 무엇이냐구요?

위의 소스를 잘 살펴보시기 바랍니다. 잘 보시면 한글은 한글대로 따로, 영문은 영문대로 따로 출력하고 있습니다. 이것이 무엇을 의미하는 것일까요? 예! 그렇습니다. 우리는 안타깝게도 한글과 영문을 혼용해서 쓰는 시대에 살고 있습니다. 따라서 우리는 종종 사용자로부터 다음과 같은 문자열을 받게 됩니다.

```
char *MixedString = "저의 코넷 주소는 freeyong@soback.kornet.nm.kr 입니다";
```

무엇이 문제이겠습니까?

XDrawString 함수는 오로지 1 바이트 문자권 폰트만 출력합니다. 이에 비하여 XDrawString16 함수는 반대로 2 바이트 문자권 폰트만 출력합니다. 그렇다면 위와 같이 한글과 영문이 섞여 있는 경우에는 문자열을 주욱 검색하면서 한글과 영문을 따로 출력해주어야 한다는 결론이 나옵니다. :<

이 부분이 약간의 노력을 필요로 한다고 할 수 있습니다. 영문과 한글의 조화를 맞추는 것이 현재로서는 아주 중요합니다. 어찌 하다보니 우리나라도 영어문화권이 되버린 듯한 씁쓸한 기분을 버릴 수가 없네요. 특히 컴퓨터 분야에 있어서 한글만 잘되고 영문은 대신에 꽝이 되버린다면 아무도 그거 쓰지 않습니다.

따라서 여러분은 폰트 로딩/폰트를 GC 에 등록/폰트 출력 이렇게 간단한 과정을 넘 어서서 폰트에 대해여 자유자재로 처리하실 수 있는 능력이 필요하게 됩니다.

6.2.3 한글과 영문을 구별해서 찍기

여기서부터는 좀 어려워지기 시작합니다. 제가 여기에 적어놓은 함수는 한텀 소스에서 한글처리 부분을 보고 고대로 저의 한글 fvwm 2.0.42의 한글 처리루틴으로 복사하고 약간을 수정한 것입니다. 이 모든 부분을 이해하실 필요는 없습니다. 그리고 어느 정도 실력이 되시는 분들은 이미 다 알고 계시는 내용이라고 믿습니다. 여러분이 주목하셔야 할 부분은 소스 안에서 설명을 드리겠습니다.

```
void HanXDrawString(Display *d, Drawable w, GC gc, XFontStruct *hfont,
        XFontStruct *efont, int x, int hy, int ey,
        char *str, int len, int hcode, int ftype )
{
  int n;
  char *p;
  XChar2b buf[256];

/*

  이 함수는 보통의 아스키 영문자와 한글이 혼합되어 있는 문자열을 받습니다.
```

```
    여기서는 str 이라는 변수입니다.

*/

  while (len > 0) {
    if (*str & 0x80) {
    /*
        바로 0x80 과 AND 연산을 함으로써 바이트의 8 번째 비트가 1 로 되어 있는
        가를 확인합니다. 그래서 참이면 그 바이트와 다음 바이트 두 개는 한글을
        구성한다고 할 수 있습니다.
    */

      for (p = str; len > 0 && (*p & 0x80); p += 2, len -= 2)
        ;

    /*
       여기에 for ( ) ; 라고 하는 재미있는 문장이 있습니다. 현재 한글이 처음으
       로 나타난 위치부터 어디까지 한글이 지속되는가를 체크해서 한글 문자열만
       추려내어 처리하고자 하는 겁니다.
    */

      n = HanXCharConvert(str,buf,p-str, hcode, ftype );
      XSetFont( d, gc, hfont->fid );
      XDrawString16(d,w,gc,x,hy,buf,n);
      /* 바로 여기서 XDrawString16 함수를 써서 출력하고 있습니다 */
      x += XTextWidth16(hfont,buf,n);
      /* XTextWidth16 이라는 함수는 눈여겨 보셔야겠죠? */
    }
    else {
    /* 영문일 때 처리하는 루틴입니다 */
      for (p = str; len > 0 && !(*p & 0x80); p++, len--)
        ;
      /* 마찬가지로 계속적으로 ASCII 문자만 나올 때까지 포인터 이동 */
      XSetFont( d, gc, efont->fid );
      XDrawString(d,w,gc,x,ey,str,p-str);
      x += XTextWidth(efont,str,p-str);
    }
    str = p;
  }
  XSetFont( d, gc, efont->fid );
}

/*
   위에서 쓸 데 없이 계속 XSetFont 를 쓰는 이유는 하나의 GC 에다가 한 번은
   한글 폰트를 등록했다가, 또 한 번은 영문 폰트를 등록하는 작업을 하기 때문
   입니다.
*/
```

자, 설명을 드리죠. 위에서 사용한 테크닉은 한글 프로그래밍을 하시려는 분들은 아주 기본적으로 익히셨어야 하는 내용입니다. 한글 한 글자는 2 바이트이며, 첫번째 바이트의 MSB 는 1 로 셋팅된다는 사실이죠. 그래서 한글인지 아닌지를 체크하는 가장 기본적이면서도 유일한 방법은 바로 비트매스크 연산을 해보는 겁니다. 0x80 이니깐 이진수로는 1000 0000 입니다. 이 숫자와 & 연산을 하면 둘 다 1 인 경우만 참이 되므로 과연 MSB 가 1 인가 아닌가를 식별해낼 수 있는 것입니다.

자세한 내용은 C 언어책을 보시고 비트매스크 연산을 공부하시기 바랍니다. 제가 권하고 싶은 책은 커니건 & 리치의 "C Programming Language" 입니다. 커니건과 리치가 어떤 사람인지는 아시죠? :) 모르시면 꼭 알아보세요. 기초 상식에 해당하니깐요 :)

6.2.4 폰트에 대한 중요한 정보 알아내기

한글 출력 벌써부터 이렇게 어렵군요. (근데 한글 입력은 더욱 더 황당할 겁니다)

이제부터 다루고자 하는 내용은 우리가 처할 수 있는 어떤 경우에라도 자유자재로 폰트를 처리하는 방법에 대한 것입니다. XDrawString 이나 XDrawString16 이라는 함수는 우리가 정해주지 않아도 폰트 고유의 특성에 따라 문자열을 예쁘게 한 줄로 나타내줍니다. 하지만 두 줄 이상이 되는 문자열은? 줄간격을 어떻게 할까요? 출력된 문자열은 도대체 얼마의 폭을 가지고 있는지... 등등 문자열을 여러분 마음껏 출력하기 위해서는 폰트에 대한 전반적인 정보 또는 필요에 따라 한 글자 한 글자에 대한 정보를 알아내야 합니다.

근데 폰트에 대한 정보는 어디에 있다고 했죠? 예. 맞습니다. 그것은 X 서버에 있습니다. 우리는 보통 가장 간단한 형태로서 XLoadFont 함수를 써서 폰트 ID 를 얻어옵니다.

가장 간단한 방법

```
XQueryTextExtents16(display, font_ID, string, nchars,
direction_return, font_ascent_return,
                        font_descent_return, overall_return)
      Display *display;
      XID font_ID;
      XChar2b *string;
      int nchars;
      int *direction_return;
      int *font_ascent_return, *font_descent_return;
      XCharStruct *overall_return;
```

우리가 사용하고자 하는 함수는 바로 XQueryTextExtents16 이라는 아주 긴 이름을 가진 함수입니다. 우리 비영어 문화권 사람들에게는 힘들지 몰라도 그들에게는 쉬운 함수명이겠죠? X 로 시작하는 함수는 바로 Xlib 함수임을 의미하는 것이고 Query 조사하다, TextExtents 는 텍스트의 영역 범위, 뒤에 16 을 붙인 것은 1 바이트권 문자가 아니라 2 바이트권 문자임을 나타내주고 있습니다. 일단 위에서 폰트 ID 변수형이 XID 인 것은 여러분이 생각해 보십시요. 2 바이트권 문자열이므로 XChar2b 형 배열을 사용하였고 nchars 는 그 배열의 크기를 알려줍니다 이제부터 4 개의 포인터 전달변수들이 있군요. 여러분이 생각하기에 왜 이 녀석들은 포인터로 주어졌을까요? 맞습니다. 이 변수들에다가 우리가 원하는 정보를 얻어 오려고 하는 것입니다.

direction_return 은 FontLeftToRight, FontRightToLeft 라는 정보가 들어 있습니다 영어를 좀만 읽으실 수 있다면 무슨 뜻인지 아실 겁니다. 보통의 경우 FontLeftToRight 이겠죠? 문자열의 출력이 좌 –< 우 인가? 아니면 그 반대인가 입니다.

font_ascent_return과 font_descent_return 에 대한 이야기는 한글 문자와는 다른 영문 문자만의 특성에 대해서 알아야 얘기가 될 것 같군요. 우리 문자는 이러한 개념이 없다고나 할까요? 영문자 y 를 예로 들어봅시다. 보통 우리가 중학교 처음 들어가서(윽! 요즘은 초등학교에서부터 하나요?) 영어 시간에 하는 일은 알파벳 쓰기였습니다. 일단 문방구에 가서 따로 제작된 영어 노트를 한 권 삽니다. 공책을 펼치면 마치 오선지처럼 줄이 주욱주욱 그어져 있고, 자세히 보시면 파란줄 말고 빨간줄이 그어져 있던 것이 기억나십니까? 그 빨간줄은 baseline 이라고 합니다. a b c 와 같은 문자는 모두 그 빨간줄 위에 씌여집니다. 하지만 g j y 같은 문자들은 빨간줄 아래로 내려가는 부분이 있지요? 이제 감이 오셨을 겁니다. font_ascent_return (ascent의 의미와 descent의 의미를 사전을 찾아보시면 아주 명확해집니다)는 그 빨간 baseline 위로 몇 픽셀만큼 올라가 있는가? 그리고 font_descent_return은 baseline 아래로 얼마만큼인가를 나타내줍니다. 우리 한글과는 전혀 관련이 없다고 할 수 있죠. :) 하지만! 그렇지도 않습니다. 우리가 한글과 영문을 혼용할 때 우리 한글은 영문과 조화를 이루기 위해서 항상 baseline 위에 씌여져야 하겠죠? 엉뚱하게 y 자의 밑부분과 평행하게 씌여지면 영 보기 싫어집니다.

overall_return은 글자 그대로 전반적인 정보를 가져다 줍니다. 새로운 자료형이군요. XCharStruct 라는.... 이 자료형이 어떻게 이루어져 있는지는 Xlib.h 헤더 화일에서 찾아보면 다음과 같습니다.

```
/*
 * per character font metric information.
 */
typedef struct {
    short       lbearing;       /* origin to left edge of raster */
    short       rbearing;       /* origin to right edge of raster */
    short       width;          /* advance to next char's origin */
    short       ascent;         /* baseline to top edge of raster */
    short       descent;        /* baseline to bottom edge of raster */
    unsigned short attributes;  /* per char flags (not predefined) */
} XCharStruct;
```

이 함수를 사용하는 이유는 과연 지금 출력하려고 하는 문자열이 어느 만큼의 영역을 차지할 것인가를 가늠하는 것입니다. 문자열의 높이(height)는 다음과 같습니다.

```
ascent 멤버 + descent 멤버
```

위로 얼마만큼이며, 아래로 얼마만큼인지를 더하면 됩니다. 한글은 descent = 0 :) 그래도 일관성을 위해서 항상 이렇게 써주십시요.

다음으로 문자열의 폭(width)은 어떻게 계산할까요?

```
rbearing 멤버 - lbearing 멤버   또는 그냥 width 멤버를 사용
```

1 바이트권 문자에 대해서는 XQueryTextExtents 를 씁니다.

더 좋은 방법 위의 방법은 어떤 정보를 알고자 할 때 항상 X 서버에게 요청을 해야 하므로 네트워크 로드가 많이 거리게 되며, 프로그램이 느려지는 주범이 됩니다. 그래서 프로그램다운 프로그램을 짜는 경우에는 대부분 XLoadFont 로 폰트로딩하고, XQueryTextExtents(16) 같은 함수를 사용하지 않고 다른 방법을 사용합니다.

XLoadFont 대신에 XLoadQuerFont 함수를 씁시다.

XLoadQueryFont 는 폰트를 로딩하면서 Query 한다는 의미입니다.

```
XFontStruct *XLoadQueryFont(display, name)
          Display *display;
          char *name;
```

XLoadFont에서처럼 폰트명을 전달해주는 것은 같지만, 보시는 바와 같이 반환되는 값이 Font 형이 아니라 XFontStruct 라는 자료형입니다.

```
typedef struct
  XExtData    *ext_data;      /* hook for extension to hang data */
  Font        fid;            /* Font id for this font */ /* 주목! */
  unsigned    direction;      /* hint about direction the font is painted */
  unsigned    min_char_or_byte2;/* first character */
  unsigned    max_char_or_byte2;/* last character */
  unsigned    min_byte1;      /* first row that exists */
  unsigned    max_byte1;      /* last row that exists */
```

```
  Bool        all_chars_exist;/* flag if all characters have non-zero size*/
  unsigned    default_char;   /* char to print for undefined character */
  int         n_properties;   /* how many properties there are */
  XFontProp   *properties;    /* pointer to array of additional properties*/
  XCharStruct min_bounds;     /* minimum bounds over all existing char*/
  XCharStruct max_bounds;     /* maximum bounds over all existing char*/
  XCharStruct *per_char;      /* first_char to last_char information */
  int         ascent;         /* log. extent above baseline for spacing */
  int         descent;        /* log. descent below baseline for spacing */
} XFontStruct;
```

자료형이 꽤 복잡한 편에 속하는군요. :) 주목하실 것은 폰트 ID 가 XFontStruct 자료형의 fid 멤버에 저장되어 있다는 사실입니다. 여러분이 폰트 ID 를 쓰고 싶으실 때는 fid 멤버값을 사용하세요.

XSetFont 함수를 사용하실 때는 바로 폰트 ID 를 사용하셔야겠죠?

일단 구조체에다가 몽땅 폰트 관련 정보를 가져왔으므로 이제는 어떤 정보를 알아 내기 위해 X 서버와 통신할 필요가 없다는 장점이 있습니다.

여러분이 XFontStruct 구조체를 가지고 계시면 다음과 같은 함수들을 통해서 여러가지 정보를 캐내실 수 있습니다.

- XTextWidth16

```
int XTextWidth(font_struct, string, count)
      XFontStruct *font_struct;
      char *string;
      int count;

int XTextWidth16(font_struct, string, count)
      XFontStruct *font_struct;
      XChar2b *string;
      int count;
```

  주어진 문자열(string)이 차지할 픽셀 폭을 알아냅니다.

- XTextExtents16

```
XTextExtents(font_struct, string, nchars, direction_return,
              font_ascent_return,
              font_descent_return, overall_return)
      XFontStruct *font_struct;
      char *string;
      int nchars;
      int *direction_return;
      int *font_ascent_return, *font_descent_return;
      XCharStruct *overall_return;

XTextExtents16(font_struct, string, nchars, direction_return,
                font_ascent_return,
                font_descent_return, overall_return)
      XFontStruct *font_struct;
      XChar2b *string;
      int nchars;
      int *direction_return;
      int *font_ascent_return, *font_descent_return;
      XCharStruct *overall_return;
```

내용은 XQueryTextExtents16 과 같습니다.

오늘의 설명은 여기서 마치기로 하겠습니다. 사실 폰트에 대한 설명은 그림을 보여 드리면서 하는 것이 필요하다는 생각이 드는군요. 서점에 가보시면 많은 참고 서적이 있으므로 그냥 슬쩍 폰트 Property 에 관한 부분을 잘 살펴보시기 바랍니다.

그리고 한글 프린팅을 위해서는 한텀 소스의 hangul.c 부분을 잘 훑어보시기 바랍니다. 또는 한글 fvwm 2.0.42 소스에서 libs 디렉토리에 가보시면 HangulSupport.c 라는 것이 있습니다. 그것을 살펴보시던지요...

제 7 절 X 프로그래밍 기초 : 이벤트처리

지금까지는 그 자체로는 실질적인 프로그램을 만들기에는 불가능한 가장 기본적인 개념들만을 익혀왔지만, 이제부터는 좀 달라질 것입니다. 여러분은 이제 본격적인 프로그래밍에 들어가실 준비가 되었습니다.

오늘 배울 내용은 "GUI 프로그래밍"에서 가장 핵심적인 개념인 "이벤트 event"에 대한 것입니다. 이미 윈도그 프로그래밍 같은 것을 해본 적이 있는 분들은, 그리고 특히 옛날 도그 시절부터 C 로 프로그래밍을 해보신 분들은 지금부터 말씀드리는 몇 가지 얘기에 대하여 금방 이해하실 수 있을 겁니다. 전혀 모르시는 초보분의 경우에는 일단 모든 것을 이해할 수는 없으므로 단지 윈도우 프로그래밍은 모두들 "이벤트" 처리 방식의 독특한 프로그래밍 스타일을 갖는다는 문구 하나만 기억해 주시기 바랍니다.

7.1 이벤트 처리 방식과 순차적인 처리 방식 차이점

우선 순차적인 처리 방식에 대해서 알아보도록 하겠습니다.

C 를 보통 절차언어라고 부릅니다. 모든 일에는 순서가 있다! 가장 인간적인 사고 방식을 모델로 하고 있고 결국 어떤 언어든 간에 인간의 논리는 항상 원인 –< 결과의 순서를 가지고 생각하는 방식의 변화, 철학의 변화없이는 앞으로도 항상 남아있을 것입니다. 여러분이 지금 하고 있는 모든 일이 다 절차를 갖습니다. 그리고 그것은 항상 모든 프로그래밍의 기본이며 나중에 이벤트 처리 방식이라 할 지라도 스케일을 작게 보면 결국에는 자그마한 순차 처리 방식들의 결합이라고 생각할 수도 있습니다.

보통 셸 스크립트와 같이 단 한 번의 입력을 받고 단 한 번의 처리로 끝나는 프로그램들이 전형적인 예입니다.

그에 반해서 이벤트 처리 방식은 "수많은 버튼, 언제 클릭할 지 모르는 변덕장이 마우스"라는 것을 특징으로 합니다. 윈도우 시스템이 되면서부터 프로그래머들은 정말로 많은 고민을 하기 시작했지요. 도대체 순서라는 것을 발견하기가 힘들다는 것이 문제입니다. 어떤 버튼을 누른 다음 어떤 버튼을 누를 것인지는 예측할 수가 없습니다. 순진한 사용자는 아무렇게 마음내키는 대로 버튼을 눌러댑니다. 도대체 마우스가 어디로 움직일 지도 예견할 수가 없습니다.

일이 이러하기 때문에 X 윈도우 프로그래밍 방식은 아주 판이하게 다른 어떤 일정 한 패턴을 갖습니다. 사실 나중에 보시면 아시겠지만 이벤트 처리방식이라는 것도 아주 쉽습니다. 그리고 대형 프로그램을 만들 때 좋지요. 그리고 이벤트 처리방식은 그 무엇보다도 C++ 과 같은 오브젝트 오리엔티디드 프로그래밍 언어와 제일 잘 맞습니다. 하지만 X 윈도우는 아직 C 프로그래밍 언어로 되어 있습니다. 하지만 C++ 로 짜도 상관없고 어떻게 보면 C++ 로 짜는 것이 더욱 알맞는지도 모릅니다. 실제 X 프로그래밍을 하다보면 OOP 비슷한 스타일을 많이 사용하니까요.

이벤트 처리 방식의 프로그래밍은 보통 이런 식으로 생각합니다.

버튼을 몇 개 만든다. 각 버튼이 눌려지면 수행될 독특한 함수들을 지정해준다. 그리고 나서 우리는 버튼이 눌려지기만을 기다린다. 그리고 버튼이 눌려질 때마다 예를 들어 1 번 버튼은 원을, 2 번 버튼은 사각형을, 그리고 3 번 버튼은 직선을 그리도록 한다. 즉 어떤 객체를 먼저 만들고 나서 그것에다가 특정 기능을 부여하는 방식입니다. 이런 프로그래밍 방식이기 때문에 비주얼 C 니 비주얼 베이식이니 하는 것이 가능한 것입니다. 기억하십시요. 먼저 어떤 버튼을 만듭니다. 그리고 그 버튼에다가 원하는 기능을 하나씩 부여합니다. 눌려지면 실행을 합니다. 예를 들어 "Exit" 버튼을 만들면 우리는 프로그램을 종료하면 됩니다.

7.2 복잡한 일은 X 서버가 처리합니다. 우리는 단지...

자신이 프로그래밍하는 플랫포옴을 믿는 일은 참으로 중요합니다. 윈도그95 프로그래머들은 윈도그95를 믿고, X 프로그래머들은 X 윈도우를 믿습니다. 전자는 M$가 세계 시장에서 이긴다는 것을 믿고 후자은 X 윈도우 그 자체의 능력과 가능성을 믿습니다. 윈도그95 자체를 믿지는 않겠지요? :)

X 윈도우가 뜨지 마자 정신없이 돌아다니는 마우스, 또는 언제 타이핑될 지 모르는 키보드, 이 모든 것에 대해서 여러분은 걱정할 필요가 없습니다. 마우스가 어디로 움 직이고 있는지 그리고 키보드 입력을 받으면 어떻게 할 것인지에 대해서는 X 서버가 담당합니다. 그는 마치 여러분에게 카드 패를 돌리는 딜러와도 같습니다. 여러분이 만든 프로그램은 함께 실행되고 있는 그 많은 창들과 공존합니다. 도대체 어느 창에서 마우스 클릭이 벌어졌는지에 대해서 걱정할 필요없습니다. 여러분의 창에서 클릭이 이루어지면 X 서버가 알아서 여러분의 프로그램에게 "이벤트"를 보냅니다.

심지어 나는 키보드 입력은 필요없다. 단지 마우스만으로 작동하는 프로그램이므로 "키보드 입력이라는 이벤트"는 통보하지 말아달라! 고 요청하면 X 서버는 알아서 키보드 입력은 여러분에게 주지 않습니다.

여러분은 while 문장 하나를 가지고 무한루프를 돌면서 만약 이벤트가 생긴다면 그것이 마우스 클릭인지 아니면 키보드 입력인지, 마우스 클릭이면 몇 번째 마우스 버튼인지 몇 개를 동시에 눌렀는지 등을 확인하고 알맞는 행동을 취해주면 됩니다. 개념상 프로그래밍은 쉽습니다. 여러분 프로그램의 성공은 오로지 여러분의 창조력에 달린 것이지 사실 윈도그95인지 X 윈도우인지는 상관없습니다. 심지어 꼭 X 윈도우 프로그래밍을 C 언어로 하란 법도 없습니다. Tcl/Tk 와 같은 멋진 스크립팅 언어들도 많고 그 강력함을 입증받고 있습니다. 단지 C 프로그래밍 언어로 X 윈도우 프로그래밍을 한다는 것은 여러분에게 아주 탄탄한 기초를 마련해주고 다른 어떤 프로그래밍도 쉽게 할 수 있는 자신감을 준다는 것을 말씀드리고 싶습니다.

7.3 일반적인 코드 형태를 보고 눈에 익힙시다.

이 소스는 완벽한 소스가 아니라 그냥 개념적인 소스일 뿐입니다.

```
Display display;

XEvent xe;                          /* 이벤트 처리 구조체 */
Window window1;


/* X 서버랑 접속을 합니다. XOpenDisplay 함수 사용 */

/* 간단한 창 하나를 만들어둡니다. */

/* 원하는 이벤트만을 서버에게 등록합니다 */
XSelectInput( display, window1, StructureNotifyMask | ExposureMask
                                | ButtonPressMask );

XMapWindow( display, window1 );

/* 여기서부터 본격적인 프로그램의 로직이 전개됩니다 */

while(1)        /* WHILE 문의 시작 */
{
        /* 이벤트가 생기기를 기다리고 있습니다 */
        XNextEvent( display, &xe );

        /* 이벤트가 생기면 다음 줄을 실행합니다. 이벤트가 어떤 종류의 이벤
           트인지를 식별해내고 그에 알맞는 행동을 합니다                    */
```

```
        switch ( xe.type )      {

            case Expose:
                printf("Expose 이벤트 발생\n");
                break;

            case ButtonPress:
                printf("버튼이 눌려졌습니다.\n");
                break;

            case ConfigureNotify:
                printf("창의 상태가 바뀌었습니다.\n");
                break

            case ClientMessage:
                printf("다른 클라이언트가 메세지를 보내왔습니다\n");
                break;

            default:
                printf("발생한 이벤트를 무시합니다\n");
                break;
        }
    }           /* WHILE 문의 끝 */
```

위에서 보여드린 패턴을 머리 속에 항상 있어야 합니다. 버스를 타고 가면서도 그리고 화장실에서 볼 일을 볼 때도 음악을 듣고 있어도 여자 친구를 만나서 두 눈을 바라보고 있으면서도... 잠결에도...

무엇이든 전형적인 패턴이라는 것이 있습니다. 그러한 패턴을 눈치채시는 것이 가장 중요합니다. 그럼 이제 차근차근 알아가보도록 합시다.

7.4 패턴 분석

제일 먼저 눈에 띄는 것은 바로 XEvent 라는 새로운 자료형입니다. C 프로그램이든 뭐든 간에 프로그래밍의 기본 요소는 "자료형과 알고리즘"입니다. 후자는 여러분의 몫이라 치고 일단 자기가 사용하는 자료형에 대해서 익숙해져 있지 않거나 대충 알고 넘어가게 된다면 프로그래밍의 반에 익숙하지 않고 반을 대충 해결한 것과 동일 합니다. 자료형에 대해서 즉시 알아보도록 합시다.

```
/*
 * this union is defined so Xlib can always use the same sized
 * event structure internally, to avoid memory fragmentation.
 */
typedef union _XEvent {
        int type;               /* must not be changed; first element */
        XAnyEvent xany;
        XKeyEvent xkey;
        XButtonEvent xbutton;
        XMotionEvent xmotion;
        XCrossingEvent xcrossing;
        XFocusChangeEvent xfocus;
        XExposeEvent xexpose;
        XGraphicsExposeEvent xgraphicsexpose;
        XNoExposeEvent xnoexpose;
        XVisibilityEvent xvisibility;
        XCreateWindowEvent xcreatewindow;
```

```
        XDestroyWindowEvent xdestroywindow;
        XUnmapEvent xunmap;
        XMapEvent xmap;
        XMapRequestEvent xmaprequest;
        XReparentEvent xreparent;
        XConfigureEvent xconfigure;
        XGravityEvent xgravity;
        XResizeRequestEvent xresizerequest;
        XConfigureRequestEvent xconfigurerequest;
        XCirculateEvent xcirculate;
        XCirculateRequestEvent xcirculaterequest;
        XPropertyEvent xproperty;
        XSelectionClearEvent xselectionclear;
        XSelectionRequestEvent xselectionrequest;
        XSelectionEvent xselection;
        XColormapEvent xcolormap;
        XClientMessageEvent xclient;
        XMappingEvent xmapping;
        XErrorEvent xerror;
        XKeymapEvent xkeymap;
        long pad[24];
} XEvent;
```

XEvent 는 <X11/Xlib.h>에 정의되어 있습니다. 지금 당장 Xlib.h 헤더화일을 편집기로 열어서 내용을 확인하십시요. 헤더 화일의 내용은 언젠가는 한 번 몽땅 읽어봐야 할 때가 오겠지요?

보시다시피 XEvent 는 공용체(유니언 union)입니다. 공용체가 무엇인지 모르신다면 일단 X 윈도우 프로그래밍은 접어두시고 지금 당장 가장 쉬운 C 프로그래밍 책에서 공용체에 대해서 공부하시기 바랍니다.

XEvent 공용체의 구조는 전체적으로 다음과 같습니다.

```
typedef union _XEvent {
        int type;               /* 꼭! 첫번째 멤버여야 한다. */
        ...
        X*Event <변수명>
        ...
        long pad[24];
} XEvent;
```

공용체의 크기는 공용체의 멤버 중 가장 큰 자료형의 크기와 같습니다. 마지막 멤버를 보면 XEvent 는 long 형 자료 24 개 정도의 크기를 갖는 것을 알 수 있습니다. 그리고 아마도 제일 작은 자료형은 int type 이겠죠?

중간에 무수히 많이 들어가 있는 X*Event 자료형을 한 번 봅시다. XAnyEvent, XKeyEvent, XButtonEvent, XMotionEvent, XExposeEvent 등등의 X 로 시작하고 Event로 끝나는 이름의 구조체가 있습니다. 그 구조체 자료형들 또한 모두 Xlib.h 에 정의되어 있습니다. 한 번 살펴보죠.

```
typedef struct {
        int type;             /* <-- 주목! */
        unsigned long serial; /* # of last request processed by server */
        Bool send_event;      /* true if this came from a SendEvent request */
        Display *display;/* Display the event was read from */
        Window window;  /* window on which event was requested in event mask */
} XAnyEvent;

typedef struct {
```

```
        int type;             /* <-- 주목! */
        unsigned long serial; /* # of last request processed by server */
        Bool send_event;      /* true if this came from a SendEvent request */
        Display *display;     /* Display the event was read from */
        Window window;        /* "event" window it is reported relative to */
        Window root;          /* root window that the event occured on */
        Window subwindow;     /* child window */
        Time time;            /* milliseconds */
        int x, y;             /* pointer x, y coordinates in event window */
        int x_root, y_root;   /* coordinates relative to root */
        unsigned int state;   /* key or button mask */
        unsigned int keycode; /* detail */
        Bool same_screen;     /* same screen flag */
} XKeyEvent;

typedef struct {
        int type;             /* <-- 주목! */
        unsigned long serial; /* # of last request processed by server */
        Bool send_event;      /* true if this came from a SendEvent request */
        Display *display;     /* Display the event was read from */
        Window window;        /* "event" window it is reported relative to */
        Window root;          /* root window that the event occured on */
        Window subwindow;     /* child window */
        Time time;            /* milliseconds */
        int x, y;             /* pointer x, y coordinates in event window */
        int x_root, y_root;   /* coordinates relative to root */
        unsigned int state;   /* key or button mask */
        unsigned int button;  /* detail */
        Bool same_screen;     /* same screen flag */
} XButtonEvent;

typedef struct {
        int type;             /* <-- 주목! */
        unsigned long serial; /* # of last request processed by server */
        Bool send_event;      /* true if this came from a SendEvent request */
        Display *display;     /* Display the event was read from */
        Window window;        /* "event" window reported relative to */
        Window root;          /* root window that the event occured on */
        Window subwindow;     /* child window */
        Time time;            /* milliseconds */
        int x, y;             /* pointer x, y coordinates in event window */
        int x_root, y_root;   /* coordinates relative to root */
        unsigned int state;   /* key or button mask */
        char is_hint;         /* detail */
        Bool same_screen;     /* same screen flag */
} XMotionEvent;

typedef struct {
        int type;             /* <-- 주목! */
        unsigned long serial; /* # of last request processed by server */
        Bool send_event;      /* true if this came from a SendEvent request */
        Display *display;     /* Display the event was read from */
        Window window;
        int x, y;
        int width, height;
```

```
        int count;           /* if non-zero, at least this many more */
    } XExposeEvent;
```

여기서는 몇 개의 구조체만을 예로 들었을 뿐입니다. 옆에 약간의 설명이 들어있으니 그것을 참고하시면 됩니다. 모두들 구조체인데 구조체 멤버들을 서로 비교해가면서 보십시요. 여기서 여러분 스스로 관찰을 해내면 X 프로그래밍은 재미있습니다.

XAnyEvent 의 내용은 한 번 머리 속에 넣고 다른 X*Event 구조체 멤버들을 살펴보십시요. 그러면 여지없이 모든 구조체에는 XAnyEvent 의 내용이 모두 들어 있다는 것을 알 수 있습니다.

7.5 이벤트 자료형 사용법

모든 이벤트 구조체들을 대표하고 있는 공용체 XEvent 를 프로그램에서 일단 사용합니다. X 서버가 우리들 프로그램에게 어떤 이벤트가 발생했다고 하면 바로 XEvent를 보내줍니다. 그러면 우리는 우선! XEvent 의 가장 기초적인 자료형인 int type의 내용을 살펴봅니다. 그 type 값을 가지고 이 이벤트가 과연 마우스과 관련된 것인지 아니면 키보드와 관련된 것인지 등을 판별합니다. 판별이 되면 그에 알맞는 자료형 즉 XEvent의 공용체 멤버들을 적절하게 사용합니다. 마우스와 관련되면 XEvent 에서 마우스와 관련된 멤버인 XButtonEvent, XMotionEvent 같은 것을 이용하여 이벤트를 해석합니다. 키보드에 관련되어 있다면 이번에는 그 자료형들 대신에 XKeyEvent 를 사용합니다.

각 구조체는 또 다시 자기에게 알맞는 구조체 멤버들을 가지고 있습니다. 그것을 보고 해석하면 됩니다. 만약 마우스에 관련되었다면 unsigned int button 멤버를 보고 어떤 버튼인지를 해석합니다. 만약 키보드에 관련되었다면 unsigned int keycode 를 보고 해석합니다.

그러한 과정을 바로 switch (xe.type) 문장에서 하고 있는 것입니다. case 문에 나타난 Expose, ButtonPress 등은 매크로입니다. X.h 에 정의되어 있습니다.

```
#define KeyPress              2
#define KeyRelease            3
#define ButtonPress           4
#define ButtonRelease         5
#define MotionNotify          6
#define EnterNotify           7
#define LeaveNotify           8
#define FocusIn               9
#define FocusOut              10
#define KeymapNotify          11
#define Expose                12
#define GraphicsExpose        13
#define NoExpose              14
#define VisibilityNotify      15
```

7.6 창마다 자기가 원하는 이벤트만을 예약하기

예제 소스를 보시면 다음과 같은 부분이 있습니다.

```
XSelectInput( display, window1, StructureNotifyMask | ExposureMask
                                | ButtonPressMask );

XMapWindow( display, window1 );
```

XSelectInput이라는 함수가 보이는군요. 새로운 함수니까 잠시 눈에 익혀두어야겠죠? man 명령을 떠올리셔야 합니다. man XSelectInput

```
NAME
      XSelectInput - 이벤트 입력을 선택합니다.

SYNTAX
      XSelectInput(display, w, event_mask)
            Display *display;
            Window w;
            long event_mask;

ARGUMENTS
      display    X 서버와의 접속 ID
      event_mask 이벤트 매스크
      w          이벤트 선택을 적용할 창 ID
```

매스크(Mask)라고 하는 기법 또한 아주 중요한 C 프로그래밍 실전 기법입니다. 모르시는 분은 당장 C 프로그래밍 책을 보십시요. 제가 생각엔 원판 커니건 & 리치의 아주 얇지만 그 어떤 C 프로그래밍 해설책보다도 뛰어난 "C 프로그래밍 랭귀지" 라는 책을 권합니다. 항상 가지고 다니기에 참 좋습니다. 쓸데 없이 쉽게 쓰지도 어렵게 쓰지도 않은 명저이지요.

위에서보면 우리는 각 창마다 XSelectInput 이라는 함수를 통해서 자기가 관심갖는 이벤트만을 등록할 수 있다는 것을 알 수 있습니다. 예를 들어 OK 나 CANCEL 버튼을 표시하는 창의 입장에서 키보드 입력이나 마우스의 이동 같은 것은 전혀 필요없겠지요? 버튼이 눌렸나 안눌렸나만 확인하면 됩니다.

```
StructureNotifyMask | ExposureMask | ButtonPressMask
```

비트연산 OR 또한 아주 많이 접하는 기술입니다.

```
#define NoEventMask                 0L
#define KeyPressMask                (1L<<0)
#define KeyReleaseMask              (1L<<1)
#define ButtonPressMask             (1L<<2)
#define ButtonReleaseMask           (1L<<3)
#define EnterWindowMask             (1L<<4)
#define LeaveWindowMask             (1L<<5)
#define PointerMotionMask           (1L<<6)
#define PointerMotionHintMask       (1L<<7)
#define Button1MotionMask           (1L<<8)
#define Button2MotionMask           (1L<<9)
#define Button3MotionMask           (1L<<10)
#define Button4MotionMask           (1L<<11)
#define Button5MotionMask           (1L<<12)
#define ButtonMotionMask            (1L<<13)
```

여러분은 (1L>>2) 라는 표현에 익숙해지셔야 합니다. 앞으로도 자주 나오고 기본적인 실전 테크닉이니까요.

그럼 간단한 예를 들어볼까요?

간단한 예

간단한 버튼을 하나 만들어서 클릭하면 Hello, Linuxers 라는 문자열을 출력한 후 종료하는 예를 보여드리겠습니다.

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <stdio.h>
```

```
int main( int argc, char* argv[] )
{
    Display *dpy;
    Window  w;
    Font    f;
    GC      gc;
    XEvent  xe;

    dpy = XOpenDisplay( NULL);
    w   = XCreateSimpleWindow( dpy, RootWindow( dpy, 0 ), 50, 50,
                                100, 50, 5, BlackPixel( dpy, 0 ),
                                WhitePixel( dpy, 0 ) );
    gc = XCreateGC( dpy, w, 0L, ( XGCValues * ) NULL );
    f  = XLoadFont( dpy, "fixed" );
    XSetFont( dpy, gc, f );
    XSetLineAttributes( dpy, gc, 5, LineSolid,
                        CapRound, JoinRound );

    XSelectInput( dpy, w, ExposureMask | ButtonPressMask | EnterWindowMask
                        | LeaveWindowMask );

    XMapWindow( dpy, w );

    XFlush( dpy );

    while ( True )
    {
            XNextEvent( dpy, &xe );

            switch ( xe.type )      {
            case Expose:
                printf("Expose 이벤트가 발생하였습니다.\n");
                XSetForeground( dpy, gc, BlackPixel( dpy, 0 ) )
                XDrawString( dpy, w, gc, 40, 30, "O K", 3 );
                break;
            case ButtonPress:
                printf("버튼을 누르셨습니다!\n");
                exit(1);
                break;
            case EnterNotify:
                printf("마우스가 창 안으로 들어왔습니다.\n");
                XSetForeground( dpy, gc, BlackPixel( dpy, 0 ) );
                XDrawRectangle( dpy, w, gc, 10, 10, 80, 30 );
                break;
            case LeaveNotify:
                printf("마우스가 창 밖으로 나갔습니다.\n");
                XSetForeground( dpy, gc, WhitePixel( dpy, 0 ) );
                XDrawRectangle( dpy, w, gc, 10, 10, 80, 30 );
                break;
            default:
                printf("모르는 이벤트입니다. 무시합니다.\n");
                break;
            }
    }
```

```
        return 0;
}
```

위 예제를 event.example.c 라는 이름으로 저장하고 다음과 같이 해줍니다.

```
gcc -o event.example event.example.c -lX11 -L/usr/X11/lib
```

실행은 한텀 창과 같이 한글이 출력될 수 있는 곳에서 실행해주십시요.

```
./event.example
```

예제에 대한 설명

XSelectInput 함수가 나오기 전까지 나오는 모든 것들은 이미 우리가 잘 알고 있는 내용입니다. 서버와 접속을 하고 창을 하나 만듭니다. 폰트 출력과 사각형 그리기를 할 것이므로 GC 를 만들어두는 것도 잊지 않습니다. 시스템의 기본인 fixed 폰트를 로딩하고 GC 세팅도 그럭저럭 해둡니다.

1. 창을 MAP하기 전에 우선 원하는 이벤트를 골라둔다. XSelectInput 함수를 통해서 우리는 원하는 창에다가 우리가 관심을 갖는 이벤트 만을 등록해둔다. 그러면 나머지 이벤트들은 그냥 무시되버리고 만다. 여기서 우리가 원하는 것은 일단 아주아주 중요한 이벤트의 하나인 Expose 이벤트, 마우스 버튼이 클릭되었는가를 알아보는 이벤트 그리고 창 속으로 마우스 포인터가 들어갔는가, 나왔는가를 알려주는 이벤트를 설정하였습니다.

   ```
   ExposureMask | ButtonPressMask | EnterWindowMask | LeaveWindowMask
   ```

2. Expose 이벤트의 중요성 Expose는 영어의 뜻 그대로 "밖으로 드러나다"라는 의미입니다. 윈도우 시스템에서 고려해줘야 할 것 중 하나는 한 화면을 동시에 많은 창들이 공유하고 있다는 사실입니다. 창들이 이리저리 서로를 가리기도 하고 다시 나타나기도 하며 아이콘으로 되었다가 다시 원상태로 돌아오기도 합니다. 크기가 늘기도 하지요.

   특히나 다른 창에 가렸다가 다시 나타나게 될 때는 남들에 의해서 가려졌던 부분들을 다시 그려줘야 합니다. 그런데 여러분이 Xt/모티프 같은 수준에서는 느끼지 못 하셨을지 모르겠지만 Xlib 은 저수준이므로 여러분 스스로가 남들에 의해 가려졌다가 다시 나타날 때 지워졌던 부분을 복구해주어야 합니다. 실제로 모티프 같은 것들은 이미 Xlib 의 함수를 써서 여러분 몰래 다시 그려주는 작업을 대신 해주고 있을 뿐입니다.

   Expose 이벤트는 여러분의 창이 맨 처음 XMapWindow 함수에 의해서 뿅! 하고 나타날 때 처음으로 발생합니다. 대부분의 프로그램들은 창이 맨 처음 뿅! 하고 나타날 때 자기 자신을 여러 모로 치장합니다. 또한 남들에 의해 가려졌다가 다시 뿅! 하고 여러분에게 나타낼 때는 적절하게 지워진 부분을 복구해주어야 합니다. 여기서는 아주 간단한 방식으로 처리했습니다. Expose 이벤트가 발생할 때마다 다시 O K 라는 글씨를 써주는 것이지요.

3. while (True) 문장을 통해서 이벤트 처리 무한 루프를 돕니다. XNextEvent 함수를 사용하면 여러분이 원하는 이벤트 중 하나가 생길 때까지 기다립니다. 이벤트가 안생기면 가만히 있습니다. 그러다가 생기면 파라미터로 주어진 XEvent 공용체에다가 이벤트 정보를 적어서 보내줍니다.

   ```
   switch ( xe.type )
   ```

자, 이벤트라는 것은 종류가 많기 때문에 그것이 마우스와 관련된 것인지 키보드와 관련된 것인지를 알아야 할 필요가 있습니다. 그것에 맞게 적절하게 처리해야 하니까요.

```
case Expose:

   Expose 이벤트 발생시에는 적절하게 자기 자신을 그려주거나 복구
   해주어야 합니다. XDrawString 함수로 "O K"라고 써줍니다.
   먼저 XSetForeground 함수로 전경색을 검정으로 해줍니다.

case ButtonPress:
```

버튼이 눌리면(어떤 버튼이든 상관하지 않고) exit(1) 종료합니다.

case EnterNotify:

마우스가 창 안으로 들어가면 XDrawRectangle 함수로 사각형을 테두리에 그려줍니다. 여러분의 마우스가 들어왔다는 것을 확실하게 보여주기 위해서입니다.

case LeaveNotify:

마우스가 나가면 나갔다는 것을 보여주기 위해 전경색을 하얀색으로 즉, 배경색과 같게 해서 사각형을 그려줍니다. 그러면 지워지는 효과가 나겠지요?

default:

알 수 없는 이벤트는 무시합니다.

위에서 예로 든 것은 여러분이 아주 쉽게 접할 수 있는 버튼 하나를 아주 저수준에서 해결한 것입니다. 보통 버튼을 클릭하기 위하여 마우스를 가져다 대면 버튼 전체의 색상이 변한다든지 하는 것을 볼 수 있습니다. 바로 이벤트에 맞게 자기 모양을 그리면 되는 것입니다. 원리는 아주 간단하지요?

버튼 하나를 만들어 보았습니다. 기존의 라이브러리만 쓰는 것이 아니라 여러분이 저수준으로 아주 재미있는 행동을 하는 버튼을 만들 수도 있습니다. 다른 사람은 흉내낼 수 없는 재미있는 일을 하는 것, 이것은 저수준 함수 Xlib 을 배우지 않고 는 이룰 수 없는 일입니다.

제 8 절　X 프로그래밍, X 로케일 이용 출력

아직 적은 수이기는 하지만 리눅스의 전파와 더불어 X 윈도우에 대한 관심도 점점 더 해가는 듯 한 느낌을 받습니다.

우리가 한국인인 이상 피할 수 없는 문제는 우리의 글의 올바른 표현이라고 할 수 있습니다. 자기 나라 말을 사랑하지 않거나 관심을 갖고 발전시키지 않는 민족은 이 세상에서 사라지는 것 아니겠습니까? 그리고 자기 말을 갖지 않는 민족이 도대체 어떤 자부심을 가질 수 있을까요?

컴퓨터 정보 문화의 시대에 살고 있는 우리에게 한글은 선조들의 노력을 이어받아 더욱 더 발전시키고 현대화시켜야 할 살아있는 언어라는 생각이 듭니다. 최소한 발전은 안시키더라도 애착을 갖고 사용은 해야 할 것 아니겠습니까?

윈도그 사용자 시절 무지하게도 항상 이런 말을 자신에게 하면서 컴퓨터를 사용해 왔던 것 같습니다.

”한글 윈도우는 안돼... 역시... 한글화만 하면 영 아니야... 한글 안돼...”

웃기는 사실은 엉터리로 한글화를 시킨 M$ 측이나 미국 가서 나라의 기술을 팔아먹고 있는 한국의 컴퓨터 기술자들에게 비난을 보내는 것이 아니라 ”한글” 자체를 들먹이고 있었다는 것입니다.

사실 1 바이트 문자권에서 탄생한 컴퓨터라는 환경 속에서 힘든 싸움을 하고 있는 것이 사실입니다. 힘든 일이지만 누구나 할 수 있는 일이라면 벌써 다른 사람들이 다 해놓을 겁니다. 힘든 일을 잘 하라고 비싼 돈 들여가며 귀중한 시간을 쏟아부어가며 프로그래머가 되려는 것 아닐까요?

1 바이트 문자보다는 약간 어려운 과정을 거치지만 그렇게 어려운 것도 아닙니다. 예전에 도그 시절에는 한글 입출력 모두를 매번 만들어서 사용했던 기억도 납니다. 아주 간단히 한글 기능 자체를 내장해버리니 속 편할 수 있었지만 지금의 소프트웨어 개발 상황에는 전혀 맞지 않는 구닥다리 방식으로 변화하고 말았습니다.

X 윈도우가 제공하고 있는 많은 기능들을 충분히 활용하면서 한글을 쉽게 표현할 수 있는 방법에 대해서 일단은 출력에 관한 부분을 다시 한 번 소개해드립니다.

8.1 X 윈도우 자체의 로케일 기능을 사용한 한글 출력

8.1.1 X 윈도우에서의 한글 출력 방식에 대하여

- 저수준의 방법 : XDrawString16 함수 이용
- 고수준의 방법 : XmbDrawString, XwcDrawString 함수 이용

우선 저수준의 방법은 한텀, 그리고 한글화된 윈도우 관리자들에서 사용하고 있는 방식으로서 고수준의 방법과 구별되는 기준은 바로 X 윈도우 시스템이 제공하고 있는 국제화(I18N, Internationalization)와 지역화(L10N, Localization) 기술을 사용하는가 여부에 달려있습니다. XDrawString16 함수를 사용하는 방법을 소개해 드린 적이 있는 것으로 압니다. 그 글을 읽지 않으셨다면 그 글을 먼저 읽어보시기 바랍니다.

저수준의 방법은 한글 문자열(char 형)을 한글과 영문으로 나누어서(바이트의 최상위 비트를 보고 판단) 한글 부분은 XChar2b 형 배열에 넣어서 2 바이트용 함수인 XDrawString16 함수로 출력하고 영문 부분은 따로 1 바이트 전용 함수인 XDrawString 함수로 출력하는 방식을 사용합니다.

저수준이므로 상당히 번거롭다고 하지 않을 수 없습니다. 한글과 영문을 섞어서 쓰는 경우에는 정말 신경쓸 일이 많습니다. 문자열을 읽어서 일일이 한글이 출력되는 화면 폭을 계산하고 그 다음에 영문을 출력하고 그 폭을 계산한 후 다시 한글을 출력하고...

고수준의 방법은 바로 X 윈도우 시스템이 R5 부터 지원하기 시작한 국제화/지역화 기법을 사용하는 것입니다. 같은 1 바이트 문자권이라 할 지라도 영어에 맞춰진 ASCII 문자로는 불가능한 유럽언어(프랑스어, 독일어 등...)와 애초부터 ASCII로는 택도 없는 2 바이트 이상의 문자권인 한국어, 중국어, 일본어 등에 대한 시스템 차원의 지원이 논의되면서 프로그래머들이 쉽게 국제적인 프로그램을 만들 수있도록 고민한 결과입니다. 국제화란 프로그램이 어떤 언어 환경에서든 사용될 수 있도록 유연하게 프로그래밍하는 것을 말하며 그렇게 국제화된 프로그램을 특정 언어나 문화에서 사용할 수 있도록 적응시키는 것을 지역화라고 부릅니다.

이미 솔라리스나 HP-UX 등에서 상용 X 윈도우를 보신 분들은 이것이 무엇을 뜻하는지 아실 수 있을 겁니다. 윈도그95 같은 것을 프랑스판, 독일어판, 한국어판, 중국어판, 일본어판 이렇게 닭질을 해가면서 따로따로 만드는 것과는 차원이 다릅니다. 간단한 설정 하나로 같은 X 윈도우 시스템이 여러 개의 언어로 동작하도록 설계되어 있습니다. 로그인 화면부터가 아예 달라지지요.

8.1.2 X 윈도우 시스템이 지원하는 I18N, L10N 이용하기

이제 알아볼 내용은 바로 고수준의 한글 출력에 대한 것입니다.

어떤 절차를 거쳐야 하는지에 대해서 알아보도록 합시다.

LOCALE 데이터베이스 지원을 누가 하는가?

유닉스 시스템의 로케일 지원은 일단 X 윈도우 시스템이 아닌 ANSI/POSIX 규격 차원의 지원이 있습니다. 대부분의 상용 유닉스나 아는 바 없지만 FreeBSD 와 같은 운영체제에서 지원하고 있는 것으로 알고 있습니다. 지금 현재 GNU NLS 팀에서 열심히 프로그램 메세지 지역화를 하고 계신 분들의 결과가 조만간 우리 곁에 오게 되겠지요? 그 분들에게 격려를 보냅니다.

리눅스는 아직 C 라이브러리에 대해서 제대로 모르기 때문에 확실하게 말씀드릴 수 없습니다. C 라이브러리 5.4.X 에서 로케일에 대한 지원이 있다고는 하는 것 같은데 제대로 확인이 되질 않는데다가 저의 관심 영역을 로우레벨 프로그래밍이 아니라서...

두번째로는 운영체제 자체의 지원과는 완전 별도로 X 윈도우 자체적으로 해결해주는 방식이 있습니다. 바로 오늘 X 윈도우 자체가 지원하는 로케일 지원을 사용하여 해결할 것입니다. 전자나 후자 모두 중요한데 일단 X 윈도우 자체적으로도 지원을 해준다는 것은 어떤 플랫포옴인가에 상관없이 같은 X 윈도우만 돌아간다면 어디서든 여러분의 프로그램이 제대로 동작하리란 보장을 받을 수 있다는 장점이 있습니다.

리눅스 C 라이브러리에 대한 연구는 시간을 내서 좀 더 해봐야 할 것 같습니다. 많은 사람들이 뛰어들어 주기를 바랄 뿐입니다. 리눅스는 개방 체제이므로 자원자가 많은 곳의 분위기를 따르지 않겠어요? 아시아의 젊은이들이 많이 뛰어들 수록 리눅스 자체는 국제적인 운영체제로 다시 태어날 수 있으리라 봅니다.

여러분의 X 윈도우 시스템을 점검합니다.

ANSI 규격에서 정의한 바와 같습니다. C 셸을 쓰는 분들은 setenv LANG ko 와 같은 표현을 써보신 적이 있을 겁니다. 이 글은 ANSI 규격의 setlocale 함수에 관련된 모든 사항을 자세히 설명하기 위한 것은 아니므로 따로 공부해보시기 바랍니다. 아주 중요한 이슈 중의 하나인 것은 분명한 것 같습니다.

일단은 /usr/X11R6/lib/X11/locale 디렉토리와 내용이 있는지를 확인해보십시요. 리눅스 배포판에서 X 윈도우 시리즈를 제대로 설치했다면 있어야 할 내용입니다.

```
C/              iso8859-3/      iso8859-8/      ko/             th_TH.TACTIS/
compose.dir     iso8859-4/      iso8859-9/      koi8-r/         zh/
en_US.utf/      iso8859-5/      ja/             locale.alias    zh_TW/
iso8859-1/      iso8859-6/      ja.JIS/         locale.dir
iso8859-2/      iso8859-7/      ja.SJIS/        tbl_data/
```

ko 라는 디렉토리가 있어야 합니다. 그 안에는 XLC_LOCALE 이라는 화일이 들어있습니다. 그 화일 안에 한국어의 출력에 대한 관련 정보가 들어있습니다.

보통은 LANG 세팅이 없는 경우 기본적으로 C 디렉토리에 있는 내용을 사용합니다.

위 화일들이 제대로 있는 것을 확인한 후에 꼭 다음 세팅을 해주어야 합니다.

본셸 계열이라면...

```
export LANG=ko
```

C 셸 계열이라면...

```
setenv LANG ko
```

나중에 같은 프로그램에 대해서 그냥 LANG 세팅을 재미삼아 ja 라고 바꿔보십시요. 그러면 재미있는 일이 벌어질 것입니다. 여러분이 네스케이프에서 옵션 메뉴에서 코딩 옵션만 바꿔주면 텍스트의 내용이 한글로 보였다가 일본어로 보였다 하는 원리를 쉽게 이해하실 수 있으리라 봅니다.

이제부터 프로그래밍을 해봅시다.

프로그래밍이란 전형적인 패턴이 있게 마련이지요. 그리고 확실한 패턴(물론 좋은 패턴 :>)을 익혀두는 것이 좋습니다.

예제를 손수 타이핑해 보시는 것이 좋을 겁니다. 일단 컴파일을 해보고 제대로 된다는 것을 확인하시기 바랍니다.

```
                         X Locale 을 이용한 한글 출력


*/
#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
/*
#include <locale.h>
*/
#include <X11/Xlocale.h>

void DrawCenteredMbString ( Display *, Window, XFontSet, GC, char*, int, int,
```

```
                                int, int, int );

int
main (argc, argv)
        int argc;
        char *argv[];
{
        Display *dpy;
        Window  w;
        XFontSet fontset;
        char **missing_charsets;
        int num_missing_charsets;
        char *default_string;
        int i;
        GC      gc;
        XEvent  xe;

        char *program_name = argv[0];

        if ( setlocale (LC_ALL,"") == NULL )       {
                (void) fprintf (stderr, "%s: cannot set locale.\n",
                program_name );
                exit (1);
        }

        if ( !XSupportsLocale () ) {
                (void) fprintf (stderr,"%s: X does not support locale %s.\n",
                program_name, setlocale( LC_ALL, NULL ) );
                exit (1);
        }

        if ( XSetLocaleModifiers("") == NULL )    {
                (void) fprintf (stderr,
                        "%s: Warning: cannot set locale modifiers.\n",
                        program_name);
        }

        dpy = XOpenDisplay ( NULL);
        w   = XCreateSimpleWindow ( dpy, RootWindow( dpy, 0 ), 50, 50,
                                        100, 50, 5, BlackPixel( dpy, 0 ),
                                        WhitePixel( dpy, 0 ) );
        gc = XCreateGC ( dpy, w, 0L, ( XGCValues * ) NULL );

        fontset = XCreateFontSet (dpy, "-*-*-*-*-*-*-16-*-*-*-*-*-*-*",
                        &missing_charsets, &num_missing_charsets,
                        &default_string);
        if ( num_missing_charsets > 0 ) {
                (void) fprintf (stderr,
                        "%s: The following charsets are missing:\n",
                        program_name);
                for ( i=0; i < num_missing_charsets; i++ )
                        (void)fprintf (stderr, "%s:   %s\n", program_name,
                                missing_charsets[i]);
                (void) fprintf (stderr, "%s: The string is %s", program_name,
                                default_string);
```

```
                (void) fprintf (stderr,
                                "%s: of any characters from those sets.\n",
                                program_name);
                XFreeStringList (missing_charsets);
        }

        XSetLineAttributes ( dpy, gc, 5, LineSolid, CapRound, JoinRound );

        XSelectInput ( dpy, w, ExposureMask | ButtonPressMask
                        | EnterWindowMask | LeaveWindowMask );

        XMapWindow ( dpy, w );

        XFlush ( dpy );

        while ( True )
        {
                XNextEvent ( dpy, &xe );

                switch ( xe.type )      {

                case Expose:
                        printf ("Expose 이벤트가 발생했습니다.\n");
                        XSetForeground ( dpy, gc, BlackPixel( dpy, 0 ) );
                        DrawCenteredMbString ( dpy, w, fontset, gc,
                                        "안녕 Hello",
                                        10, 0, 0, 100, 50 );
                        break;
                case ButtonPress:
                        printf ("버튼을 누르셨습니다!\n");
                        exit (1);
                        break;
                case EnterNotify:
                        printf ("마우스가 창 안으로 들어왔습니다.\n");
                        XSetForeground ( dpy, gc, BlackPixel( dpy, 0 ) );
                        XDrawRectangle ( dpy, w, gc, 0, 0, 100, 50 );
                        break;
                case LeaveNotify:
                        printf ("마우스가 창 밖으로 나갔습니다.\n");
                        XSetForeground ( dpy, gc, WhitePixel( dpy, 0 ) );
                        XDrawRectangle ( dpy, w, gc, 0, 0, 100, 50 );
                        break;
                default:
                        printf ("모르는 이벤트입니다. 무시합니다.\n");
                        break;
                }
        }
        return 0;
}

void
DrawCenteredMbString (dpy, w, fontset, gc, str, num_bytes, x, y, width, height )
        Display *dpy;
        Window w;
        XFontSet fontset;
```

```
        GC gc;
        char *str;
        int num_bytes;
        int x,y, width, height;
{
        XRectangle boundingbox;
        XRectangle dummy;
        int originx, originy;

        (void) XmbTextExtents (fontset, str, num_bytes, &dummy, &boundingbox);

        originx = x + ( width - boundingbox.width )/2 - boundingbox.x;
        originy = y + ( height - boundingbox.height)/2 - boundingbox.y;

        XmbDrawString (dpy, w, fontset, gc, originx, originy, str, num_bytes);
}
```

자, 컴파일을 해보도록 합시다. 약간의 주의가 필요합니다.

이 프로그램은 꼭 리눅스에서만 컴파일하란 법은 없습니다. (확인은 안했지만...)

시스템 수준에서 로케일을 지원하는 경우에는 다음과 같이 컴파일해야 합니다.

```
gcc -o test_locale test.locale.c -lX11 -L/usr/X11/lib
```

리눅스 시스템인 경우에는 X 윈도우 자체의 로케일을 사용할 것이므로 다음과 같이 해주어야만 합니다.

```
gcc -o test_locale test.locale.c -DX_LOCALE -lX11 -L/usr/X11/lib
```

순서대로 설명을 해보도록 하겠습니다.

시스템이 로케일을 지원하는가?

setlocale 함수를 써서 시스템 차원에서 또는 X 윈도우 시스템 차원에서 로케일을 지원하는지 여부에 대해서 알아봅니다. 실행되는 플랫포옴에서 로케일을 지원하지 않는다면 다른 조치를 취하든지 아니면 실행을 중지해야 할 것입니다. 제대로 된 결과가 나오질 않을테니까요.

```
setlocale (LC_ALL,"") == NULL 이 문장으로 확인을 하고 있습니다.
```

X 윈도우 차원에서의 로케일 지원 여부 확인

XSupportsLocale 함수와 XSetLocaleModifiers 함수를 사용하여 로케일 지원여부를 물어봅니다.

후자의 함수는 "X 입력 서버"에 관계된 것이므로 지금으로서는 설명드릴 수가 없고 상당히 전문적인 얘기로 빠질 가능성이 많습니다. 조만간에 "X 입력 서버" 제작에 대해서 따로 소개를 해드리기로 하겠습니다. 궁금증을 참기 어려운 분은 그냥 오렐리 출판사의 X 윈도우 시리즈 Volume One Xlib 프로그래밍 매뉴얼을 보시면 됩니다.

폰트셋(FontSet)이라는 개념....

폰트셋이라는 개념은 글자 그대로 폰트의 한 세트를 말합니다. 어떤 특정 언어의 코드에 맞는(예를 들어 ASCII, ISO8859-1, EUC-KR, ISO-2022-KR 등등) 문자 폰트들이 이미 마련되어 있어야 출력을 할 수 있겠지요? 즉 환경 변수 LANG 이 지시하는 코딩 스타일에 맞는 폰트들의 한 세트를 지정해주어야 합니다.

모티프 프로그래밍을 해보신 분들이라면 폰트셋이라는 개념에 대해서 어느 정도 아실 겁니다. 일단은 X 윈도우 로케일을 이용하는 방식은 폰트셋을 사용한다는 사실을 정확히 아셔야 합니다. 생각보다 상당히 간편하게 만듭니다.

XCreateFontSet 함수를 이용하지요.

두번째 인수로 그냥 -*-*-*-*-*-*-16-*-*-*-*-*-*-* 이라고 XLFD 방식의 폰트명을 지정해주었으므로 16 포인트 폰트들을 찾아내서 폰트셋에 등록하게 됩니다. 만약 16 포인트 폰트가 없다면 가장 가까운 폰트를 찾아서 X 윈도우 자체가 스케일링을 해줍니다. 이 때가 X 윈도우의 가장 취약한 부분인데... 2 바이트권의 엄청난 양의 폰트를 스케일링하게 될 때는 마치 X 서버가 멈춰버리는 듯 한 현상이 일어납니다. 폰트를 모두 변환하느라 엄청난 시간을 소비하고 있기 때문이지요. 위 숫자를 15나 다른 숫자로 바꾸어서 한 번 해보시기 바랍니다. 그 차이을 느끼실 수도 있을 겁니다.

문자열의 출력은 간단히 XmbDrawString 함수 이용

위에서는 DrawCenteredMbString 라는 사용자 정의 함수 내에서 멀티 바이트 코딩에 사용하는 출력 함수 XmbDrawString 를 사용하였습니다. 위에서 보시는 바와 같이 옛날에 저수준 출력함수를 사용하여 한글을 한글대로 영문은 영문대로 따로 닭질을 할 필요가 없습니다. -_-

한글 코딩된 2 바이트 문자를 그냥 쓰면 됩니다. 단지 위에서는 문자 출력 위치를 조정하기 위하여 XmbTextExtents 함수와 같은 것을 사용하였습니다.

제 9 절 마감하면서

이제까지 출력의 문제만을 다루었지만 프로젝트 중인 "X 입력서버"의 진척 상황에 맞추어서 한글 입력의 문제를 다루어보기로 하겠습니다. 암튼 출력에 대한 설명이 여러분에게 많은 도움이 되길 바랍니다. 일단은 조금만 노력하면 거의 모든 메세지를 한글화 패치할 수 있습니다.

남이 안해주면 스스로 나서는 것, 리눅서들의 멋진 자세 아니겠습니까?

제일 멋진 운영체제로 우리가 가꾸어 나갑시다.

시간이 좀처럼 나질 않는군요. 배울 것은 많은데 시간은 총알같이 흘러가고 ... 시간을 좀 더 효율적으로 써서 여러분을 자주 뵐 수 있게 되길 바랍니다.

그럼...