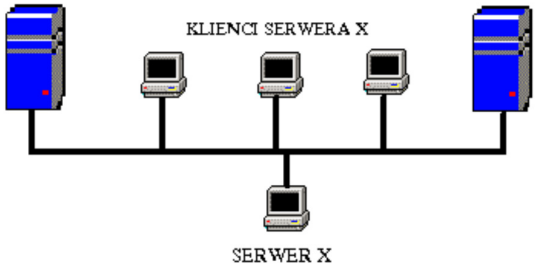
GRAFIKA KOMPUTEROWA

Podstawy programowania w środowisku X Window

Budowa systemu X Window



X był od samego początku projektowany jako system zorientowany sieciowo. Serwer odpowiada głównie za wyświetlanie grafiki, obsługę myszy, czy klawiatury. Wszelkie aplikacje X są klientami z korzystającymi z funkcji udostępnianych przez serwer. Klienci mogą się znajdować na tej samej maszynie co serwer lub na zupełnie innej. Komunikacja pomiędzy klientami a serwerem odbywa się zgodnie z protokołem X. Aplikacje-klienci wysyłają do serwera komunikaty w stylu "proszę, narysuj mi okno o takich współrzędnych". Serwer natomiast powiadamia klienta o zdarzeniach typu naciśnięcie klawisza myszki przez użytkownika.

Application
X Toolkit
Xt Intrinsics
Xlib, XCB
X protocol

Najniższym poziomem jest **X protocol** określający standard przesyłania danych pomiędzy klientem i serwerem. Następny poziom odpowiedzialny jest za operacje na oknach, klawiaturze i myszy. Może być one realizowany za pośrednictwem biblioteki **Xlib** (jak w przypadku naszego ćwiczenia) lub nowszej **XCB**, która w przeciwieństwie do starszej wersji umożliwia asynchroniczną komunikację z serwerem. Kolejnym poziomem jest poziom **Xt Intrinsics** umożliwiający korzystanie ze standardowych bloków takich jak menu, przyciski, suwaki, okna dialogowe itp. Najwyższym poziomem jest poziom **X Toolkit** uzależniony od producenta - najpopularniejsze środowiska to Motif, Gtk+ czy Qt.

Praca z językiem C i bibliotekami X Window

Pliki nagłówkowe zawierają zwykle wymagane deklaracje obiektów (zmiennych, funkcji, ...) zdefiniowanych w plikach bibliotecznych dołączanych do programu w fazie konsolidacji. Niezbędne dla nas pliki znajdują się w podkatalogach „X11" umieszczonych w katalogach „/usr/X11R6/lib" (pliki biblioteczne) i „/usr/X11R6/include" (pliki nagłówkowe).
Najprostszy program pracujący w środowisku X Window wygląda następująco:

```
#include <X11/Xlib.h>
main(int argc, char *argv[])
{
   Display *wyswietlacz;
   wyswietlacz = XOpenDisplay("");
   XCloseDisplay(wyswietlacz);
}
```

Jest to program, który nic nie robi, nawiązuje tylko połączenie z X serwerem i od razu porzuca je, bez sprawdzenia poprawności wykonania się operacji. Nie daje on żadnych wizualnych efektów. Należy podkreślić, że pusty łańcuch ("") jako argument funkcji XOpenDisplay oznacza użycie takiego wyświetlacza jaki jest zdefiniowany przez zmienną systemową DISPLAY.
Kompilacji i konsolidacji takiego programu umieszczonego w pliku „prog.c" dokonać można poleceniem:

```
gcc -o prog -I/usr/X11R6/include -L/usr/X11R6/lib \
-R/usr/X11R6/lib -lX11 prog.c
```

Gdzie
`-o` - określenie nazwy pliku z programem wykonywalnym (tutaj: „prog"),
`-I` - określenie położenia zbiorów nagłówkowych,
`-L` - określenie położenia bibliotek,
`-R` - określenie położenia bibliotek dołączanych dynamicznie (dopiero podczas działania programu),
`-l` - określenie dołączanych bibliotek.
Pominięcie opcji -I, -L, -R spowoduje wykorzystanie ustawień ze zmiennej systemowej LD_LIBRARY_PATH.

Podstawowe konstrukcje programistyczne w środowisku X Window - program przykładowy

Poniższy program wyświetla okno z napisem. Wyjście z programu następuje w wyniku naciśnięcia klawisza „q".
Dyrektywy i definicje:

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#define TRUE 1
#define FALSE 0
```

Plik nagłówkowy Xlib.h zawiera daklarację funkcji i struktur danych Xlib, natomiast Xutil.h używany jest przez funkcje użytkowe Xlib.

Stałe łańcuchowe:

```
char hello[]="To jest okno";
char NazwaIkony[]="Ikona";
char NazwaOkna[]="Okno";
```

Program główny (argc - liczba argumentów linii polecenia, argv - tablica wskazań do tych argumentów):

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

Deklaracja zmiennych - najpierw wskaźnik definiujący wyświetlacz do którego wysyłamy dane - dany X serwer:

```
    Display *wyswietlacz;
```

Zmienna odpowiedzialna za nasze okno:

```
    Window okno;
```

Struktura zawierające informacje o parametrach graficznych, niezbędna dla wszystkich elementów, które chcemy rysować - kontekst graficzny:

```
    GC graficznykontekst;
```

Struktura do zapisywania zdarzeń:

```
    XEvent zdarzenie;
```

Informacja o stanie klawiatury:

```
    KeySym klawisz;
```

Parametry takie jak rozmiar i położenie okna (definicja znajduje się w Xutli.h):

```
    XSizeHints info;
```

Numer używanego ekranu:

```
    int ekran;
```

Kolory tła i pierwszego planu:

```
    unsigned long pierwszyplan, tlo;
```

Bufor do zapamiętywania znaków z klawiatury:

```
    char bufor[8];
```

Licznik klawiszy:

```
    int liklawiszy;
```

Zmienna do kontrolowania momentu zakończenia programu:

```
int zakoncz;
```

Początek programu:

Otwarcie połączenia z X serwerem, parametr "" oznacza, że wyświetlacz ma być ustawiony na podstawie zmiennej systemowej DISPLAY, można też podać konkretny komputer:

```
wyswietlacz = XOpenDisplay("");
```

Pobranie numeru domyślnego ekranu (w naszym przypadku i tak jest to tylko jeden ekran o numerze 0):

```
ekran = DefaultScreen(wyswietlacz);
```

Makra WhitePixel i BlackPiksel zwracają odpowiednie wartości liczbowe opisujące kolor biały i czarny dla naszego wyświetlacza, użytkownik ma dzięki temu pewność, że będą to odpowiednio kolor biały i czarny bez względu na maksymalną ilość kolorów, ilość bitów na piksel itp.:

```
tlo = WhitePixel(wyswietlacz, ekran);
pierwszyplan = BlackPixel(wyswietlacz, ekran);
```

Ustawienie parametrów okna poprzez wpisanie ich do struktury info. Pola x, y tej struktury opisują położenie górnego lewego rogu okna; pola width i hight określają szerokość i wysokość okna; pole flags informuje, które z pól tej struktury zostały zdefiniowane. Odpowiednie funkcje mogą modyfikować tę strukturę:

```
info.x = 100;
info.y = 150;
info.width = 275;
info.height = 120;
info.flags = PPosition | PSize;
```

Stworzenie okna. Nie powoduje ono wyświetlenia okna, a jedynie stworzenie odpowiednich struktur danych. Parametrami funkcji tworzącej okno są: wyświetlacz, na którym ma być stworzone okno, okno rodzic (dokładniej: jego uchwyt) - w naszym przykładzie jest nim pulpit (okno domyślne), położenie i rozmiary, grubość ramki (7) i wreszcie kolorystyka okna. Stworzone okno identyfikowane jest później przez wyświetlacz i uchwyt (numer) okna:

```
okno = XCreateSimpleWindow(wyswietlacz,
      DefaultRootWindow(wyswietlacz),
      info.x, info.y, info.width, info.height,
      7, pierwszyplan, tlo);
```

Określenie własności okna takich jak: nazwa okna, nazwa ikony, wskaźnik na mapę bitową definiującą ikonę (tu None), liczba i parametry wołania programu oraz dane zawarte w strukturze info:

```
XSetStandardProperties(wyswietlacz, okno, NazwaOkna,
      NazwaIkony, None, argv, argc, &info);
```

Utworzenie kontekstu graficznego z wartościami domyślnymi (0, 0) dla określenia maski i wartości:

```
graficznykontekst = XCreateGC(wyswietlacz, okno, 0, 0);
```

Wpisanie do kontekstu graficznego informacji o kolorze pierwszego planu i tła:

```
XSetBackground(wyswietlacz, graficznykontekst, tlo);
XSetForeground(wyswietlacz, graficznykontekst, pierwszyplan);
```

Wybór zdarzeń, które mają być sygnalizowane naszemu oknu. Nas interesuje tylko naciśnięcie klawisza, potrzebne do zakończenia programu, oraz zdarzenie Expose, pojawiające się w przypadku konieczności odrysowania okna (gdy np. dzięki zamknięciu jakiegoś okna nasze okno zostało odsłonięte):

```
XSelectInput(wyswietlacz, okno, (KeyPressMask | ExposureMask));
```

Wyświetlenie stworzonego okna na wierzchu wszystkich okien:

```
XMapRaised(wyswietlacz, okno) ;
```

Ustawienie zmiennej kończącej program:

```
zakoncz = FALSE;
```

Pętla, aż zmienna zakoncz nie zmieni wartości (warto zwrócić uwagę na podwójny znak równości):

```
while (zakoncz == FALSE){
```

Pobranie kolejnego zdarzenia. Funkcja ta czeka, aż nie pojawi się jedno ze zdarzeń określonych funkcją XSelectInput:

```
    XNextEvent(wyswietlacz, &zdarzenie);
```

Wykonanie akcji w zależności od typu zdarzenia (odpowiada instrukcji CASE w Pascalu):

```
    switch(zdarzenie.type){
```

Zdarzenie Expose:

```
        case Expose:
```

Wartość pola xexpose.count równa zero może oznaczać dwie możliwości: tylko jedno zdarzenie Expose miało miejsce, albo jest to ostatnie zdarzenie z grupy zdarzeń typu Expose:

```
        if (zdarzenie.xexpose.count == 0){
```

Narysowanie zawartości okna - w naszym przypadku sprowadza się ono do wypisania funkcją XDrawImageString tekstu zawartego w zmiennej hello. Parametrami tej funkcji są: wyświetlacz, okno (uchwyt), kontekst graficzny, położenie napisu (pierwszej litery) w oknie, napis i jego długość. W naszym programie mamy do czynienia tylko z jednym oknem i można by było zarówno wyświetlacz jak i okno podać normalnie. Bezpieczniej jest jednak odczytać z pól rekordu o nazwie zdarzenie, o który wyświetlacz i o które okno chodzi w danym zdarzeniu:

```
            XDrawImageString(zdarzenie.xexpose.display,
               zdarzenie.xexpose.window, graficznykontekst, 105, 65,
               hello, strlen(hello));}
```

Break oznacza koniec akcji dla zdarzenia Expose:

```
            break;
```

Kolejne zdarzenie to zdarzenie informujące o zmianach układu klawiatury - "konfiguracji":

```
        case MappingNotify:
```

Chcąc zabezpieczyć się przed taką zmianą należy zawrzeć w programie następującą instrukcję:

```
            XRefreshKeyboardMapping(&zdarzenie);
```

Koniec akcji dla zdarzenia MappingNotify:

```
            break;
```

Wreszcie zdarzenie związane z naciśnięciem klawiszy

```
        case KeyPress:
```

Konwersja naciśniętych klawiszy na łańcuch znaków. Funkcja XLookupString zwraca informacje o długości tego łańcucha - zazwyczaj jest to wartość jeden.

```
            liklawiszy = XLookupString(&zdarzenie, bufor, 8, &klawisz, 0);
```

Jeśli był to właśnie tylko jeden klawisz i był to w dodatku klawisz „q”, to należy zakończyć program:

```
            if ((liklawiszy == 1) && (bufor[0] == 'q'))
               zakoncz = TRUE;
```

Koniec instrukcji switch i koniec pętli:

```
        }
    }
```

Przed zakończeniem programu należy jeszcze wykonać:

Wyczyszczenie i zwolnienie pamięci potrzebnej na kontekst graficzny:

```
XFreeGC(wyswietlacz, graficznykontekst);
```

Usunięcie okna:

```
XDestroyWindow(wyswietlacz, okno);
```

Zamknięcie połączenia z wyświetlaczem:

```
XCloseDisplay(wyswietlacz);
```

Koniec programu:

```
}
```

Kod programu przykładowego

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#define TRUE 1
#define FALSE 0

char hello[]="To jest okno";
char NazwaIkony[]="Ikona";
char NazwaOkna[]="Okno";

main(int argc, char *argv[]){
   Display *wyswietlacz;
   Window okno;
   GC graficznykontekst;
   XEvent zdarzenie;
   KeySym klawisz;
   XSizeHints info;
   int ekran;
   int zakoncz;

   wyswietlacz = XOpenDisplay("");
   ekran = DefaultScreen(wyswietlacz);
   unsigned long pierwszyplan, tlo;
   char bufor[8];
   int liklawiszy;
   tlo = WhitePixel(wyswietlacz, ekran);
   pierwszyplan = BlackPixel(wyswietlacz, ekran);
   info.x = 100;
   info.y = 150;
   info.width = 275;
   info.height = 120;
   info.flags = PPosition | PSize;
   okno = XCreateSimpleWindow(wyswietlacz,
   DefaultRootWindow(wyswietlacz),
      info.x, info.y, info.width, info.height, 7,
      pierwszyplan, tlo);
   XSetStandardProperties(wyswietlacz, okno, NazwaOkna, NazwaIkony,
      None, argv, argc, &info);
   graficznykontekst = XCreateGC(wyswietlacz, okno, 0, 0);
   XSetBackground(wyswietlacz, graficznykontekst, tlo);
   XSetForeground(wyswietlacz, graficznykontekst, pierwszyplan);
   XSelectInput(wyswietlacz, okno, (KeyPressMask | ExposureMask));
   XMapRaised(wyswietlacz, okno);
   zakoncz = FALSE;

   while (zakoncz == FALSE){
      XNextEvent(wyswietlacz, &zdarzenie);
      switch(zdarzenie.type){
         case Expose:
            if (zdarzenie.xexpose.count == 0)
               XDrawImageString(zdarzenie.xexpose.display,
                  zdarzenie.xexpose.window,
                  graficznykontekst, 105, 65,
                  hello, strlen(hello));
            break;
         case MappingNotify :
            XRefreshKeyboardMapping(&zdarzenie);
            break;
         case KeyPress:
            liklawiszy = XLookupString(&zdarzenie, bufor, 8,
               &klawisz, 0);
            if ((liklawiszy == 1) && (bufor[0] == 'q'))
               zakoncz = TRUE;
      }
   }
   XFreeGC(wyswietlacz, graficznykontekst);
   XDestroyWindow(wyswietlacz, okno);
   XCloseDisplay(wyswietlacz);
}
```

Wybrane stałe i funkcje biblioteczne

```
XSetLineAttributes(
   Display*     /* display */,
   GC           /* gc */,
   unsigned int /* line_width */,
   int          /* line_style */,
   int          /* cap_style */,
   int          /* join_style */
);

/* LineStyle */

#define LineSolid 0
#define LineOnOffDash 1
#define LineDoubleDash 2

/* capStyle */

#define CapNotLast 0
#define CapButt 1
#define CapRound 2
#define CapProjecting 3

/* joinStyle */

#define JoinMiter 0
#define JoinRound 1
#define JoinBevel 2
```

```
typedef struct {
   short x, y;
} XPoint;

typedef struct {
   short x1, y1, x2, y2;
} XSegment;

extern XDrawArc(//ew. XFillArc
   Display*     /* display */,
   Drawable     /* d */,
   GC           /* gc */,
   Int          /* x */,
   Int          /* y */,
   unsigned int /* width */,
   unsigned int /* height */,
   int          /* angle1 */,
   int          /* angle2 */
);

extern XDrawArcs(
   Display*     /* display */,
   Drawable     /* d */,
   GC           /* gc */,
   XArc*        /* arcs */,
   int          /* narcs */
);

extern XDrawImageString(
   Display*        /* display */,
   Drawable        /* d */,
   GC              /* gc */,
   int             /* x */,
   int             /* y */,
   _Xconst char*   /* string */,
   int             /* length */
);

extern XDrawImageString16(
   Display*           /* display */,
   Drawable           /* d */,
   GC                 /* gc */,
   int                /* x */,
   int                /* y */,
   _Xconst XChar2b*   /* string */,
   int                /* length */
);

extern XDrawLine(
   Display*           /* display */,
   Drawable           /* d */,
   GC                 /* gc */,
   int                /* x1 */,
   int          /* y1 */,
   int          /* x2 */,
   int          /* y2 */
);

extern XDrawLines(
   Display*  /* display */,
   Drawable  /* d */,
   GC        /* gc */,
   XPoint*   /* points */,
   int       /* npoints */,
   int       /* mode */
);

extern XDrawPoint(
   Display*  /* display */,
   Drawable  /* d */,
   GC        /* gc */,
   int       /* x */,
   int       /* y */
);

extern XDrawPoints(
   Display*  /* display */,
   Drawable  /* d */,
   GC        /* gc */,
   XPoint*   /* points */,
   int       /* npoints */,
   int       /* mode */
);

extern XDrawRectangle( //XFillRectangle
   Display*     /* display */,
   Drawable     /* d */,
   GC           /* gc */,
   int          /* x */,
   int          /* y */,
   unsigned int /* width */,
   unsigned int /* height */
);

extern XDrawRectangles(
   Display*     /* display */,
   Drawable     /* d */,
   GC           /* gc */,
   XRectangle*  /* rectangles */,
   int          /* nrectangles */
);

extern XDrawSegments(
   Display*     /* display */,
   Drawable     /* d */,
   GC           /* gc */,
```

```
    XSegment*       /* segments */,
    int             /* nsegments */
);

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

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

extern XDrawText(
    Display*        /* display */,
    Drawable        /* d */,
    GC              /* gc */,
    int             /* x */,
    int             /* y */,
    XTextItem*      /* items */,
    int             /* nitems */
);

extern XDrawText16(
    Display*        /* display */,
    Drawable        /* d */,
    GC              /* gc */,
    int             /* x */,
    int             /* y */,
    XTextItem16*    /* items */,
    int             /* nitems */
);

/* Data structure for "image" data, used by image manipulation routines.
*/
typedef struct _XImage {
    int width, height;  /* size of image */
    int xoffset;        /* number of pixels offset in X direction */
    int format;         /* XYBitmap, XYPixmap, ZPixmap */
    char *data;         /* pointer to image data */
    int byte_order;     /* data byte order, LSBFirst, MSBFirst */
    int bitmap_unit;    /* quant. of scanline 8, 16, 32 */
    int bitmap_bit_order;           /* LSBFirst, MSBFirst */
    int bitmap_pad;                 /* 8, 16, 32 either XY or ZPixmap */
    int depth;                      /* depth of image */
    int bytes_per_line;             /* accelarator to next line */
    int bits_per_pixel;             /* bits per pixel (ZPixmap) */
    unsigned long red_mask;         /* bits in z arrangment */
    unsigned long green_mask;
    unsigned long blue_mask;
    XPointer obdata;                /* hook for the object routines to hang on*/
    struct funcs {                  /* image manipulation routines */
        struct _XImage *(*create_image)();
        int (*destroy_image) (struct _XImage *);
        unsigned long (*get_pixel) (struct _XImage *, int, int);
        int (*put_pixel) (struct _XImage *, int, int, unsigned long);
        struct _XImage *(*sub_image)(struct _XImage *, int, int,
        unsigned int, unsigned int);
        int (*add_pixel) (struct _XImage *, long);
    } f;
} XImage;

/*
* These macros are used to give some sugar to the image routines
* so tha naive people are more comfortable with them.
*/
#define XDestroyImage(ximage) \
    ((*((ximage)->f.destroy_image))((ximage)))
#define XGetPixel(ximage, x, y) \
    ((*((ximage)->f.get_pixel))((ximage), (x), (y)))
#define XPutPixel(ximage, x, y, pixel) \
    ((*((ximage)->f.put_pixel))((ximage), (x), (y), (pixel)))
#define XSubImage(ximage, x, y, width, height) \
    ((*((ximage)->f.sub_image))((ximage), (x), (y), (width), (height)))
#define XAddPixel(ximage, value) \
    ((*((ximage)->f.add_pixel))((ximage), (value)))
extern XImage *XCreateImage(
    Display*     /* display */,
    Visual*      /* visual */,
    unsigned int /* depth */,
    int          /* format */,
    int          /* offset */,
    char*        /* data */,
    unsigned int /* width */,
    unsigned int /* height */,
    int          /* bitmap_pad */,
    int          /* bytes_per_line */
);
extern XImage *XGetImage(
    Display*     /* display */,
    Drawable     /* d */,
    int          /* x */,
    int          /* y */,
    unsigned int /* width */,
    unsigned int /* height */,
```

```
   unsigned long   /* plane_mask */,
   int             /* format */
);
extern XPutImage(
   Display*        /* display */,
   Drawable        /* d */,
   GC              /* gc */,
   XImage*         /* image */,
   int             /* src_x */,
   int             /* src_y */,
   int             /* dest_x */,
   int             /* dest_y */,
   unsigned int    /* width */,
   unsigned int    /* height */
);
/* ImageFormat -- PutImage, GetImage */
#define XYBitmap 0 /* depth 1, XYFormat */
#define XYPixmap 1 /* depth == drawable depth */
#define ZPixmap 2 /* depth == drawable depth */
```

Przykładowe fragmenty kodu

```
XSetBackground(wyswietlacz, graficznykontekst, tlo);
XSetForeground(wyswietlacz, graficznykontekst, pierwszyplan);
XSetLineAttributes(wyswietlacz, graficznykontekst, 20,
LineSolid, CapRound, JoinMiter);

XPoint punkty[] = {
   {10, 10},
   ......
   {90, 90}
};

int li_punkty;

XSegment odcinki[] = {
   {10, 10, 40, 60},
   ......
   {100, 50, 80, 10}
};
int li_odcinki;

li_punkty = sizeof(punkty) / sizeof(XPoint);
li_odcinki = sizeof(odcinki) / sizeof(XSegment);

XDrawLines(wyswietlacz, okno, graficznykontekst, punkty,
   li_punkty, CoordModeOrigin);
   /* CoordModeOrigin, CoordModePrevious */

XDrawSegments(wyswietlacz, okno, graficznykontekst, odcinki,
   li_odcinki);

XImage *obraz;

if ((liklawiszy == 1) && (bufor[0] == 'i')){
   obraz = XGetImage(wyswietlacz, okno, 0, 0, 100, 100,
      0xffffffff, XYPixmap);
   przetworz(obraz);
   XPutImage(wyswietlacz, okno, graficznykontekst,
   obraz, 0, 0, 0, 0, 100, 100);
   XDestroyImage(obraz);
}
```