**X lib Programming**  
*(lecture programs)*

- **X server**: Controls the Input/Output Resources of a host: *(display, keyboard, mouse)*.

- **X clients**: Applications that runs at any host in the Internet (may be different from the X server's host).

- The communications between the X clients and the X server is TCP.

The default port# for the X server is 6000.
Examples of X lib Programs

**Drawing Points: xpoints.c**

```c
main(argc, argv)
int argc;
char **argv;
{
    Display *display;
    Window root, window;
    long fgcolor, bgcolor;
    int screen, pointx, pointy;
    long eventmask = ButtonPressMask|ExposureMask|KeyPressMask;
```
XEvent event;
XGCValues gcval;
GC draw;
Colormap cmap;
XColor color, ignore;
char *colorname = "red";
The above are definitions that will be used throughout the program.

if (!(display = XOpenDisplay(argv[1]))) {
    perror("XOpenDisplay");
    exit(1);
}

Opens a TCP connection to an X server running at the host specified by argv[1]. If
argv[1] is NULL, it contacts the server running at the same machine where the client is
running.
The format for argv[1] is: host:0
Examples: 128.82.4.67:0
dogwood.cs.odu.edu:0
localhost:0 (same as NULL).

root = RootWindow(display, screen = DefaultScreen(display));

Creates a root window. In X every window must have a parent window
and this is the parent of all other windows.

fgcolor = BlackPixel(display, screen);
bgcolor = WhitePixel(display, screen);

Obtains the pixel values for the black and white colors.

window = XCreateSimpleWindow(display, root, 0, 0, 200, 200, 2, fgcolor, bgcolor);

Creates the application main window on display as child for root at position 0,0.
The window size is 200x200 with border of 2 pixels. The window's foreground color
(fgcolor) is black and its background (bgcolor) color is white.

cmap = DefaultColormap (display, screen);
XAllocNamedColor(display, cmap, colorname, &color, &ignore);
\[\text{fgcolor} = \text{color.pixels};\]
\[\text{gcval.foreground} = \text{fgcolor};\]
\[\text{gcval.background} = \text{bgcolor};\]
\[\text{draw} = \text{XCreateGC(display,window,GCForeground|GCBBackground,\&gcval)};\]

The above statements are used to create a "red" pen called \text{draw} (see the statement: \text{char*colorname} = "red"; at the beginning of the program).

\[\text{XSelectInput(display,window,eventmask)};\]

Ask the server to report the events specified by \text{eventmask} (defined at the beginning of the program as: \text{long eventmask = ButtonPressMask|ExposureMask|KeyPressMask;}).

\[\text{XMapWindow(display,window)};\]

Make the window visible on the screen.

The following loop monitors and process the events sent by the X server

\[\text{for (;;)} \{\]
\[\quad \text{XWindowEvent(display,window,eventmask,\&event);}\]
\[\}

This is a "blocking" call, i.e., the program will stop here until an event arrives from the X server.

\[\text{switch (event.type)} \{\]
\[\quad \text{case Expose:}\]
\[\quad \quad \text{XClearWindow(display,window);}\]
\[\quad \quad \text{break;}\]

Whenever an Expose event arrives, the window is cleared.

An expose event can be generated by e.g., covering and uncovering the window, closing and opening the window.

\[\text{case ButtonPress:}\]
\[\quad \text{XDrawPoint(display,window,draw, event.xbutton.x,event.xbutton.y);}\]
\[\quad \text{break;}\]
Whenever any button is pressed a point is drawn (red color since the draw penn is used) at the x,y position where the event occurred: event.xbutton.x, event.xbutton.y).

---

case KeyPress:
exit(0);
Whenever any Key is pressed the program exits.

default:
    fprintf(stderr,"Unexpected event: %d\n",event.type);
} } }

---

**Drawing Circles**

The program `xcircles.c` is similar to `xpoints.c` but it draws filled circles.

Here is the code that achieve that:

....
int radious = 6;
....

case ButtonPress:
    pointx = event.xbutton.x - radious;
    pointy = event.xbutton.y - radious;
    XFillArc (display,window,draw, pointx, pointy,
            2*radious, 2*radious,0,360*64);
break;

---

**Drawing Lines**

The program `xlines.c` is similar to `xpoints.c` but it draws lines.
The user odd clicks (1, 3, ...) draws a point while the even clicks (2, 4, ...) draws lines between the current position and the previous position of the mouse.
Here is the code that achieve that:

case ButtonPress:
    if (FirstPt) {
        FirstPt=FALSE;
    } else {
        // draw a line...
    }
pointx = event.xbutton.x;
pointy = event.xbutton.y;
XDrawPoint(display, window, draw, pointx, pointy);
break;
Odd clicks draws a point
}
else {
    FirstPt = TRUE;
    XDrawLine(display, window, draw, pointx, pointy,
              event.xbutton.x, event.xbutton.y);
    break;
}
Even clicks draws a line between the previous mouse position and
the current position.
}

---

**Multiuser Applications**

This program *xylines.c* which is similar to *xlines.c* but it draws lines on
multiple windows on different/same displays to allow multiple users to interact with the
same program at the same time. This provide the foundations of what we call "multiuser"
applications or groupware, e.g., group games.

```
#define MAXUSERS 4
#include <X11.../xwindows/xlib.h>
#include <stdio.h>
#define TRUE 1
#define FALSE 0

main(argc, argv)
int argc;
char **argv;
{
    int i;
    int j;
    Display *display[MAXUSERS];
    Window root[MAXUSERS], window[MAXUSERS];
    long fgcolor,bgcolor;
    int screen, pointx[MAXUSERS], pointy[MAXUSERS];
    int eventmask = KeyPressMask|ExposureMask|ButtonPressMask;
    XEvent event;
    XGCValues gcval;
    GC draw[MAXUSERS];
    int FirstPt[MAXUSERS];
    int NC;
```
The above are the declarations used in the rest of the program.

\[ NC = \text{argc-1}; \]

The following loop open the display of NC users.
For example: The program can be called as:

\% xylinies xanth.cs.unc.edu:0 cyclone.cs.odu.edu:0

In such case two windows will appear on xanth and cyclone, assuming each machine is running an X server.

\[
\text{for (i=0; i< NC; i++)}
\text{if (!(display[i] = XOpenDisplay(argv[i+1])))} \{
\text{perror("XOpenDisplay");}
\text{exit(1);}
\}
\]

\[
\text{FirstPt[i] = TRUE;}
\text{root[i] = RootWindow(display[i],screen = DefaultScreen(display[i]));}
\text{fgcolor = BlackPixel(display[i],screen);}
\text{bgcolor = WhitePixel(display[i],screen);}
\]

\[
\text{window[i] = XCreateSimpleWindow(display[i],root[i],0,0,200,200,2,}
\text{fgcolor,bgcolor);}
\]

\[
\text{gcval.foreground = fgcolor;}
\text{gcval.background = bgcolor;}
\text{draw[i] = XCreateGC(display[i],window[i],GCForeground|GCBackground,&gcval);}
\]

\[
\text{XMapWindow(display[i],window[i]);}
\text{XFlush(display[i]);}
\]

The XFlush call forces the client to send the requests it has (flush them) to the server, since X model allow clients to buffer and batch requests and only send them to the server when appropriate.

\[
\text{XSelectInput(display[i],window[i],eventmask);}
\]
The following loop monitors all displays in a non-blocking fashion using XCheckWindowEvent function (in contrast the call XWindowEvent used earlier in xpoints.c was a blocking function).

```c
for (;;) {
    for (j=0; j<NC; j++){

        if (XCheckWindowEvent(display[j],window[j],eventmask,&event))

            switch (event.type) {

            case Expose:
                for (i=0; i<NC; i++)
                    XClearWindow(display[i],window[i]);
                for (i=0; i<NC; i++)
                    XFlush(display[i]);
                break;

            case ButtonPress:
                if (FirstPt[j]) {
                    FirstPt[j]=FALSE;
                    pointx[j] = event.xbutton.x;
                    pointy[j] = event.xbutton.y;
                    for (i=0; i<NC; i++)
                        XDrawPoint(display[i],window[i],draw[i],pointx[j],pointy[j]);
                    for (i=0; i<NC; i++)
                        XFlush(display[i]);
                    break;
                }
                else {
                    FirstPt[j]=TRUE;
                    for (i=0; i<NC; i++)
                        XDrawLine(display[i],window[i],draw[i],pointx[j],pointy[j],
                             event.xbutton.x, event.xbutton.y);
                    for (i=0; i<NC; i++)
                        XFlush(display[i]);
                    break;
                }
            }
            case KeyPress:
                for (i=1; i<NC; i++)
                    XDestroyWindow(display[i], window[i]);
                for (i=0; i<NC; i++)
```
The program `xbuttoms` creates the following interface, it has six buttons, clicking into a button executes the function associated with that button.

```
#define MAX(A,B) ((A) > (B) ? (A) : (B))
#define SB_WIDTH   40      /* Button Width */
#define SB_HEIGHT   15      /* Button Height */
#define SB_LONGLABEL   50      /* Longest MenuButton label */
#define SB_CURSOR   XC_hand1   /* Cursor for inside MenuButton */
This will change the cursor shape to hand1 whenever it enters a button.

#define SB_INPUTMASK   ExposureMask | EnterWindowMask | LeaveWindowMask | ButtonPressMask | ButtonReleaseMask
XSetWindowAttributes setwinattr;
```
int ls(), du(), pwd(), w(), clear(), csh(), date(), cal(), Exit;
int chldeath();

struct {
    Window window;
    char label[SB_LONGLABEL];
    int (*func)();
    char active;
    int pid;
} MenuButton[9];

For each button there is an entry that holds information about the button:

- window id,
- its label (e.g., ls),
- a pointer to a function to be executed whenever the button is activated,
- a flag to indicate whether the button is active or not, and
- the process id of the process that executes the button's function.

#define POSX 800
#define POSY 200
#define WIDTH 142
#define HEIGHT 85

#define lsButton 0
#define wButton 1
#define dateButton 2
#define duButton 3
#define clearButton 4
#define calButton 5
#define pwdButton 6
#define cshButton 7
#define quitButton 8

Display *display;

Window main_window; /* The main utility window */
XEvent event; /* Incoming event */
int screen; /* Display screen number */
GC gc; /* A Graphics Context to use */
XGCValues values;
XFontStruct *font_info;

char *dname;
The above definitions will be used throughout the program.

---

```c
main(argc, argv)
int argc;
char **argv;
{
  int i;

  signal(SIGCHLD, chldeath);

  To catch the SIGCHLD signal whenever a child process is terminated in order to clean up the process info by invoking the function chldeath.

  dname= (char *) getenv("DISPLAY");

  To get the DISPLAY value in order to open windows on that display.

  /* Connect to the X Server */
  if ( (display=XOpenDisplay(argv[1])) == NULL ) {
    fprintf(stderr, "Could not open display");
    exit(1);
  }

  screen = DefaultScreen(display);
  /* Create a Window with geometry WIDTHxHEIGHT+POSX+POSY */
  main_window = XCreateSimpleWindow( display, RootWindow(display,screen),
                                        POSX, POSY, WIDTH, HEIGHT, 2, BlackPixel(display,screen),
                                        WhitePixel(display,screen) );

  load_font();

  /* Create a default graphics context */
  values.foreground = BlackPixel( display, screen );
  values.background = WhitePixel( display, screen );
  gc = XCreateGC( display, main_window,
                  GCForeground|GCBackground, &values);
```

---

MakeButton( 1,  1, "ls", ls, lsButton);
MakeButton( 1, 31, "du", du, duButton );
MakeButton( 1, 61, "pwd", pwd, pwdButton );
MakeButton( 50,  1, "w",w, wButton );
MakeButton( 50, 31, "clear",clear,clearButton );
MakeButton( 50, 61, "csh", csh,cshButton );
MakeButton( 99,  1, "date",date, dateButton );
MakeButton( 99, 31, "cal", cal, calButton );
MakeButton( 99, 61, "Quit",Exit, quitButton );

To make the 9 buttons, column by column. The arguments are:

- the x, y coordinates of the upper-left corner (e.g., 1,1),
- the button label (e.g., "ls"),
- the function to be executed (e.g., ls),
- the index of that button in the MenuButton array

(e.g., lsButton which is 0).

XSelectInput( display, main_window, ExposureMask );
XMapWindow(display, main_window);

for( ;; ) {
  for(i=0; i<=8;i++)
    if (XCheckWindowEvent(display, MenuButton[i].window, SB_INPUTMASK, &event ))
      HandleButton( i, &event );

This loops over each window button i, and if there is an event handle it.
}

------------------

load_font()
{
  char *fontname = "9x15";
  if ((font_info = XLoadQueryFont(display, fontname)) == NULL) {
    (void) fprintf(stderr,"Could not get font\n");
    exit( -1 );
    XSetFont(display, gc, font_info->fid);
  }
}

MakeButton( x, y, label, fun, id)
int id;
int x,y; /* Where to put it */
char *label; /* What to put in it */
int (*fun)();
{ 
    Cursor tempcursor;

    strncpy( MenuButton[id].label, label, SB_LONGLABEL );

    MenuButton[id].func = fun;

    MenuButton[id].active = FALSE;

    MenuButton[id].window = XCreateSimpleWindow( display, main_window, 
        x, y, SB_WIDTH, SB_HEIGHT, 1,
        BlackPixel(display,screen), WhitePixel(display,screen) );

    Fills in the MenuButton structure for button i.

    XSelectInput( display, MenuButton[id].window, SB_INPUTMASK );
    tempcursor = XCreateFontCursor( display, SB_CURSOR );
    XDefineCursor( display, MenuButton[id].window, tempcursor );

    Change the cursor to hand1 whenever the cursor is inside the window.

    setwinattr.backing_store = Always;
    XChangeWindowAttributes(display, MenuButton[id].window, 
        CWBackingStore, &setwinattr);

    Save the contents of a window whenever it is covered in order to restore it back
    whenever it uncovered.

    XMapWindow( display, MenuButton[id].window );
}

====================================

int HandleButton( id, event )
int id;
XEvent *event;
{

    if (MenuButton[id].active) return;
    switch( event->type ) {
    case Expose:
        ExposeButton( id );
        break;
    
    case EnterNotify:
        XDrawRectangle( display, MenuButton[id].window,
            gc, 1,1, SB_WIDTH-3, SB_HEIGHT-3 );
        break;

    Draws a smaller rectangle inside the button.
case LeaveNotify:
    ExposeButton( id );
    break;

case ButtonPress:
    if ( event->xbutton.button == Button1 ) {
        XDrawRectangle( display, MenuButton[id].window, gc, 0,0, SB_WIDTH-1,SB_HEIGHT-1 );
    }
    break;
if the left button (Button1) is pressed, another rectangle to "darken" the border of the button indicating that the button is active.

case ButtonRelease:
    MenuButton[id].func();
    break;
Upon the release of a pressed button, the function associated with the button is executed.

default:
    break;
}
}

================================

ExposeButton( id )
int id;
{
    int width, center;

    XClearWindow( display, MenuButton[id].window );

    width = XTextWidth( font_info, MenuButton[id].label, strlen(MenuButton[id].label) );
    center = MAX((SB_WIDTH-width)/2,4);
    XDrawString( display, MenuButton[id].window, gc, center, font_info->ascent, MenuButton[id].label, strlen(MenuButton[id].label) );
    XFlush(display);
}

ExecButton(id, label)
int id;
char *lable;
{

if (MenuButton[id].active) return;
If the button is already active, return.

MenuButton[id].active=TRUE;
if ( (MenuButton[id].pid=fork()) == 0 )
  execlp ("xterm","xterm","-display",dname,"-T",label,"-e","./call",label, NULL);
Create a new process (using UNIX fork()) and execute an xterm.
The xterm appears on -display dname and its title -T label.
Inside the xterm we execute -e ./call label.
./call is a shell script with the following content:

$1
  read x
That means the first argument $1 will be executed (e.g. "ls -lt") and then
the read statement will block until a user enter any character inside the xterm.

}

ls()
{
  ExecButton(lsButton,"ls -lt");
}

du()
{
  ExecButton(duButton,"du");
}

pwd()
{
  system("pwd");
}

w()
{
  ExecButton(wButton,"w");
}
clear()
{
  system("clear");
}
csh()
{
    ExecButton(cshButton, "csh");
}

date()
{
    ExecButton(dateButton, "date");
}

cal()
{
    ExecButton(calButton, "cal");
}

======================

Exit()
{
    int i;
    printf("BY BY\n");
    for(i=0; i<8; i++){
        if (MenuButton[i].pid != 0 )
            kill(MenuButton[i].pid, SIGKILL);
    }
    XDestroyWindow(display,main_window);
    exit(0);
}

chldeath()
{
    int status;
    int cpid;
    while ((cpid = wait3(&status, WNOHANG, 0)) > 0 ) {
        inform(cpid);
    }
    signal(SIGCHLD, chldeath);
}

inform(cpid)
int cpid;
{
    int i;
    for (i=0; i<9; i++)
if ( cpid == MenuButton[i].pid){
    MenuButton[i].active=FALSE;
    MenuButton[i].pid = 0;
    ExposeButton(i);
}

Find the button i associated with the dead process and cleanup the button's data structure MenuButton[i]