CS 349 - User Interfaces **Winter 2018**

Lecture 3: January 8, 2018

Lecturer: Keiko Katsuragawa *Notes By: Harsh Mistry*

3.1 Windowing Systems

3.1.1 Windowing System

- Handles input device events
- Exposes output methods to display graphics
  - basic drawing primitives, bitmaps, text
- Manages windows as a place for visual application content
- A **windowing system** provides ”low-level” input, output, and window management capabilities to the operating system.

3.1.2 X windows

- Developed in 1984
- Standard windowing system for Unixes
- Free and Cross-Platform
- Unix standard windowing system
  - handles input, draws graphics, create windows, ...
  - free and cross-platform
- Essentially a protocol
  - does not specify style of user interface
  - not a ”window manager”
  - A windowing system provides ”low-level” input, output and window management capabilities to the operating systems

3.1.2.1 X Windows Design Criteria

- Implementable on a variety of displays
- Applications must be device independent
- Must be network transparent
- Support multiple, concurrent application displays

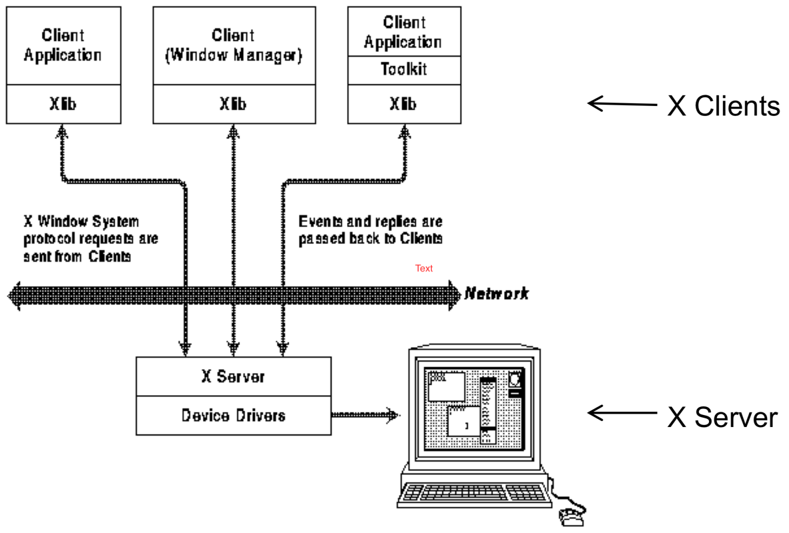
- Support output to overlapping windows (... even when partially obscured)
- Support a hierarchy of resizeable windows
- Support many different applications
- High-performance, high-quality text, 2-D graphics, imaging ..
- System should be extensible

3.1.2.2 X Client Server Architecture

- An X Client handles all application logic
- An X server handles all display output and user input
- Server handles request from client, process data as requested, and returns results to client

3.1.2.3 Why Client-Server?

- Goal was flexibility and economy
- Many X clients can exist, while only one X server delvers the user interface



3.1.2.4 Structure of a typical X program

- Perform X Client initialization
- Connect to the X server
- Perform X related initialization
- Event loop

```
get next event from the X Server
handle the event:
    if the event was a quit message, exit the loop
do any client-initiated work
send drawing requests to the X Server
```

- close down the connection to the X Server
- perform client cleanup

3.1.2.5 Xlib

- library to wrap low level X Window protocol
  - to avoid implementing message passing for every new program
- uses buffered input and output queues
  - need to flush them: XSync, XFlush
- Xlib functions:
  - connection operations: e.g. XOpenDisplay, XCloseDisplay, ...
  - connection operation requests: e.g. XCreateWindow, XCreateGC,...
  - connection information requests: e.g. XGetWindowProperty, ...
  - local event queue operations: e.g. XNextEvent, XPeekEvent, ...
  - local data operations: e.g. XLookupKeysym, XParseGeometry, XSetRegion, XCreateImage, XSaveContext, ...
- Xlib data types:
  - Display, Window, GC, XSizeHints, XWhitePixel, XBlackPixel, etc.

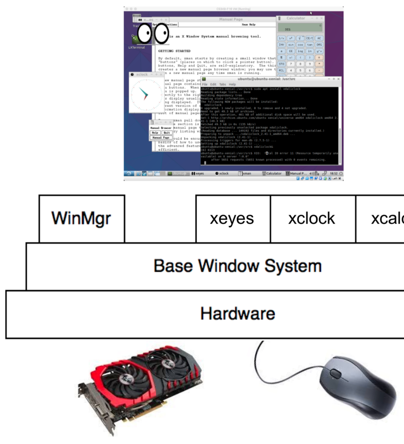
3.1.2.6 Compile and run an X application

```
g++ -o null null.cpp -L/usr/X11R6/lib -lX11 -lstdc++ ./null
```

3.1.2.7 Sample Code

Refer to Pages 14-16 of course slides for code examples

3.1.3 Windowing System Architecture



3.1.3.1 Base Window System (BWS)

- Lowest level abstraction for windowing system
- Has routines for creating, destroying, managing windows
- Also routes mouse and keyboards input to correct window
- Ensures only one application changing frame buffer (video memory) at a time

3.1.3.2 Canvas Abstraction

- BWS controls application's access to the window contents using a ”drawing canvas abstraction”
- The application is shielded from details of frame buffer, visibility of window, and all other application windows
- Each window has its own coordinate system

3.1.3.3 Window Manager

- Provides interactive components for windows (menus, close box, resize capabilities)
- Creates the look and feel of each window
- Application owns the contents of the window, but the WM owns the application window itself!
- Separating the BWS from the Window Manager enables :
  - Alternative look and feels for windowing system
  - Different windowing paradigms (i.e. Xmonad for tiled windows)
- Additionally separating the BWS from the Window Manager results in a more robust implementation since BWS and WM are separate processes