The Xynth Windowing System

Table of Contents

1. Xynth Introduction

Xynth is an embedded and portable interface between display hardware (the mouse, keyboard, and video displays) and the desktop environment that works on many hardware, including embedded devices.

1.1 What's Windowing System

A windowing system is a system for sharing a computer's graphical display presentation resources among multiple applications at the same time. In a computer that has a graphical user interface, you may want to use a number of applications at the same time. Using a separate window for each application, you can interact with each application and go from one application to another without having to reinitiate it. Having different information or activities in multiple windows may also make it easier for you to do your work.

A window system enables the computer user to work with several programs at the same time. Each program runs in its own window, which is a rectangular area of the screen. Most window systems allow windows to overlap, and provide means for the user to perform standard operations such as moving/resizing a window, sending a window to the foreground/background, minimizing, maximizing a window, etc.

From a programmer's point of view, a window system implements graphical primitives such as rendering fonts or drawing a line on the screen, effectively providing an abstraction of the graphics hardware.

For human-computer interaction. WIMP stands for the "window, icon, menu, pointing device" paradigm that characterizes most commercial graphical user interfaces from 1984 to the present. It was developed at the Xerox Parc.

1.2 What's Xynth

The name Xynth comes from the coordinate system, which is the heart of the Xynth Windowing System design.

Xynth is a GPL licensed free software project. It aims to provide a lightweight GUI supported windowing system for Linux-based and/or real-time embedded systems. Launched at the end of 2002, the software has, over 2 years of development, for the moment become a stable and reliable one securing widespread applications in a variety of products and programs.

It is not a window manager or a window server, though it provides features similar to both. In the X Window System, for example, Xynth would be roughly equivalent to the X server plus a window manager. On the other hand, Xynth brings some of the features of modern graphical windowing systems to the embedded programming community, aiming to create a new GUI architecture designed to be usable on systems ranging from embedded to desktops.

1.3 Requirements of Real-time Embedded Systems for GUI

Typical computer desktop "graphics stack" isn't well suited to embedded applications.

Embedded devices frequently have highly constrained resources and can afford neither the program storage space nor the memory footprint of desktop graphics software. All that memory costs money, requires board space, and consumes power. Embedded systems frequently have unique needs that can't be met by desktop graphics system components. These include the requirement for a customized look and feel, control over what functions are available to users, speed of loading, unusual display or input device characteristics, etc.

1.4 Technical Highlights of Xynth

The Xynth Windowing System is scalable, customizable, portable, modular, reliable, lightweight, fast, and operating system independed because of these fact below.

1.4.1 Client/Server Architecture

Xynth uses a network layer comminication design between the server and clients. TCP/IP, Unix Domanin Sockets, or minimal socket stack of our own may be choosen as network layer, this increases the portability, and independency. So even if the operating system has no network support/stack this will never be a problem for development.

1.4.2 Direct Video Memory Access

On a client/server architectured windowing system, when a client wants to draw a pixel to the screen, it passes the drawing data to the server and then server writes the data to the video buffer.

Unlike any other competitors using client/server architecture, all clients and server has direct access to the video memory buffer. This means that any client that is drawing on the screen, will never wait for the server response. Another income of this design is the huge decrease of the load on server, and network layer. As a result, the system is high-performed and light-weighted.

1.4.3 Double Buffer Rendering

Although the direct access to video memory hase valuable advantages it has a handicap, flickering. To avoid flickering double buffer rendering used in clients. While double buffering increasing the allocated memory, also inreases the speed due to the windowing system concept (refresh, move, resize). On the other hand if needed, this feature may be switched off.

1.4.4 Threads Safe

All the code in Xynth source package is threads safe, including server api, client library, and widget set. This feature gives the developer a customizable, and reliable platform.

1.4.5 Internal Driver Structure

The server manages input devices with polling on them. Any custom device driver can easily plugged or unplugged, within the source code a keyboard (console) and mouse driver set (PS2, IMPS2, USB, etc.) is given. This driver structure gives modularity to the system.

Xynth has output driver system set, drivers included within the source are svgalib and linux frame buffer implementation. Any output driver can be easily used for video memory buffer.

1.4.6 SysV IPC

Any modern, embedded, or real-time operating system has the support of IPC (Inter Proccess Communication). As a result of this fact, system gains reliablity, independency, and speed.

1.4.7 Built in Window Manager

Again, unlike any competitor Xynth server has minimal set of window managing ability which reduces the foot print.

1.4.8 Ported Applications, APIs

Mplayer-1.0pre5, well known award winning media player.

SDL-1.2.7, low level graphics library.

GTK+-2.4.13, high level widget library written in C. Any program written with GTKv2 is ready to run on Xynth (Mozilla, Abiword, Xchat, Gqview, Gaim, Anjuta, etc.)

Links-2.1pre15, an embedded web browser with HTML 4.0 and Javascript support.

1.5 Features of Xynth

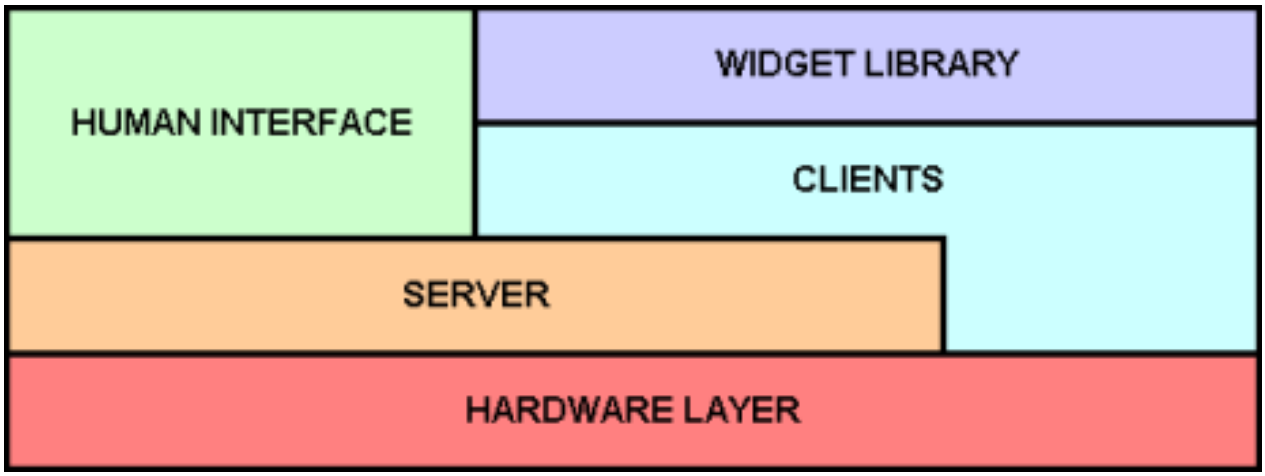
- Network Layer
- Direct Memory Access

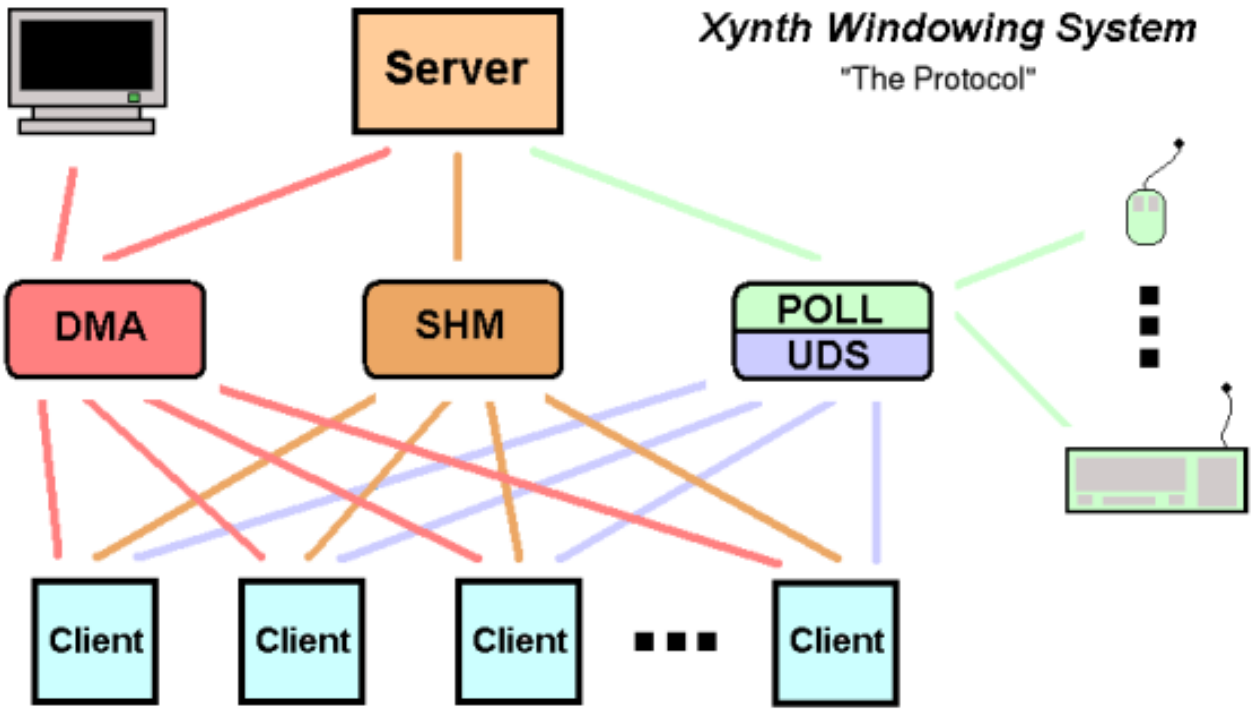
- Overlapped client window - server management
- 8-way Move, Resize
- Runtime Theme Pluging Support
- Built-in image renderer xpm, png
- Antialiased fonts with Freetype Library.
- No dependencies except FBDev or SVGALib
- Device independent basic low-level graphics library
    - rgbcolor, colorrgb, setpixel, getpixel, hline , vline, fillbox, putbox, putboxmask, getbox, putboxpart, putboxpartmask, copybox, scalebox, getsurface, setsurfacevirtual, setsurface
- Overlay Drawing Ability
    - rgbcolor_o, colorrgb_o, setpixel_o, getpixel_o, hline_o, vline_o, fillbox_o, putbox_o, putboxmask_o, getbox_o, putboxpart_o, putboxpartmask_o, copybox_o, getsurface_o, setsurfacevirtual_o, setsurface_o
- Anti Flicker Double Buffer Rendering
- Threads Safe
- Built-in window manager.
- Applications / Libraries
    - XynthDesktop
    - XynTherminal
    - XynthLoad
    - XynthMines
    - Mplayer (Perfect)
    - links (Perfect)
    - SDL (Perfect)
    - GTK 2.4.x (Perfect)
- Low Memory and CPU Usage and Foot Print.
    - In 1024x768x32bits mode with 253 clients open Memory usage is ~2,5M
    - Static linked binary is ~125K
    - Minimal booting Linux distrobution will be less than 2 Mb.

2. Milestones of Xynth

- Socket API
- Windowing System Design
- Svgalib
- Mouse / Keyboard drivers
- Polling System
- SysV IPC API
- Direct Memory Access
- Posix Threads
- Frame Buffer Driver API
- Theme Support
- Image API
- True Type Anti-Aliased Font API
- GTKv2 port
- SDL port
- Mplayer port
- Embedded distrobution design
- Links port, and further developing

3. System architecture of Xynth





4. What is in source package

- Demo
  - Child : Demo program using client library and low-level graphics library to show the usage of child windows, handlers.
  - Desktop : Demo program using client library and low-level graphics library to show the usage of child windows, temp windows, handlers, and desktop manager.
  - Load : Demo program using client library and low-level graphics library to show the usage of threads mechanism.
  - Mines : Demo program using widget library to show the usage of widget library.
  - Simple : Demo program using client library and low-level graphics library to show the usage of main windows.
  - Temp : Demo program using client library and low-level graphics library to show the usage of temp windows.
  - Term : Demo program using client library and low-level graphics library to show the usage of child windows, poll system.
  - Widget : Demo program using widget library to show the usage of widget api.
- Src
  - Fonts : TrueType fonts, that clients will use.
  - Lib : Client library, plus low-level graphics library.
  - Server : Server program.
  - Themes : XPM themes, that will converted to .so shared plugins.
  - Widget : Widget library.
- Tools
  - Bdf : Tool to convert .bdf font files to .c or .h source files.
  - Freetype : Tool to show how to use freetype on svgalib.

- Gaim : Gaim patch, to make it work on Xynth.
- Gtk : GTK+-2.4.x port patch.
- Kmap : Tool to create keyboard map files for Xynth.
- Links : Links2 port patch.
- Mplayer : Mplater port patch
- Mtrr : Tool to show the usage of MTRR.
- Png : Png tool using libpng.
- Sdl : SDL port patch.
- Theme : Tool to convert XPM theme files to .so shared plugins.

5. Why use Xynth

- ANSI C
- Simple architecture
- Minimum dependency
- Fast
- Small
- Portable
- Easly customizable
- Suitable for fast developments
- Work on many operating systems

5.1 Usage areas of Xynth

- Handheld consumer products PDAs, cellphones
- Factory automated equipments
- Settop-boxes, TVs, Kiosks, ATMs
- Medical instruments
- Commercial airlines, cockpit displays, terminals
- Desktop systems
- Any device that has graphical user interface
- Military devices

5.2 Comparison to others

	Xynth	MiniGUI	MicroWindows	DirectFB	QT/Embedded	PicoGUI
Development Status	Stable, Continues	Stable, Continues	Unstable, Continues	Stable, Continues	Stable, Continues	Unstable, Discontinued

Portability	Excellent	Excellent	Excellent	Good	Fair	Better
License	GPL	Commercial	LGPL/Commercial	LGPL	QPL/Commercial	LGPL
Robustness / Reliability	Excellent	Good	Very Poor	Good	Bad	Very Poor
Threads Safe	Yes	Yes	Yes	No	Yes	No
Configurability and Customizabil İty	Excellent	Good	Poor	Fair	Poor	Poor
Consumption of system resources	Lowest	Lower	Low	High	Highest	High
Efficiency	High	High	Low	Low	Lowest	Low
Footprint	100K	500K	600K	1 MB	1,5 MB	1 MB
Ported Libraries	SDLv2 GTKv2	None	FLTKv1	GTKv1	None	SDLv2
Ported Applications	Mplayer Links Mozilla	Xine Konquerror	Smpeg	None	None	Xchat

6. Xynth Live Embedded Distribution CD

6.1 What is in CD

- Doc : Documentation
- Opt : Pre compiled binaries of ported demo applications
    - GTK+-2.4.13 : Xynth port of GTK.
        - Xchat 2.4.0
        - Gqview 1.4.5
        - Gimp2
    - SDL-1.2.7 : Xynth port of SDL
        - Ltris-1.0.7
        - Lbreakout2-2.1.5
    - Mplayer 1-0pre5 : Xynth port of mplayer
    - Links2 : Xynth port of Links
- Src
    - Xynth-0.6.11 : Xynth source
    - Xynth-pkg-0.1.0 : Xynth extras package to prepera your own embedded distrobution.

6.2 How to use

Just put the bootable Xynth Demo CD to the first cdrom device. And boot the machine. At this point this is important that you should have a IMPS2 mouse plugeed to PS2 port, and a vesa competible vga card.

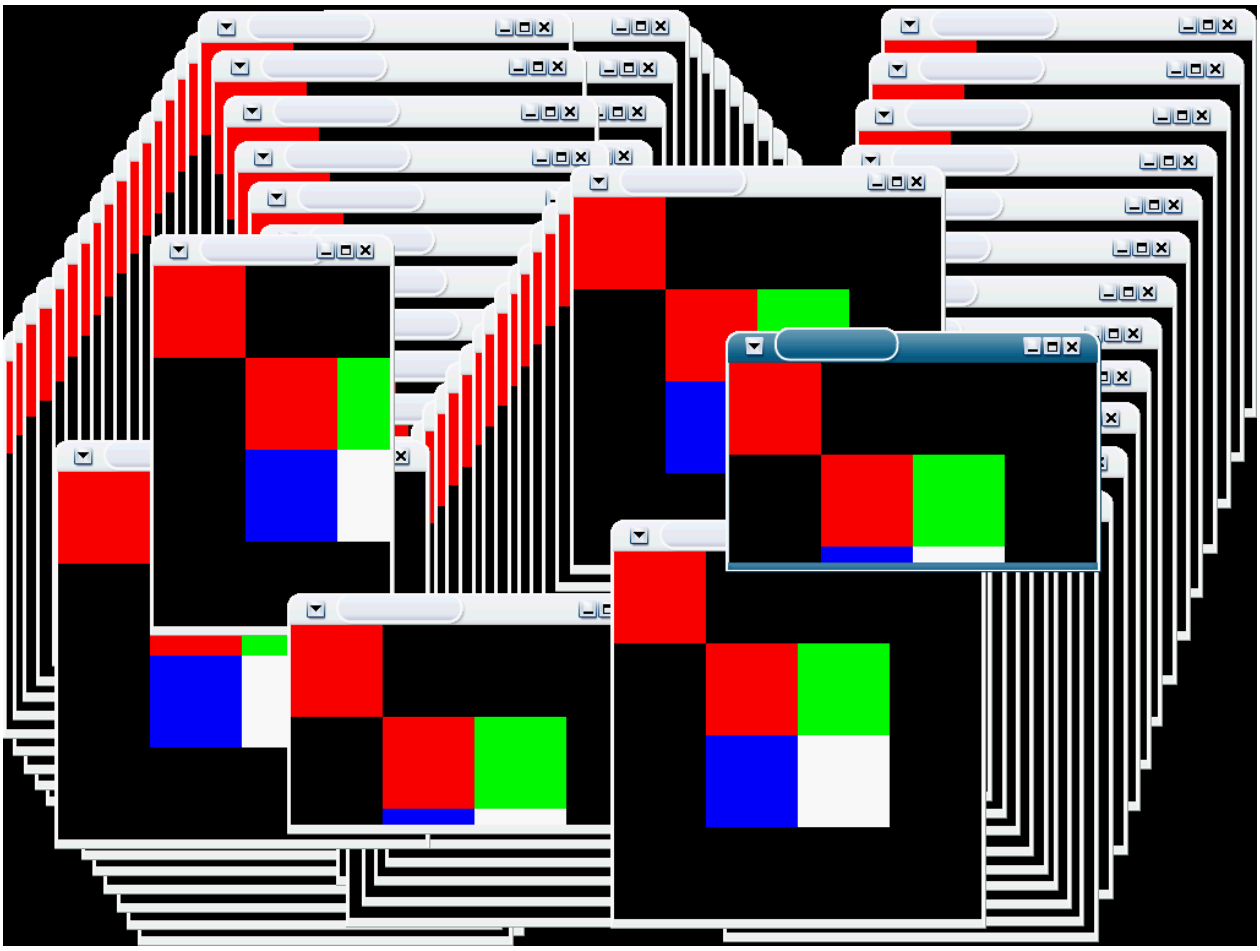
6.3 Howto prepare your own

- tar –jxvf xynth-source.tar.bz2
  cd xynth
  edit Makefile.inc
  make
  make install
- tar –jxvf xynth-pkg.tar.bz2
  cd xynth-pkg
  edit install
  ./install
- Prepare a mini distrobution.
- Copy the binaries you just compiled in to your distrobution.
- Burn them all as a bootable cd.
- Ready to run.

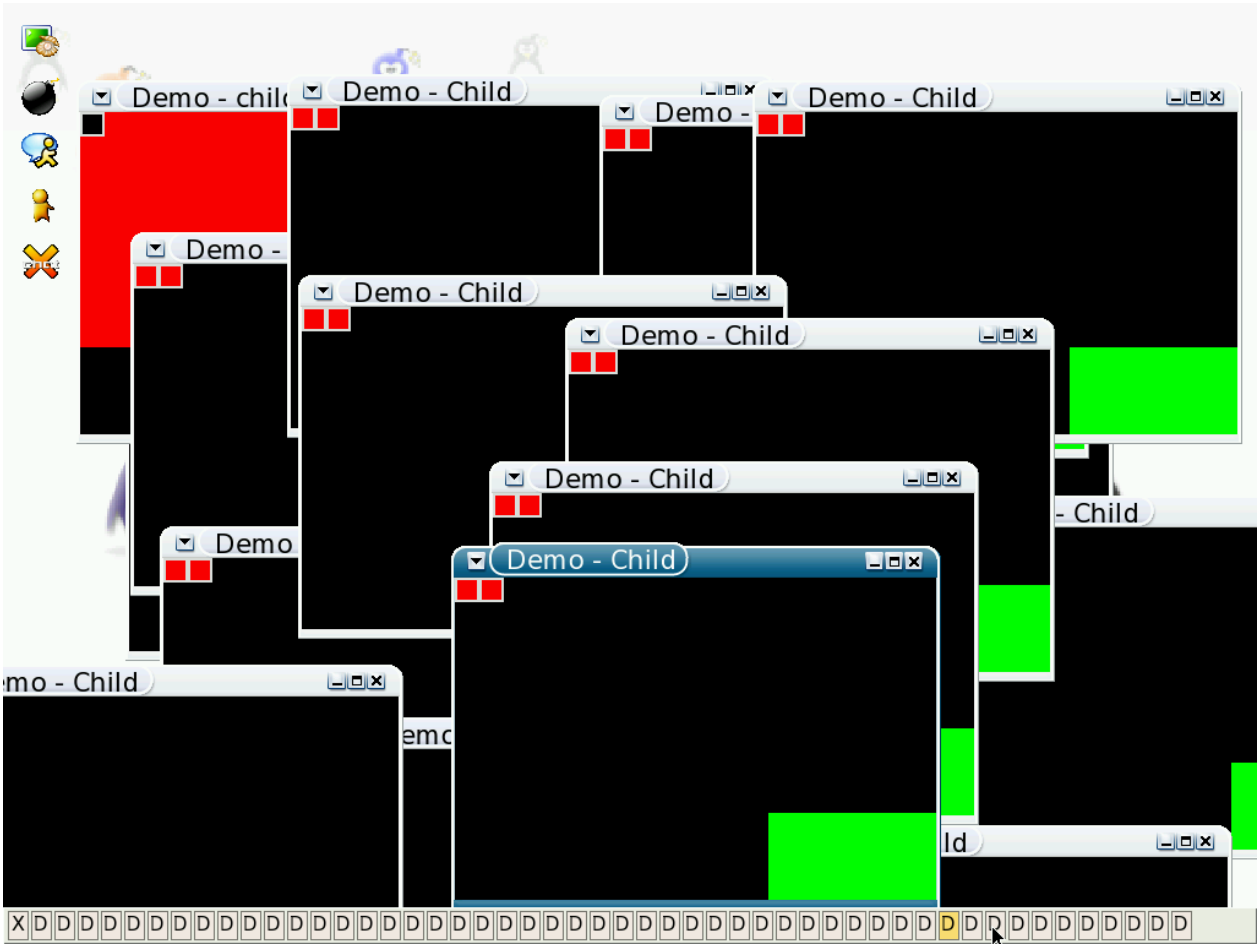
6.4 Tested Platforms

CPU	VGA
amd athlon 2500+	Nvidia FX5700
amd athlon 2500+	Nvidia FX5200
amd athlon 2500+	ATI 9200SE
intel p4 2800	I865
intel p4 2000	I810
intel p4 2000	I845
intel p3 1000	Nvidia Gforce4
intel celeron 1.7	Nvidia TNT2 M64 16Mb
intel celeron 800	Nvidia Gforce2 200mx

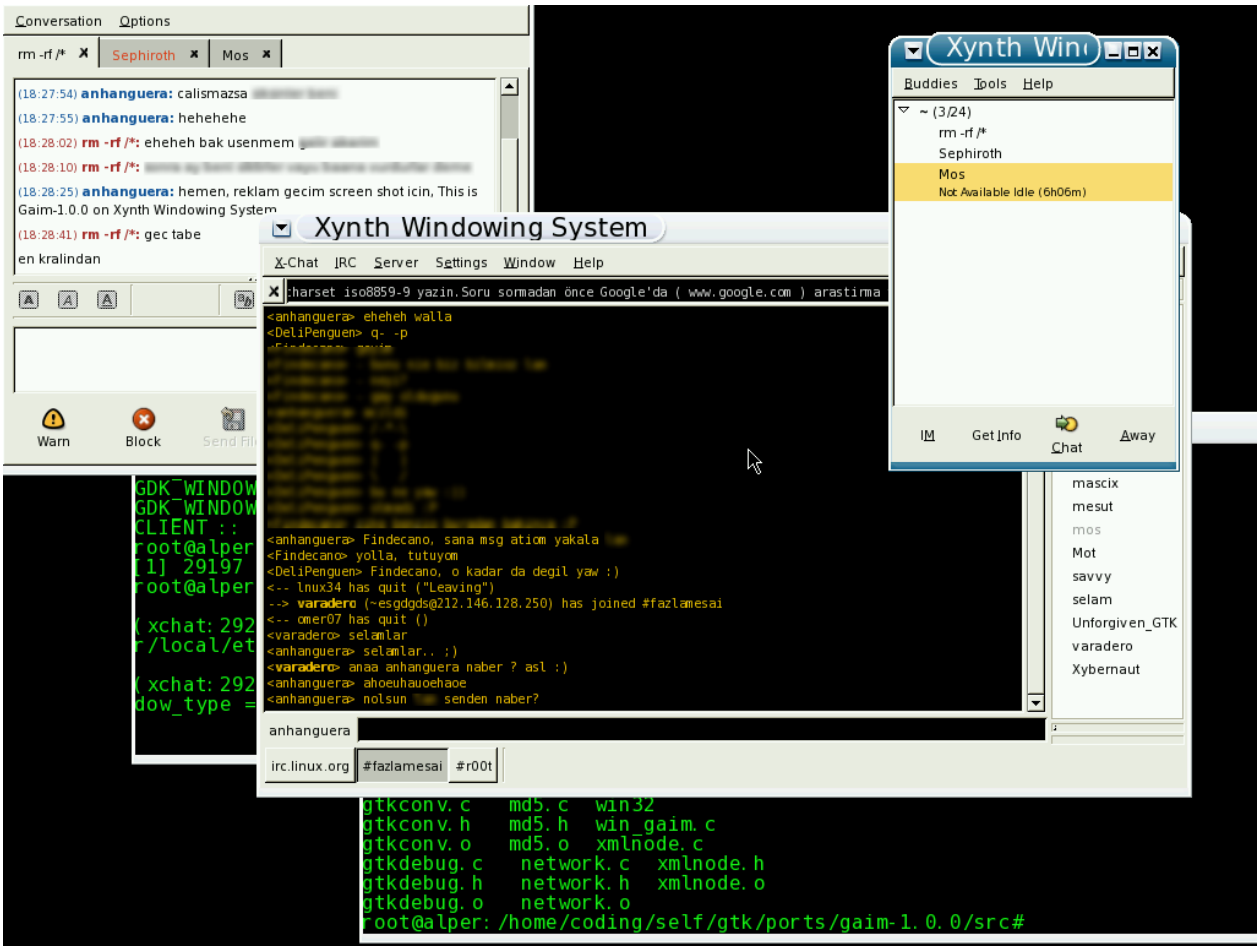
6.5 Screen Shots



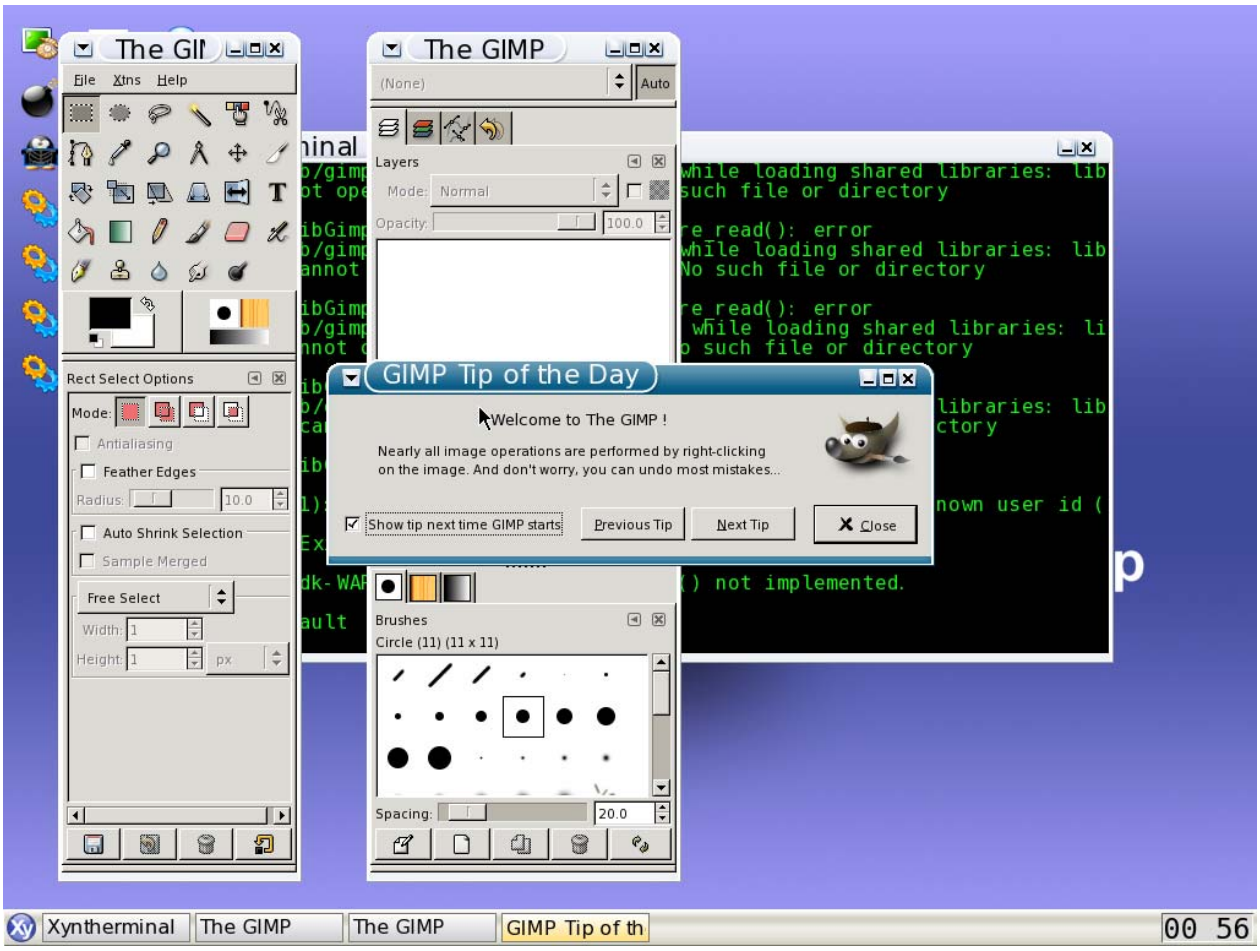
Server + 253 Clients



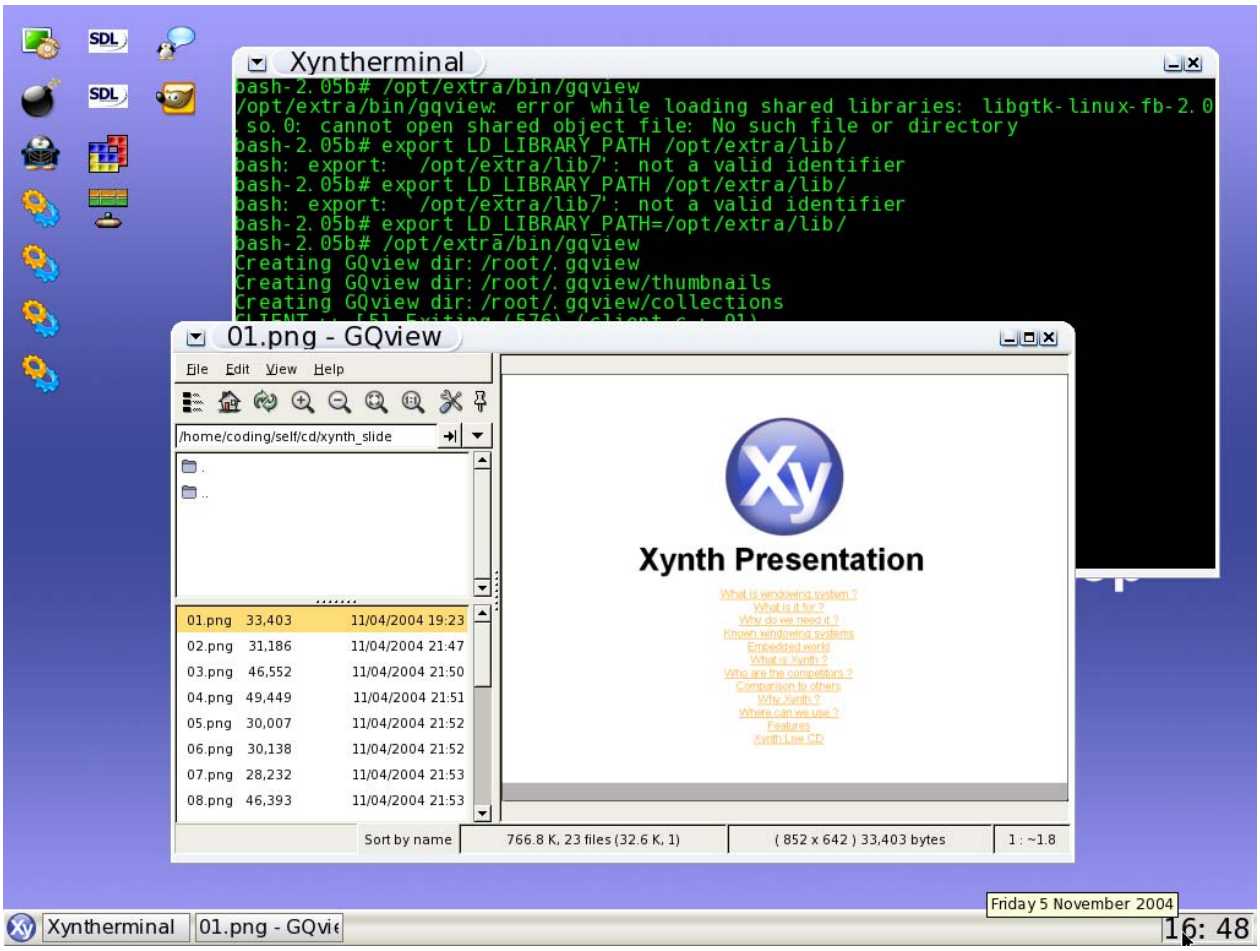
Desktop Demo



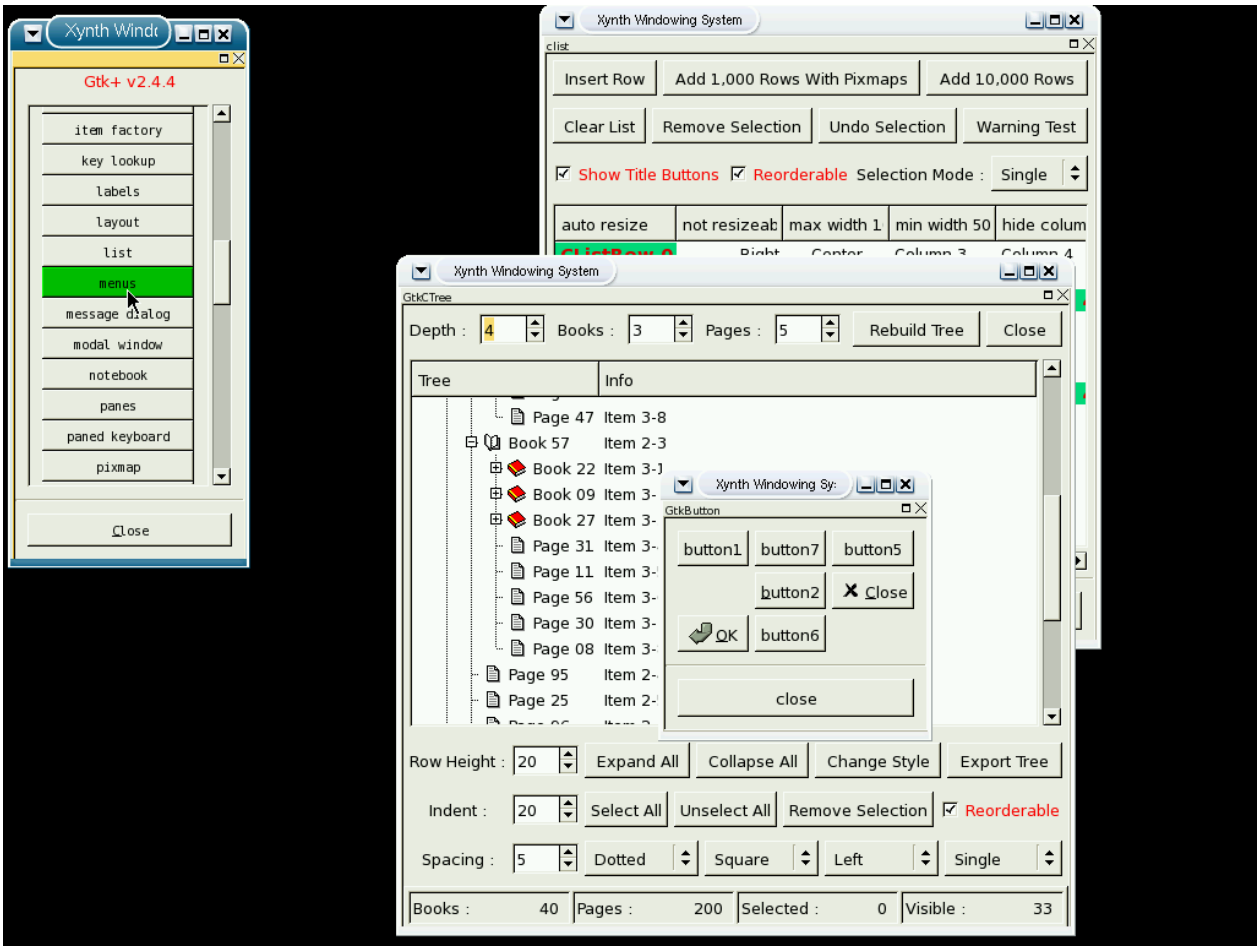
GTK port - Gaim



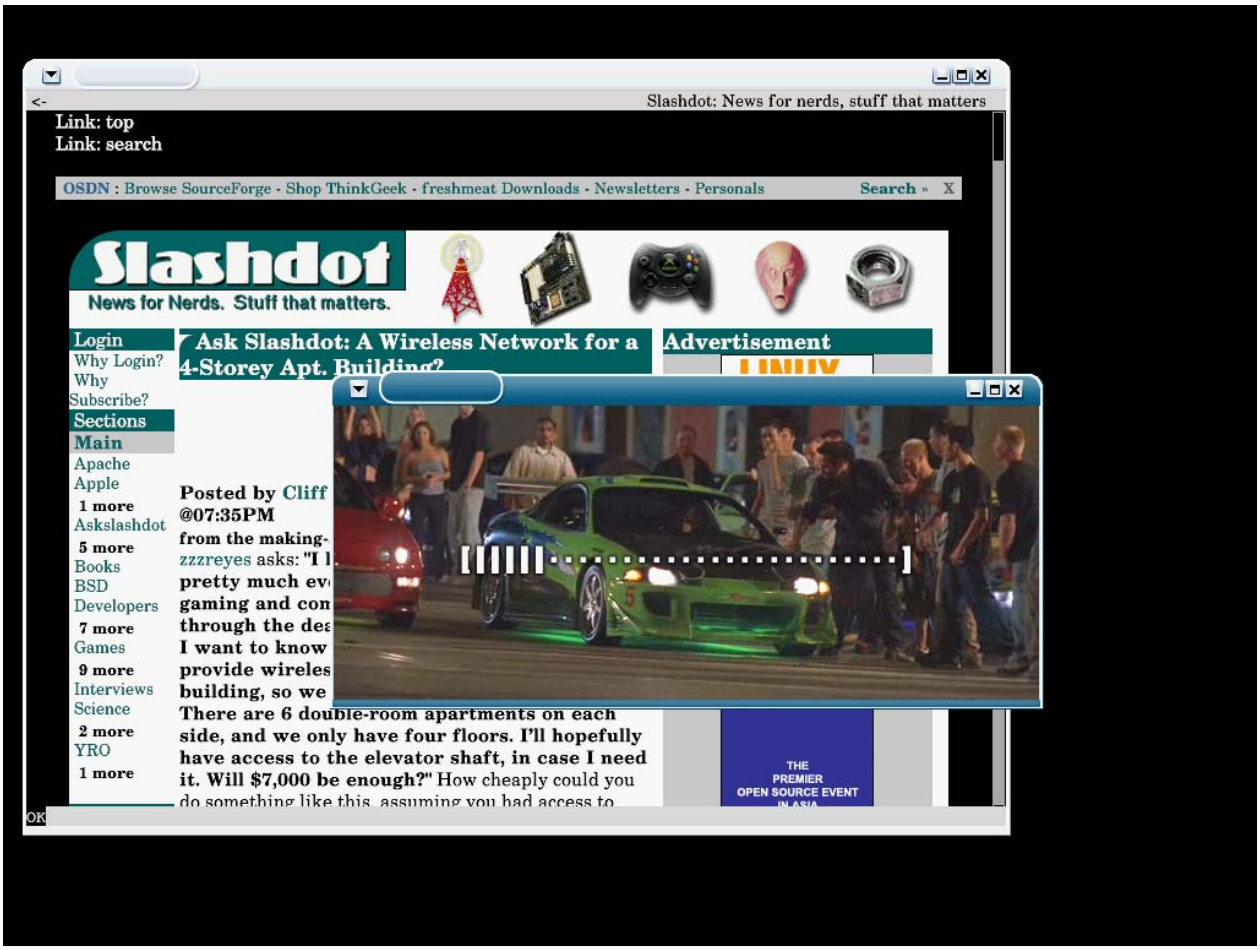
GTK port – Gimp



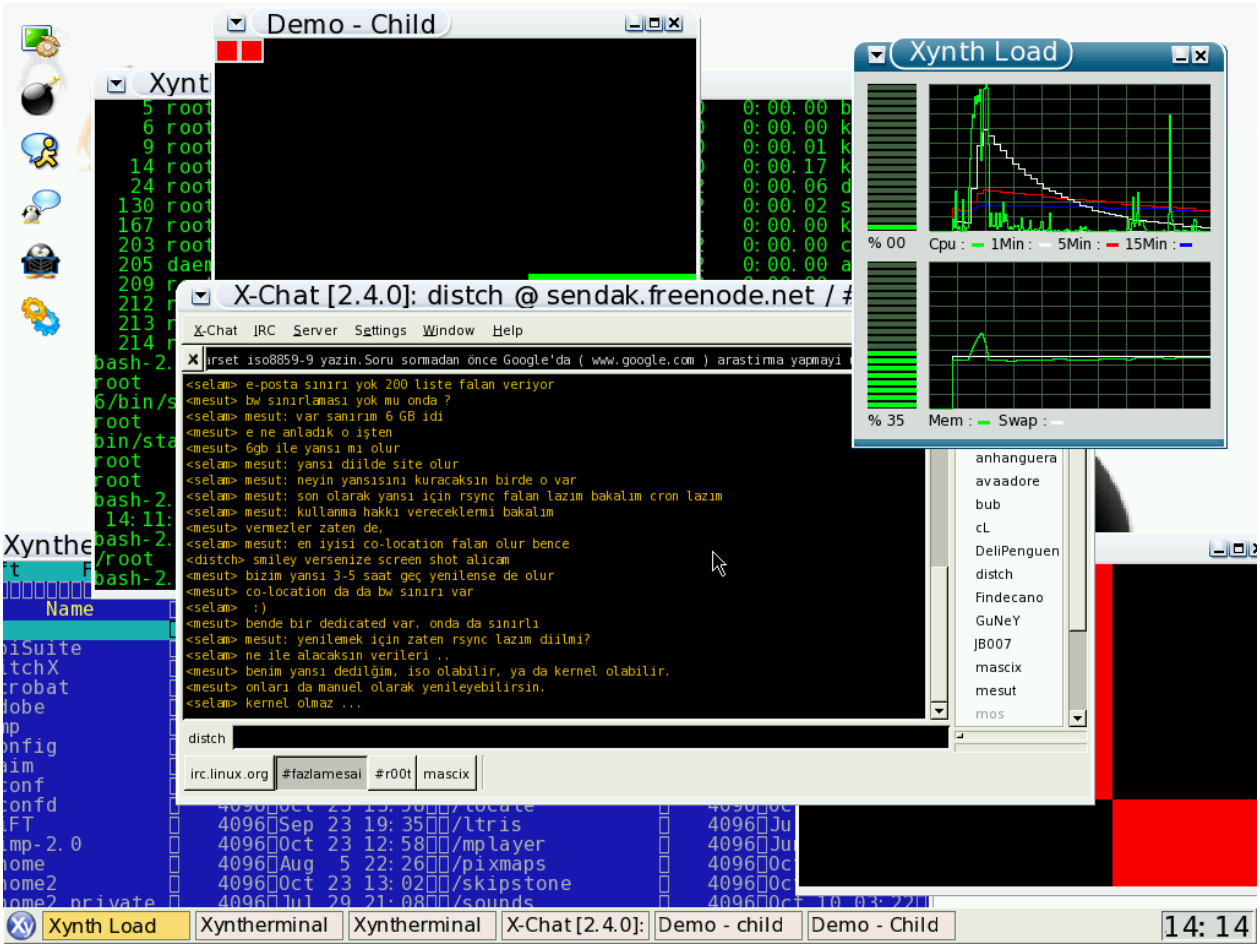
GTK port – gqview

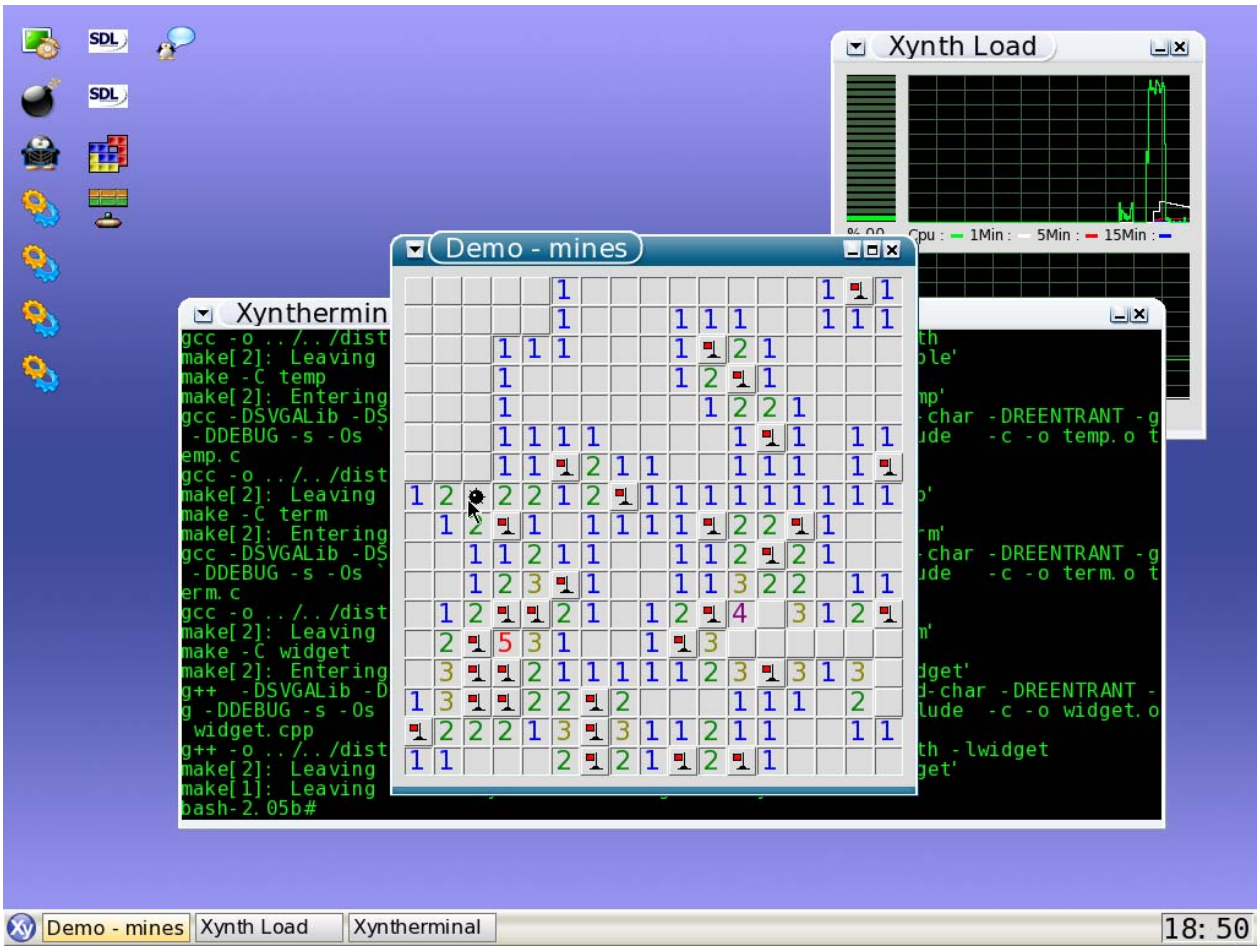


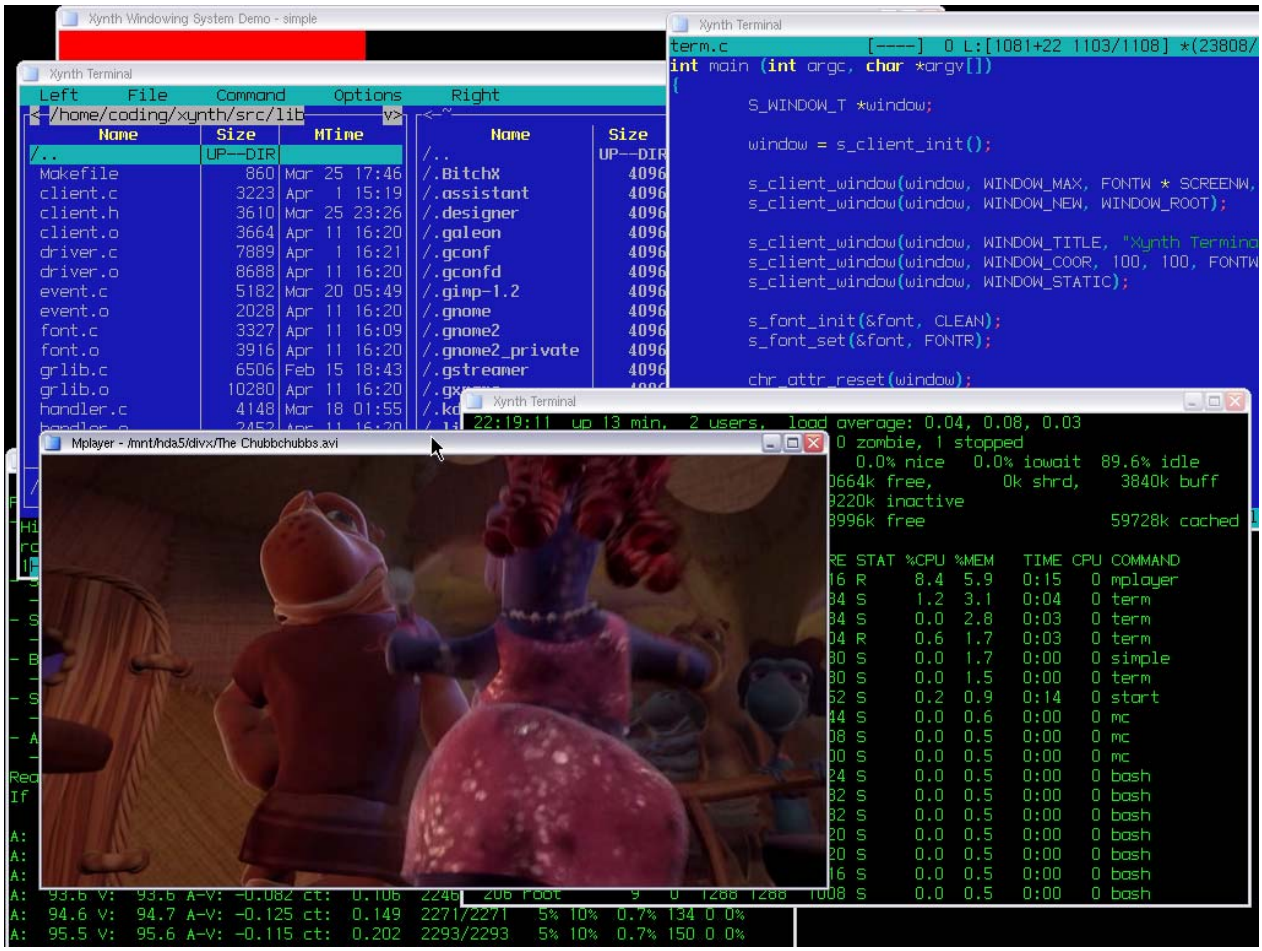
GTK port – gtk demo


<-
Slashdot: News for nerds, stuff that matters
Link: top
Link: search
OSDN : Browse SourceForge - Shop ThinkGeek - freshmeat Downloads - Newsletters - Personals
Search » X
Slashdot
News for Nerds. Stuff that matters.
Login
Why Login?
Why Subscribe?
Sections
Main
Apache
Apple
1 more
Askslashdot
5 more
Books
BSD
Developers
7 more
Games
9 more
Interviews
Science
2 more
YRO
1 more
Ask Slashdot: A Wireless Network for a 4-Storey Apt. Building?
Posted by Cliff
@07:35PM
from the making-
zzzreyes asks: "I
pretty much ev
gaming and con
through the de
I want to know
provide wireles
building, so we
There are 6 double-room apartments on each side, and we only have four floors. I'll hopefully have access to the elevator shaft, in case I need it. Will $7,000 be enough?" How cheaply could you
Advertisement
THE
PREMIER
OPEN SOURCE EVENT
IN ASIA
OK

Links

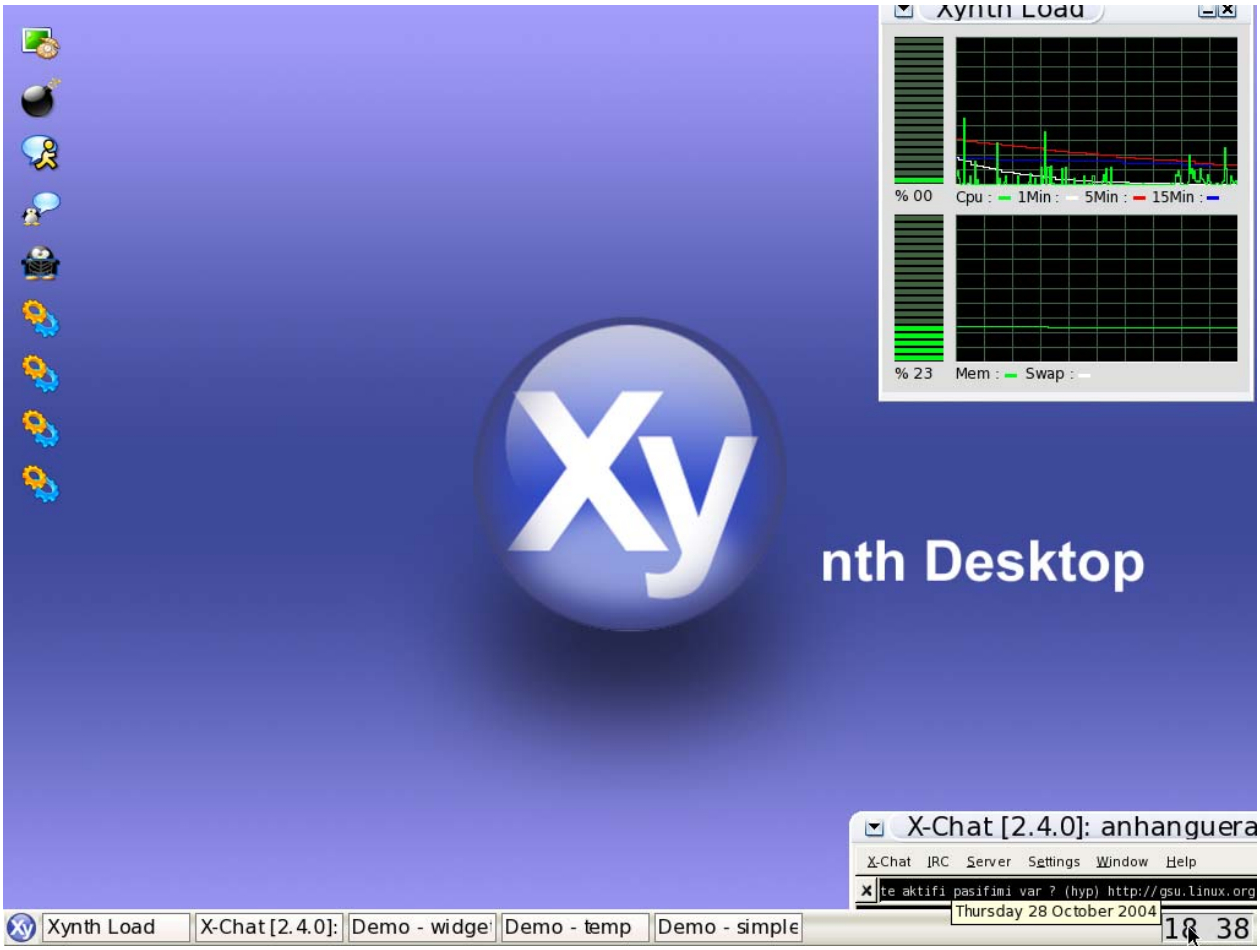

Demo - Child
Xynth Load
% 00
Cpu : 1Min : 5Min : 15Min :
% 35
Mem : Swap :
X-Chat [2.4.0]: distch @ sendak.freenode.net / #
X-Chat IRC Server Settings Window Help
<selam> e-posta sınırı yok 200 liste falan veriyor
<mesut> bw sınırlaması yok mu onda ?
<selam> mesut: var sanırım 6 GB idi
<mesut> e ne anladık o işten
<mesut> 6gb ile yansı mı olur
<selam> mesut: yansı diilde site olur
<selam> mesut: neyin yansısını kuracaksın birde o var
<selam> mesut: son olarak yansı için rsync falan lazım bakalım cron lazım
<selam> mesut: kullanma hakkı vereceklermi bakalım
<mesut> vermezler zaten de,
<selam> mesut: en iyisi co-location falan olur bence
<distch> smiley versenize screen shot alicam
<mesut> bizim yansı 3-5 saat geç yenilense de olur
<mesut> co-location da da bw sınırı var
<selam> :)
<mesut> bende bir dedicated var. onda da sınırlı
<selam> mesut: yenilemek için zaten rsync lazım diilmi?
<selam> ne ile alacaksın verileri ..
<mesut> benim yansı dediğim, iso olabilir, ya da kernel olabilir.
<mesut> onları da manuel olarak yenileyebilirsin.
<selam> kernel olmaz ...
distch
irc.linux.org #fazlamesai #r00t mascix
anhanguera
avaadore
bub
cL
DeliPenguen
distch
Findecano
GuNeY
JB007
mascix
mesut
Xynth Load
Xyntherminal
Xyntherminal
X-Chat [2.4.0]:
Demo - child
Demo - Child
14: 14
Load Demo


Xynth Load
Demo - mines
Xynthermin
Demo - mines
Xynth Load
Xyntherminal
18: 50
Mines Demo

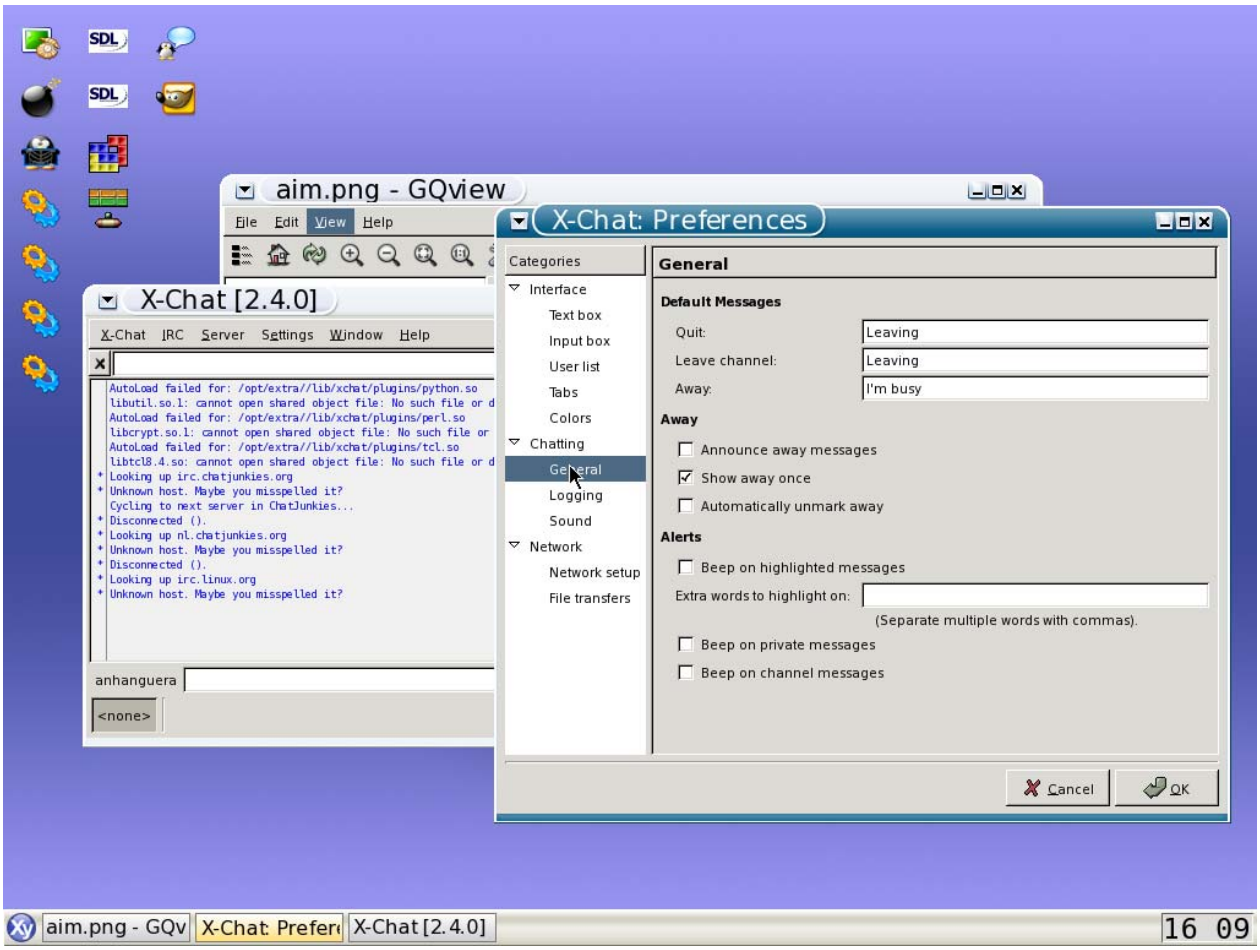


Mplayer port

SDL port – ltris plus lbreakout2



Demo Desktop



GTK port + xchat

7. API Referance

7.1 Types

```
typedef enum {
      SOC_DATA_NOTHING              = 0x0,
      SOC_DATA_NEW                  = 0x1,
      SOC_DATA_SHOW                 = 0x2,
      SOC_DATA_CLOSE                = 0x3,
      SOC_DATA_TITLE                = 0x4,
      SOC_DATA_EVENT                = 0x5,
      SOC_DATA_EXPOSE               = 0x6,
      SOC_DATA_DISPLAY              = 0x7,
      SOC_DATA_FORMDRAW             = 0x8,
      SOC_DATA_CONFIGURE            = 0x9,
      SOC_DATA_CONFIGURE_NOFORM     = 0xA,
      SOC_DATA_DESKTOP              = 0xB
} S_SOC_DATA;

typedef enum {
      WINDOW_NEW                    = 0x001,
      WINDOW_COOR                   = 0x002,
      WINDOW_DRAW                   = 0x004,
      WINDOW_SHOW                   = 0x008,
      WINDOW_TITLE                  = 0x010,
      WINDOW_RESIZEABLE             = 0x020,

      NO_FORM                       = 0x040,
      WINDOW_ROOT                   = 0x080,
      WINDOW_MAIN                   = 0x100,
      WINDOW_TEMP                   = 0x200,
      WINDOW_CHILD                  = 0x400,
      WINDOW_DESKTOP                = 0x800
} S_WINDOW;

typedef enum {
      SURFACE_REAL                  = 0x1,
      SURFACE_VIRTUAL               = 0x2
} S_SURFACE_MODE;

typedef enum {
      MOUSE_CURSOR_WAIT,
      MOUSE_CURSOR_CROSS,
      MOUSE_CURSOR_IBEAM,
      MOUSE_CURSOR_SIZEV,
      MOUSE_CURSOR_SIZEH,
      MOUSE_CURSOR_SIZES,
      MOUSE_CURSOR_SIZEB,
      MOUSE_CURSOR_SIZEA,
      MOUSE_CURSOR_ARROW,
      MOUSE_CURSOR_POINT,
      MOUSE_CURSOR_SPLITV,
      MOUSE_CURSOR_SPLITH,
      MOUSE_CURSOR_FORBID,
      MOUSE_CURSOR_UPARROW,
```

```
      MOUSE_CURSOR_MAX
} S_MOUSE_CURSOR;

typedef enum {
      KEYCODE_ALTF                  = 0x01,
      KEYCODE_CTRLF                 = 0x02,
      KEYCODE_CPLCKF                = 0x04,
      KEYCODE_NMLCKF                = 0x08,
      KEYCODE_SHIFTF                = 0x10,
      KEYCODE_ALTGRF                = 0x20
} S_KEYCODE_FLAG;

typedef enum {
      QUIT_EVENT                    = 0x00001,
      KEYBD_EVENT                   = 0x00002,
      KEYBD_RELEASED                = 0x00004,
      KEYBD_PRESSED                 = 0x00008,
      MOUSE_EVENT                   = 0x00010,
      MOUSE_OVER                    = 0x00020,
      MOUSE_RELEASED                = 0x00040,
      MOUSE_PRESSED                 = 0x00080,
      MOUSE_CLICKED                 = 0x00100,
      MOUSE_HINT                    = 0x00200,
      MOUSE_HINT2                   = 0x00400,
      EXPOSE_EVENT                  = 0x00800,
      EXPOSE_CHNGX                  = 0x01000,
      EXPOSE_CHNGY                  = 0x02000,
      EXPOSE_CHNGW                  = 0x04000,
      EXPOSE_CHNGH                  = 0X08000,
      DESKTOP_EVENT                 = 0x10000,
      EVENT_MASK  = QUIT_EVENT | EXPOSE_EVENT | KEYBD_EVENT | MOUSE_EVENT
} S_EVENT;

typedef enum {
      MOUSE                         = 0x1,
      KEYBD                         = 0x2
} S_HANDLER;

typedef struct s_list_node_s {
      void *next;
      void *element;
} s_list_node_t;

typedef struct s_list_s {
      int nb_elt;
      s_list_node_t *node;
} s_list_t;

typedef struct s_rect_s {
      int x;
      int y;
      int w;
      int h;
} s_rect_t;

typedef struct s_image_s {
      int w;
```

```
	int h;
	char *buf;
	char *mat;
	s_rect_t handler;
	unsigned int *rgba;
	s_list_t *layers;
} s_image_t;

typedef struct s_font_s {
	FT_Face face;
	FT_Library library;

	int yMin;
	int yMax;

	char *str;
	int size;
	s_image_t *img;
} s_font_t;

typedef struct s_pollfd_s {
	int fd;
	int (*in) (s_window_t *, int);
	int (*ierr) (s_window_t *, int);
} s_pollfd_t;

typedef struct s_mouse_s {
	int x;
	int y;
	int b;
	int rx;			/* to set wheel buttons */
	int ry;
	int clicks;		/* click count          */
	int buttons;		/* buttons bitwise ORed */
	struct timeval time;	/* event time           */
	int px;			/* prev. pressed coor.  */
	int py;
	int pb;			/* prev. pressed button  */
	int pbuttons;		/* prev. pressed buttons */
	struct timeval ctime;	/* prev. clicked time    */
	S_MOUSE_CURSOR cursor;
} s_mouse_t;

typedef struct s_keybd_s {
	int ascii;
	int scancode;
	int state[128];
	char button[40];
	S_KEYCODE_FLAG flag;
} s_keybd_t;

typedef struct s_expose_s {
	int change;
	s_rect_t rect;
} s_expose_t;

typedef struct s_desktop_client_s {
```

```
      int id;
      int pri;
      int title_l;
      char *title;
} s_desktop_client_t;

typedef struct s_dekstop_s {
      s_list_t *clients;
} s_desktop_t;

typedef struct s_event_s {
      int id;
      S_EVENT type;
      s_mouse_t *mouse;
      s_keybd_t *keybd;
      s_expose_t *expose;
      s_desktop_t *desktop;
} s_event_t;

typedef struct s_eventq {
      s_thread_t tid;
      s_list_t *queue;
        s_thread_cond_t *cond;
      s_thread_mutex_t *mut;
} s_eventq_t;

typedef struct s_handler_keybd_s {
      int flag;
      char *button;
      void (*p) (s_window_t *, s_event_t *, s_handler_t *);
      void (*r) (s_window_t *, s_event_t *, s_handler_t *);
} s_handler_keybd_t;

typedef struct s_handler_mouse_s {
      int x;
      int y;
      int w;
      int h;
      int button;
      void (*p) (s_window_t *, s_event_t *, s_handler_t *);
      /* button pressed  */
      void (*r) (s_window_t *, s_event_t *, s_handler_t *);
      /* button released */
      void (*c) (s_window_t *, s_event_t *, s_handler_t *);
      /* button clicked  */
      void (*o) (s_window_t *, s_event_t *, s_handler_t *);
      /* on over         */
      void (*ho) (s_window_t *, s_event_t *, s_handler_t *);
      /* on over && hint */
      /* on over, but mouse button is still pressed  */
      void (*hr) (s_window_t *, s_event_t *, s_handler_t *);
      /* button realesed && hint */
      /* mouse button released, but the prev. press was not on us */
      void (*oh) (s_window_t *, s_event_t *, s_handler_t *);
      /* button over && hint2 */
      /* not on over, but was on over */
      void (*hoh) (s_window_t *, s_event_t *, s_handler_t *);
```

```
	/* button over && hint && hint2 */
	/* not on over, but was on over. and button is still pressed */
	void (*rh) (s_window_t *, s_event_t *, s_handler_t *);
	/* button realesed && hint2 */
	/* mouse button released outside, but the prev. press was on us */
	/* wheel buttons has no realesed event                          */
	int overed;
} s_handler_mouse_t;

struct s_handler_s {
	char *name;
	S_HANDLER type;
	union {
		s_handler_mouse_t *mouse;
		s_handler_keybd_t *keybd;
	};
	void *user_data;
};

typedef struct s_childs_s {
	s_list_t *list;
	s_thread_mutex_t *mut;
} s_childs_t;

typedef struct s_surface_s {
	S_SURFACE_MODE mode;
	int bytesperpixel;
	int bitsperpixel;
	int colors;
	int blueoffset;
	int greenoffset;
	int redoffset;
	int bluelength;
	int greenlength;
	int redlength;

	int width;  /* These are the real sizes of our buffer */
	int height;

	char *vbuf; /* The buffer that holds clients window.  */
	s_rect_t buf; /* This is our bufs virtual part                  */
	s_rect_t win;

	char *linear_buf; /* mapped shared buffer                       */
	int linear_buf_width; /* See s_client_surface_linear()          */
	int linear_buf_height;
	unsigned long linear_mem_base;

	int *id;
	int shm_mid;
	unsigned char *matrix;
} s_surface_t;

typedef struct s_client_s {
	int id;
	int pri;
	int resizeable;
```

```
	int alwaysontop;
	char *title;
	S_EVENT event_copy2parent_mask;
	void (*atloop) (s_window_t *, S_SOC_DATA);
	void (*atevent) (s_window_t *, s_event_t *);
	void (*atcevent) (s_window_t *, s_window_t *, s_event_t *);
	void (*atexit) (s_window_t *);
	void *user_data;
} s_client_t;

struct s_window_s {
	int running;

	S_WINDOW type;
	s_list_t *pollfds;
	s_list_t *handlers;
	s_client_t *client;
	s_surface_t *surface;

	s_event_t *event;
	s_eventq_t *eventq;

	s_childs_t *childs;

	s_thread_t tid;
	s_window_t *parent;
};
```

7.2 Server internal

```
typedef enum {
	TOP_L = 0x0,
	TOP_1,
	TOP_2,
	TOP_3,
	TOP_4,
	TOP_5,
	TOP_R,
	LEFT,
	RIGHT,
	BTM_L,
	BTM,
	BTM_R,
	FORM_MAX
} THEME_FORM;

typedef enum {
	CLOSE = 0x0,
	MAXIMIZE,
	HIDE,
	MENU,
	BTNS_MAX
} THEME_BTN;

typedef enum {
	INACTIVE,
	ACTIVE,
```

```
	PRESSED
} THEME_STATE;

typedef struct s_theme_s {
	int title_full;
	int text_color[2];
	int text_v_off[2];
	s_image_t form[2][FORM_MAX];
	s_image_t button[3][BTNS_MAX];
	struct {
		int h;
		int w;
		int w_;
	} form_min;
} s_theme_t;

typedef enum {
	SURFACE_CLOSED,
	SURFACE_OPENED,
	SURFACE_REDRAW,
	SURFACE_REFRESH,
	SURFACE_CHANGED,
	SURFACE_PRIORITY
} S_SURFACE_CHNGF;

typedef struct s_clients_s {
	int soc;
	int pid; /* parent id */
	int resizeable;
	int alwaysontop;
	S_WINDOW type;
	s_rect_t buf;
	s_rect_t win;
	/* theme related */
	struct {
		char *str;
		int hy[2];
		int hh[2];
		s_image_t img[2];
	} title;
	s_rect_t form[FORM_MAX];
	s_rect_t button[BTNS_MAX];
} s_clients_t;

#if defined(SW_CURSOR)
typedef struct s_cursor_s {
	int x;
	int y;
	int xyid;
	s_image_t *img;
	s_image_t images[MOUSE_CURSOR_MAX];
} s_cursor_t;
#endif

typedef struct s_server_s {
	int mode;
	int id[S_CLIENTS_MAX];
```

```
      int pri[S_CLIENTS_MAX];
      s_clients_t client[S_CLIENTS_MAX];
#if defined(SW_CURSOR)
      /* SW Cursor */
      s_cursor_t cursor;
#endif
        s_theme_t theme;
        s_window_t *window;
      /* mouse */
      int mh;
} s_server_t;

s_server_t *server;

/* event.c */
void s_server_event_parse_keyboard (void);
int s_server_event_parse_mouse (void);
void s_server_event_changed (void);
void s_server_event_parse (S_EVENT event);

/* id.c */
int s_server_id_get (void);
int s_server_id_find (int socket);
void s_server_id_del (int id);

/* kbd.c */
void s_server_kbd_signal_handler (int v);
void s_server_kbd_atexit (void);
void s_server_kbd_state_clear (void);
void s_server_kbd_switch_handler (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_kbd_window_close_handler (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_kbd_server_quit_handler (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_kbd_set_attr (void);
void s_server_kbd_init (void);
void s_server_kbd_code2name (int scancode, char *name);
int s_server_kbd_code2ascii (char *name);
int s_server_kbd_name2code (char *name);
void s_server_kbd_keymap_load (void);
void s_server_kbd_uninit (void);
void s_server_kbd_update (void);

/* mouse.c */
void s_server_mouse_setcursor (S_MOUSE_CURSOR c);
void s_server_mouse_draw (void);
int s_server_mouse_update (void);
void s_server_mouse_uninit (void);
int s_server_mouse_in_f (s_window_t *window, int s);
void s_server_mouse_init (void);
#if defined(SW_CURSOR)
void s_server_cursor_init (void);
void s_server_cursor_image_set (int which, int c0, int c1, unsigned int *c);
void s_server_cursor_matrix_add (void);
void s_server_cursor_matrix_del (void);
void s_server_cursor_draw (void);
```

```
void s_server_cursor_select (S_MOUSE_CURSOR c);
void s_server_cursor_position (int x, int y);
#endif

/* priority.c */
void s_server_pri_set (S_SURFACE_CHNGF flag, ...);
void s_server_pri_set_ (S_SURFACE_CHNGF flag, int id, s_rect_t *c0, s_rect_t
*c1);
int s_server_id_pri (int id);
int s_server_pri_id (int pri);
void s_server_pri_del (int id);

/* server.c */
void s_server_init (int mode);
void s_server_mtrr_init (void);
void s_server_uninit (void);
void s_server_quit (s_window_t *window);
void s_server_goto_back (void);
void s_server_comefrom_back (void);

/* socket.c */
int s_server_socket_listen_new (int id);
int s_server_socket_listen_show (int id);
int s_server_socket_listen_title (int id);
int s_server_socket_listen_display (int id);
int s_server_socket_listen_configure (int id);
int s_server_socket_listen_desktop (int id);
int s_server_socket_listen_close (int socket);
int s_server_socket_listen_parse (int socket);
int s_server_socket_listen_accept (void);
int s_server_socket_request_event (int id);
int s_server_socket_request_close (int id);
int s_server_socket_request_expose (int id, s_rect_t *changed);
int s_server_socket_request_desktop (int id);
int s_server_socket_request (S_SOC_DATA req, int id, ...);
void s_server_socket_uninit (void);
void s_server_socket_init (void);

/* surface.c */
void s_server_surface_init (int mode);
void s_server_surface_uninit (void);
void s_server_surface_matrix_find (s_rect_t *coor, int *dm);
void s_server_surface_matrix_add (int id, s_rect_t *coor);
void s_server_surface_matrix_add_this (int id, s_rect_t *coor, s_rect_t
*mcoor, char *mat);
void s_server_surface_matrix_del (int id);
void s_server_surface_matrix_del_this (int id, s_rect_t *mcoor, char *mat);
void s_server_surface_clean (s_rect_t *coor);
void s_server_surface_background (s_rect_t *coor);
void s_server_surface_lock_real (void);
void s_server_surface_refresh (void);

/* window.c */
void s_server_window_new (int id);
void s_server_window_title (int id, char *title);
void s_server_putbox (s_window_t *window, int id, s_rect_t *coor, int x, int
y, s_image_t *img);
```

```
void s_server_putmat (s_window_t *window, int id, s_rect_t *coor, int x, int
y, s_image_t *img);
void s_server_window_form (int id, s_rect_t *_coor_);
#define s_server_window_matrix_add(a, b) s_server_window_matrix(a, a, b)
#define s_server_window_matrix_del(a, b) s_server_window_matrix(a,
S_MATRIX_DELETED, b)
void s_server_window_matrix (int id, int mi, s_rect_t *_coor_);
void s_server_window_calculate (int id);
int s_server_window_is_parent_of_any_temp (int pid);
int s_server_window_is_parent_temp (int pid, int cid);
void s_server_window_close_temps_all (void);
void s_server_window_close_temps (int id);
void s_server_window_close_id (int id);
void s_server_window_close (s_window_t *window);
void s_server_window_move_resize (int id, s_rect_t *new);
void s_server_window_maximize (s_window_t *window);

/* window_handler.c */
void s_server_window_btn_resize_oh (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_resize_u_o (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_resize_ur_o (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_resize_r_o (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_resize_dr_o (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_resize_d_o (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_resize_dl_o (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_resize_l_o (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_resize_ul_o (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_menu_p (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_menu_oh (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_menu_r (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_hide_p (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_hide_oh (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_hide_r (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_maximize_p (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_maximize_oh (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_maximize_r (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_close_p (s_window_t *window, s_event_t *event,
s_handler_t *handler);
```

```
void s_server_window_btn_close_oh (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_close_r (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_move (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_resize_up (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_resize_up_left (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_resize_up_right (s_window_t *window, s_event_t
*event, s_handler_t *handler);
void s_server_window_btn_resize_left (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_resize_right (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_resize_down (s_window_t *window, s_event_t *event,
s_handler_t *handler);
void s_server_window_btn_resize_down_left (s_window_t *window, s_event_t
*event, s_handler_t *handler);
void s_server_window_btn_resize_down_right (s_window_t *window, s_event_t
*event, s_handler_t *handler);
void s_server_window_handlers_del_mouse (void);
void s_server_window_handlers_add_mouse (int id);

/* window_move_resize.c */
void s_server_fillbox (int x, int y, int w, int h, int c);
void s_server_delbox (int x, int y, int w, int h);
void s_server_window_lines_draw (int fx, int fy, int fw, int fh);
void s_server_window_lines_clear (int fx, int fy, int fw, int fh);
void s_server_window_move (s_window_t *window);
void s_server_window_resize_up (s_window_t *window);
void s_server_window_resize_up_left (s_window_t *window);
void s_server_window_resize_left (s_window_t *window);
void s_server_window_resize_down_left (s_window_t *window);
void s_server_window_resize_down (s_window_t *window);
void s_server_window_resize_down_right (s_window_t *window);
void s_server_window_resize_right (s_window_t *window);
void s_server_window_resize_up_right (s_window_t *window);

/* theme.c */
void s_server_theme_init (void);
void s_server_theme_set (char *name);
void s_server_theme_uninit (void);
```

7.3 Client library

```
/* child.c */
int s_child_add (s_window_t *window, s_window_t *child);
int s_child_del (s_window_t *window, s_window_t *child);
int s_childs_init (s_window_t *window);
int s_childs_uninit (s_window_t *window);

/* client.c */
int s_client_init (s_window_t **window);
void s_client_uninit (s_window_t *window);
void s_client_exit (s_window_t *window);
```

```
void s_client_quit (s_window_t *window);
int s_client_child_find (s_window_t *parent, s_window_t *window, s_event_t
*event);
void * s_client_loop_event (void *arg);
void * s_client_loop (void *arg);
void * s_client_main (void *arg);
void s_client_atloop (s_window_t *window, void (*f) (s_window_t *,
S_SOC_DATA));
void s_client_atevent (s_window_t *window, void (*f) (s_window_t *, s_event_t
*));
void s_client_atcevent (s_window_t *window, void (*f) (s_window_t *,
s_window_t *, s_event_t *));
void s_client_atexit (s_window_t *window, void (*f) (s_window_t *));

/* driver.c */
void bpp_setpixel (s_surface_t *surface, int x, int y, int c);
void bpp_setpixel_o (s_surface_t *surface, int id, int x, int y, int c);
int bpp_getpixel (s_surface_t *surface, int x, int y);
int bpp_getpixel_o (s_surface_t *surface, int id,  int x, int y);
void bpp_hline (s_surface_t *surface, int x1, int y, int x2, int c);
void bpp_hline_o (s_surface_t *surface, int id,  int x1, int y, int x2, int
c);
void bpp_vline (s_surface_t *surface, int x, int y1, int y2, int c);
void bpp_vline_o (s_surface_t *surface, int id,  int x, int y1, int y2, int
c);
void bpp_fillbox (s_surface_t *surface, int x, int y, int w, int h, int c);
void bpp_fillbox_o (s_surface_t *surface, int id,  int x, int y, int w, int h,
int c);
void bpp_putbox (s_surface_t *surface, int x, int y, int w, int h, char *sp,
int bw);
void bpp_putbox_mask (s_surface_t *surface, int x, int y, int w, int h, char
*sp, char *sm, int bw);
void bpp_getbox (s_surface_t *surface, int x, int y, int w, int h, char *dp);
void bpp_putbox_o (s_surface_t *surface, int id,  int x, int y, int w, int h,
char *sp, int bw);
void bpp_putbox_mask_o (s_surface_t *surface, int id,  int x, int y, int w,
int h, char *sp, char *sm, int bw);
void bpp_getbox_o (s_surface_t *surface, int id,  int x, int y, int w, int h,
char *dp);

/* event.c */
int s_event_mouse_state (s_window_t *window, s_event_t *event,
s_handler_mouse_t *mouse, int over);
void s_event_parse_mouse (s_window_t *window, s_event_t *event);
int s_event_parse_keybd (s_window_t *window, s_event_t *event);
int s_event_parse_expos (s_window_t *window, s_event_t *event);
int s_event_changed (s_window_t *window);
int s_event_init (s_event_t **event);
int s_event_uninit (s_event_t *event);

/* eventq.c */
int s_eventq_init (s_window_t *window);
int s_eventq_uninit (s_window_t *window);
int s_eventq_add (s_window_t *window, s_event_t *event);
int s_eventq_get (s_window_t *window, s_event_t **event);
int s_eventq_wait (s_window_t *window, s_event_t **event);
```

```
/* font.c */
int s_font_init (s_font_t **font, char *name);
int s_font_uninit (s_font_t *font);
int s_font_set_size (s_font_t *font, int size);
int s_font_set_str (s_font_t *font, char *str);
int s_font_get_glyph (s_font_t *font);

/* grlib.c */
int s_rgbcolor (s_surface_t *surface, int r, int g, int b);
void s_colorrgb (s_surface_t *surface, int c, int *r, int *g, int *b);
void s_setpixel (s_surface_t *surface, int x, int y, int c);
void s_setpixelrgb (s_surface_t *surface, int x, int y, int r, int g, int b);
void s_setpixelrgba (s_surface_t *surface, int x, int y, int r, int g, int b,
int a);
int s_getpixel (s_surface_t *surface, int x, int y);
void s_getpixelrgb (s_surface_t *surface, int x, int y, int *r, int *g, int
*b);
void s_hline (s_surface_t *surface, int x1, int y, int x2, int c);
void s_vline (s_surface_t *surface, int x, int y1, int y2, int c);
void s_fillbox (s_surface_t *surface, int x, int y, int w, int h, int c);
void s_putbox (s_surface_t *surface, int x, int y, int w, int h, char *sp);
void s_putboxmask (s_surface_t *surface, int x, int y, int w, int h, char *sp,
char *sm);
void s_getbox (s_surface_t *surface, int x, int y, int w, int h, char *dp);
void s_putboxpart (s_surface_t *surface, int x, int y, int w, int h, int bw,
int bh, char *sp, int xo, int yo);
void s_putboxpartmask (s_surface_t *surface, int x, int y, int w, int h, int
bw, int bh, char *sp, char *sm, int xo, int yo);
void s_copybox (s_surface_t *surface, int x1, int y1, int w, int h, int x2,
int y2);
void s_getsurfacevirtual (s_surface_t *s, int w, int h, int bitspp, char
*vbuf);
int s_copybuffer (char *sb, int sbitspp, char **db, int dbitspp, int w, int
h);
void s_scalebox (s_surface_t *surface, int w1, int h1, void *_dp1, int w2, int
h2, void *_dp2);

/* handler.c */
int s_handler_init (s_handler_t **hndl);
int s_handler_uninit (s_handler_t *hndl);
int s_handler_find_cmp_f (void *p1, void *p2);
s_handler_t * s_handler_find (s_window_t *window, S_HANDLER type, char *name);
int s_handler_add (s_window_t *window, s_handler_t *hndl);
s_handler_t * s_handler_set (s_window_t *window, S_HANDLER type, char *name);
int s_handlers_init (s_window_t *window);
int s_handlers_uninit (s_window_t *window);

/* image.c */
int s_image_hex2int (char *str);
int s_image_get_mat (s_image_t *img);
int s_image_get_buf (s_surface_t *surface, s_image_t *img);
void s_image_get_handler (s_image_t *img);
int s_image_init (s_image_t **img);
void s_image_free_buf (s_image_t *img);
void s_image_free_mat (s_image_t *img);
void s_image_free_rgba (s_image_t *img);
int s_image_uninit (s_image_t *img);
```

```
int s_image_layer_init (s_image_t **lyr);
int s_image_layer_uninit (s_image_t *lyr);
int s_image_layers_init (s_image_t *img);
int s_image_layers_uninit (s_image_t *img);

/* image_png.c */
void s_image_png (char *file, s_image_t *img);

/* image_xpm.c */
void s_image_xpm (char *file, s_image_t *img);

/* list.c */
int s_list_init (s_list_t *li);
int s_list_eol (s_list_t *li, int i);
void * s_list_get (s_list_t *li, int pos);
int s_list_remove (s_list_t *li, int pos);
int s_list_add (s_list_t *li, void *el, int pos);
void * s_list_find (s_list_t *list, void *node, int (*cmp_func) (void *, void
*));
int s_list_get_pos (s_list_t *list, void *node);

/* pollfd.c */
int s_pollfd_init (s_pollfd_t **pfd);
int s_pollfd_uninit (s_pollfd_t *pfd);
int s_pollfd_find_cmp_f (void *p1, void *p2);
s_pollfd_t * s_pollfd_find (s_window_t *window, int fd);
int s_pollfd_add (s_window_t *window, s_pollfd_t *pfd);
int s_pollfds_init (s_window_t *window);
int s_pollfds_uninit (s_window_t *window);

/* socket.c */
int s_socket_recv_ (int s, void *read_buf, int total_size);
int s_socket_send_ (int s, void *write_buf, int total_size);
#define s_socket_recv(a, b, c) if (s_socket_recv_(a, b, c) != c) {return -1; }
#define s_socket_send(a, b, c) if (s_socket_send_(a, b, c) != c) {return -1; }
int s_socket_request_new (s_window_t *window, int socket);
int s_socket_request_title (s_window_t *window, int socket);
int s_socket_request_display (s_window_t *window, int socket);
int s_socket_request_configure (s_window_t *window, int socket, S_WINDOW
form);
int s_socket_request_desktop (s_window_t *window, int socket, int id);
int s_socket_request (s_window_t *window, S_SOC_DATA req, ...);
int s_socket_listen_event (s_window_t *window, int socket);
int s_socket_listen_expose (s_window_t *window, int socket);
int s_socket_listen_desktop (s_window_t *window, int socket);
S_SOC_DATA s_socket_listen_parse (s_window_t *window, int socket);
S_SOC_DATA s_socket_listen (s_window_t *window, int timeout);
void s_socket_uninit (s_window_t *window);
int s_socket_in_f (s_window_t *window, int s);
int s_socket_ierr_f (s_window_t *window, int s);
int s_socket_init (s_window_t *window);

/* surface.c */
int s_surface_init (s_window_t *window);
void s_surface_create (s_window_t *window);
void s_surface_shm_attach (s_window_t *window);
void s_surface_linear (s_window_t *window);
```

```
void s_surface_uninit (s_window_t *window);
void s_surface_changed (s_window_t *window, s_rect_t *changed);

/* thread.c */
int s_thread_mutex_init (s_thread_mutex_t **mut);
int s_thread_mutex_destroy (s_thread_mutex_t *mut);
int s_thread_mutex_lock (s_thread_mutex_t *mut);
int s_thread_mutex_unlock (s_thread_mutex_t *mut);
int s_thread_cond_init (s_thread_cond_t **cond);
int s_thread_cond_destroy (s_thread_cond_t *cond);
int s_thread_cond_signal (s_thread_cond_t *cond);
int s_thread_cond_wait (s_thread_cond_t *cond, s_thread_mutex_t *mut);
void * s_thread_run (void *farg);
s_thread_t s_thread_create (void * (*f) (void *), void *farg, int detach);
int s_thread_cancel (s_thread_t tid);
int s_thread_join (s_thread_t tid, void **ret);
void s_thread_exit (void *ret);

/* window.c */
void s_window_set_title (s_window_t *window, char *fmt, ...);
void s_window_form_draw (s_window_t *window);
void s_window_show (s_window_t *window);
void s_window_set_coor (s_window_t *window, int form, int x, int y, int w, int
h);
void s_window_set_resizeable (s_window_t *window, int resizeable);
void s_window_set_alwaysontop (s_window_t *window, int alwaysontop);
void s_window_new (s_window_t *window, S_WINDOW type, s_window_t *parent);
```

7.4 Widget library

```
class SRect {
public:
      int rectX;
      int rectY;
      int rectW;
      int rectH;

      void rectSet (int x, int y, int w, int h);
      SRect (int x = 0, int y = 0, int w = 0, int h = 0);
      SRect (const SRect &copy);
      ~SRect (void);
      void operator = (const SRect &copy);
};

class SSize {
public:
      int sizeW;
      int sizeH;

      void sizeSet (int w, int h);
      SSize (int w = 0, int h = 0);
      ~SSize (void);

      int operator < (const SSize &copy);
      int operator > (const SSize &copy);
};
```

```
class SObject {
public:
      typedef enum {
            Auto       = 0x0,
            Fixed      = 0x1,
            Minimum        = 0x2,
            FreeResize = 0x4
      } ResizeMode;
      typedef enum {
            Center
      } Alignment;

      SObject *objectChild;
      SObject *objectParent;

      s_window_t *objectWindow;

      int objectAlignment;
      int objectResizeMode;

      SSize objectSizeMin;
      SSize objectSizeMax;
      SRect objectRectBuffer;
      SRect objectRectContents;

      int objectRGBColor (int r, int g, int b);
      void objectFillBox (int x, int y, int w, int h, int color);

      void objectSetSizeMin (int w, int h);
      void objectSetSizeMax (int w, int h);
      void objectSetRectBuffer (int x, int y, int w, int h);
      void objectSetRectContents (int x, int y, int w, int h);

      void objectDelFromParent (void);
      void objectSetParent (SObject *parent);

      void draw (SObject *object = NULL);
      void expose (int x, int y, int w, int h, SObject *object = NULL);
      void geometry (int x, int y, int w, int h, SObject *object = NULL);

      virtual void objectDraw (void) = 0;
      virtual void objectGeometry (int x, int y, int w, int h) = 0;

      SObject (SObject *parent = NULL, int alignment = SObject::Center, int
resizemode = SObject::Auto);
      virtual ~SObject (void);
};

class SHandler {
public:
      SObject *handlerObj;
      s_handler_t *handlerHndl;

      void handlerAdd (void);
      void handlerDel (void);
      int handlerFind (void);
```

```
      SHandler (SObject *obj, S_HANDLER type);
      ~SHandler (void);
};

class SWindow : public SObject {
public:
      void windowSetTitle (char *fmt, ...);
      void windowFormDraw (void);
      void windowSetCoor (int x, int y, int w, int h, int with_form =
NO_FORM);
      void windowSetResizeable (int resizeable);
      void windowShow (void);
      void windowMain (void);

      void windowAtExit (s_window_t *window);
      void windowAtEvent (s_window_t *window, s_event_t *event);

      virtual void objectDraw (void);
      virtual void objectGeometry (int x, int y, int w, int h);

      SWindow (S_WINDOW type = WINDOW_MAIN, s_window_t *parent = NULL);
      ~SWindow (void);
};

class SLayoutCell : public SObject{
public:
      SObject *layoutCellObject;

      virtual void objectDraw (void);
      virtual void objectGeometry (int x, int y, int w, int h);

      SLayoutCell (SObject *object = NULL);
      ~SLayoutCell (void);
};

class SLayout : public SObject {
public:
      int layoutRows;
      int layoutCols;
      int layoutColSpacing;
      int layoutRowSpacing;
      SLayoutCell **layoutCells;

      void layoutSetColSpacing (int spacing);
      void layoutSetRowSpacing (int spacing);
      void layoutInsertObject (SObject *object, int row, int col);

      virtual void objectDraw (void);
      virtual void objectGeometry (int x, int y, int w, int h);

      SLayout (SObject *parent, int rows = 1, int cols = 1);
      ~SLayout (void);
};

class SFrame : public SObject {
public:
      typedef enum {
```

```
            NoFrame           = 0x00,
            Box               = 0x01,
            Panel             = 0x02,
            WinPanel          = 0x03,
            HLine             = 0x04,
            VLine             = 0x05,
            StyledPanel       = 0x06,
            PopupPanel        = 0x07,
            MenuBarPanel      = 0x08,
            ToolBarPanel      = 0x09,
            LineEditPanel     = 0x0a,
            TabWidgetPanel    = 0x0b,
            GroupBoxPanel     = 0x0c,
            MShape            = 0x0f
      } frameShape;

      typedef enum {
            Plain             = 0x10,
            Raised            = 0x20,
            Sunken            = 0x30,
            MShadow           = 0xf0
      } frameShadow;

      int frameStyle;
      int frameLineWidth;
      int frameMidLineWidth;

      void frameRectContents (void);
      void frameSetStyle (int style);
      void frameSetLineWidth (int width);
      void frameSetMidLineWidth (int width);

      virtual void objectDraw (void);
      virtual void objectGeometry (int x, int y, int w, int h);

      SFrame (SObject *parent, int style = (SFrame::NoFrame | SFrame::Plain));
      ~SFrame (void);
};


class SButton : public SFrame {
public:
      SHandler *buttonHandler;

      virtual void buttonPressed (int button);
      virtual void buttonReleased (int button);

      static void button_cb_o (s_window_t *window, s_event_t *event,
s_handler_t *hndl);
      static void button_cb_p (s_window_t *window, s_event_t *event,
s_handler_t *hndl);
      static void button_cb_c (s_window_t *window, s_event_t *event,
s_handler_t *hndl);
      static void button_cb_r (s_window_t *window, s_event_t *event,
s_handler_t *hndl);
      static void button_cb_hr (s_window_t *window, s_event_t *event,
s_handler_t *hndl);
```

```
      static void button_cb_ho (s_window_t *window, s_event_t *event,
s_handler_t *hndl);
      static void button_cb_oh (s_window_t *window, s_event_t *event,
s_handler_t *hndl);
      static void button_cb_hoh (s_window_t *window, s_event_t *event,
s_handler_t *hndl);
      static void button_cb_rh (s_window_t *window, s_event_t *event,
s_handler_t *hndl);

      virtual void objectDraw (void);
      virtual void objectGeometry (int x, int y, int w, int h);

      SButton (SObject *parent, int style = SFrame::NoFrame);
      ~SButton (void);
};
```

8. Resources

http://www.xynth.org

http://gsu.linux.org.tr/~distch/projects

Alper Akçan

distchx@yahoo.com

Ali Çağlar Oral

caglaroral@yahoo.com