Porting X Windows System to Operating System Compliant with Portable Operating System Interface

Andrey V. Zhadchenko[1], Kirill A. Mamrosenko[2], Alexander M. Giatsintov[3]
Center of Visualization and Satellite Information Technologies
Scientific Research Institute of System Analysis
Moscow, Russia

***Abstract*—Now-a-days graphical interface is very important for any operating system, even the embedded ones. Adopting existing solutions will be much easier than developing your own. Moreover, a lot of software may be reused in this case. This article is devoted to X Window System adaptation for Portable Operating System Interface (POSIX) compliant real-time operating system Baget. Many encountered problems come from the tight connection between X and Linux, therefore it is expected to encounter these issues during usage of X on non-Linux systems. Discussed problems include, but not limited to the absence of dlopen, irregular file paths, specific device drivers. Instructions and recommendations to solve these issues are given. A comparison between XFree86 and Xorg implementations of X is discussed. Although synthetic tests show Xorg performance superiority, XFree86 consumes fewer system resources and is easier to port.**

Keywords*—X Window System; X11; X.Org Server; Xorg; XFree86; Portable Operating System Interface (POSIX); graphics; Realtime Operating System (RTOS)

I. Introduction

Although some real-time operating systems have not included graphic output for a long time, it's absence or obsolescence may be a huge disadvantage nowadays. Many target devices, such as onboard computers in cars or aircrafts, started to utilize displays for showing gauges and providing additional information to the user. In order to support this features, RTOS must provide some graphic API for applications. Options are limited to adopting existing interfaces or writing your own. The most straightforward solution to this problem - development of your own interface will allow you to minimize API overheads, which may be very crucial for embedded devices. Major disadvantage of this approach will be low portability of used applications. Also a lot of existing applications must be ported manually in order to use them. On the other side, adopting common graphic solutions will result in decreasing development cycle time by acquiring updates and tests from community and standardization of API usage for software. However, some problems will be encountered during porting. Moreover, it may be hard to further improve this software in terms of performance due to adopted architecture.

Host real-time operating system is called Baget [1]. It provides different standards for developers: POSIX 1003.1 [2], ARINC 653 [3], C++11. All programs must be statically compiled in order to run. Baget focuses on system reliability: many self-test facilities are implemented, including, but not limited to tasks and OS automatic restart, stack overflow checks, object validation. Baget can run various tasks simultaneously with separated process contexts [4]. Current graphical subsystem is server-client X windows system implementation named XFree86 with version 4.8.0. Although XFree86 [5] supports up to the X11R6.6 protocol version, which is barely enough to run modern applications, absence of many important extensions, for example, Xrender [6], implies heavy limits upon software.

Nowadays existing free software solutions in display servers for operating systems are limited to two options: X Windows System and Wayland [7]. However, there is a big ideological difference between them. X started it's history a long time ago in the '80s and was developed as an all-around solution. The protocol supports a lot of operations, including window management and draw operations. For example, it is possible to implement an application menu via X11 calls. Although X API is rich, nowadays a lot of software does not use it directly due to its complexity. Over time a lot of frameworks emerged upon X, for example, GTK. As a lot of frameworks implemented its API and draw capabilities, drawing directly through X become obsolete. Moreover, some server capabilities were moved into kernel, for example, User Mode Setting became Kernel Mode Setting [8], [9]. To replace X, which has a lot of excessive functionality and complexity, Wayland was created. Its protocol is completely unaware of window content and does not support any API that allows drawing your application. The scope of Wayland is window interaction, buffer management, etc. To fill window content you need to use another API or framework, for example, Qt, EGL, or even X. Although Wayland is considered to be a better display server protocol (mostly because it is created in a way to be a window system and nothing more), it's implementations are highly dependent on relatively new Linux kernel API named Direct Rendering Infrastructure [10]. Adopting these interfaces will surely become a difficult task. As for X, although its modern implementation Xorg also can use new Linux kernel subsystems [11], it still supports less complicated backends. Moreover, not only all modern frameworks are ready to be used with X, old software is compatible with modern server implementations. Considering all these arguments it was decided to adopt a modern window manager called X.org [12]. It was forked from the XFree86 project in 2004. Nowadays X.org, which is widely used in the most popular Linux distributives, is de facto standard X11 implementation with frequent updates. In comparison to it, XFree86, which is the current Baget OS windowing system, was released in 2008.

The main objective is to adapt the newest X.org version to Baget to be able to use the whole X ecosystem, including

modern codebase and libraries. Although X uses a client-server architecture, most porting problems come from the server-side. It is divided into two parts: Driver Independent X (DIX) that represents server logic, request processing, drawing routines, extensions, etc., and Driver Dependent X (DDX), which is responsible for handling hardware, input and output devices and holds most of OS-specific code. To be able to use Xorg with the operating system, or rather API, that is not supported by developers, matching DDX must be implemented. Codebase already has some implementations for Linux, Wayland, Windows, etc. However, they are dependant on API that generic POSIX-compliant OS might not have. Of course, writing new DDX from scratch would be the best way in terms of compatibility, but time and resources spent on this will be huge. On the other hand, patching the present code would reduce the initial workload in exchange for testing. As a base of our solution, we decided to use the existing DDX for Linux systems. Obtained during this research solutions and recommendations may be used to port modern X to another operating system with similar to Baget properties, because a lot of encountered problems will arise during adaptation to any non-Linux based OS.

In addition to that, issues regarding XFree86 adaptation will be considered, as well as performance and system resource usage in comparison to modern X.org.

To sum up, there are two main questions:

1) How to port Xorg to Baget OS and what problems are expected to arise with other POSIX-compliant systems?
2) Is it worth to port Xorg instead of xFree86?

This paper is organized as follows:

- A review of issues that arise when porting X server to new operating system with constrained functionality and proposed solutions.
- Performance comparison between XFree86 implementation and modern Xorg implementation.
- Conclusions and future work.

II. Related Work

Although X Window System was developed in 1984, there is a shortage of available articles and proceedings that discuss various aspects of the protocol and implementations. Discussion is primarily done on specialized forums and mailing lists. Most articles were done in the 90's and discussed earlier versions of X protocol. Even less information can be found on porting X Window System implementations to operating systems, other than Linux.

Recent publications on X Window System discuss general programming issues that arise when using X11 API [13]. Performance is critical for the windowing systems, so a lot of work is done on using GPU acceleration for drawing operations to reduce CPU workload. In [14] authors try to reduce cpu overhead by implementing resource-sharing protocols. Some attempts on porting X server implementation to embedded system result in a practically complete overhaul of the server internal systems [15].

III. Portability Problems and Solutions

A. Static Compilation

In the Baget OS, processes are pre-defined and starting along with the system itself. Due to this, executable code cannot be loaded dynamically. This requires all of the process code to be packed into a single object file during system compilation.

Although machine independent part of the X.org server can be statically built with ease and have a corresponding option, some problems come from the DDX component. It is necessary to change makefile scripts and rename entry point function to be able to call it later.

Another problem comes from X architecture. Nowadays, two sets of libraries are used to build X infrastructure. Ones are required by the server-side, ones by client applications. Some libraries as libXau are used in both. But others may have functions with the same names and different prototypes. Because of that, all the libraries cannot be combined into a single static library so client and server parts have to be maintained separately.

B. Dlopen Abscence

As mentioned above, executables cannot be loaded dynamically. Due to this, the dlopen function family is not supported. In order to be flexible Xorg uses a module system, that heavily relies on this API because it is used during module loading and devices initialization. Every loaded module must define a global variable called <modulename>ModuleData, which contains version, setup and teardown functions. This object would be searched during the startup of the X server to fill ModuleDesc structure and register module.

However, these steps can be done with a minor alteration of source code even without dlopen. Functions inside the loader component must be stubbed. In every loadable module, which will be used later, special function must be implemented. It will allocate required function tables and ModuleDesc structure, perform registering and call module setup. This function must be called during DDX initialization instead of normal routine.

Despite the simplicity, this approach restricts available drivers. On the other hand, considering Baget usage as an embedded system, the target platform and list of devices are known during compilation. Due to this, it is not reasonable to use dynamic drivers.

C. Device Drivers

Device drivers in Xorg are part of modules. A module can contains one or multiple drivers at once. Module responsibilities are registration of itself and it's drivers. While module initialization and driver initialization are called once, device can be turned on or turned off multiple times. Input devices are monitored by server, so when Xorg detects an event on device file descriptor (i.e. poll reports that descriptor state is changed or SIGIO raises), driver is called to proceed input. It is expected that driver would generate an event that will be further processed by X server.

An example of action sequence for mouse driver *xorg-xf86-input-mouse*:

1) Mouse movement generates I/O on mouse file descriptor.
2) Xorg server detects I/O and calls driver function *MouseReadInput*.
3) Driver reads data from desctiptor and proceed it accoring to protocol and other settings.
4) Depending on data, driver calls xf86PostMotionEvent or xf86PostButtonEvent.
5) Xorg constructs and put events (for example, single MotionEvent) in server queue.

Modern DDX part of X.org uses a special interface to handle input drivers called udev [16], which is a Linux kernel device manager. However, old mouse and keyboards modules is still supported. Due to this, previous drivers *xorg-xf86-input-keyboard* and *xorg-xf86-input-mouse* were used as base due to it's simple requirements. Keyboard driver relies on input from stdin. Mouse driver expects a device file from which mouse packets data can be read. On top of that OS must provide either select, poll or SIGIO API to watch file descriptors for both drivers.

For output drivers, implementing standard screen functions is enough to use graphics with software rendering done by X. However, this driver heavily depends on the hardware platform and therefore will not be discussed in this article.

D. Xkbcomp Utility

When the key is pressed keyboard driver delivers scancode to the X server. In order to handle different types of keyboards and language sets, there must be a mapping between scancodes and actual symbols. To compile keymaps for current keyboard X server uses utility named xkbcomp [17]. During Xorg's initialization, the main process forks, redirects streams and executes this program. Xkbcomp is separated from the X server because it relies on client-side libraries, thus cannot be compiled into the server due to conflicting functions. But Baget does not support fork and exec API because processes are pre-defined.

It is possible to solve this problem in several ways.

Firstly, it can be achieved with rewriting xkbcomp utility with X server libraries instead of client ones. However, some parts of the code will be heavily altered and it will be hard to update it with new versions later.

Secondly, we can imitate fork by creating another process that manages xkbcomp during system startup. Communication and synchronization between different sides will be done with two FIFO's instead of streams redirection. There are some disadvantages of this approach. Some kind of protocol must be implemented to pass arguments to xkbcomp. Moreover, FIFO is visible to other processes that can write or read these files. On top of that additional process and FIFO files create unnecessary complexity and reduces clarity for users.

Although X server and X client cannot be simply linked into one executable, some tricks with GNU utilities ld and objcopy may help to get around this problem.

- Compile xkbcomp without running linker with gcc flag *-c*.
- Create relocatable output file with ld option *-r* from xkbcomp and X client libraries.
- Using objcopy flag *-G enter_func* create same file with all symbols hidden except *enter_func*.

After these steps, it is possible to link xkbcomp into the Xorg server process because all conflicting symbols are not visible after objcopy. After that, we can run xkbcomp from entry function *enter_func* directly or create a special thread for that purpose. Also, it is possible to get rid of FIFO by using unnamed pipes or memory since it is the same process now. Unlike the first solution, the required changes to source code are tiny. The main disadvantage of this approach is almost doubling X server executable size from 23MB to 40MB due to copying a lot of binary code from X client object files.

E. Local Sockets

X.org normally creates a UNIX socket to handle all incoming local requests. POSIX implementation of *socket* in Baget only supports AF_INET domain, therefore creation of UNIX domain socket is impossible. However, all requests can proceed if the client and server will consider using localhost instead of a socket file. This is possible to achieve by changing communication callbacks for local connections in header files.

F. User ID and Group ID

The majority of embedded devices are not supposed to be directly (e.g. terminal access) used. Because of this, Baget does not have a user identifier (UID) or group identifier (GID). But X.org have a lot of checks related to UID. The simplest solution is either stubbing UID functions to return 0, which is equal to root UID in UNIX, or ignoring all checks.

G. Server and Applications Resources

In order to use X.org without Filesystem Hierarchy Standard [18] a user must provide path to fonts, log files, configuration files, and other resources. This can be done during compilation.

Moreover, some applications are using libXt API [19] to draw interfaces. This may cause unclear behavior because libXt will look for app-defaults folder to find resources related to running program. In case of its absence, no warning will be generated, but the colors and shapes of drawn objects will be incorrect. It is possible to solve this problem by modifying libXt source code or setting environmental variable XAPPLRESDIR.

H. Launching Statically Compiled X.org and Applications

Some tweaks must be done to successfully launch X.org after static compilation.

First of all, to correctly call the renamed entry function of X.org, *argc* and *argv* should be carefully generated and provided. Arguments must include display number at least, for example, *:0*. Before running any X application environmental variable *DISPLAY* needs to be set accordingly.

Nowadays X.org will not display cursor and use mouse driver until desktop environment, which is mostly not required

for an embedded system, is loaded. However, while launched in retro mode (with argument *-retro*), X.org will provide a working cursor and mouse.

I. XFree86 Issues

Since X.org was forked from XFree86, all issues discussed before is the same for both windowing managers with two exceptions. Firstly, dlopen is not mandatory for XFree86 - modules can be statically linked by default. Secondly, xkb-comp is not required to obtain a keyboard configuration. As for device drivers, its API has not changed much since then.

IV. Performance Comparison Results between XFree86 and Xorg and Evaluation

A tool named x11perf [20] was used to measure performance difference. x11perf tests server by calling X protocol commands multiple times and recording the number of operations per second. Both xFree86 and Xorg were installed on a test stand with the following characteristics: single-core MIPS 400mhz CPU and 4GB 400mhz RAM.

x11perf results include over 350 different tests. However, it may be combined into groups. For example, “Copy 10x10 from window to window”, “Copy 100x100 from window to window”, “Copy 100x100 from window to pixmap” will belong to the copy group.

Fig. 1 and 2 represent performance ratio of Xorg to XFree86 in different test groups. All values are normalized to the XFree86 result taken as 100%. For example, “Copy 500x500 from pixmap to pixmap” result is 194 and 271 operations per second for xFree86 and Xorg, respectively. That means the dot will be at 140% since Xorg result is 140% of xFree86. Each dot on the plot represents performance change for a particular test. Test names are omitted. The dotted line represents the mean value of a group. For this plots' data, please refer to Table 1 and Table 2.

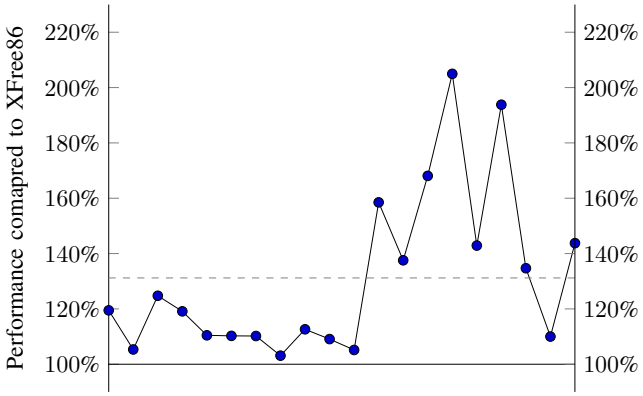

Fig. 1. x11perf Char Group Tests

Due to X.org's increase in complexity compared with XFree86, some operations may fall behind, for example, X protocol NoOperation call declined from 1070000 to 543000 per second. Despite that, a lot of test suits show better results, such as groups of char and copy operations, +30% and +10% respectively. Overall performance increase is 46%. Although most of it comes from tests utilizing windows operations, such as creating, moving, mapping and resizing.

Table I. x11perf Char Group Tests Data

Test name	Performance compared to XFree86 (%)
Char in 80-char line (6x13)	119,4779
Char in 70-char line (8x13)	105,336
Char in 60-char line (9x15)	124,7475
Char16 in 40-char line (k14)	119,1176
Char16 in 23-char line (k24)	110,4551
Char16 in 7/14/7 line (k14, k24)	110,2564
Char in 80-char image line (6x13)	110,2041
Char in 70-char image line (8x13)	103,1008
Char in 60-char image line (9x15)	112,6263
Char16 in 40-char image line (k14)	109,0909
Char16 in 23-char image line (k24)	105,157
Char in 80-char aa line (Charter 10)	158,5086
Char in 30-char aa line (Charter 24)	137,5635
Char in 80-char aa line (Courier 12)	168,0851
Char in 80-char a line (Charter 10)	204,9587
Char in 30-char a line (Charter 24)	142,8894
Char in 80-char a line (Courier 12)	193,8053
Char in 80-char rgb line (Charter 10)	134,7044
Char in 30-char rgb line (Charter 24)	110
Char in 80-char rgb line (Courier 12)	143,7653

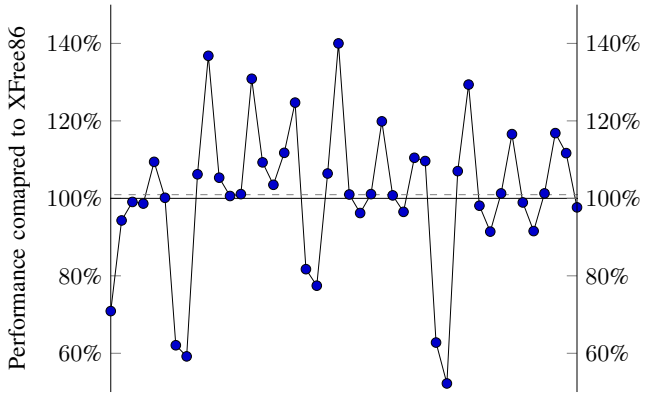

Fig. 2. x11perf Rectangle Group Tests

As for memory usage, while X.org consumes 3 MB RAM in idle state, while XFree86 uses only 1 MB.

V. Conclusions

X Windows System (specifically X.org v1.20.3) was successfully ported to Baget operating system. Porting X to non-POSIX systems may be unreasonable due to the usage of signals, file descriptors, polls, and other specific API. As for POSIX-compliant systems, one must implement device drivers and solve platform-specific issues. Covered in this article topics and methods can be used to adapt X to various systems with similar to Baget properties.

Two X implementations, Xorg and XFree86, were compared. Although XFree86 is older than the newest X.org release, it is easier to port. Another advantage is lower system requirements, for example, memory usage, which can be decisive for embedded systems. Xorg's superiority in performance may be spoiled in case of heavy usage of simple operations and program preference to draw small objects rather than big. As an example, a lot of either old or simple software such as standard utilities, calculators, notepads, etc. directly use such

TABLE II. X11 PERF RECTANGLE GROUP TESTS DATA

Test name	Performance compared to XFree86 (%)
1x1 rectangle	58,9443
10x10 rectangle	93,9502
100x100 rectangle	99,0637
500x500 rectangle	98,6301
1x1 stippled rectangle (8x8 stipple)	109,4218
10x10 stippled rectangle (8x8 stipple)	100,1506
100x100 stippled rectangle (8x8 stipple)	38,9021
500x500 stippled rectangle (8x8 stipple)	31,085
1x1 opaque stippled rectangle (8x8 stipple)	106,2124
10x10 opaque stippled rectangle (8x8 stipple)	136,7975
100x100 opaque stippled rectangle (8x8 stipple)	105,3333
500x500 opaque stippled rectangle (8x8 stipple)	100,6036
1x1 tiled rectangle (4x4 tile)	101,0929
10x10 tiled rectangle (4x4 tile)	130,8725
100x100 tiled rectangle (4x4 tile)	109,2896
500x500 tiled rectangle (4x4 tile)	103,5
1x1 stippled rectangle (17x15 stipple)	111,7521
10x10 stippled rectangle (17x15 stipple)	124,7163
100x100 stippled rectangle (17x15 stipple)	77,6224
500x500 stippled rectangle (17x15 stipple)	70,8904
1x1 opaque stippled rectangle (17x15 stipple)	106,4128
10x10 opaque stippled rectangle (17x15 stipple)	140
100x100 opaque stippled rectangle (17x15 stipple)	100,9967
500x500 opaque stippled rectangle (17x15 stipple)	96,063
1x1 tiled rectangle (17x15 tile)	101,0929
10x10 tiled rectangle (17x15 tile)	119,8758
100x100 tiled rectangle (17x15 tile)	100,7634
500x500 tiled rectangle (17x15 tile)	96,3855
1x1 stippled rectangle (161x145 stipple)	110,4701
10x10 stippled rectangle (161x145 stipple)	109,6311
100x100 stippled rectangle (161x145 stipple)	40,6897
500x500 stippled rectangle (161x145 stipple)	8,4746
1x1 opaque stippled rectangle (161x145 stipple)	107,0281
10x10 opaque stippled rectangle (161x145 stipple)	129,3706
100x100 opaque stippled rectangle (161x145 stipple)	98,0663
500x500 opaque stippled rectangle (161x145 stipple)	90,604
1x1 tiled rectangle (161x145 tile)	101,2774
10x10 tiled rectangle (161x145 tile)	116,5803
100x100 tiled rectangle (161x145 tile)	98,927
500x500 tiled rectangle (161x145 tile)	90,7692
1x1 tiled rectangle (216x208 tile)	101,2774
10x10 tiled rectangle (216x208 tile)	116,8421
100x100 tiled rectangle (216x208 tile)	111,6773
500x500 tiled rectangle (216x208 tile)	97,619

X API. On the other hand, modern frameworks, such as Qt and GTK, prefer to internally draw windows and use XPutImage call, which favors Xorg. Speaking in terms of functionality, XFree86 seriously falls behind due to a lack of modern X extensions.

There are some options for the future development of this research. First of all, since we adopted existing DDX for Xorg, we need to test graphic output very carefully and check if our changes do not break anything. Although we ran several programs such as x11perf and more complicated examples as browser and office suite, this validation process may be deceiving. We need to use existing or invent a new method to ensure correctness. Secondly, there are a lot of frameworks built upon X, such as Qt and GTK. Supporting them will be a benefit in terms of versatility because nowadays a lot of programs do not directly use X API but these frameworks. As was mentioned in the introduction - some inner parts or ideas of X evolved into the new interfaces. Adopting one of them, named KMS, may be very useful. Currently, it is the main backend for X server instead of fbdev that we used. Moreover, new technologies, such as Wayland, use it too. KMS API will allow us to ensure compatibility of Baget graphic subsystem with the upcoming changes and ease future development. Also only display features of X Window System are used, accelleration via 2D or 3D accellerators is not used. In the future, we plan to adapt DRI framework in the X server for Baget operating system in order to use GPU accellerated 3D applications.

ACKNOWLEDGMENTS

Publication is made as part of national assignment for SRISA RAS (fundamental scientific research 47 GP) on the topic No.0065-2019-0001 (AAAA-A19-119011790077-1).

REFERENCES

[1] A. N. Godunov and V. A. Soldatov, "Baget real-time operating system family (features, comparison, and future development)," *Programming and Computer Software*, vol. 40, no. 5, pp. 259–264, Sep. 2014. [Online]. Available: https://doi.org/10.1134/S036176881405003X

[2] The IEEE and The Open Group, *The Open Group Base Specifications Issue 6 – IEEE Std 1003.1, 2004 Edition*. New York, NY, USA: IEEE, 2004. [Online]. Available: http://www.opengroup.org/onlinepubs/009695399/

[3] A. C.M., S. Nair, and M. G.H., "Arinc 653 api and its application – an insight into avionics system case study," *Defence Science Journal*, vol. 63, no. 2, p. 223–229, Mar 2013.

[4] A. N. Godunov and V. A. Soldatov, "Experience creating a compact real-time operating system," *Software Engineering*, vol. 10, no. 2, pp. 51–58, Sep 2019.

[5] Bill Ball, *The New XFree86 Book*. Course Technology, 2001.

[6] K. Packard, "A New Rendering Model for X," in *Proceedings of the FREENIX track, 2000 USENIX annual technical conference*. San Diego, California, USA: USENIX Association, Jun. 2000.

[7] Kristian Høgsberg, "The wayland protocol," https://wayland.freedesktop.org/docs/html/.

[8] Konstantin V. Pugin, Kirill A. Mamrosenko, and Alexander M. Giatsintov, "Visualization of graphic information in general-purpose operating systems," *Radioelektronika, Nanosistemy, Informacionnye Tehnologii*, vol. 11, no. 2, pp. 217–224, 2019.

[9] I. Efremov, V. Reshetnikov, and K. Mamrosenko, "Methods of developing graphics subsystem drivers," vol. 33, pp. 425–429, 08 2018.

[10] Brian Paul, "Introduction to the direct rendering infrastructure," http://dri.sourceforge.net/doc/DRIintro.html.

[11] J. Madieu, *Linux Device Drivers Development: Develop Customized Drivers for Embedded Linux*. Packt Publishing, 2017.

[12] Alan Coopersmith, Matt Dew, and Bart Massey, *The X New Developer's Guide*. X.Org Foundation, Oct. 2012.

[13] R. J. Maloney, *Low Level X Window Programming*. Springer International Publishing, 2017. [Online]. Available: http://link.springer.com/10.1007/978-3-319-74250-2

[14] S. Kato, K. Lakshmanan, Y. Ishikawa, and R. Rajkumar, "Resource sharing in gpu-accelerated windowing systems," in *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, Apr 2011, p. 191–200. [Online]. Available: http://ieeexplore.ieee.org/document/5767151/

[15] D. Jeong, Kamalneet, N. Kim, and S. Lim, "Gpu-based x server on top of egl and openvg," in *2009 Digest of Technical Papers International Conference on Consumer Electronics*. IEEE, Jan 2009, p. 1–2. [Online]. Available: http://ieeexplore.ieee.org/document/5012187/

[16] G. Kroah-Hartman, "udev – A Userspace Implementation of devfs," in *Proceedings of the Linux Symposium*, Ottawa, Ontario, Canada, Jul. 2003, pp. 249–257. [Online]. Available: https://www.kernel.org/doc/mirror/ols2003.pdf

[17] Erik Fortune, "The X Keyboard Extension: Protocol Specification," https://www.x.org/releases/current/doc/kbproto/xkbproto.html, May 1996.

[18] Rusty Russell, Daniel Quinlan, and Christopher Yeoh, "Filesystem hierarchy standard," https://refspecs.linuxfoundation.org/FHS_2.3/fhs-2.3.pdf.

[19] Joel McCormack, Paul Asente, and Ralph R. Swick, "X toolkit intrinsics - c language interface," https://www.x.org/releases/X11R7.7/doc/libXt/intrinsics.html.

[20] Joel McCormack, Phil Karlton, Susan Angebrandt, Chris Kent, Keith Packard, and Graeme Gill, "X11perf - x11 server performance test program," https://www.x.org/releases/X11R7.7/doc/man/man1/x11perf.1.xhtml.