“i3Blocks”

Project

Submitted to Department of Computer Science and Engineering,
University Institute of Engineering and Technology,
M. D. University, Rohtak

Submitted towards partial fulfillment for award of degree
Of
BACHELOR OF TECHNOLOGY (Computer Science and Engineering)

Submitted By:
Prateek Yadav
Roll No. 19508
(B.Tech (Computer Science and Engineering))

Submitted To:
Mrs. Amita Dhankhar,
Assistant Professor,
Department of Computer Science and Engineering

UNIVERSITY INSTITUTE OF ENGINEERING AND TECHNOLOGY,
MAHARSHI DAYANAND UNIVERSITY, ROHTAK – 124001
NOVEMBER, 2019

ACKNOWLEDGEMENT

The completion of any work depends upon the cooperation, coordination and combined effect of several resources of knowledge, energy and time. Therefore, I approach this important matter of acknowledgement through these lines trying my best to give full credits where it deserves.

I would like to express my special thanks of gratitude to my professor **Mrs. Amita Dhankhar** as well as our Director, **Dr. Rahul Rishi**, who gave me the golden opportunity to do this wonderful project on **i3Blocks**, which also helped me in doing a lot of Research and I came to know about so many new things I am really thankful to them.
Secondly i would also like to thank my parents and friends who helped me a lot in finalizing this project within the limited time frame.

I shall remain indebt to the faculty members of University Institute of Engineering and Technology, Maharshi Dayanand University, Rohtak for their cooperation, kindness and general help extended to me during the completion of this work.

Prateek Yadav

Roll No. 19508

B. Tech (Computer Science and Engineering)

University Institute of Engineering and Technology

MDU, Rohtak - 124001

ABSTRACT

i3Blocks project is directed towards use of ‘i3blocks’ in such a way that the user can easily change or navigate through spotify playlists. Initially this porject wasn’t started for the purpose of submision but figured its good enough to be a seperate project.
i3blocks is a piece of software under which the created script works for displaying and controlling the music currently being playedd via SPOTIFY(music streaming service).
- Why this script?
- i3blocks are used in very controlled or specific environment, like it works primarily with LINUX hence user must have linux installed to make it work. Also, user must have i3WM for default use of i3blocks as its main top resources and information bar i.e: mostly like TASK BAR on WINDOWS but being far more customisable and open source.
Current testing and daily driver setup/workplace looks like this with I3BLOCKS being at top with SPOTIFY script.

TABLE OF CONTENT

INTRODUCTION

Prerequisites and Requirements.

- Application of i3Blocks are mainly focused to high to advanced linux users who emphasis on efficiency and snappiness of their workflow. User is required to possess extensive knowledge for some programming languages like C and other low level languages.

- Basic requirements are as follows :
    - Linux distribution
    - Specific desktop environment
    - Display manager
    - Window manager
    - i3Blocks binary

- System requirements :
    - Any x86_64 compatible machine
    - Minimum of 512MiB RAM
    - Minimum of 800MiB disk space for bare minimalist install.

WHY LINUX?

- Open source nature : Linux is completely an open source project. You can have a look at the source code of a Linux OS

- Secure :I t is just the way Linux works that makes it a secure operating system. Overall, the process of package management, the concept of repositories, and a couple more features makes it possible for Linux to be more secure than Windows.When you have Windows installed, you need to download/purchase an antivirus program to keep your computer safe from hackers and malware. However, Linux does not require the use of such Anti-Virus programs. Of course, a couple of software tools still exist to help you keep your system free from threats, but it is often unnecessary when you have a Linux system.

- Runs on older hardware : With Linux, you could even utilize one of your oldest computer systems to achieve a task. However, that does not mean that every Linux distribution would work with **256 MB** of RAM coupled with an outdated processor. However, there are options which you can install on such low-end systems as well. So, being an efficient OS, Linux distributions could be fitted to a range of systems (low-end or high-end). In contrast, Windows operating system has a higher hardware requirement.Overall, even if you compare a high-end Linux system and a high-end Windows-powered system, the Linux distribution would take the edge. Well, that is the reason most of the servers across the world prefer to run on Linux than on a Windows hosting environment.

- Perfect for Programmers : Linux supports almost all of the major programming languages (*Python, C/C++, Java, Perl, Ruby, etc*.). Moreover, it offers a vast range of applications useful for programming purposes. The Linux terminal is superior to use over Window's command line for developers. You would find many libraries developed natively for Linux. Also, a lot of programmers point out that the package manager on Linux helps them get things done easily. Interestingly, the ability of bash scripting is also one of the most compelling reasons why programmers prefer using Linux OS. Linux also brings in native support for SSH, which would help you manage your servers quickly. You could include things like apt-get commands which further makes Linux one of the most popular choices of the programmers.
- Software Updates : Microsoft pushes a software update when it receives a set of problems or if something major needs to be fixed. On the other hand, you would observe a software update to address a little problem. So, with Linux, you will notice more updates to fix the problems you might be facing. You will not only encounter a larger number of software updates, but you will also observe much faster software updates.
- Customisation : If you like tweaking your system's looks, Linux is just perfect for you. Apart from installing themes, you have tons of beautiful icon themes. In addition to that, you can use Conky to display system information on the desktop in the coolest way possible. Needless to say that you can do a lot around Wallpapers in Linux.
- Variety of Distributions : There are no flavors of Windows. Yes, you may have different plans & packages which differ in licensing terms, the period of activation, packaged features, and price. In contrast, you will find tons of Linux distribution catered for a different set of needs. So, you can choose to install any of the available Linux distros according to your requirements.
- Better community :You do not need to hire an expert to solve a problem you are facing on your Linux system. You just need to search for a similar thread on the web for a solution or post a thread to let others solve the problem. Within minutes of posting a thread on any of the Linux forums, you may expect a reply along with a detailed solution which would finally help resolve your problem at no cost! There are a lot of active Linux users who are always ready to respond to a relevant thread one might have created. The number of community users active on such forums is more than the number of active members on any Windows-focused forum.
- Reliability : Using Linux, you will not have to worry about re-installing it just to experience a faster and a smoother system. Linux helps your system run smooth for a longer period.
- Interoperability : Windows plays REALLY well with Windows. Linux plays well with everyone. I've never met a system I couldn't connect Linux to. That includes OS X, Windows, various Linux distributions, OS/2, Playstations... the list goes on and on.Without the help of third-party software, Windows isn't nearly as interoperable. And we haven't even touched on formats. With OpenOffice, you can open/save in nearly any format (regardless of release date).

LINUX AND LINUX DISTRIBUTIONS

LINUX

Linux is a family of open source Unix-like operating systems based on the Linux kernel, an operating system kernel first released on September 17, 1991, by Linus Torvalds. Linux is typically packaged in a Linux distribution.

Distributions include the Linux kernel and supporting system software and libraries, many of which are provided by the GNU Project. Many Linux distributions use the word "Linux" in their name, but the Free Software Foundation uses the name *GNU/Linux* to emphasize the importance of GNU software, causing some controversy.

Popular Linux distributions include Debian, Fedora, and Ubuntu. Commercial distributions include Red Hat Enterprise Linux and SUSE Linux Enterprise Server. Desktop Linux distributions include a windowing system such as X11 or Wayland, and a desktop environment such as GNOME or KDE Plasma. Distributions intended for servers may omit graphics altogether, or include a solution stack such as LAMP. Because Linux is freely redistributable, anyone may create a distribution for any purpose.

Linux was originally developed for personal computers based on the Intel x86 architecture, but has since been ported to more platforms than any other operating system. Linux is the leading operating system on servers and other big iron systems such as mainframe computers, and the only OS used on TOP500 supercomputers (since November 2017, having gradually eliminated all competitors). It is used by around 2.3 percent of desktop computers. The Chromebook, which runs the Linux kernel-based Chrome OS, dominates the US K–12 education market and represents nearly 20 percent of sub-$300 notebook sales in the US.

Linux also runs on embedded systems, i.e. devices whose operating system is typically built into the firmware and is highly tailored to the system. This includes routers, automation controls, televisions, digital video recorders, video game consoles, and smartwatches. Many smartphones and tablet computers run Android and other Linux derivatives. Because of the dominance of Android on smartphones, Linux has the largest installed base of all general-purpose operating systems.

Linux is one of the most prominent examples of free and open-source software collaboration. The source code may be used, modified and distributed—commercially or non-commercially—by anyone under the terms of its respective licenses, such as the GNU General Public License.

Early days : The Unix operating system was conceived and implemented in 1969, at AT&T's Bell Laboratories in the United States by Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna. First released in 1971, Unix was written entirely in assembly language, as was common practice at the time. In 1973 in a key, pioneering approach, it was rewritten in the C programming

language by Dennis Ritchie (with the exception of some hardware and I/O routines). The availability of a high-level language implementation of Unix made its porting to different computer platforms easier.

Due to an earlier antitrust case forbidding it from entering the computer business, AT&T was required to license the operating system's source code to anyone who asked. As a result, Unix grew quickly and became widely adopted by academic institutions and businesses. In 1984, AT&T divested itself of Bell Labs; freed of the legal obligation requiring free licensing, Bell Labs began selling Unix as a proprietary product, where users were not legally allowed to modify Unix. The GNU Project, started in 1983 by Richard Stallman, had the goal of creating a "complete Unix-compatible software system" composed entirely of free software. Work began in 1984. Later, in 1985, Stallman started the Free Software Foundation and wrote the GNU General Public License (GNU GPL) in 1989. By the early 1990s, many of the programs required in an operating system (such as libraries, compilers, text editors, a Unix shell, and a windowing system) were completed, although low-level elements such as device drivers, daemons, and the kernel, called GNU/Hurd, were stalled and incomplete.

Linus Torvalds has stated that if the GNU kernel had been available at the time (1991), he would not have decided to write his own. Although not released until 1992, due to legal complications, development of 386BSD, from which NetBSD, OpenBSD and FreeBSD descended, predated that of Linux. Torvalds has also stated that if 386BSD had been available at the time, he probably would not have created Linux.

MINIX was created by Andrew S. Tanenbaum, a computer science professor, and released in 1987 as a minimal Unix-like operating system targeted at students and others who wanted to learn the operating system principles. Although the complete source code of MINIX was freely available, the licensing terms prevented it from being free software until the licensing changed in April 2000.

Creation : In 1991, while attending the University of Helsinki, Torvalds became curious about operating systems. Frustrated by the licensing of MINIX, which at the time limited it to educational use only, he began to work on his own operating system kernel, which eventually became the Linux kernel.

Torvalds began the development of the Linux kernel on MINIX and applications written for MINIX were also used on Linux. Later, Linux matured and further Linux kernel development took place on Linux systems. GNU applications also replaced all MINIX components, because it was advantageous to use the freely available code from the GNU Project with the fledgling operating system; code licensed under the GNU GPL can be reused in other computer programs as long as they also are released under the same or a compatible license. Torvalds initiated a switch from his original license, which prohibited commercial redistribution, to the GNU GPL. Developers worked to integrate GNU components with the Linux kernel, making a fully functional and free operating system.

Current Development : Greg Kroah-Hartman is the lead maintainer for the Linux kernel and guides its development. William John Sullivan is the executive director of the Free Software Foundation, which in turn supports the GNU components. Finally, individuals and corporations develop third-party non-GNU components. These third-party components comprise a vast body of work and may include both kernel modules and user applications and libraries.

Linux vendors and communities combine and distribute the kernel, GNU components, and non-GNU components, with additional package management software in the form of Linux distributions.

LINUX DISTRIBUTIONS

A Linux distribution (often abbreviated as distro) is an operating system made from a software collection, which is based upon the Linux kernel and, often, a package management system. Linux users usually obtain their operating system by downloading one of the Linux distributions, which are available for a wide variety of systems ranging from embedded devices (for example, OpenWrt) and personal computers (for example, Linux Mint) to powerful supercomputers (for example, Rocks Cluster Distribution).

A typical Linux distribution comprises a Linux kernel, GNU tools and libraries, additional software, documentation, a window system (the most common being the X Window System), a window manager, and a desktop environment. Most of the included software is free and open-source software made available both as compiled binaries and in source code form, allowing modifications to the original software. Usually, Linux distributions optionally include some proprietary software that may not be available in source code form, such as binary blobs required for some device drivers. A Linux distribution may also be described as a particular assortment of application and utility software (various GNU tools and libraries, for example), packaged together with the Linux kernel in such a way that its capabilities meet the needs of many users. The software is usually adapted to the distribution and then packaged into software packages by the distribution's maintainers. The software packages are available online in so-called repositories, which are storage locations usually distributed around the world. Beside glue components, such as the distribution installers (for example, Debian-Installer and Anaconda) or the package management systems, there are only very few packages that are originally written from the ground up by the maintainers of a Linux distribution.

Almost six hundred Linux distributions exist, with close to five hundred out of those in active development. Because of the huge availability of software, distributions have taken a wide variety of forms, including those suitable for use on desktops, servers, laptops, netbooks, mobile phones and tablets, as well as minimal environments typically for use in embedded systems. There are commercially backed distributions, such as Fedora (Red Hat), openSUSE (SUSE) and Ubuntu (Canonical Ltd.), and entirely community-driven distributions, such as Debian, Slackware, Gentoo and **Arch Linux(choice of distribution for project)**. Most distributions come ready to use and pre-compiled for a specific instruction set, while some distributions (such as Gentoo) are distributed mostly in source code form and compiled locally during installation.

Package management under linux distributions : Distributions are normally segmented into *packages*. Each package contains a specific application or service. Examples of packages are a library for handling the PNG image format, a collection of fonts or a web browser.

The package is typically provided as compiled code, with installation and removal of packages handled by a package management system (PMS) rather than a simple file archiver. Each package intended for such a PMS contains meta-information such as a package description, version, and "dependencies". The package management system can evaluate this meta-information to allow package searches, to perform an automatic upgrade to a newer version, to check that all dependencies of a package are fulfilled, and/or to fulfill them automatically.

Although Linux distributions typically contain much more software than proprietary operating systems, it is normal for local administrators to also install software not included in the distribution. An example would be a newer version of a software application than that supplied with a distribution, or an alternative to that chosen by the distribution (for example, KDE Plasma Workspaces rather than GNOME or vice versa for the user interface layer). If the additional software is distributed in source-only form, this approach requires local compilation. However, if additional software is locally added, the "state" of the local system may fall out of synchronization with the state of the package manager's database. If so, the local administrator will be required to take additional measures to ensure the entire system is kept up to date. The package manager may no longer be able to do so automatically.

Most distributions install packages, including the kernel and other core operating system components, in a predetermined configuration. Few now require or even permit configuration adjustments at first install time. This makes installation less daunting, particularly for new users, but is not always acceptable. For specific requirements, much software must be carefully configured to be useful, to work correctly with other software, or to be secure, and local administrators are often obliged to spend time reviewing and reconfiguring assorted software.

Some distributions go to considerable lengths to specifically adjust and customize most or all of the software included in the distribution. Not all do so. Some distributions provide configuration tools to assist in this process.

By replacing *everything* provided in a distribution, an administrator may reach a "distribution-less" state: everything was retrieved, compiled, configured, and installed locally. It is possible to build such a system from scratch, avoiding a distribution altogether. One needs a way to generate the first binaries until the system is *self-hosting*. This can be done via compilation on another system capable of building binaries for the intended target (possibly by cross-compilation).

ARCH Linux

Arch Linux is a Linux distribution for computers based on x86-64 architectures. The Arch Linux repositories contain both libre, and nonfree software, and the default Arch Linux kernel contains nonfree proprietary blobs, hence the distribution is not endorsed by the GNU project.

Arch Linux adheres to five principles: simplicity, modernity, pragmatism, user centrality and versatility. In general, the principles maintain minimal distribution-specific changes, minimal breakage with updates, pragmatic over ideological design choices, user-friendliness, and minimal bloat.

A package manager written specifically for Arch Linux, pacman, is used to install, remove and update software packages. Arch Linux uses a rolling release model, meaning there are no "major release" of completely new versions of the system; a regular system update is all that is needed to obtain the latest Arch software; the installation images released by the Arch team are simply up-to-date snapshots of the main system components.

Arch Linux has comprehensive documentation, which consists of a community wiki known as the ArchWiki.

Early days : Inspired by CRUX, another minimalist distribution, Judd Vinet started the Arch Linux project in March 2002. The name was chosen because Vinet liked the word's meaning of "the principal," as in "arch-enemy".

Originally only for 32-bit x86 CPUs, the first x86_64 installation ISO was released in April 2006.

Vinet led Arch Linux until 1 October 2007, when he stepped down due to lack of time, transferring control of the project to Aaron Griffin.

The end of i686 support was announced in January 2017, with the February 2017 ISO being the last one including i686 and making the architecture unsupported in November 2017. Since then, the community derivative Arch Linux 32 can be used for i686 hardware.

Package manager and Repository Security : To facilitate regular package changes, pacman (The Arch Package Manager, its name a contraction of "package manager") was developed by Judd Vinet to provide Arch with its own package manager able to track dependencies. It is written in C.

All packages are managed using the pacman package manager. Pacman handles package installation, upgrades, removal, and downgrades, and features automatic dependency resolution. The packages for Arch Linux are obtained from the Arch Linux package tree and are compiled for the x86-64 architecture. It uses binary packages in the tar.xz format or tar.zst (for zstd compression), with .pkg placed before this to indicate that it is a pacman package (giving .pkg.tar.xz or .pkg.tar.zst).

Until pacman version 4.0.0 Arch Linux's package manager lacked support for signed packages. Packages and metadata were not verified for authenticity by pacman during the download-install

process. Without package authentication checking, tampered-with or malicious repository mirrors can compromise the integrity of a system. Pacman 4 allowed verification of the package database and

packages, but it was disabled by default. In November 2011 package signing became mandatory for new package builds, and as of 21 March 2012 every official package is signed.

In June 2012, package signing verification became official and is now enabled by default in the installation process.

AUR (Arch User Repository) : In addition to the repositories, the Arch User Repository (AUR) provides user-made PKGBUILD scripts for packages not included in the repositories. These PKGBUILD scripts simplify building from source by explicitly listing and checking for dependencies and configuring the install to match the Arch architecture. Arch User Repository helper programs can further streamline the downloading of PKGBUILD scripts and associated building process. However, this comes at the cost of executing PKGBUILDs not validated by a trusted person; as a result, Arch developers have stated that the utilities for automatic finding, downloading and executing of PKGBUILDs will never be included in the official repositories.

Users can create packages compatible with pacman using the Arch Build System and custom PKGBUILD scripts. This functionality has helped support the Arch User Repository, which consists of user contributed packages to supplement the official repositories.

The Arch User Repository provides the community with packages that are not included in the repositories. Reasons include:

- Licensing issues: software that cannot be redistributed, but is free to use, can be included in the Arch User Repository since all that is hosted by the Arch Linux website is a shell script that downloads the actual software from elsewhere. Examples include proprietary freeware such as Google Earth and RealPlayer.
- Modified official packages: the Arch User Repository also contains many variations on the official packaging as well as beta versions of software that is contained within the repositories as stable releases.
- Rarity of the software: rarely used programs have not been added to the official repositories (yet).
- Betas or "nightly" versions of software which are very new and thus unstable. Examples include the "firefox-nightly" package, which gives new daily builds of the Firefox web browser.

PKGBUILDs for any software can be contributed by ordinary users and any PKGBUILD that is not confined to the Arch User Repository for policy reasons can be voted into the community repositories.

WHY ARCH?

- GUI Installers (easy to install nowadays if compared to early installations)

Arch Linux used to be very painstaking to install. In fact, the single process of installing Arch Linux used to drive curious potential users away but that isn't the case any longer and it is with thanks to GUI installers such as Anarchy and Zen installer). It is 2019 and Arch Linux is a lot easier to install than it was before; that negates a reason why you shouldn't install it.

- Wide range of desktop environment to chose from

  The Desktop Environment of any distro is what helps you interact with your system from the time you boot the OS to when you shut it down. It is compatible with more D.E than might want to even try out yourself and its aesthetics can be 100% customized to suit your taste.

- Arch Wiki

  The Arch Wiki is a vast library of documentation about [almost] any task you can complete on Arch Linux, its derivatives, and sometimes even other distros! It contains an overview of Arch Linux and a description of what to expect from it, FAQ and facts about it, an installation guide, post-installation tutorials.

- Pacman

  Pacman is a command-line tool and the default package installer in Arch Linux and it is less wordy than the package managers of other distros e.g. APT in Ubuntu.

- AUR

  The Arch User Repository (AUR) is a collection of applications and tools that are installable on Arch Linux but not yet available in the official Arch repository. It is maintained by the Arch Linux user community as a means to conveniently find and install those apps without hunting them down and compiling them from source.

- Originality

  Unlike other popular distros like Ubuntu which is based on Debian, Arch Linux was built from scratch independent of any other Linux distribution. Granted, Ubuntu has gone on to birth several other distros but so also has Arch Linux e.g. Manjaro Linux. So in as much as Arch Linux is open source and its developers are free to incorporate ideas from anywhere they choose, Arch Linux will always be pacesetters in the community.

- Ease of Use, Flexibility, and Customization

  Arch Linux may seem *stiff* from the outside but it is a completely flexible distro. First, it lets you decide which modules to use in your OS when installing it and it has the Wiki to guide you. Also, it doesn't bombard you with several [often] unnecessary applications but ships with a minimal list of default software.

DESKTOP ENVIRONMENT

In computing, a desktop environment (DE) is an implementation of the desktop metaphor made of a bundle of programs running on top of a computer operating system, which share a common graphical user interface (GUI), sometimes described as a graphical shell. The desktop environment was seen mostly on personal computers until the rise of mobile computing. Desktop GUIs help the user to easily access and edit files, while they usually do not provide access to all of the features found in the underlying operating system. Instead, the traditional command-line interface (CLI) is still used when full control over the operating system is required.

A desktop environment typically consists of icons, windows, toolbars, folders, wallpapers and desktop widgets (see Elements of graphical user interfaces and WIMP). A GUI might also provide drag and drop functionality and other features that make the desktop metaphor more complete. A desktop environment aims to be an intuitive way for the user to interact with the computer using concepts which are similar to those used when interacting with the physical world, such as buttons and windows.

While the term *desktop environment* originally described a style of user interfaces following the desktop metaphor, it has also come to describe the programs that realize the metaphor itself. This usage has been popularized by projects such as the Common Desktop Environment, K Desktop Environment, and GNOME.

Implemention : On a system that offers a desktop environment, a window manager in conjunction with applications written using a widget toolkit are generally responsible for most of what the user sees. The window manager supports the user interactions with the environment, while the toolkit provides developers a software library for applications with a unified look and behavior.

A windowing system of some sort generally interfaces directly with the underlying operating system and libraries. This provides support for graphical hardware, pointing devices, and keyboards. The window manager generally runs on top of this windowing system. While the windowing system may provide some window management functionality, this functionality is still considered to be part of the window manager, which simply happens to have been provided by the windowing system.

Applications that are created with a particular window manager in mind usually make use of a windowing toolkit, generally provided with the operating system or window manager. A windowing toolkit gives applications access to widgets that allow the user to interact graphically with the application in a consistent way.

Early days : The first desktop environment was created by Xerox and was sold with the Xerox Alto in the 1970s. The Alto was generally considered by Xerox to be a personal office computer; it failed in the marketplace because of poor marketing and a very high price tag.[*dubious – discuss*] With the Lisa, Apple introduced a desktop environment on an affordable personal computer, which also failed in the market.

The desktop metaphor was popularized on commercial personal computers by the original Macintosh from Apple in 1984, and was popularized further by Windows from Microsoft since the 1990s. As of 2014, the most popular desktop environments are descendants of these earlier environments, including the Aero environment used in Windows Vista and Windows 7, and the Aqua environment used in macOS. When compared with the X-based desktop environments available for Unix-like operating systems such as Linux and FreeBSD, the proprietary desktop environments included with Windows and macOS have relatively fixed layouts and static features, with highly integrated "seamless" designs that aim to provide mostly consistent customer experiences across installations.

Microsoft Windows dominates in marketshare among personal computers with a desktop environment. Computers using Unix-like operating systems such as macOS, Chrome OS, Linux, BSD or Solaris are much less common; however, as of 2015 there is a growing market for low-cost Linux PCs using the X Window System or Wayland with a broad choice of desktop environments. Among the more popular of these are Google's Chromebooks and Chromeboxes, Intel's NUC, the Raspberry Pi, etc.[*citation needed*]

Examples : The most common desktop environment on personal computers is Microsoft Windows' built-in interface. It was titled Luna in Windows XP, Aero in Windows Vista and Windows 7, Metro in Windows 8 and 8.1, and Fluent in Windows 10. Also common is Aqua, included with Apple's macOS.

Mainstream desktop environments for Unix-like operating systems use the X Window System, and include KDE, GNOME, Xfce, and LXDE, any of which may be selected by users and are not tied exclusively to the operating system in use.

A number of other desktop environments also exist, including (but not limited to) CDE, EDE, GEM, IRIX Interactive Desktop, Sun's Java Desktop System, Jesktop, Mezzo, Project Looking Glass, ROX Desktop, UDE, Xito, XFast. Moreover, there exists FVWM-Crystal, which consists of a powerful configuration for the FVWM window manager, a theme and further adds, altogether forming a "construction kit" for building up a desktop environment.

X window managers that are meant to be usable stand-alone — without another desktop environment — also include elements reminiscent of those found in typical desktop environments, most prominently Enlightenment.[*citation needed*] Other examples include OpenBox, Fluxbox, WindowLab, Fvwm, as well as Window Maker and AfterStep, which both feature the NeXTSTEP GUI look and feel. However newer versions of some operating systems make self configure.

The Amiga approach to desktop environment was noteworthy: the original Workbench desktop environment in AmigaOS evolved through time to originate an entire family of descendants and alternative desktop solutions. Some of those descendants are the Scalos, the Ambient desktop of MorphOS, and the Wanderer desktop of the AROS open source OS. WindowLab also contains features reminiscent of the Amiga UI. Third-party Directory Opus software, which was originally just a navigational file manager program, evolved to become a complete Amiga desktop replacement called Directory Opus Magellan.

There is the Workplace Shell that runs on IBM OS/2 or eComStation.

DISPLAY MANAGER

A display manager, or login manager, is typically a graphical user interface that is displayed at the end of the boot process in place of the default shell. There are various implementations of display managers, just as there are various types of window managers and desktop environments. There is usually a certain amount of customization and themeability available with each one.

Examples :

- Console

- CDM — Ultra-minimalistic, yet full-featured login manager written in Bash.
- Console TDM — Extension for *xinit* written in pure Bash.
- nodm — Minimalistic display manager for automatic logins.
- Ly — Experimental ncurses display manager.
- tbsm — A pure bash session or application launcher. Supports X and Wayland sessions.

- Graphical

- GDM — GNOME display manager.
- LightDM — Cross-desktop display manager, can use various front-ends written in any toolkit.
- LXDM — LXDE display manager. Can be used independent of the LXDE desktop environment.
- SDDM — QML-based display manager and successor to KDM; recommended for Plasma and LXQt.
- XDM — X display manager with support for XDMCP, host chooser.

In the X Window System, X display manager is a graphical login manager which starts a login session on an X server from the same or another computer.

A login screen shown by the KDM display manager.

A display manager presents the user with a login screen. A session starts when a user successfully enters a valid combination of username and password.

When the display manager runs on the user's computer, it starts the X server before presenting the user the login screen, optionally repeating when the user logs out. In this condition, the DM realizes in the X Window System the functionality of getty and login on character-mode terminals. When the display manager runs on a remote computer, it acts like a telnet server, requesting username and password and starting a remote session.

Specifically for this project we'll be using **LightDM** as our display manager,

LightDM is a free and open-source X display manager that aims to be lightweight, fast, extensible and multi-desktop. It can use various front-ends to draw User Interface, also called *Greeters*. It also supports Wayland.

LightDM is the default display manager for Edubuntu, Xubuntu and Mythbuntu since 11.10 release, for Lubuntu since 12.04 release, for Kubuntu beginning with 12.10 until 15.04 for Linux Mint and Antergos.

LightDM features :

- codebase with very few dependencies
- supports different display technologies (X, Mir, Wayland ...)
- Lightweight - low memory usage and fast performance
- Supports remote login (incoming - XDMCP, VNC, outgoing - XDMCP, pluggable)
- Comprehensive test suite.
- standards-compliance (PAM, logind, etc)
- well-defined interface between the server and user interface
- cross-desktop (greeters can be written in any toolkit)
- well-defined greeter API allowing multiple GUIs
- support for all display-manager use-cases, with plug-ins where appropriate

LightDM has a simpler code base than GDM and does not load any GNOME libraries to work, but at the cost of some features, that the user may or may not need.[17][18]

WINDOW MANAGERS

A window manager is system software that controls the placement and appearance of windows within a windowing system in a graphical user interface. Most window managers are designed to help provide a desktop environment. They work in conjunction with the underlying graphical system that provides required functionality—support for graphics hardware, pointing devices, and a keyboard, and are often written and created using a widget toolkit.

Few window managers are designed with a clear distinction between the windowing system and the window manager. Every graphical user interface based on a windows metaphor has some form of window management. In practice, the elements of this functionality vary greatly. Elements usually associated with window managers allow the user to open, close, minimize, maximize, move, resize, and keep track of running windows, including window decorators. Many window managers also come with various utilities and features: e.g. docks, task bars, program launchers, desktop icons, and wallpaper.

Early days : In the 1970s, the Xerox Alto became the first computer shipped with a working WIMP GUI. It used a stacking window manager that allowed overlapping windows. While it is unclear if Microsoft Windows contains designs copied from Apple's Mac OS, it is clear that neither was the first to produce a GUI using stacking windows. In the early 1980s, the Xerox Star, successor to the Alto, used tiling for most main application windows, and used overlapping only for dialogue boxes, removing most of the need for stacking.

Mac OS was one of the earliest commercially successful examples of a GUI that used a sort of stacking window management via QuickDraw. Currently macOS uses a somewhat more advanced window manager that has supported compositing since Mac OS X 10.0, and was updated in Mac OS X 10.2 to support hardware accelerated compositing via the Quartz Compositor.

GEM 1.1 was a window manager that supported the desktop metaphor, and used stacking, allowing all windows to overlap. It was released in the early 1980s. GEM is famous for having been included as the main GUI used on the Atari ST, which ran Atari TOS, and was also a popular GUI for MS-DOS prior to the widespread use of Microsoft Windows. As a result of a lawsuit by Apple, GEM was forced to remove the stacking capabilities, making it a tiling window manager.

During the mid-1980s, Amiga OS contained an early example of a compositing window manager called *Intuition* (one of the low-level libraries of AmigaOS, which was present in Amiga system ROMs), capable of recognizing which windows or portions of them were covered, and which windows were in the foreground and fully visible, so it could draw only parts of the screen that required refresh. Additionally, Intuition supported compositing. Applications could first request a region of memory outside the current display region for use as bitmap. The Amiga windowing system would then use a series of bit blits using the system's hardware blitter to build a composite of these applications' bitmaps, along with buttons and sliders, in display memory, without requiring these applications to redraw any of their bitmaps.

Intuition also anticipated the choices of the user by recognizing the position of the pointer floating over other elements of the screen (title bars of windows, their close and resizing gadgets, whole icons), and thus it was capable of granting nearly a zero-wait state experience to the use of the Workbench window manager.

Noteworthy to mention is the fact that Workbench was the only window manager that eventually inspired an entire family of descendant and successors: Ambient in MorphOS, Zune/Wanderer in AROS, Workbench NG (New Generation in AmigaOS 4.0 and 4.1). Workbench 4.1 was enhanced by 2D vector interface powered by Cairo libraries, and presenting a modern Porter-Duff 3D based Compositing Engine.

In 1988, Presentation Manager became the default shell in OS/2, which, in its first version, only used a command line interface (CLI). IBM and Microsoft designed OS/2 as a successor to DOS and Windows for DOS. After the success of the Windows 3.10, however, Microsoft abandoned the project in favor of Windows. After that, the Microsoft project for a future OS/2 version 3 became Windows NT, and IBM made a complete redesign of the shell of OS/2, substituting the Presentation Manager of OS/2 1.x for the object-oriented Workplace Shell that made its debut in the OS/2 2.0.

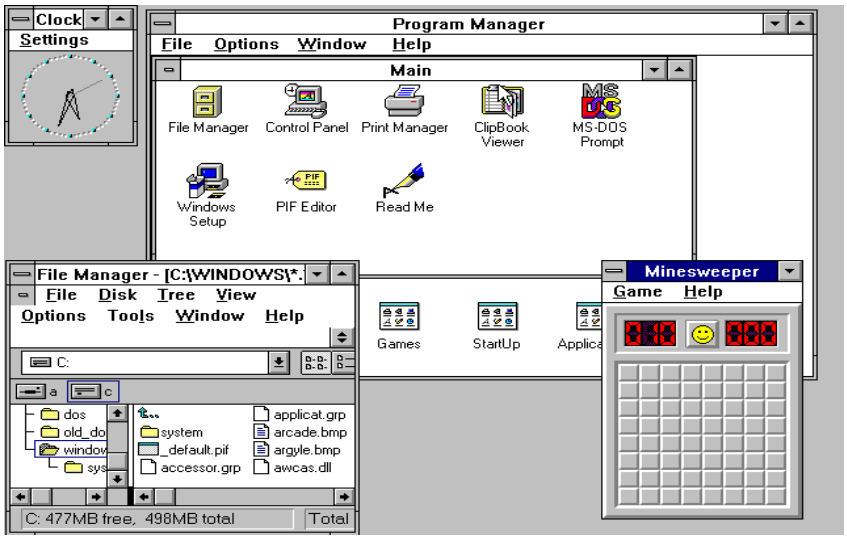
Types of window managers :

- Compositing window managers

Compositing window managers let all windows be created and drawn separately and then put together and displayed in various 2D and 3D environments. The most advanced compositing window managers allow for a great deal of variety in interface look and feel, and for the presence of advanced 2D and 3D visual effects.

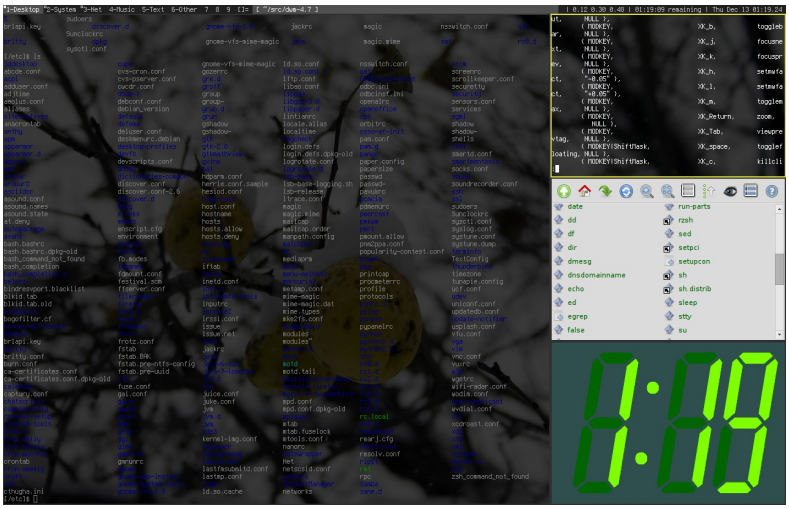
- Stacking window managers

All window managers that have overlapping windows and are not compositing window managers are stacking window managers, although it is possible that not all use the same methods. Stacking window managers allow windows to overlap by drawing background windows first, which is referred to as the painter's algorithm. Changes sometimes require that all windows be re-stacked or repainted, which usually involves redrawing every window. However, to bring a background window to the front usually only requires that one window be redrawn, since background windows may have bits of other windows painted over them, effectively erasing the areas that are covered.



- Tiling window manager

Tiling window managers paint all windows on-screen by placing them side by side or above and below each other, so that no window ever covers another. Microsoft Windows 1.0 used tiling, and a variety of tiling window managers for X are available.



- Dynamic window manager

Dynamic window managers can dynamically switch between tiling or floating window layout. A variety of dynamic window managers for X are available.

Features of window managers :

- Autohide
  An autohide facility enables menubars to disappear when the pointer is moved away from the edge of the screen.

- Borders
  A border is a window decoration component provided by some window managers, that appears around the active window. Some window managers may also display a border around background windows.

- Context Menu
  Some window managers provide a context menu that appears when an alternative click event is applied to a desktop component.

- Desktop Wallpaper
  Some window managers provide a desktop wallpaper facility that displays a background picture in the root window.

- Focus Stealing
  Focus stealing is a facility some window managers provide. It allows an application not in focus to suddenly gain focus and steal user input intended for the previously focused application.

- Iconification
  An iconification facility lets users minimize running applications to a desktop icon or taskpanel icon.

- Joined Windows
  Some window managers provide a joined windows facility that lets user join application window frames together.

- Keyboard Equivalents
  Some window managers provide keyboard equivalents that enables the keyboard to replicate mouse functionality.

- Menubar
  A menubar provides the facility to launch programs via a menu and may contain additional facilities including a start button, a taskbar, and a system tray.

- Menu Panel
  A menu panel a component of some window managers that provides the facility to launch programs using a menu. A menu panel is similar to a menubar, but appears as a floating panel, rather than a horizontal or vertical bar. The menu panel may contain additional facilities including a start button, a task panel, and a system tray.

- Mouse focus
  The mouse focus model determines how the pointing device affects the input focus within the window manager. The focus model determine which component of the graphical user interface is currently selected to receive input as the pointer is moved around the screen.

- Mouse warping
  Mouse warping is a facility that centres the pointer on the current application as it is made active.

- Multiple Desktops
  A window manager may provide a multiple desktops facility. This enables switching between several root window desktops. This prevents clutter of the root window, because applications can run on different desktops.

- Pager
  Some window managers provide a pager tool that provides the facility to switch between multiple desktops. The pager may appear as an onscreen window or as a gadget in the taskbar or taskpanel.

- Plugins

Some window managers have a modular construction that enables plug-in modules to provide features as required.

- Rollup
  A rollup facility enables windows to appear as just a titlebar on the desktop.

- Root Menu
  Some window managers provide a root menu, which appears when the root window or desktop background is touched.

- Shortcuts
  Some window managers provide a shortcut facility that lets users place icons on the root window that access specific programs or facilities.

- Tabbed Windows
  Some window managers provide a tabbed windows facility that groups applications together in common frames.

- Task Switching
  The window manager may provide various task switching facilities that let the user change the currently focused application, including: Changing the mouse focus using a pointing device, Keyboard task switching facilities (for example, by pressing Alt-Tab) Clicking on the task in a taskbar or taskpanel

- Taskbar
  Some window managers provide a taskbar that shows running applications. The taskbar may show all applications that are running including those that have been minimized, and may provide the facility to switch focus between them. The taskbar may be incorporated into a menubar on some window managers.

- Task Panel
  A task panel is similar to a taskbar, but appears as a floating panel, rather than a horizontal or vertical bar.

- Start Button
  A start button is a desktop widget that provides a menu of programs that can be launched. The start button is typically placed on a menubar at the bottom of the screen.

- Notification Area
  A Notification Area is used to display icons for system and program features that have no desktop window. It contains mainly icons to indicate status information and notifications such as arrival of a new mail message. Some systems may also show a clock in the Notification Area.

- Title Bars
  A titlebar is a window decoration component some window managers provide at the top of each window. The titlebar is typically used to display the name of the application, or the name of the

open document, and may provide title bar buttons for minimizing, maximizing, closing or rolling up of application windows.

Title Bar Buttons

Title bar buttons are included in the titlebar of some window managers, and provide the facility to minimize, maximize, rollup or close application windows. Some window managers may display the titlebar buttons in the taskbar or task panel, rather than in a titlebar.

- Virtual Desktop
  A virtual desktop (also called a scrolling desktop) is a facility some window managers provided that lets the desktop be larger than the actual screen.

Specifically for this porject we'll be using **Tiling window manager** as it provides most efficient and optimised workflow. Under Tiling window manager we'll be using i3WM.

Linux Distribution : Arch Linux

Desktop Environment : none

Display Manager : LightDM

Window Manager : Tiling Window Manager [i3WM]

i3WM (Tiling Window Manger)

i3 is a tiling window manager designed for X11, inspired by wmii, and written in C. It supports tiling, stacking, and tabbing layouts, which it handles dynamically. Configuration is achieved via plain text file and extending i3 is possible using its Unix domain socket and JSON based IPC interface from many programming languages.

Like wmii, i3 uses a control system very similar to vi. By default, window focus is controlled by the 'Mod1' (Alt key/Win key) plus the right hand home row keys (Mod1+J,K,L,;), while window movement is controlled by the addition of the Shift key (Mod1+Shift+J,K,L,;).

Sway is another window manager with the intention of providing a "drop-in replacement" for i3 using the Wayland display server protocol.

Goals :

- Possess well written, documented code, that encourages user contribution.
- Use XCB instead of Xlib.
- Implement multi-monitor features correctly, so that each workspace is assigned to a virtual screen, and monitor additions and removals are non-destructive of windows.
- Implement different modes, similar to the text editor vi and vim. That is, keys have different functions depending on the mode the window manager is in.
- Use a tree as the abstraction (and underlying data structure) for window management.
- Implement UTF-8 character encoding.

Features :

- The configuration is done via a plain text file (found in ~/.config/i3/config).
- i3 can be customized without programming.
- Contrary to other popular tiling window managers, such as dwm, awesome, and xmonad, window management is left to the user in i3. Windows are held inside containers, which can be split vertically or horizontally. They can also optionally be resized. There are also options for stacking the windows, as well as tabbing them (similar to the interface that web browsers use).
- Uses dmenu as the default program launcher, but it can be replaced.
- Write well readable, well documented code. Create additional documentation on how to extend i3 by explaining its internal workings.
  This includes being modifiable by people who do know how to program but who are not necessarily familiar with all of X11's internals. That is, document why things happen and when they happen so that the user gets a picture of the whole process a Window Manager is responsible of by just reading the source code.
- Use xcb instead of Xlib. xcb has a much cleaner API and should be faster in quite a lot of situations.

- Implement multi-monitor correctly, that is by assigning each workspace to a virtual screen. Especially make sure that attaching and detaching new monitors like video projectors works during operation and does the right thing. Also provide support for rotated monitors.
- Use a tree as data structure. This allows for more flexible layouts than the column-based approach used by other window managers.
- Implement different modes, like in vim. You can use different keybindings when in the 'resize' mode than when you are in the default mode, for example.
- Implement an IPC interface for other programs. Provide subscription to certain events and accept commands.
  This approach should be more lightweight than wmii's usage of the 9P filesystem. Furthermore, core functionality does not depend on a separate program, so that i3 runs faster, especially when your system is under load.
- Be UTF-8 clean.
- The usual elitism amongst minimal window managers: Don't be bloated, don't be fancy (simple borders are the most decoration we want to have).
  However, we do not enforce unnecessary limits such as a maximum amount of source lines of code. If it needs to be a bit bigger, it will be.

Every i3WM system operates on a specific config file which tells the manager how to display different elements on the screen.

Default configuration is generally found on ~/.config/i3/config .

Under configuration(i3WM) i3blocks is implemented in 'bar' module as to produce user defined top bar view.

Minimal config ::

This file has been auto-generated by i3-config-wizard

i3 config file (v4)

set $mod Mod4

set $ALT Mod1

font -*-gohufont-medium-r-normal-*-14-100-100-100-*-*-iso10646-1

floating_modifier $ALT

start a terminal

bindsym $mod+Return exec i3-sensible-terminal

bindsym $ALT+F1 exec urxvtc

```
# kill focused window
# bindsym $mod+Shift+Q kill
bindsym $ALT+F4 kill
# startup apps and window decs
default_orientation horizontal
workspace_layout tabbed
new_window 1pixel
# window colours: border background text
client.focused        #0F0F0F #7FB256 #F5F5FF
client.focused_inactive    #0F0F0F #7FB256 #E5E5E5
client.unfocused    #0F0F0F #5697B2 #E5E5E5
client.urgent        #0F0F0F #BC9B54 #E5E5E5
exec xrdb -l /home/doug/.Xdefaults
exec xmodmap /home/doug/.Xmodmap
exec xsetroot -mod 1 1 -fg mediumslateblue -bg lightslategray
exec xset +fp /usr/local/share/fonts/programing/
exec xset fp rehash
exec urxvtd -q -f -o
exec hsetroot -center /home/doug/wallpaper/MoonSky.jpg
exec update-notifier
exec nm-applet
exec xfce4-power-manager
# start dmenu (a program launcher)
# bindsym $mod+d exec dmenu_run
bindsym $ALT+F2 exec dmenu_run -fn "xft:droid sans:bold:pixelsize=11:antialias=true:hinting=slight"
-nb "#0f0f0f" -nf "#a6a6a6" -sb "#0f0f0f" -sf "#8f8fed"
bindsym XF86HomePage exec FireFox
bindsym $mod+F14 exec thunderbird-mail
bindsym $mod+Control+l exec geany
```

```
bindsym Menu exec thunar
bindsym $mod+Control+g exec gimp
bindsym $mod+F13 exec 'gksudo synaptic'
bindsym XF86AudioRaiseVolume exec amixer sset Master,0 2+
bindsym XF86AudioLowerVolume exec amixer sset Master,0 2-
bindsym XF86AudioMute exec amixer sset Master,0 toggle
assign [class="Chromium"] 2
for_window [class="Chromium"] border 1pixel
assign [class="Navigator"] 2
for_window [class="Navigator"] border 1pixel
assign [class="Firefox"] 2
for_window [class="Firefox"] border 1pixel
assign [class="Thunderbird"] 2
for_window [class="Thunderbird"] border 1pixel
assign [class="Gimp"] 3
for_window [class="Gimp"] border normal
for_window [class="Gimp"] floating enable
assign [class="Gnome-mplayer"] 4
for_window [class="Gnome-mplayer"] floating enable
for_window [class="XFontsel"] floating enable
for_window [title="Event Tester"] floating enable
for_window [title="Xfce-notifyd"] floating enable
for_window [class="Sakura"] floating enable
for_window [class="Sakura"] border normal
# change focus
bindsym $mod+Down focus down
bindsym $mod+Up focus up
bindsym $ALT+Tab focus right
```

```
bindsym $ALT+Shift+Tab focus left

# move focused window
bindsym $mod+Shift+J move left
bindsym $mod+Shift+K move down
bindsym $mod+Shift+L move up
bindsym $mod+Shift+colon move right
# alternatively, you can use the cursor keys:
bindsym $mod+Shift+Left move left
bindsym $mod+Shift+Down move down
bindsym $mod+Shift+Up move up
bindsym $mod+Shift+Right move right
# split in horizontal orientation
bindsym $mod+h split h
# split in vertical orientation
bindsym $mod+v split v
# enter fullscreen mode for the focused container
bindsym $mod+f fullscreen
# change container layout (stacked, tabbed, default)
bindsym $mod+s layout stacking
bindsym $mod+t layout tabbed
bindsym $mod+e layout default
# toggle tiling / floating
bindsym $mod+Shift+space floating toggle
# change focus between tiling / floating windows
bindsym $mod+space focus mode_toggle
bindsym $mod+a focus parent
# focus the child container
#bindcode $mod+d focus child
```

```
# switch to workspace
bindsym $mod+1 workspace 1
bindsym $mod+2 workspace 2
bindsym $mod+3 workspace 3
bindsym $mod+4 workspace 4
bindsym $mod+5 workspace 5
bindsym $mod+6 workspace 6
bindsym $mod+7 workspace 7
bindsym $mod+8 workspace 8
bindsym $mod+9 workspace 9
bindsym $mod+0 workspace 10
# move focused container to workspace
bindsym $mod+Shift+exclam move container to workspace 1
bindsym $mod+Shift+at move container to workspace 2
bindsym $mod+Shift+numbersign move container to workspace 3
bindsym $mod+Shift+dollar move container to workspace 4
bindsym $mod+Shift+percent move container to workspace 5
bindsym $mod+Shift+asciicircum move container to workspace 6
bindsym $mod+Shift+ampersand move container to workspace 7
bindsym $mod+Shift+asterisk move container to workspace 8
bindsym $mod+Shift+parenleft move container to workspace 9
bindsym $mod+Shift+parenright move container to workspace 10
bindsym $mod+Shift+C reload
bindsym $mod+q restart
bindsym $mod+Shift+Q exit
mode "resize" {
        bindsym j resize shrink left 10 px or 10 ppt
        bindsym Shift+J resize grow   left 10 px or 10 ppt
```

```
        bindsym k resize shrink down 10 px or 10 ppt
        bindsym Shift+K resize grow   down 10 px or 10 ppt
        bindsym l resize shrink up 10 px or 10 ppt
        bindsym Shift+L resize grow   up 10 px or 10 ppt
        bindsym semicolon resize shrink right 10 px or 10 ppt
        bindsym Shift+colon resize grow   right 10 px or 10 ppt
        bindsym Left resize shrink left 10 px or 10 ppt
        bindsym Shift+Left resize grow   left 10 px or 10 ppt
        bindsym Down resize shrink down 10 px or 10 ppt
        bindsym Shift+Down resize grow   down 10 px or 10 ppt
        bindsym Up resize shrink up 10 px or 10 ppt
        bindsym Shift+Up resize grow   up 10 px or 10 ppt
        bindsym Right resize shrink right 10 px or 10 ppt
        bindsym Shift+Right resize grow   right 10 px or 10 ppt
        # back to normal: Enter or Escape
        bindsym Return mode "default"
        bindsym Escape mode "default"
}
bindsym $mod+r mode "resize"
bar {
        status_command conky
        position top
        font -xos4-terminus-medium-r-normal--12-120-72-72-C-60-iso8859-1
        # font -*-proggysquaresz-medium-r-normal-*-11-80-*-*-*-*-iso8859-1
    workspace_buttons yes
        colors  {
        background #0F0F0F
        statusline #D5D5D5
```

```
        focused_workspace #F5F5FF #7FB256
        active_workspace #E5E5E5 #5697B2
        inactive_workspace #E5E5E5 #5697B2
        urgent_workspace #E5E5E5 #BC9B54
}
}
```

Bar module ::

```
bar {
        font pango:monaco Bold 8
        status_command i3blocks -c ~/Documents/DOTS/i3blocks/config
        position top
        mode dock
        modifier None
        colors {
                background $bg-color
                separator #757575
                #                  border             background         text
                focused_workspace  $bg-color          $bg-color          $text-color
                inactive_workspace $inactive-bg-color $inactive-bg-color $inactive-text-color
                urgent_workspace   $urgent-bg-color   $urgent-bg-color   $text-color
        }
}
```

Key binds ::

```
bindsym $mod+Mod2+KP_4 exec --no-startup-id playerctl --player=spotify previous
bindsym $mod+Mod2+KP_5 exec --no-startup-id playerctl --player=spotify play-pause
bindsym $mod+Mod2+KP_6 exec --no-startup-id playerctl --player=spotify next
```

i3Blocks and i3block

i3Blocks

A feed generator for text based status bars

i3blocks executes your command lines and generates a status line from their output.

Commands are scheduled at configured time intervals, upon signal reception or on clicks.

The generated line is meant to be displayed by the i3 window manager through its i3bar component, as an alternative to i3status.is meant to be highly flexible but intuitive.

No library package is required, just output what your status bar expects, from your favorite programming language and your preferred format.

Example :

```
[click]
full_text=Click me!
command=echo "Got clicked with button $button"
color=#F79494

# Guess the weather hourly
[weather]
command=curl -Ss 'https://wttr.in?0&T&Q' | cut -c 16- | head -2 | xargs echo
interval=3600
color=#A4C2F4

# Query my default IP address only on startup
[ip]
command=hostname -i | awk '{ print "IP:" $1 }'
interval=once
color=#91E78B

# Update time every 5 seconds
[time]
command=date +%T
interval=5
```

Properties :

- Command

The optional *command* property specifies a command line to be executed with sh -c. The command can be relative to the configuration file where it is defined. If the command outputs some text, it is used to update the block.

An exit code of 0 means success. A special exit code of *33* will set the *urgent* i3bar key to true. Any other exit code will raise an error.

```
[pacman]
full_text=c ·
command=echo "· ${full_text~~}"
color=#FFFF00
```

- Interval

The optional *interval* property specifies when the command must be scheduled.

A positive value represents the number of seconds to wait between exectutions.

```
# Print seconds since 1970-01-01
[epoch]
command=date +%s
interval=1
```

A value of *0* (or undefined) means the command is not timed whatsoever and will not be executed on startup. This is useful to trigger the command only on user input (e.g. signal or click), not before.

```
# Restart i3 on click
[restart]
full_text=Restart
command=i3-msg -q restart
#interval=0
```

The interval value *once* (or *-1*) will schedule the command only on startup. This tells i3blocks not to schedule the command again on a time basis. But events such as signals and clicks will execute the command again of course.

```
# Fetch the public IP address only on startup
[public-ip]
command=wget -qO - icanhazip.com
interval=once
```

The interval value *repeat* (or *-2*) will respawn the command as soon as it terminates. This is convenient for blocking programs which exit as soon as the awaited event arises.

```
# Print the last command entered in Bash
[history]
command=inotifywait -qq -e close_write ~/.bash_history; tail -1 ~/.bash_history
interval=repeat
```

The interval value *persist* (or *-3*) expects the command to be an infinite loop. Each line of the output will trigger an update of the block.

```
[window]
command=xtitle -s
interval=persist
```

- Signal

Blocks can be scheduled upon reception of a real-time signal (think prioritized and queueable). The range of available signal numbers is *1* to *N*, where *SIGRTMIN+N = SIGRTMAX*. (Note: there are 31 real-time signals in Linux.)

```
[caps-lock]
command=xset -q | grep Caps | awk '{ print $2, $3, $4 }'
interval=once
signal=10
```

This example block above will be scheduled once i3blocks handles the *SIGRTMIN+10* signal. This can be sent directly from an i3 binding on Caps Lock release with the following configuration:

```
bindsym --release Caps_Lock exec pkill -SIGRTMIN+10 i3blocks
```

- Format

There are several formats supported to specify which variables i3blocks must update. Some favor simplicity over flexibility but thus can be limited.

When undefined, a raw format is assumed. Each line of the output corresponds to an i3bar key, in the order of definition found in the i3bar protocol:

- the 1st line updates the *full_text*;
- the 2nd line updates the *short_text*;
- the 3rd line updates the *color*;
- the 4th line updates the *background*.

Excess lines are considered an error. Below is an example of a simple battery script.

battery.sh

```
#!/bin/bash

BAT=$(acpi -b | grep -E -o '[0-9][0-9]?%')

# Full and short texts
echo "Battery: $BAT"
echo "BAT: $BAT"

# Set urgent flag below 5% or use orange below 20%
```

```
[ ${BAT%?} -le 5 ] && exit 33
[ ${BAT%?} -le 20 ] && echo "#FF8000"
exit 0

[battery]
command=battery.sh
interval=10
```

The *json* format can update any variable.

```
[counter]
_count=0
command=printf '{"full_text":"Counter: %s", "_count":%d}\n' $_count $((_count + 1))
format=json
interval=1
```

here, keywords enclosed within square brackets identifies singular block that is to be implemented in the config file.

i3block

The configuration file uses a simplified INI file format:

```
# Properties not preceded by a section are considered global
# and merged into every section declarations.

foor=bar
[block1]
baz=qux
# This is a comment
[block2]
quux= quuz
```

In this example, *block2* contains a *foo* property equal to *"bar"* and a *quux* property equal to *" quuz"* (including the leading space). Everything after the equal sign will be part of the value, thus inline comments won't be stripped out.

At runtime, these properties are simply variables, that are passed along to the status bar program when printing is necessary. However on startup, i3blocks checks some optional properties to eventually setup the scheduling of a command.

If a block specifies a command, then all of its properties are passed as environment variables at execution, which means that the *foo=bar* property will be available from a shell script with $foo. The output of the command is used to update the values of these variables. The values are reset to default (as defined in the configuration file) before the update, so that blocks get a consistent behavior at each execution.

i3block script (Spotify)

This Script is written in python having **dependencies:**

- imports sys
- imports html
- imports time
- imports dbus
- from dbus.mainloop.glib import DBusGMainLoop
- from gi.repository import Glib

Spotify Script ::

```
#!/usr/bin/env python3

import sys

import html

import time

import dbus

from dbus.mainloop.glib import DBusGMainLoop

from gi.repository import GLib

class SpotifyBlocklet:

    BUS_NAME = 'org.mpris.MediaPlayer2.spotify'

    OBJECT_PATH = '/org/mpris/MediaPlayer2'

    PLAYER_INTERFACE = 'org.mpris.MediaPlayer2.Player'

    PROPERTIES_INTERFACE = 'org.freedesktop.DBus.Properties'
```

```
STATUS_TO_ICON = {
        'Playing': '▶',
        'Paused': '▮▮',
        'Stopped': '■',
    }
    def __init__(self):
        self.loop = GLib.MainLoop()
        DBusGMainLoop(set_as_default=True)
    def run(self):
        self.bus = dbus.SessionBus()
        self.spotify = self.bus.get_object(
            bus_name=self.BUS_NAME,
            object_path=self.OBJECT_PATH,
            follow_name_owner_changes=True,
        )
        self.connect_to_dbus_signals()
        self.show_initial_info()
        self.loop.run()
    def run_forever(self):
        while True:
            try:
                self.run()
```

```
            except dbus.exceptions.DBusException:
                time.sleep(1)
            except KeyboardInterrupt:
                break
    def connect_to_dbus_signals(self):
        self.spotify.connect_to_signal(
            signal_name='PropertiesChanged',
            handler_function=self.on_properties_changed,
            dbus_interface=self.PROPERTIES_INTERFACE,
        )
        self.bus.get_object(
            bus_name='org.freedesktop.DBus',
            object_path='/org/freedesktop/DBus',
        ).connect_to_signal(
            signal_name='NameOwnerChanged',
            handler_function=self.on_name_owner_changed,
            dbus_interface='org.freedesktop.DBus',
            arg0=self.BUS_NAME,
        )
    def on_properties_changed(self, interface_name, changed_properties, _):
        """Show updated info when playback status or track is changed"""
        self.show_info(
            status=changed_properties['PlaybackStatus'],
```

```
            metadata=changed_properties['Metadata'],
        )


def on_name_owner_changed(self, name, old_owner, new_owner):
        """Clear info when Spotify is closed"""
        if old_owner and not new_owner:
            print(' ')
            sys.stdout.flush()
    def get_property(self, property_name):
        return self.spotify.Get(
            self.PLAYER_INTERFACE, property_name,
            dbus_interface=self.PROPERTIES_INTERFACE,
        )
    def show_initial_info(self):
        self.show_info(
            status=self.get_property('PlaybackStatus'),
            metadata=self.get_property('Metadata'),
        )
    def show_info(self, status, metadata):
        status_icon = self.STATUS_TO_ICON.get(status, '?')
        artist = ', '.join(metadata['xesam:artist'])
        title = metadata['xesam:title']
        info = '{status_icon} {artist} — {title}'.format(
```

```
            status_icon=status_icon,
            artist=artist,
            title=title,
        )
        print(html.escape(info))
        sys.stdout.flush()
if __name__ == '__main__':
    SpotifyBlocklet().run_forever()
```

NOTE :: this script once executed, executes till the service is alive and then passively waits for the service to spawn or die.

Similarly for ‘STATUS_TO_ICON’ there are unique keys bound in i3Blocks config so as to control music and navigation with keyboard only. Key binds ::

```
[spotify]
label=
#command=~/.config/i3blocks/spotify.py
command=~/Documents/DOTS/i3blocks/spotify.py
color=#81b71a
interval=persist
[spotify-previous]
full_text=
color=#FFD54F
command=playerctl --player=spotify previous
[spotify-play-pause]
full_text=
```

color=#FF7043

command=playerctl --player=spotify play-pause

[spotify-next]

full_text=⏩

command=playerctl --player=spotify next

Installation and usage ::

- create a python script containing above code.
- Make the script executable with chmod +x spotify.py
- make i3blocks execute with i3WM (configure ‘bar’ module)
- create i3block in i3Blocks config under ~/.config/i3blocks/config
- reload i3WM with mod+F2.

Interacting with the block, either use keyboard shortcuts to navigate or control the behaviour script or use mouse to PLAY/PAUSE, PREVIOUS or NEXT music currently being streamed.

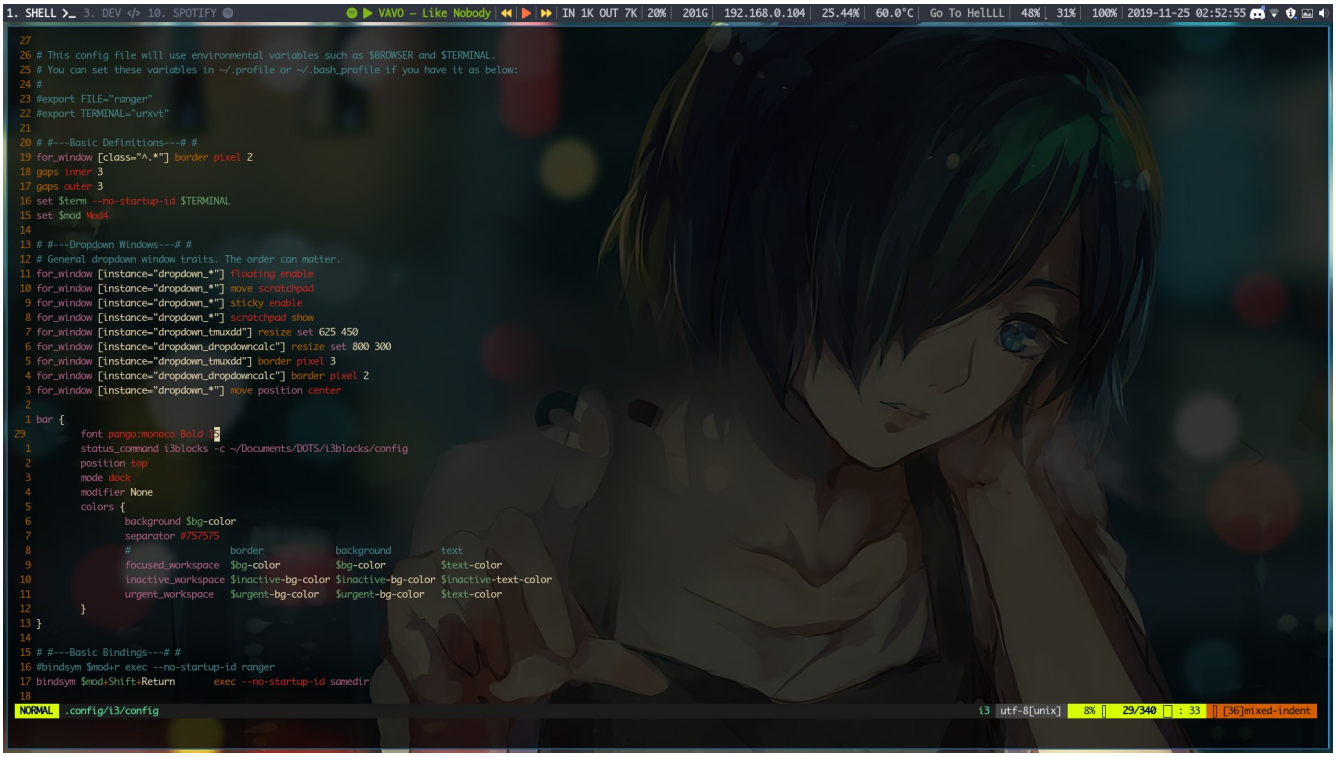
OUTPUTS :

- service non-existent

- service running with song currently playing

- service running with song currently paused

- whole screen screenshot



PROS AND CONS

PROS :

- Very fast and efficient
- No input lag with keyboard shotcuts
- spaws only when service spawns otherwise doesn’t takes up desktop space
- uses python as base language
- less complex code
- highly customisable
- full color support.

CONS :

- Dumps HTML as output hence, often displays codes for certain characters
- often only displays the spotify logo on the bar
- Padding optimisations are required.
- Onclick lyrics functionality missing.

CONCLUSION

Finally, We have successfully displayed the name and artist of currently playing song via spotify using python-dbus and spotify services.

Also there are fields where there is scope of improvement such as real time lyrics and image support for song, albums and artists. This is practically first step towards making music streaming hassel free on WM like i3. Hence, hoping to continue to build this porject further.

References ::

- Arch linux WIKI
- GITHUB
- Archlinux Documentation
- i3WM Documentation
- WIKIPEDIA
- itsFOSS