حسین سیلانی
طراح، توسعه دهنده وسیستم ادمین

Book 6\100

UNIX PHILOSOPHY

Concepts 1st Edition

کتاب مرجع فلسفه یونیکس

معرفی کوتاه کتاب

کتاب Unix Philosophy Concepts یک راهنمای تفکر و بینش‌محور است که به جای آموزش دستورات فنی یا روش‌های کار با سیستم‌عامل، ذهن خواننده را با اصول بنیادین پشت طراحی سیستم‌های شبه‌یونیکس آشنا می‌سازد. این کتاب برای علاقه مندان به فلسفه طراحی نرم‌افزار، برنامه‌نویسان، مهندسان نرم‌افزار و حتی کاربران کنجکاو نوشته شده که می‌خواهند بدانند چرا یونیکس چنین تاثیر ماندگاری در دنیای فناوری داشته است.این کتاب لزوماً برای آموزش عملی یونیکس نیست؛ بلکه یک دعوت است برای اندیشیدن عمیق‌تر به ریشه‌های طراحی نرم‌افزار..

فلسفه یونیکس
Unix Philosophy

حسین سیلانی

ویرایش اول

۱۴۰۴

سرشناسه	سیلانی، حسین،
عنوان و نام پدیدآور	فلسفه یونیکس Unix Philosophy
	حسین سیلانی؛ ویراستاری حسین سیلانی.
مشخصات نشر	تهران: کیان دانش، ۱۴۰۴.
مشخصات ظاهری	۲۷۰ ص.
شابک	۹۷۸،۶۲۲،۴۰۰،۰۱۵،۶
وضعیت فهرست نویسی	فیپا
موضوع	سیستم عامل
	Operating System
موضوع	سیستم‌های عامل (کامپیوتر)
	Operating systems (Computers)
شناسه افزوده	سیلانی، حسین، ۱۳۶۰،
رده بندی کنگره	۷۶/۷۶QA
رده بندی دیویی	۴۳۲/۰۰۵
شماره کتابشناسی ملی	۹۶۷۳۵۹۳
اطلاعات رکورد کتابشناسی	فیپا

انتشارات یافته : ناشر کتاب‌های دانشگاهی

عنوان: فلسفه یونیکس

تالیف: حسین سیلانی

صفحه آرایی و ویراستاری: حسین سیلانی

چاپ و صحافی: موسسه مهراد

طراحی جلد: حسین سیلانی

چاپ اول: ۱۴۰۴

شمارگان: ۱۰۰ جلد

شابک: ۶،۰۱۵،۴۰۰،۶۲۲،۶۹۷

خیابان ۱۲ فروردین، خیابان شهدای ژاندارمری، شماره ۱۲۶، طبقه چهارم

سخنی از نویسنده

با سلام و احترام به دوست گرامی

از اینکه این کتاب را انتخاب کرده‌اید و در این مسیر همراه من هستید صمیمانه سپاسگزارم. امیدوارم این اثر برای علاقه‌مندان به حوزه لینوکس مفید و اثربخش باشد. همچنین از شما درخواست دارم که اگر نواقص یا کاستی‌هایی در کتاب مشاهده کردید آن‌ها را نادیده نگیرید و با ارائه نظرات و پیشنهادات سازنده‌ی خود، به بهبود کیفیت این اثر کمک کنید.

شایان ذکر است که این کتاب یکی از صدها کتاب لینوکسی من است که بسیاری از آن‌ها در حال ترجمه یا نگارش هستند. این مجموعه در راستای پروژه‌های متن‌باز شخصی‌ام و با هدف طراحی و توسعه توزیع‌های لینوکس نوشته شده است. امیدوارم این تلاش‌ها گامی هرچند کوچک در جهت گسترش دانش و استفاده از سیستم‌عامل‌های متن‌باز باشد.

- **https://predator.os.ir/**
- **https://little.Psycho.ir**
- **https://emperor.os.ir/**
- **https://hubuntu.ir**

تصمیم دارم محتوای آموزشی فارسی در حوزه‌های لینوکس و نرم‌افزارهای متن‌باز را در قالب‌های متنوعی مانند کتابچه، کتاب، فلش‌کارت، فایل‌های صوتی و ویدئوهای آموزشی تولید کرده و از طریق یک سایت در دسترس عموم قرار دهم. هدف من از این کار، ایجاد یکی از بزرگترین و جامع‌ترین مراجع آموزشی فارسی در این زمینه است.برای دسترسی به سایت و اطلاع از آخرین اخبار و محتواهای آموزشی، می‌توانید از طریق راه‌های ارتباطی زیر با من در تماس باشید:

- **آدرس سایت:** learninghive.ir
- **کانال تلگرام:** @Linuxtnt
- **آیدی تلگرام:** @seilany
- **ایمیل:** h.seilany@gmail.com
- **گیت‌هاب:** hosseinseilani

منتظر حضور و همراهی شما هستم!

تقدیم به تو

یگانه فرزانه دلم

فهرست مطالب

مقدمه

سیستم عامل، به عنوان یک موجود نرم افزاری زنده، نقش حیاتی در تبدیل الکترون‌ها و سیلیکون به یک شخصیت ماشینی ایفا می‌کند. این سیستم عصبی کامپیوتر است که به آن زندگی می‌بخشد و آن را از یک مجموعه سخت‌افزاری بی‌جان به یک ابزار قدرتمند و هوشمند تبدیل می‌کند. هر سیستم عامل، فلسفه و دیدگاه سازندگان خود را در خود جای داده است و این فلسفه در طراحی، رابط کاربری و عملکرد آن نمایان می‌شود. به عنوان مثال، سیستم عامل Mac/OS که توسط شرکت اپل توسعه یافته است، با رابط کاربری بصری و شی‌گرای خود، این پیام را به کاربران منتقل می‌کند که همه چیز درست در مقابل چشمان شما قرار دارد و نیازی به جست‌وجوی پیچیده برای انجام کارها نیست. این رویکرد، کاربران را به سادگی و زیبایی در تعامل با سیستم عامل تشویق می‌کند.

از سوی دیگر، MS-DOS، سیستم عاملی که توسط مایکروسافت توسعه یافت، به عنوان رهبر بلامنازع انقلاب رایانه‌های شخصی شناخته می‌شود. این سیستم عامل تلاش کرد تا قدرت پردازنده مرکزی را به دسکتاپ کاربران بیاورد و به آن‌ها این امکان را بدهد که به طور مستقیم با هسته سیستم تعامل داشته باشند. این رویکرد، اگرچه برای کاربران حرفه‌ای جذاب بود، اما برای کاربران عادی می‌توانست چالش‌برانگیز باشد. در مقابل، OpenVMS که توسط Digital Equipment Corporation توسعه یافت، فرض را بر این گذاشت که کاربران از ماشین‌های تفکر الکترونیکی می‌ترسند و بنابراین، باید تنها چند انتخاب قدرتمند و ساده به آن‌ها ارائه شود تا بتوانند کارهای خود را انجام دهند. این سیستم عامل سعی کرد تا با کاهش پیچیدگی‌ها، کاربران را به سمت استفاده آسان از سیستم هدایت کند.در این میان، سیستم عامل یونیکس با یک مفهوم رادیکال و متفاوت پا به عرصه گذاشت. طراحان یونیکس از همان ابتدا فرض کردند که کاربران این سیستم عامل از سواد کامپیوتری بالایی برخوردار هستند و می‌دانند که چه کاری انجام می‌دهند. فلسفه طراحی یونیکس حول این ایده می‌چرخید که کاربران باید به طور کامل بر سیستم مسلط باشند و بتوانند از ابزارهای قدرتمند آن برای انجام کارهای پیچیده استفاده کنند. این رویکرد، در تضاد کامل با سیستم‌عامل‌هایی بود که سعی می‌کردند خود را با طیف گسترده‌ای از کاربران، از مبتدی تا متخصص، تطبیق دهند. طراحان یونیکس به جای تلاش برای جذب کاربران کم‌تجربه، رویکردی سخت‌گیرانه در پیش گرفتند: اگر نمی‌توانید سیستم را درک کنید، به اینجا تعلق ندارید. این نگرش، اگرچه برای جامعه‌ی هکرها و متخصصان جذاب بود، اما باعث شد که یونیکس نتواند به سرعت در میان کاربران عادی مقبولیت پیدا کند.

متأسفانه، دنیای تجاری نیز در آن زمان ارزش چندانی برای یونیکس قائل نبود. این سیستم عامل به عنوان یک اسباب‌بازی برای هکرها و یک کنجکاوی فنی تلقی می‌شد و تعداد کمی از شرکت‌های سودآور حاضر بودند سرمایه‌گذاری خود را روی سیستمی که از یک آزمایشگاه تحقیقاتی آمده و در دانشگاه‌ها پرورش یافته بود، به خطر بیندازند. در نتیجه، یونیکس برای بیش از ۱۵ سال به عنوان یک قهرمان گمنام در دنیای فناوری باقی ماند. با این حال، همین فلسفه رادیکال و تمرکز بر قدرت و انعطاف‌پذیری، باعث شد که یونیکس به مرور زمان به عنوان یک سیستم عامل پایدار و قدرتمند شناخته شود و در نهایت، پایه‌ای برای بسیاری از سیستم‌عامل‌های مدرن، از جمله لینوکس وmacOS ، شود.

یونیکس با وجود چالش‌های اولیه، به مرور زمان تأثیر عمیقی بر صنعت فناوری اطلاعات گذاشت. این سیستم عامل نه تنها به عنوان یک ابزار قدرتمند برای متخصصان شناخته شد، بلکه فلسفه طراحی آن نیز الهام‌بخش بسیاری از

توسعه‌دهندگان شد. تمرکز یونیکس بر سادگی، ماژولاریته و قدرت، باعث شد که این سیستم عامل به عنوان یک استاندارد در دنیای سرورها و سیستم‌های بزرگ مطرح شود. حتی امروزه، بسیاری از سیستم‌عامل‌های مدرن، از جمله لینوکس وmacOS ، از اصول طراحی یونیکس الهام گرفته‌اند و تلاش می‌کنند تا تعادلی بین قدرت و سادگی برقرار کنند.

در اوایل دهه ۱۹۸۰، اتفاقی شگفت‌انگیز رخ داد که دنیای فناوری را برای همیشه تغییر داد. شایعاتی در مورد وجود یک سیستم عامل جدید پخش شد که وعده می‌داد انعطاف‌پذیری بیشتر، قابلیت حمل بالاتر و عملکرد بهتری نسبت به هر سیستم عامل دیگری که مردم در آن زمان استفاده می‌کردند، ارائه دهد. این سیستم عامل نه تنها قدرتمند بود، بلکه با هزینه‌ای بسیار کم در دسترس قرار گرفت و تقریباً روی هر دستگاهی قابل اجرا بود. این سیستم عامل، یونیکس بود، و پیام آن به قدری جذاب و انقلابی به نظر می‌رسید که بسیاری از مردم نمی‌توانستند باور کنند که چنین چیزی واقعاً وجود دارد. اما تاریخ بارها ثابت کرده است که ما اغلب در مواجهه با ایده‌های جدید و رادیکال، ابتدا مقاومت می‌کنیم و حتی ممکن است حاملان این اخبار جدید را مورد انتقاد قرار دهیم.

یونیکس با فلسفه‌ای رادیکال و متفاوت وارد دنیای فناوری شد. این سیستم عامل نه تنها از نظر فنی پیشرفته بود، بلکه دیدگاه جدیدی را در مورد نحوه تعامل کاربران با کامپیوترها ارائه می‌داد. اما همان‌طور که یونیکس شروع به نفوذ به دنیای محاسبات کرد، با مقاومت شدیدی از سوی ساختارهای بوروکراتیک و شرکت‌های بزرگ مواجه شد. بسیاری از این شرکت‌ها که سال‌ها بر روی سیستم‌های اختصاصی و رایانه‌های بزرگ سرمایه‌گذاری کرده بودند، از فکر یک انقلاب در دنیای فناوری وحشت داشتند. آن‌ها ترجیح می‌دادند در دنیای سفارشی‌سازی‌شده‌ای که خود ساخته بودند باقی بمانند، جایی که امنیت شغلی آن‌ها به دانش دستورات ساده و محدودی وابسته بود که برای انجام کارهای روزمره‌شان کافی بود. یونیکس به دشمنی برای این ساختارهای قدیمی تبدیل شد، نه به این دلیل که ذاتاً بد یا شیطانی بود، بلکه به این دلیل که وضعیت موجود را به چالش می‌کشید. این سیستم عامل با ارائه ابزارهای قدرتمند و انعطاف‌پذیر، کاربران را قادر می‌ساخت تا کنترل کامل بر سیستم‌های خود داشته باشند و این موضوع برای بسیاری از شرکت‌هایی که سال‌ها بر روی سیستم‌های بسته و محدود سرمایه‌گذاری کرده بودند، تهدیدی بزرگ محسوب می‌شد. یونیکس نه تنها روش‌های قدیمی انجام کارها را زیر سؤال می‌برد، بلکه این پتانسیل را داشت که بسیاری از مشاغل و نقش‌های سنتی در صنعت فناوری را منسوخ کند.

با این حال، همان‌طور که زمان گذشت، یونیکس به تدریج جای خود را در دنیای فناوری باز کرد. این سیستم عامل با وجود مقاومت‌های اولیه، به دلیل قدرت، انعطاف‌پذیری و قابلیت حمل بالا، به عنوان یک استاندارد در دنیای سرورها و سیستم‌های بزرگ مطرح شد. حتی امروزه، بسیاری از سیستم‌عامل‌های مدرن، از جمله لینوکس و macOS، از اصول طراحی یونیکس الهام گرفته‌اند و تلاش می‌کنند تا تعادلی بین قدرت و سادگی برقرار کنند.

اتفاقی که در اوایل دهه ۱۹۸۰ رخ داد، نه تنها ظهور یک سیستم عامل جدید بود، بلکه ظهور یک فلسفه جدید در دنیای فناوری بود. یونیکس به ما یادآوری کرد که گاهی اوقات، ایده‌های رادیکال و انقلابی می‌توانند دنیا را تغییر دهند، حتی اگر در ابتدا با مقاومت و شک مواجه شوند. این سیستم عامل نه تنها به عنوان یک ابزار فنی، بلکه به عنوان نمادی از تغییر و پیشرفت، تأثیر عمیقی بر صنعت فناوری اطلاعات گذاشت و راه را برای نوآوری‌های بیشتر در آینده هموار کرد.برای سال‌ها، پیشگامان یونیکس در گمنامی نسبی به سر می‌بردند. ایده‌های رادیکال آن‌ها در

دنیای تجاری و جریان اصلی فناوری حمایت چندانی نداشت. حتی زمانی که یک فرد دلسوز به سخنان مدافعان محلی یونیکس گوش می‌داد، پاسخ معمول این بود: «یونیکس اشکالی ندارد، اما اگر می‌خواهید کار جدی انجام دهید، احتمالاً باید از _____ استفاده کنید.» (جای خالی را با نام سیستم عامل مورد علاقه خود پر کنید.) این پاسخ نشان‌دهنده نگرشی بود که یونیکس را بیشتر به عنوان یک ابزار دانشگاهی یا آزمایشگاهی می‌دید تا یک سیستم عامل مناسب برای محیط‌های تجاری و صنعتی. با این حال، فلسفه‌های سیستم عامل مانند ادیان هستند. وقتی کسی باور کند که حقیقت را یافته است، به راحتی آن را رها نمی‌کند. به همین دلیل، رهبران و طرفداران یونیکس با جدیت از استانداردها و اصول این سیستم عامل دفاع می‌کردند و باور داشتند که روزی جهان تغییر خواهد کرد و آن‌ها بهشت نرم‌افزاری را که تصور می‌کردند، خواهند دید.

در حالی که دنیای تجاری به ایجاد موانع برای یونیکس ادامه می‌داد، دنیای دانشگاهی با آغوش باز از این سیستم عامل استقبال می‌کرد. دانشگاه‌ها به عنوان مراکز نوآوری و تفکر آزاد، محیطی ایده‌آل برای رشد و گسترش یونیکس بودند. نسلی از جوانان که در خانه‌هایی با تلویزیون‌های رنگی، اجاق‌های مایکروویو و بازی‌های ویدیویی بزرگ شده بودند، وارد دانشگاه‌هایی می‌شدند که یونیکس را به قیمت رسانه‌های مغناطیسی که روی آن توزیع می‌شد، دریافت کرده بودند. این جوانان، با ذهن‌هایی پاک و آماده برای یادگیری، به بوم‌های تمیزی می‌مانستند که استادان بیش از حد مشتاق بودند تصویری جدید و متفاوت از محاسبات را روی آن‌ها ترسیم کنند. این تصویر، برخلاف جریان اصلی فناوری که بر سیستم‌های اختصاصی و بسته متمرکز بود، بر قدرت، انعطاف‌پذیری و آزادی یونیکس تأکید داشت.امروزه، یونیکس در شرایطی قرار دارد که زمانی غیرقابل تصور بود. این سیستم عامل به سرعت در حال پذیرش در دنیای دانشگاهی، نظامی و حتی تجاری است. یونیکس دیگر تنها یک ابزار دانشگاهی یا آزمایشگاهی نیست، بلکه به عنوان سیستم عاملی قدرتمند و قابل اعتماد در بسیاری از صنایع مورد استفاده قرار می‌گیرد. در دنیای دانشگاهی، یونیکس به انتخاب بلامنازع تبدیل شده است و کاربردهای آن در دنیای نظامی و تجاری هر روز در حال گسترش است. این سیستم عامل که زمانی به عنوان یک اسباب‌بازی برای هکرها و متخصصان دیده می‌شد، اکنون به یکی از پایه‌های اصلی دنیای فناوری اطلاعات تبدیل شده است.

من سال‌هاست که به مردم می‌گویم که تبدیل شدن یونیکس به سیستم عامل جهان فقط مسئله زمان است. با گذشت زمان، این پیش‌بینی به واقعیت نزدیک‌تر می‌شود، اگرچه هنوز به طور کامل محقق نشده است. اما از قضا، سیستم عاملی که در نهایت جهان را فتح خواهد کرد، ممکن است نام «یونیکس» را نداشته باشد. شرکت‌ها به ارزش این نام پی برده‌اند، و وکلای آن‌ها برای ثبت علامت تجاری آن عجله کرده‌اند. در نتیجه، ما شاهد ظهور رابط‌های جدید، استانداردهای پیشنهادی و برنامه‌های کاربردی متعددی هستیم که همگی تحت عنوان «سیستم‌های باز» معرفی می‌شوند. اما در پشت این نام‌ها و رابط‌ها، فلسفه یونیکس همچنان نیروی محرکه اصلی است. فلسفه یونیکس، با تأکید بر سادگی، ماژولاریته و قدرت، به عنوان الهام‌بخش بسیاری از سیستم‌عامل‌ها و فناوری‌های مدرن عمل می‌کند. حتی اگر نام یونیکس به مرور زمان کمرنگ شود، اصول و ارزش‌های آن همچنان به شکل‌های مختلف در دنیای فناوری زنده خواهند ماند. این فلسفه، نه تنها به عنوان یک روش طراحی سیستم‌عامل، بلکه به عنوان یک دیدگاه کلی در مورد نحوه تعامل انسان با ماشین، تأثیر عمیقی بر صنعت فناوری اطلاعات گذاشته است. بنابراین، حتی اگر سیستم عامل جهان «یونیکس» نامیده نشود، مطمئن باشید که فلسفه یونیکس نیروی محرکه پشت آن خواهد بود.

فصل یکم

برای توسعه سیستم‌های ارتباطی مدرن هموار کرد. این مشارکت، نشان‌دهنده اهمیت یونیکس در ایجاد ابزارهایی بود که نیازهای کاربران را به طور مستقیم برطرف می‌کردند.

- **کنت آرنولد** با کار بر روی **به‌روزرسانی صفحه نمایش**، تجربه کاربری را در یونیکس بهبود بخشید. این قابلیت به کاربران اجازه می‌داد تا تغییرات را به صورت بلادرنگ روی صفحه نمایش مشاهده کنند، که برای برنامه‌نویسان و کاربران تعاملی بسیار مفید بود. این بهبودها، یونیکس را به سیستمی کاربرپسندتر تبدیل کرد.
- **استفان بورن** با توسعه **مترجم فرمان پوسته بورن** (Bourne Shell)، یکی از مهم‌ترین ابزارهای یونیکس را ایجاد کرد. پوسته بورن به کاربران اجازه می‌داد تا دستورات را به صورت تعاملی اجرا کنند و اسکریپت‌های قدرتمندی بنویسند. این پوسته به یکی از استانداردهای دنیای یونیکس تبدیل شد و هنوز هم به طور گسترده مورد استفاده قرار می‌گیرد.
- **لورینزا گیلاس** با ایجاد **ماشین حساب تعاملی**، ابزاری ساده اما قدرتمند را به یونیکس اضافه کرد. این ماشین حساب به کاربران اجازه می‌داد تا محاسبات ریاضی را به سرعت و به راحتی انجام دهند. این ابزار، نمونه‌ای از فلسفه یونیکس بود که بر ایجاد ابزارهای کوچک و مستقل تأکید داشت.
- **استیون فلدمن** با مشارکت در **مهندسی نرم‌افزار به کمک کامپیوتر**، روش‌های جدیدی را برای توسعه و مدیریت نرم‌افزارها معرفی کرد. این مشارکت‌ها به توسعه‌دهندگان کمک کرد تا نرم‌افزارهای پیچیده‌تر و قابل اعتمادتری ایجاد کنند.
- **استیون جانسون** با توسعه **ابزار طراحی کامپایلر**، به برنامه‌نویسان این امکان را داد تا کامپایلرهای جدیدی برای زبان‌های برنامه‌نویسی مختلف ایجاد کنند. این ابزارها، نقش مهمی در گسترش زبان‌های برنامه‌نویسی و توسعه نرم‌افزارها ایفا کردند.
- **ویلیام جوی** با ایجاد **ویرایشگر متن** و توسعه **زبان دستوری شبیه به C** ، ابزارهایی را ارائه داد که به برنامه‌نویسان اجازه می‌دادند به راحتی کدهای خود را ویرایش و مدیریت کنند. ویرایشگر متن او، به نام **vi**، هنوز هم یکی از محبوب‌ترین ویرایشگرهای متن در دنیای یونیکس و لینوکس است.
- **برایان کرنیگان**، یکی از تأثیرگذارترین افراد در دنیای یونیکس، با کار بر روی **عبارات منظم**، **اصول برنامه‌نویسی**، **حروف‌چینی** و **آموزش به کمک کامپیوتر**، نقش مهمی در شکل‌گیری فلسفه یونیکس ایفا کرد. کتاب‌های او، مانند"The C Programming Language" ، به عنوان مرجع استاندارد برای برنامه‌نویسان در سراسر جهان شناخته می‌شوند.
- **مایکل لسک** با توسعه **قالب‌بندی متن سطح بالا** و **شبکه تلفنی**، ابزارهایی را ایجاد کرد که به کاربران اجازه می‌دادند متن‌های خود را به صورت حرفه‌ای قالب‌بندی کنند و از طریق شبکه‌های تلفنی ارتباط برقرار نمایند. این ابزارها، به ویژه در محیط‌های اداری و دانشگاهی، بسیار مفید بودند.
- **جان مشی** با ایجاد **مترجم فرمان**، ابزاری را ارائه داد که به کاربران اجازه می‌داد دستورات خود را به صورت تعاملی اجرا کنند. این ابزار، تجربه کاربری را در یونیکس بهبود بخشید و به یکی از ویژگی‌های کلیدی این سیستم عامل تبدیل شد.

زمانی که لینوکس توسط لینوس تورووالدز خلق شد، ایده قرض گرفتن نرم‌افزار نوشته‌شده توسط دیگران به‌طور فزاینده‌ای رایج شده بود. این ایده تا حدی پیشرفت کرده بود که ریچارد ام. استالمن آن را در مجوز عمومی خود به نام GNU GPL (General Public License) رسمیت بخشید .یک توافقنامه قانونی است که GPL برای نرم‌افزار اعمال می‌شود و به‌طور موثر تضمین می‌کند که کد منبع نرم‌افزار برای هر کسی که بخواهد از آن استفاده کند، آزادانه در دسترس باشد. تورووالدز در نهایت این مجوز را برای لینوکس پذیرفت و با این کار، امکان قرض گرفتن کد منبع لینوکس را برای هر کسی بدون ترس از درگیری‌های قانونی مرتبط با حق چاپ فراهم کرد. این تصمیم نه‌تنها به رشد سریع لینوکس کمک کرد، بلکه آن را به یکی از بزرگ‌ترین نمونه‌های موفق نرم‌افزار آزاد و متن‌باز تبدیل کرد.

لینوکس از همان روزهای اولیه نشان داد که یک سیستم‌عامل شبیه به یونیکس است. توسعه‌دهندگان آن اصول فلسفه یونیکس را با جان و دل پذیرفتند و سپس به نوشتن یک سیستم‌عامل جدید از ابتدا پرداختند. این فلسفه شامل اصولی مانند سادگی، ماژولار بودن، و استفاده مجدد از کدها بود. با این حال، نکته مهم این است که تقریباً هیچ چیز در دنیای لینوکس دیگر از ابتدا نوشته نمی‌شود. تقریباً همه چیز بر پایه کدها و مفاهیمی ساخته شده است که توسط دیگران نوشته شده‌اند. این رویکرد باعث شده است که لینوکس به گام بعدی طبیعی در تکامل یونیکس تبدیل شود، یا شاید دقیق‌تر بگوییم، جهش بزرگ بعدی برای یونیکس باشد. لینوکس نه‌تنها اصول یونیکس را حفظ کرد، بلکه آن‌ها را به سطح جدیدی از انعطاف‌پذیری و قدرت رساند.لینوکس، مانند یونیکس، سهم زیادی از توسعه‌دهندگان را در مراحل اولیه توسعه فنی خود داشت. با این حال، در حالی که تعداد توسعه‌دهندگان یونیکس در اوج خود به هزاران نفر می‌رسید، تعداد توسعه‌دهندگان لینوکس امروزه به میلیون‌ها نفر می‌رسد. این رشد عظیم تا حد زیادی به دلیل ماهیت متن‌باز و آزاد لینوکس است که به هر کسی اجازه می‌دهد در توسعه آن مشارکت کند. این مشارکت‌ها از نوشتن کدهای جدید گرفته تا گزارش باگ‌ها، بهبود مستندات، و حتی ترجمه نرم‌افزار به زبان‌های مختلف را شامل می‌شود. این جامعه گسترده و متنوع از توسعه‌دهندگان، لینوکس را به یک پلتفرم بسیار قدرتمند و انعطاف‌پذیر تبدیل کرده است که در طیف وسیعی از دستگاه‌ها، از سرورهای بزرگ گرفته تا تلفن‌های همراه، استفاده می‌شود.

یکی از دلایل اصلی موفقیت لینوکس، فلسفه‌ای است که پشت آن قرار دارد: فلسفه نرم‌افزار آزاد. این فلسفه نه‌تنها به کاربران آزادی استفاده از نرم‌افزار را می‌دهد، بلکه به آن‌ها اجازه می‌دهد تا نرم‌افزار را تغییر دهند و بهبود بخشند. این آزادی باعث شده است که لینوکس به یک پلتفرم بسیار قابل تنظیم و سازگار تبدیل شود که می‌تواند نیازهای مختلف کاربران را برآورده کند. علاوه بر این، این فلسفه باعث ایجاد یک جامعه قوی و متعهد شده است که به طور مداوم در حال بهبود و گسترش لینوکس است.

لینوکس همچنین به دلیل پایداری و امنیت بالای خود شناخته شده است. این سیستم‌عامل به‌طور گسترده‌ای در محیط‌های حساس مانند سرورهای اینترنتی، سیستم‌های بانکی، و حتی در ایستگاه‌های فضایی استفاده می‌شود. این پایداری و امنیت تا حد زیادی به دلیل ماهیت متن‌باز لینوکس است که به توسعه‌دهندگان اجازه می‌دهد به‌طور مداوم کدها را بررسی و بهبود بخشند. این فرآیند بررسی مداوم باعث می‌شود که باگ‌ها و آسیب‌پذیری‌ها به سرعت شناسایی و برطرف شوند.

یکی از دلایل اصلی موفقیت لینوکس، انعطاف‌پذیری و سازگاری آن با نیازهای مختلف است. کاربران می‌توانند لینوکس را به‌راحتی سفارشی‌سازی کنند و آن را مطابق با نیازهای خاص خود تنظیم کنند. این ویژگی باعث شده است که لینوکس در محیط‌های مختلف، از سیستم‌های شخصی گرفته تا شبکه‌های بزرگ سازمانی، مورد استفاده قرار گیرد. علاوه بر این، لینوکس به‌دلیل متن‌باز بودن، از امنیت بالاتری برخوردار است، زیرا هزاران توسعه‌دهنده در سراسر جهان به‌طور مداوم کد آن را بررسی و بهبود می‌بخشند.

اصول فلسفه یونیکس

فلسفه یونیکس بر پایه اصولی ساده اما عمیق بنا شده است که اگر به درستی اجرا شوند، می‌توانند به ایجاد سیستم‌ها و نرم‌افزارهایی کارآمد، انعطاف‌پذیر و قابل اعتماد منجر شوند. این اصول به‌ظاهر ساده، اما در عمل بسیار قدرتمند، به‌عنوان پایه‌ای برای طراحی سیستم‌عامل‌ها و نرم‌افزارها در طول دهه‌ها مورد استفاده قرار گرفته‌اند. در ادامه، این اصول را به تفصیل بررسی می‌کنیم تا درک بهتری از اهمیت و تأثیر آنها به دست آوریم.

۱. کوچک زیبا

یکی از اصول کلیدی فلسفه یونیکس این است که "کوچک زیبا است". این اصل بر این باور استوار است که برنامه‌ها و ابزارهای کوچک می‌توانند مزایای قابل توجهی نسبت به همتایان بزرگتر خود داشته باشند. برنامه‌های کوچک نه‌تنها ساده‌تر و قابل درک‌تر هستند، بلکه انعطاف‌پذیری بیشتری نیز دارند. این انعطاف‌پذیری به کاربران و توسعه‌دهندگان اجازه می‌دهد تا این ابزارها را به روش‌های خلاقانه و منحصر به فردی ترکیب کنند، حتی اگر این روش‌ها توسط طراح اصلی پیش‌بینی نشده باشند. این ویژگی باعث می‌شود که ابزارهای کوچک به‌عنوان بلوک‌های ساختمانی قدرتمندی عمل کنند که می‌توانند برای حل مشکلات پیچیده‌تر مورد استفاده قرار گیرند.

۲. هر برنامه یک کار را به خوبی انجام دهد

اصل دوم فلسفه یونیکس این است که هر برنامه باید یک کار را به خوبی انجام دهد. این اصل بر تمرکز و سادگی تأکید دارد. با محدود کردن عملکرد یک برنامه به یک کار واحد، می‌توان از پیچیدگی‌های غیرضروری جلوگیری کرد و کدهای اضافی را حذف نمود. این رویکرد نه‌تنها باعث کاهش سربار و افزایش کارایی می‌شود، بلکه انعطاف‌پذیری برنامه را نیز افزایش می‌دهد. برنامه‌هایی که یک کار را به خوبی انجام می‌دهند، می‌توانند به‌راحتی با سایر ابزارها ترکیب شوند و به عنوان بخشی از یک سیستم بزرگتر عمل کنند.

۳. نمونه‌سازی سریع

نمونه‌سازی سریع یکی دیگر از اصول مهم فلسفه یونیکس است. در بسیاری از روش‌های طراحی نرم‌افزار، نمونه‌سازی تنها بخش کوچکی از فرآیند طراحی محسوب می‌شود. اما در فلسفه یونیکس، نمونه‌سازی به‌عنوان وسیله‌ای اصلی برای ایجاد طراحی‌های مؤثر مورد استفاده قرار می‌گیرد. با ایجاد سریع نمونه‌های اولیه،

فصل ۲

قدم کوچک برای بشر

در همان زمان که تمایل به استفاده از اتومبیل‌های کوچک در آمریکا شروع به شکوفا شدن کرد، گروهی از محققان در آزمایشگاه AT&T Bell در نیوجرسی به نکته‌ای مهم در دنیای نرم‌افزار پی بردند. آنها متوجه شدند که همانطور که اتومبیل‌های کوچک می‌توانند در شرایط خاص عملکرد بهتری نسبت به اتومبیل‌های بزرگ داشته باشند، برنامه‌های نرم‌افزاری کوچک نیز از مزایای قابل توجهی برخوردارند. این برنامه‌ها نسبت به برنامه‌های بزرگتر سریع‌تر اجرا می‌شوند، نگهداری آنها آسان‌تر است، و به‌ویژه قابلیت سازگاری بالاتری دارند. این ویژگی‌ها به آن‌ها این امکان را می‌دهد که در محیط‌های مختلف به‌راحتی مورد استفاده قرار گیرند و سریع‌تر توسعه یابند. این ایده که برنامه‌های کوچک، ساده و ماژولار بر برنامه‌های پیچیده و بزرگتر ارجحیت دارند، به‌طور کلی به فلسفه‌ای به نام «فلسفه یونیکس» منجر شد. یکی از اصول اصلی این فلسفه این است که «هر برنامه باید یک کار خاص را به‌خوبی انجام دهد». این جمله که به‌عنوان یکی از ارکان مهم یونیکس شناخته می‌شود، به این معناست که برنامه‌های یونیکس به‌طور کلی باید هرکدام به‌طور اختصاصی و مختصر به یک وظیفه خاص بپردازند. به‌عنوان مثال، بسیاری از ابزارها و دستورات یونیکس برای انجام کارهایی خاص طراحی شده‌اند که می‌توانند به راحتی با سایر برنامه‌ها و ابزارهای یونیکس ارتباط برقرار کنند.

اصل کوچک زیباست

این مثال ساده از یک برنامه کپی فایل، به خوبی نشان می‌دهد که چگونه حتی یک وظیفه به ظاهر ساده می‌تواند به مراحل متعدد و پیچیده‌ای تقسیم شود. با این حال، فلسفه یونیکس به ما یادآوری می‌کند که **سادگی** و **کوچک نگه داشتن برنامه‌ها** کلید موفقیت هستند. در ادامه، این ایده را بیشتر بررسی می‌کنیم و نشان می‌دهیم که چگونه می‌توان با تمرکز بر کوچک‌ترین نرم‌افزار عملی، به راه‌حل‌های کارآمد و مؤثر دست یافت. اصل «کوچک زیباست (Small is Beautiful)» که یکی از ارکان فلسفه یونیکس است، بر اهمیت طراحی برنامه‌های کوچک و کارآمد تأکید دارد. این اصل به‌ویژه در دنیای نرم‌افزار بسیار پررنگ است و نشان می‌دهد که حتی در مواجهه با وظایف پیچیده، بهترین راه‌حل‌ها اغلب از ترکیب برنامه‌های کوچک، ساده و تخصصی به دست می‌آید. به عبارت دیگر، «هر برنامه باید یک کار خاص را به‌خوبی انجام دهد» و این کار باید به‌صورت کاملاً مؤثر و بدون پیچیدگی‌های غیرضروری انجام شود.

برای روشن‌تر شدن این مفهوم، مثال ساده‌ای از یک برنامه برای کپی کردن فایل‌ها می‌توان زد. در دنیای نرم‌افزارهای پیچیده، ممکن است یک برنامه کپی فایل تنها یک وظیفه ساده مانند کپی کردن داده‌ها از یک محل به محل دیگر انجام دهد، اما در یونیکس این فرآیند به یک وظیفه کوچک و تخصصی تقسیم می‌شود. به‌عنوان مثال، ممکن است چندین برنامه کوچک و تخصصی برای انجام مراحل مختلف این فرآیند در نظر گرفته شود:

۱. برنامه‌ای برای انتخاب فایل‌های مبدا.
۲. برنامه‌ای برای بررسی صحت فایل‌ها و اطمینان از نبود خطا.
۳. برنامه‌ای برای نوشتن فایل‌ها در مقصد.

۲. **کپی داده‌ها** :به جای بررسی وجود فایل‌ها و پرسیدن سؤالات اضافی، می‌توان به سادگی عملیات کپی را انجام داد. اگر فایل منبع وجود نداشته باشد یا فایل مقصد قابل نوشتن نباشد، برنامه می‌تواند یک پیام خطای ساده نمایش دهد.
۳. **بستن فایل‌ها** :پس از انجام عملیات کپی، فایل‌ها باید بسته شوند. این مرحله نیز می‌تواند به سادگی انجام شود.

با این ساده‌سازی، برنامه کپی فایل نه تنها کوچک‌تر و سریع‌تر می‌شود، بلکه نگهداری و توسعه آن نیز آسان‌تر خواهد بود.

تئودور استورجن وقانون ۹۰ درصد

تئودور استورجن، نویسنده علمی‌تخیلی، زمانی نوشت که **۹۰درصد داستان‌های علمی‌تخیلی خام است** . همین امر در مورد اکثر نرم‌افزارهای سنتی نیز صدق می‌کند. بخش بزرگی از کد در هر برنامه‌ای به چیزی غیر از انجام واقعی وظیفه بیان شده اختصاص داده شده است. این کد اضافی نه تنها برنامه را پیچیده‌تر می‌کند، بلکه احتمال بروز خطاها را نیز افزایش می‌دهد. فلسفه یونیکس بر این ایده استوار است که برنامه‌ها باید کوچک، ساده و متمرکز بر انجام یک کار خاص باشند. این اصل نه تنها باعث افزایش کارایی و قابلیت اطمینان برنامه‌ها می‌شود، بلکه به کاربران اجازه می‌دهد تا با ترکیب این برنامه‌های کوچک، کارهای بزرگ و پیچیده‌تری را انجام دهند. در این متن، به بررسی این ایده می‌پردازیم و نشان می‌دهیم که چگونه برنامه‌های کوچک و تخصصی می‌توانند با همکاری یکدیگر، سیستم‌های قدرتمند و انعطاف‌پذیری ایجاد کنند .با وجود اینکه قانون استرجن می‌تواند به عنوان یک اصل کلی مفید عمل کند، لازم است به این نکته توجه شود که این قانون یک ارزیابی عمومی است و ممکن است در همه شرایط و موارد صدق نکند. در ادامه، چند نکته کلیدی در مورد این قانون ارائه می‌شود که هنگام استفاده از آن باید مدنظر قرار گیرد:

۱. **ذهنیت مفهوم «بی‌ارزش» یا «چرند»** :«تعریف چیزی به عنوان «بی‌ارزش» یا «چرند» اغلب ذهنی است و می‌تواند مورد بحث قرار گیرد. در بسیاری از موارد، تشخیص اینکه چه چیزی باید بی‌ارزش تلقی شود، دشوار است. این موضوع به ویژه زمانی چالش‌برانگیز می‌شود که افراد در مورد ارزش‌گذاری پدیده‌ها اختلاف نظر داشته باشند. برای مثال، اگر دو نفر به ژانرهای ادبی متفاوتی علاقه‌مند باشند، ممکن است در مورد اینکه کدام کتاب‌ها بی‌ارزش هستند و کدام‌ها ارزشمند، با هم توافق نداشته باشند.
۲. **تخمین ۹۰ درصدی** :عدد ۹۰ درصد صرفاً یک تخمین تقریبی است و به معنای دقیق و علمی نیست. این عدد بیشتر برای بیان مفهوم «اکثریت قریب به اتفاق» استفاده می‌شود و لزوماً در همه شرایط قابل تعمیم نیست. برای مثال، ممکن است در برخی حوزه‌ها تنها ۸۰ درصد آثار بی‌کیفیت باشند، در حالی که در حوزه‌های دیگر این عدد به ۹۵ درصد برسد.
۳. **عدم کاربرد جهانی قانون استرجن** :این قانون در همه شرایط و حوزه‌ها قابل اعمال نیست. به عنوان مثال، در حوزه‌هایی که ورود به آن‌ها دشوار است یا استانداردهای بالایی دارند، ممکن است اکثر آثار تولیدی از کیفیت قابل قبولی برخوردار باشند. برای نمونه، اگر یک سازمان اطلاعاتی ۹۰ درصد

ممکن تمرکز می‌کنند. این رویکرد باعث می‌شود که برنامه‌ها ساده‌تر، قابل درک‌تر و نگهداری‌پذیرتر باشند. علاوه بر این، برنامه‌های کوچک به راحتی می‌توانند با برنامه‌های دیگر ترکیب شوند تا کارهای پیچیده‌تری را انجام دهند. این ویژگی باعث می‌شود که برنامه‌های کوچک نه تنها برای انسان‌ها، بلکه برای ماشین‌ها نیز سازگارتر باشند.یکی از مزایای کلیدی برنامه‌های کوچک این است که آن‌ها را برای مقابله با موقعیت‌هایی که در زمان نوشتن برنامه قابل پیش‌بینی نبوده‌اند، مجهز می‌کنند. به عنوان مثال، اگر یک فرمت داده جدید ظهور کند، یک برنامه کوچک که تنها بر یک فرمت داده خاص متمرکز است، می‌تواند به راحتی با برنامه‌های دیگر ترکیب شود تا با فرمت جدید کار کند. در حالی که یک برنامه بزرگ و پیچیده ممکن است نیاز به بازنویسی گسترده داشته باشد تا بتواند با فرمت جدید سازگار شود. برنامه‌های کوچک و متمرکز بر یک کار خاص، مزایای متمایزی نسبت به برنامه‌های بزرگ و پیچیده دارند. سادگی آن‌ها نوشتن، درک و نگهداری آن‌ها را آسان‌تر می‌کند. علاوه بر این، برنامه‌های کوچک انعطاف‌پذیری و سازگاری بیشتری دارند، که این امر باعث می‌شود که آن‌ها بهتر بتوانند با تغییرات آینده مقابله کنند. در نهایت، با نوشتن برنامه‌های کوچک به جای برنامه‌های بزرگ، شما نه تنها کارایی سیستم را بهبود می‌بخشید، بلکه برنامه‌های خود را برای مقابله با موقعیت‌هایی که نمی‌توانستید پیش‌بینی کنید، مجهز می‌کنید. این رویکرد نه تنها باعث افزایش کارایی می‌شود، بلکه باعث می‌شود که برنامه‌ها به راحتی قابل نگهداری و توسعه باشند.

نگاه کردن به یک باگ

بیایید لحظه‌ای درباره یک "باگ" صحبت کنیم – نه آن نوع باگ‌های نرم‌افزاری که برنامه‌نویسان با آن‌ها دست و پنجه نرم می‌کنند، بلکه یک باگ واقعی، یک حشره عجیب و غریب که در طبیعت زندگی می‌کند. در منطقه اطراف دریاچه ویکتوریا، که منبع رودخانه نیل است، گونه‌ای عجیب از حشره پرنده به نام "مگس دریاچه" زندگی می‌کند. این حشره که به دلیل رفتار و چرخه زندگی منحصر به فردش شناخته شده است، موضوع تحقیقات و فیلم‌های مستند کاوشگر مشهور، ژاک کوستو، بوده است. کوستو در فیلم‌های خود نشان می‌دهد که این حشرات در توده‌های غلیظ و مه‌مانند روی سطح دریاچه و جنگل‌های اطراف تجمع می‌کنند. این تجمع‌ها گاهی چنان متراکم و وسیع هستند که می‌توان آن‌ها را با پدیده‌های طبیعی مانند فواره‌های آب یا گردبادهای کوچک اشتباه گرفت.

ژاک کوستو در بررسی‌های خود متوجه شد که چرخه زندگی این حشرات بسیار کوتاه و غیرمعمول است. مگس‌های دریاچه تنها برای مدت بسیار کوتاهی، حدود ۶ تا ۱۲ ساعت، به عنوان حشره بالغ زندگی می‌کنند. حتی اگر آن‌ها توسط پرندگان شکار نشوند، عمر بزرگسالی آن‌ها چیزی بیش از یک بال زدن کوتاه در زیر نور خورشید نیست. این کوتاهی عمر سوال جالبی را مطرح می‌کند: یک حشره که تمام دوران بزرگسالی خود را در کمتر از یک روز می‌گذراند، چگونه از این زمان محدود استفاده می‌کند؟ پاسخ ساده است: این حشره تمام انرژی خود را صرف تولید مثل و بقای گونه خود می‌کند. در واقع، مگس‌های دریاچه تلاش می‌کنند در چند ساعت کاری را انجام دهند که بسیاری از موجودات دیگر، از جمله انسان‌ها، سال‌ها برای آن زمان صرف می‌کنند.

این حشرات کوچک اولویت‌های خود را به وضوح مشخص کرده‌اند: بقای گونه. آن‌ها تنها یک ماموریت در زندگی دارند و آن را به بهترین شکل ممکن انجام می‌دهند. این رویکرد متمرکز و ساده به بقا، الهام‌بخش توسعه‌دهندگان

نقش بازخورد در فرهنگ یونیکس

بازخورد کاربران نقش بسیار مهمی در توسعه نرم‌افزار در محیط یونیکس ایفا می‌کند. این بازخورد می‌تواند به برنامه‌نویس کمک کند تا بفهمد کدام بخش‌های سیستم نیاز به بهبود دارند و کدام ویژگی‌ها باید اضافه یا حذف شوند. در حقیقت، فرهنگ یونیکس بر این باور استوار است که بهترین طراحی‌ها از طریق آزمایش و بازخورد کاربران به وجود می‌آیند، نه از طریق تدوین مستندات طولانی و پیچیده.در رویکرد یونیکس، تمرکز بر تجربه عملی و تعامل با سیستم در مراحل اولیه است. این رویکرد بر خلاف روش‌های سنتی مهندسی نرم‌افزار که معمولاً در ابتدای پروژه طراحی‌های دقیقی انجام می‌دهند، اولویت را به ایجاد یک سیستم اولیه می‌دهد که قابل مشاهده و قابل لمس برای کاربر است. این سیستم اولیه نه تنها به کاربر این امکان را می‌دهد که عملکرد محصول را به طور ملموس مشاهده کند، بلکه احساساتی درباره کارایی، راحتی و جذابیت آن نیز در ذهن او شکل می‌گیرد. این ارتباط اولیه به کاربر کمک می‌کند تا درک بهتری از قابلیت‌ها و محدودیت‌های سیستم پیدا کند و در صورت لزوم بتواند به سرعت تغییرات کوچک و مؤثری را اعمال کند. در حالی که تغییرات در سیستم‌های اولیه به راحتی انجام می‌شود، تغییرات عمده در مراحل بعدی بسیار دشوارتر و زمان‌بر هستند. بنابراین، این مرحله ابتدایی یک گام مهم در فرآیند توسعه است که از سوی یونیکس به عنوان یک شروع سریع و مؤثر در نظر گرفته می‌شود.

ویژگی مهم سیستم اول این است که این سیستم نه تنها مفهومی را به نمایش می‌گذارد بلکه تخیل و ایده‌های جدید را در ذهن کاربر ایجاد می‌کند. به عبارت دیگر، مشاهده سیستم اولیه می‌تواند جرقه‌ای از خلاقیت را در ذهن کاربر روشن کند. کاربر ممکن است شروع به تصور کاربردهای جدیدی کند که در ابتدا توسط طراحان پیش‌بینی نشده‌اند. این فرآیند، یک چرخه خودتقویت‌شونده است، زیرا ایده‌های جدید به بهبود سیستم کمک می‌کنند و توسعه‌دهندگان متوجه می‌شوند که در حال ساخت چیزی فراتر از پیش‌بینی‌های اولیه خود هستند. این جرقه‌های خلاقانه می‌توانند به توسعه محصول به شکلی کاملاً جدید و نوآورانه منجر شوند.در این مرحله، رویکرد یونیکس نسبت به روش‌های مهندسی سنتی یک جهش محسوب می‌شود. در حالی که در روش‌های سنتی، کاربر بیشتر به دنبال مشاهده ظاهری و ویژگی‌های اولیه سیستم است، توسعه‌دهندگان یونیکس در همان لحظات اولیه بر نحوه استفاده از نمونه اولیه و ارزیابی عملکرد آن تمرکز می‌کنند. این رویکرد به نوعی پیش‌بینی‌گرانه است و به کاربر و توسعه‌دهنده این امکان را می‌دهد که از ابتدا به تجربه عملی سیستم بپردازند. به همین دلیل، فرآیند طراحی در یونیکس، برخلاف روش‌های سنتی که بیشتر به دنبال مستندسازی و مشخصات دقیق است، به طور مداوم و تکراری پیش می‌رود. کاربران و توسعه‌دهندگان در این فرآیند مشترکاً به سیستم نگاه می‌کنند و می‌توانند به طور مستمر نیازهای جدیدی را شناسایی و آن‌ها را به طراحی وارد کنند.

پس از این مرحله، فرآیند طراحی تکراری آغاز می‌شود. کاربران بازخوردهای اولیه خود را ارائه می‌دهند، و این بازخوردها به توسعه‌دهندگان کمک می‌کند تا ارزیابی کنند که آیا در مسیر درستی هستند یا خیر. در این مرحله، به طور معمول طراحی اولیه به تدریج به سمت تکامل پیش می‌رود و ممکن است تغییرات عمده‌ای در آن اعمال شود. اگر کاربر احساس کند که سیستم به طور کامل مطابق با نیازهای او نیست، ممکن است درخواست‌های طراحی مجدد کامل بدهد. اما در اکثر موارد، کاربران ممکن است فقط بخشی از سیستم را نپسندند و پیشنهاداتی برای بهبود آن ارائه دهند. این بازخوردها به همکاری مستمر میان کاربر و توسعه‌دهندگان منجر می‌شود که در نهایت

فصل ۵

جنبش نرم افزار متن باز

یونیکس، به عنوان یکی از قدرتمندترین و تأثیرگذارترین سیستم‌عامل‌های تاریخ کامپیوتر، موفقیت خود را مدیون فلسفه‌ای است که در طراحی و توسعه آن به کار گرفته شده است. این موفقیت تصادفی نبود، بلکه نتیجه‌ی تلاش‌های هماهنگ و مشارکتی ده‌ها و سپس صدها برنامه‌نویس بود که به‌طور جمعی به ایجاد سیستمی پرداختند که نه‌تنها انعطاف‌پذیر و قدرتمند بود، بلکه توانایی اهرم‌کردن تلاش‌های فردی را نیز داشت. این ایده‌ی اهرم‌کردن تلاش‌ها، یکی از کلیدی‌ترین عوامل موفقیت یونیکس محسوب می‌شود. در اوایل توسعه‌ی یونیکس، برنامه‌نویسان متوجه شدند که کارهای زیادی وجود دارد که به‌تنهایی نمی‌توانند انجام دهند. محدودیت‌های زمانی، منابع و دانش فنی باعث می‌شد که پیشرفت پروژه‌ها کند شود. اما به‌جای اینکه هر فرد به‌صورت جداگانه بر روی مشکلات مشابه کار کند، آن‌ها شروع به همکاری و اشتراک‌گذاری کدها، ابزارها و ایده‌ها کردند. این همکاری نه‌تنها باعث صرفه‌جویی در زمان و انرژی شد، بلکه اثرات کار هر فرد را چندین برابر کرد. به عبارت دیگر، آن‌ها از اهرم نرم‌افزاری استفاده کردند تا کارایی خود را افزایش دهند.

اهرم نرم‌افزاری به این معناست که با استفاده از ابزارها، کتابخانه‌ها و کدهای از پیش نوشته‌شده، برنامه‌نویسان می‌توانند کارهای پیچیده‌تر را با تلاش کم‌تری انجام دهند. به‌عنوان مثال، اگر یک برنامه‌نویس نیاز به نوشتن یک برنامه‌ی خاص داشت، به‌جای اینکه از صفر شروع کند، می‌توانست از کدها یا ابزارهایی که دیگران قبلاً نوشته‌اند استفاده کند. این کار نه‌تنها زمان توسعه را کاهش می‌داد، بلکه کیفیت نرم‌افزار را نیز افزایش می‌داد، زیرا کدهای استفاده‌شده قبلاً تست و بهبود یافته بودند.یکی از ویژگی‌های کلیدی یونیکس که به این اهرم‌کردن کمک کرد، طراحی ماژولار آن بود. یونیکس از تعداد زیادی ابزار کوچک و مستقل تشکیل شده است که هر کدام وظیفه‌ی خاصی را انجام می‌دهند. این ابزارها می‌توانند به‌راحتی با هم ترکیب شوند تا کارهای پیچیده‌تری را انجام دهند. این طراحی ماژولار به برنامه‌نویسان اجازه می‌داد تا از ابزارهای موجود استفاده کنند و آن‌ها را به روش‌های جدیدی ترکیب کنند، بدون اینکه نیاز باشد همه‌چیز را از ابتدا بنویسند.

علاوه بر این، فرهنگ اشتراک‌گذاری و همکاری که در جامعه‌ی یونیکس وجود داشت، نقش مهمی در موفقیت آن ایفا کرد. برنامه‌نویسان نه‌تنها کدهای خود را با دیگران به اشتراک می‌گذاشتند، بلکه مشکلات و چالش‌های خود را نیز در میان می‌گذاشتند تا دیگران بتوانند راه‌حل‌هایی ارائه دهند. این فرهنگ باز و مشارکتی باعث شد که یونیکس به‌سرعت پیشرفت کند و به یک سیستم‌عامل قدرتمند و انعطاف‌پذیر تبدیل شود.یکی دیگر از عوامل موفقیت یونیکس، سادگی و زیبایی طراحی آن بود. طراحان یونیکس همواره به دنبال ایجاد ابزارهایی بودند که کارایی بالایی داشته باشند و در عین حال ساده و قابل فهم باشند. این سادگی باعث می‌شد که برنامه‌نویسان جدید به‌راحتی بتوانند با سیستم کار کنند و در توسعه‌ی آن مشارکت کنند. همچنین، این سادگی باعث می‌شد که ابزارهای یونیکس به‌راحتی قابل ترکیب و استفاده‌ی مجدد باشند، که این خود به اهرم‌کردن تلاش‌ها کمک می‌کرد.

این محدودیت‌های انسانی به‌طور واضح بر سیستم‌های کامپیوتری تأثیر می‌گذارند. هر سیستمی که مجبور باشد منتظر ورودی کاربر باشد، نمی‌تواند به سرعت و کارایی مطلوب خود عمل کند. به عبارت دیگر، سیستم‌های کامپیوتری هنگامی که نیاز به تعامل با انسان‌ها دارند، به اندازه کافی سریع عمل نمی‌کنند. در اینجا یک نکته مهم وجود دارد: سیستم‌های یونیکس، برخلاف بسیاری از سیستم‌های دیگر، سعی می‌کنند وظایف خود را تا حد ممکن بدون نیاز به دخالت مستقیم انسان انجام دهند. اکثر دستورات در یونیکس فقط زمانی از کاربر درخواست ورودی می‌کنند که در شرف انجام عملیاتی خطرناک یا غیرقابل برگشت باشند، مانند حذف یا تعمیر فایل‌های سیستمی. به همین دلیل، دستورات یونیکس معمولاً با حداکثر سرعت خود اجرا می‌شوند و نیازی به توقف و انتظار برای ورودی کاربر ندارند. این ویژگی‌ها توضیح می‌دهند که چرا سیستم‌های یونیکس حتی در شرایطی که هدف اصلی آنها افزایش کارایی نباشد، باز هم عملکرد خوبی دارند. این سیستم‌ها به‌خوبی می‌توانند تشخیص دهند که نقطه ضعف اصلی در بسیاری از تعاملات انسان و ماشین، خود انسان‌ها هستند و نه ماشین. بنابراین، اگر سیستم‌های کامپیوتری به‌گونه‌ای طراحی شوند که بیشتر خودکار عمل کنند و کمتر نیاز به ورودی و دخالت مستقیم کاربر داشته باشند، می‌توانند به‌طور مؤثرتری از توان پردازشی خود بهره‌برداری کنند و کارایی بهتری ارائه دهند.

تجزیه کننده های فرمان CUI اغلب بزرگ و زشت هستند

تجزیه‌کننده ورودی کاربر) یا همان (parser مسئول خواندن و پردازش داده‌های ورودی از کاربر است و آن‌ها را به شکلی ترجمه می‌کند که نرم‌افزار کاربردی قادر به درک و پردازش آن‌ها باشد. این فرایند به‌طور چشم‌گیری پیچیده است، زیرا تجزیه‌کننده باید قادر باشد هر ورودی‌ای را که کاربر به‌طور قابل تصور (و حتی غیرقابل تصور!) وارد می‌کند، به درستی شبیه‌سازی کند. این ویژگی، باعث می‌شود که تجزیه‌کننده فرمان معمولی به‌طور قابل توجهی بزرگ و پیچیده شود. در برخی موارد، تجزیه‌کننده دستورالعمل‌ها ممکن است به اندازه‌ای پیچیده باشد که حتی برنامه‌نویسی بیشتر از خود برنامه اصلی نیاز داشته باشد. برای درک بهتر این پیچیدگی، فرض کنید یک برنامه برای فرمت کردن هارد دیسک وجود دارد. از آنجایی که احتمال از دست دادن داده‌ها در این فرآیند بسیار زیاد است، به‌طور معمول از کاربر خواسته می‌شود که تایید کند آیا واقعاً می‌خواهد همه چیز را از روی دیسک پاک کند یا خیر. به‌عنوان مثال:

```
FORMAT V١,٠ Rev.A
```

در حال فرمت کردن درایو:C

اما پیچیدگی زمانی شروع می‌شود که کاربر تصمیم نداشته باشد که به طور قطعی چه اقدامی انجام دهد. یک کاربر مبتدی ممکن است "راهنما" را وارد کند تا اطلاعاتی کلی درباره فرمت کردن دریافت کند. در حالی که یک کاربر با تجربه‌تر ممکن است "?" را وارد کند تا فهرستی از گزینه‌های مختلف فرمت کردن را مشاهده کند. برخی دیگر از کاربران ممکن است سعی کنند از دستور فرمت با استفاده از کاراکترهای خاص یا وقفه‌های سیستم خارج شوند.

محیط لینوکس :استفاده از برنامه ها به عنوان فیلتر

برنامه‌نویسی در لینوکس بر پایه‌ی اصول و قوانینی استوار است که اگرچه ممکن است به صورت رسمی نوشته نشده باشند، اما توسط برنامه‌نویسان به طور گسترده‌ای رعایت می‌شوند. یکی از این اصول، طراحی برنامه‌ها به گونه‌ای است که بتوانند به عنوان فیلتر عمل کنند. اما عملکرد یک برنامه به عنوان فیلتر به چه معناست؟ در اینجا سعی می‌کنیم این مفهوم را با جزئیات بیشتری بررسی کنیم و به برخی از دستورالعمل‌هایی که برنامه‌نویسان لینوکس برای طراحی چنین برنامه‌هایی استفاده می‌کنند، بپردازیم. قبل از ورود به این بحث، لازم است مفهوم مهمی در لینوکس به نام **stdio**را توضیح دهیم.

مفهوم stdio در لینوکس

زمانی که یک برنامه در لینوکس اجرا می‌شود، معمولاً سه کانال استاندارد ورودی/خروجی (I/O) در اختیار دارد که به ترتیب **stdin**، **stdout** و **stderr**نامیده می‌شوند. این سه کانال، پایه‌های ارتباطی برنامه با محیط خارجی را تشکیل می‌دهند **stdin** .(ورودی استاندارد) برای دریافت داده‌ها از کاربر یا منبع دیگری استفاده می‌شود، **stdout**(خروجی استاندارد) برای ارسال خروجی برنامه به مقصدی مانند صفحه نمایش، و **stderr**(خطای استاندارد) برای ارسال پیام‌های خطا یا اطلاعات دیباگ به کار می‌رود. به طور پیش‌فرض، **stdin**ورودی کاربر را از ترمینال دریافت می‌کند، **stdout**خروجی را به ترمینال ارسال می‌کند، و **stderr**نیز پیام‌های خطا را در ترمینال نمایش می‌دهد.

انعطاف‌پذیریstdio

یکی از ویژگی‌های قدرتمند لینوکس این است که این کانال‌های ورودی/خروجی به هیچ دستگاه خاصی "سیم‌کشی" نشده‌اند. این بدان معناست که کاربر می‌تواند در زمان فراخوانی برنامه، مشخص کند که داده‌ها از کجا دریافت شوند یا به کجا ارسال گردند. برای مثال، **stdin**می‌تواند به جای دریافت داده از ترمینال، از یک فایل، خروجی یک برنامه دیگر (از طریق لوله‌های لینوکس)، یا حتی از یک منبع شبکه‌ای مانند یک سرور دور دست تغذیه شود. به طور مشابه، **stdout**می‌تواند به جای نمایش خروجی در ترمینال، به یک فایل هدایت شود تا کاربر بتواند آن را بعداً بررسی کند. این انعطاف‌پذیری به برنامه‌نویسان و کاربران اجازه می‌دهد تا برنامه‌ها را به روش‌های خلاقانه‌ای به هم متصل کنند و کارهای پیچیده را با ترکیب برنامه‌های ساده انجام دهند.

برنامه‌ها به عنوان فیلتر

برنامه‌هایی که به عنوان فیلتر عمل می‌کنند، معمولاً داده‌ها را از **stdin**دریافت کرده، پردازش می‌کنند و نتیجه را به **stdout**ارسال می‌کنند. این برنامه‌ها اغلب ساده و متمرکز بر یک وظیفه خاص هستند. برای مثال، دستور grepدر لینوکس یک فیلتر متنی است که خطوطی از ورودی را که با یک الگوی خاص مطابقت دارند، انتخاب می‌کند و به خروجی ارسال می‌کنند. به همین ترتیب، دستور sortیک فیلتر است که خطوط ورودی را مرتب می‌کند و نتیجه را به خروجی می‌فرستد.

فصل ۷

فلسفه یونیکس :ده اصل کوچکتر

فلسفه یونیکس، به عنوان یکی از پایه‌های اصلی دنیای کامپیوتر و نرم‌افزار، مجموعه‌ای از اصول و مفاهیم است که نه تنها بر سیستم‌عامل‌های مبتنی بر یونیکس، بلکه بر بسیاری از سیستم‌های دیگر نیز تأثیر عمیقی گذاشته است. این اصول، که هسته اصلی فلسفه یونیکس را تشکیل می‌دهند، به‌عنوان بستری برای توسعه و پیشرفت دنیای یونیکس عمل می‌کنند. این اصول چنان ریشه‌دار و اساسی هستند که به ندرت کسی می‌تواند آنها را به چالش بکشد و همچنان خود را یک "فرد یونیکس" بداند. هرگونه تلاش برای زیر سؤال بردن این اصول، ممکن است باعث ایجاد سوءظن در جامعه یونیکس (که لینوکس نیز بخشی از آن است) شود، زیرا چنین فردی ممکن است تعهدی به مفاهیم و ارزش‌های یونیکس نداشته باشد.پس از بررسی و دفاع از این اصول، که می‌توان آنها را به نوعی "جزمات دین یونیکس" نامید، اکنون زمان آن است که به سراغ برخی از دکترین‌ها و آموزه‌های این فلسفه برویم. توسعه‌دهندگان یونیکس و لینوکس با تمام وجود تلاش می‌کنند تا یکپارچگی این اصول را حفظ کنند و از آنها در برابر هرگونه تغییر ناخواسته یا انحراف محافظت نمایند. این اصول، که تا اینجا مورد بحث قرار گرفته‌اند، به‌عنوان پایه‌های غیرقابل‌انکار یونیکس شناخته می‌شوند. با این حال، دکترین‌هایی که در ادامه بررسی می‌شوند، بیشتر در دسته‌ی "بله، من با این کار کنار می‌آیم" قرار می‌گیرند. این بدان معناست که اگرچه ممکن است همه افراد جامعه یونیکس با تمامی این دکترین‌ها موافق نباشند، اما جامعه به‌عنوان یک کل، معمولاً خود را با آنها هماهنگ می‌کند.

یکی از ویژگی‌های جالب توجه در فلسفه یونیکس، تأکید آن بر **چگونگی انجام کارها** است، نه لزوماً **چرایی انجام آنها** .در بسیاری از موارد، روش‌های انجام کارها در یونیکس به‌گونه‌ای است که سال‌هاست به همان شکل انجام می‌شوند و تغییر چندانی نکرده‌اند. این موضوع ممکن است برای برخی عجیب به نظر برسد، اما باید در نظر داشت که یونیکس، مانند بسیاری از مذاهب و سنت‌های تثبیت‌شده، دارای آداب و رسوم خاص خود است. این سنت‌ها گاهی اوقات بدون دلیل خاصی، جز اینکه "همیشه به این شکل انجام شده‌اند"، ادامه می‌یابند. لینوکس، به‌عنوان یکی از مشتقات یونیکس، این ویژگی را به نوعی حفظ کرده است، اما با بسته‌بندی جدید و مدرن‌تری ارائه می‌دهد.یکی از مفاهیمی که در اینجا به طور خاص به آن پرداخته نشده است، اما در دنیای لینوکس و یونیکس بسیار مهم است، مفهوم **منبع باز** (Open Source)است. این مفهوم، که توسط بسیاری از انقلابیون تصادفی لینوکس با جدیت دنبال می‌شود، در فرهنگ لینوکس ریشه‌دار است و سال‌هاست که به‌عنوان یکی از اصول اصلی این سیستم‌عامل شناخته می‌شود. پیشگامانی مانند ریچارد استالمن (Richard M. Stallman)و دیگران، سال‌ها پیش این مفهوم را به‌عنوان یکی از ارکان اصلی جنبش نرم‌افزار آزاد مطرح کردند. با این حال، به دلیل اهمیت بالای این موضوع، سزاوار است که در جای دیگری با عمق بیشتری به آن پرداخته شود.

جالب است که برخی از اصول و مفاهیم یونیکس، حتی توسط طرفداران سیستم‌عامل‌های دیگر نیز پذیرفته شده‌اند. این موضوع نشان‌دهنده‌ی این است که ایده‌های خوب در دنیای کامپیوتر به سرعت گسترش می‌یابند و مرزهای سیستم‌عامل‌ها را در می‌نوردند. توسعه‌دهندگان نرم‌افزار در سیستم‌های دیگر، گاهی اوقات مفاهیم یونیکس را کشف می‌کنند که در نگاه اول ممکن است برای محیط‌شان مناسب به نظر نرسند، اما در عمل شایستگی خود را نشان می‌دهند. این مفاهیم سپس در طراحی‌های آنها گنجانده می‌شوند و گاهی اوقات منجر به ایجاد سیستم‌ها و برنامه‌هایی می‌شوند که طعم و بوی یونیکس دارند.در نهایت، فلسفه یونیکس نه تنها به‌عنوان یک مجموعه‌ی ثابت از اصول و قوانین، بلکه به‌عنوان یک چارچوب فکری و روش‌شناسی برای توسعه نرم‌افزار و سیستم‌عامل‌ها عمل می‌کند. این فلسفه، با تأکید بر سادگی، ماژولاریته، و استفاده‌ی مجدد از کدها، به توسعه‌دهندگان این امکان را می‌دهد که سیستم‌هایی کارآمد و قابل اعتماد ایجاد کنند. حتی اگر برخی از این اصول در نگاه اول قدیمی یا غیرضروری به نظر برسند، اما تجربه‌ی چند دهه‌ای یونیکس و لینوکس نشان داده است که این اصول همچنان ارزشمند و کاربردی هستند.

اجازه تنظیمات محیط به کاربر

داستان توسعه **UWM** (Unix Window Manager)و تأثیر آن بر دنیای رابط‌های کاربری و مدیران پنجره، نمونه‌ای جالب از نحوه‌ی شکل‌گیری ایده‌های نوآورانه و تأثیر آنها بر فناوری‌های بعدی استUWM .، که سال‌ها پیش به‌عنوان یک مدیر پنجره برای سیستم پنجره‌ی X توسعه داده شد، بسیاری از قابلیت‌هایی را که امروزه در سیستم‌های پنجره‌ای مدرن بدیهی به‌نظر می‌رسند، معرفی کرد. این قابلیت‌ها شامل توانایی جابجایی پنجره‌ها، تغییر اندازه‌ی آنها، تغییر ترتیب انباشتگی و بسیاری دیگر از ویژگی‌هایی است که امروزه بخشی

کامپیوتر خانگی آتاری :مهندسی انسانی به عنوان هنر

در اوایل دهه ۱۹۸۰، دو رایانه خانگی آتاری ۸۰۰ و نسخه ارزان‌تر آن، آتاری ۴۰۰، توانستند سهم قابل توجهی از بازار رایانه‌های خانگی را به خود اختصاص دهند. این دو دستگاه به‌ویژه به عنوان ماشین‌هایی برای علاقه‌مندان به بازی‌های رایانه‌ای شناخته شدند و شهرت خود را بیشتر به دلیل گرافیک پیشرفته و قابلیت‌های صوتی خود به دست آوردند. بازی Star Raiders که یک شبیه‌سازی پیشتازان فضا بود و در آن ستاره‌ها و اژدرهای فوتونی در فضای سه‌بعدی حرکت می‌کردند، یکی از بازی‌هایی بود که موفقیت آتاری را در بازار داخلی بیشتر کرد. البته این موفقیت طولی نکشید و در نهایت این دستگاه‌ها توسط رایانه‌های شخصی محبوب‌تری مانند IBM و Commodore ٦٤که قیمت مناسب‌تری داشتند، جایگزین شدند. طراحی گرافیک‌های ساده "بازیکن-موشک (Sprites) "و استفاده از پردازنده لیست نمایشگر، این رایانه‌ها را در مقایسه با رقبای جدیدتر ابتدایی می‌ساخت. در همان دوران، طراحان سیستم‌عامل یونیکس و آتاری هدف مشترکی داشتند که ساخت سیستمی برای انجام بازی‌ها بود. در یونیکس، این هدف به بازی‌های فضایی محدود نمی‌شد و به‌طور کلی به تمامی بازی‌ها تعمیم داده می‌شد. این ممکن است یکی از دلایل موفقیت این سیستم‌ها باشد، چرا که مردم اغلب به چیزی که برایشان لذت‌بخش است، علاقه‌مند می‌شوند و با اشتیاق بیشتری به آن می‌پردازند. امروزه بسیاری از توسعه‌دهندگان نرم‌افزار برای لذت بردن از کار خود، نرم‌افزارهایی برای لینوکس می‌نویسند. برای این افراد، نرم‌افزار نویسی به‌عنوان یک سرگرمی و فعالیتی به سبک گیک تلقی می‌شود. این تمایل به سرگرم شدن، حتی فراتر از نیازهای اجتماعی و بقا، نشان‌دهنده یک نیاز درونی است که ممکن است برای کسانی که خارج از دنیای کامپیوتر هستند، عجیب به نظر برسد، اما بسیاری از متخصصان لینوکس آن را یک سرگرمی عالی می‌دانند.

کریس کرافورد، یکی از کارکنان آتاری در آن زمان، تأثیر زیادی بر طراحی سیستم‌عامل کامپیوتر خانگی آتاری داشت. نرم‌افزارهای کاربردی (که عمدتاً بازی‌ها بودند) او استانداردهایی را تعیین کردند که تمام نرم‌افزارهای بعدی آتاری بر اساس آن‌ها قضاوت می‌شدند. کرافورد بسیاری از اصول طراحی خود را در کتاب De Re Atari: A Guide to Effective Programmingمستند کرده است، جایی که او فرآیندهای فکری انسان‌ها را با فرآیندهای فکری کامپیوترها مقایسه می‌کند. به اعتقاد کرافورد، کامپیوترها موجوداتی باهوش هستند، اما فاقد ویژگی‌های فیزیکی هستند. فرآیندهای فکری آنها "مستقیم، تحلیلی و خاص" هستند، در حالی که انسان‌ها الگوهای فکری تداعی، یکپارچه و پراکنده‌ای دارند. این تفاوت در فرآیندهای فکری، مانعی در ارتباط میان انسان‌ها و کامپیوترها ایجاد می‌کند. کرافورد باور داشت که هدف برنامه‌نویس نه افزایش قدرت هومونکولوس (موجود کوچک درون کامپیوتر) بلکه شکستن این مانع ارتباطی است.

این رویکرد کرافورد شباهت زیادی به فلسفه طراحی سیستم‌عامل یونیکس داشت. در حالی که کرافورد بر این باور بود که مهم‌ترین هدف نرم‌افزار، برقراری ارتباط با انسان است، در سیستم‌عامل یونیکس اولویت اصلی برقراری ارتباط با سایر برنامه‌ها بود. هدف نهایی این بود که برنامه یونیکس باید بتواند با انسان‌ها تعامل کند، اما این امر به‌عنوان یک هدف ثانویه در نظر گرفته می‌شد. از لحاظ طراحی رابط کاربری، رایانه خانگی آتاری و سیستم‌عامل یونیکس به طور چشمگیری با هم تفاوت داشتند. نرم‌افزارهای آتاری بیشتر سعی داشتند که گزینه‌ها را محدود کنند

فصل ۱۱

یک برنامه جامع

هنگامی که رولینگ استونز سی دی *Forty Licks* خود را در ۱ اکتبر ۲۰۰۲ منتشر کرد، همه انتظار داشتند که این آلبوم بلافاصله به صدر جدول موسیقی برسد. اگرچه آلبوم به رتبه دوم دست یافت، استونز با آرامش پذیرفت که همیشه نمی‌توان به آنچه می‌خواهید دست یافت. در حقیقت، این مجموعه بهترین آهنگ‌های گروهی که شاید یکی از برجسته‌ترین نام‌ها در تاریخ راک باشد، در حالی که در صدر جدول قرار گرفت، مجموعه دیگری از بهترین آهنگ‌ها توسط تنها یک نفر، یعنی الویس آرون پریسلی، توانست صدر جدول را به خود اختصاص دهد. این آلبوم با عنوان *ELVIS 30 #1 Hits* که در تاریخ ۲۴ سپتامبر ۲۰۰۲ منتشر شد، توانست موفقیت‌های بزرگی را کسب کند و در مقابل تمام قدرتی که استونز داشت، بهترین جایگاه ممکن را برای خود رقم بزند.موسیقی استونز، که به عنوان یکی از نمادهای موسیقی راک شناخته می‌شود، به شدت تحت تأثیر شور و هیجان غیرقابل توصیف آنها قرار داشت. در عین حال، بسیاری از ذهن‌های متعجب و مشکوک در تلاش بودند که بدانند چرا یک هنرمند که عمری طولانی‌تر از استونز دارد، می‌تواند در چنین موقعیتی قرار گیرد. چرا استونز نتواست از جایگاه خود فراتر برود و چرا الویس توانست چنین سلطنتی بر موسیقی داشته باشد؟پاسخ به این سوال ساده است: موسیقی استونز نوعی موسیقی بسته و مختص به خود بود، در حالی که الویس به صورت بازاری عمل می‌کرد و موسیقی را به شیوه‌ای نوآورانه از فرهنگ‌های مختلف و سبک‌های موسیقی مختلف، از راک اند رول تا بلوز، کانتری، گاسپل و تصنیف‌های مختلف، به یک کوله‌پشتی عظیم تبدیل کرده بود. الویس، در عین حال که در دنیای موسیقی مشغول کار بر روی قطعات خود بود، از دیگران نیز الهام می‌گرفت. این روش بر بازار تاثیر زیادی داشت، زیرا او توانست تأثیرات بیشتری از سایر هنرمندان در کارهای خود بگنجاند.

در دنیای موسیقی استونز، این گروه به‌عنوان یکی از برجسته‌ترین گروه‌های موسیقی راک شناخته می‌شود که بسیاری از شب‌ها را صرف نوشتن و اجرای آهنگ‌های جدید کردند. میک جگر و کیت ریچاردز هر دو جزو نبوغ‌های خلاقانه گروه بودند که توانستند اثری ماندگار از خود به جای بگذارند. اما این کار نیز همچنان نوعی تلاش بسته و اختصاصی بود. به عبارت دیگر، استونز در پروژه‌های خود به‌طور نادر با سایر هنرمندان همکاری می‌کردند، و این بدان معنا بود که آهنگ‌هایی که از آن‌ها شنیده می‌شد بیشتر نتیجه کار خود گروه بود تا همکاری‌های گسترده با دیگران. این تلاش‌ها محدود به یک محیط بسته و انتزاعی بودند که در آن نمی‌توانستند از خلاقیت‌های دیگر هنرمندان بهره‌برداری کنند. در حقیقت، استونز بیشتر با محدودیت‌های خود دست و پنجه نرم می‌کرد تا اینکه بتواند از همکاری با دیگران بهره‌برداری کند.

در مقابل، الویس به شکلی متفاوت و به گونه‌ای کاملاً باز عمل می‌کرد. او تمام سبک‌ها و ایده‌های مختلف موسیقی را پذیرفته و در آثار خود ادغام می‌کرد. الویس نه تنها از موسیقی خود لذت می‌برد، بلکه با بازآفرینی و وام‌گیری از دیگر هنرمندان و ترانه‌سراها نیز به موفقیت‌های بزرگی دست می‌یافت. در واقع، یکی از ویژگی‌های برجسته الویس استفاده از "بازآفرینی" در موسیقی‌اش بود، که این امر به او اجازه می‌داد تا تأثیر بسیار بیشتری بر دنیای سرگرمی و موسیقی بگذارد. او توانست با این شیوه، نام خود را در تاریخ موسیقی جاودانه کند و همزمان فضای بیشتری برای دیگر هنرمندان فراهم کند تا آثار خود را ارائه دهند.

می‌دهد تا سیستم‌های پیچیده را به شیوه‌ای ساختاریافته و قابل درک توسعه دهند. با این حال، مانند هر ابزار قدرتمند دیگری، OOPنیز نیاز به استفاده صحیح و مسئولانه دارد تا از ایجاد سیستم‌های پیچیده و غیرقابل نگهداری جلوگیری شود. OOPتا زمانی که تغییر تکتونیکی بعدی در دنیای فناوری رخ دهد، به عنوان یک نیروی محرکه در توسعه نرم‌افزار باقی خواهد ماند. تا آن زمان، توسعه‌دهندگان باید به دنبال ایجاد سیستم‌هایی باشند که نه تنها قدرتمند و کارآمد باشند، بلکه ساده و قابل درک نیز باشند. این همان چیزی است که فلسفه یونیکس و OOPدر کنار هم به ما می‌آموزند.

برنامه نویسی افراطی

برنامه‌نویسی افراطی (XP) و فلسفه یونیکس دو رویکرد مهم در دنیای توسعه نرم‌افزار هستند که هر کدام به شیوه‌ای خاص به حل مشکلات و بهبود فرآیندهای توسعه می‌پردازند. اگر به تعریف برنامه‌نویسی افراطی نگاهی بیندازید(http://www.extremeprogramming.org/what.html) ، ممکن است این تصور به وجود آید که مخترعان XP فلسفه یونیکس را گرفته‌اند، برچسبی جدید روی آن زده‌اند و آن را به نام خود معرفی کرده‌اند. شاید این تصور درست باشد، اما در هر صورت، روش XP بسیاری از اصولی را که توسعه‌دهندگان یونیکس دیروز و توسعه‌دهندگان لینوکس امروز همواره بر آن تأکید داشته‌اند، تأیید می‌کند: یک نمونه اولیه بسازید، آن را سریع اجرا کنید و منتظر نمانید. به اعتبار جامعهXP ، این روش خدمات بزرگی در رسمی‌کردن توسعه تکراری و انتقال ارزش آن به مدیران غیرفنی و اجرایی جهان انجام داده است XP .بسیاری از اصولی را که برنامه‌نویسان یونیکس همواره بر آن تأکید داشته‌اند، پذیرفته و آن را برای افرادی که در دنیای بازاریابی و حسابداری فعالیت می‌کنند، قابل درک و جذاب کرده است. این جامعه در حال آموزش این است که بهترین راه برای توسعه نرم‌افزار ممکن است این باشد که بنشینید، آن را امتحان کنید، و سپس با برنامه‌نویس در مورد آنچه در آن دوست دارید یا ندارید، صحبت کنید. این رویکردی است که به شدت مورد استقبال قرار گرفته است.

یکی از حوزه‌های تمرکز که متفکران XP آن را به وضوح بیان کرده‌اند، موضوع آزمایش است. طرفداران یونیکس معمولاً نرم‌افزار خود را با انتشار نمونه‌های اولیه برای کاربران مشتاق که همیشه در پی آخرین فناوری‌ها هستند، آزمایش می‌کنند. این روش به جامعه یونیکس کمک کرده است، زیرا در این شرایط معمولاً تعداد آزمایش‌کنندگان زیاد است. با این حال، این روش گاهی اوقات می‌تواند خلأهایی ایجاد کند. از سوی دیگر، رویکرد XP این است که ابتدا نرم‌افزار تست را بنویسید. این روش به شما کمک می‌کند تا نیازهای تجاری نرم‌افزار را زودتر شناسایی و برطرف کنید .در عمل، آزمایش اولیه در روش XP یک موهبت متفاوت است. مشارکت دادن مشتریان در مراحل اولیه آزمایش می‌تواند بسیار مفید باشد. با این حال، گاهی اوقات توسعه‌دهندگان در اوایل آزمایش‌ها آنقدر درگیر می‌شوند که به خواسته‌های مشتری توجه کافی نمی‌کنند. حتی بدتر از آن، از آنجایی که الزامات در طول زمان تغییر می‌کنند، معمولاً باید به عقب برگردید و آزمایش‌ها را دوباره انجام دهید، کاری که همیشه در طول دوره‌ای که به پایان پروژه توسعه نرم‌افزار نزدیک می‌شود، انجام نمی‌شود. هنگامی که این رویکرد به حد افراط کشیده می‌شود، این فرض به وجود می‌آید که فرد بهتر است از قبل همه الزامات را بداند زیرا قرار است آزمایش‌هایی را برای آن الزامات بنویسد.

LEARNING HIVE

contact me:

seilany.ir
learninghive.ir
emperor-os.ir
Hubuntu.ir
predator-os.ir
telegram: @seilany

HosseinSeilani

Designer,Developer and Linux sysadmin

فیلم بلندهم

Founder and Developer of Emperor-OS, Hubuntu and Predator-OS. I bring significant experience as a Linux/Windows sysadmin and graphical web design,UX/UI to the Open Source Community.

LITTLE-PSYCHO
DESTRUCTIVE LINUX

Emperor-OS

Predator-OS