مقایسه تطبیقی معماری سیستم‌های عامل

علیرضا فرجی
دانشجو ارشد مهندسی کامپیوتر گرایش شبکه های کامپیوتری، دانشگاه جامع امام حسین(ع)
سید جواد موسوی حسینی
دانشجو ارشد مهندسی کامپیوتر گرایش شبکه های کامپیوتری، دانشگاه جامع امام حسین(ع)

چکیده
میکرو کرنل که گنو ماخ و کرنل سیستم‌عامل مینیکس نمونه‌هایی از آن هستند، از یک کرنل بسیار ریز با حجم پایین تشکیل‌شده که فقط وظایف اساسی سامانه مانند راه‌اندازی و انتقال ارتباطات میان پردازشی سطح پایین میان سرویس‌دهنده‌ها و دادن دسترسی‌های لازم به آن‌ها را بر عهده دارد و مابقی کارها توسط مجموعه‌ای از سرویس‌دهنده‌ها که در حالت کاربر سیستم‌عامل، روی کرنل قرار می‌گیرند و با یکدیگر در ارتباط هستند انجام می‌پذیرد. این ساختار کرنل، سیستم‌عامل را بسیار منعطف کرده و به توسعه‌دهندگان اجازه می‌دهد با قرار دادن اجزای دلخواه خود به‌صورت موردنیاز، سامانه خود را برای مقاصد خاص خود طراحی کنند. در این مقاله کلیاتی در مورد هسته سیستم‌های عامل و انواع ریز هسته‌ها و تفاوت ساختاری و عملکردی گونه‌های متفاوت آن ارائه می شود.

واژگان کلیدی: میکرو کرنل، سیستم‌عامل، معماری، سیمبین، هسته

مقدمه

سیستم عامل مهم ترین نرم افزاری است که روی کامپیوتر اجرا می شود. حافظه و فرآیندهای کامپیوتر و همچنین تمامی نرم افزارها و سخت افزارهای آن را مدیریت می کند. همچنین به شما این امکان را می دهد که بدون اینکه بدانید چگونه به زبان کامپیوتر صحبت کنید، با کامپیوتر ارتباط برقرار کنید. بدون سیستم عامل، کامپیوتر بی فایده است. یک سیستم عامل (OS) تمام برنامه ها و برنامه های دیگر را در رایانه مدیریت می کند و توسط یک برنامه بوت در رایانه بارگذاری می شود. این برنامه ها را قادر می سازد تا با سخت افزار کامپیوتر تعامل داشته باشند. از طریق یک رابط برنامه کاربردی تعیین شده، برنامه های کاربردی خدمات را از سیستم عامل (API) درخواست می کنند. هسته نرم افزاری است که شامل اجزای اصلی سیستم عامل است. برای اجرای برنامه های دیگر، هر کامپیوتری باید حداقل یک سیستم عامل نصب داشته باشد.

معماری انواع سیستم عامل ها

سیستم عامل ها منابع سخت افزاری کامپیوتر را کنترل می کنند. هسته و پوسته بخش هایی از سیستم عامل هستند که عملیات ضروری را انجام می دهند. زمانی که کاربر دستوراتی را برای انجام هر عملیاتی می دهد، درخواست به قسمت پوسته می رود که به عنوان مفسر نیز شناخته می شود. سپس بخش پوسته برنامه انسانی را به کد ماشین ترجمه می‌کند و سپس درخواست را به قسمت هسته منتقل می‌کند. وقتی کرنل درخواست را از پوسته دریافت می‌کند، درخواست را پردازش می‌کند و نتیجه را روی صفحه نمایش می‌دهد. هسته همچنین به عنوان قلب سیستم عامل شناخته می شود زیرا هر عملیاتی توسط آن انجام می شود.

ارتباطات بین فرآیندی

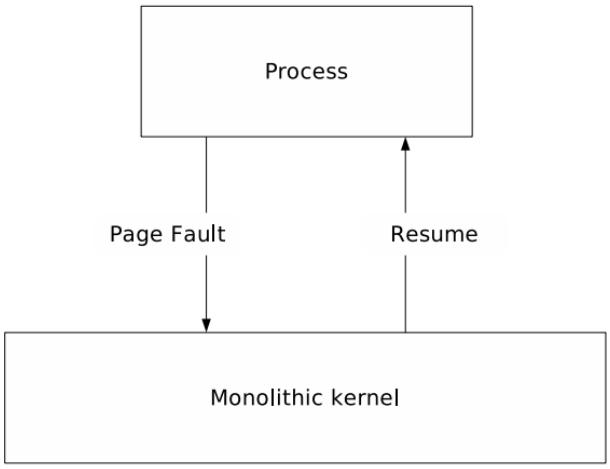
ارتباطات بین فرآیندی[1] به معنای نمایش یک برنامه در حافظه است. ممکن است از بخش‌های کوچک‌تر تعریف‌شده توسط برنامه‌نویس به نام «رشته‌ها» تشکیل شود. Thread ها امکان اجرای موازی بخش‌های مختلف یک برنامه را فراهم می‌کنند. نخ کوچک‌ترین واحد کد اجرایی است. یک فرآیند می‌تواند از چندین رشته تشکیل‌شده باشد. در بخش‌های بعدی یک‌رشته و یک فرآیند برابر هستند، اگر غیرازاین بیان‌نشده باشد. اولین مفهوم ارتباط بین فرآیندی (IPC) سیگنال نامیده می‌شود. به‌طور گسترده‌ای در سیستم‌های یونیکس استفاده می‌شود. سیگنال‌ها ثابت‌های عددی از پیش تعریف‌شده هستند، به‌عنوان‌مثال. KILL، STOP و غیره که توسط کاربر، سیستم‌عامل یا فرآیند دیگری به یک فرآیند ارسال می‌شوند. سیگنال‌ها توسط به‌اصطلاح کنترل‌کننده سیگنال، رویه‌های ساده که به هر فرآیند تعلق دارند، دریافت می‌شوند(Leva et al., 2013). این سیستم سریع است، اما مشکل این است که سیگنال‌ها باید اعداد از پیش تعریف‌شده باشند. سیگنال‌های موجود را نمی‌توان تغییر داد، زیرا در این صورت، فرآیندها به روشی متفاوت ازآنچه انتظار می‌رفت واکنش نشان می‌دهند. سیگنال‌های جدید باید استاندارد شوند که برای پیاده‌سازی تنها یک برنامه بسیار خسته‌کننده است. راه‌حل دیگر برای ارتباط، به‌اصطلاح سوکت ها هستند. یک فرآیند خود را به یک سوکت (یا بیشتر) متصل می‌کند و به آن گوش می‌دهد، یعنی از آن به بعد، می‌تواند پیام‌هایی را از فرآیندهای دیگر دریافت کند. بیشتر سوکت ها فول دوبلکس هستند. صاحب یک سوکت، یعنی فرآیندی که به سوکت متصل می‌شود، سرور نیز نامیده می‌شود، درحالی‌که سایر فرآیندها کلاینت نامیده می‌شوند. ازآنجایی‌که پیام‌های ارسال‌شده از طریق سوکت‌ها به مقادیر عددی محدود نمی‌شوند، اطلاعات می‌توانند فراتر از سیگنال‌های کنترلی باشند. هیچ محدودیتی در توسعه‌پذیری وجود ندارد تا زمانی که سرور بداند پیام‌های دریافتی چه معنایی دارند. سیستمی که از سوکت ها قوی‌تر است، صف‌های پیام هستند. یک صف پیام که به‌عنوان یک صف fifo ساخته‌شده است، تمام پیام‌های دریافتی را که توسط فرآیندهای دیگر ارسال می‌شود ذخیره می‌کند و آن‌ها را بر اساس اولویت آن‌ها مرتب می‌کند. یک فرآیند می‌تواند بیش

[1] InterProcessCommunication

از یک صف پیام داشته باشد که هر صف پیام مسئول انواع مختلفی از پیام‌ها است. درحالی‌که هسته‌های یکپارچه از سیگنال‌ها و سوکت‌ها برای اطمینان از ارتباطات بین فرآیندی استفاده می‌کنند، رویکرد µ-kernel از صف‌های پیام استفاده می‌کند. این اجازه می‌دهد که تمام قسمت‌های سیستم قابل تعویض هستند. اجزای سیستم هسته‌های یکپارچه تا حدودی "hardwired" هستند. این از توسعه‌پذیری جلوگیری می‌کند. اولین میکرو هسته‌ها IPC را ضعیف پیاده‌سازی کردند و در سوئیچ‌های زمینه کند بودند. مفاهیم جدیدی برای غلبه بر این کمبودهای عملکرد ارائه می‌کند(Gerbessiotis, 2020).

مدیریت حافظه

مدیریت حافظه[2] هسته‌های یکپارچه هر آنچه برای مدیریت حافظه لازم است را در فضای هسته پیاده‌سازی می‌کنند. این شامل استراتژی‌های تخصیص، مدیریت حافظه مجازی و الگوریتم‌های جایگزینی صفحه می‌شود.



شکل ۱ -مدیریت حافظه هسته یکپارچه

یکی از سرورها وظیفه مدیریت خطاهای صفحه و رزرو حافظه جدید را بر عهده دارد. هر بار که خطای صفحه رخ می‌دهد، درخواست باید از طریق هسته به پیجر برسد. پیجر باید برای دسترسی به حافظه و بازگشت به حالت کاربر وارد حالت ممتاز شود. سپس نتیجه را به فرآیند آغازگر (دوباره از طریق هسته) ارسال می‌کند. کل فرآیند رسیدگی به خطاهای صفحه یا رزرو صفحات حافظه جدید خسته‌کننده و زمان‌بر است. برای حل افت عملکرد، µ-kernel نسل دوم استراتژی‌های دقیق‌تری از مدیریت حافظه، مدیریت اولیه دارند: نقشه، اعطای و تراش. یک فرآیند اگر بخواهد این صفحات را به اشتراک بگذارد، صفحات حافظه را به فرآیند دیگری نگاشت می‌کند. وقتی فرآیندی صفحاتی را به فرآیند دیگری اعطا می‌کند، دیگر نمی‌تواند به آن‌ها دسترسی داشته باشد و تا زمانی که فرآیند اعطا آن‌ها را تخلیه نکند، تحت کنترل فرآیند دیگر هستند. فلاشینگ صفحات حافظه داده‌شده و نقشه‌برداری شده را دوباره به دست می‌آورد. این سیستم اکنون به‌صورت زیر عمل می‌کند: µ-kernel کل حافظه سیستم را هنگام راه‌اندازی به یک فرآیند ذخیره می‌کند، فرآیند سیستم پایه که (مانند سایر فرآیندها) در فضای کاربر قرار دارد. اگر فرآیندی نیاز به حافظه داشته باشد، دیگر مجبور نیست راه را از طریق هسته طی کند، بلکه مستقیماً از فرآیند سیستم پایه می‌پرسد. ازآنجایی‌که هر فرآیند فقط می‌تواند صفحات

[2] Memory managment

حافظه‌ای را که قبلاً متعلق به خود بود، اعطا/نقشه‌برداری/فلاش کند، حفاظت از حافظه همچنان وجود دارد. به‌این‌ترتیب سربار سوئیچ‌های زمینه به حداقل کاهش می‌یابد و عملکرد افزایش می‌یابد.

امنیت و ثبات

امنیت[3] محافظت از فرآیندهای سیستم در برابر تغییر توسط کاربر یا سایر فرآیندها یکی از ویژگی‌های مهم هسته است. با معرفی چندوظیفه‌ای (و چند رشته‌ای) مشکلات جدیدی در مورد جداسازی حافظه و فرآیندها به وجود آمد. این مشکلات شامل مسائلی مانند شرایط مسابقه[4]، حفاظت از حافظه[5] و خود امنیت سیستم است. هسته باید این امکان را داشته باشد که در صورت شکست فرآیند، عملکرد سیستم تحت تأثیر قرار نگیرد. اگر در مورد فرآیندهایی صحبت کنیم که در فضای کاربر اجرا می‌شوند، این‌یک کار کم‌وبیش ساده است؛ اما چه اتفاقی می‌افتد، اگر فرآیندی در هسته از کار بیفتد؟ به دلیل «hardwiring» فرآیندهای سیستم و وابستگی ناشی از رویکرد یکپارچه، می‌توان فرض کرد که سایر فرآیندها نیز از کار بیفتند و منجر به توقف در کل سیستم شود. حذف فرآیندهای سیستم از فضای هسته راهی برای غلبه بر این مشکلات است. استدلال دیگری برای یک میکرو کرنل واقعی، اندازه کد آن است. اطمینان از صحت یک هسته کوچک آسان‌تر از یک هسته بزرگ است. به‌این‌ترتیب مسائل مربوط به ثبات با این رویکرد ساده‌تر حل می‌شوند(Tanenbaum & Woodhull, 1997).

ارتباط I/O

ارتباط ورودی/خروجی[6] از طریق وقفه‌هایی که توسط سخت‌افزار صادرشده یا به آن ارسال می‌شود، کار می‌کند. هسته‌های یکپارچه و اکثر هسته‌های میکرو نسل اول، درایورهای دستگاه را در فضای هسته اجرا می‌کنند. وقفه‌های سخت‌افزاری مستقیماً توسط فرآیندهای هسته مدیریت می‌شوند. برای افزودن یا تغییر ویژگی‌های ارائه‌شده توسط سخت‌افزار، تمام لایه‌های بالای لایه تغییریافته در هسته یکپارچه نیز باید در بدترین حالت تغییر کنند. مفهوم به‌اصطلاح ماژول برای دستیابی به استقلال و جدایی بیشتر از هسته معرفی شد. یک ماژول بخش‌هایی از یک درایور را نشان می‌دهد و در طول زمان اجرا قابل بارگیری نیست. به‌این‌ترتیب، درایورهایی که سیستم موردنیاز نیستند، بارگذاری نمی‌شوند و حافظه حفظ می‌شود؛ اما ماژول‌های کرنل هنوز به هسته باینری وابسته هستند. این بدان معناست که ماژول‌هایی که با هسته یکپارچه نسل A (Linux ۲.۴.۰) کار می‌کنند، برای همکاری با جانشین آن (Linux ۲.۴.۲) اعطا نمی‌شوند. سازگاری منبع اغلب تضمین می‌شود، اما نه همیشه. اگر مفاهیم در هسته یکپارچه بیش‌ازحد تغییر کند، ماژول‌ها فقط به یک کامپایلر مجدد نیاز ندارند، بلکه به یک انطباق کامل کد نیاز دارند. رویکرد µ-kernel به‌طور مستقیم ارتباط I/O را مدیریت نمی‌کند. این فقط ارتباط را تضمین می‌کند. درخواست‌ها از یا به سخت‌افزار به‌عنوان پیام‌هایی توسط µkernel به سرورهای موجود در فضای کاربر هدایت می‌شوند. اگر سخت‌افزار یک وقفه ایجاد کند، µ-kernel پیامی را به سرور درایور دستگاه می‌فرستد و دیگر کاری برای انجام آن ندارد. سرور درایور دستگاه پیام را می‌گیرد و به درایور دستگاه مناسب می‌فرستد. به‌این‌ترتیب امکان اضافه کردن درایورهای جدید، تعویض درایور منیجر[7] بدون تعویض درایورها یا حتی مبادله کل سیستم مدیریت درایور بدون تغییر هیچ بخش دیگری از سیستم وجود دارد. نشان می دهد که قرار دادن درایورها در فضای کرنل، همانطور که توسط اولین نسل از هسته میکرو انجام شد، برای دستیابی به عملکرد قابل قبول، مطلوب نیست. به این ترتیب اندازه بزرگ می شود و هسته نمی تواند به طور کامل در حافظه کش پردازنده نگهداری شود(Tanenbaum, 2023).

---

[3] Security and Stability
[4] race conditions
[5] memory protection
[6] I/O Communication
[7] Drive manager

مدیریت فرایند

مدیریت فرآیند[8] شامل وظایف مختلفی مانند ایجاد، زمان‌بندی، خاتمه فرآیندها و قفل بن بست می‌شود. پردازه برنامه ای است که در حال اجراست که بخش مهمی از سیستم عامل های امروزی است. سیستم عامل باید منابعی را تخصیص دهد که فرآیندها را قادر به اشتراک گذاری و تبادل اطلاعات کند. همچنین منابع هر فرآیند را از روش های دیگر محافظت می کند و امکان همگام سازی بین فرآیندها را فراهم می کند. این وظیفه سیستم عامل است که تمام فرآیندهای در حال اجرا سیستم را مدیریت کند. با انجام وظایفی مانند زمان‌بندی فرآیند و مانند تخصیص منابع، عملیات را مدیریت می‌کند(de Oliveira et al., 2020).

شکل ۲-ساختار مدیریت فرایند

رابط کاربری

رابط کاربری[9] (UI) به بخشی از یک سیستم عامل، برنامه یا دستگاه اطلاق می شود که به کاربر اجازه ورود و دریافت اطلاعات را می دهد. یک رابط کاربری مبتنی بر متن را نمایش می دهد و دستورات آن معمولاً در خط فرمان با استفاده از صفحه کلید تایپ می شوند. با یک رابط کاربری گرافیکی عملکردها با کلیک کردن یا حرکت دادن دکمه ها، نمادها و منوها با استفاده از یک دستگاه اشاره گر انجام می شود(Hui et al., 2022).

مدیریت فایل سیستمی

مدیریت فایل سیستمی[10] یکی از ویژگی های اساسی اما مهم ارائه شده توسط سیستم عامل است. مدیریت فایل در سیستم عامل چیزی نیست جز نرم افزاری که فایل های (باینری، متنی، پی دی اف، اسناد، صدا، ویدئو و غیره) موجود در نرم افزارهای کامپیوتری را مدیریت می کند. سیستم فایل در سیستم عامل قادر به مدیریت فردی و همچنین گروهی از فایل های موجود در سیستم کامپیوتری است.(Peterson & Silberschatz, 1985) سیستم فایل در سیستم عامل به ما در مورد مکان، مالک، زمان ایجاد و اصلاح، نوع و وضعیت یک فایل موجود در سیستم کامپیوتری می گوید. موارد زیر برخی از وظایفی است که توسط مدیریت فایل سیستم عامل هر سیستم کامپیوتری انجام می شود:

۱- به ایجاد فایل های جدید در سیستم کامپیوتری و قرار دادن آنها در مکان های خاص کمک می کند.

[8] Process management
[9] User interface
[10] File system managment

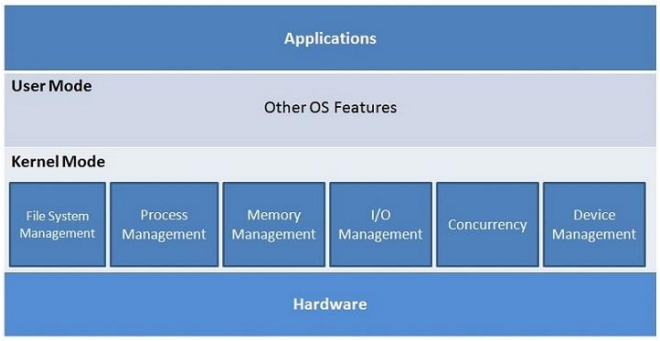
۲- این به مکان یابی آسان و سریع این فایل ها در سیستم کامپیوتری کمک می کند.
۳- این فرآیند به اشتراک گذاری فایل ها را بین کاربران مختلف بسیار آسان و کاربر پسند می کند.
۴- این کمک می کند تا فایل ها را در پوشه های جداگانه ای به نام دایرکتوری ها ذخیره کنید. این دایرکتوری ها به کاربران کمک می کنند تا فایل ها را سریع جستجو کنند یا فایل ها را بر اساس نوع یا کاربردشان مدیریت کنند.
۵- این به کاربر کمک می کند تا داده های فایل ها را تغییر دهد یا نام فایل را در فهرست ها تغییر دهد.

پیاده‌سازی

پیاده سازی[11] سازماندهی یک سیستم عامل مرزهای مختلف بین اجزای آن و مرزهای بین این اجزا و برنامه های کاربر را مشخص می کند. این مرزها می تواند منطقی باشد، یعنی برای برنامه نویس قابل مشاهده باشد یا فیزیکی، یعنی برای برنامه نویس شفاف باشد(Tanenbaum, 2023).

هسته یکپارچه

هسته یکپارچه[12] کل سیستم‌عامل در حالت سوپروایز و در فضای هسته فعالیت می‌کند. هسته یکپارچه فضای مجازی را بالاتر از سخت‌افزار رایانه به وجودمی آورد و در مقابل معماری‌هایی مثل ریزهسته مطرح می‌شود. این فضا شامل درخواست‌های سیستمی اولیه و ابتدایی است که برای همه سرویس‌های سیستم‌عامل مانند مدیریت پروسه، هم‌زمانی و مدیریت حافظه موردنیاز است هم‌زمان برای راه‌اندازها ماژول‌های مناسب را در اختیار می‌گذارد(**Hui et al., 2022**).



شکل ۳-معماری یکپارچه

گنو/لینوکس ۷

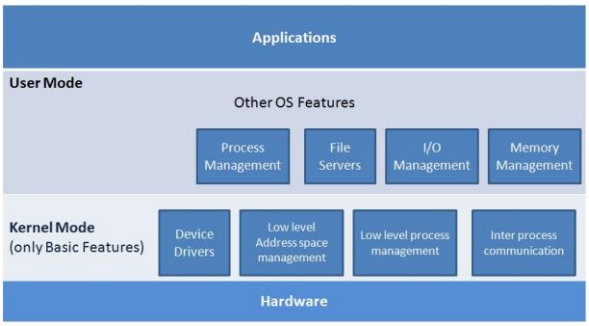
لینوکس ۷[13] یک پیاده‌سازی رایگان و منبع باز یونیکس است که توسط هزاران نفر توسعه‌یافته است. این‌یک نماینده معمولی از یک هسته یکپارچه است. به‌طور مداوم تقویت می‌شود، اغلب ساختار خود را تغییر می‌دهد. تغییر بخش‌های هسته به معنای کامپایل مجدد کامل است(License, 2008). تمام توابع سیستم، ازجمله کل فرآیند و مدیریت حافظه، زمان‌بندی فرآیند و رشته، عملکرد ورودی/خروجی و درایورها در فضای هسته پیاده‌سازی می‌شوند. ارتباطات ورودی/خروجی که توسط به‌اصطلاح ماژول‌ها ارائه می‌شود که می‌توانند در طول زمان اجرا وارد و حذف شوند، در مقابل هسته ساخته می‌شوند، یعنی: اگر هسته تغییر کند، مجموعه ماژول‌ها

11 IMPLEMENTATIONS
12 Monolithic Kernel
13 GNU/Linux۷

نیز تغییر می‌کند. اندازه تخمین زده‌شده یک هسته یکپارچه متوسط حدود بیست‌تا سی مگابایت است که منجر به فرآیند تعمیر و نگهداری خسته‌کننده می‌شود(Roch, 2004).

هسته ترکیبی

هسته ترکیبی[14] ویژگی‌های ریزهسته و هم ویژگی‌های هسته یکپارچه را باهم دارد. مدل‌های سنتی مورداستفاده در هسته سیستم‌عامل‌ها، مدل‌های هسته یکپارچه و ریزهسته بوده‌اند (نانو کرنل و اکزوکرنل حالات کوچک‌تری از ریزهسته هستند). برخلاف ریزهسته ها، در یک هسته پیوندی، همه یا اکثریت سرویس‌های سیستم‌عامل، در بخش فضای هسته قرار دارند، مشابه هسته یکپارچه، ولی برخلاف هسته‌های یکپارچه، نمی‌توان در زمان اجرا بودن سیستم، ماژول‌هایی را به هسته‌های پیوندی حذف یا اضافه کرد(**Malallah et al., 2021**).



شکل ۴ – معماری هسته ترکیبی

ماخ

ماخ [15]یک میکرو هسته از نسل اول است که در دانشگاه کارنگی ملون طراحی و توسعه‌یافته است. این سیستم، در میان سایرین، پایه Next، Mac OS X را نشان می‌دهد و پایه و اساس بسیاری از طرح‌های µ-kernel را ساخته است. این به‌عنوان یک هسته بسیار قابل‌حمل با اندازه کوچک تصور می‌شد که فقط حداقل مجموعه‌ای از توابع هسته را شامل می‌شود. عملکرد ضعیف ماخ در مقایسه با هسته‌های یکپارچه به این فرض منجر شد که µ-kernel نمی‌توانند سریع باشند؛ اما µ-kernel نسل دوم ثابت کردند که فقدان عملکرد به دلیل مشکلات پیاده‌سازی است، نه به دلیل طراحی µ-kernel. چندین اشتباه هسته میکرو نسل اول (به‌عنوان‌مثال ماخ) را می‌توان اشاره کرد:

i) برای سهولت حمل‌ونقل، طراحان یک‌لایه اضافی بین هسته و CPU معرفی کردند. به‌این‌ترتیب، فقط لایه باید برای یک پردازنده معین بهینه شود، اما این رویکرد اشتباه بود [۳].

ii) به دلیل عملکرد ضعیف، درایورهای دستگاه به فضای هسته بازگردانده شدند. این منجر به یک هسته متورم شد که دیگر نمی‌توانست در حافظه پنهان پردازنده بماند.

iii) ارتباط بین فرآیندی Mach بسیار خسته‌کننده و وقت‌گیر است(DAVID et al., 1992).

ویندوز NT

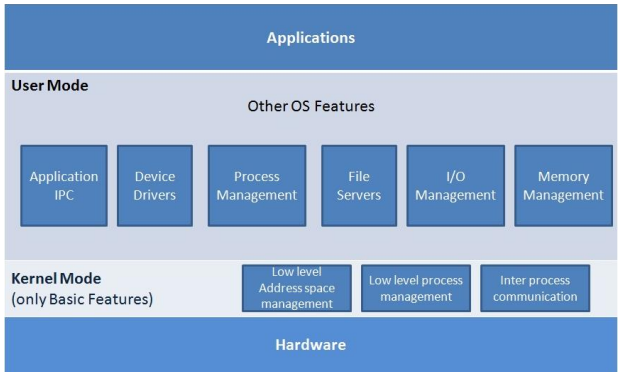
14 Hybrid kernel

15 Mach

مایکروسافت هسته را برای ویندوز ۹NT خود در ابتدای دهه ۱۹۹۰ معرفی کرد. برنامه‌ریزی‌شده بود که یک میکرو کرنل باشد، اما به دلیل عدم عملکرد، مایکروسافت تصمیم گرفت بسیاری از خدمات سیستم را به فضای هسته بازگرداند، ازجمله، درایورهای دستگاه و پشته‌های ارتباطی. این باعث نفخ هسته شد و بزرگ‌تر از اکثر هسته‌های یکپارچه آن زمان شد(Solomon, 1998). NT (شامل هسته) به‌عنوان یک سیستم‌عامل شی گرا طراحی‌شده است؛ بنابراین، تمام ساختارهای اساسی، مانند فرآیندها، رشته‌ها، درایورهای دستگاه و سایر ساختارها به‌عنوان اشیاء پیاده‌سازی می‌شوند و توسط یک مدیر شی مدیریت می‌شوند. هسته از طریق یک‌لایه به‌اصطلاح [۱۶](HAL) با سخت‌افزار صحبت می‌کند که به نفع انتقال به دیگر معماری‌های سیستم است.

میکرو کرنل

در طراحی میکرو کرنل‌ها[۱۷] سعی بر آن می‌شود که اکثر سرویس‌های مانند فایل سیستم و ... در فضای حافظه کاربری اجرا شوند و در عوض سرویس‌های اصلی و اساسی سیستم‌عامل ازجمله تخصیص دهنده حافظه (نه سرویس مدیریت حافظه) و سرویس زمان‌بندی و سرویس مدیریت پیام‌ها را در فضای هسته اصلی سیستم‌عامل قرارداد.



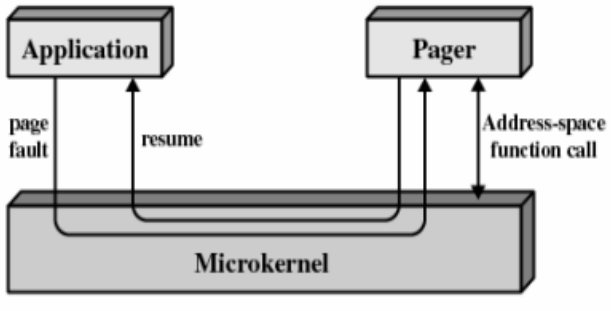
شکل ۵-طراحی میکرو کرنل

مطابق با شکل زیر یک سرور برای مدیر حافظه اصلی، یک سرور برای مدیر پردازش، یک سرور برای مدیریت سیستم فایل و غیره وجود دارد. در معماری ریزهسته به دلیل آنکه فرایندها (دایمون یا سرورها) در فضای کاربر وجود دارند، پس به‌نوعی عملیات تعویض متن تا حدی حذف خواهد یافت. اما از سوی دیگر به دلیل وجود سرورهای هسته در فضای کاربر و نیاز به دسترسی به فضای هسته، عملاً عملیات تعویض متن بسیار افزایش خواهد یافت. پس می‌توان نتیجه گرفت که ریزهسته ها نسبت به هسته‌های یکپارچه کندتر هستند(Ulmanu, 2019).

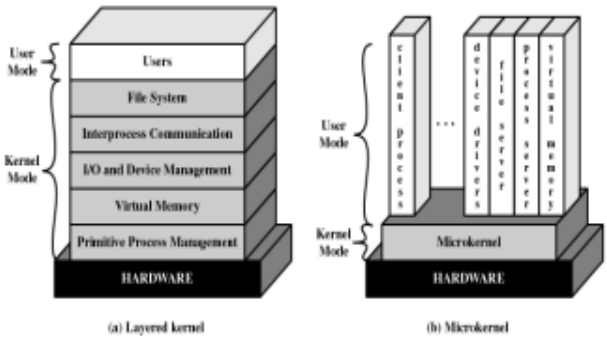
برخلاف معماری یکپارچه، به دلیل آنکه سرویس‌های هسته به‌صورت سرورهایی مجزا درون فضای کاربر در حال اجرا هستند، ازکارافتادن یکی از آن‌ها بر عملکرد هسته تأثیر نمی‌گذارد. پس می‌توان نتیجه گرفت که این معماری نسبت به معماری یکپارچه پایدارتر است(Saeki et al., 2020).

این عملیات در ۳ دسته کلی زیر قرار می‌گیرند:۱ -مدیریت سطح پایین حافظه ۲ ارتباطات بین فرایندها ۳ مدیریت ورودی خروجی و وقفه

[16] Hardware Abstraction Layer
[17] microkernel



شکل ۶-طراحی میکرو کرنل



شکل ۷-معماری میکرو کرنل

QNX10

محبوب‌ترین سیستم‌عامل خالص مبتنی بر میکرو برای برنامه‌های بلادرنگ است. برنامه‌های بی‌درنگ بر پیش‌بینی پذیری و ثبات تأکیددارند. نمونه‌هایی از دستگاه‌های بی‌درنگ دستگاه‌های تعبیه‌شده مانند ماکروویو، ماشین ظرف‌شویی، سیستم‌های ایمنی خودرو، تلفن‌های همراه و غیره هستند. هدف اصلی آن بازار تعبیه‌شده است و به‌عنوان نسخه دسکتاپ نیز موجود است. فقط اساسی‌ترین اولیه‌ها مانند سیگنال‌ها، تایمرها و برنامه‌ریزی‌ها در فضای هسته قرار می‌گیرند که منجر به یک هسته به‌اندازه ۶۴ کیلوبایت می‌شود. همه اجزای دیگر، به‌عنوان‌مثال: پشته‌های پروتکل، درایورها و سیستم‌های فایل، خارج از هسته اجرا می‌شوند. همه فرآیندها از طریق یک گذرگاه پیام مجازی منفرد ارتباط برقرار می‌کنند که به شما امکان می‌دهد هر مؤلفه‌ای را در حین پرواز وصل یا وصل کنید. هسته QNX (به نام نوترینو) سازگار با posix است، در C پیاده‌سازی شده است و بنابراین می‌توان آن را به‌راحتی برای پلتفرم‌ها و سیستم‌های عامل مختلف تنظیم کرد. توسعه‌دهندگان به‌راحتی می‌توانند هسته QNX را حذف کنند و عملکردهای ناخواسته را حذف کنند(Balyabin & Petrenko, 2022)، به‌عنوان‌مثال: حفاظت از حافظه که به‌عنوان یک ماژول پیاده‌سازی می‌شود. اگر برنامه (های) موردنظر به ویژگی نیاز نداشته باشند، می‌توان آن را به‌راحتی و بدون ویرایش کد منبع حذف کرد. QNX به دلیل استقلال فضای هسته، درایورها را در طول زمان اجرا، آزمایش و اشکال‌زدایی می‌دهد.

L4

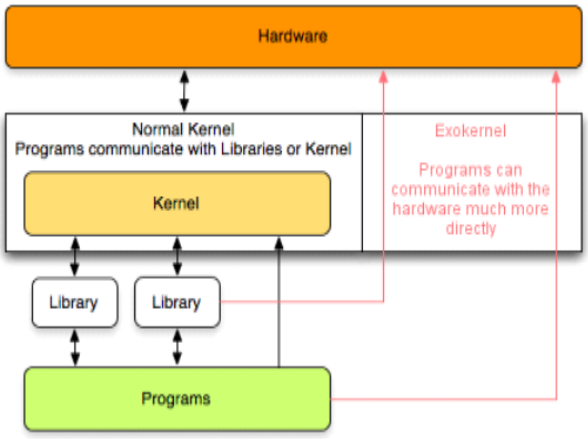
هسته L4 در TU Dresden (Van Roosmalen & Gritzki, 2020) توسط گروه معماری سیستم‌ها با همکاری مرکز تحقیقات IBM Watson پیاده‌سازی شد. این به نسل دوم میکرو هسته‌ها تعلق دارد. این به همراه هسته نوترینوی QNX ثابت

کرد که میکرو هسته‌ها به همان سرعتی هستند که همتای یکپارچه خوددارند و امکان گسترش آسان را فراهم می‌کنند. عملکرد با اندازه کوچک (۱۲ کیلوبایت کد) و ارتباطات بین فرآیندی بهینه‌شده (IPC) به دست می‌آید. تنها به دلیل سه انتزاع اصلی و هفت فراخوانی سیستم، در بالای این انتزاعات که در فضای هسته پیاده‌سازی می‌شوند، به L4 اجازه می‌دهد تا کاملاً در حافظه پنهان سطح اول پردازنده قرار گیرد. همه عملکردهای دیگر، مانند مدیریت حافظه، درایورهای دستگاه، مدیریت وقفه، پشته‌های پروتکل و غیره در فضای کاربر قرار دارند. به‌این‌ترتیب، L4 فقط دسترسی به سخت‌افزار را کنترل می‌کند و مدیریت thread اولیه را انجام می‌دهد. وقفه‌های سخت‌افزاری به‌عنوان پیام به فضای کاربر ارسال می‌شوند. هسته در پردازش آن‌ها دخالتی ندارد. مدیریت حافظه به‌طور کامل در فضای کاربر انجام می‌شود و سوئیچ‌های فضای آدرس سریع از طریق کدهای بهینه‌سازی شده برای پردازش تضمین می‌شوند.(Roch, 2004)

اگزو کرنل

اگزو کرنل[18] نوعی هسته سیستم‌عامل است که در MIT ایجادشده و به دنبال ارائه مدیریت در سطح برنامه منابع سخت‌افزاری است. ساختار اگزو کرنل برای تقسیم حفاظت از منابع از مدیریت برای بهبود سفارشی‌سازی خاص برنامه طراحی‌شده است.(Hui et al., 2022)

اگزوکورنل ها به دلیل عملکرد محدود، اندازه کوچکی دارند. سیستم‌عامل‌های سنتی همیشه بر عملکرد، عملکرد و منطقه برنامه‌هایی که بر روی آن‌ها ساخته می‌شوند، تأثیر می‌گذارند، زیرا سیستم‌عامل بین برنامه‌های کاربردی کاربر و منابع سخت‌افزاری قرار می‌گیرد. سیستم‌عامل اگزو کرنل تلاش می‌کند تا با حذف این مفهوم که یک سیستم‌عامل باید انتزاعی‌هایی برای ساخت برنامه‌ها ارائه کند، به این مشکل رسیدگی کند. ایده این است که تا آنجا که ممکن است انتزاعات کمتری را به توسعه‌دهندگان تحمیل کنیم و به آن‌ها آزادی استفاده از انتزاعات را در صورت نیاز و در صورت نیاز فراهم کنیم. هدف اصلی یک اگزوکرنل اطمینان از عدم وجود انتزاع اجباری است که همان چیزی است که یک هسته خارجی را از هسته‌های میکرو و یکپارچه متفاوت می‌کند.



شکل ۸-ساختار اگزو کرنل

برخی از ویژگی‌های سیستم‌عامل اگزوکرنل عبارت‌اند از:

۱- پشتیبانی برتر برای کنترل اپلیکیشن

[18] Exokernel

۲- امنیت را از مدیریت جدا می‌کند
۳- چکیده‌ها به‌طور ایمن به یک سیستم‌عامل کتابخانه نامعتبر منتقل می‌شوند
۴- یک رابط سطح پایین ارائه می‌دهد
۵- سیستم‌عامل‌های کتابخانه قابلیت حمل و سازگاری را فراهم می‌کنند

مزایای سیستم‌عامل اگزوکرنل عبارت‌اند از:

۱- بهبود عملکرد برنامه‌ها
۲- استفاده کارآمدتر از منابع سخت‌افزاری از طریق تخصیص و ابطال دقیق منابع
۳- توسعه و تست آسان‌تر سیستم‌عامل‌های جدید
۴- هر برنامه فضای کاربری مجاز است مدیریت حافظه بهینه خود را اعمال کند

برخی از ایرادات سیستم‌عامل exokernel عبارت‌اند از:

۱- کاهش قوام
۲- طراحی پیچیده رابط اگزو کرنل

سیستم‌های لایه‌ای

سیستم‌های لایه‌ای[19]یک تعمیم از رویکرد سازمان‌دهی سیستم‌عامل به‌عنوان سلسله مراتبی از لایه‌ها است که هر یک‌بر اساس لایه زیر آن ساخته‌شده است. اولین سیستمی که به این روش ساخته شد، سیستم THE بود که در Technische Hogeschool Eindhoven در هلند توسط E. W. Dijkstra (۱۹۶۸) و دانش آموزانش ساخته شد. سیستم THE یک سیستم دسته‌ای ساده برای یک کامپیوتر هلندی، Electrologica X۸ بود که دارای ۳۲ هزار کلمه ۲۷ بیتی بود (بیت‌ها در آن زمان گران بودند). سیستم دارای ۶ لایه بود، لایه ۰ با تخصیص پردازنده، سوئیچینگ بین فرآیندها در هنگام بروز وقفه یا تایمرها سروکار داشت. در بالای لایه ۰، سیستم از فرآیندهای متوالی تشکیل‌شده بود که هر یک می‌توانست بدون نگرانی در مورد این واقعیت که چندین پردازش روی یک پردازنده اجرا می‌شود، برنامه‌ریزی‌شده باشد. به‌عبارت‌دیگر، لایه، چند برنامه‌نویسی اساسی CPU را فراهم می‌کند. لایه ۱ مدیریت حافظه را انجام داد. فضایی را برای پردازش‌ها در حافظه اصلی و در یک درام کلمه K۵۱۲ برای نگهداری قسمت‌هایی از فرآیندها (صفحات) که جایی در حافظه اصلی وجود نداشت، اختصاص داد. در بالای لایه ۱، فرآیندها نیازی به نگرانی در مورد اینکه آیا در حافظه هستند یا روی درام، نبودند. نرم‌افزار لایه ۱ مراقب بود تا اطمینان حاصل کند که صفحات هر زمان که نیاز بود به حافظه آورده می‌شدند(Saeki et al., 2020).

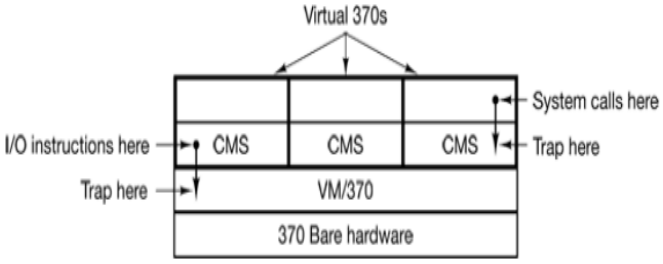
لایه ۲ ارتباط بین هر فرآیند و کنسول اپراتور را مدیریت می‌کرد. بالای این لایه هر فرآیند به‌طور مؤثر کنسول اپراتور خود را داشت. لایه ۳ مدیریت دستگاه‌های ورودی/خروجی و بافر کردن جریان‌های اطلاعات به و از آن‌ها را بر عهده داشت. در بالای لایه ۳، هر فرآیند می‌تواند به‌جای دستگاه‌های واقعی با ویژگی‌های خاص، با دستگاه‌های I/O انتزاعی با ویژگی‌های خوب سروکار داشته باشد. لایه ۴ جایی بود که برنامه‌های کاربر پیدا شدند. آن‌ها نیازی به نگرانی در مورد فرآیند، حافظه، کنسول یا مدیریت I/O نداشتند. فرآیند اپراتور سیستم در لایه ۵ قرار داشت. تعمیم بیشتر مفهوم لایه‌بندی در سیستم MULTICS وجود داشت. به‌جای لایه‌ها، MULTICS دارای یک سری حلقه‌های متحدالمرکز توصیف شد که حلقه‌های داخلی دارای امتیاز بیشتری نسبت به لایه‌های بیرونی هستند (که درواقع همان چیزی است). هنگامی‌که یک‌رویه در یک حلقه بیرونی می‌خواست یک‌رویه را در یک حلقه داخلی فراخوانی کند، باید معادل یک فراخوانی سیستمی را ایجاد می‌کرد، یعنی یک دستورالعمل TRAP که پارامترهای آن قبل از ادامه

[19] Layered Systems

فراخوانی به‌دقت ازنظر اعتبار بررسی می‌شدند. اگرچه کل سیستم‌عامل بخشی از فضای آدرس هر فرآیند کاربر در MULTICS بود، سخت‌افزار این امکان را فراهم می‌آورد که رویه‌های جداگانه (بخش‌های حافظه، درواقع) به‌عنوان محافظت در برابر خواندن، نوشتن یا اجرا تعیین شوند. درحالی‌که طرح لایه‌بندی THE درواقع فقط یک کمک طراحی بود، زیرا تمام بخش‌های سیستم درنهایت به یک برنامه شی واحد متصل شدند، در MULTICS، مکانیسم حلقه در زمان اجرا بسیار وجود داشت و توسط سخت‌افزار اعمال می‌شد. مزیت مکانیزم حلقه این است که به‌راحتی می‌توان آن را به ساختار زیرسیستم‌های کاربر گسترش داد. به‌عنوان‌مثال، یک استاد می‌تواند برنامه‌ای برای تست و نمره دادن به برنامه‌های دانش‌آموز بنویسد و این برنامه را در ring n اجرا کند، درحالی‌که برنامه‌های دانشجویی در ring n + ۱ اجرا می‌شود تا نتوانند نمرات خود را تغییر دهند(Malallah et al., 2021).

ماشین‌های مجازی[20]

نسخه‌های اولیه OS/۳۶۰ کاملاً سیستم‌های دسته‌ای بودند. بااین‌وجود، تعداد زیادی از ۳۶۰ کاربر می‌خواستند اشتراک‌گذاری زمانی داشته باشند، بنابراین گروه‌های مختلف، چه در داخل و چه خارج از IBM تصمیم گرفتند تا سیستم‌های اشتراک‌گذاری زمانی را برای آن بنویسند. سیستم اشتراک‌گذاری زمانی رسمی آی‌بی‌ام، TSS/۳۶۰، با تأخیر تحویل داده شد و زمانی که بالاخره رسید، آن‌قدر بزرگ و کند بود که تعداد کمی از سایت‌ها به آن تبدیل شدند. درنهایت پس‌ازاینکه توسعه آن حدود ۵۰ میلیون دلار مصرف کرد، رها شد (گراهام، ۱۹۷۰)؛ اما گروهی در مرکز علمی IBM در کمبریج، ماساچوست، سیستم کاملاً متفاوتی را تولید کردند که IBM درنهایت آن را به‌عنوان یک محصول پذیرفت و اکنون به‌طور گسترده در مین‌فریم‌های باقی‌مانده آن استفاده می‌شود. این سیستم که در ابتدا CP/CMS نام داشت و بعداً به VM/ 370تغییر نام داد (Seawright and MacKinnon, ۱۹۷۹)، بر اساس یک مشاهدات زیرکانه بود: یک سیستم اشتراک زمانی چند برنامه‌نویسی و یک ماشین توسعه‌یافته با رابط راحت‌تر از سخت‌افزار خالی را فراهم می‌کند. ماهیت VM/ 370 جداسازی کامل این دو عملکرد است.



شکل ۹ – ساختار VM/370 با CMS

قلب سیستم که به‌عنوان مانیتور ماشین مجازی شناخته می‌شود، روی سخت‌افزار خالی اجرا می‌شود و چند برنامه‌نویسی را انجام می‌دهد و نه یک، بلکه چندین ماشین مجازی را برای لایه بعدی فراهم می‌کند. بااین‌حال، برخلاف تمام سیستم‌عامل‌های دیگر، این ماشین‌های مجازی ماشین‌های توسعه‌یافته، با فایل ها و سایر ویژگی‌های خوب نیستند.(O'Regan, 2021) در عوض، آن‌ها کپی‌های دقیقی از سخت‌افزار خالی هستند، ازجمله حالت هسته/کاربر، ورودی/خروجی، وقفه‌ها و هر چیز دیگری که ماشین واقعی دارد. ازآنجایی‌که هر ماشین مجازی با سخت‌افزار واقعی یکسان است، هرکدام می‌توانند هر سیستم‌عاملی را اجرا کنند. مستقیماً روی

[20] Virtual Machines

سخت‌افزار لخت اجرا می‌شود. ماشین‌های مجازی مختلف می‌توانند و اغلب انجام می‌دهند، سیستم‌عامل‌های مختلفی را اجرا می‌کنند، برخی یکی از نوادگان OS/360 را برای پردازش دسته‌ای یا تراکنش اجرا می‌کنند، درحالی‌که برخی دیگر یک سیستم تعاملی تک کاربره به نام CMS (سیستم مانیتور مکالمه) را برای اشتراک‌گذاری زمانی تعاملی اجرا می‌کنند. کاربران هنگامی‌که یک برنامه CMS یک فراخوانی سیستمی را اجرا می‌کند، تماس به سیستم‌عامل در ماشین مجازی خودش محبوس می‌شود، نه در VM/ 370، درست همان‌طور که اگر روی یک ماشین واقعی به‌جای یک ماشین مجازی اجرا می‌شد، می‌ماند. سپس CMS دستورالعمل‌های سخت‌افزار ورودی/خروجی معمولی را برای خواندن دیسک مجازی خود یا هر آنچه برای انجام تماس موردنیاز است، صادر می‌کند. این دستورالعمل‌های ورودی/خروجی توسط VM/ 370به دام می‌افتند که سپس آن‌ها را به‌عنوان بخشی از شبیه‌سازی سخت‌افزار واقعی خود انجام می‌دهد.(Tanenbaum, 2023).

مزایای سازمان ریزهسته

۱- **واسط‌های یکنواخت**: نیازی وجود ندارد که فرایندها بین خدمات سطح کاربر و سطح هسته تمایزی قائل شوند چراکه تمام این خدمات با استفاده از ارسال پیام فراهم می‌شود.

۲- **قابلیت گسترش**: اضافه کردن خدمات جدید یا حتی فراهم کردن خدمات چندگانه در یک محیط کاری را آسان می‌کند.

۳- **قابلیت انعطاف**: نه‌تنها ممکن است خصیصه‌های جدید به یک سیستم‌عامل اضافه شود بلکه ممکن است از خصوصیات موجود کاست تا به یک طراحی کوچک‌تر و کارآمدتر دست‌یافت.

۴- **قابلیت حمل**: همه یا حداقل اکثر کد مربوط به ویژگی‌های خاص پردازنده در ریزهسته قرار دارد؛ بنابراین تغییرات لازم برای حمل سیستم‌عامل به پردازنده جدید کمتر می‌شود؛ و به‌قرار گیری در گروه‌های منطقی گرایش می‌یابند.

۵- **قابلیت اطمینان**: یک ریزهسته می‌تواند مورد آزمایش‌های سخت قرار گیرد. استفاده آن از تعداد کمی واسط‌های برنامه‌سازی کاربردی، مجال تولید کد باکیفیت برای خدمات سیستم‌عامل، در خارج از هسته را افزایش می‌دهد.

۶- **حمایت از سیستم توزیعی**: پیام گرایی ارتباطات ریزهسته، آن را به یک سیستم توزیعی گسترش می‌دهد. حمایت از سیستم‌عامل شیءگرا: می‌توان از یک رویکرد شیءگرایی در نظام طراحی ریزهسته و ایجاد گسترش مؤلفه‌ای سیستم‌عامل استفاده کرد(Xiao et al., 2019).

توسعه پذیری و قابلیت حمل

توسعه پذیری و قابلیت حمل[21] برجسته ترین واقعیت برای µ-kernel است. علاوه بر اندازه آن، یکی از بزرگترین تفاوت ها با هسته های یکپارچه است. افزودن ویژگی‌های جدید به یک سیستم یکپارچه به معنای کامپایل مجدد کل هسته، اغلب شامل کل زیرساخت درایور است. اگر یک روال مدیریت حافظه جدید دارید و می خواهید آن را در یک معماری یکپارچه پیاده سازی کنید، ممکن است به اصلاح سایر قسمت های سیستم نیاز باشد. در مورد µ-kernel سرویس ها از طریق سیستم پیام از یکدیگر جدا می شوند. کافی است مدیر حافظه جدید را دوباره پیاده سازی کنید. فرآیندهایی که قبلاً از مدیر دیگر استفاده می کردند، متوجه تغییر نمی شوند. µ-kernel نیز انعطاف‌پذیری خود را در حذف ویژگی‌ها نشان می‌دهند. به‌این‌ترتیب یک µ-kernel می‌تواند پایه‌یک سیستم‌عامل دسکتاپ و همچنین ابزارهای بلادرنگ در سیستم‌های تک تراشه باشد. از سوی دیگر، میکرو هسته‌ها باید برای پردازنده‌ای که قرار است روی آن اجرا شود، بسیار بهینه‌شده باشند. نشان داده شد که تنها معرفی یک «لایه سازگاری» بین هسته و پردازنده کافی نیست همان‌طور که با هسته‌های میکرو نسل اول انجام شد. به‌این‌ترتیب، میکرو-کرنل‌ها مستقل از ماشین نگهداری می‌شوند و

[21] Extensibility and Portability

به‌راحتی قابل‌انتقال هستند. متأسفانه، این رویکرد مانع از دستیابی آن µ-kernel به عملکرد لازم و درنتیجه انعطاف‌پذیری شد. نشان می‌دهد که معرفی چنین لایه‌ای بین هسته و پردازنده چندین پیامد دارد:

۱. چنین میکرو هسته‌ای نمی‌تواند از سخت‌افزار خاصی استفاده کند.
۲. نمی‌تواند اقدامات احتیاطی را برای دور زدن یا اجتناب از مشکلات عملکرد سخت‌افزار خاص انجام دهد.
۳. µ-kernel پایین‌ترین لایه سیستم‌عامل را تشکیل می‌دهند؛ بنابراین حتی الگوریتم‌های مورداستفاده در داخل µ-kerneland مفاهیم داخلی آن به‌شدت به پردازنده وابسته هستند.

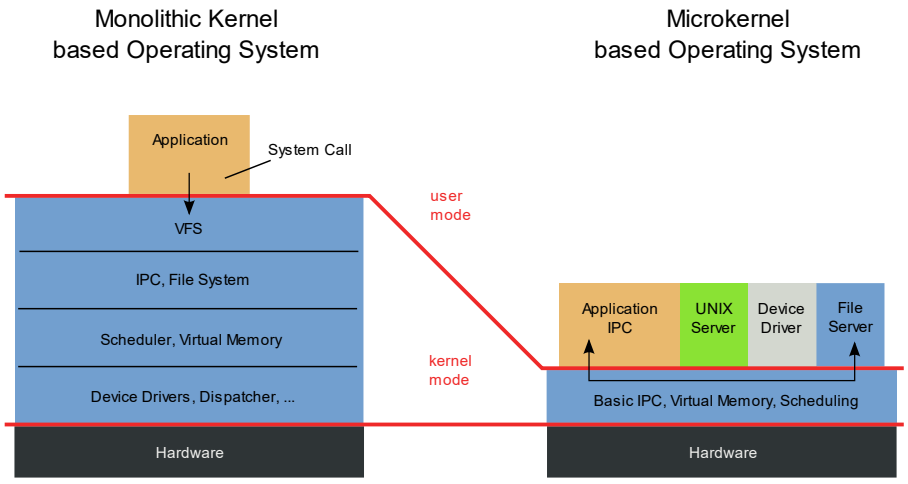
مقایسه هسته‌های یکپارچه و ریزهسته

مزایا micro: در صورت کرش کردن یک بخش از کرنل سیستم پایدار است و به فعالیت اش ادامه می‌دهد وریسک امنیتی را فقط در همان بخش از زیر کرنل باقی نگه می‌دارد. انعطاف‌پذیری سیستم را افزایش می‌دهد. قابلیت حمل افزایش می‌یابد. دستورات کمتر در میکرو کرنل = سهولت تبدیل زبان در حمل کردن؛ قابلیت سازگاری با محیط‌های توزیع‌شده و شبکه وجود دارد.(در میکرو کرنل می‌توان بسیاری از سرویس‌ها را پخش کرد)؛ در صورت اضافه شدن سخت‌افزار جدید می‌توان قابلیت جدیدی را به آن اضافه کرد. قابلیت اطمینان سیستم بالا می‌رود. مزیت معماری میکرو کرنل به پایداری و امنیت بیشتر آن به نسبت معماری Monolithic است، به دلیل اینکه اگر یک فرایند کرنل مانند driver device از کار بیافتد بقیه سرورهای سیستم‌عامل سالم می‌ماند. از طرف دیگر معماری Monolithic کارایی بهتری دارد (حدود ۵ درصد) و سریع‌تر اجرا می‌شود.(Tanenbaum, 2023)

معایب micro: روند توسعه بسیار سخت و طولانی؛ رفع خطای بسیار سخت؛ آسیب‌پذیری‌ها به‌سختی کشف می‌شود؛ کرنل حافظه بیشتری رو در رم به خودش اختصاص می‌دهد (نسبت به مونولیک کرنل). به دلیل اینکه تمام ارتباطات از طریق میکرو کرنل صورت می‌گیرد سرعت آن پایین است و ارتباط بین برنامه‌ها را میکرو کرنل بر عهده دارد (Liu et al., 2021).

مزایا monolithic: توسعه سریع؛ رفع خطای کدها بسیار راحت است؛ میزان اشغال رم توسط کرنل کمتر است.

معایب monolithic: درصورتی‌که کرنل پایدار نباشد کل سیستم ناپایدار می‌شود؛ درصورتی‌که کرنل دارای حفره امنیتی باشد این ناامنی در کل سیستم نمود دارد به همین خاطر کرنل مازولار رو مایکروسافت با میکرو کرنل ادغام کرد مخصوصاً بعد از اهمیت embedded device ها مثل ویندوزفون ها و البته اینترنت اشیا(Liu et al., 2021).



شکل ۱۰: مقایسه micro kernel با monolithic

وظایف ریزهسته: برقراری ارتباط بین مشتری و خدمتگزار (تبادل پیام بین آن‌ها)؛ **زمان‌بندی** و ایجاد چند برنامگی؛ مدیریت سطح پایین حافظه مثل ثبات‌های سخت‌افزاری؛ مدیریت سطح پایین ورودی- خروجی مثل ثبات‌های کنترل‌کننده O/I(Saeki et al., 2020)

سیستم‌عامل Symbian نمونه‌ای از معماری میکرو کرنل

Symbian سیستم‌عاملی ست برای وسایل دستی و همراه پس می‌بایست قدرت ارائه بالاترین کارایی در پائین ترین امکانات سخت‌افزاری. یکی دیگر از ویژگی‌هایی که Symbian قدرت پایداری و Stability این سیستم می‌بایست در سطح بسیار بالائی قرار بگیرد. از دیگر ویژگی‌های Symbian ویژگی ست بانام Active Objects که در آن در لحظاتی که از CPU استفاده نمی‌شود به‌طورکلی آن را خاموش می‌کند و به‌این‌ترتیب در مصرف باتری بسیار صرفه‌جوئی می‌گردد. در حقیقت هسته اصلی Symbian بر پایه معماری میکرو کرنل بناشده است بدین معنی که در ساختار آن کمترین رجوع و استفاده از کرنل توسط سیستم‌عامل انجام می‌شود وکلا این هسته شامل دو عنصر مدیریت حافظه و Schedular می‌باشد و در آن خبری از پشتیبانی از فایل های سیستمی و یا عناصر شبکه وجود ندارد و در حقیقت این‌گونه وظایف به عهده سرورهای خارج از کرنل واگذارشده است تا با محدود کردن وظایف کرنل سیستم‌عامل کند و سنگین نگردد.(Gos & Zabierowski, 2020)

نتیجه‌گیری و جمع‌بندی

۴L و QNX ثابت کرده‌اند که سرعت دیگر استدلالی علیه هسته‌های μ نیست. توسعه‌پذیری و قابلیت حمل اضافی آن‌ها را برای برنامه‌های مختلف از سیستم‌های تعبیه‌شده به دسکتاپ تعیین می‌کند. آن‌ها راحت‌تر از همتایان یکپارچه خود قابل نگهداری هستند. با توجه به استقلال بخش‌های مختلف که از طریق ارسال پیام امکان‌پذیر شده است، سیستم‌های μ-کرنل به‌راحتی قابل توسعه و اصلاح هستند. ارتباطات بین فرآیندی را می‌توان با الگوریتم‌های ارتباطی هوشمندانه، اغلب به‌استثنای μ-kernel، سریع‌تر ایجاد کرد. برای دستیابی به این کارایی، میکرو هسته‌ها باید مستقیماً برای پردازنده، جایی که در نظر گرفته‌شده‌اند، بهینه شوند. به‌این‌ترتیب می‌توان از پتانسیل کامل یک پردازنده به‌طور مستقیم استفاده کرد. هسته‌های μ اجازه می‌دهند چندین سیستم‌عامل را به‌طور هم‌زمان اجرا کنند. هر سیستم‌عاملی نیازی به پیاده‌سازی درایورهای دستگاه یا پشته‌های ارتباطی خود ندارد. این کار حتی عملکرد

بهتر و بهترین بهینه‌سازی را برای هر پردازنده به ارمغان می‌آورد.تمامی سیستم‌عامل‌های چندوظیفه‌ای دارای کرنل هستند؛ این کرنل است که وظیفه‌ی مدیریت منابع سیستم ازجمله حافظه، پردازش‌ها و درایورها را بر عهده دارد. دیگر اجزای سیستم‌عامل‌ها مانند ویندوز، OS X، اندروید و آی اواس در سطوحی مجزا از کرنل قرار دارند. کرنل مورداستفاده در اندروید، کرنل لینوکس است. ازآنجایی‌که هر دو کرنل لینوکس و اندروید متن‌باز هستند، بنابراین قابلیت ایجاد کاستوم کرنل به همراه تنظیمات مختلف برای اندروید وجود دارد و می‌توان از آن به‌عنوان کرنل جایگزین در دستگاه اندرویدی استفاده کرد. برای این منظور ابتدا باید دستگاه اندرویدی را روت و بوت لودر آن را آنلاک کرد.

منابع

Balyabin, A. A., & Petrenko, S. A. (2022). Platform (SDK) for Self-Healing of a Special Microkernel Operating System (KasperskyOS, QNX, Minix, osFree) Based on Cyberimmunity. 2022 XXV International Conference on Soft Computing and Measurements (SCM),

DAVID, L. B., DAVID, B. G., & DANIEL, P. J. (1992). Microkernel operating system architecture and Mach. *Journal of information processing*, 11-30.

de Oliveira, E. G., de Oliveira, M. S. F., Neto, N. R., de Paula Soldati, F., & Nassur, T. L. C. (2020). Development and evaluation of a mobile educational application to support teaching of management of process in Operating Systems. 2020 IEEE 20th international conference on advanced learning technologies (ICALT),

Gerbessiotis, A. (2020). CS 332-102: Principles of Operating Systems.

Gos, K., & Zabierowski, W. (2020). The comparison of microservice and monolithic architecture. 2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH),

Hui, W., Lin, Z., Zhongqiang, D., Tianyu, Y., Xiaolong, Z., & Tiantian, W. (2022). Component Management Framework for Operating System Based on Microkernel Architecture. 2022 China International Conference on Electricity Distribution (CICED),

Leva, A., Maggio, M., Papadopoulos, A. V., & Terraneo, F. (2013). *Control-based operating system design* (Vol. 89). Institution of Engineering and Technology.

License, G. G. P. (2008). The GNU Operating system. *URL: http://www. gnu. org/licenses/gpl. html (дата обращения: 21.03. 10)*.

Liu, B., Wu, C., & Guo, H. (2021). A Survey of Operating System Microkernel. 2021 International Conference on Intelligent Computing, Automation and Applications (ICAA),

Malallah, H., Zeebaree, S., Zebari, R. R., Sadeeq, M., Ageed, Z. S., Ibrahim, I. M., Yasin, H. M., & Merceedi, K. J. (2021). A comprehensive study of kernel (issues and concepts) in different operating systems. *Asian Journal of Research in Computer Science*, *8*(3), 16-31.

O'Regan, G. (2021). History of Operating Systems. In *A Brief History of Computing* (pp. 143-154). Springer.

Peterson, J. L., & Silberschatz, A. (1985). *Operating system concepts*. Addison-Wesley Longman Publishing Co., Inc.

Roch, B. (2004). Monolithic kernel vs. Microkernel. *TU Wien*, *1*.

Saeki, T., Nishiwaki, Y., Shinagawa, T., & Honiden, S. (2020). A robust and flexible operating system compatibility architecture. Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments,

Solomon, D. A. (1998). The Windows NT kernel architecture. *Computer*, *31*(10), 40-47.

Tanenbaum, A. S. (2023). MODERN OPERATING SYSTEMS SECOND EDITION. In.

Tanenbaum, A. S., & Woodhull, A. S. (1997). *Operating systems: design and implementation* (Vol. 2). Prentice Hall Englewood Cliffs.

Ulmanu, C. (2019). Types of operating system kernels. Conferința tehnico-științifică a studenților, masteranzilor și doctoranzilor,

Van Roosmalen, M., & Gritzki, A. (2020). Opportunities and challenges for the use of photovoltaics in building monuments–Case Study TU Dresden. IOP Conference Series: Earth and Environmental Science,

Xiao, J., Liu, X., Hu, X., Zhang, G., & Shen, J. (2019). Design of a data system for the avionics system based on the open system architecture. 2019 14th IEEE Conference on Industrial Electronics and Applications (ICIEA),

Comparative comparison of architecture of operating systems

Alireza Faraji **Seyed Javad Mousavi**

Abstract

The microkernel, of which GNU Mach and the kernel of the Minix operating system are examples, consists of a very small kernel with a low volume, which is responsible for only the basic tasks of the system, such as setting up and transferring low-level interprocessing communication between servers and giving them the necessary access. The rest of the work is done by a set of servers that are placed on the kernel in the user mode of the operating system and communicate with each other. This kernel structure makes the operating system very flexible and allows developers to design their system for their specific purposes by placing their desired components as needed. In this article, general information about the kernel of operating systems and types of microkernels and the structural and functional differences of its different types are presented.

Keywords: microkernel, operating system, architecture, Symbian, kernel