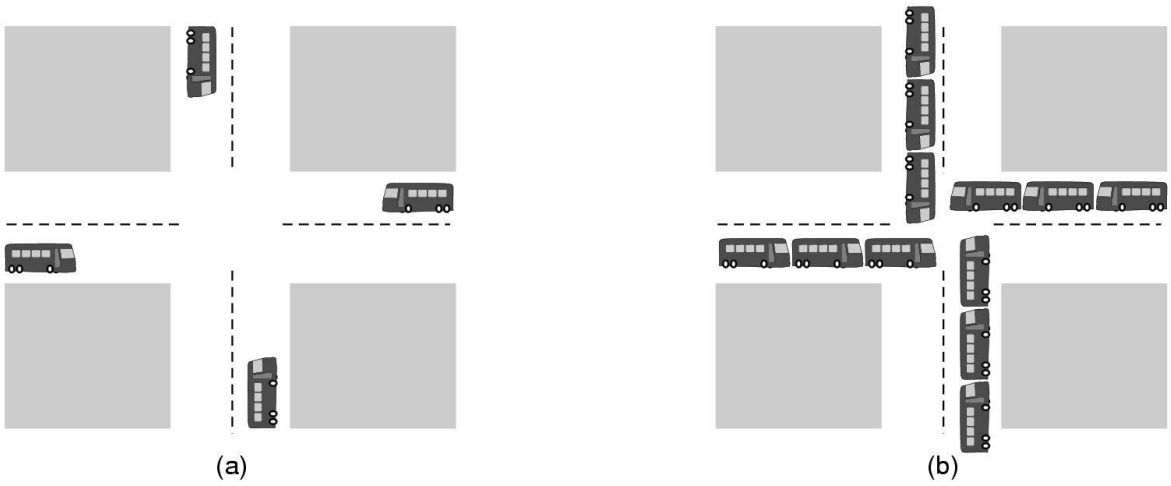
سیستم عامل پیشرفته

بن بست در سیستم های عامل (Dead lock):

مفهوم بن بست از ایجاد یا اجرای فرآیندهای هم روند یا Concurrent در یک سیستم ناشی می شود. وقتی که تعدادی فرآیند هم روند در یک سیستم وجود دارد سؤالی که مطرح می شود این است که آیا این فرآیندهای هم روند لزوماً باعث ایجاد بن بست می شوند؟ پاسخ این سؤال منفی است! فرآیندهای هم روند هنگامی ممکن است دچار بن بست شوند که شرایط خاصی را داشته باشند. هنگامی در یک سیستم بن بست رخ خواهد داد که هم روندی وجود داشته باشد، بین فرآیندهای هم روند منابع مشترک موجود باشد یعنی منبعی وجود داشته باشد که یک یا چند فرآیند به طور همزمان از آن استفاده کنند و سپس شرط mutual exclusion یا دو به دو ناسازگاری بر روی منابع مشترک اعمال شده باشد و آنگاه اگرزنجیره انتظار چرخشی در سیستم ایجاد شود سیستم دچار بن بست خواهد شد، ضمن اینکه برای رسیدن به زنجیره انتظار چرخشی باید حتماً شرط دیگری در سیستم وجود داشته باشد و آن این است که منابع تحت تکنیک hold and wait یعنی نگهداشت و انتظار، منابع را در اختیار بگیرند بنابراین بروز بن بست در یک سیستم خود ناشی از بروز برخی از اتفاقات دیگر است و شرایط خاصی باید در سیستم حکمفرما باشد تا احتمال بروز بن بست وجود داشته باشد. از جمله استراتژی‌های برخورد با بن بست می‌توان به استراتژی معروفی اشاره کرد که طی آن مطرح می شود که بررسی‌های انجام شده نشان می‌دهد بروز بن بست در یک سیستم احتمالاً هر ۵۰ سال یکبار اتفاق می افتد، بنابراین چنین پدیده ای که بروز آن به ندرت رخ خواهد داد بهتر است که اصولاً مورد بررسی قرار نگیرد. این استراتژی را استراتژی Ostrich می نامند و استراتژی است که برای سیستم‌های مهندسی مناسب است اما هیچگاه از چنین استراتژی در سیستم های مبتنی بر ریاضیات که مفاهیم آن قطعی و غیرقابل تغییر هستند استفاده نمی شود.

برای برخورد با بن بست استراتژی‌های دیگری هم وجود دارد. قبل از اینکه به این استراتژی ها بپردازیم و منابع مربوط به بن بست را اندکی مورد بحث قرار دهیم به شکل زیر دقت کنید.

@youseficlass



همانطور که در این شکل دیده می شود منبع مشترک برای چهار اتوبوسی که به عنوان فرآیندهای این سیستم مورد توجه قرار گرفته اند چهارراهی است که همه این فرآیندها یا تمام این ماشین هایی که در جهات مختلف قرار دارند باید از آن عبور کنند حال اگر شرایط به گونه ای باشد که در شکل نشان داده شده است و وجود انتظار چرخه ای را به دقت و به صراحت نشان می دهد آنگاه می توان گفت که سیستم دچار بن بست شده است چون در اینجا هر ماشینی منتظر عبور ماشین دیگری است و عملاً هیچ یک از این ماشین ها حرکتی نخواهند کرد.

دلایل بروز بن بست

یکی از دلایل بروز بن بست که شرط لازم برای ایجاد بن بست در یک سیستم است دو به دو ناسازگاری است که در این مبحث تنها به تعریف و کلیات آن بسنده می کنیم و برای دانستن روش های ایجاد آن می‌توانید به مباحث مربوط به سیستم عامل ۱ مراجعه نمائید.

همانطور که می دانید دو به دو ناسازگاری را به این شکل تعریف می کنیم که چنانچه شرایطی در یک سیستم ایجاد شود که ما مطمئن باشیم دو فرآیند هم روند که به یک منبع مشترک دسترسی دارند به گونه‌ای عمل کنند که هیچگاه این دو فرآیند به طور همزمان در داخل ناحیه بحرانی خود قرار نگیرند آنگاه عنوان خواهد شد که حتماً در سیستم دو به دو ناسازگاری ایجاد شده است.

@youseficlass

ناحیه بحرانی یا منطقه بحرانی (Critical section) یا (Critical region)

ناحیه بحرانی قسمتی از یک فرآیند است که منبع مشترک آن فرآیند با سایر فرآیندها در این قسمت مورد دسترسی قرار خواهد گرفت. روش‌های مختلفی برای ایجاد دو به دو ناسازگاری وجود دارد و همانطور که قبلاً هم عنوان شد الگوریتم‌های ایجاد دو به دو ناسازگاری باید شرایط خاصی را داشته باشند. از جمله این شرایط می توان به شرط پیشرفت و یا شرط انتظار محدود اشاره کرد. استفاده از سمافورها، مانیتورها، روش test and set lock ، استفاده از الگوریتم پیترسون (Peterson) از جمله ایده های ایجاد دو به دو ناسازگاری بین فرآیندها می باشند.

استراتژی های برخورد با بن بست

همانطور که قبلاً عنوان شد اولین استراتژی، استراتژی Ostrich بود که بیان می کرد با توجه به احتمال پایین بروز بن بست در سیستم، کافیست بن بست را در نظر نگیرید، دومین استراتژی، استراتژی detection یا استراتژی تشخیص بن بست است، سومین استراتژی را به عنوان استراتژی prevention یا جلوگیری از بروز بن بست می‌دانند و چهارمین استراتژی، استراتژی Avoidance یا اجتناب از بن بست است.

در استراتژی detection فرض بر این است که سیستم عامل به طور آزاد و به طور دلخواه منابع را در اختیار هر منبع درخواست کننده قرار می دهد. البته به شرط اینکه منبع موجود باشد و پس از تخصیص منابع چک می کند که آیا این تخصیص باعث ایجاد بن بست در سیستم شده است یا خیر؟ برای این کار تکنیک های مختلفی وجود دارد که از جمله این تکنیک ها می توان به رسم گراف تخصیص اشاره کرد و با رسم گراف تخصیص و بررسی وجود دور در این گراف می توان مشخص نمود که آیا سیستم دچار بن بست است یا خیر.

در استراتژی prevention یا جلوگیری بن بست فرض بر آن است که الگوریتم های تخصیص به گونه ای پیاده سازی شوند که مطمئن باشیم در سیستم بن بستی رخ نمی‌دهد. بنابراین قبل از اینکه منبعی به فرآیند، تخصیص داده شود به مسأله بروز بن بست دقت خواهد شد! لذا در استراتژی prevention ایده‌های مختلفی مطرح است از جمله اینکه منابع مورد شماره گذاری قرار گیرد و درخواست فرآیندها برای منابع بر حسب شماره آنها صورت گیرد و آنگاه هیچ منبعی حق نداشته باشد پس از اینکه یک منبع با شماره بیشتر را دریافت کرد درخواست منبعی با شماره کمتر را بدهد! این استراتژی باعث خواهد شد که

@youseficlass

هیچگاه در گراف تخصیص، حلقه ایجاد نشود ولی به هر حال محدودیت هایی برای سیستم بدنبال خواهد داشت.

از دیگر موارد می توان به الگوریتم banker یا بانکداران اشاره کرد که به نوعی در چهار چوب الگوریتم اجتناب یا Avoidance قرار می گیرد. در الگوریتم banker به سادگی بیان می شود تنها تخصیصی در سیستم امکان پذیر است که آن تخصیص منجر به بروز یک حالت امن یا حالت safe در سیستم شود و این استراتژی بیان می کند که اگر سیستم همواره در حالت امن باشد قطعاً بن بستی در سیستم رخ نخواهد داد اما اگر سیستم در حالت ناامن باشد ممکن است که سیستم در آینده دچار بن بست شود.

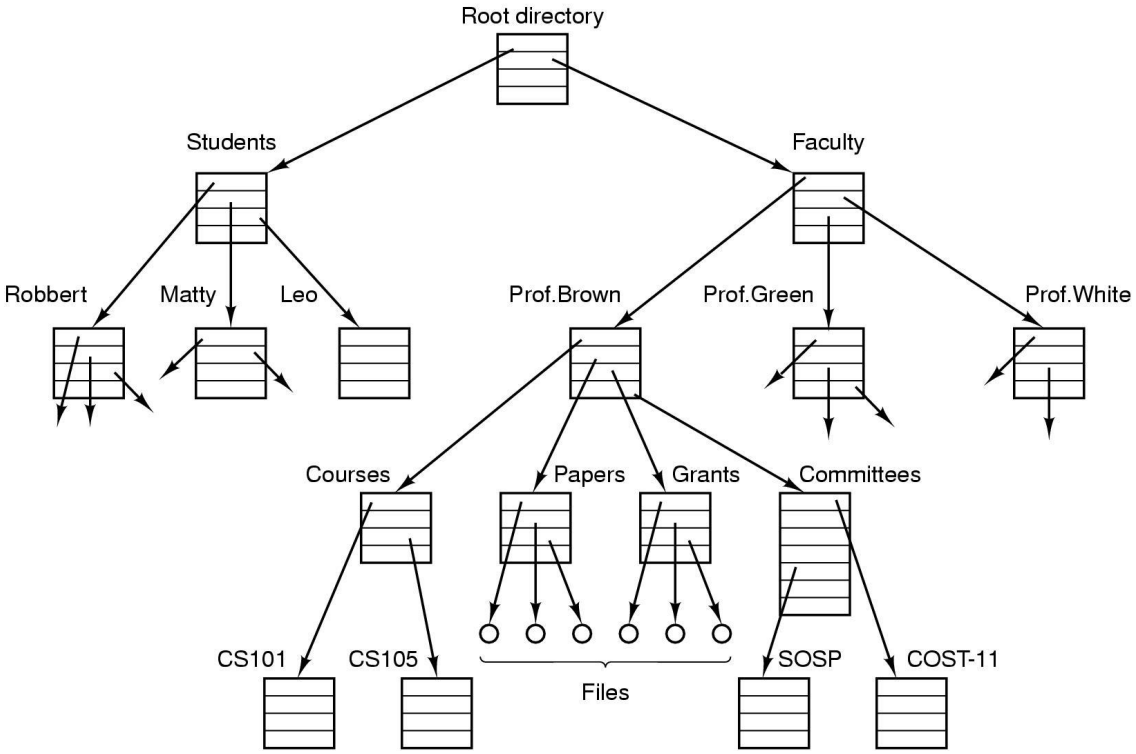
سیستم امن

نکته دیگری که باید به آن اشاره کرد، تعریف سیستم امن یا حالت امن یا safe state است. فرآیندی یا سیستمی در حالت امن قرار دارد اگر و تنها اگر ترتیبی از تخصیص‌ها وجود داشته باشد که منجر به انجام تمام فرآیندها شده و فرآیندها، منابع خود را آزاد می کنند.

سیستم‌های فایل

در ادامه مبحث مربوط به مفاهیم اصلی سیستم های عامل یکی دیگر از مواردی که مطرح می شود، مفهوم سیستم های فایل است. سیستم های فایل یا File system یک سیستم عامل، وظیفه نگهداری اطلاعات مربوط به فایل های موجود در یک سیستم را بر عهده دارد. یکی از مهم ترین قسمت‌های یک فایل سیستم، مدیریت فایل ها در غالب directory هاست که برای مدیریت فایل ها ساختارها و ساختمان های داده‌ای متفاوتی را قائل هستیم. از جمله معروفترین این ساختارها می‌توان به ساختار درختی اشاره کرد.

@youseficlass



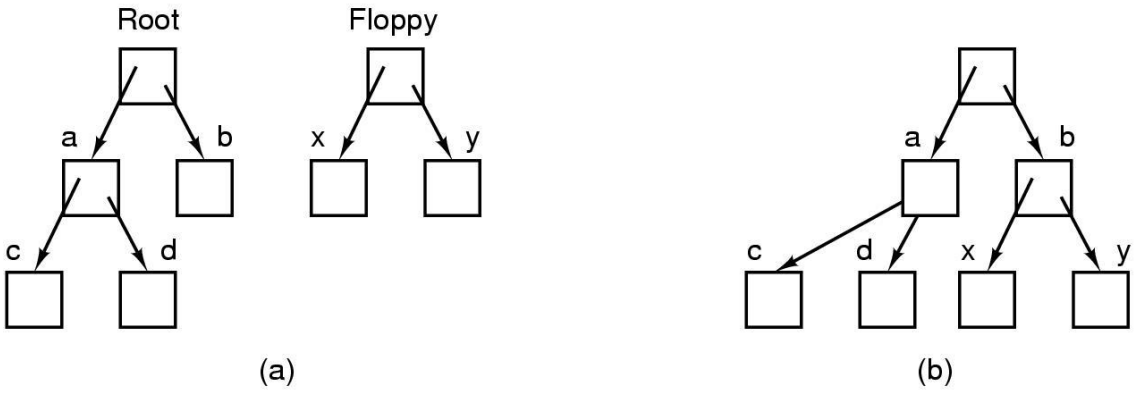
همانطوری که در این شکل دیده می شود، یک ساختار درختی شامل یک ریشه Root directory است که اطلاعات کلیه فایل های مربوط به سیستم در داخل این Root directory نگهداری می‌شود. برای Root directory، Sub directory ها یا نودهای دیگری نیز وجود دارند که با عنوان فرزندان این ریشه در نظر گرفته می شوند. به عنوان مثال در این فایل سیستم که با عنوان فایل سیستم یک دانشگاه طراحی شده است (شکل) به عنوان یک لایه بالا در سطح ADT دو گروه دانشجویان(student node) و اساتید (faculty node) به عنوان فرزندان ریشه قرار گرفته اند. به عبارت دیگر در سیستم مبتنی بر درخت که یک سیستم سلسله مراتبی است فرض می شود بعد از ریشه درخت کاربرانی که قرار است در سیستم به منابع (در اینجا همان فایل ها هستند) دسترسی پیدا کنند، در نقطه‌ای از این درخت قرار می گیرند و چنانچه کاربر به هر یک از گره های درخت سیستم فایل نسبت داده شود، می تواند به تمامی مجموعه منابعی که زیر مجموعه این گره قرار دارند دسترسی داشته باشد. در شکل با فرض اینکه ما دو گروه اصلی یعنی دانشجویان و اساتید (Student , faculty) را داریم آنگاه به ازای هر یک از این گروه‌ها افراد منتسب به

@youseficlass

این گروه‌ها دارای گره‌های مربوط به خود هستند. به عنوان مثال در شکل مربوطه Professor Brown دارای یک دایرکتوری مربوط به خود می باشد که می‌تواند به صورت سلسله مراتبی در این دایرکتوری ، زیردایرکتوری مربوط به دروس یا Courses، مقالات یا Papers، امتیاز ویژه یا Grants و یا دایرکتوری مربوط به Committees یا در واقع گروه ها یا جلسات را برای خود ایجاد کند و مجدداً به صورت سلسله مراتبی در هر یک از این دارکتوری ها، زیردایرکتوری دیگری را برای خود ایجاد نماید. لذا در شکل نشان داده شده است که دایرکتوری courses دارای دو زیردایرکتوری، CS 101 و CS 105 است. بنابراین در یک ساختار درختی هر گره‌ی به تمامی اجزای زیر مجموعه خود دسترسی دارد. از مبحث مربوط به سیستم‌های عامل ۱ می‌دانید که علاوه بر ساختارهای درختی، ساختارهای یک سطحی، ساختارهای دو سطحی، ساختارهای مبتنی بر گراف‌های عمومی و ساختارهای مبتنی بر گراف‌های بدون حلقه، از دیگر ساختارهای مربوط به سیستم فایل ها هستند که هر یک دارای مزایا یا معایبی نسبت به یکدیگر هستند.

نصب (Mounting):

نکته حائز اهمیت در مورد سیستم‌های فایل مفهوم Mounting یا نصب است. استفاده از ساختارهای درختی در سیستم‌های فایل این اجازه را به سیستم می‌دهد که هر یک از دستگاه‌های سخت افزاری را به عنوان یک گره از سیستم فایل تعریف کنیم.



@youseficlass

به عنوان مثال، در شکل یک سیستم فایل با چندین گره ثابت وجود دارد و با فرض اینکه یک Floppy disk که شامل اطلاعات مربوط به فایل‌هاست وجود دارد. این دستگاه می تواند Floppy disk، CD، hard disk و یا هر دستگاه دیگری که اطلاعات فایل ها روی آن ذخیره می شود، باشد. این دستگاه می‌تواند با استفاده از عمل نصب (Mounting) به عنوان یکی از زیر شاخه‌های مربوط به Root directory یا درخت سیستم فایل، نصب گردد. و همانطور که در اینجا می بینیم با نصب Floppy بر روی زیر مجموعه گره b تمام اجزای این Floppy به عنوان گره هایی از این زیرمجموعه دیده خواهند شد و کاربر در هنگام استفاده از این سیستم فایل هیچ تفاوتی بین گره a که ممکن است یک دایرکتوری بر روی هارد دیسک باشد باگره b که شامل دایرکتوری‌های مربوط به فلاپی دیسک است قائل نخواهد شد و برخورد کاربر با هر دوی آنها یکسان خواهد بود. بنابراین عمل نصب (Mounting) یکی از قابلیتهایی است که می تواند در سیستم‌های مبتنی بر درخت و سیستم‌های فایل درختی مورد استفاده قرار گیرد.

سیستم عامل Unix یکی از سیستم‌های عاملی است که از این مفهوم استفاده فراوانی می نماید.(جهت کسب اطلاعات بیشتر در مورد مفاهیم مربوط به سیستم فایل‌ها به کتاب‌های مربوط به سیستم عامل (۱) مراجعه کنید.)

سیستم های عامل هم روند یا چند فرایندی (Multi-process)

در یک سیستم، همواره چندین فرایند بطور همزمان اجرا می شوند و یا ممکن است که برای رسیدن به یک هدف خاص با یکدیگر همکاری کنند. در نتیجه باید نحوه ارتباط و همکاری این فرایندها را با یکدیگر مشخص کنیم. به عبارت دیگر مشخص شود این فرایندها چگونه می توانند با یکدیگر داده رد و بدل کنند و یا در اصطلاح عام با یکدیگر صحبت کنند. در سیستم های عامل برای ارتباط فرایندها با یکدیگر ابزارهای متنوع و گوناگونی تعبیه شده است. از جمله این موارد سیستم های مبتنی بر message passing است که پیغام یا Message ابزاری برای ارتباط بین فرایندها خواهد بود.

حافظه مشترک یا Shared memory وسیله دیگری برای برقراری ارتباط بین فرایندهاست. موارد دیگری که می توان به آنها اشاره کرد استفاده از مفاهیم مربوط به وقفه‌ها و ارسال سیگنال بین فرایندهاست. یکی از راه های ارتباطی بین فرایندها که در سیستم عامل Unix و در برخی سیستم های عامل دیگر نیز مورد استفاده قرار می گیرد استفاده از مفهوم Pipe است. استفاده از Pipe باعث می شود که یک فرایند

@youseficlass

بتواند داده های مربوط به خود را به فرایند دیگر ارسال کند. Pipe بین دو فرایند قرار دارد و ارتباط بین دو فرایند را برقرار خواهد کرد. Pipe برای یک فرایند مشابه یک فایل است. بطوریکه فرایند داده های مربوط به خود را در این فایل می نویسد و فرایند دیگر داده های مربوطه را از این فایل خواهد خواند و بدینسان ارتباط بین دو فرایند برقرار خواهد شد.

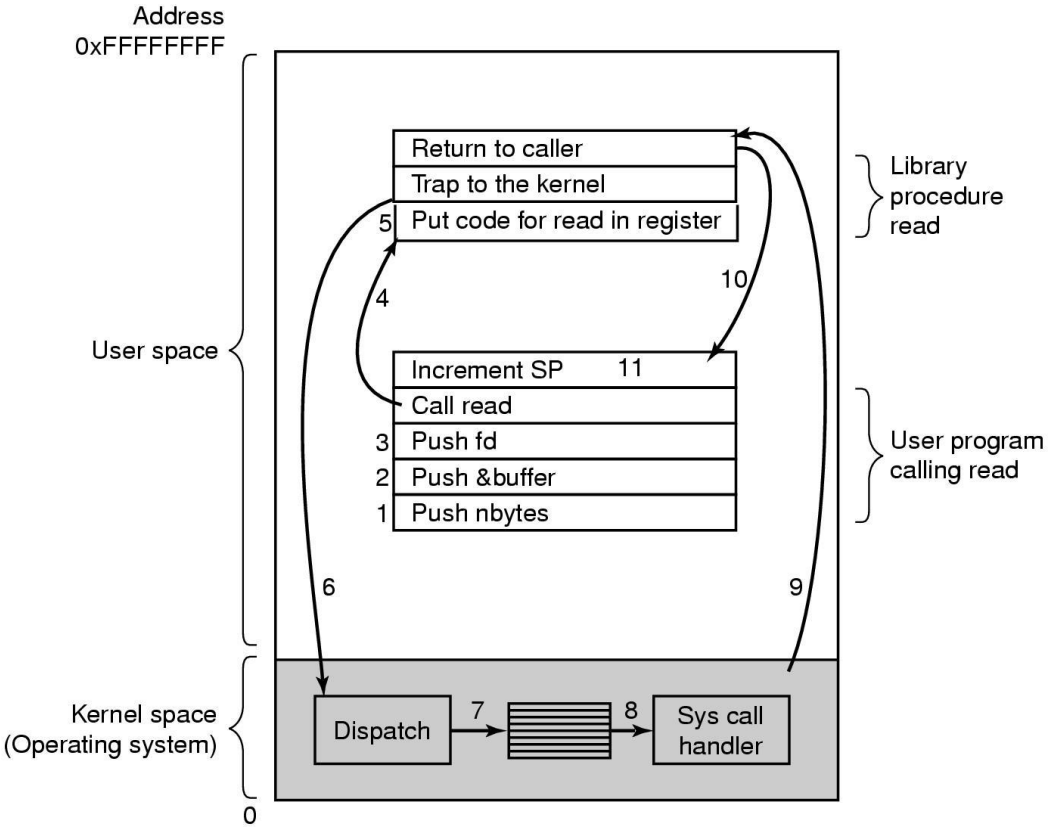
فراخوانی‌های سیستمی (system call) :

از دیگر مفاهیمی که در یک سیستم عامل مورد توجه قرار می‌گیرد مفهوم System call یا فراخوانی‌های سیستمی است.System call ابزاری است که از طرف سیستم عامل در اختیار کاربران قرار می‌گیرد تا با استفاده از آنها فراخوانی‌های سیستمی قابل انجام باشد. System call سرویسی است که سیستم عامل به کاربران می دهد و کاربران با استفاده از این سرویس می توانند برخی عملیاتی که انحصار آنها بر عهده سیستم عامل است درخواست نمایند و سپس سیستم عامل این عملیات را برای آنها انجام می دهد.

در مباحث مربوط به سیستم عامل و در طراحی سیستم عامل‌هایی که امنیت را به خوبی رعایت می کنند یک کاربر حق نخواهد داشت مستقیماً به سخت افزار دسترسی پیدا کرده و سخت افزار را تغییر دهد. بنابراین در چنین مواردی که نیاز به آنها وجود دارد برای جلوگیری از بهم ریختگی سیستم، در داخل سیستم عامل فراخوانی‌های سیستمی تعبیه می‌شود و این فراخوانی‌ها در اختیار کاربر قرار می گیرد و کاربر با استفاده از این فراخوانی های در اختیار، می‌تواند از سیستم عامل برخی از اعمال را درخواست کرده و پس از انجام این درخواست سیستم عامل کارهای مربوطه را انجام می دهد.

به طور کلی و در یک دید کلان می توان مراحل انجام شده در فراخوانی یک System Call را با یکدیگر مرور کرد. فرض کنید که یک فراخوانی سیستمی به نام read برای خواندن مجموعه ای از داده‌ها از یک فایل مشخص فراخوانی شده است.

@youseficlass



همانطور که در شکل نشان داده شده است اولین مرحله این System Call قراردادن آرگومان‌های مربوط به این فراخوانی سیستمی در پشته است. با توجه به شکل، مراحل ۱ ، ۲ و ۳، سه آرگمان مربوط به فراخوانی سیستمی را در پشته مربوط به سیستم، Push می‌کند، تا هنگامی که فراخوانی سیستمی فعال شود، بتواند از اطلاعات قرارداده شده در پشته سیستمی استفاده کرده و عملیات فراخوانی سیستمی را راهبری کند.

مرحله چهارم، فراخوانی زیر برنامه Read است که به عنوان یک فراخوانی سیستمی در سیستم تعبیه شده است. با فراخوانی این زیر برنامه که در شکل با call Read نشان داده شده است، مرحله پنجم مرحله‌ای خواهد بود که کدهای مربوط به این زیر برنامه در داخل registerهای مربوط به (CPU) نگهداری می شود تا مراحل مربوط به پردازش آغاز شود. با قرار داده شدن این داده‌ها در ثبات های مربوط به cpu

@youseficlass

عملیات اجرای فراخوانی سیستمی آغاز می شود. در مورد مرحله چهارم، ذکر این نکته ضروری است که برای اجرای این System call ، از جمله ثبات هایی که احتمالاً ممکن است مقدار بگیرند، می‌توان به ثبات IP به عنوان Instruction pointer و سایر ثبات هایی که برای واکشی دستورالعمل‌ها نیز مورد نیاز هستند اشاره کرد. پس از مقدارگیری ثبات ها، cpu آماده اجرای دستورالعمل مربوطه است.

مرحله بعد ارسال یک وقفه نرم افزاری یا یک Trap به سمت هسته سیستم عامل است. در این مرحله هسته سیستم عامل کنترل اجرا را بر عهده خواهد گرفت. زیرا فراخوانی سیستمی ، درخواست یک عملیات از سیستم عامل است و عملیات درخواست شده در Kernel یا هسته سیستم عامل انجام می شود. با ارسال وقفه نرم افزاری به Kernel، عملیات Dispatching انجام خواهد شد. در طی این عملیات فرایند مربوط به فراخوانی سیستمی وارد مجموعه فرایندهای فعال شده و کنترل را بر عهده می‌گیرد و دستورات مربوط به آن انجام خواهد شد. دستورات مربوط به فراخوانی سیستمی در داخل قسمتی به نام System call Handler یا مدیر فراخوان سیستمی قرار دارد. مرحله هشتم شامل اجرای دستورات مربوط به System call Handler است. System call Handler یا مدیر فراخوان سیستمی ، چیزی جزمجموعه‌ای از کدها نیست که پس از فراخوانی System call اجرای آنها امکانپذیرخواهد بود. مرحله نهم ، بازگشت به کسی خواهد بود که فراخوانی سیستمی را درخواست کرده است. در مرحله دهم اطلاعات مورد نیاز از پشته یا Stack حذف خواهد شد و بدینسان مراحل مربوط به مدیریت یک System call در یک سیستم، پایان خواهد یافت.

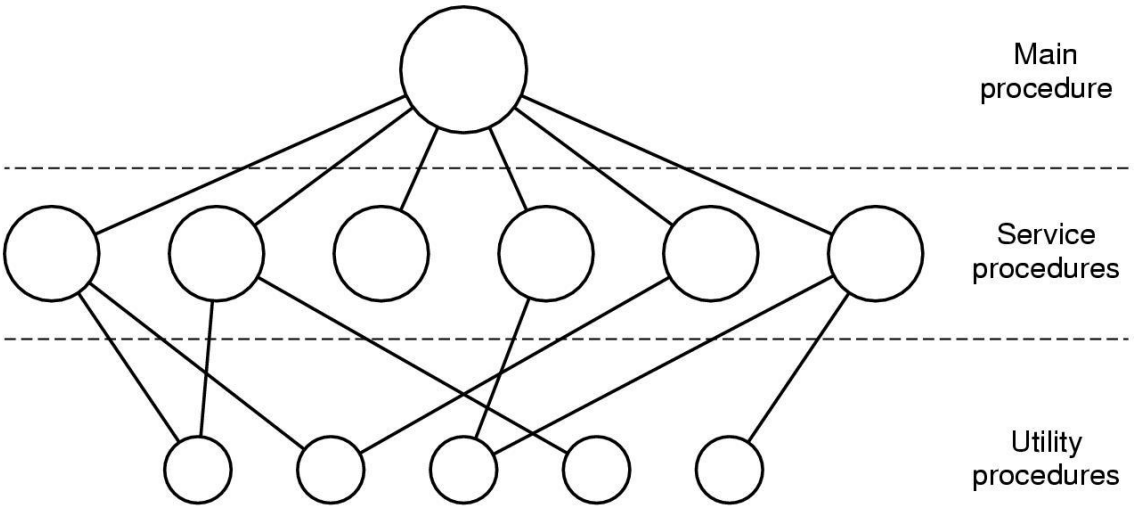
سیستم‌های Monolithic

متداولترین نوع سیستم‌های عامل و یا به عبارت دیگر معماری سیستم عامل را به سیستم‌های Monolithic یا سیستم‌های Simple نسبت می‌دهند که هر دو به معنای یک سطحی یا ساده بودن است. یک سیستم Monolithic سیستمی است که تقریباً دارای هیچ ساختاری نیست، به عبارت دیگر، سیستم عاملی است که شامل مجموعه‌ای از برنامه‌ها یا زیر برنامه‌هاست. این برنامه ها به گونه‌ای تنظیم شده اند که هر یک از آنها هنگامی که نیازمندی ایجاب کند، تنها می توانند یک برنامه یا زیربرنامه دیگر از این مجموعه را فراخوانی کنند. بنابراین مجموعه این زیربرنامه ها که امکان فراخوانی یکدیگر را دارند به عنوان سیستم

@youseficlass

عامل در نظر گرفته می شوند. در این نوع معماری تقریباً هیچ طرح خاصی برای چیدن برنامه ها رعایت نمی شود.

اگر چه سیستم‌های Monolithic یک سیستم ساده هستند و ساختاری ندارند اما می‌توان اندک ساختاری برای آنها درنظر گرفت. به عنوان مثال سرویس‌هایی که به وسیله سیستم های عامل آماده شده اند و دیگران می توانند از این سرویس ها استفاده کنند ساختار اندکی را برای این سیستم عامل ها ایجاد می‌کنند. برای استفاده از این سرویس‌ها کافی است که پارامترهای مربوط به زیربرنامه هایی که در اثر این سرویس ها فعال می شوند در یک مکان مشخص و خوش تعریف، قرار داده شود و سپس یک Trap یا وقفه نرم افزاری فعال گردد. در این سیستم‌های عامل هر System call با یک رویه سرویس عجین شده است.



به عنوان مثال اگر بخواهیم اندک ساختار مربوط به سیستم های Monolithic را نمایش دهیم شکل، این مسئله را نشان می‌دهد. همانطور که در این شکل می بینید در یک سیستم Monolithic تنها یک زیربرنامه اصلی وجود دارد که به نوعی در بالاترین سطح قرار گرفته است. تعدادی زیربرنامه سرویس وجود دارد که این زیربرنامه های سرویس، مدیریت فراخوانی های سیستمی را بر عهده خواهند داشت و برنامه اصلی که در سطح بالایی قرار دارد در هنگام نیاز، این زیربرنامه های سرویس را فراخوانی خواهد کرد. علاوه بر این دو نوع زیربرنامه که قسمت های اصلی یک سیستم عامل را تشکیل می‌دهند، سومین لایه زیربرنامه های

@youseficlass

کاربردی هستند. همانطور که در شکل نشان داده شده است این زیربرنامه ها می‌توانند توسط زیربرنامه های سرویس متفاوتی استفاده شوند و زیربرنامه های کاربردی به طور خاص برای هیچ سرویسی طراحی نشده‌اند. به عبارت دیگر زیربرنامه های کاربردی به اجرای دقیق تر و صحیح تر زیربرنامه‌های سرویس کمک می‌کنند.

معماری لایه‌ای سیستم‌های عامل

دومین معماری سیستم های عامل معماری لایه‌ای می‌باشد. در معماری لایه ای، سیستم عامل از چندین لایه تشکیل شده است که هر لایه تنها می‌تواند با لایه پایینی و بالایی خود ارتباط داشته باشد و ارتباط با لایه‌های دیگر مجاز نیست. به عنوان مثال می توان به یکی از ساده ترین سیستم‌های عامل یعنی سیستم عامل (The) اشاره کرد.

سیستم عامل THE (The Operating system) دارای ٦ لایه است. لایه صفر یا پایین ترین لایه، دارای وظیفه تخصیص cpu به فرایندها و انجام عملیات multi programming است. لایه ۱، لایه مدیریت حافظه است.لایه ۲، وظیفه انجام عملیات پردازشی و ارتباط بین فرایندها را بر عهده دارد.لایه ۳، مدیریت ورودی و خروجی را انجام می‌دهد و لایه ٤ برنامه‌های کاربران را اجرا می‌کند و در بالاترین لایه یا لایه سطح ۵، Operator یا کاربر سیستم قرار دارد.

در چنین سیستم عاملی لایه های مختلف فقط می توانند با لایه زیرین و بالایی خود ارتباط داشته باشند. به عنوان مثال در اینجا هیچگاه لایه ۵ که کاربر سیستم است نمی تواند درخواست مستقیمی به لایه صفر برای تخصیص Processor ارسال کند و حتماً این درخواست باید از طریق لایه های دیگر انجام شود. برای یک سیستم عامل کلی تر ممکن است لایه هایی با وظیفه مندی های دیگری تعریف شود و آنچه که در این شکل بررسی شد تنها ساختار یک سیستم عامل نویی بود.

نکته دیگر، مفهومی است که در برخی دیگر از سیستم های عامل مانند سیستم عامل Multics مورد استفاده قرار می گیرد. در سیستم عامل Multics به جای استفاده از مفهوم لایه‌ها از مفهوم حلقه‌ها استفاده می شود. در اصطلاح در این نوع سیستم های عامل می گویند ring به جای Layer استفاده می‌شود. وقتی

@youseficlass

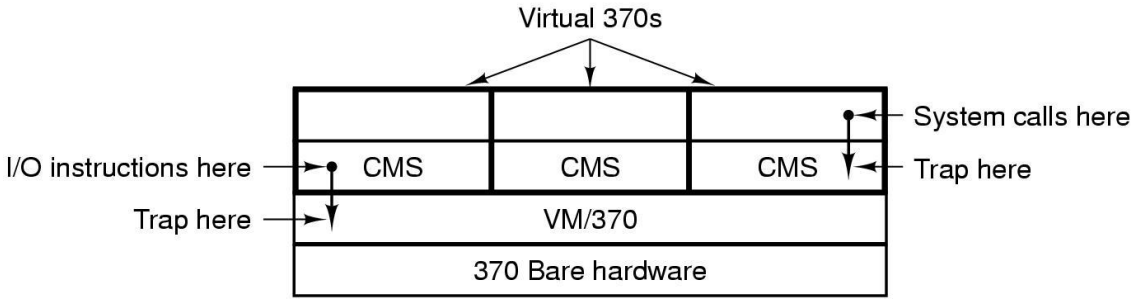
که شکل یک سیستم عامل لایه ای به یک سیستم عامل حلقه ای تغییر می کند آنگاه حلقه‌های داخلی به مراتب سطح امنیت بالاتری از حلقه های خارجی خواهند داشت و این مسئله باعث خواهد شد که دسترسی به لایه ها یا حلقه های داخلی با سختی بیشتری انجام شود و امنیت سیستم عامل افزایش یابد. در این حالت برای ارتباط بین حلقه ها معمولاً از Trap یا وقفه نرم افزاری استفاده می شود.

ماشین مجازی (Virtual Machine)

یکی از مفاهیمی که در سیستم های عامل و بخصوص در سیستم‌های عامل لایه ای مورد استفاده می باشد مفهوم Virtual Machine یا ماشین مجازی است. Virtual Machine یا ماشین مجازی مفهوم سطح بالاست که ممکن است در هر نوع سیستم عاملی مورد توجه قرار گیرد. Virtual Machine یک نوع نرم افزار است. به عبارت دیگر، برخلاف اسم آن که کلمه ماشین معمولاً تداعی کننده یک مفهوم سخت افزاری است در Virtual Machine، یک نرم افزاری داریم که این نرم افزار سعی می‌کند نقش یک ماشین را بازی کند. به عبارت دیگر Virtual Machine یک شبیه ساز است که عملیات شبیه سازی سخت افزاری را انجام می دهد. با تعریف Virtual Machine شما قادر خواهید بود که به طور همزمان چندین Virtual Machine را بر روی یک سخت افزار اجرا کنید. معمولاً Virtual Machine بر روی اصل سخت افزار و یا به عبارتی دیگر بر روی bare hardware یا سخت افزار لخت اجرا می شود و وظیفه آن شبیه سازی رفتار یک سخت افزار خاص است.

به عنوان مثال، می‌توان Virtual Machine مربوط به ماشین PC را بر روی یک ماشین Mainframe اجرا کرد و از این دیدگاه رفتار ماشین PC بر روی ماشین Mainframe شبیه سازی شده است. یکی از مزایای Virtual Machine در این است که معمولاً می‌توان چندین Virtual Machine را به طور همزمان بر روی یک ماشین اجرا کرد و این خود به این معنی است که می توان همزمان بر روی یک ماشین با استفاده از Virtual Machine سیستم های عامل متعدد را اجرا نمود.

@youseficlass



به عنوان مثال در شکل یک Virtual Machine از ماشین IBM370 نشان داده شده است. همان طور که در این شکل می بینید: Virtual Machine 370 بر روی hardware لخت یا سخت افزار اصلی اجرا شده است و اجازه انجام Multi Programming را داده است. به عبارت دیگر بر روی Virtual Machine370 برنامه های مختلف دیگری در لایه بالاتر در حال اجرا هستند. می‌توان بر روی یک ماشین Virtual Machineهای مختلفی را اجرا کرد و هر Virtual Machine از نظر لایه ای که بر روی آن قرار دارد مشابه سخت افزار یا hardware اصلی عمل می‌کند و تمام اجزاء مربوط به سخت افزار اصلی شامل هسته، I/O، Interrupt و موارد مشابه آن بر روی آن قرار دارد و هر کدام از این Virtual Machine ها می توانند یک سیستم عامل مخصوص خود را اجرا کنند.

علاوه بر این بر روی هر Virtual Machine می توان یک یا چند CMS یا Conventional Monitoring System یا سیستم های متداول کاربردی را اجرا کرد. اگر بخواهید نمونه ی جدیدتری از Virtual Machine ها را مورد بررسی قرار دهید، می توان از سیستم های مبتنی بر PC یاد نمود، که در این نوع سیستم ها نیز از Virtual Machine استفاده می شود. با توجه به مفهوم Virtual Machine آنگاه مفهوم دیگری در سیستم های عامل و در معماری سیستم های عامل مطرح می شود که آن را Exokernel می نامیم.

Exokernel

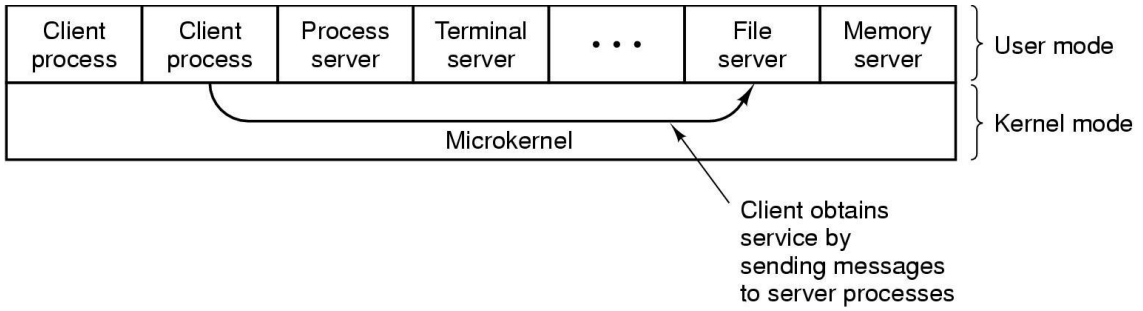
@youseficlass

Exokernel یک مفهوم نرم افزاری است. در یک سیستم ممکن است Virtual Machine های متفاوتی وجود داشته باشد. یک ماشین دارای منابع ثابتی است و هنگامی که تعدادی Virtual Machine بر روی یک سخت افزار اجرا می‌شوند، مبارزه برای در اختیار قرار گرفتن منابع ماشین شدت می گیرد. ممکن است Virtual Machine ها به عنوان یک نرم افزار برای دسترسی به منابع سخت افزاری ماشین رقابت کنند. Exokernel در سطح Kernel نرم افزاری است که وظیفه مدیریت منابع در قبال Virtual Machine ها را بر عهده دارد. به عبارت دیگر Exokernel در Kernel mode اجرا می شود و منابع را به Virtual Machine اختصاص می دهد و امنیت منابع را بررسی می کند. منظور از امنیت منابع در اینجا این است که یک Virtual Machine نتواند به منابعی که در اختیار Virtual Machine دیگر است دسترسی پیدا کند. وظیفه دیگر Exokernel، ردیابی منابعی است که در اختیار هر Virtual Machine است بدین معنا که بتواند تخصیص منابع را مدیریت کند.

بنابراین با استفاده از ترکیب Exokernel و Virtual Machine ها می‌توان بستری را فراهم کرد که بر روی یک ماشین بیش از چندین سیستم عامل به طور همزمان کار کنند.

Client- Server

یکی دیگر از مدل‌ها یا معماری‌های مربوط به سیستم عامل را می توان در قالب سیستم‌های عامل Client-Server یا سرویس گیرنده – سرویس دهنده در نظر گرفت. در یک سیستم عامل Client-Server از مفهوم Micro Kernel استفاده می شود. در چنین سیستم عاملی تمامی فعالیت‌های مربوط به سیستم عامل، در قالب فرایندهای سرویس دهنده انجام می‌شود. به عبارت دیگر هر فعالیت و یا هر سرویسی که توسط سیستم عامل ارائه می شود دارای یک سرویس دهنده است.

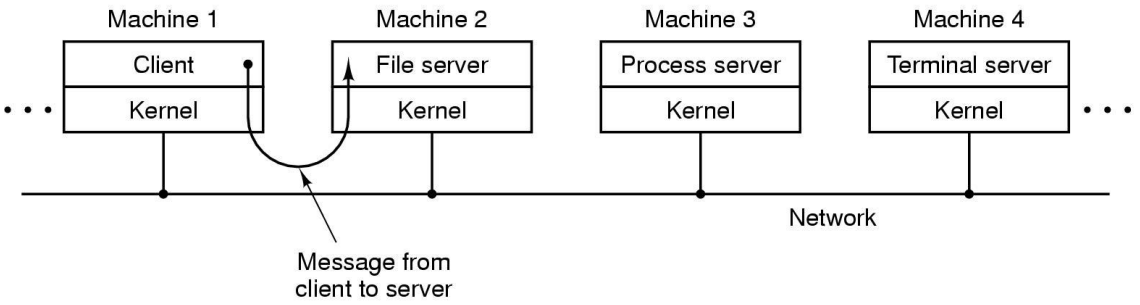


@youseficlass

مثلاً همان طور که در شکل می بینید در سطح سیستم عامل، سرویس دهنده فرایندها، سرویس دهنده ترمینال ها، سرویس دهنده فایل ها، سرویس دهنده حافظه و سرویس دهنده های دیگری که مفاهیم نرم افزاری هستند، وجود دارد.

فرایندهایی که این سرویس‌ها را نیاز دارند درخواست خود را برای سرویس دهنده مربوطه ارسال می کنند. بنابراین فرایندهای سرویس گیرنده یا برنامه های کاربردی، برنامه‌هایی هستند که چنانچه نیاز به زمان بندی دارند درخواست مربوط به خود را به سرویس دهنده مربوط به زمان بندی فرایندها ارسال می کنند. اگر نیاز به عملیات فایل دارند درخواست مربوطه را به سرویس دهنده فایل ارسال می کنند. همان طور که در شکل نشان داده شده است معمولاً چنین خواهد بود که فرایندهای سرویس دهنده همگی در مد کاربری انجام می‌شوند و هیچ کدام در مد هسته قرار نمی‌گیرند. در نتیجه در سیستم های مبتنی بر Client و Server ، مد هسته، مد ریز هسته یا Micro kernel است. Micro kernel تنها کاری که در اینجا برای ما انجام می دهد این است که ارتباط بین فرایندهای سرویس دهنده و سرویس گیرنده را نظارت می کند. این نظارت از این جهت خواهد بود که آیا این سرویس گیرنده اجازه درخواست چنین سرویسی را دارد یا نه؟ وموارد از این قبیل.

بنابراین این مسئله باعث می شود که در سطح Kernel پردازش چندانی انجام نشود و هسته به ریز هسته تبدیل شود. یکی از مزایایی که برای سیستم های Client-Server قائل هستیم، این است که از این مفهوم می توان بر روی ماشین های توزیع شده استفاده نمود.



@youseficlass

در این شکل همانطور که می بینید هر یک از سرورها و Clientها می توانند بر روی ماشین های مختلفی قرار گیرند و ارتباط بین آنها از طریق پیغام باشد. ساختار نشان داده شده در این شکل بسیار مشابه ساختار و معماری یک سیستم عامل Client-Server است.

@youseficlass