www.ElitesJournal.ir

مجله نخبگان علوم و مهندسی

Journal of Science and Engineering Elites

ISSN 2538-581X

جلد ۲- شماره ۱- سال ۱۳۹۶

مروری بر مدیریت فرایند در سیستم عامل های توزیع شده

مصطفی جهان پور

کارشناسی ارشد مهندسی کامپیوتر،گرایش نرم افزار، دانشگاه آزاد اسلامی بروجرد

Mostafa9898a@gmail.com

ارسال: اسفند ماه ۹۵ پذیرش: اردیبهشت ماه ۹۶

خلاصه

سیستم عامل های توزیع شده بخش بسیار مهمی از کار طراحی و اجرای نرم افزار برای مدیریت سیستم هایی حاوی تعداد زیادی پردازنده را تشکیل می دهند، از طرفی پژوهش ها در حوزه های تحقیقات علوم کامپیوتر پیرامون رایانش توزیع شده در چند دهه گذشته به میزان قابل توجهی افزایش یافته است. یکی از موضوعاتی که در این گونه سیستم عامل ها مطرح می شود بحث مدیریت فرایند است که برای مدیریت فرایند در یک سیستم عامل توزیع شده مکانیزم هایی وجود دارد که امکان انتقال فرایند، دانلود، اشکال زدایی از راه دور، حالت برداری و شبیه سازی راه دور سیستم عامل را ایجاد می کند. امکانات مدیریت فرایند توسط هسته سیستم عامل توزیع شده قابل تحقق است. در این مقاله مروری داریم بر سیر تکامل سیستم عامل های توزیع شده، نحوه مدیریت فرایند، مهاجرت، خوشه بندی، نحوه تراکنش پیام و سایر مباحث پیرامون سیستم عامل توزیع شده، همچنین یک نمونه از این گونه سیستم عامل ها به نام آموبا (آمیب) را مد نظر قرار داده و مدیریت فرایند، ارتباطات درون فرایندی و مکانیزم های کارآمد بودن آن را مورد بررسی قرار می دهیم.

کلمات کلیدی: سیستم عامل توزیع شده، مدیریت فرایند، آموبا (آمیب)، مهاجرت.

۱. مقدمه

روند تکامل و پیشرفت کامپیوترها در چند دهه اخیر به گونه ای بوده است که ماشین های بزرگ و یکپارچه که اجازه دسترسی کامل آن تنها به یک کاربر داده می شد جای خود را به ماشین هایی داده است که به کاربران متعددی اجازه دسترسی هم زمان را می دهند. با مرور زمان و پیشرفت معماری کامپیوترها، ماشین ها کوچکتر شدند و مجددا به کامپیوترهای تک کاربره تبدیل گردیده و کامپیوتر شخصی یا (PC)[1] نامیده شدند و نهایتا تا به امروز با متصل نمودن این کامپیوترهای شخصی به یک سیستم مرکزی به منظور استفاده از دیسک ها، پرینتر ها، پردازنده ها و... رایانش توزیع شده به وجود آمده است. از اواخر دهه ۱۹۷۰ میلادی تحقیقات پیرامون رایانش توزیع شده، علی الخصوص سیستم عامل های توزیع شده میزان قابل توجهی از پژوهش ها در سطوح عالی را به همراه داشته و در حال حرکت سریع به سمت حوزه های تحقیقات علوم کامپیوتر در محیط دانشگاه می باشد. در دهه های پیش رو انتظار می رود هزینه تراشه های CPU[2] به کاهش خود ادامه داده و نهایتا منجر به سیستم هایی شود که

[1] Personal Computer
[2] Central Processing Unit

حاوی تعداد زیادی از پردازنده ها می باشند و طراحی و اجرای نرم افزار برای مدیریت فرایندها و استفاده از تمام نیروی رایانش آن ها بخشی بسیار مهم محسوب می گردد که این امکانات جهت مدیریت فرایند توسط هسته سیستم عامل توزیع شده قابل تحقق است و آن ها را می توان توسط سرویس های فضای کاربری استدلال نمود: سرویس اشکال زدا، سرویس توازن بار، سرویس نمونه سازی یونیکس، سرویس بازرسی و ... [۳-۴].

سیستم عامل های توزیع شده مختلفی را می توان نام برد مثلا K۲یک نمونه از این گونه سیستم عامل های توزیع شده ی یکپارچه است که معماری آن به گونه ای است که مشکلات مدیریت منابع در شبکه های ناهمگن را مرتفع کرده و کاربرد توزیع شده دارد [۹]. اما یکی از مطرح ترین سیستم عامل های توزیع شده سیستم عامل آموبا یا آمیب [1] است. این سیستم عامل با عملکردی بالا طراحی گردیده جهت تعامل بین کلاینت و سرور با استفاده از مدل RPC [2] به معنای رویه فراخوانی از راه دور. در سال ۱۹۹۰ در دانشگاه Vrije آمستردام گروهی تحت هدایت پروفسور تنن باوم گرد هم آمدند و تا کنون در مورد سیستم عامل های توزیع شده تحقیق می کنند. نتیجه این تحقیقات سیستم عامل توزیع شده آموبا است که برای محیط هایی با تعداد کامپیوتر زیاد طراحی شده است البته این سیستم عامل برای دانشگاه ها و موسسات تحقیقاتی رایگان است. آموبا یک سیستم عامل چند منظوره توزیع شده است. این سیستم عامل می تواند از چندین ماشین بهره بگیرد و با آن ها طوری رفتار کند که انگار یک سیستم واحد و یکپارچه است. به طور کلی، کاربران از تعداد و محل پردازنده هایی که برای اجرای دستورات وی به کار گرفته می شود، مطلع نیستند. همچنین اطلاعات و محل سرورهای فایلی که فایل های آن ها در آن ذخیره می شود نیز نامشخص است. از دیدگاه کاربر، آموبا درست همانند یک سیستم عامل سنتی اشتراک زمانی است. آموبا یک پروژه ناتمام است و هنوز در مرحله تکمیل به سر می برد. این سیستم عامل به عنوان بستری برای تحقیق و توسعه کد در سیستم های موازی و زبان ها و پروتکل های مرتبط با آن به کار می آید. همچنین این سیستم می تواند یونیکس را شبیه سازی کند و ظاهری درست همانند یونیکس دارد اما به عنوان جایگزینی برای یونیکس نیست و تنها عملکردی مشابه دارد. این سیستم عامل برای دانشجویان و محققانی که می خواهند کد منبع سیستم عامل را مشاهده و طریقه عملکرد آن را از نزدیک ببینند، بسیار مناسب است. آموبا همچنین برای برنامه نویسی به روش توزیع شده (چند کاربر به طور مجزا روی چند پروژه مختلف کار کنند) و سیستم های موازی (یک کاربر از ۵۰ پردازنده استفاده کند تا شطرنجی را بتواند به طور موازی بازی کند) استفاده می شود. البته ایجاد نرم افزارها برای این سیستم عامل ساده است، همانند دستور make در یونیکس، یک دستور به نام amake وجود دارد [۱۸, ۲۰, ۲۳].

۲. اهداف طراحی سیستم عامل آموبا

هدف اصلی از طراحی آموبا به شرح زیر است: توزیع به معنای اتصال چندین ماشین به یکدیگر، موازی سازی به معنای اجرای یک کار روی چندین پردازنده، شفافیت به معنای کلکسیون کردن کامپیوترهای مختلف و نمایش آن به صورت یک سیستم واحد، بازدهی که منظور از آن دستیابی به تمام موارد بالا، با کیفیت مناسب است [۲۳].

۱.۲. توزیع[3]

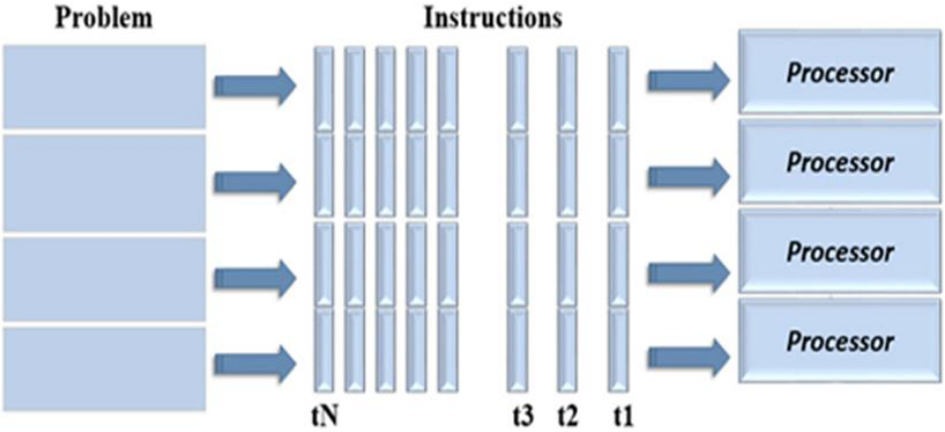
یعنی مجموعه ای از کامپیوترهای محاسبه گر و خود مختار که هر کدام می توانند چندین کاربر داشته باشند و برای کاربران به نظر می رسد که این مجموعه کامپیوترها واحد هستند ، و این سیستم همگرا است، برخی منابع در مورد توزیع شدگی سیستم عامل توزیع شده بیان می کنند که یک سیستم عامل توزیع شده مجموعه ای است از کامپیوترهای مستقل که کاربران آن را به

[1] Amoeba
[2] Remote Procedure Call
[3] Distribution

صورت یک سیستم واحد می بینند [۸، ۱۶] به عنوان نمونه یک معماری توزیع شده برای سیستم عامل های چند هسته ای پیشنهاد شده که مزیت اصلی آن در دسترس بودن منابع می باشد که در پروژه های صنعتی و تحقیقات و در سطح شرکت ها برای صدور گواهینامه ایمنی برنامه های کاربردی استفاده می شود [۱] آموبا سیستمی توزیع شده است و در آن چندین ماشین که به یکدیگر متصل هستند را کنترل و مدیریت می کند. نیازی نیست که این ماشین ها همه از یک نوع باشند. این ماشین ها می توانند در یک شبکه LAN به یکدیگر متصل شوند. آموبا از پروتکل شبکه قدرتمند FLIP استفاده می کند. اگر یک ماشین آموبا بیشتر از یک رابط شبکه داشته باشد، به طور خودکار به عنوان مسیریاب بین چند شبکه به کار گرفته خواهد شد و شبکه های LAN مختلف را به یکدیگر متصل خواهد کرد [۲۳].

۲.۲. موازی سازی[1]

اگر یک کار را روی چندین پردازنده هم زمان انجام دهیم به عنوان مثال وقتی یک مساله به بخش های مجزایی شکسته شود که بتوانند به صورت هم زمان حل شوند و هر بخش نیز خودش به دنباله ای از دستورالعمل ها شکسته شود، دستورالعمل های هر بخش به صورت هم زمان روی پردازنده های مختلفی اجرا می شوند. پردازش موازی منجر به بهره برداری بهتر از سیستم می شود [۶، ۲۲]. آموبا سیستمی موازی است. این یعنی می توان یک کار یا یک برنامه را با چندین پردازنده مختلف به طور موازی انجام داد تا سرعت بیشتری در اجرای آن به دست آورد. هر چه تعداد ریزپردازنده های موجود در سیستم بیشتر باشد، نتیجه بهتری حاصل می شود. به این مجموعه از ریزپردازنده ها، موتور محاسبه گفته می شود[۲۳].



شکل ۱. شکسته شدن مساله به بخش های مجزا و حل شدن به صورت هم زمان روی پردازنده های مختلف

۳.۲. شفافیت[2]

یعنی نمایش سیستم توزیع شده به صورت یک کامپیوتر واحد که اقسام مختلفی دارد[۸] و یکی از اهداف کلیدی محسوب می شود. نیازی نیست کاربر از تعداد یا محل پردازنده ها آگاه باشد و یا حتی بداند فایل هایش در کجا ذخیره شده. همچنین مشکلاتی چون تکرار فایل ها در نقاط مختلف به طور خودکار رفع می شود و نیازی به دخالت دستی کاربر نیست. به عبارت دیگر، کاربر به یک ماشین متصل نمی شود و به یک سیستم کلی متصل می شود. مفهومی به نام کامپیوتر شخصی وجود ندارد. وقتی کاربر به ماشین متصل می شود، کاربر دستور اتصال به سروری را وارد نمی کند که از چندین پردازنده بهره بگیرد. از دید کاربر کل سیستم مثل یک سیستم اشتراک زمانی است[۲۳].

---

[1] Parallelism
[2] Transparency

۴.۲. بازدهی

در سیستم های عامل، پایداری و بازدهی همواره یک دغدغه کلیدی بوده است. بنابراین تلاش های زیادی برای رفع این دغدغه ها انجام شده است. مکانیزم پایه ارتباطات بهبود های زیادی داشته است و بر اساس همین تلاش ها، پیغام های ارسالی و پاسخ های دریافتی در کمترین تاخیر ممکن ارسال و دریافت می شوند. همچنین امروزه امکان انتقال داده ها با پهنای باند بالا میان کامپیوترها رواج یافته است بلوک های داده پایه اصلی زیرسیستم ها و برنامه های با بازدهی بالا در آموبا هستند [۲۳].

۳. معماری سیستم

از آن جایی که سیستم های توزیع شده و موازی از سیستم های کامپیوتر شخصی کاملا متفاوت هستند، شایان ذکر است که در ابتدا نوع پیکربندی سخت افزاری که برای استفاده از آموبا مورد نیاز است، (هرچند امروزه گرایش به مکانیزه کردن جنبه های مختلف مدیریت اجزای سیستم توزیع شده وجود دارد.) را باید شرح داد. یک سیستم عادی آموبا از سه کلاس اصلی عملکردی در هر ماشین تشکیل می شود [۱۲, ۲۳].

۱.۳. اولین کلاس

هر کاربر یک دستگاه برای استفاده و اجرای رابط کاربری خود دارد، این رابط کاربری مبتنی بر سیستم پنجره ای X است. این ایستگاه می تواند یک ایستگاه کاملا معمولی باشد، یا یک پایانه بسیار خاص X این دستگاه کاملا به اجرای رابط کاربری بر می گردد و کاری به باقی سیستم ها ندارد.

۲.۳. دومین کلاس

یک سبد از پردازنده ها وجود دارند که به طور خودکار به هر کاربر تخصیص داده می شوند. این پردازنده ها می توانند بخشی از یک کامپیوتر با چند پردازنده و یا بخشی از یک شبکه با چند کامپیوتر باشند، حتی می توان مجموعه ای از مادربوردها را به هم متصل کرد و این سیستم عامل را روی آن اجرا کرد. به طور معمول هر پردازنده چندین مگابایت حافظه شخصی دارد، این حافظه نیازی هم نیست حافظه به اشتراک گذاشته شده باشد، چرا که هر پردازنده می تواند هر فضایی را که می خواهد تهیه کند. البته حافظه اشتراکی برای پردازنده ها ممنوع نیست. ارتباطات به این طریق انجام می شود که بسته ها از طریق شبکه LAN[۱] ارسال می شود و تمام پردازش ها در سبد پردازنده ها انجام می شود.

۳.۳. سومین کلاس

سرورهای مختص یک کار خاص وجود دارند، مثلا سرورهایی وجود دارند که به آن ها فایل سرور می گویند و کارشان فقط در مورد فایل ها و دیسک ها است. این سرورها همیشه در حال اجرا هستند. این پردازنده ها می توانند متعلق به سبد پردازنده ها باشند یا به طور مستقل فعالیت کنند. تمام این مولفه ها باید از طریق یک شبکه سریع به یکدیگر متصل شوند. در حال حاضر تنها شبکه های اترنت پشتیبانی شده است اما در آینده دیگر شبکه ها نیز پیاده سازی می شوند.

۴. مدل های سیستم های محاسبات توزیع شده

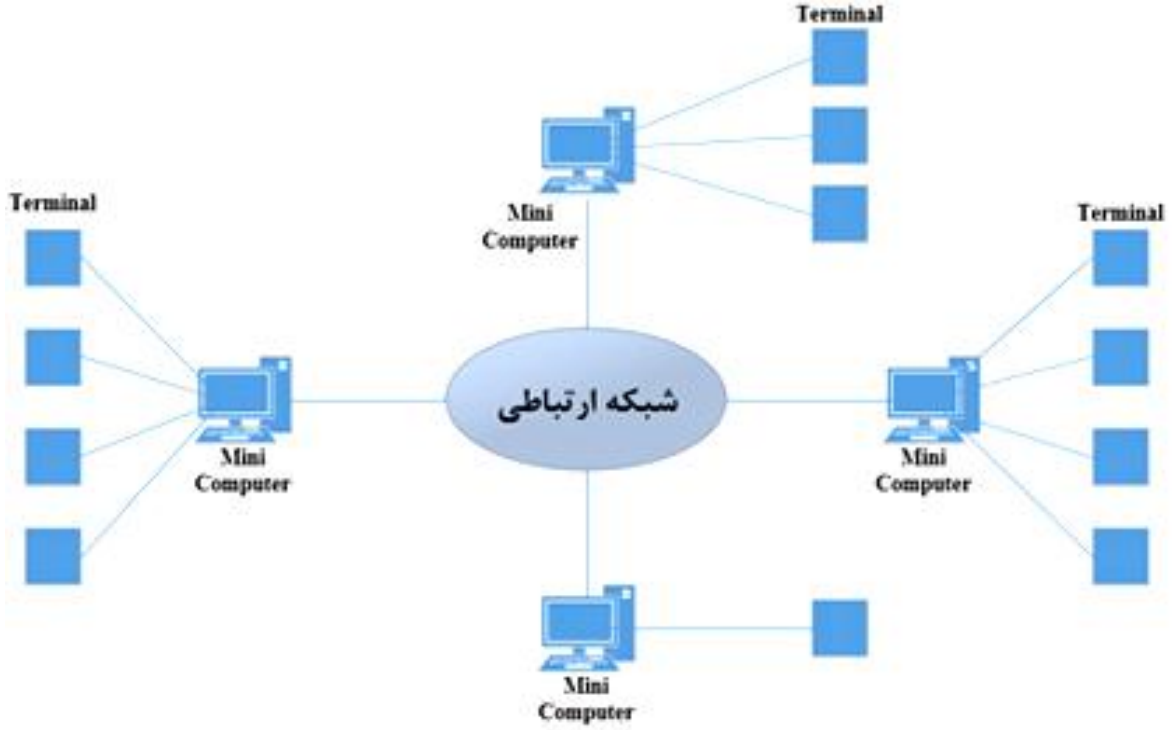
مدل های گوناگونی برای ساخت سیستم عامل های توزیع شده استفاده می شود. این مدل ها را می توان در حالت کلی به ۵ دسته تقسیم کرد: ۱. مینی کامپیوتر، ۲. ایستگاه کاری، ۳. ایستگاه کاری- سرویس دهنده، ۴. اشتراک پردازنده، ۵. ترکیبی.

۱.۴. مدل مینی کامپیوتر

مدل مینی کامپیوتر یک بسط ساده از سیستم اشتراک زمانی مجتمع است. یک سیستم محاسبات توزیع شده که براساس این مدل پیاده شده است متشکل است از چند مینی کامپیوتر (می تواند سوپر کامپیوتر بزرگ هم باشد) که توسط یک شبکه

[۱] Local Area Network

ارتباطی به هم متصل شده اند. هر مینی کامپیوتر به طور معمول چند کاربر دارد که به طور هم زمان در سیستم ثبت شده اند. در این نوع چندین ترمینال به مینی کامپیوتر متصل شده اند. هر کاربر که در یک میتی کامپیوتر ثبت شده است به مینی کامپیوترهای دیگر دسترسی دور دارد. شبکه به کاربر این اجازه را می دهد که به منابع دور که روی یک کامپیوتر جدا از کامپیوتری که کاربر در آن ثبت نام کرده، دسترسی داشته باشد. مدل مینی کامپیوتر در مواقعی استفاده می شود که نیاز به اشتراک منابع با کاربران دور مورد نیاز باشد.



شکل ۲. ARPANET[1] **جدید، یکی از نمونه هایی است که براساس مدل مینی کامپیوتر پیاده سازی شده است**

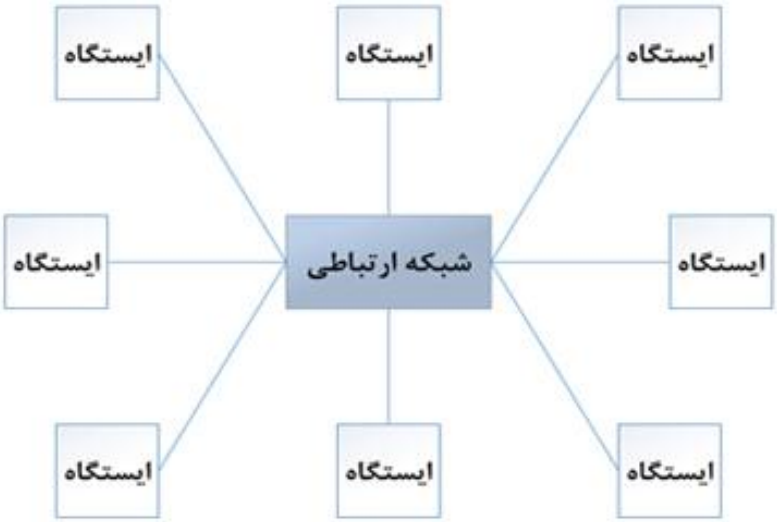
۲.۴. مدل ایستگاه کاری

یک سیستم محاسبات توزیع شده که بر پایه مدل ایستگاه کاری بنا نهاده شده متشکل است از چند ایستگاه کاری که توسط یک شبکه ارتباطی به هم متصل شده اند. یک شرکت و یا یک دپارتمان دانشگاهی ممکن است چندین ایستگاه کاری داشته باشد که در یک ساختمان و یا مجتمع پخش شده اند. هر کامپیوتر مجهز به دیسک خودش است برای یک کاربر سرویس دهی می کند. هدف اصلی در مدل ایستگاه کاری اتصال ایستگاه های کاری توسط یک شبکه محلی پرسرعت است و به این طریق ایستگاههای کاری بیکار ممکن است توسط کاربرانی که ایستگاه های کاری دیگر ثبت نام کرده اند برای انجام کارهایشان مورد استفاده قرار گیرند. البته این کار در حالتی اتفاق می افتد که ایستگاه های کاری دیگر قدرت کافی را برای پردازش کارخود، در کامپیوتر خود را نداشته باشند. پیاده سازی این مدل به آن سادگی که در دید اول به نظر می رسد، نیست، چرا که باید چندین مساله را حل کرد. اولا سیستم چگونه یک ایستگاه کاری بیکار را تشخیص خواهد داد. دوما چگونه یک پردازه برای اجرا از یک ایستگاه کاری، برای اجرا به یک ایستگاه کاری دیگر منتقل خواهد شد. سوما برای یک پردازهٔ خارجی، چه چیزی اتفاقی می افتد وقتی که کاربری به یک سیستم بیکار که در حال اجرای این پردازه خارجی بود، ثبت نام کند.

برای مسائل اول و دوم روش های ساخت یافته ای وجود دارد. ولی برای مساله سوم روش هایی ابداعی وجود دارد که به این شرح هستند. روش اول این است که به پردازه خارجی این اجازه داده شود تا منابع ایستگاه کاری را با پردازه کاربری که ثبت نام کرده به اشتراک بگذارد. روش دوم این است که پردازه خارجی از بین برود. ولی یک مشکل به وجود می آید این است

[1] Advanced Research Projects Agency Network

که پردازهٔ ایستگاه کاری دور، گم می شود. روش سوم مهاجرت دادن پردازهٔ خارجی به ایستگاه کاری صاحب آن است و اجرا در آنجا ادامه خواهد یافت. پیاده سازی این روش مشکل است چرا که سیستم باید از مفاهیم مهاجرت فرایند ها، پشتیبانی کند.



شکل ۳. شبکه ارتباطی و ایستگاه ها

۳.۴. مدل ایستگاه کاری– سرویس دهنده

مدل ایستگاه کاری[۱]، مدلی بر پایه ایستگاه های کاری شخصی بود که در آن هر یک دارای دیسک و سیستم فایلی مخصوص به خود بودند. به ایستگاه های کاری که دارای دیسک مخصوص به خود هستند (ایستگاه کاری دیسک دار) و به ایستگاه هایی که دیسک مخصوص به خود را ندارند، (ایستگاه کاری بی دیسک) اطلاق می گردد. با ازدیاد شبکه های پر سرعت، ایستگاه های کاری بی دیسک در محیط های شبکه زیادتر از ایستگاه های کاری دیسک دار مورد استفاده قرار گرفته اند، که این باعث شده که مدل ایستگاه کاری - سرویس دهنده، زیادتر از مدل ایستگاه کاری برای تولید و ساخت سیستم های توزیع شده مرد استفاده قرار گیرد. همانطور که در شکل قابل ملاحظه است سیستم های مبتنی بر مدل ایستگاه کاری–سرویس دهنده، شامل چند مینی کامپیوتر و چند ایستگاه کاری هستند که توسط یک شبکه ارتباطی به هم متصل شده اند. خیلی از این ایستگاه های کاری، از نوع بی دیسک هستند ولی ممکن است بعضی از آن ها از نوع دیسک دار باشند. ذکر این نکته مهم است که وقتی از ایستگاه های کاری بی دیسک در شبکه استفاده می شود سیستم فایلی که توسط این سیستم ها استفاده می شود باید توسط ایستگاه کاری دیسکدار پشتیبانی شود. و یا این که باید یک مینی کامپیوتر هم برای ذخیره سازی فایل ها در نظر گرفته شود. در این مدل، مینی کامپیوترها برای این مقاصد مورد استفاده قرار می گیرند:

اول: یک یا چند مینی کامپیوتر برای پیاده سازی سیستم فایلی

دوم: مینی کامپیوترهای دیگر برای ارائه سرویس های دیگر مثل سرویس های بانک اطلاعاتی و یا سرویس های چاپ. از این رو هر مینی کامپیوتر به صورت یک ماشین سرویس دهنده برای فراهم آوردن انواع متفاوت سرویس ها استفاده می شود. این مدل دارای مزایایی نسبت به مدل ایستگاه کاری است که به شرح زیر هست:

۱- در حالت عمومی، پیاده سازی آن ارزان تر و به صرفه تر است.

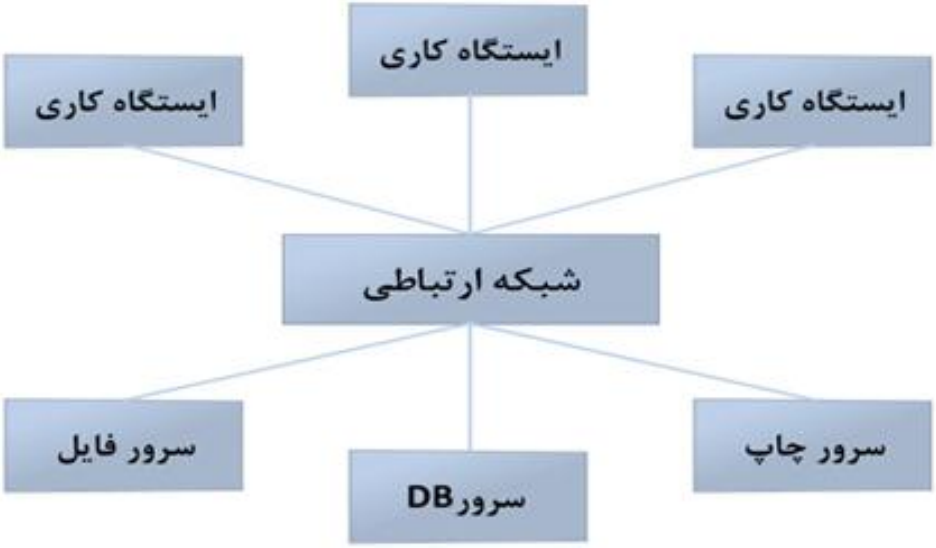
۲- از دیدگاه نگهداری سیستم، سیستم های بی دیسک بر سیستم های دیسکدار ارجحیت دارند (تعمیر، پشتیبان گیری و...)

[۱] Workstation

۳- در مدل ایستگاه کاری- سرویس دهنده، از آنجایی که همه فایل ها توسط سرور فایل مدیریت می شود، کاربران این قابلیت انعطاف را دارند که بتوانند از هر ایستگاه کاری بدون توجه به این که کدام کاربرد درست ثبت نام کرده و کدام ایستگاه در دسترس است، استفاده کنند و دسترسی به اطلاعات در حالت کلی به یک صورت خاص انجام می گیرد.

۴- در مدل مذکور از روش درخواست- پاسخ برای استفاده از سرویس ها استفاده می شود و پیاده سازی این روش به سختی پیاده سازی روش مهاجرت فرایندها نمی باشد. پروتوکل درخواست- پاسخ با عنوان مدل مشتری- سرویس دهنده در ارتباطات شناخته شده است.

۵- کاربر ضامن زمان پاسخ است چرا که سیستم برای اجرای فرایندهای خارجی استفاده نمی شود.



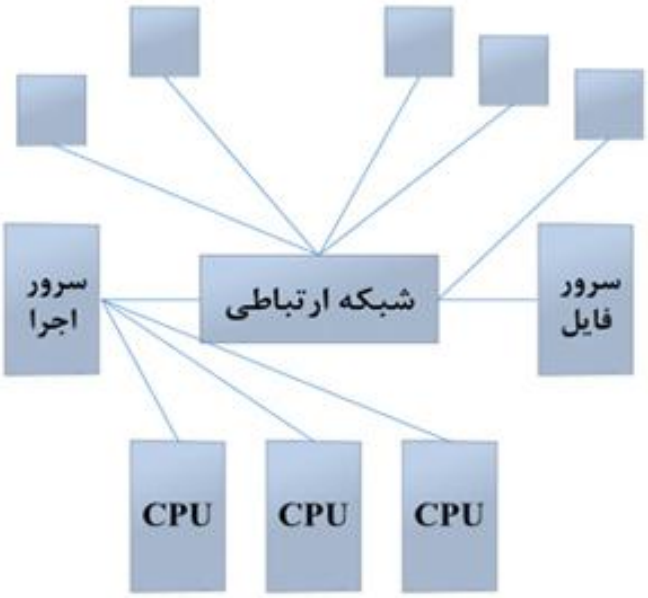
شکل ۴. شبکه ارتباطی بین ایستگاه های کاری و سرور ها

۴.۴. مدل اشتراک پردازنده یا ائتلاف پردازنده

مدل اشتراک پردازنده نیازمند این ملاحظه است که در اکثر اوقات یک کاربر نیازی به قدرت پردازشی ندارد ولی در یک لحظه برای مدت کوتاهی ممکن است نیاز به قدرت پردازشی خیلی بالا پیدا کند. بنابراین برخلاف مدل ایستگاه کاری- سرویس دهنده که برای هر کاربری، یک پردازنده داشتیم در مدل اشتراک پردازنده، پردازنده های سیستم به هم پیوند می خورند تا میان کاربران وقتی که یکی از آن ها نیازمند پردازش شد به اشتراک گذاشته شود. این ائتلاف از پردازنده ها شامل تعداد زیادی از ریز کامپیوترها و مینی کامپیوترها (که ممکن به صورت همگن یا ناهمگن باشند) هست که به شبکه پیوند خورده اند. هر پردازنده موجود در ائتلاف، دارای حافظه مخصوص به خود برای اجرا و بارگذاری یک برنامه سیستمی یا یک برنامه کاربردی و یا یک برنامه کاربردی از سیستم محاسبات توزیعی است. همانگونه که در شکل قابل مشاهده است در مدل ساده ائتلاف پردازنده ها، پردازنده های موجود در ائتلاف هیچ ترمینالی ندارند که به طور مستقیم به آن ها متصل شده باشد و کاربران از طریق ترمینال هایی که بوسیله دستگاههای خاصی به شبکه متصل هستند. به این پردازنده ها دسترسی دارند. مثل ترمینال های X. یک سرور ویژه که سرور اجرا نام دارد، پردازنده های موجود در ائتلاف را در زمان نیاز اختصاص می دهد و در حالت کلی آن ها را مدیریت می کند. وقتی که کاربری، کاری را برای اجرا تسلیم می کند، سرور اجرا، تعداد لازم از پردازنده ها را به طور موقت برای اجرای کار این کاربر به وی اختصاص می دهد.

در مقایسه با مدل ایستگاه کاری- سرویس دهنده، مدل ائتلاف پردازنده امکان استفاده بهتری از پردازنده های موجود در سیستم محاسبات توزیع شده را فراهم می آورد. علت این است که در این مدل همه قدرت پردازشی سیستم برای کاربرانی که در حال حاضر ثبت نام کرده اند قابل استفاده است در حالی که در مدل ایستگاه کاری- سرور چنین چیزی صادق نیست چرا

که در مدل فوق الذکر ممکن است در بعضی از اوقات بعضی از ایستگاهها بیکار باشند ولی نمی توان از آن ها برای اجرای کارهای کاربران دیگر سیستم استفاده نمود. مدل ائتلاف پردازنده برای برنامه هایی که تعامل زیادی دارند زیاد کارآیی ندارد مخصوصاً برنامه هایی که از سیستم های گرافیکی یا پنجره ای استفاده می کنند و این همه به خاطر پائین بودن سرعت رسانه های انتقالی شبکه است. و برای این کار مدل ایستگاه کاری- سرور مناسب تر است [۸ و۲۱]



شکل ۵. مدل اشتراک پردازنده یا ائتلاف پردازنده

۵.۴. مدل ترکیبی

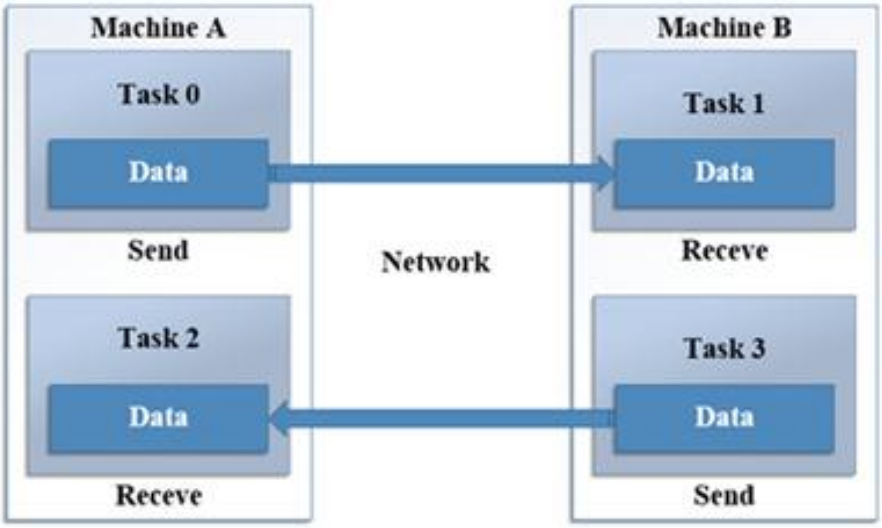
فراتر از این ۴ مدل که تشریح شد، مدل ایستگاه کاری- سرویس دهنده، مدلی است که بیشتر از دیگران مورد استفاده قرار گرفته است. علت هم این است که بسیاری از کاربران کارهای تعاملی ساده ای را انجام می دهند مثلا ویرایش، ارسال نامه الکترونیکی، اجرای برنامه های کوچک و...، این مدل برای این نوع کارهای کوچک و ساده ایده آل می باشد. در محیط هایی مثل محیط های کاری که گروههای بزرگی از کاربران که اغلب کارهایی را اجرا می کنند که نیازمند محاسبات حجیمی می باشد، مدل ائتلاف پردازنده ایده آل می باشد و مناسب تر از دیگر مدل ها است. با ترکیب این دو مدل یعنی مدل ایستگاه کاری- سرور و مدل ائتلاف پردازنده می توان یک مدل ترکیبی برای ساخت سیستم محاسبات توزیع شده استفاده کرد. مدل ترکیبی بر پایه مدل ایستگاه کاری- سرویس دهنده ساخته می شود ولی امکانات مدل ائتلاف پردازنده هم به آن اضافه می شود. در مدل ترکیبی علاوه بر اجرای مؤثر کارهایی که محاسبات سنگینی دارند، به کارهای تعاملی تضمین پاسخ داده می شود و روشی که برای این کار پیش گرفته می شود این است که به این کارها اجازه داده می شود که روی ماشین ایستگاه کاری محلی کاربر اجرا شوند. با این همه پیاده سازی سیستم عامل توزیع شده بر پایه این مدل، در مقایسه با مدل های ایستگاه کاری- سرور و مدل ائتلاف پردازنده، هزینه زیادی را لازم دارد [۸].

۵. ساختار سیستم عامل های توزیع شده[1]

می توان گفت ساختار سیستم های توزیع شده به گونه ای است که صورت جهان را تغییر داده است مثلا وقتی شما با یک مرورگر به اینترنت وصل می شوید، در واقع به یک وب سرور در جایی دیگر از سیاره متصل می شوید که به نظر می رسد این یک سیستم ساده ی کلاینت سرور توزیع شده است. یا وقتی شما با یک وب سرویس مدرن مانند گوگل و فیسبوک تماس دارید شما تنها با یک ماشین تعامل ندارید، در واقع در پشت صحنه ی این خدمات شما با مجموعه بزرگی از ماشین ها تعامل

[1] Distributed Operating System

دارید که برای ارائه این خدمات با یکدیگر همکاری می کنند[۵]. این ساختار به طور معمول یا از الگوی شیء گرا پیروی می کند یا از الگوی مبتنی بر پیغام. سیستم عامل های مبتنی بر پیغام یک کرنل یا هسته پیام عبوری بر روی هر نود قرار می دهند و از پیام های صریح برای پشتیبانی ارتباط بین پردازه ای استفاده می کنند. کرنل یا هسته ارتباط محلی و ارتباطات از راه دور را پشتیبانی می کند که گاهی اوقات از طریق یک پردازش مدیر شبکه ای مجزا اجرا شده است. در یک سیستم سنتی ، همانند UNIX، دسترسی به خدمات سیستم از طریق فراخواندن پردازه (پراسیجرکال) انجام می شود در حالیکه در سیستم عامل توزیع شده ی مبتنی بر پیغام ، درخواستها از طریق ارسال پیغام می باشند. سیستم عامل های مبتنی بر پیغام برای ساختاربندی سیستم های عامل جذاب می باشند زیرا سیاست و خط مشی، که در پردازش های سرور رمزگذاری شده است ، مجزای از مکانیسم اجرا شده در کرنل می باشد. مدلی که مورد علاقه محققان می باشد ، مدل کلاینت–سرور می باشد که در آن یک پردازش کلاینت خواهان برخی سرویس (برای مثال خواندن برخی داده ها از یک فایل) یک پیغام به سرور ارسال می کند و سپس منتظر یک پیغام پاسخ می ماند که در آن سیستم تنها دو پریمیتیو ارائه می کند: SEND و RECEIVE. پریمیتیو SEND مقصد را معین می کند و یک بافر فراهم می نماید. پریمیتیوهای RECEIVE بیان می کنند که پیغام از چه کسانی مطلوب می باشد (شامل "هرکسی") و یک بافر فراهم می کند جایی که پیغام ورودی ذخیره خواهد شد.

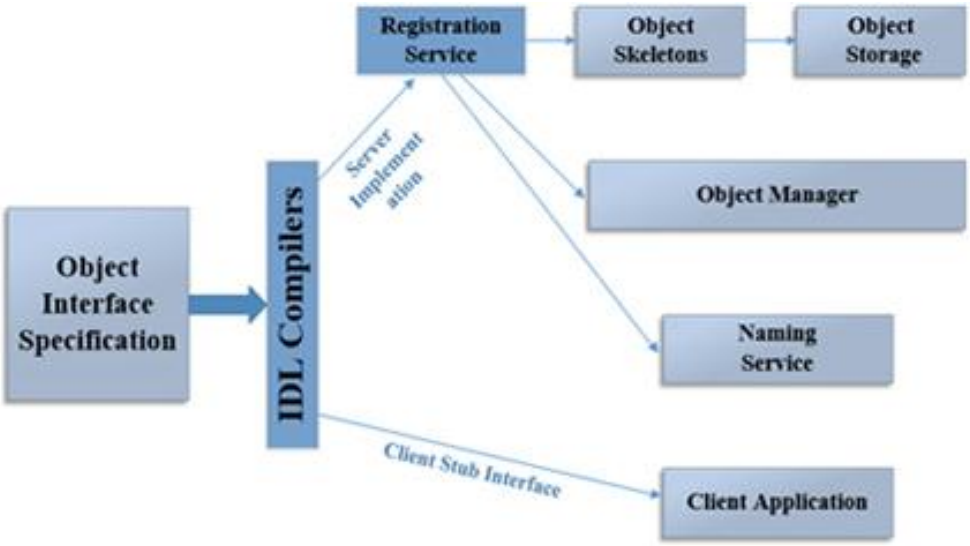


شکل ۶. الگوی مبتنی بر پیغام در سیستم عامل توزیع شده

اما در الگوی شیء گرا یا الگوی مبتنی بر شیء ، سیستم عامل های توزیع شده شیء گرا خدمات و منابع را در هویت هایی به نام اشیاء محصور می کنند. اشیاء مشابه با نمونه های انواع داده های انتزاعی می باشند. آن ها به عنوان ماژول های مجزا مرکب از عملیات های خاص نوشته می شوند که رابط های ماژول را تعریف می کنند. درخواست کلاینت برای دسترسی به سیستم به وسیله فراخوانی شیء سیستم مناسب دریافت می شود. مکانیسم فراخوانی مشابه با فراخوانی پردازه محافظت شده می باشد. اشیاء قابلیت را به میزانی که پردازش های سرور در سیستم های مبتنی بر پیغام انجام می دهند، محصور می کنند (در بر می گیرند). اغلب سیستم های شی گرا در بالای سیستم عامل موجود ، به طور معمول UNIX، ساخته شده اند. مثال هایی از چنین سیستم هایی شامل آرگوس۱، کرونوس۲ و ادن۳ می باشند. این سیستم ها اشیایی را پشتیبانی می کنند که به فراخوانی ارسال شده از طریق مکانیسم های ارسال پیغام UNIX پاسخ می دهند. ماخ یک سیستم عامل با مشخصه های متمایز می باشد. یک سیستم سازگار UNIX ساخته شده است تا مستقل از ماشین باشد؛ این بر روی طیف گسترده ای از تک پردازنده ها تا چند

[۱] Argus
[۲] Cronus
[۳] Eden

پردازنده ها اجرا می شود و دارای یک کرنل کوچک می باشد که حافظه مجازی و زمانبندی پردازش را تنظیم می کند و خدمات (سرویس) دیگر را بالای کرنل می سازد. ماخ مکانیسم هایی را اجرا می کند که توزیع را فراهم می کنند. خصوصا از طریق یک ابزار که اشیاء حافظه نامیده شده اند تا حافظه را بین وظایف مجزا در حال اجرا بر روی ماشین های مختلف به اشتراک بگذارد.

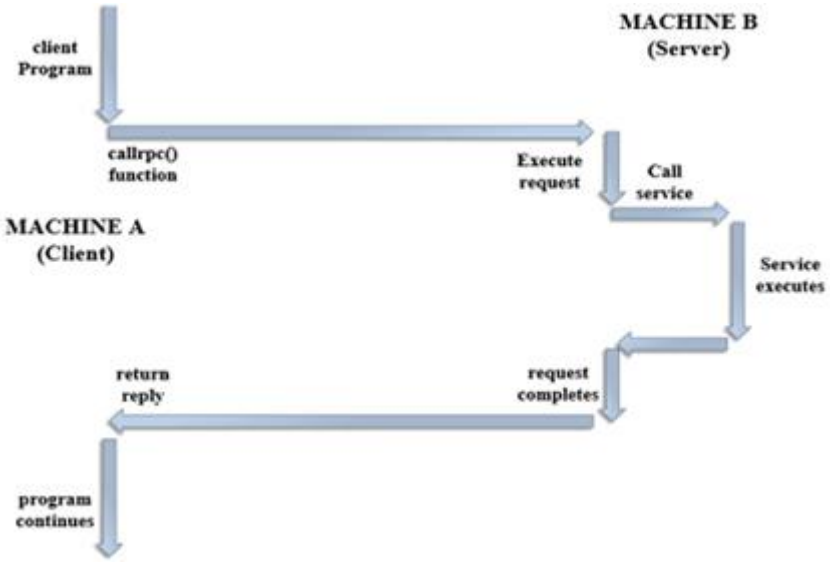


شکل ۷. ساختار سیستم عامل های توزیع شده

۱.۵. مفهوم (RPC) و آشنایی با نحوه کار آن

RPC یا Remote Procedure Call یک تکنیک قدرتمند برای ساخت اپلیکیشن های توزیع شده مبتنی بر کلاینت-سرور می باشد. اگر بخواهیم برای Remote Procedure Call که معنای آن فراخوانی رویه ها و فانکشن ها از راه دور مثالی بزنیم بیان خواهد شد که دو پردازش ممکن است بر روی سیستم یکسان باشند یا آن ها ممکن است بر روی سیستم های متفاوت با یک شبکه متصل کننده آن ها باشند. با استفاده از RPC، برنامه نویسان اپلیکیشن های توزیع شده از جزئیات رابط با شبکه محافظت می کنند. عدم وابستگی RPC به انتقال، اپلیکیشن را از المان های فیزیکی و منطقی مکانیسم های ارتباطات داده ایزوله می سازد و به اپلیکیشن اجازه می دهند تا از تنوعی از انتقال ها استفاده کنند. (البته بسیاری از برنامه های کاربردی از Broadcast برای ارتباطات استفاده می کنند اما تعداد کمی از سیستم عامل های اولیه این امکان را برای کاربران فراهم می کنند) [۳و۱۴] .

وقتی که یک RPC ساخته شد، آرگومان های در حال فراخوانی به پردازه از راه دور فرستاده می شوند و فراخواننده منتظر یک پاسخ از پردازه از راه دور می باشد. جریان فعالیت در طول فراخوانی RPC بین دو سیستم شبکه شده شکل می گیرد. کلاینت یک فراخوانی پردازه انجام می دهد که یک درخواست به سرور ارسال می کند و منتظر می ماند. از پردازش نخ جلوگیری می شود تا زمانی که یک پیغام پاسخ دریافت گردد یا زمان به اتمام برسد. وقتی که درخواست می رسد، سرور یک دیسپچ روتین فراخوانی می کند که سرویس درخواست کننده را اجرا می کند و پاسخ را به کلاینت ارسال می کند. بعد از تکمیل فراخوانی RPC، برنامه کلاینت ادامه می یابد. RPC خصوصا اپلیکیشن های شبکه را پشتیبانی می کند. یک پردازه از راه دور منحصرا توسط سه عامل شناسایی شده است. شماره برنامه، شماره نسخه، شماره پردازه.



شکل ۸ Remote Procedure Call (RPC)

یک پردازه از راه دور منحصرا توسط سه عامل شناسایی شده است. شماره برنامه، شماره نسخه، شماره پردازه. شماره برنامه یک گروه از پردازه های از راه دور مرتبط را شناسایی می کند که هر کدام دارای شماره پردازه منحصر به فرد می باشند. یک برنامه ممکن است از یک یا چند نسخه تشکیل شود. هر نسخه متشکل از یک کلکسیونی از پردازه ها می باشد که برای فراخوانی از راه دور در دسترس می باشند. شماره های نسخه سبب می شوند تا نسخه های متعدد یک پروتکل RPC به طور هم زمان در دسترس باشند. هر نسخه حاوی یک تعداد از پردازه ها می باشد که می تواند به صورت از راه دور فراخوانی شوند. هر پردازه دارای یک شماره پردازه می باشد.

۶. ویژگی های سیستم عامل توزیع شده

سیستم عامل های توزیع شده چند ویژگی اصلی دارند ازجمله: تفکیک کاربردی بر اساس قابلیت ها و خدمات و توانایی ها و هدف و هویتشان ، توزیع ذاتی مثل اطلاعات متفاوت توسط افراد مختلف که ایجاد و حفظ شده است ، قابلیت اعتماد یعنی بک آپ دراز مدت در محل های مختلف دارند ، مقیاس پذیری به معنای قابلیت سیستم برای افزایش منابع و اقتصادی بودن یعنی به اشتراک گذاری منابع با هویتهای بسیار برای کمک به کاهش هزینه مالکیت . هویت های متعدد در یک سیستم توزیع شده می توانند به صورت هم زمان و احتمالا به صورت خودگردان (مستقل) فعالیت کنند. وظایف به طور مستقل انجام شده اند و اقدامات توسط تبادل پیغام ها هماهنگ شده اند. همچنین، هویت ها هتروژن (ناهمگن)[۱] می باشند و شکست ها مستقل هستند. به طور کلی ، هیچ فرایند یا هویت مجزا وجود ندارد که دارای دانش کل وضعیت سیستم باشد.

همچنین سیستم های توزیع شده اهداف مشخصی را دنبال می کنند از جمله : هتروژنیتی یا همگنی یعنی وجود تعامل علیرغم تفاوت های سخت افزاری ، برنامه نویسی ، فرمت ها و ... ، شفافیت یعنی مخفی ماندن پیچیدگی ها از دید کاربر ، مدیریت شکست یعنی توانایی در برابر اختلالات ، مقیاس پذیری یعنی قابلیت افزایش منابع ، هم زمانی یعنی دسترسی مشترک به منابع ، قابلیت توسعه ، نقل مکان و تعادل بار ، وجود امنیت در کل سیستم و مسائل دیگری که موسسات تحقیقاتی و شرکت ها مطرح نموده اند به همین دلیل با توجه به اینکه تولید برنامه های کاربردی برای سیستم ها از جنبه های مختلفی مانند مدیریت فرایند ، ارتباطات توزیع شده در سیستم و قدرت تحمل خطا بسیار پیچیده است، می توان گفت تولید برنامه های کاربردی برای این سیستم عامل ها نیز کار دشواری است [۷و۳].

---

[۱] Heterogeneous

۷. رویکرد Cloud (رویکرد ابر)

Cloud یک سیستم عامل توزیع شده می باشد که یک مجموعه از نودها را به یک سیستم به لحاظ مفهومی متمرکز ادغام می کند [۱۷]. سیستم متشکل از سرورهای کامپیوتر، سرورهای داده و ایستگاه های کار کاربر می باشد. یک سرور کامپیوتر یک ماشین می باشد که برای استفاده به عنوان یک موتور محاسبه ای در دسترس می باشد. یک سرور داده یک ماشینی می باشد که به صورت یک مخزن ذخیره داده برای داده های با طول عمر بالا عمل می کند. یک ایستگاه کار کاربر ماشینی می باشد که یک محیط برنامه نویسی برای توسعه اپلیکیشن ها فراهم می کند و یک رابط با سرورهای داده و کامپیوتر برای اجرای این اپلیکیشن ها بر روی سرورها ارائه می کنند. به یک دیسک پشتیبانی شده با یک سرور کامپیتور توجه کنید. این همچنین می تواند به عنوان یک سرور داده برای دیگر سرورها در نظر گرفته شود. Cloud یک کرنل بومی به نام Ra می باشد. این در حال حاضر بر روی کامپیوترهای Sun-3/50 و Sun-3/60 به عنوان یک همکار با ایستگاه های یدک خورشیدی (واحدهای در حال اجرا) می باشد که رابط های کاربری فراهم می کنند. رایانش ابری یک توسعه اصلی در تکنولوژی اطلاعات می باشد که دارای اهمیت قابل مقایسه با مین فریم، مینی کامپیوتر و میکروکامپیوتر و اینترنت می باشد. رایانش ابری دارای پتانسیل برای ایجاد یک سهم قابل توجه در فعالیت اقتصادی در سرتاسر جهان می باشد. این پتانسیل وقتی محقق خواهد شد که محصولات و خدمات رایانش ابری[1] قابل حمل و دارای قابلیت همکاری باشند.

۸. مزایای سیستم عامل جهت مدیریت فرایند

استفاده از یک سیستم عامل اجازه می دهد تا زمان اجرا با مدیریت فرایندهای کاربر، پردازش ها مدیریت شود که این کار می تواند استفاده از سیستم را کارآمدتر کند، در طراحی های اولیه مدیریت فرایند در سیستم عامل های توزیع شده ، مکانیزم هایی لازم است که بتوانند کارایی طرح های اولیه مدیریت فرایند در سیستم عامل های همه منظوره موجود را افزایش دهند. قابلیت افزوده شده باید با ویژگی های انواع سیستم های توزیع شده خود را سازگار کنند: ایستگاه های کاری شخصی، ماشین های سرور مشترک و سیستم های میزبان متصل توسط یک شبکه محلی سریع. در حالت عادی ایستگاه های کاری توسط یک فرد مورد استفاده قرار می گیرند اما اگر هیچکس از آن ها استفاده نکند تبدیل می شوند به منبع محاسباتی برای سایر کاربران و ایستگاه ها. ماشین های با سرور مشترک هم یک سیستم فایل توزیع شده اند و دارای قابلیت هایی مثل دروازه هایی به اینترنت، دسترسی به پرینتر، درایوهای نواری و... هستند. منظور از سیستم های میهمان نیز سیستم عامل های سنتی است که به کمک اشتراک گذاری نرم افزار به سیستم توزیع شده متصل می شوند [۲, ۴].

۹. مدیریت فرایند در یک سیستم توزیع شده

اگر بخواهیم یک سیستم توزیع شده ی همه منظوره با زبان های مختلف، فایل سیستم های مختلف و نرم افزار ها و سخت افزارهای مختلف ایجاد کنیم محیط ما یک محیط برنامه ریزی ناهمگن است و روند طراحی باید به گونه ای باشد که امکان اجرای نرم افزار های موجود را فراهم کند. در این محیط برنامه های کاربران مختلف به طور مداوم یک پردازنده فیزیکی را به اشتراک می گذارند، بنابراین آن ها باید در فضاهای آدرس جداگانه اجرا شوند. از آنجا که تمام ماشین ها سیستم فایلشان محلی نیست طبیعتا برنامه ها باید از روی شبکه موجود دانلود شوند و این کار چند ثانیه طول می کشد. مکانیزم ارتباطی برنامه های کاربردی توزیع شده، تراکنش پیام است، یک جفت پیام: یک درخواست پیام از یک فرایند سرویس گیرنده به یک سرور است که به دنبال آن یک پیام پاسخ از سرور به سرویس گیرنده فرستاده می شود. در راس آن، فراخوان روال راه دور

[1] Cloud Computing

وجود دارد. هنگامی که با دقت طراحی و اجرا شود، تراکنش پیام از یکی از کارآمدترین پروتکل های ارتباطی را برای شبکه های محلی، هم از نظر تاخیر و هم از نظر توان عملیاتی، تشکیل می دهد.

۱۰. نحوه تراکنش پیام

هنگامی که یک فرایند سرویس گیرنده درخواستی ارسال می کند، تا زمان رسیدن پاسخ مسدود می کند؛ هنگامی که سرور درخواست طلب کند، تا زمان دریافت یک درخواست، مسدود می کند. استفاده از تراکنش های پیام، پیامدهای متعددی برای طراحی محیط برنامه نویسی دارد. ابتدا، فرایندها هر تراکنش پیام را مسدود می کنند. بنابراین، جا به جایی سوییچ ها رخ می دهد: یکی زمانی که فرایند، فرایند دیگر را مسدود می کند و دیگری بعد از اینکه فرایند دوباره برای اجرای اصلی غیرمسدود می شود. اگر تراکنش های پیام بسیار سریع انجام شوند، سوییچینگ فرایند نیز بهتر است که سریع باشد.
دوم، تراکنش های پیام موازی انجام نمی شود: هنگامی که سرویس گیرنده اجرا می شود، سرویس دهنده منتظر درخواست می ماند و هنگامی که سرویس دهنده اجرا می شود، سرویس گیرنده منتظر پاسخ می شود. تنها یک فرایند در آن واحد اجرا می شود، البته روی ماشین های مختلف. از یک روش می توان برای اجرای تراکنش های غیرمسدود استفاده نمود: سوییچ فرایند نیازی به رقابت با تراکنش های پیام در سرعت ندارد و عمل موازی سازی را می توان با ارسال درخواست به همه سرویس دهنده ها به طور هم زمان انجام داد. با این حال، در این روش با یک مجموعه مشکلات جدید مواجه می شویم. در بسیاری از سیستم های توزیع شده برای تسهیل نوشتن نرم افزاری که از سیستم موازی استفاده می کند، از فرایندهای سبک وزن و مسدود سازی تراکنش های پیام استفاده می شود. برای مهاجرت فرایندها در سیستم های توزیع شده، مکانیزم هایی پیاده سازی یا پیشنهاد شده اما تا کنون الگوریتمی پیشنهاد نشده که از مهاجرت برای توازن بار استفاده نماید. با توجه به زمان مورد نیاز برای مهاجرت یک فرایند گسترده (به ترتیب ده ثانیه)، مهاجرت برای توازن بار[1] به نظر چندان سودمند نمی آید. با این حال، می تواند در محیط ایستگاه های کاری فردی مفید باشد؛ چون در این محیط ها ایستگاه های کاری بلااستفاده به صورت منبع پردازش به دیگران اجاره داده می شوند و هنگامی که مالکان آن ها بازگشتند، آن را پس می گیرند [۴].

۱۱. سیستم عامل توزیع شده آموبا (آمیب)

آموبا یک سیستم عامل توزیع شده مبتنی بر الگوی ارتباط فرایندهای سرویس گیرنده با سرویس ها از طریق تراکنش های پیام است. این سیستم عامل از قابلیت هایی برای دسترسی به سرویس ها و اشیایی که این سرویس ها اجرا می کنند، استفاده می کند. یکی از این قابلیت ها مرجع ۲۵۶ بیتی برای یک شئ است که ۶۴ بیت اول پورت است و اشاره به مدیریت سرویس توسط شی دارد؛ ۶۵ بیت بعدی برای استفاده سیستم به عنوان مکان در دسترس است؛ ۱۲۸ بیت باقیمانده برای شناسایی شئ به سرویس تخصیص می یابد. یکی از قابلیت هایی که به این شیوه تولید شده است و شامل بین های کافی است، این احتمال وجود دارد که یک کاربر غیرمجاز حدس بزند که می توان از یکی از قابلیت های اهداف چشم پوشی کرد. از این قابلیت ها برای محافظت استفاده می شود و همچنین به عنوان مکانیزم اولیه برای رسیدگی به درخواست ها تا عملیات و اهداف انجام شوند. هنگامی که یک سرویس گیرنده درخواستی را می فرستد، سیستم از پورت برای تعیین نوع سرویسی که باید درخواست را اداره نماید، استفاده می کند. یک سرویس دهنده نیز از طریق عملیات موقعیت یابی پیدا می شود، یعنی از طریق انتشار بسته های (شما کجا هستید). سرویس دهنده از بخش خصوصی این قابلیت برای شناسایی شئ استفاده می کند. بعد از رسیدگی به یک درخواست، سرور پاسخ می دهد. بیشتر سرویس ها در فضای کاربری اجرا می شوند. هسته آمیب تنها یک سرویس حداقل ایجاد می کند: امکان تراکنش پیام، مدیریت فرایند و مسیر دسترسی به تجهیزات جانبی. برای مثال، سرویس فایل یک

[1] Load balancing

سرویس فضای کاربری بدون امتیازات ویژه است به جز دانش قابلیت های لازم برای گرفتن دیسک هایی که در آن ها فایل ها ذخیره شده اند. تراکنش های پیام مسدود شده اند و سیستم بافری ایجاد نمی کند. هنگامی که سرور فرمان getrequest را فراخوانی می کند (پورت، قابلیت، بافر درخواست)، (پورت سرور سیستم را شناسایی می کند)، سرور مسدوی می شود تا اینکه یک درخواست از راه برسد. سرور با فرمان putreply (replybuffer) یک پاسخ باز می فرستد که مسدود نشود. هنگامی که سرویس گیرنده فرمان trans (capacity، requestbuffer، replybuffer) را فراخوانی نماید، تا زمانی که پاسخ از سمت سرور برسد، مسدود می شود.

در صورت بروز خطا، به سرویس گیرنده گفته می شود سرور قابل دسترسی نمی باشد یا هیچ پاسخی دریافت نشده است. در مورد اول، سرویس گیرنده می تواند دوباره سعی کند؛ در مورد دوم، سرویس گیرنده باید جایی که خطا قبل و یا بعد از اجرای درخواست رخ داده را پیدا کند (مگر اینکه درخواست idempotent باشد؛ که در این مورد درخواست همیشه باید به صورت مطمئن تکرار شود). هنگامی که تراکنش یک سرویس گیرنده با خطا مواجه شود، پاسخ از دست می رود. یک درخواست هسته، درخواستی برای یک عملیات روی یک هدف است که توسط هسته حفظ شده است. یک درخواست هسته، یا فراخوان سیستم، یک تراکنش با سرویس هسته است. بنابراین، در اصل، آمیب تنها می تواند فراخوان سیستم را برای انجام تراکنش های پیام داشته باشد. با این حال، در عمل، بهتر است برخی از درخواست های سرویس هسته به صورت فراخوان های معمولی سیستم اجرا شوند. از آنجایی که تراکنش های آمیب مسدود می شوند، از آن ها نمی توان برای موازی سازی استفاده نمود. آمیب از فرایندهای موازی سازی برای دستیابی به آن استفاده می کند. آمیب فرایندهای سبک وزنی را انجام می دهد که برای بهره وری، تعدادی از وظایف می توانند یک فضای آدرس را به اشتراک بگذارند. یک فضای آدرس با تعدادی از وظایف موجود در آن، یک خوشه را تشکیل می دهند [۴, ۱۱].

۱۲. سرور کرنل (هسته)[1]

هسته یا کرنل آموبا قطعات حافظه و ارتباطات درون فرایندی را مدیریت می کند و از فرایندهایی که شامل چند نخ هستند پشتیبانی می کند[۱۹]، این هسته برای تحقق انتزاع فرایند در آمیب با سه شئ اصلی سروکار دارد. یک خوشه یک فضای آدرس دهی مجازی است که شامل تعدادی قطعه و تعدادی موضوع کنترل به نام وظایف است. علت مشترک بودن فضای آدرس میان وظایف، بهره وری است: وظایف می توانند اطلاعات را در یک حافظه مشترک، بین یکدیگر به صورت کارآمدتری مبادله نمایند و از آنجایی که وظایف زمینه کمی دارند، سوییچ کردن وظیفه می تواند خیلی سریع اتفاق افتد. از مفهوم وظایف در سیستم های توزیع شده مدرن زیادی استفاده شده است که در بین این سیستم عامل هاV [۲۴] ، Mesa [۲۵] و Topaz قابل توجه اند.

۱.۱۲. قطعات[2]

یک قطعه یک بخشی خطی از حافظه است. قطعه یک شئ است که توسط سرویس کرنل مدیریت می شود. قطعات با نگاشت اشیاء در فضای آدرس یک خوشه با استفاده از فراخوان سیستم map_seg ایجاد می شوند. این امکان وجود ندارد که یک قطعه موجود را نگاشت کرد؛ بنابراین، قطعات را نمی توان میان خوشه ها به اشتراک گذاشت. فراخوان برای مدیریت قطعه در شکل نشان داده شده است.

[1] Kernel
[2] Segment

System Calls
sid = seg_map(segment, address, lengh, how)
seg_unmap(sid)
seg_grow(sid, newlength)
seg_info()
Transactions
Seg_Create(kernelcap, incap, outcap)
Seg_Read(capability, offset, buffer, count)
Seg_Write(capability, offset, buffer, count)
Seg_Length(capability)
Seg_Delete(capability)

شکل ۹. تراکنش ها و فراخوان های سیستم مدیریت قطعه

تفاوت اساسی میان تراکنش ها و فراخوان های سیستم، این است که فراخوان های سیستم توسط هسته محلی اداره می شود و بنابراین، هسته محلی تنها برای کنترل اشیاء محلی قابل استفاده است. با این حال، تراکنش ها برای سرویسی رسیدگی می شوند که ممکن است محلی یا راه دور باشند، تراکنش Seg_Read بنابراین برای خواندن محتویات قطعات راه دور استفاده می شوند. با این حال، تراکنش ها باید به سرویسی آدرس دهی شوند که ممکن است محلی یا راه دور باشد، بنابراین تراکنش Seg_Read را می توان برای خواندن محتویات قطعات راه دور به کار برد. فراخوانی seg_map یک قطعه جدید ایجاد می کند که آن را به محتویات قطعه مقداردهی می نماید که با segment نشان داده می شود و این خود یک توانایی محسوب می شود. how گزینه های نگاشت مثل نگاشت فقط خواندنی را مشخص می کند که تحت تاثیر فراخوان Seg_grow و ... قرار دارد. این فراخوان یک مقدار صحیح کوچک به نام شناسه قطعه را باز می گرداند که قطعه نگاشت شده را نشان می دهد. این عدد صحیح در فراخوان های seg_unmap و seg_grow مورد استفاده قرار می گیرد. هنگامی که یک قطعه از نگاشت خارج می شود، از فضای آدرس یک خوشه خارج می گردد اما به صورت یک شئ به حیات خود در حافظه ادامه می دهد. Seg_unmap قالبیت این شئ حافظه را برای دستکاری بیشتری با تراکنش ها جهت مدیریت قطعه باز می گرداند. Seg_info اطلاعات مربوط به فراخوانی نگاشت فعلی حافظه خوشه را باز می گرداند. تراکنش های مدیریت قطعه برای خودشان کار می کنند.

۲.۱۲ خوشه ها[1]

سرویس کرنل نیز خوشه ها را مدیریت می کند که با ارسال یک درخواست CreatCluster به سرور هسته ای ایجاد می شوند. پارامتر مورد نظر برای درخواست، یک توصیف کننده خوشه است که با توصیف وضعیت وظایف آن، فضای آدرس که در آن این کارها انجام خواهند شد و پردازنده ای که در آن خوشه باید اجرا شود، حالت اولیه خوشه را توصیف می کند. شکل ۹، یک توصیف کننده خوشه را نشان می دهد.

[1] Cluster

Host Descriptor
Accounting & Scheduling
Exception Handler
Number of Segments
Mapping Descriptors ⋮
Number of Tasks
Task Descriptors ⋮

شکل ۱۰. توصیف کننده خوشه

توصیف کننده میزبان، نوع پردازنده ای خوشه در آن اجرا می شود را توصیف می نماید. ورودی های آن دارای یک نوع و مقدار است. برای مثال، در نوع "instruction set"، مقادیری مثل VAX، M68000 یا NS32000 فرض می شوند و مجموعه دستورالعمل هایی توصیف می شوند که گره خوشه به آن تعلق دارد. از نوع گزینه های مستقل از مجموعه دستورالعمل برای نشان دادن اینکه خوشه نیاز به گزینه های مجموعه دستورالعمل ها دارد یا خیر استفاده می شود، مثل نقطه اعشاری یا مجموعه دستورالعمل های پیشرفته. نوع اندازه حافظه دارای مقداری است که نشان دهنده حداکثر اندازه ای است که فضای آدرس خوشه برای رشد ممکن است به آن نیاز داشته باشد. انواع احتمالی دیگری نیز وجود دارند؛ انواع جدید را به سادگی می توان افزود. سرویس کرنل تعدادی از انواع مفید را تشخیص می دهد و از مقادیر آن ها برای تعیین اینکه آیا می تواند خوشه را اداره کند یا نه استفاده می نماید. انواع دیگر را می توان توسط سرویس های کاربر مورد استفاده قرار داد که توصیف کننده های خوشه را دستکاری می کنند (مثل، سرویس نمونه سازی).

زمینه اداره استثناء باعث می شود که پورت سرویس در هنگام وجود استثناء ها آن ها را اداره کند. سپس به دنبال mapping descriptors، برای هر قطعه فضای آدرس خوشه وجود دارد. این هسته با انجام فراخوان های seg_map که توسط هر یک از توصیف کننده های نگاشت مشخص شده است، فضای آدرس مجازی خوشه را ایجاد می کند. درنهایت، فهرستی از توصیف کننده های وظیفه وجود دارد، برای هر وظیفه در خوشه، وضعیت هر وظیفه در خوشه را نشان می دهد. این یکی از مزایای داشتن تراکنش ها به عنوان تنها راه برای برقراری ارتباط با خارج از خوشه است: کرنل آمیب حالت کمی برای وظایف حفظ می کند. حالت یک وظیفه شامل مواردی مثل قابل اجرا بودن یا مسدود شدن در یک نشانه یا شرایط متغیر، مقدار شمارنده برنامه، اشاره گر پشته، حرف وضعیت پردازنده و ثبات های دیگر است و اگر یک تراکنش در حال انجام باشد، شامل وضعیت آن نیز می شود. توجه کنید که در هر لحظه، یک وظیفه می تواند تنها شامل دو تراکنش باشد: می تواند در حال انجام یک تراکنش با یک سرور باشد و در عین حال، به درخواست سرویس گیرنده نیز بپردازد. وضعیت باید برای تراکنش های در حال انجام حفظ شود. در ادامه، باید به موضوع شروع و توقف خوشه های با وظایفی بپردازیم که در میانه یک تراکنش قرار دارند. بنابراین، سرور هسته تمام اطلاعاتی که برای راه اندازی خوشه جدید نیاز دارد را در اختیار دارد. این سرور یک قابلیت خوشه را سرویس گیرنده ای باز می گرداند که مورد درخواست وی بوده است بنابراین تنها این سرویس گیرنده به عنوان مالک خوشه جدید می تواند کنترل خوشه را در دست بگیرد.

۳.۱۲. رابط سرور کرنل

شکل ۱۰ رابط با سرور کرنل را برای مدیریت فرایند فهرست می نماید. اولین استدلال این درخواست، قابلیت شئ است که درخواست به آن اشاره دارد. درخواست CreateCluster اشاره به شئ ای دارد که تا کنون وجود نداشته است؛ قابلیت آن یک قابلیت هسته ای است که در مقابل کاربران غیرمجازی که به هسته خوشه ها دسترسی پیدا کرده اند، محافظت انجام می دهد. ACK پاسخ خطا یا موفقیت عمومی را نشان می دهد. در مورد خطا، یک دلیل خوب ارائه می کند. نیمی از فراخوان های سیستم به عنوان تراکنش هایی با هسته پیاده سازی می شوند و بقیه نیز در دام هسته می افتند. علت پیاده سازی برخی از این فراخوان ها به صورت دام، یکی مربوط به بهره وری است. اقداماتی مثل MakeTask یا P اغلب انجام می شوند و باید حتی المقدور با حداقل دستورات پیاده سازی شوند. توجه داشته باشید که فراخوان های سیستم تنها روی خوشه هایی تاثیر دارند که با آن ها مشکل دارند. فراخوان تراکنش به حفاظت مکانیزم حفاظتی آموبا نیاز دارد.

CreateCluster یک خوشه جدید ایجاد می کند. استدلال آن یک توصیف کننده خوشه است که خوشه ای که باید آغاز شود را توصیف نماید. برای هر وظیفه خوشه، توصیف کننده شامل یک توصیف کننده وظیفه است که شمارنده برنامه، اشاره گر پشته و محتویات ثبات را می دهد و برای هر قطعه، نگاشت اطلاعات در نگاشت توصیف کننده ها ارائه می شود. در بخش های بعد به مسائل مربوط به ایجاد خوشه ها در یک حالت دلخواه باز می گردیم، مثل حالاتی که برای مهاجرات مورد نیاز هستند.

Transactions with kernel Service

Cluster creation and deletion (cluster may delete self):

CreateCluster(*KernelCap, ClusterDesc*); returns *ClusterCap*

DeleteCluster(*ClusterCap*); returns *ClusterDesk*

Interrupting clusters:

Signal(*ClusterCap, SignalType, Parameter*); returns *ack*

Kernel System Calls

Task management:

MakeTask(*Program Counter, StackPointer*); returns *ack*

ExitTask(); does not return

Synchronization:

P(*Semaphore*);

V(*Semaphore*);

Sleep(*Condition*);

Wakeup(*Condition*);

شکل ۱۱. درخواست های کرنل و فراخوان های سیستم برای مدیریت فرایند

deleteCluster یک خوشه را حذف می کند و توصیف کننده خوشه آن را باز می گرداند. توصیف کننده خوشه بازگشتی را می توان به یک فرمان CreateCluster داد و خوشه در جایی که متوقف شده بود، کار را ادامه می دهد، اگر به خاطر این مساله نباشد که خوشه های دیگر با این روش با یکدیگر ارتباط برقرار می کنند، می توان گفت که آن خوشه میان DeleteCluster و CreateCluster از بین می رود. تعلیق یا مهاجرت یک خوشه از این سخت تر است.

قطعات با انجام یک عملیات seg_map روی هر قطعه توصیف شده در توصیف کننده قطعه، ایجاد می شوند. محتویات قطعاتی که قابلیت های آن ها فراخوان ارائه شده است، محتویات اولیه را تشکیل می دهد. یک قطعه تهی با مشخص کردن قابلیت تهی ایجاد می شود. اجرای یک خوشه ممکن است با ارسال یک سیگنال دچار اختلال گردد. یک سیگنال باعث می شود که یک خوشه در مسیر خود متوقف شود و وضعیت آن به debugger server فرستاده شود. برای کنترل سیگنال، اشکال

زدا می توانند وضعیت خوشه را قبلی از ادامه اجرا، بازرسی کند و آن را تغییر دهد. تراکنش های مربوط به سرویس کرنل که پیش از این توصیف شدند، تحت حفاظت مکانیزم محافظت مبتنی بر قابلیت نرمال سیستم آموبا قرار دارند: یک برنامه کاربردی تنها می تواند خوشه هایی روی پردازنده هایی ایجاد کند که در آن قابلیت کرنل وجود داشته باشد. درنتیجه، برای مثال، یک کاربر غیرمجاز را می توان از اجرای خوشه ها روی ایستگاه کاری کاربر دیگر منع کرد. علاوه بر این، قطعات با مکانیزم قابلیت خود محافظت می شوند. می توان گفت که قطعه خصوصی کاربر را نمی توان بدون اجازه با قطعه دیگر نگاشت کرد. سیگنال ها را تنها می توان توسط نگهدارنده های یک مالک قابلیت خوشه، به خوشه ها ارسال نمود. فراخوان هایی که ما توصیف می کنیم به این مکانیزم حفاظتی سنگین نیازی ندارند، برای اینکه آن ها تنها روی خوشه هایی تاثیر می گذارند که کار آن ها انجام شده باشد. بنابراین، فراخوان های مدیریت وظیفه و فراخوان های هماهنگ سازی وظیفه به طور مطمئن به عنوان فراخوان های سیستم واقعی پیاده سازی می شوند، که به دلیل پیاده سازی کارآمد آن ها، عملکرد آن ها بسیار حایز اهمیت است. با یک فراخوان سیستم MakeTask یک کار جدید ایجاد می شود. این پارامترها، یک شمارنده برنامه و یک اشاره گر پشته هستند. وظیفه جدید اجرای خود را در آدرسی شروع می کند که توسط شمارنده برنامه نشان داده شده است. این کار جدید را نمی توان در میانه تراکنش آغاز کرد؛ ثبات ها تعریف شده نیستند. یک کار می تواند خود را توسط فراخوان ExitTask حذف کند.

برای همگام سازی، چهار فراخوان ارائه می شود: P وV روی تیرهای راهنمای باینری اجرا و Sleep و Wakeup روی متغیرهای شرطی اجرا می گردند. فرمان Sleep یک وظیفه را به حالت خواب و دستور Wakeup هر وظیفه ای که به صورت مشروط به خواب می رود را بیدار می کند. این طرح های اولیه در اصل همانند طرح های موجود در سیستم توزیع شده توپاز و پیشینیان آن هستند، Mesa در حالت نرمالP و V به طور کامل در فضای کاربری اجرا می شوند. اگر تیر راهنما از قبل توسط کار دیگری رزرو شده باشد، به یک فراخوان سیستم در P نیاز است؛ فراخوان V نیز تنها زمانی نیاز است که کار دیگری در حالت انتظار برای آن، مسدود شده باشد.

۱۳. سرور اشکال زدا[1]

هنگامی که یک خوشه آموبا به دلیل وجود یک استثناء محبوس می شود، می توان به یک اشکال زدا استناد کرد. سرور اشکال زدا، یک خوشه فضای کاربری بدون مزایای خاص، می توان روی همان هسته به عنوان خوشه معیوب باقی بماند اما علاوه بر این می توان راه دور باشد. با این حال، برای اشکال زدایی راه دور، می توان از سرویس دهنده فرایند کمک گرفت. در این بخش، می توانیم مکانیزم هایی برای اداره استثناءها و سیگنال ها توصیف نماییم. استثناءها و سیگنال ها متفاوت هستند اما به صورت یکسان اداره می شوند. یک استثناء در اصل یک رویداد غیر هم زمان است که توسط یک خوشه روی خود استثناءهای معمول به وجود آمده است و تقسیم بر صفر می شود و به حافظه مجازی غیرموجود رسیدگی می کند و سعی در اجرای غیر دستورالعمل ها دارد. یک سیگنال یک رویداد غیر هم زمان است که توسط یک خروجی منبع به یک خوشه ایجاد می شود. سیگنال ها معمولا توسط انسان هایی به وجود می آیند که روی کلید وقفه روی پایانه خود ضربه می زنند و قصد دارند که اجرای یک خوشه را خاتمه دهند یا حداقل جریان معمول اجرای آن را دچار وقفه کنند. سیگنال ها نقش مهمی در مهاجرت دارند. سیگنال ها و استثناءها روی اجرای یک خوشه وقفه ایجاد می کنند. استثناءها معمولا یک تله سخت افزاری ایجاد می کنند که توسط هسته اداره می شود. به طور مشابه، سیگنال ها نیز در هسته ای به پایان می رسند که در آن خوشه اجرا می شود. سیگنال ها و استثناءها باعث بروز اتفاقاتی می شوند: ۱. تمام کارهایی که در خوشه اجرا می شوند متوقف می شوند. روی یک پردانده چندگانه، نمی توان تمام وظایف را به طور تجزیه ناپذیر متوقف کرد. ۲. تراکنش های فعال منجمد می شوند:

[1] Debug Server

پروتکل تراکنش به پیام های ورودی با یک عبارت " دوباره سعی کن، این خوشه منجمد است" پاسخ می دهد. این باعث می شود که سازمان های پروتکل ارسالی سعی کنند که همان پیام را بعدا دوباره ارسال کنند و تا زمانی که این پاسخ داده شود، دست از کار بر نمی دارند. ۳. یک توصیف کننده خوشه برای خوشه علامت دار ایجاد شده است و هنگامی که خوشه ایجاد شود، کرنل یک درخواست PleaseDebug را به سروری می فرستد که قابلیت آن در زمینه قابلیت سیگنال توصیف کننده خوشه باشد. ۴. سپس کرنل منتظر پاسخ از سرور اشکال زدا می شود که ممکن است شامل یک توصیف کننده خوشه اصلاح شده باشد. ۵. خوشه اجرا را احتمالا در وضعیت اصلاح شده، جمع بندی می کند.

تراکنش های در حال انجام، تنها در چند وضعیت به خوبی منجمد می شوند: سرورها می توانند در حین انتظار برای یک درخواست ورودی (اما نه بعد از زمانی که درخواست شروع به ورود کرده است) یا حین پردازش یک درخواست (بین تکمیل getrequest و فراخوانی putreply) منجمد شوند. سرویس گیرنده ها تنها می توانند در زمان ارسال درخواستی که تکمیل شده و زمانی که پاسخ شروع به ورود می کند، منجمد شوند. علاوه بر این، سرویس گیرنده ها یا سرویس دهنده ها را نمی توان در زمانی منجمد نمود که پروتکل منتظر تصدیق است. خوشه هایی که نه سرویس گیرنده اند و نه سرویس دهنده همیشه می توانند منجمد شوند. بنابراین، اگر پیام ها نیاز به انتشار دوباره داشته باشند (منتظر یک پیام تایید)، تراکنش ها قابل توقف نیستند. همچنین در زمانی که تراکنش ها ممکن است متوقف نشوند، محدودیت طولی دارند (توسط حداکثر تعداد انتشارات دوباره، حداکثر تعداد بسته ها در یک پیام، زمان انتشار دوباره و حداکثر طول عمر بسته) و معمولا کوتاه هستند. پاسخ هایی که سرور اشکال زدا می تواند به درخواست PleaseDebug بدهد، ادامه می یابد یا حذف می شود. مورد اول به خوشه اجازه تداوم اجرا را می دهد؛ اگر توصیف کننده خوشه اصلاح شده با پاسخ همراه باشد، وضعیت خوشه در ابتدا تصویب می گردد. مورد دوم خوشه را راه اندازی نمی کند بلکه آن را حذف می نماید [۴, ۱۱].

۱۴. مخزن پردازنده[1]

بیشتر فرایندها به عنوان رویدادی از اجرای یک برنامه روی یک کامپیوتر توسط یک مفسر فرمان ایجاد و مدیریت خواهند شد اما فرایندهای دیگر ممکن است موارد جدیدی ایجاد کنند. به قابلیتی نیاز است که امکان برقراری ارتباط با کرنل آموبا را بدهد. بیشتر کاربران برای اجرای کرنل آموبا روی ایستگاه کاری خود به قابلیت ایجاد خوشه دسترسی خواهند داشت به همین دلیل کاربران می توانند فرایندهای جدیدی روی ایستگاه کاری مخصوص خود ایجاد کنند. توانایی های مربوط به ایجاد فرایندها روی پردازنده مخزن، معمولا توسط سرویس "مخزن پردازنده" نگه داشته می شود که به عنوان یک عامل برای اجرای برنامه ها از جانب پردازش های کاربر عمل می کند. توازن بار را می توان توسط سرویس مخزن پردازنده انجام داد یعنی در زمانی که پردازنده های مخزن به طور صحیح تخصیص داده شده باشند [۴, ۱۰].

۱۵. مهاجرت[2]

اگرچه خوشه ها بعد از راه اندازی، به ندرت به یک میزبان جدید حرکت می کنند، مهاجرت یک مفهوم مرکزی در مکانیزم های مدیریت فرایند آمیب محسوب می شود. دلیل این است که بارگذاری خوشه های جدید در حافظه، گرفتن رونوشت های هسته ای، ایجاد نقاط بازرسی و انجام اشکال زدایی راه دور، همگی شبیه مهاجرت یک خوشه هستند. در واقع، اگر بتوانیم یک خوشه را از یک ماشین به ماشین دیگر مهاجرت دهیم، دانلود، حالت برداری، اشکال زدایی و ... می توانند ساده باشند. متعادل سازی باز توسط جابه جایی خوشه یکی از حوزه هایی است که به خوبی درک نشده و نمی دانیم که آیا واقعا با نوع فعلی ایستگاه های کاری و شبکه ها میسر واقع می شود یا خیر. برای مثال، مهاجرت یک خوشه ۵ مگابایتی حداقل ۷ ثانیه زمان نیاز

[1] Processor Pool
[2] Migration

دارد، چون برای کپی محتویات حافظه در یک اترنت ۱۰ مگابیتی به این زمان نیاز داریم؛ برنامه های ۵ مگابایتی اصلا غیرعادی نیستند، به ویژه که برای مهاجرت انتخاب شده باشند: خوشه هایی با عمر طولانی طول بزرگی دارند. بنابراین، مهاجرت نسبتا گران است و سود عملیات مهاجرت نیز باید زیاد باشد، با وجود این، مهاجرت می تواند مفید باشد. هنگامی که مالک یک ایستگاه کاری در شب خارج می شود، ایستگاه کاری می تواند خود را به یک پردازنده مخزن تبدیل و برای بقیه سیستم یک سرویس اجراکننده پردازش محسوب شود. هنگامی که مالک، صبح باز می گردد و دوباره وارد می شود، خوشه های مهمان که در حال اجرا هستند می توانند به ایستگاه دیگری منتقل شوند. هسته ها مکانیزم مهاجرت خوشه را اجرا می کنند. آن ها یک خط مشی را پیاده سازی نمی کنند؛ تصمیم گیری برای انتقال یک خوشه و جایی که آن خوشه باید منتقل شود، در سطوح بالاتر سرویس انجام می شود. فرایندی که انتقال را دستور می دهد به سرور فرایند معروف است. هنگامی که یک خوشه از یک ماشین به ماشین دیگر می رود، هسته ماشین قدیمی باعث می شود که محتویات حافظه و توصیف کننده خوشه برای هسته ماشین جدید در دسترس قرار گیرد. هسته موجود در ماشین جدید خوشه را در حافظه بارگذاری می نماید و آن را راه اندازی می کند. ما به این دو هسته، میزبان قدیمی و میزبان جدید می گوییم، فرایند می تواند همه چیز را برای کنترل سیگنال های خوشه به صورت اشکال زدایی خوشه راه اندازی نماید. ابتدا، سرور فرایند یک سیگنال به خوشه ارسال می کند که باعث می شود میزبان قدیمی در مسیرهای خود منجمد شود و یک توصیف گر خوشه به سرور فرایند ارسال می کند (سرور فرایند برای این خوشه مثل آشکارساز خطا رفتار می کند). سپس، سرور فرایند توصیف گر خوشه را در یک درخواست RunCluster به میزبان جدید می فرستد. هنگامی که میزبان جدید درخواست RunCluster را دریافت می کند، قطعات لازم را تولید می کند و با ارسال درخواست های SegRead به میزبان قدیمی آن ها را مقداردهی می کند و آن ها را در فضای آدرس خوشه جدید نگاشت می کند. با سرویس گیرنده (میزبان جدید) و سرویس دهنده (میزبان قدیمی) می توان به طور مستقیم حافظه نگاشت شده را ارسال و دریافت نمود؛ بنابراین محتویات حافظه یک خوشه باید با سرعت بالای نصف یک مگابایت در ثانیه، روی اترنت کپی شود. هنگامی که تمام محتویات قطعه کپی شد، میزبان جدید خوشه را شروع و به سرور فرایند پاسخی شامل قابلیت خوشه جدید را ارسال می کند. بعد از آن، سرور فرایند با یک درخواست DeleteCluster به میزبان قدیمی، خوشه قدیمی را حذف می کند. در حالی که عمل مهاجرت انجام می شود، خوشه در میزبان قدیمی در حالت منجمد قرار دارد. بنابراین، هسته به تمام پیام ها برای خوشه منجمد با یک پیام " دوباره تلاش کنید، این خوشه منجمد است" پاسخ می دهد. بعد از حذف خوشه، آن پیام ها دوباره در برخی نقاط ظاهر می شوند و هسته نیز با این پیام پاسخ می دهد: " این پورت برای این آدرس ناشناخته است". سپس، فرستنده عملیات مکان یابی انجام می دهد تا محل جدید خوشه را پیدا کند و ارتباط دوباره برقرار خواهد شد.

پروتکلی که برای کار با تراکنش های پیام در حین مهاجرت استفاده می شود نسبت به پروتکلی که اینجا توصیف شد، زیرکانه تر است اما توصیف کامل آن به فضای خیلی زیادی نیاز دارد. برای حفظ معنای حداکثر یک بار در تراکنش های پیام آموبا، سرویس گیرنده و سرویس دهنده باید از پورت های ارتباطی منحصربه فرد به نحوی استفاده کنند که برای مثال، عملیات مکان یابی آدرس سرور را به اشتباه ندهد [۴].

۱۶. سرویس شبیه سازی[۱]

یکی از مهمترین برنامه های کاربردی مکانیزم های عمومی برای کنترل سیگنال ها، تله ها و استثناءها است که اجازه شبیه سازی هر نوع محیط سیستم عاملی را می دهد. آمیب در یک محیط یونیکس توسعه یافته است، ما دو شکل از شبیه سازی یونیکس را پیاده سازی کردیم: جداسازی فراخوان های سیستم در سطح کد منبع C یا در سطح فراخوان سیستم. درک مورد اول راحت

[۱] Emulation Service

تر است و همراه با سرویس های پشتیبان اصلاح شده، عملکرد مناسبی ارائه می دهد. مورد دوم پیچیده تر است اما از آن می توان برای ارائه قابلیت های باینری استفاده کرد: باینری هایی که تحت یونیکس معمولی اجرا می شوند را می توان طوری تنظیم کرد که تحت آمیب بدون تغییر حتی یک بیت اجرا شوند. کتابخانه برای شبیه سازی یونیکس در سطح کد منبع عملا بدون تغییر تحت سلطه مدیریت جدید فرایند باقی خواهد ماند. نسخه کرنل که شبیه سازی یونیکس روی آن اجرا می شود، یک جدول از جفت های (قابلیت وظیفه، قابلیت شبیه ساز) را حفظ می کند. هنگامی که یک وظیفه وارد تله می شود و یک ورودی در جدول پیدا شود، ثبات ها (PC، SP، PSW و ثبات های همه منظوره) و آدرس بردار وقفه به شبیه ساز ارسال می گردد. شبیه ساز از تراکنش هایی با SegmentServer (یک سرور برای خواندن و نوشتن حافظه است که توسط سرور فرایند تحت نظام مدیریت جدید فرایند جایگزین خواهد شد) برای رسیدن به محتویات حافظه خوشه استفاده می کند. این سرور مقادیر جدیدی برای ثبات ها به هسته باز می گرداند. شبیه ساز روی یونیکس اجرا می شود که از قبل برای انجام تراکنش ها اصلاح شده بود. شبیه ساز فراخوان سیستم را روی یونیکس تفسیر می کند و نتایج را باز می گرداند. هم در این طرح و هم در طرح جدید، هسته آموبا هیچ اطلاعی درباره فراخوان های سیستم یونیکس ندارد. هنگامی که وظیفه یک کاربر در تله می افتد، آشکارساز فراخوانی می شود. تفاوت میان سیستم کاری و موردی که ما اجرا می کنیم به این شکل است: در مورد قدیمی، فرایندهایی که باید شبیه سازی گردند از طریق شبیه ساز، شبیه سازی می شود که بیشتر حالات آن را نیز در نظر می گیرد؛ وضعیت مربوط به شبیه ساز تنها شامل ثبات ها است. در طرح جدید، این وضعیت همان توصیف گر خوشه خواهد بود. خوشه هایی که باید شبیه سازی شوند، توسط شبیه ساز ایجاد نمی گردند. در طرح قدیمی، حافظه از طریق تراکنش با سرور قطعه خوانده می شود. در طرح جدید، حافظه را می توان به طور مستقیم توسط شبیه ساز خواند و نوشت، برای اینکه در فضای آدرس مخصوص به خود نگاشت شده است. هنگامی که به تجربیاتی در زمینه این ترتیبات برسیم، تصمیم خواهیم گرفته که آیا این مسیر جدید از هسته به شبیه ساز یونیکس خیلی طولانی است یا خیر. اگر طولانی باشد، باید یک نمایش از یک وضعیت سبک وزن ایجاد کنیم که به جای وضعیت فعلی (توصیف گر خوشه) به شبیه ساز داده می شود. شبیه ساز حافظه خوشه شبیه سازی شده را در فضای آدرس خود نیز شبیه سازی می کند و امکان دسترسی به حافظه را فراهم می نماید. برای ساخت یک مجموعه منسجم از طرح های اولیه برای مدیریت فرایند که شامل مهاجرت خوشه ها، حالت برداری، اشکال زدایی و شبیه سازی رابط های سیستم عامل دلخواه باشد، به طرح دقیقی نیاز دارد که نه تنها شامل مکانیزم هایی است که به طور مستقیم با مدیریت فرایند سروکار دارند، بلکه شامل تمام محیط پیرامون نیز می شود. هسته آموبا حداقل قابلیت ها را دارد: مدیریت فرایند و ارتباطات درون فرایندی. بنابراین، یک حداقل مقدار از وضعیت وجود دارد که باید در زمان مهاجرت خوشه، به جای دیگری منتقل شود. مکانیزم ارتباطات درون فرایندی آموبا برای موفقیت طرح حیاتی است. اول، سازمان های ارتباطاتی با استفاده از مکانیزم نامگذاری مستقل از موقعیت نام گذاری می شوند و از سرویس تعیین مکان پایه برای پیدا کردن جایی که بسته ها باید ارسال شوند، استفاده می کنند. هیچ یک از دستگاه های مهاجرت نباید نگران تغییر مسیر پیام ها باشند، نباید آدرس ها بدون ارسال رها شوند؛ میزبانان سابق می توانند وجود یک خوشه را درست بعد از انجام مهاجرت، فراموش کنند.

دوم، سادگی پروتکل های آموبا کمک فوق العاده ای به قابلیت حمل و نقل خوشه ها می کند. این پروتکل تنها چند حالت دارد که در آن حالت ها می تواند به مدت دلخواه باقی بماند و می توان گفت که به سادگی می توان یک خوشه را در این حالات با استفاده از پیام های "من منجمدم، مزاحم نشوید" ، به جای دیگری منتقل کرد. هنگامی که این پروتکل در هر یک از حالات دیگر باشد، کرنل آمیب آنقدر صبر می کند تا پروتکل به حالت قابل انتقال برسد. مهمترین نتیجه ای که می توان از این طرح گرفت، این است که می توان یک مکانیزم ساده ایجاد کرد که برای تحقق دانلود، مهاجرت، کنترل استثناءها، حالت برداری، شبیه سازی و اشکال زدایی در سیستم عامل های توزیع شده که جزء حیاتی سیستم های کامپیوتری می باشند کارآمد باشد. در برخی منابع برای بررسی فرایند ها فقط موقعیت های به وجود آوردن فرایند و مهاجرت فرایند ها مورد بررسی قرار

گرفته شده است اما در این مقاله تمامی مسائل پیرامون مبحث مدیریت فرایند در سیستم عامل های توزیع شده به تفصیل شرح داده شد. [۴, و۱۵و ۱۳].

۱۷. نتیجه گیری

در این مقاله مروری داشتیم بر روند تکامل سیستم عامل های توزیع شده و چگونگی مدیریت فرایند در این گونه سیستم عامل ها. همچنین یک نمونه سیستم عامل توزیع شده به نام آموبا یا آمیب مد نظر قرار داده شد و اهداف، معماری و مدل های مختلف آن مورد بررسی قرار گرفت و مدیریت فرایند و ارتباطات درون فرایندی در آن را بررسی نموده و به این نتیجه رسیدیم که میتوانیم مکانیزمی ایجاد کنیم تا برای تحقق دانلود ، مهاجرت ، کنترل استثناها ، حالت برداری ، شبیه سازی و اشکال زدایی کارآمد باشد. طی ۲۰ سال گذشته گرایش به سمت شبکه های پر سرعت تر، سیستم های توزیع شده و معماری کامپیوتر های چند پردازنده ای، نشان می دهد که در آینده استفاده از سیستم های توزیع شده و نیاز به سیستم عامل های توزیع شده افزایش یافته و این سیستم عامل ها به شکل چشمگیری درکاربردهای مختلف مورد استفاده قرار بگیرند. به همین منظور پیشنهاد می شود که محققان و پژوهشگران علوم کامپیوتر با توجه به مطالبی که در این مقاله بیان شد راهکارهایی جهت مدلسازی و پیاده سازی یک سیستم عامل توزیع شده در محیط های دانشگاهی ارائه داده و مباحث مختلفی بعنوان نمونه مقیاس پذیری را در آن مورد بررسی قرار دهند.

۱۸. مراجع

1. Pérez, H., Gutiérrez, J.J., Peiró, S., & Crespo, A. (2017, January). Distributed architecture for developing mixed-criticality systems in multi-core platforms. Journal of Systems and Software, 123, 145-159.

2. Hollis, S.J., Ma, E., & Marculescu, R. (2016, October). NOS: A nano-sized distributed operating system for many-core embedded systems. Conference: 2016 IEEE 34th International Conference on Computer Design (ICCD).

3. Tanwar, S., Aggarwal, Y., & Dewan, S. (2013, October). Distributed Operating System. International Journal of Research in Information Technology (IJRIT) , 1(10), 50-56.

4. Mullender, S.J. (1987, September). Process Management in a Distributed Operating System. Proceedings of the International Workshop on Experiences with Distributed Systems, 38-51.

5. Remzi Arpaci, D., & Dusseau, A.A. (2013, August). Operating Systems: Three Easy Pieces. Lulu Press.

6. Kai, H., Dongarra, J., & C. Fox, G. (2011, October). Distributed and Cloud Computing: From Parallel Processing to the Internet of Things. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

7. Buckl, C., Knoll, A., & Schrott, G. (2006, November). Model-Based Development of Fault-Tolerant Embedded Software. Leveraging Applications of Formal Methods, Verification and Validation, 2006. ISoLA 2006. Second International Symposium.

8. Pradeep K, S. (1996, December). Distributed Operating Systems: Concepts and Design (1 ed.). Wiley-IEEE Press.

9. Kon, F., Campbell, R.H., D Mickunas, M., Nahrstedt, K., & Ballesteros, F.J. (2000, August). 2K: a distributed operating system for dynamic heterogeneous environments. High-Performance Distributed Computing, 2000. Proceedings. The Ninth International Symposium, , 201-208.

10. William, S. (2009). Operating Systems: Internals and Design Principles (6th ed.). NJ, USA: Prentice Hall Press Upper Saddle River.

11. Tanenbaum, A.S., Kaashoek, M.F., Renesse, R.V., & Bal, H.E. (1991, July). The Amoeba distributed operating system — A status report. Computer Communications, Department of Mathematics and Computer Science, Vrije Universiteit, De Boelelaan 1081, NL-1081, HV, Amsterdam, The Netherlands, 14(6), 324-335.

12. Sloman, M., & Moffett, J. (1993, December). Policy Hierarchies for Distributed Systems Management. IEEE Journal on Selected Areas in Communications, 11(9), 1404 - 1414.

13. Abraham, S., Galvin, P.B., & Gagne, G. (2008, July). Operating System Concepts. Wiley Publishing.

14. Kaashoek, M.F., & Tanenbaum, A.S. (1991, May). Group communication in the Amoeba Distributed Operating System. Conference: Distributed Computing Systems, 1991., 11th International Conference, , 222-230.

15. Tanenbaum, A.S., & Mullender, S.J. (1981, July). An overview of the Amoeba distributed operating system. ACM SIGOPS Operating Systems Review, 15(3), 51-64.

16. Andrew S, T. (2007). Modern Operating Systems. NJ, USA: Prentice Hall Press Upper Saddle River.

17. Dasgupta, P., LeBlanc, Jr, R.J., Ahamad, M., & Ramachandran, U. (1991, November). The Clouds Distributed Operating System. IEEE Computer Society Press Los Alamitos, 24(11), 34-44.

18. Tanenbaum, A.S., Van Renesse, R., Van Staveren, H., J. Sharp, G., & Mullender, S.J. (1990, December). Experiences with the Amoeba distributed operating system. Communications of the ACM, 33(12), 46-63.

19. Mullender, S.J., Rossum, G.V., Tanenbaum, A.S., Renesse, R.V., & Staveren, H.V. (1990, May). Amoeba: a distributed operating system for the 1990s. Published in: Computer, 23(5), 44 - 53.

20. Renesse, R.V., & Tanenbaum, A.S. (1989, March). The performance of the Amoeba distributed operating system. Article in Software Practice and Experience, , 223-234.

21. Carson, J.H. (1998, March). A distributed operating system for a workstation environment. Computers and Communications, 1988. Conference Proceedings., Seventh Annual International Phoenix Conference, , 213-217.

22. Gary, N. (2000). Operating Systems: A Modern Perspective, Second Edition. Published by Addison-Wesley Longman Inc.

23. Tanenbaum, A.S., & Sharp, G.J. (1998, April). The Amoeba Distributed Operating System. Vrije Universiteit, De Boelelaan 1081a, Amsterdam, The Netherlands.

24. Cheriton, D.R., & Zwaenepoel, W. (1983, October). The Distributed V Kernel and its Performance for Diskless Workstations. Proc Ninth ACM Symp, on Operating Systems Principles, , 128-140.

25. Lampson, B.W., & Redell, D.D. (1980, February). Experience with Processes and Monitors in Mesa. Communications of the ACM, 23(2), 105-117.