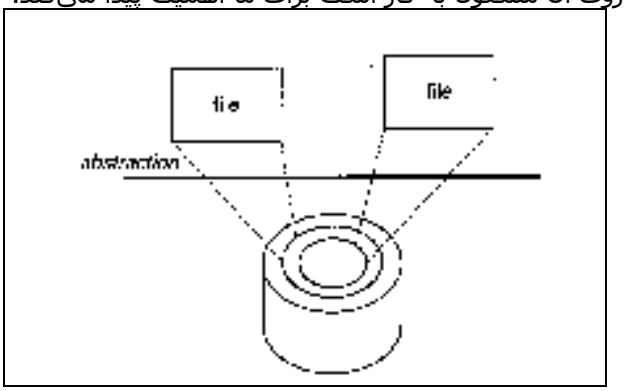
آشنایی با تئوری ماشین‌های مجازی

کامپیوترهای مدرن امروزی جزء وسائل بسیار پیشرفته ساخته شده به دست بشر تلقی می‌گردند و دلیل اینکه ما توانستیم به چنین کاری نائل آییم، به احتمال قریب به یقین بخاطر دانش ما در مدیریت پیچیدگی‌ها [۱]می‌باشد. کامپیوترها از میلیونها چیپ تشکیل شده‌اند که هر کدام از بیلیونها ترانزیستور استفاده می‌کنند و تمام این قطعات به وسایل ورودی/خروجی [۲]و شبکه‌های مختلف متصل هستند تا در نهایت سیستمهای نرم‌افزاری بتوانند بر روی تمام اینها سوار شده و کار نهایی را انجام دهند. نرم‌افزارهای متعدد گرافیکی، آموزشی و ... نیز وجود دارند که کارهای ما را به انجام می‌رسانند.

کلید اصلی مدیریت پیچیدگی در سیستمهای کامپیوتری همانا انجام کارها بصورت انتزاعی [۳]و قرار دادن ظواهری [۴] بین هر مرحله و مرحله بعد است تا کاربر اصلا حس نکند که این میان چه اتفاقی در حال افتادن است و او در چه سطحی قرار گرفته است. این انتزاع باعث می‌شود که طراحی در مراحل بالاتر آسان شده و بدون توجه به مراحل پایین انجام شود. به عنوان مثال این حقیقت که هارد دیسک به سکتورها و شیارهایی تقسیم می‌شود عملا با وجود سیستم‌عامل برای یک نرم‌افزار بی‌معنا شده و نرم‌افزار تنها هارددیسک را بصورت یک محیط کاملا قابل لمس و با یک سری فایل که هر کدام اندازه خاصی دارند می‌بیند. در این حالت یک برنامه‌نویس براحتی می‌تواند برنامه خود را بنویسد، بدون توجه به اینکه واقعا در لایه‌های پایینتر چه دنیایی از پیچیدگی قرار دارد!

انتزاع خود دارای مراحل بیشماری است که از مراحل سطح پایین که عملا با سخت‌افزار محض سروکار دارد شروع شده و به مراحل سطح بالا که مراحل انتزاعی نرم‌افزاری هستند ختم می‌گردد. در مراحل سطح پایین فقط سخت افزار و اجزاء فیزیکی وجود دارند. در مراحل سطح بالا اجزاء تشکیل دهنده همه نرم‌افزاری هستند و محدودیتهای سخت‌افزاری را ندارند. در این کتاب ما بیشتر به سطوحی که در واقع جداساز بین مراحل سخت افزاری و نرم‌افزاری هستند می‌پردازیم، عملا سطوح حد واسط بین سخت‌افزار و نرم‌افزار، از جایی که نرم‌افزار از سخت‌افزار جدا شده و عملا ماشین فیزیکی که نرم‌افزار بر روی آن مشغول به کار است برای ما اهمیت پیدا می‌کند.



فایلها در واقع انتزاعی از دیسک هستند. یک سطح از انتزاع در واقع ظاهر ساده‌تری را برای کاربران بالایی خود فراهم می‌آورد.

نرم‌افزار کامپیوتر توسط یک *ماشین* اجرا می‌شود (اصطلاحی که از اوان ایجاد کامپیوتر باب بوده است، امروزه معمولا از کلمه پلاتفورم بیشتر بجای ماشین استفاده می‌کنند). از دید سیستم‌عامل، یک ماشین عملا از قطعات سخت افزاری همچون یک و یا بیشتر CPU و همچین مقداری RAM و وسایل ورودی/خروجی تشکیل شده است. منتها فراموش نکنید که کاربرد این واژه نسبی است، یعنی همانطور که گفته شد هرگاه سیستم‌عامل از کلمه ماشین استفاده کند، منظور آن اجزاء صرفا سخت‌افزاری است، ولی هرگاه یک نرم‌افزار عادی از واژه ماشین استفاده کند، منظورش سیستم‌عامل مورد استفاده به همراه گوشه‌ای از جزئیات سخت‌افزاری است که توسط لایه سیستم‌عامل جداسازی نشده است و نرم‌افزار باید مستقیما با آنها درگیر باشد.

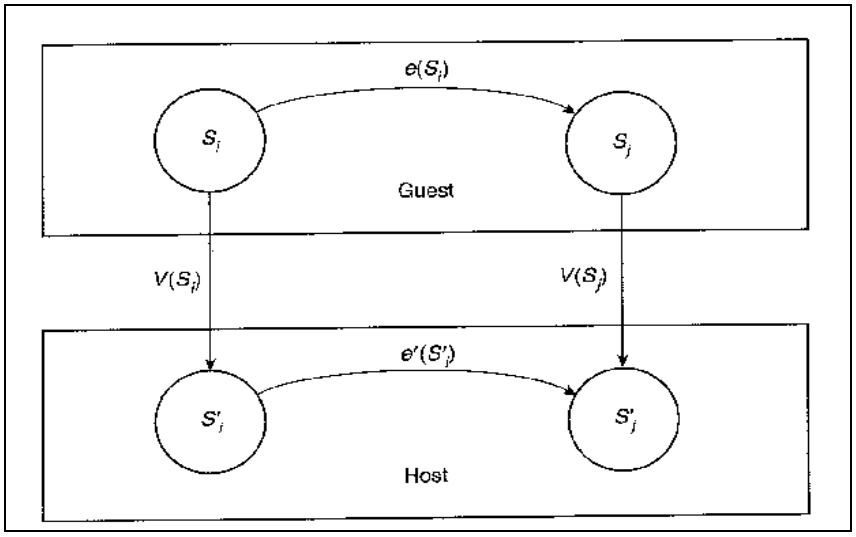
به اندازه کافی در مورد انتزاع بحث کردیم، حال اجازه دهید در مورد آن فاکتور دوم که مدیریت پیچیدگی را آسان می‌کند صحبت کنیم: قرار دادن ظواهری آسان بین هر مرحله از انتزاع. وجود چنین ظواهری باعث می‌شود که بتوان عملا مراحل

طراحی یک کامپیوتر را توسط چندین تیم، مثلا یک تیم نرم‌افزاری و یک تیم سخت‌افزاری انجام داد، بدون آنکه آنها چندان با کار یکدیگر کاری داشته باشند. وجود جدول دستوری CPU یک [۵] نمونه از این ظاهر سازی است. به عنوان مثال طراحان AMD و یا Intel ریزپردازنده‌هایی را طراحی می‌کنند که از جدول دستوری استاندارد IA-32 استفاده می‌کند. از آن طرف کامپایلر نویسان RedHat و یا مایکروسافت نیز کامپایلرهایی می‌نویسند که دستورات را به این استاندارد دی‌کد کند. اگر هر دو این گروهها کار خود را به درستی انجام دهند، نرم‌افزار کامپایل شده بر روی هر ماشینی که از استاندارد IA-32 پشتیبانی کند به درستی و تمام و کمال اجرا می‌شود. ظاهر موجود در سیستم‌عامل نیز مثال خوب دیگریست که می‌توان به آن اشاره کرد. هر سیستم‌عامل با داشتن مجموعه دستورات خود عملا ظاهری را برای برنامه‌نویسان برنامه‌های مختلف فراهم می‌کند که آنها تنها می‌توانند با صدا زدن آن توابع کار خود را براحتی انجام دهند، بدون توجه به اینکه واقعا چه اتفاقی در پایین در حال رخ دادن است. حتی ممکن است کل ساختار یک سیستم‌عامل در طی سالیان عوض شود و با سخت‌افزارهای متعددی کار کند، و لیکن او خود مسئول بروز نگاه داشتن توابع خود است و تا زمانی که اینکار را انجام دهد، تمام برنامه‌هایی که حتی چند سال پیش نیز برای آن نوشته شده بودند، کماکان به کار خود ادامه خواهند داد. حال اگر چندین سیستم‌عامل از یک مجموعه دستورات پیروی کنند، عملا نرم‌افزار نوشته شده برای یکی از آنها بر روی بقیه نیز براحتی کار خواهد کرد.

بر خلاف تمام مزایایی که این ظواهر فراهم می‌کنند، آنها خود عامل محدودیت هستند. برنامه‌هایی که برای یک مجموعه دستورات سیستم‌عاملی نوشته شده‌اند برای سیستم‌عامل دیگر کار نمی‌کنند. تنها استاندارد IA-32 وجود ندارد و CPU های دیگری نیز هستند که برای استانداردهای دیگر نوشته شده‌اند و این یعنی که سیستم‌عاملهای طراحی شده برای IA-32 بر روی آنها کار نخواهند کرد و این باعث بوجود آمدن سیستم‌عاملهای مختلف (مثلا ویندوز و لینوکس) می‌گردد. به عنوان یک قانون کلی باید گفت که وجود تنوع در مجموعه دستورالعملها و قوانین و کلا ظواهر مختلف باعث بوجود آمدن سیستم عاملهای مختلف و طبعا برنامه‌های مختلف می‌گردد که این خود باعث ابداعات بسیار و جلوگیری از رکود فکری می‌شود؛ و لیکن در عمل این خود بزرگترین عامل محدود کننده ارتباط بین سیستمهای مختلف است، مخصوصا در دنیای شبکه شده امروزی که دوست داریم نرم‌افزارها را به همان آسانی که داده‌های خود را منتقل می‌کنیم، از سیستمی به سیستم دیگر منتقل کرده و با آنها کار کنیم.

حتی پایینتر از لایه‌هایی که ارتباط بین سخت‌افزار و نرم‌افزار را فراهم می‌کنند، خود لایه‌های سخت افزاری مشکل‌ساز بوده و عملا محدودیتهایی را حتی در مراحل بالاتر برای ما فراهم می‌کنند. امروزه با ظهور سیستم‌عاملهای پیشرفته و زبانهای سطح بالا، عملا کاربران و برنامه‌نویسان چندان با مشکلات محض سخت‌افزاری دست و پنجه نرم نمی‌کنند و اصلا از آن پایینها خبر ندارند، و لیکن در معدود مواردی باز هم این مشکلات چهره کریه خود را نمایش می‌دهند و کاربر/برنامه‌نویس سطح بالای ما را با مشکل روبرو می‌کنند. نباید فراموش کنیم که بسیاری از سیستم عاملها صرفا برای معماری خاصی از CPUها طراحی شده‌اند، بدین معنا که تمام سخت افزار موجود در ماشین تنها از طریق سیستم‌عامل نصب شده قابل دسترسی است که این خود دو نوع مشکل را ایجاد می‌کند: اول اینکه باید در عمل سیستم‌عاملی را انتخاب کنیم که اکثر برنامه‌های مورد علاقه ما را پشتیبانی کند، در واقع بتواند ارتباط آنها را با سخت‌افزار ماشین مورد بحث برقرار نماید، حال تکلیف برای آن سری دیگر از نرم‌افزارهای بسیار خوبی که ممکن است در سیستم‌عاملهای دیگری که ماشین ما را پشتیبانی نمی‌کنند باشد و ما با کار کردن به آنها خو گرفته‌ایم چیست؟ و مشکل دوم که از اولی نیز بزرگتر است این است که ما تمام ماشین خود را وقف یک سیستم‌عامل کرده‌ایم و در صورتی که این سیستم‌عامل به هر دلیلی خراب شود، حتی اگر سخت افزار ما نیز سالم باشد (که معمولا هست)، دیگر اصلا نمی‌توانیم از ماشین استفاده کنیم!

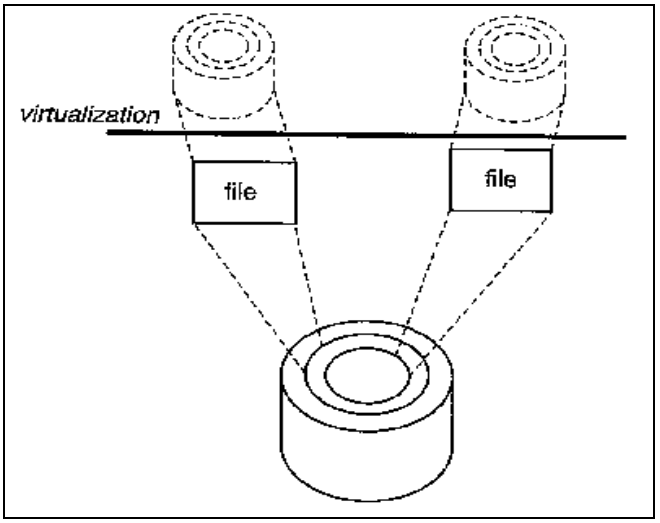
مجازی سازی روشی را برای خلاص شدن از شر این قوانین که هر روز هم بیشتر می‌شوند فراهم می‌کند. هنگامی که یک سیستم و یا زیر سیستم، مثلا یک CPU و یا هارد دیسک مجازی‌سازی می‌شوند، ظواهر مربوط به آنها و تمامی منابعی که از طریق آن ظواهر قابل دسترسی هستند به ظواهر و منابعی از سیستم واقعی که در حال کار است نگاشت می‌شوند. اگر بخواهیم مجازی‌سازی را بصورت رسمی تعریف کنیم، باید گفت که مجازی‌سازی عبارتست از یک همریختی [۶]که یک سیستم مهمان [V] را به یک سیستم میزبان [۸] نگاشت می‌کند(بر اساس تعریف Popek و Goldberk در سال 1974). این همریختی که در شکل زیر نشان داده شده است، حالت مهمان را به حالت میزبان نگاشت می‌دهد (تابع V در شکل) و به ازاء هر سری از عملیات، e، که باعث تغییر حالت در مهمان شود (تابع e حالت Si را به Sj تبدیل می‌کند) عملیات مشابهی، e´، در میزبان انجام می‌شود که Si´ را به Sj´ می‌برد. هر چند که این همریختی عملا می‌تواند نشان دهنده انتزاع نیز باشد، ولی ما در اینجا بین انتزاع و ظواهر فرق می‌گذاریم، بدین صورت که مجازی سازی عملا باعث بیشتر و یا کمتر انتزاعی تر شدن سیستم نشده و عملا مقدار انتزاع بین ماشین مجازی و واقعی یکسان است.



تعریف علمی مجازی سازی همانا ساخت یک همریختی بین سیستم مهمان و میزبان می‌باشد

اجازه دهید مثالی عملی از این مورد را بازگو کنیم. همان مثال هارد دیسک معروف را در نظر بگیرید. اگر بخواهیم یک هارد دیسک واقع در سیستم‌عامل را به چندین هارددیسک مجازی تقسیم کنیم، به ازای هر هارددیسک مجازی یک فایل بزرگ بر روی هارد واقعی تعریف می‌کنیم. هارد دیسکهای مجازی نیز برای خود سکتور و قطاع مشخصی را دارند، هر چند که اندازه آن به اندازه هارد سیستم میزبان نمی‌باشد. برنامه مجازی سازی عمل نگاشت بین هارددیسک واقعی و هارددیسکهای مجازی را فراهم می‌آورد (تابع V در همریختی). عمل نوشتن درون هارددیسک مهمان (تابع e در همریختی)، در عمل باعث ایجاد تقاضایی برای نوشتن در هارد دیسک میزبان می‌گردد (تابع e´ در همریختی) در این مثال عمل نوشتن بر روی هارددیسکهای مجازی با تغییر سکتور و قطاع سر و کار داشت و از این لحاظ انتزاعی در مقایسه با حالت واقعی شکل نگرفت.[۹]

مفهوم مجازی سازی نه تنها می‌تواند در مورد اجزاء مختلف کامپیوتری عملی شود، بلکه می‌تواند یک کامپیوتر کامل را نیز شبیه‌سازی نماید. به عنوان مثال ماشین مجازی نصب شده بر روی یک کامپیوتر Apple می‌تواند یک ویندوز ۳۲ بیتی را که می‌شود نرم‌افزارهای ویندوزی را نیز درون آن نصب کرد، شبیه‌سازی کند. عملا یک ماشین مجازی می‌تواند محدودیتهای سخت‌افزاری و نرم‌افزاری یک ماشین واقعی را پشت سر گذاشته و درجه بیشتری از راحتی و جابجایی را برای نرم‌افزارها فراهم آورد.



پیاده‌سازی دیسکهای مجازی. مجازی سازی می‌تواند منابع و یا رابطهای متفاوتی را در یک درجه از انتزاع فراهم آورد

انواع مختلفی از ماشین‌های مجازی وجود دارد که می‌توان بصورت همزمان آنها را بر روی یک ماشین نصب کرد و این ماشین‌های مجازی برای هر کاربر محیط دلخواه او را شبیه‌سازی کنند. حتی می‌توان چندین ماشین مجازی را بر روی چندین ماشین واقعی در محیط کار نصب نمود که اینکار درصد امنیت را نیز بالا می‌برد. می‌توان یک سرور قوی با چندین پردازشگر را عملا به چندین ماشین مجازی تقسیم نمود تا هم سرعت خود سرور پایین نیاید و هم اینکه از پردازشگرهای دیگر نهایت استفاده انجام شود. [۱۰]

ماشین‌های مجازی همچنین می‌توانند از تکنیکهای emulation برای[۱۱] اجرای یک برنامه بر روی سیستمهای مختلف استفاده کنند. به عنوان مثال سیستمی که از دستوران PowerPC استفاده می‌کند می‌تواند عملا به قسمی emulate شود که به نظر آید از دستورات IA-32 استفاده می‌کند، بدین صورت می‌توان برنامه‌هایی که صرفا بر روی IA-32 اجرا می‌شوند را درون این سیستم نیز اجرا نمود. این کار می‌تواند هم در سطح سیستم (همان simulation) و با شبیه‌سازی کل سیستم‌عامل انجام شود (مثلا نصب ویندوز درون Macintosh) و هم در سطح برنامه و یا پردازش در حال انجام (به عنوان مثال اجرای Excel درون سیستم عاملهای Sun Solaris و یا SPARC). علاوه بر emulation، ماشین‌های مجازی می‌توانند در هنگام اجرا بصورت آنی فایلهای binary در حال کار را نیز بهینه کنند و سرعت پردازشها را بالا ببرند.

ماشین‌های مجازی که در اینجا مثال زده شد تنها آنهایی هستند که برای کار با معماری ماشین‌های واقعی موجود طراحی شده‌اند. با اینحال ماشین‌های مجازی وجود دارند که برای آنها هیچ معماری فیزیکی واقعی وجود ندارد. امروزه دیگر برای طراحان زبانهای برنامه‌نویسی امری بسیار عادی است که در ابتدا ماشین مجازی را طراحی کنند و زبان خود را به گونه‌ای بسازند که تمام دستورالعملها را به دستورالعملهایی که برای آن ماشین مجازی قابل فهم است ترجمه کند. حال هر ماشین واقعی که این ماشین مجازی را درون خود نصب داشته باشد، براحتی می‌تواند برنامه‌های نوشته شده برای آن زبان خاص را در خود اجرا نماید. قدرت این روش عملا در زبان برنامه‌نویسی سطح بالای جاوا به همراه ماشین مجازیش اثبات شده است.

گروههای مختلفی همچون برنامه‌نویسان، طراحان زبان و کامپایلرنویسان و طراحان سخت‌افزار در امر ماشین‌های مجازی سرمایه‌گذاری کرده و ماشین‌های مختلفی را می‌سازند. بدون توجه به نوع ماشین مجازی و کارهایی که انجام می‌دهد، یک سری تکنولوژیهای مشترک بین تمامی این ماشین‌ها وجود دارد که برای ساخت یک ماشین مجازی،باید از آن تکنولوژیها استفاده کرد. هدف این مقاله کوچک در وهله اول آشنایی با این تکنولوژیهای مشترک می‌باشد و در مرحله بعد آشنایی بسیار مقدماتی با ماشین‌های مجازی مختلف اهداف بعدی ما را تشکیل می‌دهند.

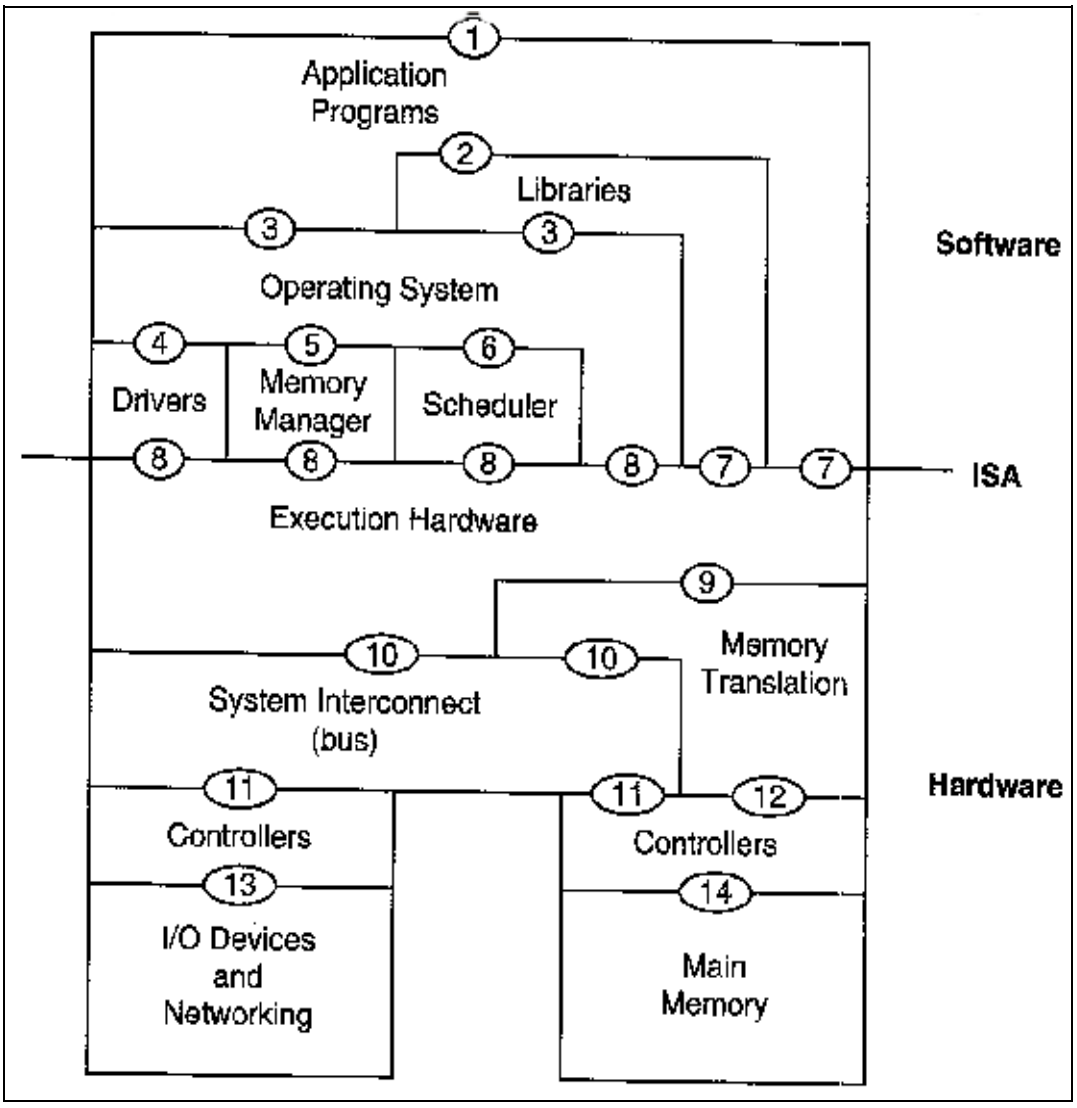
معماری کامپیوتر

همانطور که در آینده خواهید دید، هدف اصلی یک ماشین مجازی در واقع ساخت پلی است بین معماریهای مختلف کامپیوتری و همین وفاداری به اصول یک معماری است که یک ماشین مجازی خوب را که می‌تواند کل آن معماری را شبیه سازی کند، می‌سازد. بنابر این بهتر است از همین ابتدا معماری کامپیوتر را تعریف کرده و خلاصه‌ای از آن را بیاوریم.

معماری یک ساختمان در واقع نحوه کار آن ساختمان و شکل آنرا به ساکنانش نمایش می‌دهد، نه جزئیات موجود در ساختمان، همچون جزئیات لوله‌کشی ساختمان و اینکه از چه نوع لوله‌ای استفاده شده و یا شرکت سازنده آجرهای ساختمان چه شرکتی بوده و یا جزئیات دیگر. در کامپیوتر هم اوضاع به همین منوال است و وقتی از کلمه معماری استفاده می‌شود، منظور جزئیات آن سیستم نیست، بلکه شکل و شمایل آن سیستم برای کسانی که می‌خواهند با آن کار کنند و امکاناتی که سیستم فراهم می‌آورد مورد نظر است. معماری یک سیستم معمولا با توضیح دقیق رابطی [۱۲] که آن سیستم فراهم می‌آورد به همراه واکنش منطقی منابع موجود در سیستم در صورتی که از طریق رابط با آنها کار شود، بیان می‌شود. به نحوه پیاده‌سازی [۱۳] یک معماری درون سیستم، پیاده‌سازی می‌گویند. تا واقعا به یک مثال نگاه نکنیم، مفهوم این قسمت را متوجه نمی‌شوید. به شکل زیر نگاه کنید. همانطور که می‌بینید برنامه‌های موجود در سیستم در یک مستطیل نشان داده شده‌اند (شماره ۱) و کتابخانه‌های موجود در مستطیل دیگری نمایش یافته‌اند (شماره ۲). به خطی که بین این دو مستطیل قرار دارد همان رابط می‌گوییم، به عبارت دیگر اگر یک برنامه بخواهد با یک کتابخانه کار کند، باید با این خط صحبت کند و این خط از طرف دیگر با کتابخانه موجو رابطه برقرار کرده و ارتباط بین این دو مستطیل را به نحوی که خود می‌خواهد برقرار می‌کند. بنابر این در این شکل مفهوم رابط به خوبی مشخص است. به هر یک از این مستطیل ها نیز پیاده‌سازی و یا همان implementation می‌گوییم. معماری یک مفهوم است، همین که در اینجا می‌بینید نرم‌افزارها باید در یک مستطیل بوده و کتابخانه‌ها در مستطیل دیگر، این خود معماری است، نقشه‌ای است که به ما میفهماند بین کتابخانه‌ها و نرم‌افزارها باید فاصله وجود داشته باشد و نباید در یک جا قرار گیرند. ولی نحوه پیاده‌سازی هر مستطیل و اینکه مثلا درون مستطیل نرم‌افزارها (شماره ۱) چه اتفاقی باید بیفتد و چه کارهایی انجام گیرد همان پیاده‌سازی نام دارد. بنابر این می‌توان گفت که برای هر معماری، چندین نوع پیاده‌سازی وجود دارد.

مثلا یک نوع پیاده‌سازی که سرعت را بالا می‌برد ولی در عوض مصرف انرژی را افزایش می‌دهد، و یا پیاده‌سازی دیگری که سرعت کمتری دارد، ولی مصرف انرژی کمتری را نیز طلب می‌کند.[۱۴]

حال اگر دوباره به شکل نگاه کنید، رابطهای مختلفی را می‌بینید. رابط دیگری که در پایین نرم‌افزارها و بین نرم‌افزارها و سیستم‌عامل قرار دارد، رابط شماره ۳ است. رابط دیگری بین کنترلرها و bus وجود دارد که سیگنالها را از bus به کنترلرها می‌فرستد (رابط ۱۱)، رابطی که مربوط به سیگنالهای دسترسی به حافظه است که از پردازشگر بیرون آمده‌اند (رابط ۱۲) و ... از میان تمامی این رابطها، ما به آنهایی که بین مرز نرم‌افزار/سخت‌افزار هستند علاقه بیشتری داریم.



معماریهای مختلف کامپیوتر. لایه‌های نشان داده شده بصورت عمودی با رابطهای خود در ارتباطند.

همانطور که در شکل می‌بینید، ISA که [۱۵] خود از رابطهای ۷ و ۸ تشکیل شده است، نقش واسط بین لایه سخت‌افزاری و نرم‌افزاری را بازی می‌کند. مفهوم ISA اولین بار در کامپیوترهای IBM 360 سری mainframe مطرح شد. این کامپیوترها به گونه‌ای طراحی شده بودند که می‌توانستند از سخت‌افزارهای متعددی استفاده کنند و بدین صورت میشد یک معماری خاص را به چندین صورت مختلف و با هزینه‌های مناسب برای هر مشتری پیاده کرد. این تنوع در سخت‌افزارها باعث می‌شد که قوانین سختی بین لایه سخت‌افزاری و نرم‌افزاری حاکم گردد تا برنامه‌های نوشته شده برای این معماری بتواند بر روی همه انواع پیاده‌سازیها کار کند. این شد که لایه ISA معنی واقعی خود را پیدا کرد و اولین

بار در آنجا شکل گرفت.

دو قسمت از ISA در تعریف یک ماشین مجازی نقش ایفا می‌کند. قسمت اول قسمتی است که با نرم‌افزارهای سیستم بصورت مستقیم در تماس است (رابط شماره ۷ در شکل) که از آن به عنوان ISA سطح کاربر یاد می‌کنیم. قسمت دیگر قسمتی است که تنها با نرم‌افزار مدیر مثلا سیستم‌عامل سر و کار دارد و مسئولیت مدیریت منابع سخت‌افزاری را بر عهده دارد. به این قسمت ISA سیستم [۱۷] می‌گوییم. طبیعی است که نرم‌افزار مدیر می‌تواند تمام کارهای ISA سطح کاربر را نیز انجام دهد. در شکل، رابط شماره ۷ تنها از ISA سطح کاربر [۱۶] تشکیل شده، در حالی که رابط ۸، هم ISA سیستم و هم ISA کاربر را پوشش می‌دهد.

برنامه ای که از ABI استفاده می‌کند می‌تواند به منابع سخت‌افزاری و سرویسهای موجود در سیستم دسترسی داشته باشد. ABI خود از دو قسمت تشکیل شده است. قسمت اول مجموعه‌ای از تمام دستورالعملهایی است که یک کاربر می‌تواند به آنها دسترسی داشته باشد (رابط شماره ۷ در شکل)، بدین صورت کاربر خود می‌تواند بطور مستقیم با سخت افزار تماس داشته باشد؛ و قسمت دیگر نحوه دسترسی به منابع سیستمی را مشخص می‌کند. یک کاربر برای دسترسی به منابع سیستمی نمی‌تواند بطور مستقیم از رابط شماره ۷ استفاده کند و برای اینکار باید ابتدا با سیستم‌عامل تماس برقرار کرده (رابط شماره ۳ در شکل) و در صورت اجازه سیستم عامل، به این منابع دسترسی پیدا کند. به رابط شماره ۳، رابط صدا زن سیستم [۱۸]می‌گویند. رابط صدا زن می‌تواند مجموعه‌ای از دستورالعملها را شامل شود که سیستم‌عامل می‌تواند از طرف کاربر انجام دهد (البته بعد از احراز هویت کاربر و اطمینان از اینکه کاربر اجازه اجرای این دستورالعملها را دارد[۱۹]). صدا زدن این رابط نیز دقیقا همانند فراخوانی یک تابع عادی است، با این تفاوت که از قوانین خاص خود پیروی کرده و برای ارسال متغیر به آن نیز معمولا از stack و مقادیر ذخیره شده در حافظه استفاده می‌شود. اجراء برنامه‌ای که از ABI استفاده می‌کند بر روی سیستم‌های دیگر منوط به استفاده از سیستم‌عامل و ISA مشابهی است که برنامه تحت آن نوشته شده است.

استفاده از API دقیقا استفاده از رابط شماره ۲ بجای رابط شماره ۳ است. به بیان دیگر بجای ارتباط با سیستم‌عامل و درخواست از او برای اجرای دستورات سیستمی، از یک سری کتابخانه بدین منظور استفاده می‌کنیم. البته از طریق کتابخانه‌ها می‌توان به سرویسهایی که سیستم عاملها نیز فراهم می‌آورند دسترسی داشت. معمولا از چنین کتابخانه‌هایی در نرم‌افزارها استفاده نمی‌شود، مگر اینکه این کتابخانه‌ها بسیار معروف و پایدار باشند، چرا که استفاده از چنین کتابخانه‌ای عملا باعث وابستگی برنامه به آن کتابخانه خاص می‌شود و در هر سیستمی که بخواهیم آنرا نصب کنیم، باید آن کتابخانه نیز موجود باشد. از کتابخانه‌های بسیار معروفی که می‌توان نام برد، کتابخانه clib است که از زبان C تحت UNIX پشتیبانی می‌کند. توابع موجود در API معمولا از یک و یا چند ABI استفاده می‌کنند. حتی بعضی از API ها صرفا در حکم کاغذ کادو [۲۰]هستند، بدین معنا که از خودشان هیچ نداشته و صرفا با صدا زدن آن API، دستورات سیستم عاملی مورد نظر از طرف API اجرا می‌شود.[۲۱]

مبانی ماشین‌های مجازی

برای اینکه بخواهیم مفهوم ماشین مجازی را درک کنیم، ابتدا باید تعریف کنیم که اصلا منظور ما از *ماشین* چیست؟ همانطور که قبلا گفته شد، مفهوم *لغت ماشین* کاملا بسته به دیدگاه گوینده دارد. از دید یک پردازش که در حال اجرای برنامه کاربر است، ماشین همان فضای حافظه منطقی اختصاص داده شده به او به همراه رجیسترها و دستورالعملهایی است که به پردازش [۲۲]اجازه اجراء کد را می‌دهد. یک پردازش به هیچ عنوان نمی‌تواند بطور مستقیم با دستگاههای ورودی/خروجی کار کند و برای اینکار نیاز به سیستم‌عامل دارد. بطور کلی ماشین از دید یک پردازش همان سیستم‌عامل و لایه‌های سخت‌افزاری زیر سیستم‌عامل است که می‌تواند از طریق سیستم‌عامل به آنها دسترسی داشته باشد.

از دیدگاه یک سیستم عامل، یک سیستم کامل به همراه تمامی پردازشها که توسط کاربران مختلف در حال اجرا هستند از طریق ماشین اجرا می‌شود. بنابراین ماشین از دید سیستم عامل، تمامی سخت‌افزارهای موجود در لایه‌های زیرین هستند که سیستم‌عامل می‌تواند از طریق لایه ISA به آنها دسترسی داشته باشد.

در عمل یک ماشین مجازی، نرم‌افزارها (یک پردازش و یا یک سیستم کامل) را به همان گونه‌ای اجرا می‌کند که نرم‌افزار در ماشین واقعی اجرا می‌شود. ماشین مجازی در واقع ترکیبی از ماشین واقعی به همراه نرم‌افزار مجازی‌ساز است. ماشین مجازی می‌تواند منابع مختلفی را چه از نظر تعداد و چه از نظر نوع داشته باشد. به عنوان مثال یک ماشین مجازی می‌تواند پردازشهای بیشتری را نسبت به ماشین واقعی داشته باشد و حتی این پردازشها ممکن است از دستورات متفاوتی نسبت به دستورات پشتیبانی شده توسط پردازشگر واقعی، تشکیل شده باشند. لازم به ذکر است که بازدهی یکسان با سیستم واقعی از یک سیستم مجازی معمولا توقع نمی‌رود و جزء شرایط پیش فرض آن نیست. غالبا کارکرد سیستم مجازی از سیستم واقعی در اجرای نرم‌افزاری که دقیقا برای همان سیستم واقعی نوشته شده باشد، پایین‌تر است.

با توجه به قانون همریختی که قبلا بیان شد، مجازی سازی شامل دو قسمت می‌گردد: اول نگارش منابع مجازی به منابع واقعی موجود در سیستم واقعی و دوم استفاده از دستورات سیستم واقعی برای اجرای دستورات سیستم مجازی که این خود شامل شامل شبیه سازی ABI و یا ISA ماشین مجازی نیز می‌گردد.

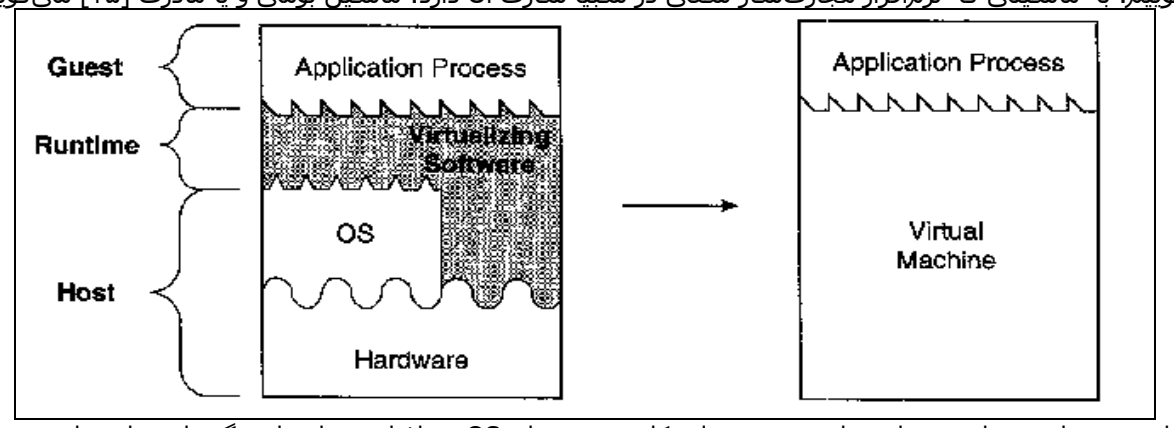
همانگونه که ماشین را از دید پردازش و دید سیستم تعریف کردیم، دو گونه ماشین‌های مجازی نیز داریم:

ماشین‌های مجازی پردازشی [۲۳]

ماشین‌های مجازی سیستمی [۲۴]

همانگونه که از نام آن بر می‌آید، یک ماشین مجازی پردازشی می‌تواند یک پردازش را بصورت مجازی اجراء نماید. شکل زیر نمونه‌ای از این ماشین را نمایش می‌دهد. در این شکل و شکل بعد، قسمتهایی که با یکدیگر تجانس دارند و یکدیگر را درک می‌کنند با رنگ یکسان نشان داده شده‌اند.

در ماشین‌های مجازی پردازشی، نرم‌افزار مجازی‌ساز در لایه ABI قرار گرفته است. این نرم‌افزار دقیقا بالای ترکیب سخت‌افزار/سیستم‌عامل و در زیر پردازش مجازی‌سازی شده جای گرفته است. این نرم‌افزار هم دستورات سطح کاربر و هم صدا زدن سیستم‌عامل توسط پردازش مجازی شده را شبیه‌سازی می‌کند. در اصطلاح به سیستمی که نرم‌افزار مجازی‌ساز روی آن اجرا می‌شود سیستم میزبان و به نرم‌افزاری که درون نرم‌افزار مجازی‌ساز اجرا می‌شود، مهمان می‌گوییم. به ماشینی که نرم‌افزار مجازی‌ساز سعی در شبیه‌سازی آن دارد، ماشین بومی و یا مادری [۲۵] می‌گوییم.



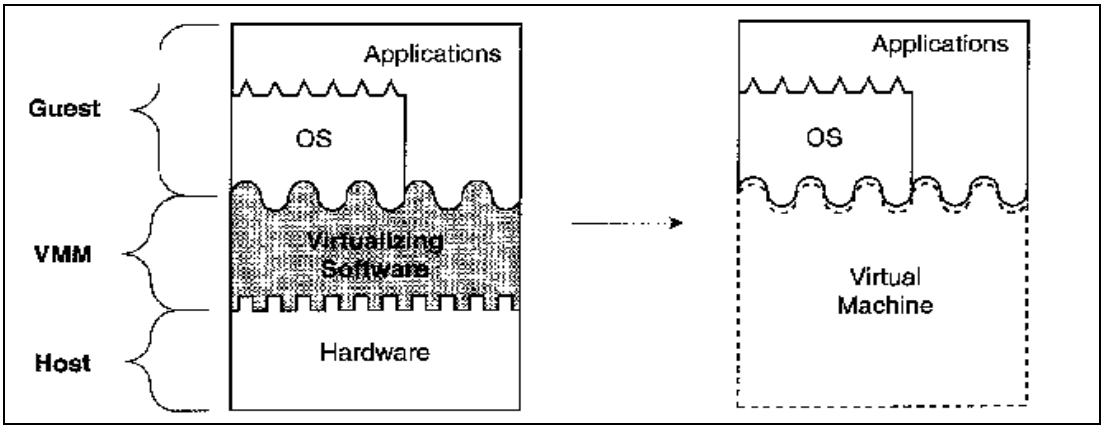
یک ماشین مجازی پردازشی. این ماشین دستورات کاربری و سطح OS نرم‌افزار مهمان را به گونه‌ای برای ماشین میزبان ترجمه می‌کند که عملا نرم‌افزار مهمان یک ماشین را می‌بیند.

نام نرم‌افزار مجازی‌ساز معمولا با توجه به اینکه پردازش و یا کل یک سیستم را مجازی‌سازی می‌کند، فرق می‌کند. به برنامه‌هایی که یک پردازش را شبیه سازی می‌کنند معمولا runtime گفته[۲۶] می‌شود. همانطور که دیدید runtimeها بین سیستم‌عامل و پردازش مجازی در حال اجرا قرار می‌گیرند.

از طرف دیگر یک مجازی‌ساز سیستمی کل یک سیستم را مجازی می‌کند. این سیستم مجازی می‌تواند یک سیستم‌عامل را بر روی خود اجرا نماید و این سیستم‌عامل مجازی به تمامی سخت‌افزار و دستگاههای ورودی/خروجی دسترسی داشته باشد. حتی این سیستم می‌تواند خروجی گرافیکی خود را نیز در بر داشته باشد. ماشین مجازی تا هنگامی که ماشین واقعی زنده باشد می‌تواند سیستم مجازی را پشتیبانی کند.

شکل بعد نمونه‌ای از این سیستم را نشان می‌دهد. همانطور که می‌بینید نرم‌افزار مجازی‌ساز بین سخت افزار و نرم‌افزار مجازی شده قرار گرفته است. در این شکل نرم‌افزار مجازی شده عملا ISA شبیه‌سازی شده توسط نرم‌افزار مجازی‌ساز را می‌بیند و نه سخت‌افزار واقعی سیستم را. البته باید گفت که در خیلی از ماشین‌های مجازی سیستمی، هم سیستم واقعی و هم سیستم مجازی از یک ISA پشتیبانی می‌کنند. در ماشین‌های مجازی سیستمی، اغلب به نرم‌افزار مجازی کننده *مانیتور ماشین مجازی* می‌گویند. این اصطلاح از همان روزهای اولیه ساخت ماشین‌های مجازی در سال ۱۹۶۰ شکل گرفت و تا به امروز همچنان پا برجاست.

ماشین مجازی ISA یک سیستم را به ISA یک سیستم دیگر (سیستم بومی) ترجمه می‌کند و با اینکار باعث می‌شود عملا یک ماشین مجازی سیستمی شکل گیرد که می‌تواند نرم‌افزارهای خاص خود را که برای گونه دیگری از سخت‌افزار طراحی شده است، اجرا نماید.



همانطور که در این قسمت کوچک از یک کتاب ۶۰۰ صفحه‌ای دیدید، ماشین‌های مجازی دیگر تنها در حد یک اسم نیستند که به راحتی بتوان از کنارشان گذشت. این رشته عملا تبدیل به یکی از تخصصهای کامپیوتری شده و دانش ویژه خود را می‌طلبد. امیدوارم در این چند صفحه کوچک با مبانی ماشین‌های مجازی آشنا شده و اگر احیانا =QEMU را نیز بر روی کامپیوتر خود نصب نموده‌اید، الان دیگر واقعا قدر آنرا بدانید و تا اندازه‌ای متوجه باشید که با چه تکنولوژی در حال کار هستید!

نویسنده: بیژن هومند

پی‌نویس:

[۱].Complexity
[۲].I/O
[۳].Abstraction
[۴].Interface
[۵].CPU Instruction Set
[۶].isomorphism
[۷].guest
[۸].host
[۹].اگر با نرم‌افزارهای مجازی‌سازی همچون QEMUو یا VMWare کار کرده باشید، دقت کرده‌اید که عملا این نرم‌افزارها یک سیستم خاص را، مثلا یک کامپیوتر با Athlon CPU و ۲۵۶ مگابایت RAMو مانیتور LG Flatron 700P شبیه‌سازی می‌کنند. برای ساخت هر سیستم مجازی باید فضای خاصی از هارددیسک را به آن اختصاص دهید و از آن به بعد با اجرای برنامه مجازی‌سازی، آن فضا برای برنامه‌های مشخص شده به عنوان یک ماشین دیگر (با سخت افزار یاد شده) لحاظ می‌گردد و این وظیفه شرکت تولید کننده ماشین مجازی است که همیشه خود را با پروتوکل‌های جدید پیش برده و بتواند واقعا سخت‌افزار یاد شده را در هر محیطی شبیه‌سازی کند. در واقع نرم‌افزارهای شبیه‌سازی همانند تونلی بین سیستم‌عاملهای مختلف عمل می‌کنند. برای درک این مطلب لازم است حتما با یکی از نرم‌افزارهای مجازی سازی بصورت عملی کار کنید و اگر تا کنون این کار را نکرده‌اید و در درک مطالب آورده شده در اینجا مشکل دارید، جای تعجب ندارد!

[۱۰].به عنوان مثال می‌توان بر روی یک سرور با ۲ پروسسور سیستم‌عامل لینوکس را نصب کرد و درون لینوکس، ویندوز را بصورت مجازی نصب نمود. در این حالت می‌توان هر دو این سیستمها را درون شبکه واقعی قرار داد و مثلا سیستم‌عامل لینوکس سرویسهای PHP و آپاچی را سرویس دهد و سیستم‌عامل ویندوز مجازی سرویسهای ASP.net و IIS. در این حالت ما عملا بر روی یک دستگاه فیزیکی، دو سیستم‌عامل را پوشش می‌دهیم و با توجه به این حقیقت که معمولا سیستم‌عامل ویندوز از پایداری کمتری برخوردار است و هر از گاهی نیاز به نصب مجدد دارد، می‌توان از فایل موجود در دستگاه میزبان که ویندوز در آن بصورت مهمان (مجازی) نصب شده است یک نسخه پشتیبان تهیه نمود و بعدا هر گاه که ویندوز دچار مشکل شود، تنها با کپی کردن فایل پشتیبان قبلی در جای فایل موجود، ویندوز را در کمتر از ۱ دقیقه (بستگی به سرعت کپی دارد!) مجدد به حالت قبل برگرداند و از نظر کاربران ما هیچ تفاوتی ایجاد نشده باشد و سیستم اصلا از خدمات‌دهی خارج نشود!

[۱۱].Emulation در لغت به معنای برتری‌جویی و انجام کاری دقیقا شبیه و یا بهتر از شخص دیگری می‌باشد. به عنوان مثال اگر من بگویم " I emulate my brother in piano" بدین معناست که من در پیانو زدن همانند برادرم و حتی بهتر از او می‌باشم. این لغت در انگلیسی روزمره دقیقا متضاد لغت Simulate که به معنای شبیه‌سازی و تظاهر در عمل است

می‌باشد. مثلا اگر من بگویم " I simulate happiness" یعنی اینکه من تظاهر به شادمانی می‌کنم. اما هر گاه در اصطلاح کامپیوتری از لغت simulate استفاده شد، منظور شبیه‌سازی کامل یک سیستم است، یعنی یک سیستم کامل با هارد دیسکها و ram و cd-drive ها و غیره را درون یک سیستم‌عامل دیگر شبیه‌سازی کنیم. اما هرگاه از اصطلاح emulate استفاده شد، منظور تنها شبیه سازی قسمتی از یک کامپیوتر واقعی است تا یک برنامه خاص را بتوان بر روی سیستم عاملهای مختلف اجرا نمود. برای اجرای یک نرم‌افزار خاص همچون photoshop درون لینوکس لازم نیست که کل ویندوز را درون لینوکس شبیه سازی کرد (که اگر اینکار را بکنیم قطعا photoshop هم در آن اجرا می‌شود)، بلکه تنها لازم است آن تکه از سخت‌افزارها و دستورالعملهایی که photoshop برای اجرا به آنها نیاز دارد را شبیه‌سازی کنیم که به این عمل emulation گوییم.

[۱۲].Interface
[۱۳].Implementation
[۱۴].یک معماری را دقیقا مثل یک جعبه سیاه در نظر بگیرید که ما می‌دانیم با زدن کلیدها (رابط) قرار است چه خروجی به ما تحویل دهد، ولی اینکه درون آن چه اتفاقی قرار است بیفتد را اصلا نمی‌دانیم. به ساخت این جعبه سیاه پیاده‌سازی می‌گوییم. وقتی که می‌خواهند یک معماری را شرح دهند، صرفا به نحوه کار با رابط (کلیدها) بسنده می‌کنند و دیگر نحوه پیاده‌سازی را توضیح نخواهند داد.
[۱۵].Instruction Set Architecture
[۱۶].User ISA
[۱۷].System ISA
[۱۸].System Call
[۱۹].کلا فلسفه مجبور کردن کاربر به صدا زدن سیستم‌عامل برای انجام عملیات مورد نظر نیز به همین دلیل بوده است.
[۲۰].Wrapper
[۲۱].ممکن است از خود بپرسید که دیگر این APIها به چه درد می‌خورند و چه فایده‌ای دارد که از آنها استفاده کنیم؟! ببینید، در این حالت صرفا API همانند یک ماکرو عمل می‌کند. فرض کنید ۱۰ تا دستور را برای انجام کاری باید هر بار در سیستم وارد کنید، خوب خیلی راحتتر بود که آنها را درون یک ظرف واحد نوشته، و تنها آن ظرف را صدا بزنید. به این ظرف ماکرو و یا procedure می‌گویند، یعنی چیزی که تنها هدفش پایین آوردن حجم تایپ شما و در عین حال یاری رساندن به حافظه شما می‌باشد. چرا کمک به حافظه؟ مشخص است، در مثال ما فرض کنید که برای انجام کاری، ۱۰ دستور را باید در ویندوز وارد می‌کردید تا کار انجام میشد و اگر همان کار را می‌خواستید در لینوکس به انجام رسانید، ۱۵ دستور کاملا متفاوت باید اجرا میشد، خوب به خاطر سپاری تمام این موارد و ترتیب اجرا هر کدام از دستورالعملها و ... چیزی نیست که یک انسان سالم واقعا علاقه‌ای به تکرار مکرر آن داشته باشد. یک API می‌تواند سیستم‌عامل را شناسایی و دستورات مقتضی را اجرا نماید. حال اگر برنامه‌نویسی از این API استفاده کند، می‌تواند مطمئن شود که برنامه‌اش یک کار واحد را هم در ویندوز و هم در لینوکس و شاید حتی سیستم‌عاملهای مختلف دیگر به انجام می‌رساند. همانگونه که می‌بینید اینگونه API ها نیز کاربرد خودشان را دارند.
[۲۲].Process
[۲۳].Process virtual-machine
[۲۴].System virtual machine
[۲۵].Native machine
[۲۶].دقت کنید که اگر runtime به صورت سر هم (همین صورتی که اینجا آورده شده) استفاده شود به معنای ماشین مجازی است که پردازش را شبیه‌سازی می‌کند. ولی run time (به صورت ۲ کلمه جدا و پشت سر هم) اصطلاح کلی‌تری است که به معنای زمان اجرای یک برنامه به کار می‌رود و ارتباطی با کار ما ندارد.