مدرسان شریف

فصل اول

«نگاه کلی به سخت‌افزار کامپیوتر»

مقدمه

سیستم عامل به عنوان یک برنامه، هم سخت‌افزار کامپیوتر را مدیریت می‌کند و هم بستری را برای اجرای برنامه‌های کاربردی و برنامه‌های سودمند فراهم می‌سازد. از این رو به عنوان واسطی میان کاربر و سخت‌افزار عمل می‌نماید. برای درک مناسب عملکرد سیستم عامل و نکات مربوط به آن، آشنایی مختصری با سخت‌افزار کامپیوتر و معماری آن ضروری است. لذا این فصل، مروری اجمالی از سخت‌افزار کامپیوتر، عملکردهای اساسی راه‌اندازی سیستم، ورودی / خروجی و ذخیره‌سازی را ارائه می‌کند.

درسنامه (۱): عناصر اصلی سیستم کامپیوتر

به طور کلی، یک کامپیوتر شامل یک یا چند نمونه از هر یک از اجزاء پردازنده، حافظه و ورودی/ خروجی است. این اجزاء به گونه‌ای به یکدیگر متصل شده‌اند تا کار اصلی کامپیوتر یعنی اجرای برنامه‌ها را فراهم کنند. از این رو چهار جزء ساختاری و اصلی زیر وجود دارند:

۱ـ پردازنده: عملیات کامپیوتر را کنترل می‌کند و اعمال پردازش داده‌ها را انجام می‌دهد. وقتی فقط یک پردازنده وجود دارد، معمولاً به آن واحد پردازشگر مرکزی ((CPU Central Processing Unit) می‌گویند.

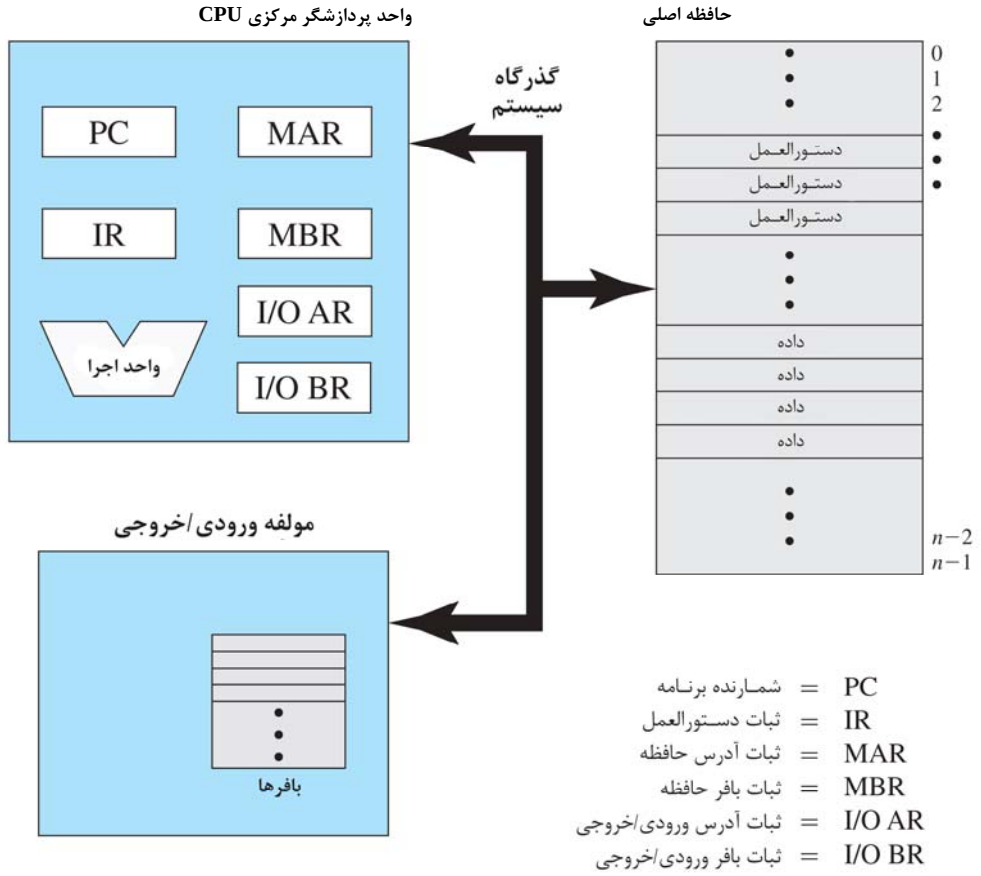
۲ـ حافظه اصلی: داده‌ها و برنامه‌ها را ذخیره می‌کند. این حافظه نوعاً ناپایدار است و به آن حافظه حقیقی یا اولیه نیز گفته می‌شود.

۳ـ مؤلفه‌های ورودی/ خروجی: داده‌ها را بین کامپیوتر و محیط خارجی آن منتقل می‌کنند. محیط خارجی شامل انواع دستگاه‌ها از جمله حافظه ثانویه، تجهیزات ارتباطی و ترمینال‌هاست.

۴ـ گذرگاه سیستم: ساختارها و راهکارهایی که ارتباط بین پردازنده، حافظه اصلی و مؤلفه‌های ورودی/ خروجی را فراهم می‌کنند.

شکل ۱ـ۱ این اجزاء را از دید کلی نشان می‌دهد. یکی از اعمال پردازنده مبادله با حافظه است. برای این منظور معمولاً از دو ثبات داخلی (برای پردازنده) استفاده می‌شود: ثبات آدرس حافظه ((Memory Address Register(MAR)، که محل خواندن یا نوشتن بعدی در حافظه را مشخص می‌کند و ثبات بافر حافظه ((Memory Buffer Register(MBR)، که در برگیرنده داده‌هایی می‌باشد که قرار است در حافظه نوشته یا از آن خوانده شود. مشابه دو ثبات فوق، از یک ثبات آدرس ورودی/ خروجی ((I/O Address register (I/OAR) که یک دستگاه ورودی/ خروجی خاص را مشخص می‌کند و از یک ثبات بافر ورودی/ خروجی ((I/O Buffer Register(I/OBR) برای تبادل داده‌ها بین پردازنده و یک مؤلفه ورودی/ خروجی استفاده می‌شود.

یک مؤلفه ورودی/ خروجی، داده‌ها را از دستگاه‌های خارجی به پردازنده و حافظه یا برعکس منتقل می‌کند. مؤلفه‌های ورودی/ خروجی دارای بافرهای داخلی هستند تا بتوانند داده‌ها را تا زمان انتقال آنها نگهداری کنند.



شکل ۱-۱. نگاه کلی به اجزاء کامپیوتر

برای این‌که یک کامپیوتر اجرای خود را شروع کند، مثلاً موقع زدن دکمه power یا راه‌اندازی مجدد و یا جهت اجرا، به یک برنامه اولیه نیاز دارد. این برنامه اولیه یا برنامه بالاآورنده (Bootstrap program) باید ساده باشد که نوعاً در حافظه فقط خواندنی (Read Only Memory (ROM)) یا حافظه فقط خواندنی برنامه‌پذیر پاک‌شدنی (Electrically erasable programmable read only memory (EEPROM)) در سخت‌افزار کامپیوتر ذخیره می‌شود. این برنامه بالا آورنده، تمام وجوه سیستم را از ثبات‌های پردازنده گرفته تا کنترلر دستگاه‌ها و محتویات حافظه ـ مقداردهی اولیه می‌کند. برنامه بالاآورنده باید چگونگی بارگذاری و شروع اجرای سیستم عامل را بداند. برای رسیدن به این هدف، برنامه بالاآورنده باید هسته سیستم عامل را یافته و داخل حافظه بارگذاری نماید.

مثال۱: کدام ثبات حاوی آدرس دستورالعمل بعدی است که باید واکنشی شود؟

۱) IR ۲) PC ۳) MBR ۴) ROM

☑ **پاسخ:** گزینه «۲» ثبات PC به عنوان شمارنده برنامه همواره حاوی آدرس دستورالعمل بعدی می‌باشد.

مثال۲: کدام گزینه به ثبات‌های قابل رؤیت برای کاربر اشاره می‌کند؟

۱) ثبات شاخص، ثبات دستورالعمل
۲) شمارنده برنامه، ثبات دستورالعمل
۳) ثبات آدرس ورودی/ خروجی، ثبات شاخص
۴) ثبات داده، ثبات شاخص

☑ **پاسخ:** گزینه «۴» ثبات‌ها به دو دسته کلی تقسیم می‌شوند؛ یکی ثبات‌های قابل رؤیت برای کاربر که به برنامه‌ساز زبان ماشین یا برنامه‌ساز زبان اسمبلی اجازه می‌دهد که با استفاده بهینه از آنها، مراجعه به حافظه اصلی را به حداقل برساند و یکی ثبات‌های کنترل وضعیت که پردازنده برای کنترل عملیات پردازنده و همچنین رویه‌های ممتاز سیستم عامل برای کنترل اجرای برنامه‌ها استفاده می‌کند. ثبات شاخص زیرمجموعه ثبات‌های آدرس محسوب می‌شود.

درسنامه (۲): پردازنده

پردازنده به عنوان مهم‌ترین مؤلفه‌ی سخت‌افزار کامپیوتر شناخته می‌شود و مشتمل بر واحد محاسبات منطقی (ALU)، واحد کنترل (CU) و ثبات‌ها است. وظیفه‌ی پردازنده، اجرای برنامه‌هایی است که درون حافظه اصلی قرار دارند. هر برنامه‌ای که بخواهد اجرا شود شامل مجموعه‌ای از دستورالعمل‌هاست و پردازنده برای اجرای برنامه‌ها لازم است که این دستورالعمل‌ها را طی فرآیندهای **واکشی** (خواندن دستورالعمل از حافظه)، **رمزگشایی** (تفسیر دستورالعمل و مشخص کردن عملی که باید انجام شود،) و **اجرا** (کنترل روند اجرای دستورالعمل) به انجام برساند. این فرایند که برای پردازش یک دستورالعمل لازم است، **چرخه دستورالعمل** نامیده می‌شود.

ثبات‌های زیر برای اجرای دستورالعمل‌ها حائز اهمیت هستند:

- شمارنده برنامه ((Program Counter (PC) : حاوی آدرس دستورالعمل است که باید واکشی شود.
- ثبات دستورالعمل ((Instruction Register (IR): حاوی آخرین دستورالعملی است که واکشی شده است.

تمام پردازنده‌ها شامل یک یا مجموعه‌ای از ثبات‌ها هستند که معمولاً تحت عنوان کلمه وضعیت برنامه ((Program Status Word (PSW) حاوی اطلاعات وضعیت هستند. این ثبات، معمولاً علاوه بر کدهای وضعیت، شامل اطلاعات دیگری مانند بیت فعال / غیرفعال کردن وقفه و بیت حالت کاربر / هسته، نیز می‌باشد.

سیستم‌های تک‌پردازنده

اکثر سیستم‌ها از یک پردازنده منفرد استفاده می‌کنند. تنوع سیستم‌های تک‌پردازنده، می‌تواند جالب توجه باشد، چرا که این سیستم‌ها از دستیار دیجیتال شخصی ((Personal Digital Assistant (PDA تا کامپیوترهای بزرگ را شامل می‌شوند. در یک سیستم تک‌پردازنده، یک پردازنده اصلی وجود دارد که قابلیت اجرای مجموعه دستورالعمل‌های همه منظوره، شامل دستورالعمل‌های پردازه‌های کاربر را دارد. تقریباً همه سیستم‌ها، پردازه‌های خاص منظوره دیگری هم دارند. آنها ممکن است به شکل پردازنده‌های خاص دستگاه، نظیر دیسک، صفحه کلید و کنترلرهای گرافیکی باشند؛ یا در کامپیوترهای بزرگ، بیشتر به شکل پردازنده‌های همه منظوره، نظیر پردازنده‌های ورودی/ خروجی که داده‌ها را به سرعت بین مؤلفه‌های سیستم انتقال می‌دهند، کار گذاشته شوند.

سیستم‌های چندپردازنده

سیستم‌های چندپردازنده (Multiprocessor)، معروف به سیستم‌های موازی یا سیستم‌های با اتصال محکم (Tightly coupled systems) از اهمیت رو به رشدی برخوردار هستند. این قبیل سیستم‌ها، دو یا چندپردازنده با ارتباط نزدیک به هم دارند که به صورت مشترک از گذرگاه سیستم و گاهی ساعت، حافظه و دستگاه‌های جانبی استفاده می‌کنند.

سیستم‌های چندپردازنده سه مزیت اصلی دارند که عبارتند از:

۱ـ بازده بالا: با زیاد شدن تعداد پردازنده‌ها، انتظار داریم که کار انجام شده‌ی بیشتری در زمان کمتر بدست آوریم. با این وجود، ضریب تسریع با N پردازنده، N نیست و کمتر از N خواهد بود. زمانی که چندین پردازنده روی یک وظیفه با یکدیگر همکاری دارند، به منظور حفظ درستی کار تمام قسمت‌ها، مقداری سربار تولید می‌شود. این سربار علاوه بر رقابت برای منابع مشترک، منافع مورد انتظار ناشی از افزایش تعداد پردازنده‌ها را نیز کاهش می‌دهد.

۲ـ صرفه اقتصادی: سیستم‌های چندپردازنده هزینه کمتری نسبت به سیستم‌های تک‌پردازنده متعدد معادل خود دارند، زیرا می‌توانند دستگاه‌های جانبی، حافظه ثانویه و مولدهای برق را به صورت مشترک استفاده کنند.

۳ـ قابلیت اطمینان بالا: اگر وظایف بین پردازنده‌های متعدد به درستی توزیع شوند، در این صورت، خرابی یک پردازنده باعث توقف کل سیستم نمی‌شود و تنها افت کارایی را در پی خواهد داشت. قابلیت اطمینان بالای یک سیستم کامپیوتری در بسیاری از برنامه‌های کاربردی امری حیاتی است.

قابلیت تداوم ارائه سرویس، متناسب با سطح بقای سخت‌افزار، تنزل مطبوع (Graceful Degradation) نامیده می‌شود.

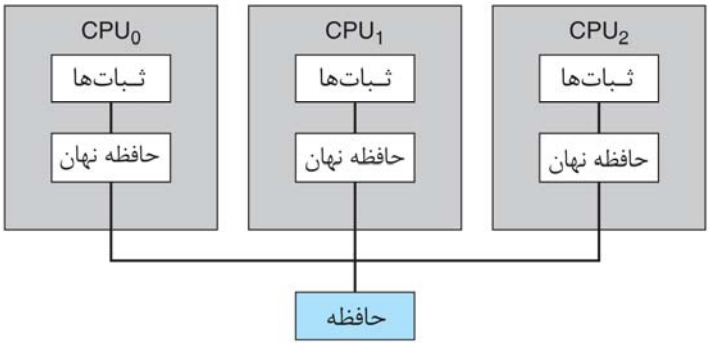
سطح بالاتری از تنزل مطبوع تحمل‌پذیر خطا (Fault Tolerant) نامیده می‌شود. سیستم‌هایی با این قابلیت می‌توانند به رغم از کار افتادن هر یک از تک مؤلفه‌ها، همچنان به عملیات خود ادامه دهند. باید توجه داشته باشید که تحمل‌پذیری خطا مستلزم راهکاری است که خطا را تشخیص، شناسایی و حتی‌الامکان اصلاح نماید.

سیستم‌های چندپردازنده‌ای که امروزه استفاده می‌شوند، دو نوع هستند:

۱ـ چندپردازشی نامتقارن ((Asymmetric Multiprocessing (AMP) : هر پردازنده وظیفه‌ای خاص دارد. یک پردازنده راهبر (Master)، سیستم را کنترل می‌کند. سایر پردازنده‌ها چشم به راهبر دارند و یا وظایف از پیش تعریف شده‌ای را انجام می‌دهند. این طرح یک رابطه راهبر ــ پیرو (Master - slave) تعریف می‌کند. پردازنده‌ی بالادست (راهبر)، کارها را زمان‌بندی کرده و در اختیار پردازنده‌های پایین‌دست (پیرو) قرار می‌دهد.

۲ـ چندپردازشی متقارن ((Symmetric Multiprocessing (SMP): هر پردازنده تمامی وظایف سیستم عامل را انجام می‌دهد. SMP به این معنی است که تمامی پردازنده‌ها نظیر هم هستند؛ هیچ‌گونه رابطه راهبر ـ پیرو میان پردازنده‌ها وجود ندارد. شکل ۱ـ۲ یک نمونه از معماری SMP را نشان می‌دهد. توجه داشته باشید که هر یک از پردازنده‌ها، مجموعه‌ی ثبات‌های خاص خود را درکنار یک حافظه نهان خصوصی (یا محلی) دارند؛ با وجود این، همه پردازنده‌ها در حافظه فیزیکی مشترک هستند.

سیستم‌های چندپردازنده متقارن (SMP)، امکان اشتراک پویای پردازه‌ها و منابع (مانند حافظه) را میان پردازنده‌های مختلف فراهم ساخته و اختلاف کارایی بین پردازنده‌ها را کاهش می‌دهند.



شکل ۱ـ۲. معماری چند پردازشی متقارن

سولاریس نمونه‌ای از یک سیستم چندپردازشی متقارن (SMP) است که به صورت یک نسخه تجاری از یونیکس توسط شرکت سان میکروسیستمز (Sun microsystems) طراحی شده است.

سیستم عامل‌های پیشرفته، اعم از ویندوز (xp، 7 و 8) ، Mac OS X و لینوکس، از SMP پشتیبانی می‌کنند.

تفاوت چندپردازشی متقارن و نامتقارن می‌تواند ناشی از سخت‌افزار یا نرم‌افزار باشد. ممکن است سخت‌افزار خاصی سبب تفاوت پردازنده‌ها شود، یا این که نرم‌افزاری به گونه‌ای نوشته شود که یک پردازنده را راهبر و بقیه را پیرو قلمداد کند. برای نمونه، نسخه ۴ سیستم عامل sun OS شرکت سان میکروسیستمز، چند پردازشی نامتقارن را فراهم می‌کند، در حالی که نسخه ۵ (سولاریس) در همان سخت‌افزار به صورت متقارن است.

پردازنده‌های چند هسته‌ای

نگرش اخیر طراحی پردازنده این است که بتوان در یک تراشه‌ی منفرد، **هسته‌های** (cores) محاسباتی متعدد کار گذاشت. در اصل این‌ها، تراشه‌های چندپردازنده هستند. یک کامپیوتر چند هسته‌ای که ما آن را به عنوان یک تراشه چندپردازنده می‌شناسیم، دو یا چند پردازنده (هسته) را در یک قطعه سیلیکونی ترکیب می‌کند. به عنوان نمونه، هر هسته، متشکل از تمام مؤلفه‌های یک پردازنده مستقل، مانند ثبات‌ها، واحد محاسباتی منطقی، سخت‌افزار خط لوله (Pipe Line)، واحد کنترل به علاوه دستورالعمل حافظه نهان سطح اول ((level 1 (L1) و حافظه نهان داده می‌باشد. تراشه‌های چندهسته‌ای جدید شامل حافظه نهان سطح دوم (L2) و سطح سوم (L3) نیز هستند.

فصل دوم

« نگاه کلی به سیستم عامل »

مقدمه

سیستم عامل برنامه‌ای است که به عنوان واسطی میان کاربر کامپیوتر و سخت‌افزار کامپیوتر عمل می‌کند. همچنین می‌توان گفت سیستم عامل، نرم‌افزاری است که سخت‌افزار کامپیوتر را مدیریت می‌کند.

هدف سیستم عامل، فراهم کردن محیطی است که کاربر بتواند برنامه‌ها را به روشی **آسان** و **کارآمد** اجرا کند.

مبنا و تفاوت سیستم عامل‌های مختلف نیز براساس همین هدف پایه‌گذاری شده است. هدف سیستم عامل کامپیوترهای بزرگ (Mainframe)، بهینه‌سازی بهره‌وری سخت افزار است؛ سیستم عامل کامپیوترهای شخصی (PC)، با هدف کاربردهای تجاری، بازی‌های پیچیده و پشتیبانی از این‌گونه برنامه‌ها طراحی شده‌اند. هدف سیستم عامل کامپیوترهای دستی و موبایل‌ها، ایجاد محیطی است که کاربر بتواند به سادگی برای اجرای برنامه‌ها با کامپیوتر ارتباط برقرار کند. از این رو برخی از سیستم عامل‌ها با هدف سادگی و سهولت، برخی با هدف کارآمدی و دسته‌ای دیگر برای برآوردن توأم این دو هدف طراحی می‌شوند.

اهداف مهم سیستم عامل

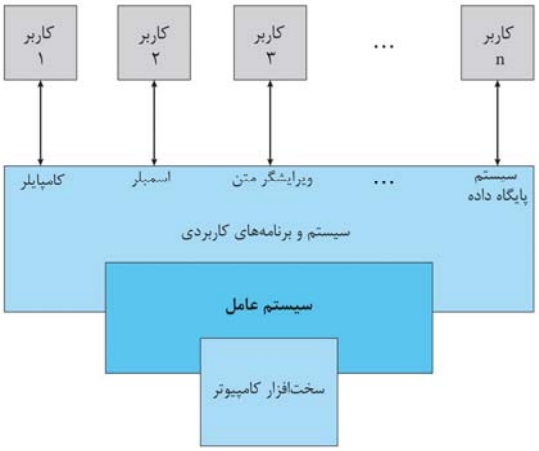
- **سادگی و سهولت: استفاده ساده‌تر و راحت‌تر از کامپیوتر**
- **کارآمدی: استفاده از منابع سیستم با روشی کارآمد**
- **قابلیت تکامل: قابلیت توسعه، آزمون و معرفی کارکردهای جدید سیستمی**

سیستم عامل عرصه بسیار وسیعی را دربر می‌گیرد. این فصل، دیدگاهی کلی از موضوع‌های سیستم عامل را ارائه می‌کند و در فضایی ساده، به برخی اصول بنیادی مطرح در سیستم عامل می‌پردازد. ابتدا نقش سیستم عامل و عملکرد آن مطرح می‌شود. سپس سیر تکاملی آنها از نظر تاریخی، تشریح می‌گردد. در ادامه، ساختار سیستم عامل و عملیات بررسی شده و در نهایت درباره محیط‌های محاسباتی صحبت می‌شود.

درسنامه (۱): نقش سیستم عامل

همان‌گونه که در شکل ۲ـ۱ مشاهده می‌کنید، هر سیستم کامپیوتری می‌تواند به چهار بخش تقسیم شود:

۱ـ سخت افزار ۲ـ سیستم عامل ۳ـ برنامه‌های کاربردی ۴ـ کاربران



شکل ۲ـ۱ . چشم‌انداز انتزاعی بخش‌های یک سیستم کامپیوتری

سخت افزار (واحد پردازشگر مرکزی، حافظه و دستگاه‌های ورودی/خروجی)، منابع محاسباتی پایه را فراهم می‌کند. برنامه‌های کاربردی مانند واژه‌پردازها، صفحه‌های گسترده و مرورگرهای اینترنت، مسیرهایی را فراهم می‌کنند که این منابع بتوانند برای حل مسائل محاسباتی کاربران به کار گرفته شوند.

سیستم عامل چگونگی استفاده از سخت‌افزار را بین برنامه‌های کاربردی متعدد کاربران مختلف، کنترل و هماهنگ می‌کند.

سیستم عامل ابزاری را برای استفاده‌ی آسان از سخت افزار، نرم‌افزار و داده‌ها در عملیات سیستم کامپیوتری فراهم می‌کند.

سیستم عامل مشابه یک حکومت عمل می‌کند و به تنهایی نمی‌تواند کارکرد کارآمدی داشته باشد. سیستم عامل بستری را فراهم می‌کند که سایر برنامه‌ها بتوانند کار خود را به نحو مطلوب به انجام برسانند. در ادامه، به منظور درک بهتر نقش سیستم عامل، آن را از دو دیدگاه (دید کاربر و سیستم) بررسی می‌کنیم.

دید کاربر

اغلب کاربران کامپیوتر، در مقابل کامپیوتری می‌نشینند که دارای یک صفحه نمایش، صفحه کلید، ماوس و یک سیستم واحد است. چنین سیستمی برای یک کاربر طراحی می‌شود تا بتواند از منابع آن به صورت یک جانبه استفاده کند. در اینجا، سیستم عامل بیشتر با هدف **سهولت استفاده**، با نیم نگاهی به **کارآیی** و بدون توجه به **بهره‌وری منبع** طراحی می‌شود. البته، کارآیی برای کاربر اهمیت دارد؛ اما این قبیل سیستم‌ها، به جای در نظر داشتن نیازمندی‌های کاربران متعدد، برای جلب نظر یک تک کاربر بهینه‌سازی می‌شوند.

در شرایطی دیگر، کاربر پشت ترمینالی می‌نشیند که به یک کامپیوتر بزرگ متصل است. در همان حال، کاربران دیگری هم هستند که به همان کامپیوتر از طریق ترمینال دیگری دسترسی دارند. این کاربران منابع را به اشتراک می‌گیرند و ممکن است تبادل اطلاعات هم داشته باشند. در اینجا، سیستم عامل با هدف **افزایش بهره‌وری منبع** طراحی می‌شود. در واقع این اطمینان را می‌دهد که کل زمان موجود پردازنده، حافظه و ورودی/خروجی به صورت کارآمد استفاده می‌شوند و هیچ کاربری سهمی بیش از سهم خود به دست نمی‌آورد.

در شرایطی دیگر، می‌توان کاربرانی را در نظر گرفت که با نشستن در پشت سر ایستگاه‌های کاری به شبکه‌های سایر ایستگاه‌های کاری و کارگزارها (servers) متصل می‌شوند. این کاربران، یک سری منابع اختصاصی در اختیار دارند؛ اما آن‌ها منابعی مانند شبکه و کارگزارها (کارگزارهای فایل، محاسبات و چاپ) را نیز به اشتراک می‌گیرند. از این رو، سیستم عامل آن‌ها به گونه‌ای طراحی می‌شود تا تعادلی میان استفاده‌ی شخصی و بهره‌وری منبع به وجود بیاورد.

برخی دیگر از کامپیوترها جایگاه اندکی برای دید کاربر دارند یا فاقد آن هستند. به عنوان مثال، کامپیوترهای تعبیه شده در لوازم خانگی و خودروها ممکن است یک صفحه‌کلید عددی داشته باشند و برای نمایش حالت، چراغ‌هایی را روشن یا خاموش کنند؛ اما این کامپیوترها و سیستم عامل آن‌ها چنان طراحی می‌شوند که بدون دخالت کاربر اجرا داشته باشند.

دید سیستم

از نقطه نظر کامپیوتر، سیستم عامل برنامه‌ای بسیار نزدیک به سخت افزار است. در این مورد، می‌توانیم سیستم عامل را به عنوان یک **تخصیص‌گر منبع** یا یک **مدیر منابع** در نظر بگیریم.

یک سیستم کامپیوتر، شامل منابع سخت افزاری و نرم‌افزاری زیادی است که در حل یک مسأله مفید هستند؛ از جمله می‌توان به موارد زیر اشاره نمود:

- زمان واحد پردازشگر مرکزی (CPU)
- فضای حافظه
- فضای ذخیره‌سازی فایل
- دستگاه‌های ورودی/ خروجی
- و ...

سیستم عامل به عنوان مدیر این منابع، باید در رویارویی با درخواست‌های متعدد و حتی ضد و نقیض منابع، تصمیم بگیرد که چگونه آن‌ها را به برنامه‌ها و کاربران مورد نظر تخصیص دهد تا از تداخل آنها با یکدیگر جلوگیری کند و بتواند سیستم کامپیوتری را به طور **کارآمد**، **صحیح** و **عادلانه** به کار اندازد.

نکته۱: تخصیص منابع در جایی که کاربران متعددی می‌خواهند به یک کامپیوتر بزرگ یا کامپیوتر کوچک یکسان دسترسی داشته باشند، اهمیت ویژه‌ای پیدا می‌کند.

هر سیستم عامل = یک برنامه‌ی کنترل

یک برنامه‌ی کنترل، اجرای برنامه‌های کاربر و دستگاه‌های مختلف ورودی/ خروجی را مدیریت می‌کند تا از بروز خطاها و استفاده نامناسب از کامپیوتر جلوگیری نماید.

تعریف سیستم عامل

در حالت کلی، تعریف جامع و کاملی از سیستم عامل وجود ندارد. هدف اصلی سیستم‌های کامپیوتری، اجرای برنامه‌های کاربر و حل آسان‌تر مسائل است. سخت‌افزار کامپیوتر در راستای این هدف ساخته شد. چون سخت‌افزار محض به تنهایی قابل استفاده نیست، بنابراین برنامه‌های کاربردی به وجود آمدند. این برنامه‌ها، به بعضی از عملیات مشترک نیاز دارند، مانند عملیاتی که دستگاه‌های ورودی / خروجی را کنترل می‌کنند. بنابراین کارکردهای متداول کنترل و تخصیص منابع، در یک قطعه نرم‌افزاری، به نام سیستم عامل جمع می‌شوند.

به علاوه، تعریف عمومی پذیرفته شده‌ای از بخش‌های سیستم عامل نداریم؛ یک دیدگاه ساده آن است که سیستم عامل شامل هر چیزی است که فروشنده تحت عنوان «سیستم عامل» برای شما بسته‌بندی کرده و می‌فروشد. با وجود این، امکاناتی که لحاظ می‌شوند، تنوع زیادی در میان سیستم‌ها دارند. برخی سیستم‌ها کمتر از یک مگابایت فضا می‌گیرند و حتی فاقد یک ویرایشگر تمام‌صفحه هستند. در حالی که سیستم‌های دیگری هم هستند که چندین گیگابایت فضا گرفته و به صورت کامل مبتنی بر سیستم‌های پنجره‌ای و گرافیکی هستند. یک تعریف رایج به صورت زیر است:

سیستم عامل، برنامه‌ای است که در تمام زمان‌ها (تمام زمان روشن بودن کامپیوتر) در حال اجراست که معمولاً **هسته (Kernel)** نامیده می‌شود. در کنار هسته، دو نوع برنامه‌ی دیگر هم هستند که عبارتند از:

۱ـ برنامه‌های سیستمی: مربوط به سیستم عامل هستند، اما بخشی از هسته محسوب نمی‌شوند.

۲ـ برنامه‌های کاربردی: شامل تمام برنامه‌هایی می‌شود که به عملیات سیستم، ارتباطی ندارند.

امروزه با نگاهی به سیستم عامل‌های دستگاه‌های موبایل (کامپیوترهای دستی) مشاهده خواهیم کرد که تعداد ویژگی‌های ساخت سیستم‌های عامل در حال افزایش است. سیستم عامل‌های موبایل اغلب نه تنها شامل یک هسته بلکه شامل **میان‌افزار(middleware)** نیز هستند. میان‌افزار یک مجموعه از چارچوب‌های نرم‌افزاری است که سرویس‌های اضافی را جهت ایجاد کنندگان برنامه‌های کاربردی فراهم می‌کند. برای مثال، هر دو سیستم عامل‌های مهم موبایل (Android , iOS) نشان می‌دهند که شامل یک هسته همراه با میان‌افزاری است که پایگاه داده‌ها، چندرسانه‌ای‌ها و گرافیک‌ها را پشتیبانی می‌کند.

مثال۱: کدام گزینه در مورد وظیفه سیستم عامل صحیح‌تر است؟

۱) کنترل منابع کامپیوتر و ایجاد بستری برای اجرای برنامه‌های کاربردی

۲) یک میان افزار، میان سخت افزار و نرم افزار

۳) یک برنامه کاربردی بزرگ است که استفاده از سخت افزار را کامل می‌کند

۴) اجرای برنامه‌های کاربردی مختلف را هماهنگ و اجرا می‌کند

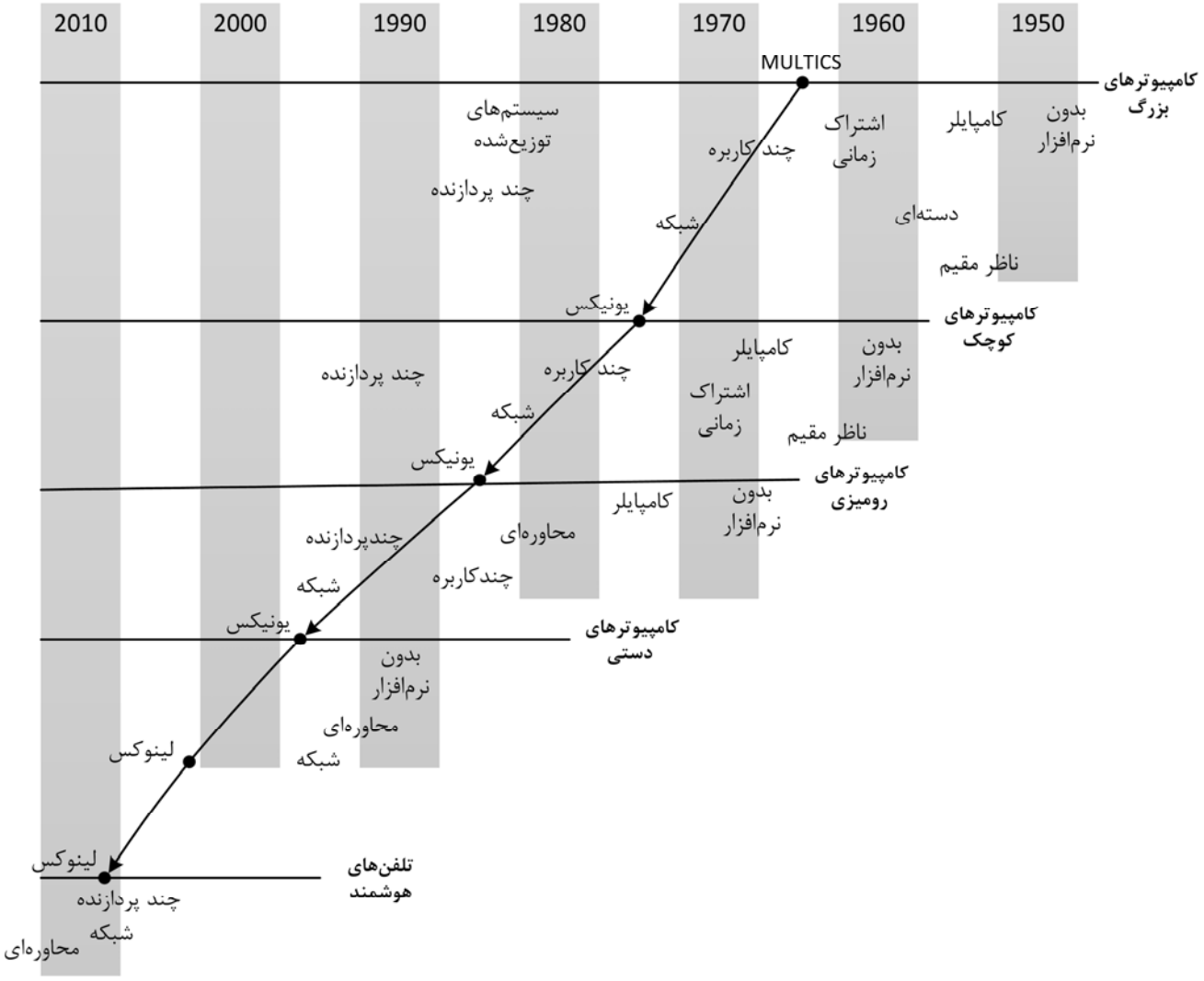
☑ **پاسخ:** گزینه «۱» مهمترین وظیفه سیستم عامل کنترل منابع کامپیوتر و ایجاد یک بستر مناسب برای اجرای برنامه‌های کاربردی مختلف است.

درسنامه (۲): سیر تکامل سیستم عامل

سیستم عامل نیز مانند هر فناوری دیگری، در طول سالیان متمادی، ایجاد و تکامل یافته است. در این بخش می‌خواهیم نگاهی کوتاه به سیر تکامل سیستم عامل بیندازیم. از آنجایی که سیستم عامل ارتباطی تنگاتنگ با معماری کامپیوتر دارد، سیر تکامل سیستم عامل، به نوعی داستان تکامل نسل‌های مختلف کامپیوتر نیز هست.

یک آزمایش ساده بر روی سیستم عامل کامپیوترهای بزرگ (Mainframe) و میکروکامپیوترها نشان می‌دهد که بسیاری از ویژگی‌هایی که قبلاً تنها بر روی کامپیوترهای بزرگ قابل استفاده بود، برای میکروکامپیوترها نیز تعبیه شده است.

مفاهیم پایه در سیستم عامل‌ها یکسان هستند؛ بدین ترتیب که برای طبقه‌بندی کامپیوترهای مختلف اعم از: کامپیوترهای بزرگ، کامپیوترهای کوچک، ریزکامپیوترها و کامپیوترهای دستی اختصاصی شده‌اند. برای شناخت سیستم عامل‌های جدید، لازم است اطلاعاتی را در مورد شکل مهاجرت و سابقه ویژگی‌های سیستم عامل‌های مختلف بدانیم. این سیر تکامل و مهاجرت سیستم عامل‌ها و ویژگی‌های آنها در شکل ۲ـ۲ مشخص شده است.



شکل ۲ـ۲. سیر تکامل و مهاجرت مفاهیم سیستم عامل و ویژگی‌های آنها

فصل سوم

« ساختارهای سیستم عامل»

مقدمه

از نظر داخلی، سیستم عامل‌ها تنوع زیادی در ساخت خود دارند، چرا که با راهکارهای متفاوت فراوانی سازماندهی می‌شوند. طراحی یک سیستم عامل جدید کار خطیری است. لازم است که اهداف سیستم قبل از شروع مرحله‌ی طراحی به خوبی تعریف شوند. این اهداف، اساس گزینش میان الگوریتم‌ها و راهبردهای مختلف را تشکیل می‌دهند.

از چند دیدگاه می‌توان یک سیستم عامل را بررسی کرد. یک دیدگاه سرویس‌های متمرکزی است که سیستم آنها را تدارک می‌بیند؛ دیدگاه دیگر، از طریق رابطی است که در اختیار کاربران و برنامه‌نویسان قرار می‌گیرد؛ دیدگاه سوم روی مؤلفه‌ها و اتصالات بین آنها تأکید دارد. در این فصل، این سه وجه سیستم عامل‌ها را با نشان دادن دیدگاه‌های کاربران، برنامه‌نویسان و طراحان سیستم عامل بررسی کرده و سرانجام به انواع مختلف ساختارهای سیستم عامل و مزایا و معایب هر کدام خواهیم پرداخت.

درسنامه (۱): سرویس‌های سیستم عامل

سیستم عامل بستری را برای اجرای برنامه‌ها فراهم می‌کند و سرویس‌های ویژه‌ای برای برنامه‌ها و کاربران برنامه‌ها مهیا می‌کند. البته ارائه سرویس‌های ویژه از سیستم عاملی به سیستم عامل دیگر متفاوت است. اما می‌توانیم سرویس‌های مشترکی را در بین آنها داشته باشیم. این سرویس‌های سیستم عامل، به برنامه‌نویس کمک می‌کنند تا کار برنامه‌نویسی آسان‌تر انجام شود. شکل ۳ـ۱ چشم‌انداز کلی از سرویس‌های مختلف سیستم عامل و چگونگی ارتباط میان آنها را نشان می‌دهد.

شکل ۳ـ۱ . مروری بر سرویس‌های سیستم عامل

یک مجموعه از سرویس‌های سیستم عامل، کارکردهایی دارند که به کاربر کمک می‌کنند:

- **رابط کاربر :** همه سیستم عامل‌ها یک **رابط کاربر** (**User Interface(UI)**) دارند. این رابط می‌تواند شکل‌های مختلفی داشته باشد. یکی از آنها **رابط خط فرمان** (**Command – Line Lnterface**) مکانیزمی برای برقراری ارتباط با سیستم عامل با استفاده از فرمان‌های متنی است. دیگری **رابط دسته‌ای** (**Batch Interface**) است که در آن فرمان‌ها و راهنماهای کنترل آن‌ها، داخل فایل‌ها قرار داده شده و آن فایل‌ها اجرا می‌شوند. رایج‌ترین

رابط مورد استفاده، **رابط کاربر گرافیکی** (**Graphical User Interface (GUI)**) است. در این حالت، رابط یک سیستم عامل (ویندوز) با یک دستگاه اشاره‌گر (مثل ماوس) برای هدایت ورودی/خروجی، انتخاب منوها و گزینه‌ها و یک صفحه کلید برای وارد کردن متن است. برخی سیستم‌ها، دو یا همگی این سه رابط را فراهم می‌کنند.

- **اجرای برنامه‌ها:** سیستم باید بتواند یک برنامه را در حافظه بارگذاری کرده و آن برنامه را اجرا کند. برنامه باید بتواند اجرای خود را به صورت عادی یا غیرعادی (با نشان دادن خطا) به اتمام برساند.
- **عملیات ورودی/خروجی:** برنامه‌ای که در حال اجرا است، ممکن است به ورودی/خروجی نیاز داشته باشد که در این راستا، امکان دارد به یک فایل یا یک دستگاه ورودی/خروجی نیاز پیدا کند. برای دستگاه‌های خاص ممکن است عملیات ویژه‌ای مطلوب باشد (مانند نوشتن در یک CD یا DVD یا پاک کردن صفحه نمایش). معمولاً به دلایل حفاظتی، کاربران نمی‌توانند دستگاه‌های ورودی/خروجی را مستقیماً کنترل کنند. بنابراین، سیستم عامل باید ابزاری برای انجام ورودی/خروجی فراهم نماید.
- **پردازش سیستم فایل:** سیستم فایل از اهمیت خاصی برخوردار است. واضح است که برنامه‌ها به خواندن و نوشتن فایل‌ها و دایرکتوری‌ها نیاز دارند. علاوه بر این، باید بتوانند آنها را ایجاد و حذف کنند، یک فایل مشخص را جستجو کرده و اطلاعات فایل را لیست نمایند. برخی برنامه‌ها مجوزهای مدیریتی دارند تا اجازه یا منع دسترسی به فایل‌ها یا دایرکتوری‌ها را براساس مالکیت فایل اعمال کنند. بسیاری از سیستم عامل‌ها، سیستم فایل‌های مختلفی را فراهم می‌کنند که گاهی حق انتخاب را به کاربر داده و به منظور فراهم نمودن امکانات یا مشخصه‌های کارایی خاص فراهم می‌شوند.
- **ارتباطات:** شرایط زیادی پیش می‌آید که یک پردازه نیاز دارد تا اطلاعاتی را با پردازه‌های دیگر مبادله کند. چنین ارتباطی می‌تواند بین پردازه‌های در حال اجرای یک ماشین یا میان پردازه‌های سیستم‌های کامپیوتری متفاوتی که از طریق یک شبکه کامپیوتری به همدیگر متصل هستند، صورت گیرد. ارتباطات می‌توانند از طریق **حافظه مشترک** یا به وسیله **ارسال پیام** پیاده‌سازی شوند که در آن بسته‌های اطلاعاتی توسط سیستم عامل میان پردازه‌ها مبادله می‌شوند.
- **تشخیص خطا**: سیستم عامل نیاز دارد که به صورت مداوم خطاهای ممکن را تشخیص دهد. خطاها ممکن است در پردازنده و سخت‌افزار حافظه (از جمله خطای حافظه یا قطع برق)، در دستگاه‌های ورودی/خروجی (از جمله خطای توازن در نوار، قطع اتصال در شبکه یا عدم وجود کاغذ در چاپگر) و در برنامه کاربر (از جمله سرریز محاسباتی، تلاش برای دسترسی غیرقانونی به مکانی از حافظه یا استفاده بیش از حد از زمان پردازنده) اتفاق بیافتد. سیستم عامل باید برای هر نوع خطا، عملیات مناسب آن را برای تضمین صحت و سازگاری محاسبات در پیش گیرد. البته، در رابطه با چگونگی واکنش سیستم عامل و تصحیح خطاهای آن تنوع وجود دارد. امکانات عیب‌یابی می‌توانند توانایی‌های کاربر و برنامه‌نویس را به صورت قابل توجهی در استفاده کارآمد از سیستم تحقق ببخشند. مجموعه‌ی دیگری از کارکردهای سیستم عامل هستند که نه برای کمک به کاربر، بلکه برای حصول اطمینان خود سیستم عامل از کارآمدی عملیات است. سیستم‌هایی که کاربران متعددی دارند، می‌توانند کارآمدی را با اشتراک منابع کامپیوتر بین کاربران تحقق بخشند.
- **تخصیص منبع:** زمانی که چندین کاربر یا چندین کار اجرای همزمان دارند، منابع باید به هر کدام از آن‌ها تخصیص یابد. انواع مختلفی از منابع توسط سیستم عامل مدیریت می‌شوند. برخی (مانند چرخه‌های پردازنده، حافظه اصلی و حافظه فایل) ممکن است کد تخصیص ویژه‌ای داشته باشند. درحالی که بقیه (از جمله دستگاه‌های ورودی/خروجی) ممکن است درخواست کلی‌تر و انتشار کُد داشته باشند. برای نمونه، سیستم عامل‌ها برای تعیین استفاده از پردازنده به بهترین شکل ممکن، روال‌های زمان‌بندی پردازنده را دارند که سرعت پردازه، کارهایی که باید اجرا شوند، تعداد ثبات‌های در دسترس و سایر عوامل را در نظر می‌گیرند. ممکن است روال‌هایی هم برای تخصیص چاپگرها، مودم‌ها، درایورهای حافظه USB و سایر دستگاه‌های جانبی وجود داشته باشند.
- **حسابداری:** به دنبال این هستیم که بدانیم کدام کاربران، چه مقدار و از چه نوع منابع کامپیوتری استفاده می‌کنند. این گونه ثبت و نگهداری اطلاعات می‌تواند برای حسابرسی (تا امکان صدور صورت حساب برای کاربران داشته باشیم) یا برای جمع‌آوری آمار استفاده شود. آمارهای بهره‌برداری، می‌تواند ابزار ارزشمندی برای محققانی باشد که می‌خواهند سیستم را دوباره برای بهبود سرویس‌های محاسباتی پیکربندی نمایند.
- **حفاظت و امنیت:** مالکین اطلاعات ذخیره شده در یک سیستم کامپیوتری چند کاربره یا متصل به شبکه ممکن است بخواهند روی استفاده از آن اطلاعات کنترل داشته باشند. زمانی که چندین پردازه مجزا به‌طور همروند اجرا می‌شوند، نباید اجرای یک پردازه روی اجرای پردازه‌های دیگر یا خود سیستم عامل تداخل ایجاد کنند. حفاظت متضمن آن است که تمام دسترسی‌ها به منابع سیستم کنترل شوند. امنیت سیستم در مقابل بیگانگان نیز اهمیت دارد. چنین امنیتی معمولاً با الزام تشخیص هویت هر کاربر با ابزاری مانند کلمه عبور در ورود به سیستم شروع می‌شود تا به منابع سیستم دست پیدا کند. علاوه بر این، دستگاه‌های ورودی/خروجی خارجی شامل مودم‌ها و کارت‌های شبکه را از دسترسی‌های غیرمجاز مصون می‌دارد و تمام این گونه اتصالات را برای تشخیص نفوذها ثبت می‌کند. اگر قرار است سیستمی حفاظت شده و امن باشد، باید جوانب احتیاط در سرتاسر آن رعایت شود.

رابط کاربر سیستم عامل

همان گونه که قبلاً متذکر شدیم، روش‌های متعددی برای ارتباط کاربران با سیستم عامل وجود دارند. در این قسمت، دو رهیافت را بررسی می‌کنیم. یکی رابط خط فرمان یا **مفسر فرمان** (**Command Interpreter**) را فراهم می‌کند که به کاربران اجازه می‌دهد تا مستقیماً فرمان‌هایی را وارد کنند تا توسط سیستم عامل انجام شوند. دیگری به کاربران اجازه می‌دهد تا از طریق یک **رابط کاربر گرافیکی** (GUI) با سیستم عامل ارتباط برقرار کنند.

مفسر فرمان

برخی سیستم عامل‌ها مفسر فرمان را در هسته قرار می‌دهند. سیستم‌های دیگری مانند ویندوز XP و یونیکس، مفسر فرمان را به صورت یک برنامه‌ی خاصی می‌بینند که در زمان آغاز یک کار یا زمانی که کاربر برای اولین بار به سیستم وارد می‌شود (در سیستم‌های محاوره‌ای)، به اجرا در می‌آید.

در سیستم‌هایی که مفسرهای فرمان متعددی برای انتخاب دارند، مفسرها به پوسته‌ها (Shells) معروف هستند.

برای مثال، در سیستم‌های یونیکس و لینوکس، یک کاربر می‌تواند از میان چندین پوسته متفاوت، از جمله *Korn shell , Bourne – Again shell , C shell , Bourne shell* و سایرین، انتخاب داشته باشد. اکثر پوسته‌ها کارکرد مشابهی فراهم می‌کنند و این که کاربر کدام پوسته را برای استفاده انتخاب می‌کند، عموماً به سلیقه فردی بستگی دارد.

کار اصلی مفسر فرمان، گرفتن و اجرای فرمان بعدی کاربر است.

رابط کاربر گرافیکی

راهبرد دوم ارتباط با سیستم عامل از طریق یک رابط کاربر گرافیکی کاربرپسند یا GUI است. در این راهبرد، کاربران به جای این که مجبور باشند فرمان‌ها را مستقیماً از طریق یک رابط خط فرمان وارد کنند، از یک سیستم پنجره‌ای و منویی مبتنی بر ماوس استفاده می‌کنند که با یک اشاره‌گر متحرک در دسکتاپ مشخص می‌شود. کاربر، ماوس را حرکت می‌دهد تا اشاره‌گر آن روی تصاویر یا آیکون‌های صفحه نمایشگر (دسکتاپ) قرار بگیرد که معرف برنامه‌ها، فایل‌ها، دایرکتوری‌ها و کارکردهای سیستم می‌باشند. بسته به محل اشاره‌گر ماوس، کلیک یک دکمه ماوس می‌تواند برنامه‌ای را احضار کند، یک فایل یا دایرکتوری (معروف به پوشه) را انتخاب نماید یا منویی را باز کند که حاوی فرمان‌هایی است.

رابط‌های کاربر گرافیکی برای اولین بار در اوایل دهه ۱۹۷۰ میلادی در انجام بخشی از تحقیقات Xerox PARC ظهور کردند. اولین GUI در سال ۱۹۷۳ ، به کامپیوتر Xerox Alto اضافه شد. رابط‌های گرافیکی با پیدایش کامپیوترهای اپل مکینتاش در دهه ۱۹۸۰، گسترش بیشتری پیدا کردند. رابط کاربر سیستم عامل مکینتاش (Mac OS) نیز تغییرات گوناگونی پشت سر گذاشته است. مهمترین آن رابط *Aqua* بود که در Mac OS X به کار گرفته شد. اولین نسخه ویندوز مایکروسافت، براساس افزودن یک رابط GUI به سیستم عامل MS-DOS بود. بعدها نسخه‌های متعددی از ویندوز عرضه شدند که برای داشتن ظاهری بهتر و مرتب‌تر تغییر چهره دادند و پیشرفت‌های متعددی در کارکرد خود از جمله در مرورگر ویندوز داشتند.

انتخاب بین استفاده از رابط‌های خط فرمان یا GUI، به علایق شخصی بستگی دارد.

به عنوان قاعده کلی، اغلب کاربران یونیکس ترجیح می‌دهند از رابط‌های خط فرمان استفاده کنند؛ زیرا رابط‌های پوسته قدرتمندی را فراهم می‌سازند. برعکس، اغلب کاربران ویندوز دوست دارند از محیط GUI ویندوز استفاده کنند.

مثال۱: مزیت اصلی رابط خط فرمان کدام گزینه است؟

۱) سهولت در وارسی
۲) کارآیی در امنیت
۳) انتقال سریع اطلاعات
۴) استفاده کمتر از منابع سخت‌افزاری

☑ **پاسخ:** گزینه «۴» رابط فرمان به دلیل داشتن سرعت بالا، استفاده کمتری از منابع سخت‌افزاری دارد.

درسنامه (۲): فراخوان‌های سیستم

همانطور که در فصل قبل (در بخش عملیات مد دوگانه) متذکر شدیم، فرایندهای کاربر در مد کاربر اجرا می‌شوند، از این رو به بخش محدودی از حافظه دسترسی دارند. ولی یک فرایند برای استفاده از منابع سیستم و اجرای دستورالعمل‌ها، نیاز به ارتباط با سیستم عامل و فراخوانی سرویس‌های آن دارد. این درحالی است که سرویس‌های سیستم عامل در بخشی از حافظه قرار دارند که متعلق و محفوظ برای سیستم عامل است و دسترسی به آنها در مد کاربر امکان‌پذیر نیست.

فراخوان‌های سیستمی، ابزاری در اختیار برنامه و پردازه‌های کاربر قرار می‌دهند تا از سیستم عامل بخواهند کارهایی را که برای سیستم عامل محفوظ هستند، از طرف برنامه‌ی کاربر انجام دهد.

فراخوان سیستم، رابطی برای سرویس‌های مهیا شده توسط سیستم عامل را فراهم می‌کند.

لحظه‌ای که برنامه کاربر، سرویس را از سیستم عامل از طریق فراخوان سیستمی درخواست می‌کند، باید برای انجام این درخواست، از مد کاربر به مد هسته انتقال یابد. فرایند فراهم شدن فراخوان‌های سیستمی و انجام آن، شامل جزئیات زیادی است و سیستم‌ها به دفعات زیاد، فراخوان سیستمی را در هر ثانیه اجرا می‌کنند. این در حالی است که اکثر برنامه‌نویسان هرگز جزئیات سطح پایین استفاده از فراخوان‌های سیستمی را نمی‌بینند. برنامه‌نویسان برنامه‌های کاربردی نوعاً برنامه‌ها را مطابق یک **رابط برنامه‌نویسی کاربردی** (**Application programming Interface** (**API**)) طراحی می‌کنند.

API، مجموعه‌ای از توابع را که در دسترس یک برنامه‌نویس کاربردی است و همچنین پارامترهای ارسالی به هریک از توابع و مقادیر برگشتی مورد انتظار برنامه‌نویس را توصیف می‌کند. سه نمونه از رایج‌ترین API ها که در اختیار برنامه‌نویسان کاربردی قرار دارند، عبارتند از:

۱ـ Win 32 API: برای سیستم‌های ویندوز

۲ـ POSIX API: برای سیستم‌های مبتنی بر POSIX، که تمام نسخه‌های یونیکس، لینوکس و Mac OSX را شامل می‌شود.

۳ـ Java API: برای طراحی برنامه‌های اجرایی روی ماشین مجازی جاوا

در پشت صحنه، توابعی که یک API را می‌سازند، فراخوان‌های سیستمی واقعی را از طرف برنامه نویس کاربردی احضار می‌کنند. برای مثال، تابع Create process در Win32 (برای ایجاد پردازه جدید استفاده می‌شود)، در حقیقت فراخوان سیستمی ( )NTCreateprocess را در هسته ویندوز فراخوانی می‌کند.

چرا برنامه‌نویس کاربردی به جای این که فراخوان‌های سیستمی واقعی را احضار کند، برنامه‌نویسی با API را ترجیح می‌دهد؟

دلایل متعددی برای انجام این کار وجود دارد. یک مزیت برنامه‌نویسی با API، به قابلیت حمل برنامه مربوط می‌شود؛ یک برنامه‌نویس کاربردی که برنامه‌ای را با API طراحی می‌کند، می‌تواند این انتظار را داشته باشد که برنامه‌اش در هر سیستمی که همان API را پشتیبانی کند، کامپایل و اجرا خواهد شد. به علاوه، فراخوان‌های سیستمی واقعی، غالباً جزئیات بیشتری داشته و کار برنامه نویس با آنها نسبت به API فراهم شده، دشوار است. اغلب یک همبستگی قوی میان یک تابع موجود در API و فراخوان سیستمی مرتبط با آن در هسته وجود دارد.

سیستم پشتیبانی زمان اجرا برای اکثر زمان‌های برنامه‌نویسی، یک **رابط فراخوان سیستم** فراهم می‌کند که به عنوان رابطی به فراخوان‌های سیستمی فراهم شده، توسط سیستم عامل عمل می‌کند. رابط فراخوان سیستم، فراخوانی‌های تابع API را گرفته و فراخوان سیستمی لازم را از سیستم عامل احضار می‌کند. معمولاً به هر یک از فراخوان‌های سیستمی، شماره‌ای منتسب می‌شود و رابط فراخوان سیستم، جدولی را که با این شماره‌ها ایندکس (شاخص) شده است نگه می‌دارد. سپس، رابط فراخوان سیستم، فراخوان سیستمی موردنظر را از هسته سیستم عامل احضار و وضعیت فراخوان سیستمی و هرگونه مقادیر برگشتی را بر می‌گرداند.

فراخواننده لازم نیست که چیزی در رابطه با چگونگی پیاده‌سازی فراخوان سیستم یا عملیات آن در طی اجرا بداند. بلکه فقط باید مطیعAPI بوده و بداند که سیستم عامل به دنبال اجرای آن فراخوان سیستمی چه کار خواهد کرد. بنابراین، بیشتر جزئیات رابط سیستم عامل، توسط API از برنامه‌نویس پنهان مانده و توسط کتابخانه پشتیبان زمان اجرا اداره می‌شوند.

مدرسان شریف

فصل چهارم

«پردازه‌ها، نخ‌ها و زمان‌بندی پردازنده»

مقدمه

سیستم‌های کامپیوتری اولیه تنها به یک برنامه اجازه می‌دادند تا در یک زمان اجرا شود. این برنامه، ضمن در اختیار داشتن کامل سیستم به تمامی منابع سیستم دسترسی داشت. در مقایسه، سیستم‌های کامپیوتری امروزی، امکان بارگذاری برنامه‌های متعدد را داخل حافظه فراهم کرده‌اند و آنها را به‌طور همروند اجرا می‌کنند. این تکامل نیازمند کنترل سخت‌گیرانه و همچنین همکاری برنامه‌های مختلف می‌باشد. این نیازها به شکل یک پردازه نمود پیدا می‌کنند که در واقع یک برنامه درحال اجراست. هر پردازه، واحد کار سیستم اشتراک زمانی مدرن است.

یکی از وظایف بنیادی هر سیستم عامل امروزی، مدیریت پردازه‌هاست. سیستم عامل باید منابع را به پردازه‌ها تخصیص دهد و آنها را قادر به اشتراک و تبادل اطلاعات نماید؛ منابع هر پردازه را از پردازه‌های دیگر حفاظت کند و همگام‌سازی پردازه‌ها را فراهم سازد. در بسیاری از سیستم عامل‌های مدرن، مشکلات مدیریت فرایند با معرفی مفهوم نخ ترکیب شده است. در سیستم چند نخی، پردازه، خصوصیات تملک منابع را نگهداری می‌کند؛ در حالی که اجرای چندگانه و هم‌زمان، خاصیت نخ‌هایی است که در داخل یک پردازه اجرا می‌گردند. در این فصل، می‌آموزیم که پردازه‌ها و نخ‌ها چه هستند و چگونه کار می‌کنند.

درسنامه (۱): مفهوم پردازه (Process)

تمامی فعالیت‌ها در هر سیستم کامپیوتری پیرامون اجرای دستورالعمل‌های برنامه‌های کامپیوتری متمرکز می‌شوند. واژه «پردازه» به اجرای مجموعه‌ای از دستورالعمل‌های ماشین اشاره دارد. دستورالعمل‌ها به خودی خود ماهیتی ایستا دارند. هر پردازه، مجموعه دستورالعمل‌هایی است که حیات بخشیده می‌شوند. پردازه، ماهیتی پویا است که اعمالی را که به صورت کد توصیف می‌شوند، انجام می‌دهد.

یک پردازه چیزی فراتر از کد برنامه است که گاهی تحت عنوان بخش متن شناخته می‌شود. همچنین فعالیت جاری را شامل می‌شود که با مقدار شمارنده برنامه و محتویات ثبات‌های پردازنده مشخص می‌شود. به‌طور کلی یک پردازه، دارای **پشته‌ای** است که داده‌های موقتی (مانند پارامترهای توابع، آدرس‌های برگشتی و متغیرهای محلی) را دربر می‌گیرد و **بخش داده‌هایی** که شامل متغیرهای سراسری است. یک پردازه می‌تواند یک heap هم داشته باشد، حافظه‌ای که به صورت پویا در طی زمان اجرای پردازه تخصیص داده می‌شود.

لازم به تأکید است که یک برنامه به خودی خود یک پردازه نیست. هر برنامه یک موجودیت منفعل و دارای ماهیتی ایستا است. درحالی که پردازه، موجودیتی فعال با یک شمارنده برنامه می‌باشد که دستورالعمل اجرایی بعدی را مشخص می‌کند و مجموعه‌ای از منابع به آن مربوط می‌شود.

یک برنامه، زمانی که فایل اجرایی آن به درون حافظه بارگذاری می‌شود، یک پردازه محسوب می‌گردد.

ایجاد پردازه

پردازه‌ها در اکثر سیستم عامل‌ها می‌توانند اجرای همروند داشته باشند و امکان ایجاد و حذف پویای آنها وجود دارد. بنابراین، این سیستم‌ها باید راهکارهایی برای ایجاد و خاتمه پردازه‌ها فراهم کنند. معمولاً چهار رخداد اساسی، موجب ایجاد یک پردازه می‌شود:

۱ـ در محیط دسته‌ای، یک پردازه جدید در پاسخ به تحویل یک کار، ایجاد می‌شود.

۲ـ در محیط محاوره‌ای، زمانی که یک کاربر جدید از طریق پایانه اقدام به برقراری ارتباط با کامپیوتر می‌کند، یک پردازه ایجاد می‌گردد.

۳ـ سیستم عامل می‌تواند پردازه‌ی جدیدی را برای ارائه خدمتی از طرف برنامه‌ی کاربر ایجاد کند، بدون آنکه کاربر ملزم به انتظار باشد. به عنوان مثال، اگر کاربر چاپ پرونده‌ای را درخواست کند، سیستم عامل می‌تواند پردازه‌ی جدیدی ایجاد نماید تا چاپ آن را مدیریت کند.

۴ـ یک پردازه در طول دوره‌ی اجرای خود می‌تواند از طریق فراخوان سیستمی Create-process چندین پردازه‌ی جدید ایجاد نماید. به این عمل **زایش پردازه** گویند. پردازه ایجاد کننده، **پردازه‌ی والد** و پردازه‌های جدید، **فرزندان پردازه** نامیده می‌شوند. هریک از این پردازه‌های جدید هم ممکن است به نوبه خود پردازه‌های دیگری ایجاد نموده و درختی از پردازه‌ها را شکل دهند.

وقتی یک پردازه، پردازه‌ی جدیدی را ایجاد کرد، دو امکان در مورد اجرا وجود دارد:

۱ـ ممکن است پردازه والد به‌طور همزمان با پردازه فرزند اجرا شود.

۲ـ پردازه والد منتظر می‌ماند تا تمام یا برخی از پردازه‌های فرزندش خاتمه یابند.

در مورد فضای آدرس پردازه نیز دو امکان وجود دارد:

۱ـ پردازه فرزند، یک کپی از پردازه والد است (داده‌ها و برنامه‌های یکسان با والد دارد)، مانند سیستم عامل یونیکس

۲ـ پردازه فرزند، برنامه‌ای را در فضای آدرس خود بارگذاری کرده است.

✓ **در سیستم عامل یونیکس، هر پردازه جدید با فراخوان سیستمی ( )fork ایجاد می‌شود.**

✓ **در سیستم عامل ویندوز، پردازه‌ها با استفاده از تابع ( )create process ایجاد می‌شوند که مشابه ( )fork است.**

اغلب سیستم عامل‌ها از جمله خانواده ویندوز و یونیکس، پردازه‌ها را براساس شناسه پردازه (pid)، شناسایی می‌کنند که معمولاً یک عدد صحیح یکتا است.

در سیستم عامل یونیکس، پردازه‌ی جدید با فراخوان سیستمی ( )fork ایجاد می‌شود. این پردازه‌ی جدید شامل یک کپی از فضای آدرس پردازه‌ی اصلی است. این راهکار به پردازه والد اجازه می‌دهد تا به راحتی با پردازه‌ی فرزندش ارتباط برقرار کند. اجرای هر دو پردازه‌ی والد و فرزند، از دستور بعد از ( )fork ادامه می‌یابد، ولی با یک تفاوت: کد برگشتی ( )fork برای پردازه جدید (فرزند) برابر با صفر است. در حالی که شناسه پردازه فرزند که غیرصفر است به پردازه والد برگردانده می‌شود.

معمولاً، پس از فراخوان سیستم ( )fork ، فراخوان سیستم ( )exec توسط پردازه فرزند اجرا می‌شود تا فضای حافظه‌ی پردازه را با برنامه‌ی جدیدی (دلخواه) جایگزین کند. در واقع پردازه‌ی فرزند، با اجرای فراخوان سیستم ( )exec ، فرمان bin / ls / اجرا می‌کند و فضای آدرس خود را همپوشانی می‌کند.

در این روش، این دو پردازه قادر به برقراری ارتباط هستند و سپس به راه جداگانه خود ادامه می‌دهند. سپس پردازه والد می‌تواند فرزندان بیشتری تولید نماید و چنانچه در حین اجرای فرزند، والد کاری برای انجام دادن نداشته باشد، می‌تواند فراخوان سیستم ( )wait را اجرا کند تا خودش را وارد صف آماده نماید تا فرزند خاتمه یابد.

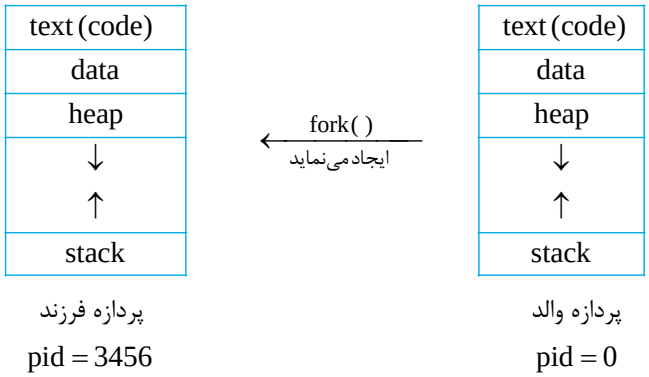
تذکر۱: اجرای ( )exec در سیستم عامل، باعث می‌شود تصویر پردازه جدید، جایگزین تصویر پردازه فعلی شود (منظور از تصویر پردازه: پشته، بخش کُد، هیپ و ... است). ( )exec در بسیاری از زبان‌های برنامه‌سازی وجود دارد.

مثال۱: اگر یک پردازه با دستور ( ) fork پردازه جدیدی را ایجاد نماید، کدام‌یک از داده‌های زیر بین پدر و فرزند به اشتراک گذاشته نمی‌شود؟

(مهندسی فناوری اطلاعات IT ـ سراسری ۹۳)

۱) Process id ۲) Stack ۳) Heap ۴) Code

پاسخ: گزینه «۱» پردازه فرزند یکی از پردازه والد است. اما پردازه فرزند، یک شناسه (pid) یکتا و متفاوت از والد دارد.



تفاوت بین پردازه والد و فرزند که به اشتراک گذاشته نمی‌شود:

process ID (get pid())

parent ID (get ppid())

تذکر: در سیستم عامل یونیکس، فراخوانی سیستمی getpid، شماره شناسه پردازه (pid) و فراخوانی سیستمی getppid، شماره شناسه پردازه والد را به‌دست می‌آورد.

مثال۲: پروسس (فرایند یا پردازه) فرزند، کدام‌یک از موارد زیر را از پروسس پدر به ارث نمی‌برد؟ (مهندسی فناوری اطلاعات IT ـ سراسری ۸۷)

۱) تایمر ۲) دایرکتوری جاری ۳) نام کاربر اجرا کننده ۴) توصیف‌گر فایل باز

☑ **پاسخ:** گزینه «۱» پردازه فرزند نمی‌تواند تایمر (زمان‌سنج) را به ارث ببرد، چرا که پردازه مانند سایر پردازه‌ها در صف آماده اجرا قرار می‌گیرد تا پردازنده به آن تخصیص یابد. همچنین دقت کنید که پردازه فرزند هیچ‌گاه منتظر رویدادهای مورد انتظار پردازه والد (پدر) نمی‌باشد.

خاتمه پردازه

پردازه زمانی که با اجرای آخرین دستور خود به انتهای کار خود می‌رسد، خاتمه یافته و با استفاده از فراخوان سیستمی ( )exit از سیستم عامل می‌خواهد که آن را حذف نماید. این حالت معمولاً با نام خاتمه عادی (طبیعی) تعبیر می‌شود.

دلیل دیگر ختم پردازه‌ها این است که با خطایی مواجه شود. از آن جمله می‌توان به موارد زیر اشاره نمود:

- **سقف زمانی**: پردازه‌ای که سقف اجرا را رعایت نکرده است.
- **نبود حافظه**: پردازه‌ای که به حافظه‌ای بیش از آنچه که سیستم می‌تواند فراهم کند، نیاز داشته باشد.
- **تجاوز از حدود**: پردازه‌ای که می‌خواهد به محل‌هایی از حافظه مراجعه کند، که مجاز نیست.
- **خطای حفاظت**: پردازه‌ای که برای دسترسی به منبعی تلاش می‌کند، که مجاز به استفاده نیست.
- **خطای محاسباتی**: پردازه‌ای که یک محاسبه غیرمجاز، مانند تقسیم بر صفر را انجام می‌دهد.
- **گذشت زمان**: پردازه‌ای که بیش از حداکثر زمانی که برایش تعیین شده است، برای بروز رویداد مشخصی منتظر مانده است.
- **خطای ورودی/خروجی**: خطایی که در ورودی/خروجی اتفاق افتاده است، ممکن است موجب خاتمه پردازه شود. مانند پیدا نکردن یک پرونده یا عمل غیرمعتبر همچون خواندن از روی چاپگر.
- **دستورالعمل نامعتبر**: پردازه‌ای که سعی می‌کند دستورالعملی را اجرا کند که وجود ندارد.
- **دستورالعمل ممتاز**: پردازه‌ای که برای اجرای دستورالعملی تلاش می‌کند که مخصوص سیستم عامل است.
- **استفاده نامناسب از داده**: داده‌ای که از نوع نامناسب یا بدون مقدار اولیه است ممکن است موجب خاتمه پردازه شود.
- **دخالت سیستم عامل**: سیستم عامل به دلایلی (مانند وجود بن بست) ممکن است پردازه‌ای را خاتمه دهد.

خاتمه یک پردازه در شرایط دیگری هم می‌تواند اتفاق بیافتد. پردازه می‌تواند با فراخوانی سیستمی مناسب، پردازه دیگری را خاتمه بدهد. معمولاً، یک چنین فراخوان سیستمی تنها توسط والد پردازه احضار می‌شود که می‌خواهد خاتمه بپذیرد. اگر جز این باشد، کاربران می‌توانند کارهای همدیگر را به راحتی از بین ببرند. یک والد به دلایل مختلفی ممکن است اجرای یکی از فرزندان خود را خاتمه بدهد که عبارتند از:

- فرزند در استفاده از برخی منابعی که به آن تخصیص یافته بود، زیاده‌روی کرده است.
- کار محوله به فرزند دیگر لازم نیست.
- پردازه والد در حال خروج است و سیستم عامل بعد از خاتمه پردازه والد اجازه نمی‌دهد که پردازه فرزندش به اجرای خود ادامه بدهد.

برخی از سیستم‌ها، از جمله VMS، اجازه نمی‌دهند که بعد از خاتمه پردازه‌ی والد، پردازه فرزند همچنان وجود داشته باشد. در این قبیل سیستم‌ها، اگر پردازه‌ای خاتمه پیدا کند (عادی و یا غیرعادی)، در این صورت باید بعد از اتمام والد، همگی فرزندان آن نیز خاتمه یابند. به این پدیده، **خاتمه آبشاری** می‌گویند که در اصل توسط سیستم عامل انجام می‌شود.

حالت‌های پردازه

در حالی که یک پردازه اجرا می‌شود، تغییر حالت می‌دهد. حالت یک پردازه تا حدودی از روی فعالیت جاری آن پردازه تعریف می‌شود. هر پردازه می‌تواند در یکی از حالت‌های زیر قرار بگیرد:

حالت جدید (New): پردازه ایجاد می‌شود.

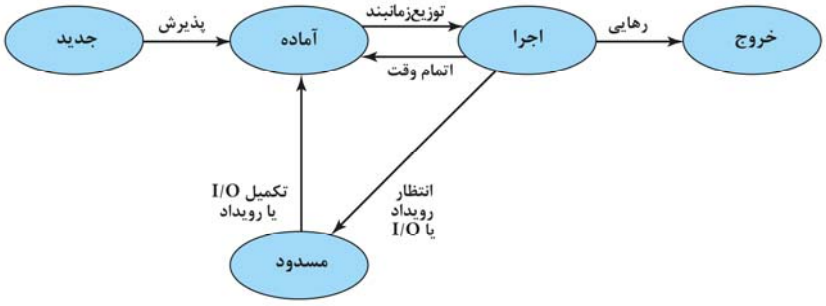
حالت اجرا (Running) : دستورالعمل‌ها اجرا می‌شوند. (فرایندی که هم اکنون درحال اجراست)

حالت آماده (Ready) : پردازه منتظر است تا به یک پردازنده گمارده شود.

حالت انتظار (Waiting) یا مسدود (Blocked): پردازه‌ای که نمی‌تواند اجرا شود و منتظر رویدادی است تا اتفاق بیافتد. (مانند تکمیل ورودی/ خروجی) در این حالت، پردازه نه پردازنده‌ای را در اختیار دارد و نه در صف پردازه‌های آماده برای دریافت پردازنده قرار می‌گیرد.

حالت خاتمه یا خروج (Terminated) : پردازه‌ای که اجرایش پایان داده شده است (اقدام به خروج طبیعی و یا دستور توقف آن صادر شده است).

این اسامی دلخواه هستند و در بین سیستم عامل‌ها فرق می‌کنند. با وجود این، حالت‌هایی که نشان می‌دهند در تمامی سیستم عامل‌ها وجود دارند. سیستم عامل‌های خاصی هم حالت‌های پردازه را با مرزبندی بهتر و ریزتری ترسیم می‌کنند که در ادامه آنها را بررسی می‌کنیم. نمودار حالت‌های پردازه، در شکل ۴ـ۱ نشان داده شده است.



شکل ۴ـ۱ . نمودار حالت پردازه

تغییر حالت‌های ممکن که در شکل فوق نیز نشان داده شده است، عبارتند از:

- **تهی ← جدید:** پردازه جدیدی با توجه به دلایل ایجاد پردازه، برای اجرای یک برنامه ایجاد می‌شود.
- **جدید ← آماده:** زمانی که سیستم عامل برای گرفتن یک پردازه دیگر آمادگی داشته باشد، پردازه‌ای را از حالت جدید به حالت آماده می‌برد.
- **آماده ← اجرا:** زمانی که وقت انتخاب یک پردازه‌ی دیگر برای اجرا رسید، سیستم عامل یکی از پردازه‌هایی که در حالت آماده است را انتخاب می‌کند. این موضوع که کدام پردازه را انتخاب کند در ادامه بررسی می‌کنیم.
- **اجرا ← خروج:** اگر پردازه‌ی جاری اعلام کند که کامل شده است یا اگر قطع شود، سیستم عامل این پردازه را پایان می‌دهد.
- **اجرا ← آماده:** متداول‌ترین دلیل این تغییر حالت، اتمام زمان مجاز برای اجرای پردازه‌ی جاری است. اما دلایل دیگری نیز برای این تغییر حالت وجود دارد. رها کردن داوطلبانه کنترل پردازنده توسط یک پردازه، از آن جمله است. دلیل دیگر برای این تغییر حالت، قبضه کردن پردازه در حال اجرا است که سطح اولویت پایین‌تری دارد. فرض کنید که پردازه‌ی A درحالت اجرا و پردازه‌ی B که سطح اولویت بالاتری دارد، مسدود است. اگر سیستم عامل بفهمد رخدادی که B منتظرش بوده، اتفاق افتاده است، با انتقال B به حالت آماده، می‌تواند پردازه A را وقفه داده و پردازه B را به حالت اجرا ببرد. در این صورت می‌گوییم سیستم عامل پردازه A را قبضه کرده است.
- **اجرا ← مسدود:** اگر پردازه‌ای چیزی را درخواست کند که به خاطرش باید منتظر بماند، درحالت مسدود گذاشته می‌شود. برای مثال ممکن است پردازه‌ای خدمتی را از سیستم عامل درخواست کند که سیستم عامل آمادگی انجام فوری آن را نداشته باشد، از جمله انجام عملیات ورودی/ خروجی و یا ارتباط با دیگر پردازه‌ها.
- **مسدود ← آماده :** زمانی که پردازه از حالت مسدود به حالت آماده می‌رود، رخدادی که منتظرش بوده، اتفاق افتاده است.
- **آماده ← خروج:** در برخی سیستم‌ها، یک پردازه‌ی والد می‌تواند هر لحظه که بخواهد، پردازه‌های فرزند خود را پایان دهد. همچنین، اگر پردازه والد پایان یابد، ممکن است تمام پردازه‌های فرزند آن نیز پایان داده شوند. این تغییر حالت به دلیل وضوح بیشتر در شکل نشان داده نشده است.

مدرسان شریف

فصل پنجم

«همروندی»

مقدمه

همه موضوعات محوری در طراحی سیستم عامل از جمله چند برنامه‌ای، چند پردازشی و پردازش توزیعی، به مدیریت پردازه‌ها و نخ‌ها مرتبط می‌شود. هم برای این موضوعات محوری و هم برای طراحی سیستم عامل، **همروندی** (**Concurrency**) از جمله اساسی‌ترین موضوعات است. همروندی، گروهی از موضوعات طراحی را دربرمی‌گیرد، از آن جمله می‌توان به **ارتباط بین پردازه‌ها** (**Inter Process Comunication (IPC)**)، اشتراک منابع و رقابت برای آن‌ها و **همگامی** (**Synchronization**) پردازه‌ها اشاره نمود.

این فصل با مقدمه‌ای بر مفاهیم ارتباط بین پردازه‌ها، همروندی و اجرای همروند پردازه‌ها آغاز می‌شود. با خواندن این بخش‌ها متوجه می‌شویم که نیاز اصلی برای حمایت از پردازه‌های همروند، امکان اعمال **انحصار متقابل** است. یعنی، امکان این‌که وقتی به پردازه‌ای قدرت انجام عمل داده شد، بتوان تمام پردازه‌های دیگر را از انجام آن عمل بازداشت. در ادامه این فصل به بررسی رویکردهای مختلف دستیابی به انحصار متقابل خواهیم پرداخت. ابتدا رویکردها و راه‌حل‌های نرم‌افزاری مورد بحث قرار می‌گیرند که نیازمند روشی به نام انتظار مشغول هستند. پس از آن، برخی از راهکارهای سخت‌افزاری در حمایت از انحصار متقابل مطرح می‌گردد. سپس به دنبال راه‌حل‌هایی می‌رویم که متضمن انتظار مشغول نبوده و به علاوه بتوانند به‌وسیله سیستم عامل حمایت شده یا توسط مترجمین زبان‌ها اعمال گردند. سه رویکرد، سمافور، ناظر و ارسال پیام نیز مورد بررسی قرار گرفته‌اند.

از چند مسأله کلاسیک همروندی، ازجمله مسأله تولیدکننده و مصرف‌کننده و خوانندگان و نویسندگان، برای تشریح مفاهیم و مقایسه رویکردهای مطرح شده در این فصل استفاده شده است.

برخی از واژه‌های کلیدی مربوط به همروندی	
عملیات اتمی Atomic Operaion	یک تابع یا عملیات غیرقابل تقسیم است که به عنوان زنجیره‌ای از یک یا چند دستورالعمل پیاده‌سازی می‌شود. به‌عبارت دیگر، هیچ پردازه دیگری قادر نخواهد بود یک حالت میانی از این عملیات را ببیند یا آن را وسط کار متوقف کند. این زنجیره دستورالعمل‌ها، یک ضمانت اجرایی دارند به صورتی که به عنوان یک گروه اجرا شوند تا کامل خاتمه یابند یا این‌که هیچ‌یک اصلاً اجرا نشوند. خاصیت اتمی، تفکیک‌پذیری پردازه‌های همروند را تضمین می‌کند.
ناحیه بحرانی Critical Section	بخشی از برنامه است که با عوامل مشترک سروکار دارد. به‌طور دقیق‌تر می‌توان گفت: بخشی از کد داخل یک پردازه است که نیازمند دسترسی به منابع مشترک رقابتی می‌باشد و تا مادامی که پردازه دیگر در آن بخش متناظر کد خود قرار دارد، نباید اجرا شود.
شرط رقابتی Race Condition	موقعیتی است که دو یا چند پردازه یا نخ، یک قلم داده مشترک را می‌خوانند و می‌نویسند و نتیجه نهایی بستگی به ترتیب اجراهای آن‌ها دارد.
انحصار متقابل Mutual Exclusion	یک نیازمندی است به‌طوری که اگر پردازه‌ای برای دسترسی به منابع مشترک در بخش بحرانی خود قرار دارد، پردازه‌های دیگر نمی‌توانند در حال اجرای بخش بحرانی خود باشند و لذا به هریک از منابع مشترک نیز دسترسی نخواهند داشت. به عبارت دیگر، در هر لحظه تنها یک پردازه مجاز است در ناحیه بحرانی خود باشد و به منابع مشترک دسترسی داشته باشد.
بن‌بست deadlock	موقعیتی است که در آن دو یا چند پردازه، از ادامه اجرا عاجز می‌شوند، چرا که هر یک منتظر دیگری است تا کارش را انجام دهد. به عبارت دیگر دچار سیکل انتظار همیشگی می‌شوند.
گرسنگی Starvation	موقعیتی است که یک پردازه قابل اجرا، برای یک مدت نامحدود توسط زمان‌بند منتظر گذاشته می‌شود، اگرچه قادر است اجرا شود، هرگز انتخاب نمی‌شود.

درسنامه (۱): ارتباط بین پردازه‌ها

پردازه‌های در حال اجرای سیستم عامل می‌توانند پردازه‌های **مستقل** و یا پردازه‌های **همکار** باشند.

هر پردازه که با هیچ‌یک از پردازه‌های دیگر، داده مشترک نداشته باشد، مستقل است.
هر پردازه که با سایر پردازه‌های دیگر اشتراک داده دارد، همکار است.

به دلایل متعدد باید محیطی فراهم شود که پردازه‌ها بتوانند با یکدیگر همکاری داشته باشند. این دلایل عبارت‌اند از:

- **اشتراک اطلاعات:** از آن‌جایی که چندین کاربر ممکن است بخواهند به یک سری اطلاعات یکسان (برای نمونه یک فایل مشترک) دسترسی داشته باشند، باید محیطی فراهم کنیم تا دسترسی همروند به این قبیل اطلاعات امکان‌پذیر باشد.
- **تسریع محاسبات:** چنانچه تمایل داریم که کار خاصی سریع‌تر اجرا شود، باید آن را به کارهای کوچک‌تری تقسیم کنیم که هریک از آن‌ها بتوانند موازی با سایر کارها اجرا شوند. توجه داشته باشید که این تسریع فقط در کامپیوتری قابل حصول است که عناصر پردازش متعددی (نظیر پردازنده‌ها یا کانال‌های ورودی/خروجی) داشته باشد.
- **پیمانه‌ای:** شاید بخواهیم سیستم را با تقسیم کارکردهای سیستم به پردازه‌ها یا نخ‌های مجزا به شیوه پیمانه‌ای ایجاد کنیم.
- **سهولت:** حتی یک کاربر منفرد هم امکان دارد که بخواهد همزمان روی چندین وظیفه کار کند. برای نمونه، یک کاربر ممکن است کار ویرایش، چاپ و کامپایل را به صورت موازی انجام بدهد.

پردازه‌های همکار به یک راهکار **ارتباط بین پردازه‌ای** (ICP) نیاز دارند تا بتوانند داده‌ها و اطلاعات را با یکدیگر مبادله کنند. دو مدل اساسی جهت ارتباط بین پردازه‌ها وجود دارد که عبارت‌اند از:

۱ـ حافظه مشترک (Shared Memory): مدل حافظه مشترک، با ناحیه‌ای از حافظه که توسط پردازه‌های همکار به اشتراک گذاشته می‌شود، برپا می‌شود. پس از آن پردازه‌ها می‌توانند با خواندن و نوشتن داده‌ها در ناحیه مشترک به تبادل اطلاعات بپردازند.

۲ـ تبادل پیام (Message Passing): در مدل تبادل پیام، ارتباط از طریق پیام‌هایی که میان پردازه‌های همکار مبادله می‌شود، صورت می‌گیرد.

هر دو مدل در سیستم عامل‌ها رواج دارند و بسیاری از سیستم‌ها هر دو مدل را پیاده‌سازی می‌کنند. تبادل پیام در مبادله‌ی داده‌هایی که حجم کمتری دارند، کارآمدتر است، چون نیازی نیست که از **برخوردها** جلوگیری کنیم. پیاده‌سازی تبادل پیام در ارتباطات بین کامپیوترها از حافظه مشترک ساده‌تر است. حافظه مشترک حداکثر سرعت و سهولت ارتباط را امکان‌پذیر می‌سازد.

حافظه مشترک سریع‌تر از تبادل پیام است، چون سیستم‌های تبادل پیام معمولاً با استفاده از فراخوان‌های سیستمی پیاده‌سازی می‌شوند و از این‌رو به دلیل مداخله هسته، زمان بیشتری لازم دارند. در مقابل، در سیستم‌های حافظه مشترک، فراخوان‌های سیستمی تنها برای برپایی نواحی حافظه مشترک به کار می‌روند. همین‌که حافظه مشترک برپا شد، تمام دسترسی‌ها به عنوان دسترسی‌های معمول حافظه تلقی شده و هیچ‌گونه نیازی به کمک هسته نخواهیم داشت.

مثال۱: چند گزاره زیر در مورد ارتباط بین پردازه‌ها صحیح است؟

الف) در ارتباط مستقیم هر پردازه که می‌خواهد ارتباط برقرار کند باید صریحاً نام گیرنده ارتباط را ذکر کند.

ب) روش تبادل پیام در محیط توزیع شده کارآمد است.

ج) در حافظه مشترک، پردازه‌ها با توافق یکدیگر می‌توانند به حافظه دیگری دستیابی داشته باشند.

۱) صفر ۲) یک ۳) دو ۴) سه

☑ **پاسخ:** گزینه «۴» هر سه گزاره صحیح هستند.

اگر پردازه‌های P و Q بخواهند با یکدیگر ارتباط برقرار کنند، باید پیام‌هایی را به یکدیگر ارسال و دریافت کنند. برای این کار باید یک پیوند ارتباطی بین آنها برقرار باشد. اگر این ارتباط مستقیم باشد، هر پردازه‌ای که می‌خواهد ارتباط برقرار کند باید صریحاً نام گیرنده ارتباط را ذکر نماید (گزاره الف).

مبادله پیام، راهکاری را فراهم می‌کند که در محیط‌های توزیع شده مفید است که در آن پردازه‌هایی که با هم ارتباط برقرار می‌کنند، ممکن است در کامپیوترهای مختلفی واقع باشد که از طریق شبکه به هم متصل هستند (گزاره ب).

در حالت عادی، سیستم عامل سعی می‌کند مانع از دستیابی یک پردازه به حافظه پردازه دیگر شود. اما در حافظه مشترک لازم است دو یا چند پردازه، برای حذف این محدودیت با هم توافق کنند. پس می‌توانند با خواندن و نوشتن داده‌ها در حافظه مشترک، اطلاعات را مبادله کنند (گزاره ج).

درسنامه (۲): اصول همگامی پردازه‌ها

گروهی از افراد را درنظر بگیرید که در ساخت یک ساختمان مشارکت دارند و با هم کار می‌کنند. در حالت مطلوب، این افراد باید با هم همگام باشند (نه این‌که لزوماً همزمان کار کنند). افراد نباید در انجام کارها عقب بیافتند و یا از هم سبقت بگیرند، به عنوان نمونه عملیات رنگ ساختمان نمی‌تواند قبل از عملیات گچ انجام شود و یا عملیات نمای کف ساختمان نمی‌تواند قبل از لوله‌کشی صورت گیرد. لذا محدوده هر عمل باید مشخص و نیز به موقع و در موعد خودش انجام شود. بنابراین می‌توان گفت، همگامی افراد نقش مهمی در پیشبرد عملیات ساخت دارد و هنگامی که وابستگی کار وجود دارد، ترتیب درست انجام کارها اهمیت پیدا می‌کند و همه افراد همکار، باید دقیقاً به صورت هماهنگ و به موقع عمل کنند.

حال در دنیای کامپیوتر و در یک سیستم تک‌پردازنده‌ای و چندبرنامه‌ای، پردازه‌ها در طول زمان در بین یکدیگر اجرا می‌شوند تا اجرای همزمان داشته باشند. اگرچه پردازش موازی واقعی حاصل نشده است و تعویض پردازه‌ها موجب سربار قابل ملاحظه‌ای است، با وجود این، **در بین هم‌اجرا شدن** پردازه‌ها مزایای زیادی را در پردازش مؤثر و سازماندهی برنامه موجب می‌شود. در سیستم‌های چندپردازنده‌ای، نه تنها ممکن است پردازه‌ها در بین یکدیگر اجرا شوند، بلکه می‌توانند واقعاً به موازات هم و با همپوشانی اجرا گردند.

در نگاه اول، ممکن است در بین هم قرارگیری و همپوشانی، حالت‌های اجرایی و مسائل کاملاً متفاوتی به‌نظر برسند. اما در حقیقت هر دوی آن‌ها را می‌توان به عنوان مثال‌هایی از پردازش همزمان دید که هر دو مسائل یکسانی را ارائه می‌کنند. در مورد سیستم تک‌پردازنده‌ای، مسائلی که به‌وسیله چند برنامه‌ای مطرح می‌شود از این حقیقت ناشی شده است که سرعت اجرای پردازه‌ها را نمی‌توان از قبل پیش‌بینی کرد. بلکه به پردازه‌های دیگر، چگونگی رفتار سیستم عامل با وقفه‌ها و به سیاست‌های زمان‌بندی سیستم عامل بستگی دارد. مشکلات مربوط به سیستم‌های تک‌پردازنده‌ای به صورت زیر بیان می‌شود:

۱ـ اشتراک منابع سراسری پرمخاطره است. برای مثال، اگر دو پردازه از متغیرهای سراسری مشترکی استفاده کنند و هر دو اعمال خواندن و نوشتن را در مورد آن‌ها انجام دهند، در این صورت ترتیب اجرای خواندن‌ها و نوشتن‌ها بحرانی است.

۲ـ مدیریت تخصیص بهینه منابع سیستم عامل مشکل است. برای مثال، ممکن است پردازه A استفاده از یک کانال ورودی/خروجی خاص را درخواست کند و به آن هم داده شود. ولی قبل از استفاده از کانال، معلق گردد. مطلوب نیست، سیستم عامل این کانال را قفل کرده و از استفاده پردازه‌های دیگر جلوگیری نماید؛ چرا که ممکن است موجب بن‌بست گردد.

۳ـ تعیین محل خطای برنامه‌سازی سخت می‌شود؛ چرا که معمولاً نتایج قطعی نیست و نمی‌توان آن‌ها را مجدداً تولید کرد.

تمام مسائل فوق، در سیستم‌های چندپردازنده‌ای هم وجود دارد، زیرا در آن‌جا هم سرعت اجرای پردازه‌ها قابل پیش‌بینی نیست. سیستم چندپردازه‌ای باید به مسائلی که از همروندی اجرای پردازه‌ها ناشی می‌شود، نیز بپردازد. برای مشاهده چگونگی ارتباط بین پردازه‌ها در عمل، به یک مثال ساده اشاره می‌کنیم:

مثال۲: رویه زیر را درنظر بگیرید:

```
void echo()
{
    chin = getchar();
    chout = chin;
    putchar(chout);
}
```

این رویه عناصر اصلی برنامه‌ای که کارش نمایش یک کاراکتر است را نشان می‌دهد. ورودی از صفحه کلید است، که هربار یک کلید گرفته می‌شود. هر کاراکتر ورودی در متغیر chin ذخیره می‌گردد. سپس به متغیر chout منتقل شده و بعد خروجی نمایش داده می‌شود. هر برنامه‌ای می‌تواند این رویه را مکرراً فراخوانی نماید تا ورودی کاربر را دریافت کرده و روی صفحه نمایش او، به نمایش گذارد.

حال فرض کنید یک سیستم تک‌پردازنده‌ای و چندبرنامه‌ای داریم که از یک کاربر حمایت می‌کند. کاربر می‌تواند از یک برنامه کاربردی به برنامه کاربردی دیگر برود و همه برنامه‌ها از یک صفحه کلید برای ورودی و از یک صفحه نمایش برای خروجی استفاده کنند. نظر به این‌که هریک از برنامه‌های کاربردی نیاز به استفاده از رویه echo دارند، منطقی است که این رویه اشتراکی بوده و در بخشی از حافظه که برای تمام برنامه‌های کاربردی سراسری محسوب می‌شود، بارگذاری شود. بنابراین برای صرفه‌جویی، فقط یک نسخه از این رویه در حافظه وجود دارد.

اشتراک حافظه در بین پردازه‌ها، از این نظر که تعامل کارآمد پردازه‌ها را میسر می‌سازد، سودمند است. اما این اشتراک می‌تواند منجر به مشکلاتی شود. دنباله زیر را درنظر بگیرید:

۱ـ پردازه P_1 رویه echo را فراخوانی و بلافاصله بعد از عمل ورودی، با وقفه مواجه می‌شود. در همین موقع، آخرین کاراکتری که وارد شده (یعنی x) در متغیر chin ذخیره شده است.

۲ـ پردازه P_2 فعال می‌شود و رویه echo را فراخوانی می‌کند و تا پایان اجرا می‌شود، یعنی ورودی و سپس نمایش کاراکتر y روی صفحه نمایش را انجام می‌دهد.

۳ـ پردازه P_1 از سر گرفته می‌شود. در این لحظه، مقدار x در chin بازنویسی شده و در نتیجه از بین رفته است. به جای آن، محتوای chin برابر y است که به متغیر chout منتقل و نمایش داده می‌شود.

بنابراین کاراکتر اول از دست رفته و کاراکتر دوم، ۲ بار نمایش داده شده است. اساس این مشکل، متغیر مشترک chin است. پردازه‌های متعددی به این متغیر دسترسی دارند و رقابت بر سر آن به‌وجود می‌آید. اگر پردازه‌ای متغیر سراسری را تغییر دهد و سپس با وقفه روبرو شود، پردازه دیگری می‌تواند مقدار این متغیر را تغییر دهد، قبل از این‌که این متغیر توسط پردازه اول مورد استفاده قرار گیرد. اما فرض کنید بگوییم در هر لحظه فقط یک پردازه می‌تواند در این رویه باشد. در این صورت دنباله قبل، نتایج زیر را خواهد داشت:

۱ـ پردازه P_1، رویه echo را فراخوانی و بلافاصله بعد از تکمیل ورودی با وقفه مواجه می‌شود. در این لحظه، آخرین کاراکتری که وارد شده است، x در متغیر chin قرار دارد.

۲ـ پردازه P_2 فعال شده و رویه echo را فراخوانی می‌کند. ولی P_1 هنوز در رویه echo قرار دارد. P_2 از ورود به این رویه منع می‌شود (مسدود می‌گردد). بنابراین P_2 معلق و منتظر فراهم شدن قابلیت دسترسی به رویه echo است.

۳ـ بعد از مدت زمان کوتاهی، P_1 از سر گرفته شده و اجرای echo را کامل می‌کند. کاراکتر مقتضی (x) نمایش داده می‌شود.

۴ـ زمانی که P_1، رویه echo را ترک می‌کند، مانع را از P_2 برمی‌دارد. (مسدود بودن P_2 را برطرف می‌کند) و زمانی که P_2 از سر گرفته می‌شود، رویه echo با موفقیت فراخوانی خواهد شد.

این مثال اهمیت ترتیب دسترسی به متغیرهای مشترک را برای ما روشن می‌سازد. همچنین به ما نشان می‌دهد که متغیرهای مشترک (و دیگر منابع مشترک) لازم است محافظت شوند و راه‌حل آن کنترل دستیابی به منابع مشترک است.

مثال۳: قطعه برنامه زیر را درنظر بگیرید:

```
const int n = 50;
int tally;
void total()
{
  int count;
  for (count = 1; count<= n; count++){
    tally++;
  }
}
void main()
{
  tally = 0;
  parbegin (total (), total () );
  write (tally);
}
```

کران پایین و کران بالای مقدار نهایی متغیر مشترک tally که به‌وسیله این برنامه همروند در خروجی ارائه می‌شود، کدام گزینه است؟ (فرض کنید پردازه‌ها با هر سرعت نسبی می‌توانند اجرا شوند).

۱) $50 < tally \leq 100$ ۲) $0 < tally \leq 50$ ۳) $2 \leq tally \leq 100$ ۴) $2 \leq tally \leq 50$

☑ **پاسخ:** گزینه «۳» لازم به ذکر است $parbegin(P_1, P_2)$ به این معنی است که اجرای برنامه اصلی را به حالت تعلیق درآورد، اجرای همروند رویه‌های P_1 و P_2 را شروع کند، وقتی اجرای آن‌ها پایان یافت، برنامه اصلی را از سر بگیرد. با توجه به مقدار $n = 50$ و اجرای همروند دو پردازه با کد ( )total، در نگاه اول به‌نظر می‌رسد محدوده متغیر مشترک tally برابر $50 \leq tally \leq 100$ باشد. اما با بررسی دقیق‌تر متوجه می‌شویم که به‌طور دقیق $2 \leq tally \leq 100$ خواهد بود.

حداکثر مقداری که متغیر tally می‌تواند داشته باشد برابر ۱۰۰ است چرا که هر پردازه حداکثر ۵۰ بار اجرا می‌شود. اما برای تعیین حداقل مقدار tally، دستور tally را به صورت دستور اسمبلی زیر درنظر بگیرید:

```
LOAD    REG,TALLY
INC     REG
STORE   TALLY,REG
```

حال فرض کنید دنباله اجراهای زیر اتفاق می‌افتد:

۱ـ پردازه P_1: مقدار tally یعنی صفر را بارگذاری می‌کند. سپس tally را یک واحد افزایش می‌دهد. اما بعد از آن پردازنده را از دست می‌دهد. (یک واحد به مقدار رجیستر افزوده شده اما هنوز این مقدار ذخیره نشده است.)

مدرسان شریف

فصل ششم

«بن‌بست (Dead lock)»

مقدمه

در این فصل قصد داریم به یکی دیگر از جنبه‌های کنترل همروندی پردازه‌ها و نخ‌ها، یعنی بن‌بست بپردازیم. در یک محیط چند برنامگی، امکان دارد که پردازه‌های متعددی برای دستیابی به تعدادی منبع متناهی به رقابت بپردازند. هر پردازه، منابعی درخواست می‌کند. اگر منابع در آن لحظه در دسترس نباشند، پردازه وارد یک حالت انتظار می‌شود. گاهی اوقات پردازه‌ای که منتظر است، هرگز نمی‌تواند دوباره تغییر حالت بدهد، چون منابعی که درخواست کرده است در اختیار پردازه‌های منتظر دیگری است. این وضعیت بن‌بست نامیده می‌شود.

این فصل با مبحثی در مورد اصول بن‌بست، شرایط و دلایل وقوع آن آغاز و با سه رویکرد اصلی در اداره و رسیدگی با بن‌بست، یعنی پیشگیری و اجتناب، کشف بن‌بست و نادیده گرفتن بن‌بست ادامه می‌یابد. رویکردهایی که در این فصل بررسی می‌کنیم، در واقع روش‌هایی هستند که سیستم عامل می‌تواند برای جلوگیری یا رفع بن‌بست‌ها به کار ببرد. اگرچه برخی برنامه‌های کاربردی می‌توانند برنامه‌هایی را که در بن‌بست هستند، شناسایی کنند، سیستم عامل‌ها نوعاً امکاناتی برای پیشگیری از بن‌بست‌ها فراهم نمی‌کنند و مسئولیت برنامه‌نویسان است که اطمینان بدهند برنامه‌های عاری از بن‌بست را طراحی می‌کنند. مسائل بن‌بست، با جهت‌گیری‌های فعلی، اعم از تعداد زیاد پردازه‌ها، برنامه‌های چند نخی، منابع بسیار زیاد در یک سیستم تأکید بر سرورهای بانک اطلاعاتی و فایل‌هایی با طول عمر زیاد به جای سیستم‌های باز هم می‌توانند محدوده وسیع‌تری داشته باشند.

درسنامه (۱): اصول بن بست

هر سیستم، شامل تعدادی منبع متناهی است که مابین تعدادی پردازه‌ی رقابت کننده توزیع می‌شوند. این منابع انواع متعددی داشته و تعدادی نمونه یکسان از هر نوع وجود دارد. فضای حافظه، دورهای پردازنده، فایل‌ها و دستگاه‌های ورودی/خروجی (مانند چاپگرها و دیسک‌گردان‌ها)، مثال‌هایی از انواع منابع به شمار می‌روند. اگر سیستمی دو پردازنده داشته باشد، در این صورت نوع منبع پردازنده، دو نمونه دارد. به طور مشابه، نوع منبع چاپگر می‌تواند پنج نمونه داشته باشد.

هر پردازه باید پیش از استفاده منبع، آن را درخواست نموده و بعد از استفاده رها سازد. هر پردازه می‌تواند برای انجام کاری که برعهده اوست، به هر تعداد از منابع که نیاز دارد، درخواست نماید. بدیهی است که تعداد منابع درخواستی، نمی‌تواند از تعداد کل منابع موجود سیستم فراتر رود. به عبارت دیگر، یک پردازه نمی‌تواند در سیستمی که تنها دو چاپگر دارد، درخواست سه چاپگر نماید.

استفاده از یک منبع توسط یک پردازه در شرایط معمول عملیات سیستم شامل سه مرحله است:

۱ـ درخواست منبع: پردازه، منبع را درخواست می‌نماید.

۲ـ استفاده از منبع: پردازه، می‌تواند روی منبع عملیات داشته باشد.

۳ـ رهاسازی منبع: پردازه، منبع را رها می‌سازد.

توجه کنید در مرحله اول، چنانچه درخواست نتواند بلافاصله اعطا شود، در این صورت پردازه‌ی درخواست کننده، باید منتظر شود تا این که بتواند منبع را کسب کند.

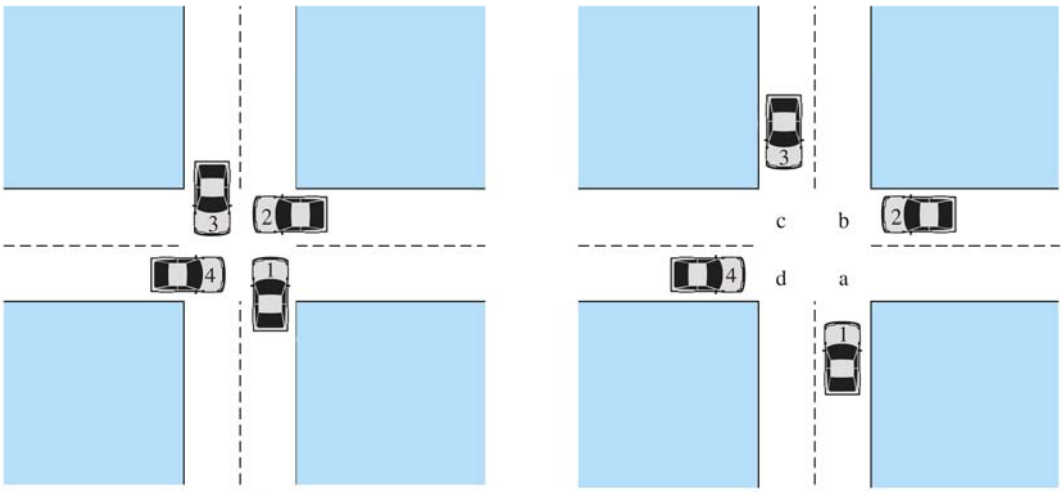
بن‌بست را می‌توان به طور رسمی چنین تعریف کرد:

یک مجموعه از پردازه‌ها در حالت بن‌بست هستند، هرگاه هریک از پردازه‌های آن مجموعه در انتظار رویدادی باشند، تنها یکی از پردازه‌های همان مجموعه می‌تواند موجب وقوع آن شود.

رویدادهایی که بیشتر در اینجا مدنظر ما قرار دارند، کسب و رهایی منبع می‌باشند. در اغلب مواقع، رویدادی که پردازه‌های دیگر منتظر آن هستند، چیزی نیست جز رها کردن منبعی که یک پردازه در اختیار گرفته و سایر پردازه‌ها در انتظار برای رهایی آن معطل مانده‌اند. به عبارت دیگر، هر پردازه در این مجموعه منتظر منبعی است که در اختیار یکی دیگر از پردازه‌های همان مجموعه قرار دارد: هیچ پردازه‌ای نمی‌تواند اجرا شود؛ رها کردن منابع امکان ندارد و همه آنها تا ابد در انتظار خواهند ماند.

در اینجا می‌خواهیم یک مثال معمول بن‌بست، خارج از بحث کامپیوتر، یعنی بن‌بست ترافیک را بررسی کنیم. شکل ۶ـ۱ شرایطی را نشان می‌دهد که در آن ۴ خودرو، تقریباً در یک زمان به چهارراه رسیده‌اند. چهار ربع (یک چهارم) این تقاطع، منابعی هستند که کنترل آنها مورد نیاز است. به ویژه اگر هر چهار خودرو بخواهند مستقیم از چهار راه عبور کنند، به صورت زیر نیازمند منابعی می‌باشند:

- خودروی ۱، که به سمت شمال می‌رود، احتیاج به قسمت‌های a و b دارد.
- خودروی ۲، که به سمت غرب می‌رود، احتیاج به قسمت‌های b و c دارد.
- خودروی ۳، که به سمت جنوب می‌رود، احتیاج به قسمت‌های c و d دارد.
- خودروی ۴، که به سمت شرق می‌رود، احتیاج به قسمت‌های b و a دارد.



شکل۶ـ۱ . نمایش بن‌بست ترافیک

اگر چهار خودرو، تقریباً همزمان به چهارراه برسند، هر کدام از ورود به تقاطع پرهیز می‌کنند، که این موجب یک بن‌بست بالقوه می‌شود. این بن‌بست بالقوه (و نه بالفعل) است، چرا که منابع مورد نیاز برای ادامه راه خودروها موجود است. اگر یکی از خودروها بالاخره جلو برود، می‌تواند عبور کند.

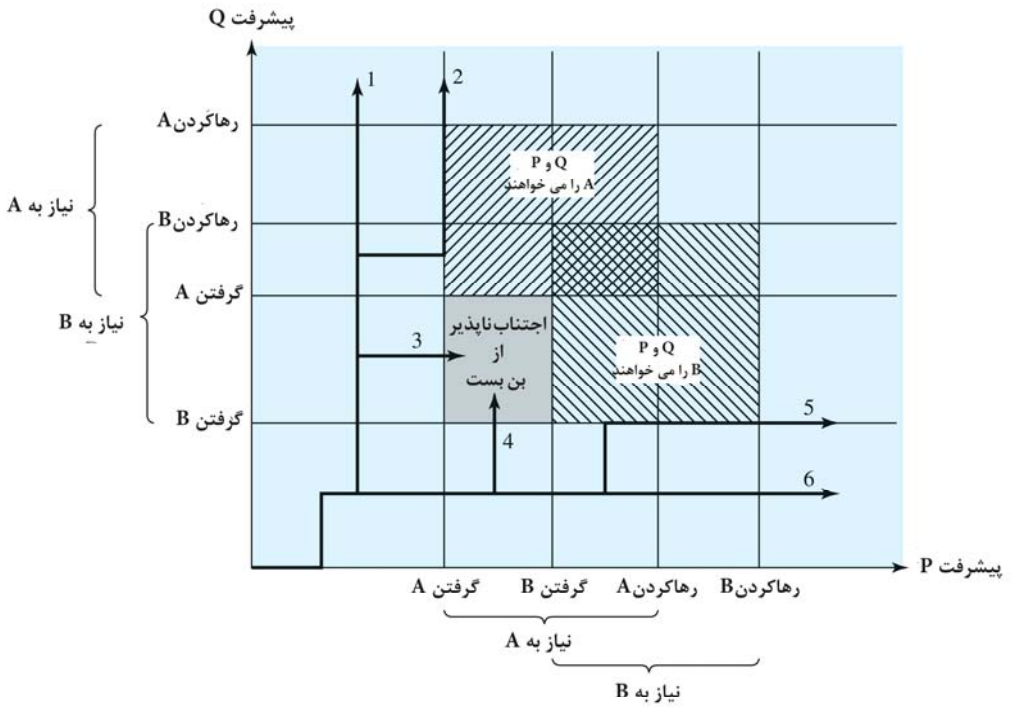
اما اگر هر چهار خودرو قوانین را نادیده گرفته و وارد تقاطع شوند، در این صورت هر خودرو یک منبع (یک چهارم از چهارراه) را در اختیار می‌گیرد، ولی نمی‌تواند پیش برود، زیرا منبع مورد نیاز دوم، هم اکنون در اختیار خودروی دیگر است. این بن‌بست بالفعل است.

حال بن‌بستی را در نظر بگیریم که پردازه‌ها و منابع کامپیوتر در آن درگیر هستند. شکل ۶ـ۲ که به آن **نمودار پیشرفت مشترک (Joint progress diagram)** می‌گوییم، پیشرفت دو پردازه رقیب برای دو منبع را نشان می‌دهد. هر پردازه برای مدت مشخص احتیاج به استفاده انحصاری از هر دو منبع دارد. پردازه‌های P و Q دارای شکل کلی زیر هستند:

Process P	Process Q
⋮	⋮
Get A	Get B
⋮	⋮
Get B	Get A
⋮	⋮
Release A	Release B
⋮	⋮
Release B	Release A
⋮	⋮

در شکل ۶ـ۲، محور x ، دلالت بر پیشرفت اجرای P و محور y ، بیانگر پیشرفت اجرای Q می‌باشد. بنابراین پیشرفت مشترک دو پردازه، توسط مسیری که از مبدأ شروع و به سمت شمال شرق می‌رود نمایش داده شده است. در یک سیستم تک پردازنده، هر بار فقط یک پردازه می‌تواند اجرا شود. پس در مسیری که شامل قسمت‌های افقی و عمودی است قسمت‌های افقی، نمایشگر دوره اجرای P و انتظار Q و قسمت‌های عمودی بیانگر نوبت اجرای Q و انتظار P می‌باشند. شکل، نواحی مختلف را نشان می‌دهد.

P و Q هر دو نیازمند منبع A هستند؛ P و Q هر دو نیازمند منبع B هستند؛ P و Q هر دو نیازمند هم A و هم B هستند. چون فرض ما بر این است که پردازه‌ها نیاز به کنترل انحصاری هر منبع دارند. این نواحی همه، جزء نواحی ممنوعه محسوب می‌شوند. یعنی مسیری که پیشرفت مشترک P و Q را نشان می‌دهد، امکان ندارد وارد این ناحیه گردد.



= P و Q هردو A را می خواهند.

= P و Q هردو B را می خواهند.

= ناحیه اجتناب ناپذیر از بن بست (وقوع بن بست حتمی است).

مسیر پیشبرد P و Q

بخش افقی مسیر، دلالت بر اجرای P و انتظار Q دارد.

بخش عمودی مسیر، دلالت بر اجرای Q و انتظار P دارد.

شکل ۶ـ۲. نمودار پیشرفت مشترک بن‌بست

شکل ۶ـ۲ ، شش مسیر مختلف اجرا را نمایش می‌دهد. می‌توان این مسیرها را به صورت زیر خلاصه نمود:

۱ـ Q منبع B و سپس منبع A را در اختیار گرفته و بعد B و A را رها می‌کند. وقتی اجرای P از سر گرفته می‌شود، قادر خواهد بود که هر دو منبع را به دست آورد.

۲ـ Q منبع B و سپس منبع A را در اختیار می‌گیرد. P اجرا شده و روی درخواست A مسدود می‌گردد. Q منابع B و A را رها می‌کند. وقتی اجرای P از سرگرفته می‌شود، قادر خواهد بود که هر دو منبع را بدست آورد.

۳ـ Q منبع B و سپس P منبع A را در اختیار می‌گیرد. بن‌بست اجتناب‌ناپذیر است (وقوع بن‌بست حتمی است)، چرا که با ادامه اجرا، Q برای A و همچنین P برای B مسدود خواهند شد.

۴ـ P منبع A و سپس Q منبع A را در اختیار می‌گیرد. بن‌بست اجتناب‌ناپذیر است (وقوع بن‌بست حتمی است)، چرا که با ادامه اجرا، Q برای A و P برای B مسدود خواهند شد.

۵ـ P منبع A و سپس منبع B را در اختیار می‌گیرد. Q اجرا شده و روی درخواست B مسدود می‌گردد. P منابع A و B را رها می‌کند. وقتی اجرای Q از سرگرفته می‌شود، قادر خواهد بود که هر دو منبع را به دست آورد.

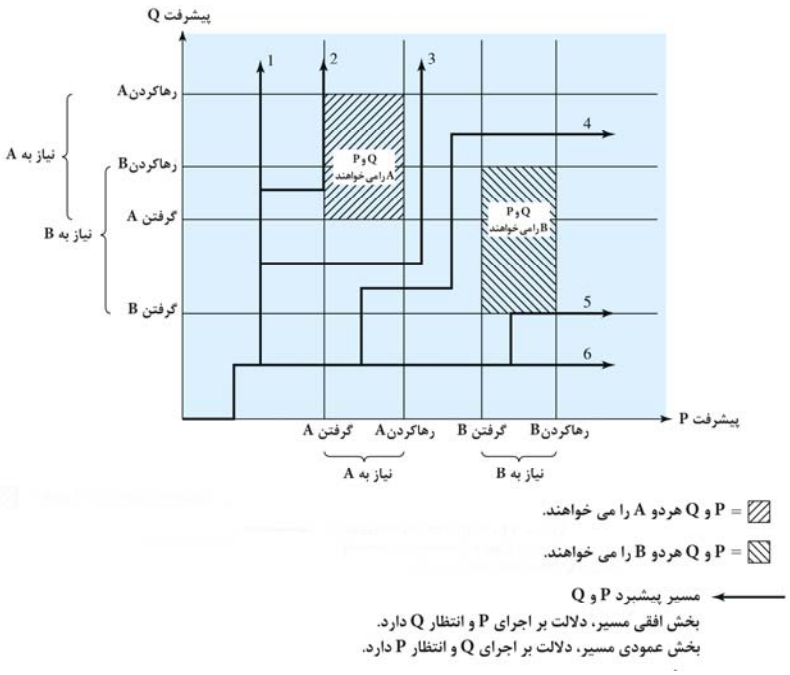
۶ـ P منبع A و سپس منبع B را در اختیار گرفته و بعد B و A را رها می‌سازد. وقتی اجرای Q از سر گرفته می‌شود، قادر خواهد بود که هر دو منبع را به دست آورد.

همانطور که در شکل ۶ـ۲ مشخص است، ناحیه خاکستری، ناحیه مرگبار است و مربوط به مسیرهای ۳ و ۴ است. اگر مسیر اجرا وارد این ناحیه مرگبار شود، در این صورت بن‌بست حتمی است. توجه کنید که وجود ناحیه مرگبار به منطق دو پردازه بستگی دارد. اما بن‌بست تنها زمانی حتمی خواهد بود که پیشرفت مشترک این دو پردازه، مسیری را تولید کند که وارد این ناحیه مرگبار گردد.

البته وقوع بن‌بست، هم به چگونگی اجرا و هم به جزئیات برنامه کاربردی بستگی دارد. برای نمونه، فرض کنید پردازه P در یک زمان به هر دو منبع نیاز دارد و دو پردازه به شکل زیر هستند:

```
Process P          Process Q
    ⋮                  ⋮
  Get A              Get B
    ⋮                  ⋮
Release A            Get A
    ⋮                  ⋮
  Get B              Release B
    ⋮                  ⋮
Release B            Release A
    ⋮                  ⋮
```

شکل ۶ـ۳ این حالت را نشان می‌دهد. همانطور که می‌بینید، بن‌بست ایجاد نمی‌شود.



شکل ۶ـ۳. نمودار پیشرفت مشترک؛ عدم بن‌بست.

همانطور که نشان داده شد، می‌توان از نمودار پیشرفت مشترک، برای ثبت سابقه اجرایی دو پردازه‌ای که در منابع شریک هستند، استفاده نمود. در شرایطی که امکان رقابت بیشتر از دو پردازه برای منبع مشترک مطرح است، نیاز به نموداری با ابعاد بیشتر خواهد بود.

فصل هفتم

«مدیریت حافظه»

مقدمه

حافظه اصلی یکی از مهم‌ترین منابع هر سیستم کامپیوتری است و مدیریت حافظه یکی از مشکل‌ترین جنبه‌های طراحی سیستم عامل است. هدف اصلی سیستم کامپیوتری، اجرای برنامه‌هاست. حداقل بخشی از این برنامه‌ها همراه با داده‌هایی که به آن دسترسی دارند، باید در زمان اجرا در حافظه اصلی باشند. با وجود اینکه امروزه بهای حافظه به‌طور قابل توجهی کاهش یافته و در نتیجه اندازه حافظه اصلی در سیستم‌های جدید به محدوده گیگابایت رشد کرده است، ولی هرگز حافظه اصلی برای نگهداری تمامی برنامه‌ها و ساختمان داده‌های مورد نیاز پردازه‌های فعال و همچنین خود سیستم عامل کافی نخواهد بود. از این‌رو، یکی از وظایف اصلی سیستم عامل، **مدیریت حافظه** است.

در واقع در یک سیستم تک برنامه‌ای، حافظه به دو بخش تقسیم می‌گردد؛ یک بخش برای سیستم عامل و یک بخش برای برنامه در حال اجرا. در سیستم چند برنامگی (همه منظوره)، بخش کاربر حافظه باید به زیربخش‌های دیگر تقسیم شود تا بتواند چند پردازه را در خود جای بدهد و چندین برنامه را در حافظه نگهداری نماید. وظیفه تقسیم‌بندی حافظه به زیربخش‌ها به صورت پویا در زمان اجرا توسط سیستم عامل انجام می‌گیرد و به این عمل، مدیریت حافظه می‌گویند. وجود مدیریت حافظه کارآمد برای یک سیستم کامپیوتری همه منظوره حیاتی است.

طرح‌های مدیریت حافظه متعددی وجود دارند که منعکس‌کننده روش‌های مختلفی است و اثربخشی هر یک به وضعیت مربوطه بستگی دارد. در این فصل، ابتدا ملزومات اولیه هر طرح مدیریت حافظه، به اختصار مطرح می‌شود. سپس راه‌های مختلف مدیریت حافظه را بحث می‌کنیم. این فصل روش‌های اولیه مبتنی بر ماشین (پارتیشن‌بندی ایستا و پویا) گرفته تا راهبردهای صفحه‌بندی و قطعه‌بندی را شامل می‌شود.

> **یک هدف مشترک بین تمام طرح‌های مدیریت حافظه این است که پردازه‌های بیشتری را همزمان در حافظه نگه دارند تا چند برنامگی ایجاد شود.**

لازم به‌ذکر است در فصل ۸، مدیریت حافظه مجازی را بر پایه به کارگیری صفحه‌بندی، قطعه‌بندی یا ترکیبی از صفحه‌بندی و قطعه‌بندی (قطعه‌بندی صفحه‌بندی شده) بررسی می‌کنیم.

درسنامه (۱): ملزومات مدیریت حافظه

در تمامی راهبردها و سیاست‌های مربوط به مدیریت حافظه، ملزومات زیر باید برآورده شوند:

- جابه‌جایی (Relocation)
- حفاظت (Protection)
- اشتراک (Sharing)
- سازمان منطقی (Logical Organization)
- سازمان فیزیکی (Physical Organization)

در ادامه هر یک از ملزومات فوق را به اختصار شرح می‌دهیم.

جابه‌جایی

در سیستم‌های چند برنامگی، معمولاً چند پردازه به صورت همزمان در حافظه اصلی می‌باشند و حافظه موجود بین آن‌ها تقسیم می‌شود. معمولاً برای برنامه‌نویس غیرممکن است که از قبل بداند چه برنامه‌های دیگری در زمان اجرای یک برنامه، مقیم حافظه خواهند بود. از طرف دیگر مایلیم بتوانیم پردازه‌های فعال را به داخل یا خارج حافظه مبادله نماییم و از طریق ایجاد مجموعه بزرگی از پردازه‌های آماده به اجرا، بهره‌وری پردازنده را به حداکثر برسانیم. بنابراین اگر حافظه اصلی، فضای کافی را برای ورود پردازه‌های جدید نداشته باشد، باید اجازه دهیم از طریق یکی از شیوه‌های زیر، امکان جابه‌جایی فراهم شود:

۱ـ **مبادله یا معاوضه** (swapping): پردازه‌های غیرفعال، به‌طور موقت به دیسک منتقل می‌شوند تا فضا برای ورود پردازه‌های فعال جدید به داخل حافظه فراهم شود.

۲ـ **حافظه مجازی** (virtual memory): بخشی از حافظه ثانویه (مانند دیسک) به عنوان حافظه

در ادامه فصل ۷ و ۸، هر یک از آن‌ها را به طور دقیق مورد بررسی قرار می‌دهیم.

حفاظت

هر پردازه باید در مقابل تداخل‌های ناخواسته پردازه‌های دیگر محافظت شود، خواه این تداخل تصادفی و خواه عمدی باشد. پس برنامه‌های پردازه‌های دیگر نباید قادر باشند برای اهداف خواندن یا نوشتن، به محل‌های حافظه یک پردازه، بدون اجازه مراجعه نمایند. از طرفی، برآوردن نیازهای جابه‌جایی می‌تواند باعث افزایش پیچیدگی و دشواری در برآورده‌شدن نیازهای حفاظتی شود.

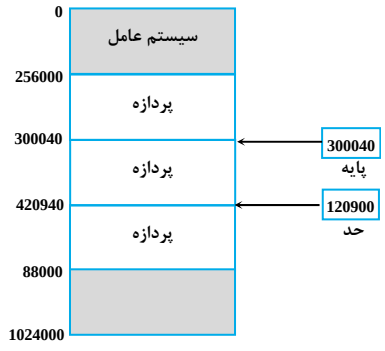
بنابراین ابتدا لازم است مطمئن شویم که هر یک از پردازه‌ها، فضای حافظه مجزایی دارند. برای انجام این‌کار، نیاز به توانایی تعیین محدوده آدرس‌های معتبر و قانونی است که پردازه می‌تواند به آن محدوده دسترسی داشته باشد و تضمین کنیم که پردازه فقط می‌تواند به این آدرس‌های معتبر دسترسی داشته باشد.

سخت‌افزار پردازنده به واسطه دو ثبات خاص این قابلیت را فراهم می‌کند.

۱ـ ثبات پایه (base): ثبات پایه، کوچکترین آدرس حافظه فیزیکی معتبر را نگه می‌دارد.

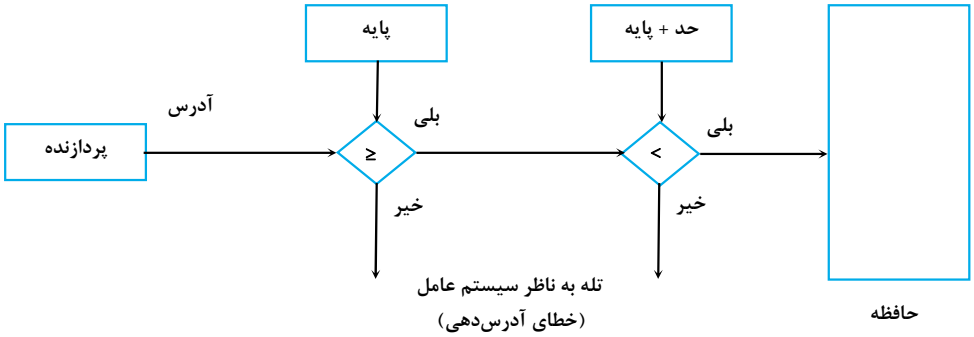
۲ـ ثبات حد (limit): ثبات حد، اندازه‌ی محدوده را مشخص می‌کند.

برای مثال، شکل ۷ـ۱ را در نظر بگیرید. اگر ثبات پایه برابر ۳۰۰۰۴۰ و ثبات حد برابر ۱۲۰۹۰۰ باشد، آنگاه، برنامه می‌تواند به صورت قانونی به آدرس‌های ۳۰۰۰۴۰ تا ۴۲۰۹۳۹ دسترسی معتبر داشته باشد.



شکل ۷ـ۱ . ثبات پایه و حد

برای حفاظت از فضای حافظه، سخت‌افزار پردازنده هر آدرس تولید شده در مُد کاربر را با مقادیر این ثبات‌ها مقایسه می‌کند. هر اقدامی توسط برنامه‌ی در حال اجرا در مُد کاربر جهت دسترسی به حافظه سیستم عامل یا حافظه کاربر دیگر، منجر به تله (Trap) نقص حفاظتی به سیستم عامل می‌شود، که به عنوان یک خطای مهلک تلقی می‌شود (شکل ۷ـ۲). این رویکرد، از تغییر تصادفی یا ناگهانی کُد یا ساختمان داده سیستم عامل و برنامه (پردازه‌های) کاربر در مقابل سایر برنامه‌های کاربر محافظت می‌کند.



شکل ۷-۲. حفاظت سخت‌افزاری حافظه با استفاده از ثبات‌های پایه و حد

ثبات‌های پایه و حد، فقط توسط سیستم عامل بارگذاری می‌شوند و برای این‌کار از دستورالعمل ممتاز خاصی استفاده می‌شود. چون دستورالعمل‌های ممتاز، فقط در مُد هسته قابل اجرا هستند و از این نظر که فقط سیستم عامل در مُد هسته اجرا می‌شود، فقط سیستم عامل می‌تواند ثبات‌های پایه و حد را بارگذاری نماید.

اشتراک

هر چند به حفاظت پردازه‌ها از دسترسی‌های غیرقانونی و غیر معتبر نیاز داریم، اما هر راهکار حفاظتی که به کار می‌رود، باید انعطاف‌پذیری لازم برای اجازه‌دادن به پردازه‌های متعدد در دسترسی به یک بخش یکسان از حافظه را فراهم سازد تا پردازه‌های همکار قادر باشند در دسترسی به یک ساختمان داده یا کُد خاص، شریک باشند.

بنابراین سیستم مدیریت حافظه باید دسترسی کنترل شده به نواحی مشترک حافظه را با در نظر گرفتن اصول حفاظتی اجازه بدهد. معمولاً راهکارهایی که برای حمایت از جابه‌جایی به‌کار می‌روند، حامی قابلیت‌های اشتراک نیز خواهند بود.

نکته۱: نیاز اشتراک به‌ویژه در محیط‌های اشتراک زمانی، مهم است.

سازمان منطقی

معمولاً در سیستم کامپیوتری، حافظه به صورت فضای آدرس خطی یا یک بعدی سازمان یافته و شامل دنباله‌ای از بایت‌ها یا کلمه‌ها است. حافظه ثانویه نیز در سطح فیزیکی به شکل مشابهی سازمان یافته است. اگرچه این ساختار منعکس‌کننده سخت‌افزار واقعی ماشین می‌باشد، ولی ارتباطی به روشی که برنامه‌ها ساخت یافته‌اند، ندارد. بسیاری از برنامه‌ها، به‌صورت مجموعه‌ای از **پیمانه‌ها (module)** با ویژگی‌های متفاوت طراحی می‌شوند. بعضی از آن‌ها غیرقابل تغییر (فقط خواندنی، فقط اجرایی) و بعضی دیگر شامل داده‌های قابل تغییر هستند. اگر سیستم عامل و سخت‌افزار کامپیوتر بتوانند به‌طور مؤثری با برنامه‌ها و داده‌های کاربر در شکل پیمانه‌ها کار کنند و همچنین برنامه‌ها به صورت پیمانه‌ای نوشته شوند، مزایایی به شرح زیر به‌دست می‌آیند:

۱ـ هر پیمانه را می‌توان به‌طور مستقل نوشت و کامپایل کرد و تمام مراجعات یک پیمانه به پیمانه‌های دیگر را سیستم در زمان اجرا حل می‌کند.

۲ـ با یک سربار مختصر، درجه‌های مختلف حفاظتی (فقط خواندنی، فقط اجرایی) می‌توانند به پیمانه‌های مختلف نسبت داده شوند.

۳ـ امکان معرفی راهکارهایی برای اشتراک‌گذاری پیمانه‌ها در بین پردازه‌ها وجود دارد.

بنابراین، سیستم و سخت‌افزار باید از وجود ساختار پیمانه‌ای در برنامه‌ها و پردازه‌ها پشتیبانی نمایند. قطعه‌بندی که یکی از راهبردهای مدیریت حافظه است، بیش از همه این نیاز را برآورده می‌سازد. این راهبرد در ادامه همین فصل بررسی می‌شود.

سازمان فیزیکی

همان‌گونه که در فصل اول، مطرح گردید، حافظه می‌تواند سطوح مختلفی داشته باشد که همان سلسله مراتب حافظه را تشکیل می‌دهد. حافظه کامپیوتر، حداقل در دو سطح سازمان یافته است: حافظه اصلی و حافظه ثانویه.

حافظه اصلی، دسترسی سریع را در ازای بهای نسبتاً زیاد ارائه می‌دهد. حافظه اصلی ناپایدار است؛ یعنی نمی‌تواند برای ذخیره دائمی مورد استفاده قرار گیرد. حافظه ثانوی کندتر و ارزان‌تر از حافظه اصلی بوده و ناپایدار هم نیست، پس حافظه ثانوی با ظرفیت زیاد برای ذخیره طول مدت برنامه و داده‌ها در دسترس قرار می‌گیرد، در حالی‌که حافظه اصلی کوچکتر، حاوی برنامه‌ها و داده‌های در حال استفاده می‌باشد و همان‌طور که در بخش جابه‌جایی عنوان شد، در صورت نبود فضای کافی برای ورود کل پردازه به حافظه اصلی می‌توان از شیوه‌هایی همچون مبادله و حافظه مجازی استفاده نمود.

در قسمت بعد، با شیوه دیگری آشنا می‌شویم که این مسئولیت یعنی روند جریان اطلاعات بین حافظه اصلی و ثانویه را به برنامه‌نویس محول می‌کند.

جای‌گذاشت (روی هم‌گذاری)

همان‌طور که قبلاً اشاره کردیم، همواره برنامه‌هایی وجود دارند که از فضای حافظه موجود بزرگتر هستند و در حافظه، فضای کافی برای ورود و بارگذاری تمام برنامه وجود ندارد. یکی از راه‌حل‌هایی که قبلاً مورد استفاده قرار می‌گرفت، روشی به‌نام جای‌گذاشت یا روی‌هم‌گذاری (Overlaying) بود.

این روش که مسئولیت آن به عهده برنامه‌نویس بود، بر این اساس کار می‌کرد که برنامه‌نویس باید برنامه‌ها را به بخش‌های مختلفی به‌نام overlay تقسیم کند و سیستم تنها آن داده‌ها و دستورالعمل‌هایی را در حافظه قرار می‌دهد که مورد نیاز برنامه است و بقیه بخش‌ها که مورد نیاز نمی‌باشد به دیسک انتقال می‌یابند. هنگامی‌که به بخش دیگر از آن برنامه نیاز داریم، عملیات مبادله از دیسک به حافظه و از حافظه به دیسک انجام می‌شود و بخشی که مورد نیاز نیست از حافظه خارج شده و بخش مورد نیاز به حافظه منتقل می‌شود.

لازم به تأکید است که مسئولیت این تکنیک بر عهده خود برنامه‌نویس است و حتی اگر کامپایلر هم در به‌کارگیری این روش، به برنامه‌نویس کمک کند، باز هم بخشی از وقت برنامه‌نویس برای استفاده از این امکان، تلف می‌شود و این کار برای او بسیار وقت‌گیر و خسته‌کننده است. از طرف دیگر، در یک محیط چندبرنامگی، فقط سیستم عامل از بخش‌های آزاد و اشغال‌شده حافظه خبر دارد. از این رو فقط سیستم عامل است که می‌تواند بهترین مدیریت را برای ورود و خروج بخش‌های مورد نظر در برنامه‌ها از حافظه اصلی انجام دهد.

مثال۱: کدام گزینه جزو ملزومات مدیریت حافظه محسوب نمی‌گردد؟

۱) جابه‌جایی ۲) سازمان منطقی ۳) سازمان فیزیکی ۴) چندبرنامگی

☑ **پاسخ:** گزینه «۴» در واقع محقق شدن چندبرنامگی از اهداف مشترک بین طرح‌های مدیریت حافظه است به گونه‌ای که پردازه‌های بیشتری را هم‌زمان در حافظه نگه دارند.

مثال۲: کدام گزینه در مورد ثبات‌های پایه و حد صحیح است؟

۱) فقط سیستم عامل می‌تواند ثبات‌های پایه و حد را بارگزاری نماید.

۲) ثبات پایه بزرگ‌ترین آدرس حافظه فیزیکی معتبر را نگهداری می‌کند.

۳) ثبات حد، آدرس محل پایان ذخیره‌سازی پردازه را ذخیره می‌کند.

۴) مقدار شمارنده برنامه می‌تواند برابر یا بیشتر از ثبات حد باشد.

☑ **پاسخ:** گزینه «۱» دستورالعمل‌های ممتاز فقط در مد هسته قابل اجرا است و از این نظر که فقط سیستم عامل در مد هسته اجرا می‌گردد، فقط سیستم عامل می‌تواند ثبات‌های پایه و حد را بارگزاری نماید.

همچنین آدرس محل شروع ذخیره‌سازی یک پردازه در حافظه اصلی در زمان اجرا در ثبات پایه ذخیره می‌شود و طول پردازه در حافظه اصلی در زمان اجرا در ثبات حد ذخیره می‌گردد.

گزینه (۴) نیز نادرست است، زیرا شمارنده برنامه و تمام مراجعات به داده‌ها نسبت به محتویات ثبات پایه تفسیر می‌شود و نباید از ثبات حد بیشتر باشد.

مثال۳: کدام گزینه عیب اصلی روش جای‌گذاشت Overlaying را بیان می‌کند؟

۱) طراحی پیچیده برنامه ۲) درگیر بودن بیش از حد برنامه‌نویس

۳) افزایش نیازمندی حافظه ۴) افزایش احتمال ایجاد خطای مهلک

☑ **پاسخ:** گزینه «۲» عیب اصلی جای‌گذاشت، درگیر بودن بیش از حد برنامه‌نویس است. مشخص ساختن قطعه‌های جای‌گذاشت نه کاری ساده و نه مطمئن است. همچنین برنامه‌نویس بایستی از نیازمندی‌های بیشتری از حافظه مطلع باشد.

فصل هشتم

«حافظه مجازی (Virtual Memory)»

مقدمه

در فصل ۷، راهبردهای مختلف مدیریت حافظه را که در سیستم‌های کامپیوتری استفاده می‌شوند، مورد بحث قرار دادیم. تمامی این راهبردها هدف یکسانی را دنبال می‌کردند: پردازه‌های بیشتر را همزمان در حافظه نگه می‌دارند تا امکان چندبرنامگی فراهم شود. با وجود این، در تمامی این راهبردها لازم است کل پردازه، پیش از اجرا به‌طور کامل در حافظه قرار بگیرند.

اما همواره از سال‌های بسیار پیش، برنامه‌هایی (پردازه‌هایی) وجود داشته‌اند که از حافظه موجود بزرگ‌تر بوده و در حافظه اصلی جای کافی برای بارگذاری تمام برنامه (پردازه) وجود نداشته است. از این‌رو، راه حل مناسبی به نام **حافظه مجازی** (**Virtual Memory**) مطرح شده است.

حافظه مجازی، تکنیکی می‌باشد که موجب می‌گردد پردازه بدون اینکه به‌طور کامل در حافظه باشد، اجرا گردد. امتیاز عمده این طرح این است که برنامه‌ها می‌توانند بزرگ‌تر از حافظه فیزیکی باشند. به علاوه حافظه اصلی را به‌صورت آرایه‌ی یکنواخت و بزرگی در نظر می‌گیرد که حافظه منطقی را از حافظه فیزیکی تفکیک می‌کند. این تکنیک موجب می‌شود تا برنامه‌نویس از محدودیت‌های حافظه نگرانی نداشته باشد. حافظه مجازی به پردازه‌ها اجازه می‌دهد به آسانی در فایل‌ها مشترک باشند و حافظه مشترک را پیاده‌سازی کنند. به علاوه، راهکار کارآمد و مؤثری را برای ایجاد پردازه مهیا می‌کند.

حافظه مجازی بر پایه به‌کارگیری صفحه‌بندی یا ترکیبی از صفحه‌بندی و قطعه‌بندی، تقریباً راهبرد همگانی برای مدیریت حافظه سیستم‌های کامپیوتری امروزی است. در این فصل، ابتدا به بررسی دقیق مفهوم حافظه مجازی و مزایای استفاده از آن می‌پردازیم و در ادامه راهکارهای مختلف مدیریت حافظه مجازی را بررسی می‌کنیم.

درسنامه (۱): حافظه و صفحه‌بندی مجازی

الگوریتم‌های مدیریت حافظه که در فصل ۷ مطرح شدند، به این دلیل ضروری هستند که دستورالعمل‌های اجرایی باید در حافظه فیزیکی قرار گیرند. اولین رهیافت برآورده کردن این نیاز این است که کل فضای آدرس منطقی، در حافظه فیزیکی قرار گیرد. به کمک بارگذاری پویا می‌توان این محدودیت را کم‌رنگ کرد، اما معمولاً کار بیشتر برنامه‌نویس را می‌طلبد.

این محدودیت که دستورالعمل‌ها باید در حافظه فیزیکی قرار بگیرند تا اجرا شوند، ضروری و منطقی به‌نظر می‌رسد، ولی چندان هم مطلوب نیست، زیرا اندازه برنامه را به اندازه حافظه فیزیکی محدود می‌کند. در حقیقت، بررسی برنامه‌های واقعی نشان می‌دهد که در بسیاری از موارد، نیاز به کل برنامه نیست. برای نمونه، مواردی که در ادامه بیان می‌شوند را در نظر بگیرید:

- برنامه‌ها اغلب کدهایی برای پردازش شرایط خطاهای غیرمعمول دارند. چون این خطاها به ندرت رخ می‌دهند، چنین کدی تقریباً هرگز اجرا نمی‌شود.
- اغلب به آرایه‌ها، لیست‌ها و جدول‌ها، حافظه‌ای بیش از اندازه‌ی مورد نیاز تخصیص می‌یابد. به عنوان نمونه، ممکن است یک آرایه با تعداد عناصر ۱۰۰×۱۰۰ تعریف شود، در حالی‌که فقط ۱۰×۱۰ عنصر را ذخیره نماید.
- بعضی از گزینه‌ها و امکانات خاص یک برنامه ممکن است به ندرت مورد استفاده قرار گیرند. برای نمونه، ممکن است بخشی از برنامه که مالیات‌هایی را برای افراد خاص محاسبه می‌کند، در بسیاری از مواقع استفاده نشود.

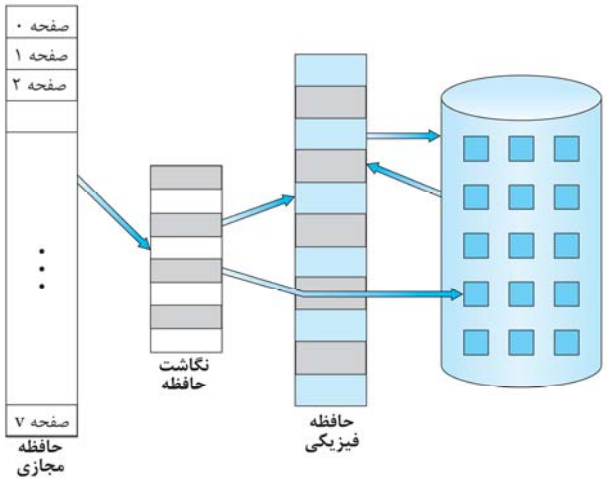
حتی در مواردی که کل برنامه مورد نیاز است، امکان دارد همزمان تمام بخش‌های برنامه لازم نشود.

قابلیت اجرای برنامه‌ای که فقط بخشی از آن در حافظه قرارگیرد، منافع زیادی به‌دنبال دارد که عبارتند از:

- اندازه برنامه دیگر به فضای فیزیکی محدود نخواهد بود. کاربران می‌توانند برنامه‌هایی برای یک فضای آدرس مجازی فوق‌العاده بزرگ بنویسند و در نتیجه کار برنامه‌نویسی هم آسان‌تر می‌شود.
- چون هر برنامه کاربر می‌تواند حافظه فیزیکی کمتری را اشغال کند، برنامه‌های بیشتری می‌توانند به‌طور همزمان اجرا شوند و در نتیجه بهره‌وری پردازنده و توان عملیاتی بدون افزایش زمان پاسخ یا زمان برگشت (زمان کل)، افزایش می‌یابد.
- برای بارگذاری یا مبادله هر برنامه کاربر به / از حافظه، به عمل I/O کمتری نیاز خواهد داشت. لذا سرعت اجرای برنامه کاربر افزایش می‌یابد.

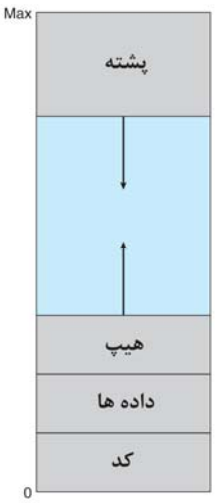
بنابراین، اجرای برنامه‌ای که به‌طور کامل در داخل حافظه قرار ندارد، هم به نفع سیستم و هم به نفع کاربر خواهد بود.

حافظه مجازی، حافظه منطقی را که توسط کاربر دیده می‌شود، از حافظه فیزیکی تفکیک می‌کند. این تفکیک موجب می‌شود حتی در حالتی که حافظه فیزیکی کوچکتری وجود دارد، باز هم حافظه مجازی بزرگی برای برنامه‌نویسان فراهم شود. (شکل ۸ـ۱) حافظه مجازی کار برنامه‌نویس را آسان‌تر می‌سازد، زیرا برنامه‌نویس دیگر لازم نیست که نگران حجم حافظه فیزیکی موجود باشد؛ لذا به‌جای آن می‌تواند بیشتر روی مسأله برنامه‌نویسی متمرکز شود.



شکل ۸ـ۱ . نمایش بزرگتر بودن حافظه مجازی از حافظه فیزیکی

فضای آدرس مجازی یک پردازه، به دید منطقی یا مجازی چگونگی ذخیره‌سازی پردازه در حافظه مربوط می‌شود. معمولاً این دیدگاه وجود دارد که پردازه در یک آدرس منطقی ویژه‌ای، مثل آدرس صفر، شروع می‌شود و همانند نمایش شکل ۸ـ۲، به‌صورت همجوار در حافظه قرار می‌گیرد. همان‌طور که در فصل پیش نیز اشاره شد، حافظه فیزیکی می‌تواند به صورت قاب‌های صفحه سازماندهی شود و قاب‌های صفحه تخصیص داده شده به یک پردازه ممکن است همجوار نباشند. این مسئولیت واحد مدیریت حافظه (MMU) است که صفحه‌های منطقی را به قاب‌های صفحه فیزیکی در حافظه نگاشت می‌کند.



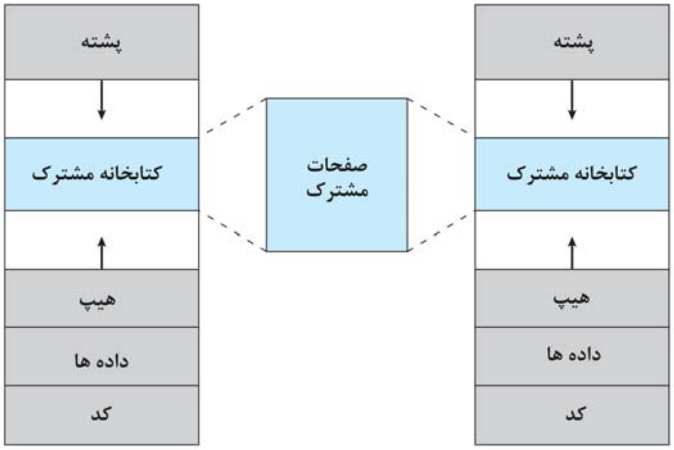
شکل ۸ـ۲. فضای آدرس مجازی

به شکل ۸ـ۲ توجه کنید، اجازه می‌دهیم هیپ در حافظه به سمت بالا رشد نماید، زیرا برای تخصیص حافظه پویا استفاده می‌شود. به‌طور مشابه، پشته در حافظه، از طریق فراخوانی متوالی توابع، به سمت پایین رشد می‌کند.

فضای خالی بزرگ (یا حفره‌ای) بین هیپ و پشته، بخشی از فضای آدرس مجازی محسوب می‌شود، ولی تنها در صورتی صفحات فیزیکی واقعی را لازم خواهد داشت که هیپ یا پشته رشد داشته باشند. فضای آدرس مجازی که حفره‌هایی را دربر دارند به فضای آدرس اسپارس (خلوت) معروف هستند. استفاده از فضای آدرس اسپارس، مفید است، چرا که حفره‌ها می‌توانند با رشد هیپ یا پشته پر شوند، یا با کتابخانه‌های پیوند پویا (یا سایر اشیای مشترک) در حین اجرای برنامه، پُر شوند.

حافظه مجازی علاوه بر تفکیک حافظه منطقی از حافظه فیزیکی، امکان اشتراک فایل‌ها و حافظه را میان دو یا چند پردازه از طریق اشتراک صفحه، فراهم می‌کند. این امکان منافع زیر را به دنبال دارد:

- کتابخانه‌های سیستمی می‌توانند از طریق نگاشت شیء مشترک به فضای آدرس مجازی، بین چندین پردازه به اشتراک گذاشته شوند. اگرچه هر پردازه، کتابخانه‌های مشترک را بخشی از فضای آدرس مجازی در نظر می‌گیرد، ولی صفحات واقعی این کتابخانه‌ها که در حافظه فیزیکی مقیم می‌باشند، بین تمام پردازه‌ها مشترک است (شکل ۸ـ۳). کتابخانه‌ای که به فضای آدرس هر یک از پردازه‌ها پیوند دارد، نوعاً نگاشت فقط خواندنی دارد.
- به‌طور مشابه، حافظه مجازی پردازه‌ها را قادر می‌سازد تا در حافظه مشترک باشند. حافظه مجازی اجازه می‌دهد که پردازه، ناحیه‌ای از حافظه را برای اشتراک با پردازه‌های دیگر ایجاد کند. پردازه‌هایی که در این ناحیه از حافظه مشترک هستند، آن را بخشی از فضای آدرس مجازی خودشان تصور می‌کنند، در حالی‌که صفحات فیزیکی واقعی مشترک هستند.
- حافظه مجازی، امکان اشتراک صفحات را در اثنای ایجاد پردازه با فراخوان سیستمی () fork فراهم می‌کند، که این امر ایجاد پردازه را سرعت می‌بخشد.



شکل ۸ـ۳. کتابخانه مشترک با استفاده از حافظه مجازی

صفحه‌بندی مجازی

در راهبرد مدیریت حافظه **صفحه‌بندی مجازی (Virtual Paging)**، همانند صفحه‌بندی ساده، پردازه به مجموعه‌ای از صفحه‌ها تقسیم می‌شود. تقسیم حافظه منطقی و حافظه فیزیکی نیز به مجموعه‌ای از قاب‌ها تقسیم می‌گردد. همچنین، سیستم عامل باید یک جدول صفحه برای هر پردازه نگهداری کند تا نشان دهد هر صفحه در کدام قاب است.

با توجه به این‌که در تمامی تکنیک‌های حافظه مجازی، از جمله صفحه‌بندی، لازم نیست که برای اجرای یک پردازه، کل پردازه به حافظه اصلی آورده شود، لذا در صفحه‌بندی مجازی، فقط صفحه‌های مورد نیاز پردازه، در قاب‌های خالی قرار می‌گیرند. این مجموعه صفحه‌های درون قاب با نام **مجموعه مقیم (Resident set)**، مشهور است. به عبارت دیگر، به مجموعه قاب‌های اختصاص داده شده به یک پردازه مجموعه مقیم آن پردازه گفته می‌شود و مدیریت آن بر عهده سیستم عامل است.

صفحه‌بندی ساده	تمام صفحه‌های یک پردازه باید در حافظه اصلی باشند تا پردازه اجرا شود، مگر این‌که، روش جای‌گذاشت (روی هم‌گذاری) به کار رفته باشد.
صفحه‌بندی مجازی	لزومی ندارد تمام صفحه‌های یک پردازه، در قاب‌های حافظه اصلی باشند تا پردازه اجرا شود؛ صفحه‌ها می‌توانند برحسب نیاز به داخل خوانده شوند.

ساختار درایه‌های جدول صفحه در صفحه‌بندی مجازی

در مبحث صفحه‌بندی ساده (فصل ۷) گفته شد که هر پردازه، جدول صفحه خود را دارد و هرگاه تمام صفحه‌های آن در حافظه اصلی بارگذاری شدند، جدول صفحه برای آن پردازه ایجاد و داخل حافظه اصلی بارگذاری می‌شود. هر درایه جدول صفحه، شماره قاب صفحه مربوط در حافظه اصلی را دربر دارد. همین ابزار، یعنی جدول صفحه، در مورد حافظه مجازی بر پایه صفحه‌بندی نیز مورد نیاز است. در این‌جا نیز معمول است که یک جدول صفحه یکتا برای هر پردازه درنظر گرفته شود. ولی درایه‌های جدول صفحه پیچیده‌تر هستند. قالب کلی یک درایه جدول صفحه در راهبرد صفحه مجازی به‌صورت زیر است:

شماره قاب	بیت‌های کنترلی حفاظت	C	R	M	P	

بیت حضور (P): بیت حضور، که با نام‌های **بیت حضور ـ غیاب** (**Present / Absent (P/A)**) و (In/Out) نیز متداول است، مشخص می‌کند کدام صفحه مجازی به صورت فیزیکی در حافظه است و کدام صفحه حاضر نیست.

به عبارت دیگر اگر این بیت ۱ باشد، یعنی صفحه در حافظه اصلی قرار دارد و اگر صفر باشد، یعنی صفحه مجازی مربوط به این درایه در حافظه حاضر نیست.

دسترسی به درایه‌ای از جدول صفحه که بیت حضور آن ۰ باشد، منجر به یک تله می‌شود. یعنی پردازه سعی کند به یک صفحه مراجعه کند که در حال حاضر بر روی دیسک قرار دارد و در حافظه اصلی حاضر نیست. این تله، **نقص صفحه** یا **Page fault** نام دارد. در بخش بعد به بررسی بیشتر این موضوع خواهیم پرداخت.

بیت تغییر (M): بیت تغییر (Modify bit)، تاریخچه دسترسی به یک صفحه را ثبت می‌کند. اگر مقدار این بیت برابر ۱ باشد، یعنی محتوای صفحه مربوط به این درایه، در حافظه اصلی تغییر کرده است. به عبارت دیگر، زمانی‌که چیزی در یک صفحه نوشته شود، سخت‌افزار به‌طور خودکار، این بیت را با ۱مقداردهی می‌کند.

کاربرد این بیت زمانی است که سیستم عامل تصمیم می‌گیرد یک صفحه را از حافظه اصلی خارج کند و یک صفحه جدیدی را جایگزین آن نماید. لذا از این بیت به صورت زیر کمک می‌گیرد:

اگر در صفحه مربوطه تغییرات صورت گرفته یا اصلاحی انجام شده باشد، یعنی صفحه اصطلاحاً «**کثیف**» (dirty) شده است و سیستم عامل باید محتوای آن را بر روی دیسک بنویسد. اما اگر این بیت صفر باشد، یعنی اصطلاحاً صفحه «**تمیز**» است و دیگر نیازی نیست محتوای صفحه روی دیسک نوشته شود؛ چون صفحه هیچ تغییری نکرده و نسخه موجود روی دیسک هنوز معتبر است.

نکته۱: از آن‌جا که این بیت، وضعیت صفحه را از نظر کثیف یا تمیز بودن مشخص می‌کند، به آن «**بیت کثیف**» نیز گفته می‌شود.

بیت مراجعه (R): بیت **مراجعه** یا **ارجاع** (**Reference**)، نیز تاریخچه دسترسی به یک صفحه را ثبت می‌کند. اگر مقدار این بیت برابر ۱ باشد، یعنی، یک مراجعه برای خواندن یا نوشتن به صفحه مربوطه صورت گرفته است. سیستم عامل بعد از بروز یک خطای صفحه، از این بیت برای تعیین این‌که کدام صفحه کاندیدای مناسب‌تری برای خارج شدن از حافظه است، استفاده می‌کند. این بیت در الگوریتم‌های جایگزینی صفحه که در ادامه همین فصل بررسی می‌کنیم، نقش مهمی را ایفا می‌کند.

بیت نهان‌سازی (C): این بیت که به **حافظه نهان غیرفعال شده**، معروف است، اجازه می‌دهد تا حافظه نهان یک صفحه غیرفعال شود. اگر مقدار این بیت برابر ۱ باشد، یعنی محتوای صفحه مربوطه، قابل بارگذاری در حافظه نهان نیست. این ویژگی برای صفحاتی اهمیت دارد که به جای حافظه به ثبات‌های دستگاه نگاشت می‌شوند. اگر سیستم عامل در انتظار رسیدن پاسخ از یک دستگاه I/O باشد، مهم است که در هر لحظه مقدار واقعی ثبات‌ها را بخواند، نه یک کپی قدیمی در حافظه نهان. با مقدارگذاری این بیت حافظه نهان صفحه غیرفعال می‌شود. بیت غیرفعال کردن حافظه نهان فقط در کامپیوترهایی کاربرد دارد که از روش I/O نگاشت ـ حافظه استفاده می‌کند؛ ماشین‌هایی که دارای فضای آدرس I/O مستقل هستند به این بیت نیازی ندارند.

بیت‌های حفاظت: بیت‌های حفاظت (**Protection**)، انواع دسترسی‌های مجاز را مشخص می‌کنند. این بیت‌ها شامل سه بیت (**Read – only, read / Write, eXecute (RWX)**) هستند. اگر $R = ۱$ باشد یعنی محتوای صفحه مربوطه فقط خواندنی و اگر $w = ۱$ باشد، یعنی محتوای صفحه مربوطه قابل تغییر است (دسترسی خواندن / نوشتن). اگر $x = ۱$ باشد، به معنای این است که صفحه مربوطه، شامل یک کد اجرایی می‌باشد و قابل اجرا است.

شماره قاب (F#): مهم‌ترین فیلد در درایه جدول صفحه، شماره قاب صفحه است و بیانگر شماره قابی از حافظه اصلی می‌باشد که صفحه مربوطه در آن قرار گرفته است.

فصل نهم

« سیستم‌های ورودی / خروجی و دیسک »

مقدمه

یکی از مهم‌ترین وظایف هر سیستم عامل، کنترل دستگاه‌های I/O (ورودی/خروجی) کامپیوتر است.

در مورد ورودی/خروجی، مسأله اصلی کارآیی است. به درستی امکانات ورودی/ خروجی، محل منازعه کارآمدی است. با نگاهی به عملکرد داخل سیستم کامپیوتر، درمی‌یابیم که سرعت پردازنده‌ها هم‌چنان در حال افزایش است و اگر یک پردازنده به اندازه کافی سریع نباشد، پیکربندی‌های چند پردازنده‌ای متقارن موجب تسریع کار می‌شوند. سرعت دستیابی حافظه اصلی نیز (با نرخ کم‌تر از افزایش سرعت پردازنده) در حال افزایش است. در هر حال، با به‌کارگیری هوشمندانه یک، دو یا حتی چند سطح از حافظه‌های نهان داخلی، زمان دستیابی به حافظه اصلی با سرعت پردازنده قابل مقایسه می‌شود؛ اما ورودی/ خروجی هم‌چنان به عنوان یک چالش عمده کارآیی، به‌خصوص در مورد حافظه دیسک، باقی مانده است.

نقش سیستم عامل در ورودی/ خروجی کامپیوتر، مدیریت و کنترل عملیات ورودی/ خروجی و دستگاه‌های ورودی/ خروجی می‌باشد. هر چند موضوعاتی در این رابطه، در فصل اول مطرح شده‌اند، در این‌جا همه را یک‌جا جمع‌آوری کرده و تصویر کاملی از ورودی / خروجی را ترسیم خواهیم کرد. در ابتدا، مبانی و اصول سازماندهی سخت‌افزاری I/O را توصیف می‌کنیم. در ادامه در مورد سرویس‌های ورودی/ خروجی که توسط سیستم عامل فراهم می‌شوند و حضور این سرویس‌ها در رابطه ورودی/ خروجی برنامه کاربردی بحث می‌کنیم.

بخش‌های بعدی این فصل اختصاص به ورودی/ خروجی دیسک مغناطیسی دارد. در سیستم‌های امروزی، دیسک، مهم‌ترین شکل ورودی/ خروجی و کلید کارآیی می‌باشد. با توصیف ساختار فیزیکی دیسک‌های مغناطیسی شروع کرده، سپس، الگوریتم‌های زمان‌بندی دیسک را مطرح می‌کنیم که ترتیب ورودی/ خروجی‌های دیسک را با هدف بهبود کارآیی، زمان‌بندی می‌کند.

درسنامه (۱): سخت‌افزار I/O

کنترل دستگاه‌های متصل به کامپیوتر، یکی از دغدغه‌های اصلی طراحان سیستم عامل محسوب می‌شود. چون دستگاه‌های ورودی/خروجی (ماوس، دیسک، CD-ROM و ...) از نظر عملکرد و سرعت متنوع هستند، از این‌رو برای کنترل آن‌ها روش‌های مختلفی لازم است. این روش‌ها، زیر سیستم I/O مربوط به هسته را تشکیل می‌دهند که بقیه‌ی هسته را از پیچیدگی مدیریت بر دستگاه‌های ورودی/ خروجی تفکیک می‌کند.

فناوری دستگاهI/O دو رفتار متضاد را نشان می‌دهد. از یک سو، افزایش استانداردسازی واسط‌های سخت‌افزاری و نرم‌افزاری را می‌بینیم؛ این رفتار به ما کمک می‌کند تا نسل‌های جدیدی از دستگاه‌ها را در کامپیوترها و سیستم عامل‌های موجود به‌کار گیریم.

از سویی دیگر، شاهد تنوع گسترده و فزاینده‌ای از دستگاه‌های I/O هستیم. بسیاری از دستگاه‌های جدید با دستگاه‌های پیشین متفاوت هستند، به‌طوری که به کارگیری آن‌ها در کامپیوتر و سیستم عامل‌ها، چالش‌برانگیز است. این چالش ناشی از تکنیک‌های سخت‌افزاری و نرم‌افزاری است. عناصر اصلی سخت‌افزار I/O مانند پورت‌ها، گذرگاه‌ها و کنترلر دستگاه‌ها، دامنه‌ی وسیعی از دستگاه‌های ورودی/خروجی را تشکیل می‌دهد. برای بسته‌بندی کردن (پنهان‌سازی) جزئیات و ویژگی‌های منحصر به فرد دستگاه‌های مختلف، هسته سیستم عامل از **درایور (گرداننده)** دستگاه‌ها استفاده می‌کند.

درایور (گرداننده) دستگاه‌ها، واسط یک شکلی را برای زیرسیستم I/O فراهم می‌کنند، همانند فراخوانی‌های سیستمی که واسط استانداردی را بین برنامه‌های کاربردی و سیستم عامل فراهم می‌کنند.

کامپیوترها از دستگاه‌های متنوعی استفاده می‌کنند. بیشتر این دستگاه‌ها در سه دسته کلی زیر قرار می‌گیرند:

۱ـ دستگاه‌های ذخیره‌سازی: مانند دیسک‌ها و نوارها

۲ـ دستگاه‌های انتقال: مانند کارت‌های شبکه و مودم‌ها

۳ـ دستگاه‌های واسط انسان: مانند صفحه نمایش، صفحه کلید و ماوس

علی‌رغم تنوع دستگاه‌های I/O که می‌توانند در کامپیوتر مورد استفاده قرار گیرند، تنها نیاز داریم بدانیم که دستگاه‌ها چگونه به کامپیوتر متصل می‌شوند و نرم‌افزار چگونه سخت‌افزار را کنترل می‌کند.

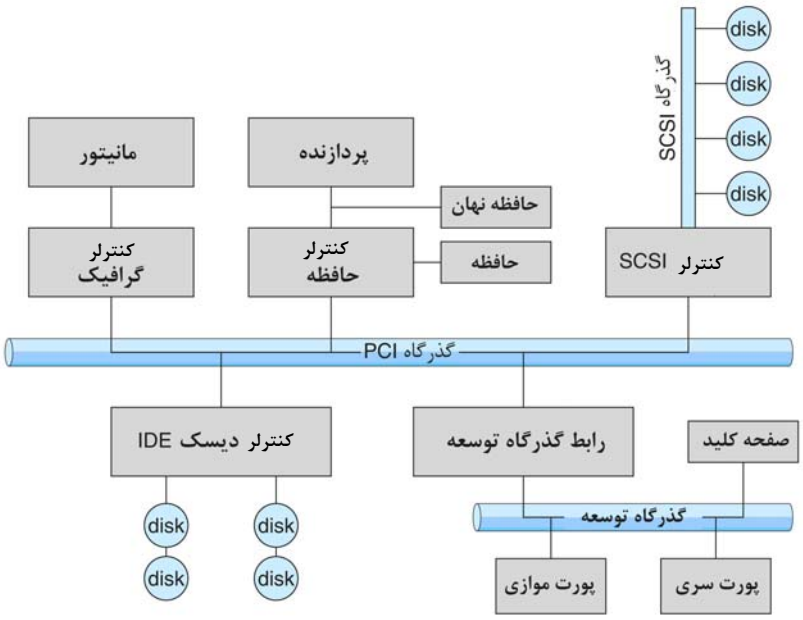
هر دستگاه با ارسال سیگنال‌هایی از طریق یک کابل یا حتی از طریق هوا، با سیستم کامپیوتری ارتباط برقرار می‌کند.

نقطه اتصال هر دستگاه با کامپیوتر، **پورت** (**Port**) نامیده می‌شود. از جمله پورت‌ها می‌توان به پورت سریال یا موازی اشاره نمود. اگر دستگاه‌ها برای ارتباط از مجموعه‌ای از خطوط مشترک استفاده نمایند، این اتصال **گذرگاه** (**Bus**) نامیده می‌شود. در نتیجه گذرگاه، مجموعه‌ای از خطوط و پروتکل‌هایی تعریف شده است که مجموعه‌ای از پیام‌ها را مشخص می‌کند که می‌توانند روی خطوط ارسال شوند.

در تعابیر و مباحث الکترونیک، پیام‌ها با الگوهایی از ولتاژ الکتریکی منتقل می‌شوند که در فاصله زمانی‌های تعریف‌شده‌ای به خطوط اعمال می‌گردد. زمانی‌که دستگاه A از طریق کابلی به دستگاه B متصل می‌شود و دستگاه B هم از طریق کابل به دستگاه C متصل می‌گردد و دستگاه C به پورتی از کامپیوتر متصل می‌شود، این ترتیب، یک **اتصال زنجیره‌ای** (**Daisy Chain**) نامیده می‌شود. یک اتصال زنجیره‌ای، معمولاً به صورت یک گذرگاه عمل می‌کند.

گذرگاه‌ها کاربرد گسترده‌ای در معماری کامپیوتر دارند و تنوع آن‌ها به روش‌های سیگنال‌دهی، سرعت، توان عملیاتی و روش‌های اتصال با هم برمی‌گردد. شکل ۹ـ۱ نمونه‌ای از ساختار گذرگاه کامپیوتر شخصی را نشان می‌دهد. این شکل، یک **گذرگاه PCI** را نشان می‌دهد که زیرسیستم پردازنده ـ حافظه را به دستگاه‌های سریع اتصال می‌دهد و یک **گذرگاه توسعه** (**Expansion Bus**) را نشان می‌دهد که دستگاه‌های نسبتاً کند مانند صفحه کلید و پورت‌های سری و USB را اتصال می‌دهد.

در سمت راست و بالای شکل، چهار دیسک روی یک **گذرگاه SCSI** (Small Computer System Interface) به هم متصل شده‌اند که این گذرگاه به یک کنترلر SCSI متصل است. گذرگاه‌های متداول دیگری برای اتصال داخلی قسمت‌های اصلی یک کامپیوتر به‌کار می‌روند، از جمله: **PCI-X** که توان عملیاتی آن تا ۴/۳GB، **PCI Express** (PCIe) با توان عملیاتی تا ۱۶GB و **Hyper Transport**، با توان عملیاتی تا ۲۰GB است.



شکل ۹ـ۱. نمونه‌ای از ساختار گذرگاه کامپیوتر شخصی

کنترلر، مجموعه‌ای از قطعات الکترونیکی است که می‌تواند بر روی یک پورت، گذرگاه یا دستگاه عمل کند. کنترلر پورت سریال، نمونه‌ای از کنترلر دستگاه ساده است. این کنترلر یک تراشه (یا بخشی از یک تراشه) است که سیگنال‌های موجود در خطوط یک پورت سریال را کنترل می‌کند.

در مقابل، کنترلر گذرگاه SCSI ساده نیست، چون پروتکل SCSI پیچیده است. کنترلر گذرگاه SCSI معمولاً به صورت یک بُرد مداری مجزا (**آداپتور میزبان (Host Adapter)**) پیاده‌سازی می‌شود و به کامپیوتر متصل می‌گردد. این کنترلر حاوی یک پردازنده، ریزکُد و مقداری حافظه اختصاصی است تا بتواند پیام‌های پروتکل SCSI را پردازش کند.

برخی دستگاه‌ها، کنترلر داخلیِ خاصِ خود را دارند: با نگاهی به دیسک گردان، متوجه می‌شوید که یک بُرد مداری، در یک طرف آن متصل است. این بُرد همان کنترلر دیسک است. این کنترلر، جنبه‌ی دیسکِ پروتکل را برای برخی از انواع اتصالات (مانند SCSI و **SATA (Serial Advanced Technology Attachment)**) پیاده‌سازی می‌کند. هم‌چنین دارای ریز کُد و پردازنده‌ای است که کارهایی مانند نگاشت سکتورهای خراب، پیش واکشی، بافرسازی و نهان‌سازی را انجام می‌دهد.

هر دستگاه I/O معمولاً دو بخش دارد:

۱ـ بخش مکانیکی ← کار اصلی را انجام می‌دهد.۲ـ بخش الکترونیکی ← کنترلر دستگاه

سیستم عامل تقریباً همیشه با کنترلر دستگاه سروکار دارد، نه با بخش مکانیکی دستگاه I/O .

هر کنترلر دستگاه، شامل تعدادی حافظه بافر داخلی و مجموعه‌ای از ثبات‌های تک منظوره برای اهداف خاص است.

کنترلر دستگاه مسئول انتقال داده‌ها بین بافرهای داخلی خود و دستگاه‌های جانبی است که تحت کنترل آن است.

I/O نگاشت حافظه

پردازنده چگونه می‌تواند فرمان‌ها و داده‌ها را به کنترلر بدهد تا انتقال I/O را انجام دهد؟ پاسخ کوتاه این است که کنترلر یک یا چند ثبات برای سیگنال‌های آدرس داده و سیگنال‌های کنترل ارتباط دارد. پردازنده از طریق خواندن و نوشتن بیت‌هایی در این ثبات‌ها، با کنترلر ارتباط برقرار می‌کند.

یک روش برقراری این ارتباط، استفاده از دستورات خاص I/O است که انتقال یک بایت یا کلمه را به یک آدرس پورت I/O مشخص می‌کند. دستور I/O، خطوطی از گذرگاه را راه‌اندازی می‌کند تا دستگاه مناسبی را انتخاب کند و بیت‌هایی را در ثبات دستگاه قرار دهد یا از آن خارج کند.

به روشی دیگر، کنترلر دستگاه می‌تواند **I/O نگاشت حافظه** را پشتیبانی کند. در این حالت، ثبات‌های کنترل دستگاه به فضای آدرس پردازنده نگاشت می‌شوند. پردازنده، درخواست I/O را با استفاده از دستورات معمول انتقال داده‌ها انجام می‌دهد تا ثبات‌های کنترل دستگاه بخواند یا بنویسد.

برخی از سیستم‌ها از هر دو تکنیک استفاده می‌کنند. برای نمونه، کامپیوترهای شخصی با استفاده از دستورات I/O، بعضی از دستگاه‌ها را کنترل می‌کنند و با استفاده از I/O نگاشت حافظه، دستگاه‌های دیگری را کنترل می‌نمایند. شکل ۹ـ۲ آدرس‌های پورت‌های معمول I/O مربوط به کامپیوترهای شخصی را نشان می‌دهد. کنترلر گرافیک دارای پورت‌های I/O برای عملیات کنترل پایه است، اما این کنترلر یک ناحیه‌ی بزرگ نگاشت حافظه هم دارد تا محتویات صفحه نمایش را نگهداری نماید. کنترلر، خروجی را در ناحیه‌ی نگاشت حافظه می‌نویسد تا در صفحه نمایش ظاهر گردد. یعنی کنترلر، تصویر صفحه نمایش را بر اساس محتویات این بخش از حافظه تولید می‌نماید. این تکنیک ساده است، به علاوه، نوشتن میلیون‌ها بایت در حافظه گرافیکی سریع‌تر از صدور میلیون‌ها دستور I/O است.

البته سهولت نوشتن در کنترلر I/O نگاشت حافظه یکی از معایب آن را جبران می‌کند: یکی از انواع خطاهای نرم‌افزاری رایج این است که نوشتن ناخواسته‌ای در اثر یک اشاره‌گر نادرست به یک ناحیه از حافظه رخ دهد، از این رو ممکن است ثبات دستگاه نگاشت به حافظه در معرض تغییر ناگهانی قرار گیرد. البته، حافظه حفاظت شده می‌تواند به کاهش احتمال وقوع این خطر کمک کند.

دستگاه	بازه‌ی آدرس I/O (هگزادسیمال)
کنترلر DMA	000- 00F
کنترلر وقفه	020- 021
زمان‌سنج	040- 043
کنترلر بازی	200- 20F
پورت سری (ثانویه)	2F8- 2FF
کنترلر دیسک سخت	320- 32F
پورت موازی	378- 37F
کنترلر گرافیکی	3D0- 3DF
کنترلر گرداننده‌ی دیسکت	3F0- 3F7
پورت سری (اولیه)	3F8- 3FF

شکل ۹-۲. مکان نسبی پورت I/O دستگاه‌ها در کامپیوتر شخصی

هر پورت I/O معمولاً شامل چهار ثبات است: ۱) وضعیت، ۲) کنترلر، ۳) ورود داده‌ها و ۴) خروج داده‌ها.

- **ثبات ورود داده‌ها:** توسط میزبان خوانده می‌شود تا ورودی دریافت گردد.
- **ثبات خروج داده‌ها:** توسط میزبان نوشته می‌شود تا خروجی ارسال گردد.
- **ثبات وضعیت:** حاوی بیت‌هایی است که توسط میزبان خوانده می‌شود. این بیت‌ها نشان‌دهنده‌ی حالت‌ها هستند، از جمله: آیا فرمان جاری تکمیل شده است، آیا بایتی در ثبات ورودی داده برای خواندن مهیا است، و آیا خطایی در دستگاه رخ داده است یا خیر؟
- **ثبات کنترلر:** ثبات کنترلر می‌تواند توسط میزبان نوشته شود تا فرمانی شروع شود یا حالت (مُد) دستگاه تغییر کند. برای نمونه، بیت خاصی در ثبات کنترل یک پورت سریال، ارتباط دو طرفه همزمان (Full-duplex) و یا دو طرفه غیرهمزمان (Half-duplex) را انتخاب می‌کند. بیت دیگری وجود دارد که بررسی توازن (Parity checking) را فعال می‌سازد. بیت سومی هم طول کلمه را برابر ۷ یا ۸ بیت تعیین می‌کند و بیت‌های دیگر، یکی از سرعت‌های پشتیبانی‌شده توسط پورت سریال را انتخاب می‌کنند.

مثال ۱: معیارهای زیر مفروض است. کدام دسته از معیارها از اهداف طراحی امکانات I/O در سیستم‌های کامپیوتری محسوب می‌شوند؟

الف) کارایی (Efficiency) ب) تمامیت (Integrity) ج) انصاف (Fairness) د) عمومیت (Generality)

۱) ب و د ۲) الف و ج ۳) ب و ج ۴) الف و د

☑ **پاسخ:** گزینه «۴» دو هدف کارایی و عمومیت در طراحی امکانات ورودی / خروجی حاکم هستند. یکی از تلاش‌های اصلی در طراحی I/O، بالا بردن کارآیی آن است. هدف اصلی دیگر عمومیت است و مطلوب این است که تمام دستگاه‌ها به صورتی یک شکل و یکنواخت، هم در چگونگی نگرش پردازه‌ها به دستگاه‌های I/O و هم در نحوه اداره دستگاه‌های I/O توسط سیستم عامل به کار روند.

مثال ۲: کدام گزینه در مورد کنترلگر صحیح نمی‌باشد؟

۱) کنترلگر، مجموعه‌ای از قطعات الکترونیکی است که می‌تواند بر روی پورت، گذرگاه یا دستگاه عمل کند.

۲) کنترلگر گذرگاه SCSI نمونه‌ای از کنترلگر دستگاه ساده است.

۳) بعضی از دستگاه‌ها کنترلگر داخلی مخصوص به خود را دارند.

۴) کنترلگر دیسک، یک برد مداری است که در یک طرف آن متصل است.

☑ **پاسخ:** گزینه «۲» کنترل SCSI ساده نیست. چون پروتکل SCSI پیچیده است، کنترلگر گذرگاه SCSI معمولاً به صورت یک برد مداری جداگانه پیاده‌سازی می‌شود که به کامپیوتر متصل می‌گردد. این کنترلگر حاوی یک پردازنده، ریزکد و مقداری حافظه است تا بتواند پیام‌های پروتکل SCSI را پردازش کند.

درسنامه (۲): مدیریت عملیات ورودی / خروجی

سؤالی که در این قسمت مطرح می‌شود این است که برنامه درخواست‌دهنده (میزبان) عملیات ورودی/خروجی، چگونه از طرف دستگاه ورودی/خروجی مطلع گردد که عملیات I/O مورد نظر به پایان رسیده است؟

همان‌طور که در بخش قبل متذکر شدیم، یکی از ثبات‌های کنترلر، ثبات وضعیت است و یکی از حالت‌هایی که این ثبات نشان می‌دهد این است که آیا فرمان جاری (عملیات I/O) به پایان رسیده است یا خیر.

در روند مدیریت انجام ورودی/خروجی سه رویکرد وجود دارد که در ادامه، آن‌ها را بررسی می‌کنیم.

سرکشی (Pooling)

کنترلر، حالت خود را از طریق بیت busy در ثبات وضعیت نشان می‌دهد. زمانی‌که کنترلر مشغول کاری است، بیت busy را انتخاب می‌نماید (نوشتن ۱ در بیت) و زمانی که برای پذیرش فرمان بعدی آمادگی دارد، بیت busy را پاک می‌کند (نوشتن صفر در بیت). حال برای آزمون این بیت و فهمیدن این‌که I/O آمادگی دارد یا خیر، پردازنده می‌تواند از یک حلقه نرم‌افزاری کمک بگیرد. یعنی پردازنده یا **انتظار مشغول** است و یا **در حال سرکشی** (pooling) می‌باشد. به عبارت دیگر در حلقه‌ای قرار دارد که بارها و بارها ثبات وضعیت را می‌خواند تا این که بیت busy پاک شود (برابر با صفر گردد).

این روش با نام **سرکشی (Pooling)** یا **نظرسنجی** مشهور است.

اگر کنترلر و دستگاه I/O، سرعت عمل داشته باشند، این روش یکی از روش‌های معقول است. اما اگر این امکان وجود داشته باشد که انتظار طولانی گردد، پردازنده لازم خواهد داشت که به کار دیگری بپردازد.

در این صورت، پردازنده چگونه بداند که کنترلر چه زمانی کارش تمام شده و آمادگی دارد؟ برای برخی از دستگاه‌ها، پردازنده باید سریعاً به دستگاه سرویس بدهد وگرنه داده‌ها از بین می‌روند. به عنوان مثال، وقتی داده‌ها از صفحه کلید می‌آیند یا در پورت سریال قرار می‌گیرند، در بافر کوچکی قرار داده می‌شوند و اگر پردازنده در خواندن آن‌ها تأخیر زیادی داشته باشد، بافر کنترلر سرریز شده و داده‌ها از بین می‌روند.

روشن است اگر سرکشی (pooling)، به دفعات زیاد صرفاً برای بررسی و آزمون آماده بودن یک دستگاه صورت بگیرد، در حالی‌که سایر پردازش‌های مفید پردازنده رها شوند، ناکارآمد خواهد بود. در چنین مواردی بهتر است ترتیبی اتخاذ گردد تا به جای این‌که پردازنده به صورت مکرر، برای تکمیل I/O به سرکشی بپردازد، کنترلر به محض آمادگی دستگاه، پردازنده را مطلع سازد. بدین ترتیب لازم نیست پردازنده منتظر تکمیل عملیات I/O بماند. راهکار سخت‌افزاری که دستگاه را قادر می‌سازد تا آمادگی خودش را به پردازنده اعلام کند، **وقفه** نام دارد.

نکته۱: روش سرکشی (Pooling) با نام روش **I/O برنامه نویسی شده (PIO)** نیز مشهور است.

وقفه

روش کار راهکار وقفه به این صورت است: سخت‌افزار پردازنده، یک خط به نام **خط درخواست وقفه (Interrupt-Request Line)** دارد که پردازنده بعد از اجرای هر دستور آن را بررسی می‌کند. اگر پردازنده تشخیص بدهد که کنترلری یک سیگنال را روی خط درخواست وقفه قرار داده است، پردازنده حالت فعلی را ذخیره می‌کند و به **روال اداره‌کننده وقفه (Interrupt-Handler Routine)** در آدرس ثابتی از حافظه می‌رود.

روال اداره‌کننده وقفه علت وقفه را تعیین می‌کند، پردازش لازم را انجام می‌دهد، یک بازیابی حالت انجام می‌دهد و دستور برگشت از وقفه را اجرا می‌کند تا پردازنده را به حالت اجرای پیش از وقوع وقفه برگرداند. می‌گوییم کنترلر دستگاه با قرار دادن یک سیگنال روی خط درخواست وقفه، یک وقفه ایجاد می‌کند، پردازنده وقفه را می‌گیرد و آن را به روال اداره‌کننده وقفه می‌فرستد و روال اداره‌کننده وقفه به آن سرویس می‌دهد و وقفه را پاک می‌کند.

این راهکار اساسی وقفه، پردازنده را قادر می‌سازد تا به یک رویداد ناهمگام، در زمان آمادگی کنترلر دستگاه برای سرویس، پاسخ بدهد. با وجود این، در سیستم عامل‌های مدرن، به امکانات حرفه‌ای‌تری از اداره کردن وقفه نیاز داریم:

۱ـ لازم است در اثنای پردازش بحرانی، وقفه به تعویق بیافتد.

۲ـ به یک روش کارآمد نیاز داریم تا وقفه را به اداره‌کننده وقفه مناسبی بفرستیم، به‌طوری‌که در ابتدا نیاز به سرکشی تمام دستگاه‌ها برای تعیین دستگاه صادرکننده وقفه نداشته باشیم.

۳ـ وقفه‌های چندسطحی نیاز داریم، به‌طوری که سیستم عامل بتواند بین وقفه‌های با اولویت بالا و اولویت پایین تمایز قائل شود و بتواند به تناسب درجه فوریت وقفه به آن‌ها پاسخ بدهد.

این سه ویژگی در سخت‌افزار کامپیوتر مدرن، توسط پردازنده و توسط سخت‌افزار کنترلر وقفه (Interrupt-controller hardware) فراهم می‌شود.

اغلب پردازنده‌ها دوخط درخواست وقفه دارند:

۱ـ وقفه پوشش‌ناپذیر (تأخیرناپذیر): آن‌هایی هستند که به رویدادهایی همچون خطاهای غیرقابل ترمیم حافظه، اختصاص دارد.

۲ـ وقفه پوشش‌پذیر (تأخیرپذیر): آن‌هایی هستند که پردازنده می‌تواند آن‌ها را به تأخیر بیندازد، پیش از اجرای توالی دستورات بحرانی که باید بدون وقفه باشند.

راهکار وقفه یک آدرس را می‌پذیرد. این آدرس، عددی است که روال اداره‌کننده وقفه‌ی خاصی را از بین مجموعه کوچکی از روال‌های اداره‌کننده وقفه انتخاب می‌کند. در بیشتر معماری‌های کامپیوتر، این آدرس یک افست در جدولی به نام **بردار وقفه** (**Interrupt Vector**) است.

این بردار، حاوی آدرس‌های روال‌های اداره‌کننده وقفه است. با استفاده از این بردار، روال اداره‌کننده وقفه مورد نظر سریعاً پیدا می‌شود و لازم نیست که تمام منابع ممکن وقفه‌ها جستجو گردد. با وجود این، کامپیوترها در عمل تعداد دستگاه‌های بیش‌تری (و از این‌رو، اداره‌کننده‌های وقفه بیش‌تری) از آن‌چه که آدرس آن‌ها در بردار وقفه هستند، دارند. یک روش مرسوم برای حل این مسأله، استفاده از **زنجیره وقفه** (**Interrupt Chaining**) است که در آن، هر یک از عناصر بردار وقفه به ابتدای لیستی از اداره‌کننده‌های وقفه اشاره می‌کند. زمانی‌که وقفه‌ای ایجاد شد، روال‌های اداره‌کننده وقفه موجود در لیست متناظر با آن، یکی یکی فراخوانی می‌شوند تا این‌که روال مناسبی پیدا شود که بتواند درخواست را سرویس بدهد. این ساختار، توازنی بین سربار ناشی از جدول وقفه بزرگ و ناکارآمدی استفاده از روال اداره‌کننده وقفه مفرد است.

شکل ۹ـ۳ طراحی بردار وقفه را برای پردازنده پنتیوم اینتل نشان می‌دهد. رویدادهای ۰ تا ۳۱ که پوشش‌ناپذیرند برای سیگنال‌دهی شرایط مختلف خطا به کار می‌روند. رویدادهای ۳۲ تا ۲۵۵ که پوشش‌پذیرند، برای اهدافی نظیر وقفه‌های حاصل از دستگاه‌ها به کار می‌روند.

شماره‌ی بردار	توضیحات	شماره‌ی بردار	توضیحات
0	خطای تقسیم	11	قطعه موجود نیست
1	استثنای خطایابی	12	خطای پشته
2	وقفه‌ی تهی	13	حفاظت کلی
3	نقطه‌ی کنترلی	14	خطای صفحه
4	سرریز تشخیص INTO	15	(توسط اینتل رزرو شده است)
5	استثنای بازه‌ی حد	16	خطای ممیز شناور
6	کد عملیاتی نامعتبر	17	بررسی هم‌ترازی (alignment)
7	دستگاه موجود نیست	18	بررسی ماشین
8	خطای مضاعف	19-31	(توسط اینتل رزرو شده است)
9	سرریز قطعه‌ی پردازنده کمکی (رزرو شده)	32-255	وقفه‌های پوشش‌پذیر
10	قطعه‌ی حالت نامعتبر وظیفه		

شکل ۹ـ۳. جدول بردار ـ رویداد پردازنده پنتیوم اینتل

هم‌چنین راهکار وقفه، سیستمی از **سطوح اولویت وقفه** را پیاده‌سازی می‌کند. این راهکار به پردازنده اجازه می‌دهد که بدون غیرفعال کردن تمام وقفه‌ها، وقفه‌های اولویت پایین را به تأخیر بیندازد و وقفه‌ای با اولویت بالا می‌تواند اجرای وقفه‌ای با اولویت پایین را پس بگیرد (قبضه کند).

هر یک از سیستم عامل‌های مدرن به روش‌های گوناگونی با راهکار وقفه برخورد می‌کنند. در زمان راه‌اندازی، سیستم عامل گذرگاه‌های سخت‌افزاری را جستجو می‌کند تا تعیین کند چه دستگاه‌هایی به سیستم متصل هستند و اداره‌کننده‌های وقفه متناظر با آن‌ها را در بردار وقفه نصب می‌کند. در اثنای I/O، کنترلر دستگاه‌های مختلف که آماده سرویس هستند، وقفه‌هایی را صادر می‌کنند. این وقفه‌ها، مشخص می‌کنند که عمل خروجی تکمیل شده است یا داده‌های ورودی آماده هستند یا خطایی پیدا شده است.

هم‌چنین راهکار وقفه، برای پردازش دامنه وسیعی از استثناها به کار می‌رود. از جمله استثناها می‌توان به تقسیم بر صفر، دستیابی به حافظه محافظت شده یا فضای آدرس نامعتبر، یا تلاش برای اجرای دستورات ممتاز در مد کاربر اشاره نمود.

رویدادهایی که منجر به وقفه می‌شوند، ویژگی مشترک دارند: همه آن‌ها از پردازنده درخواست می‌کنند که یک روال فوری و خود مشمول را اجرا کند.

دسترسی مستقیم به حافظه (DMA)

در مواقعی که باید حجم بزرگی از داده‌ها بین دستگاه ورودی/خروجی و حافظه انتقال یابند، به عنوان نمونه برای دستگاهی نظیر دیسک که انتقالات بزرگی دارد، استفاده از یک پردازنده همه‌منظوره‌ی گران‌قیمت، برای مشاهده بیت‌های وضعیت و واکشی بایت به بایت داده‌ها در ثبات کنترلر دستگاه (راهبرد I/O برنامه‌نویسی شده) معقول نیست.

از این رو، بسیاری از کامپیوترها برای کاهش مسئولیت و اجتناب از فشار بر روی پردازنده اصلی، از راهبرد دیگری به نام **دسترسی مستقیم به حافظه** (Direct-Memory-Access(**DMA**)) استفاده می‌کنند؛ و در حقیقت این کارها را به کنترلر دسترسی مستقیم به حافظه واگذار می‌نمایند.

برای شروع انتقال DMA، میزبان یک بلاک فرمان DMA را در حافظه می‌نویسد. این بلاک حاوی اشاره‌گری به منبع انتقال، اشاره‌گری به مقصد انتقال و تعداد بایت‌هایی است که باید انتقال یابند.

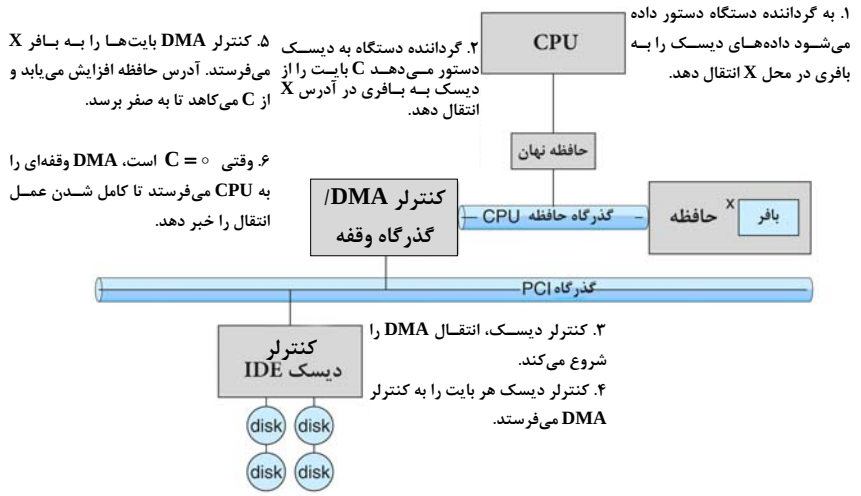
پردازنده، آدرس این بلاک فرمان را در کنترلر DMA می‌نویسد و سپس به سراغ کار دیگری می‌رود. کنترلر DMA مستقیماً بر روی گذرگاه حافظه عمل می‌کند. برای این‌کار آدرس‌ها را در گذرگاه قرار می‌دهد تا بدون نیاز به کمک پردازنده اصلی، انتقال را انجام دهد.

یک کنترلر DMA ساده، قطعه‌ی استانداردی در تمام کامپیوترهای مدرن از تلفن‌های هوشمند گرفته تا کامپیوترهای بزرگ می‌باشد.

ارتباط میان کنترلر DMA و کنترلر دستگاه از طریق یک جفت سیم به نام DMA-request و DMA-acknowledge انجام می‌شود. زمانی‌که یک کلمه از داده‌ها آماده انتقال باشد، کنترلر دستگاه، سیگنالی را روی خط DMA-request می‌گذارد. این سیگنال، موجب می‌شود تا کنترلر DMA، گذرگاه حافظه را به تصرف خودش درآورد؛ آدرس مطلوب را در خطوط آدرس حافظه گذاشته و سیگنالی را روی خط DMA-acknowledge قرار دهد. وقتی کنترلر دستگاه، سیگنال DMA-acknowledge را دریافت می‌کند، یک کلمه از داده‌ها را به حافظه انتقال داده و سیگنال DMA-request را حذف می‌کند.

زمانی‌که کل انتقال پایان یافت، کنترلر DMA وقفه‌ای را به پردازنده می‌فرستد. این فرایند در شکل ۹ـ۴ نشان داده شده است. توجه کنید که وقتی کنترلر DMA، گذرگاه حافظه را تصرف می‌کند، پردازنده موقتاً برای لحظه‌ای از دسترسی به حافظه اصلی محروم می‌شود، هر چند که هم‌چنان می‌تواند به داده‌های موجود در حافظه نهان اولیه و ثانویه خود دستیابی داشته باشد.

به طور کلی، واگذاری مسئولیت انتقال داده‌ها به DMA، موجب افزایش کارایی سیستم می‌شود.



شکل ۹ـ۴. مراحل انتقال DMA

مثال ۳: اگر کرنل (Kernel) سیستم عامل شامل دو بخش وابسته به ماشین و مستقل از ماشین باشد، کدام گزینه دسته‌بندی مناسب‌تری از توابع کرنل است؟ (مهندسی کامپیوتر ـ سراسری ۸۱)

۱) گرداننده وقفه (Interrupts) و زمان‌بندی فرآیندها (process) در بخش وابسته به ماشین و درایور (driver) دستگاه‌ها و مدیریت فرآیندها در بخش مستقل از ماشین است.

۲) گرداننده وقفه و درایور دستگاه‌ها در بخش وابسته به ماشین و مدیریت و زمان‌بندی فرآیندها در بخش مستقل از ماشین است.

۳) مدیریت فرآیندها و زمان‌بندی فرآیندها در بخش وابسته به ماشین و گرداننده وقفه‌ها و درایور دستگاه‌ها در بخش مستقل از ماشین است.

۴) درایور دستگاه‌ها و زمان‌بندی فرآیندها در بخش وابسته به ماشین و گرداننده وقفه و مدیریت فرآیندها در بخش مستقل از ماشین است.

☑ **پاسخ:** گزینه «۲» گرداننده وقفه و درایور دستگاه‌ها جزو بخش وابسته به ماشین هستند و تمام زمان‌بندها و مدیریت پردازه، حافظه، فایل و I/O جزو بخش مستقل از ماشین هستند.

مثال ۴: کدام یک از وظایف نرم‌افزار I/O مستقل از دستگاه نیست؟ (مهندسی کامپیوتر ـ آزاد ۸۶)

۱) نامگذاری ۲) اداره بن‌بست ۳) بافر کردن ۴) گزارش خطا

☑ **پاسخ:** گزینه «۲» اداره بن‌بست، از وظایف مدیریت پردازه است و جزء مدیریت I/O قرار ندارد.

نام‌گذاری، حفاظت دستگاه‌های I/O و بافر کردن و گزارش خطا، از وظایف نرم‌افزار I/O مستقل از دستگاه می‌باشد.

مثال ۵: کدام گزینه از وظایف سیستم عامل در رابطه با مدیریت سیستم I/O نیست؟ (مهندسی فناوری اطلاعات IT - آزاد ۸۶)

۱) مدیریت Spooling ۲) ایجاد و حذف فایل‌ها ۳) مدیریت بافرینگ ۴) مدیریت درایوها

☑ **پاسخ:** گزینه «۲» وظیفه ایجاد و حذف فایل‌ها، جزو وظایف سیستم عامل در رابطه با مدیریت فایل است.

درسنامه (۳): نرم‌افزار I/O

کلیدی‌ترین مفهوم در اصول نرم‌افزار I/O، «مستقل از دستگاه» است. مستقل از دستگاه یعنی برنامه کاربردی برای دسترسی به یک دستگاه ورودی/خروجی نیازی به تعیین پیشاپیش نوع آن وسیله نداشته باشد. برای مثال، یک برنامه کاربردی بتواند فایلی را از یک دیسک بدون خدشه بر سیستم عامل باز کند بی‌آن‌که شناختی از نوع دیسک و چگونگی افزودن دیسک‌های جدید و سایر دستگاه‌ها به یک کامپیوتر داشته باشد.

در حقیقت، تنوع گسترده‌ی دستگاه‌های ورودی/خروجی موجود، پیاده‌سازان سیستم عامل را با یک مسأله مواجه می‌کند. هر نوع دستگاه، مجموعه‌ای از قابلیت‌ها، تعاریف بیت کنترلی و پروتکل‌های تعامل با میزبان خاص خود را دارند که همگی با هم متفاوت‌اند. سیستم عامل را چگونه می‌توان طراحی کرد تا بتوانیم دستگاه‌های جدید را بدون بازنویسی سیستم عامل به کامپیوتر متصل کنیم؟ وقتی که دستگاه‌ها تا این اندازه تنوع دارند، چگونه یک سیستم عامل می‌تواند یک واسط آسان و یک شکل برای برنامه‌های کاربردی فراهم کند؟ در ادامه این پرسش‌ها را پاسخ خواهیم داد.

در این بخش، سازماندهی تکنیک‌ها و واسط‌هایی را برای سیستم عامل بررسی می‌کنیم که سیستم عامل را قادر می‌سازد با دستگاه‌های I/O به شیوه‌ای استاندارد و یک شکل برخورد کند.

همانند سایر مسائل پیچیده مهندسی نرم‌افزار، در این‌جا نیز از رهیافتی استفاده می‌شود که شامل **تجرید (Abstraction)**، **بسته‌بندی (Encapsulation)** و **لایه‌گذاری (Layering)** نرم‌افزار می‌باشد.

می‌توانیم جزئیات تفاوت‌های دستگاه‌های I/O را با شناسایی تعداد معدودی از انواع کلی، تجرید (انتزاعی) کنیم.

هر یک از انواع کلی، از طریق مجموعه‌ای از توابع استاندارد به نام **واسط** دستیابی می‌شوند.

تمام جزئیات تفاوت‌های دستگاه‌های ورودی/خروجی (I/O)، در ماژول‌هایی از هسته، به نام درایور دستگاه‌ها بسته‌بندی (پنهان) می‌شوند.

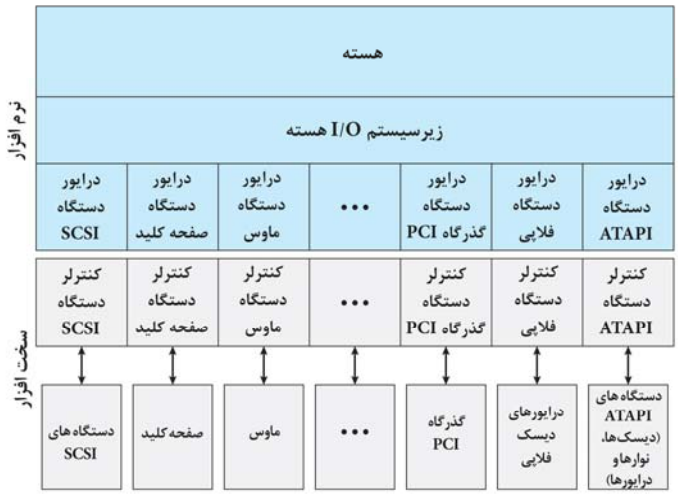
درایور دستگاه‌ها از نظر داخلی مناسب با هر دستگاه خاص منظوره است، اما یک واسطِ استاندارد را عرضه می‌کند.

شکل ۹ـ۵، سازماندهی و ارتباط لایه‌های سخت‌افزار و نرم‌افزار I/O را نشان می‌دهد. هدف لایه درایور دستگاه، مخفی کردن تفاوت‌های بین کنترلر دستگاه از زیر سیستم I/O هسته است.

فراخوان‌های سیستم I/O نیز رفتار دستگاه‌ها را در کلاس‌های کلی بسته‌بندی می‌کنند تا تفاوت‌های سخت‌افزاری را از برنامه‌های کاربردی پنهان نمایند.

مثال ۶: تفکیک و مستقل ساختن نرم‌افزار I/O از سخت‌افزار I/O چه مزیتی دارد؟

☑ **پاسخ:** کار طراحی و ایجاد سیستم عامل را ساده می‌کند. به علاوه به نفع تولیدکنندگان سخت‌افزار نیز هست. آن‌ها یا دستگاه‌های جدیدی را طراحی می‌کنند تا با واسط کنترلر میزبان موجود (مانند SCSI-2) سازگار باشد، یا درایور دستگاه‌ها را برای ارتباط سخت‌افزار جدید با سیستم عامل‌های معروف و رایج می‌نویسند. بنابراین، بدون آن‌که به انتظار نوشتن کد پشتیبان فروشندگان سیستم عامل بمانیم، می‌توانیم دستگاه‌های جانبی جدیدی به کامپیوتر متصل کنیم.



شکل ۹ـ۵. ارتباط لایه‌های سخت‌افزار و نرم‌افزار I/O

متأسفانه هر یک از انواع سیستم عامل‌ها، برای سازندگان سخت‌افزار دستگاه‌ها، استاندارد خاص خودشان را برای واسط درایور دستگاه دارند. یک دستگاه مشخص ممکن است با چندین درایور دستگاه به بازار عرضه شود. به عنوان نمونه، درایورهایی برای ویندوز ۹۵/۹۸، ویندوز NT یا ۲۰۰۰، ویندوز ویستا یا ۷ و سولاریس عرضه می‌گردد.