گزارش بخش دوم فاز دوم پروژه سیستم عامل

سید سروش هاشمی

الناز مهرزاده

بهار سلامتیان

۲۱ تیر ۱۳۹۶

۱ نحوه اجرا

برای اجرا کافی است فایل MotherBoard.java در hardware را کامپایل و اجرا کنید.

۲ معماری نرم‌افزار

این نرم‌افزار حاوی سه package اصلی است. یک package مربوط به سخت‌افزار است که شامل کلاس‌هایی مانند CPU، Core، ALU و MotherBoard است. package دیگر مربوط به سیستم عامل است. در این package کلاس‌های مربوط به فعالیت‌های سیستم عامل از جمله OS، Process، Thread، Scheduler و Channel قرار دارند. package سوم مربوط به Compiler است.

در ادامه توضیح مختصری درباره روند کار نرم‌افزار می‌دهیم.

۱.۲ روند بالا آمدن سیستم عامل، اجرای پردازه‌ها و خاموش شدن سیستم عامل

ابتدا یک شی از کلاس MotherBoard ساخته می‌شود. MotherBoard تمام سخت‌افزارهای لازم را new می‌کند. مثلا یک CPU می‌سازد، یک Screen برای نمایش خروجی‌ها به کاربر می‌سازد، و در فازهای بعدی MainMemory و SecondaryMemory خواهد ساخت. بعد از ساخت سخت‌افزارها، اقدام به boot کردن سیستم‌عامل می‌کند. در فرایند boot کردن، سخت‌افزارهایی که سیستم‌عامل برای انجام فعالیت‌های خود به آن‌ها نیاز دارد را به کلاس OS معرفی می‌کند. مثلا CPU و Screen را به OS معرفی می‌کند و درفازهای آینده MainMemory و SecondaryMemory را معرفی خواهد کرد. بعد از boot کردن سیستم‌عامل، CPU شروع به فعالیت می‌کند. چون امکان استفاده از Thread را نداریم، CPU بین تک تک Core ها گردش می‌کند و به هر کدام می‌گوید که یک instruction اجرا کنند. به طور دقیق‌تر، وقتی MotherBoard عمل boot کردن سیستم‌عامل را تمام می‌کند، تابع run در CPU را فراخوانی می‌کند. این تابع بین تمام Core ها گردش می‌کند و از هر کدام یک instruction اجرا می‌کند و در پایان هر یک دور گردش، از سیستم‌عامل سوال می‌کند که آیا قصد خاموش شدن دارد یا خیر. در صورتی که سیستم‌عامل قصد خاموش شدن نداشته باشد، CPU به کارش ادامه می‌دهد. در غیر این صورت، CPU متوقف می‌شود.

هر Core یک Thread و یک timer دارد. وقتی یک Core قرار است یک instruction اجرا کند، از thread خود می‌خواهد که instruction بعدی را بدهد. اگر thread دیگر هیچ instrcution نداشت، یا در حالت انتظار بود، یا terminate شده بود، یا timer به صدا در آمده باشد یا core هیچ thread نداشت، به سیستم‌عامل interrupt می‌دهد و سیستم‌عامل در صورتی که thread برای اجرا داشت، آن را به core می‌دهد. به طور مثال اگر ۸ core داشته باشیم، در اول کار فقط یکی از آن‌ها مشغول فعالیت است و بقیه هیچ thread ندارند تا زمانی که یک thread دیگر ساخته شود. نتیجه تمام این interrupt ها عوض شدن thread در core است که سیستم‌عامل آن را با کمک Scheduler انجام می‌دهد.

حال فرض کنید core می‌خواهد یک instruction را اجرا کند. این instruction ممکن است System Call باشد یا نباشد. در حالت اول Core به سیستم‌عامل مراجعه می‌کند و در حالت دوم خودش دستور را اجرا می‌کند.

بنابراین در کل ۲ نوع پیام بین Core و سیستم‌عامل رد و بدل می‌شود. یک نوع interrupt ها هستند و نوع دیگر System Call ها.

وقتی می‌خواهیم یک program را اجرا کنیم، ابتدا آن را کامپایل می‌کنیم. برای این کار از package مربوط به compile استفاده می‌کنیم. compiler ما فایل را می‌خواند و همان طور که در فاز قبلی توضیح دادیم، تمام دستورات شرطی و حلقه‌ها را به تعدادی conditional jump تبدیل می‌کند. به طور دقیق‌تر در ابتدا، کامپایلر فایل را parse می‌کند و دستورات را از هم جدا می‌کند و شرط‌ها و حلقه‌ها را به conditional jump ها تبدیل می‌کند. سپس کامپایلر از دستورات parse شده، instruction هایی که برای سخت‌افزار قابل فهم است می‌سازد. یک کلاس abstract به نام instruction در package سخت افزار قرار دارد. سخت‌افزار فقط و فقط دستورات موجود در پوشه hardware.cpu.instruction را پشتیبانی می‌کند. بنابراین کامپایلر از دستورات parse شده یک دنباله از اشیاء از نوع instruction می‌سازد که قرار است Core آن‌ها را اجرا کند.

وظیفه package سیستم‌عامل پاسخ‌دادن به interrupt ها و System Call هاست. همچنین وظیفه مدیریت سخت‌افزارها را به عهده دارد. در package سیستم‌عامل یک package به نام Scheduler وجود دارد که وظیفه کلاس‌های موجود در آن، Schedule کردن thread هاست. همچنین یک package به نام ScreenDriver وجود دارد که وظیفه آن مدیریت سخت‌افزار Screen است. در آینده نیز package های MainMemoryManager و FileSystem به آن اضافه خواهد شد. همچنین در حال حاضر یک package به نام ipc در package سیستم‌عامل وجود دارد که فعالیت‌های مربوط به ارتباط بین process ها را انجام می‌دهد.

این معماری طوری طراحی شده که نسبت به تغییرات احتمالی فازهای بعدی منعطف باشد. همچنین سادگی و طبیعی بودن آن موجب کم شدن تعداد باگ‌های منطقی می‌شود. به طور مثال فرض کنید می‌خواهیم یک دستور به زبان برنامه نویسی معرفی شده در تعریف پروژه اضافه کنیم. برای ساده شدن کامپایلر به جای شکستون دستور جدید به دستورات قدیمی(در صورت امکان)، یک instruction جدید در سخت‌افزار تعریف می‌کنیم و کامپایلر را تغییر می‌دهیم تا بتواند آن دستور جدید را نیز تشخیص داده و instruction مربوط به آن را بسازد. سپس اگر این دستور نیاز به System Call داشت کافی‌است System Call مورد نیاز را در کد سیستم‌عامل پیاده سازی کنیم. با انجام همین چند کار، یک دستور به زبان برنامه نویسی اضافه می‌شود.

۳ کلاس‌های مربوط به این فاز

کلاس‌های مربوط به این فاز عبارت اند از:

- کلاس SecondaryMemory: این کلاس وظیفه نگهداری داده‌های موجود در secondary memory را دارد. این کلاس فقط می‌تواند read و write انجام دهد و کار دیگری از آن ساخته نیست. در package سخت‌افزار

- کلاس SecondaryMemoryDriver در os.secondarymemory: این کلاس مسئولیت برقراری ارتباط با سخت‌افزار را دارد. در تمام سیستم عامل، هر چیزی که بخواهد به طور مستقیم و بدون هیچ واسطه‌ای با سخت افزار secondary memory ارتباط برقرار کند، باید از این کلاس استفاده کند.

- کلاس BasicSecondaryMemoryManager در os.secondarymemory: این کلاس وظیفه خواندن و نوشتن به صورت بلوکی در secondary memory را دارد. این کلاس چیزی درباره فایل‌ها نمی‌داند و فقط داده‌ها را به صورت بلوکی می‌خواند و می‌نویسد.

- کلاس FileManager در os.secondarymemory: این کلاس تمام مدیریت مربوط به فایل‌ها را انجام می‌دهد. از جمله ساخت فایل، خواندن فایل، نوشتن در فایل، پاک کردن فایل و ... . این کلاس برای تمام functionality های خود از کلاس BasicSecondaryMemoryManager استفاده می‌کند.