فرهاد امان 9931006

-1

الف) device controller بخشی از یک IO device است و شامل یک local buffer و تعدادی register است. این device controller و CPU از طریق یک bus مشترک به هم و به حافظه اصلی متصل‌اند. Driver بخشی از سیستم عامل است که مختص به هر device controller طراحی شده است. و درواقع driver راه ارتباطی و زبان مشترک سیستم عامل و device controller می‌باشد. CPU و device controller می‌توانند به صورت موازی و همزمان با هم کار کنند هنگام شروع یک عملیات IO ابتدا driver رجیسترهای مربوط به device controller را مقداردهی می‌کند این مقادیر نشان دهنده یک دستور خاص هستند. به عنوان مثال خواندن بخشی از اطلاعات دیسک. سپس device controller شروع به خواندن اطلاعات دیسک و ذخیره آن‌ها در local buffer می‌کند در این مدت CPU در حال اجرای دستورات سیستم عامل به صورت موازی است. پس از پایان عملیات خواندن device controller از طریق فرستادن یک interrupt به CPU به driver اعلام می‌کند که کار خواندن اطلاعات و ذخیره آن‌ها در local buffer به اتمام رسیده است. در نهایت سیستم دستورات مربوط به استفاده از اطلاعات مانند انتقال آن‌ها از local buffer به main memory را از طریق CPU انجام می‌دهد.
ب) سیستم وقفه باید دارای مکانیزمی برای به تاخیر انداختن وقفه‌های کم اهمیت هنگام انجام یک عمل بسیار critical باشد. به همین خاطر CPU دارای دو خط درخواست وقفه هستند یکی برای وقفه‌های nonmaskable و دیگری برای وقفه‌های maskable. وقفه‌های maskable دارای اهمیت کمتری هستند به طور مثال وقفه‌های ارسال شده توسط device controller ها. CPU می‌تواند هنگام اجرای یک پردازش critical وقفه‌های maskble را خاموش کرده وبه آن‌ها پاسخی ندهد. یکی دیگر از مشکلات سیستم وقفه این است که وقفه‌ها باید به سرعت بسیار بالا پردازش شوند. برای حل این مشکل هر المان داخل بردار وقفه‌ها به لیستی از interrupt handler ها اشاره می‌کند هنگامی که یک وقفه پیش می‌آید این لیست به ترتیب اجرا می‌شوند تا اینکه یک interrupt handler توانایی حل مشکل را داشته باشد. همچنین مکانیزم وقفه‌ها دارای سیستمی برای اولویت بندی وقفه‌ها می‌باشد. با این روش بین وقفه‌ها اولویت وجود دارد و هرکدام به ترتیب اولویت پردازش می‌شوند.

-2

الف) اولین علت قابلیت portability برنامه هنگام استفاده از API است. سیستم های مختلف دارای system call های مختص به خود هستند. استفاده مستقیم از system call های یک سیستم در برنامه باعث می‌شود که برنامه تنها امکان اجرا در همان سیستم را داشته باشد. از طرفی در صورت استفاده از API ها می‌توانیم انتظار داشته باشیم که برنامه ما در هر سیستمی که آن API را پشتیبانی می‌کند اجرا شود. دلیل بعدی سادگی استفاده از API هاست. system call ها با اینکه ارتباط تنگاتنگی با API

های مربوط به خود دارند معمولا استفاده از آن‌ها بسیار پیچیده‌تر و دارای جزئیات بیشتری است. به طور مثال هنگامی که از یک API ساده استفاده می‌کنیم امکان دارد ده ها system call صورت بگیرد که هرکدام پیچیدگی خاص خود را دارد.
ب) این اتفاق توسط runtime environment یا RTE صورت می‌گیرد که شامل یک system call interface است. این system call interface فراخوانی‌های تابع داخل API را رهگیری می‌کند و system call های مربوط به آن را از داخل سیستم عامل فراخوانی می‌کند. هر system call دارای عدد مربوط به خود است و system call interface دارای جدولی شامل این اعداد و system call مربوط به هرکدام است. system call interface پس از اجرای system call وضعیت انجام شدن آن‌را برمی‌گرداند. کسی که یک system call را صدا می‌زند هیچ نیازی به دانستن جزئیات مربوط به اجرا شدن آن system call ندارد کافی است که از قوانین API مربوطه تبعیت کند.
پ) هر system call دارای پارامترهای مختص به خود است. به طور کلی ۳ روش برای انتقال این پارامترها به سیستم عامل وجود دارد. در روش اول که ساده‌ترین روش هم هست تمام پارامترها در register ها ذخیره می‌شوند. ضعف این روش در این است که در بسیاری از اوقات تعداد و حجم پارامترهای ما بیشتر از register ها است. روش دوم ذخیره پارامترها در یک block در حافظه‌ی اصلی است و سپس آدرسی که به این block اشاره می‌کند به عنوان پارامتر در register ها قرار می‌گیرد. روش سوم استفاده از stack است پارامترها توسط برنامه در stack push می‌شوند و سپس توسط سیستم عامل از stack pop می‌شوند. به طور کلی در روش دوم و سوم ما محدودیت بسیار کم‌تری در انتقال پارامترها داریم.

3- فراخوانی سیستمی sys_write مقدار حداکثر count بایت را از ابتدای مکان buf خوانده و در فایلی که fd یا file descriptor به آن اشاره می‌کند می‌ریزد. count از نوع size_t و buf از نوع const void * و fd از نوع unsigned int است. این فراخوانی سیستمی در جواب تعداد بایت‌های نوشته شده را برمی‌گرداند.
فراخوانی سیستمی sys_close تنها یک fd از نوع unsigned int می‌گیرد و دسترسی به فایل مورد نظر را می‌بندد. در نتیجه از آن به بعد fd به آن فایل اشاره نمی‌کند. در جواب در صورت موفقیت آمیز بودن 0 و در غیر این صورت 1- برمی‌گرداند.
فراخوانی سیستمی sys_getpid هیچ پارامتری ندارد و در جواب PID پردازه‌ای که آن را صدا زده است را برمی‌گرداند.
فراخوانی سیستمی sys_sysinfo یک info از نوع struct sysinfo * می‌گیرد و این استراکت را با بعضی از آمار و مشخصات فعلی سیستم پر می‌کند. در جواب در صورت موفقیت آمیز بودن کار عدد 0 و در غیر این صورت 1- برمی‌گرداند.

```
struct sysinfo {
    long uptime;             /* Seconds since boot */
    unsigned long loads[3];  /* 1, 5, and 15 minute load averages */
    unsigned long totalram;  /* Total usable main memory size */
    unsigned long freeram;   /* Available memory size */
    unsigned long sharedram; /* Amount of shared memory */
    unsigned long bufferram; /* Memory used by buffers */
    unsigned long totalswap; /* Total swap space size */
    unsigned long freeswap;  /* Swap space still available */
    unsigned short procs;    /* Number of current processes */
    unsigned long totalhigh; /* Total high memory size */
    unsigned long freehigh;  /* Available high memory size */
    unsigned int mem_unit;   /* Memory unit size in bytes */
    char _f[20-2*sizeof(long)-sizeof(int)];
                             /* Padding to 64 bytes */
};
```

4- در ابتدا که سیستم را روشن می‌کنیم برنامه bootstrap از روی حافظه‌ی ROM خوانده می‌شود به عنوان مثال BIOS این برنامه یک firmware است که ابتدا سلامتی سیستم را بررسی می‌کند و سپس شروع به خواندن بخشی از هارددیسک به اسم MBR می‌کند این بخش شامل برنامه‌ای با نام bootloader است. bootloader وظیفه دارد که از میان پارتیشن‌های موجود در هارد پارتیشن‌های قابل بوت شدن را پیدا کند و اجازه انتخاب میان آن‌ها را به ما می‌دهد سپس آن پارتیشن را بوت کرده و کنترل سیستم را به سیستم عامل بالا آمده می‌دهد به عنوان مثال اگر سیستم عامل ویندوز و لینوکس در دو پارتیشن مختلف هارد نصب شده باشند. در MBR ما bootloader ای با نام GRUB وجود دارد هنگامی که BIOS این GRUB را اجرا کرد GRUB در میان پارتیشن‌های موجود دو پارتیشن یکی دارای سیستم عامل لینوکس و دیگری دارای سیستم عامل ویندوز را پیدا می‌کند و سپس اجازه‌ی انتخاب بین این دو پارتیشن را به ما می‌دهد.

5- به طور کلی ۳ روش برای انجام این کار وجود دارد. اولین روش نوشتن برنامه با استفاده از یک زبان مفسری مانند پایتون است. زبان‌های مفسری برای هر سیستم عامل مفسر مربوطه را دارند که کد را خط به خط خوانده و باتوجه به محیطی که در آن اجرا می‌شود آن‌را تفسیر و اجرا می‌کند.
روش دوم نوشتن برنامه توسط زبانی است که دارای ماشین مجازی می‌باشد مانند جاوا. این ماشین مجازی در تعداد زیادی از سیستم عامل‌ها پیاده سازی شده است در نتیجه برنامه در محیط‌های مختلف قابل اجرا است.
روش سوم استفاده از زبان یا API استاندارد است. در این روش برنامه باید برای هر سیستم عامل port شود این کار معمولا زمان بر خواهد بود و باید برای هر ورژن از برنامه انجام شود.

6- توجه کنید که هنگام انجام عملیات fork یک پردازه‌ی کاملا جدید ایجاد می‌شود در نتیجه دارای heap و stack مختص و مجزای خود است اگرچه که در ابتدا این اطلاعات کپی شده اطلاعات پردازه والد می‌باشند اما کاملا مجزا از هم هستند.
اما هر دو پردازه والد و فرزند به حافظه‌ی مشترک دسترسی دارند از آنجا که حافظه‌ی مشترک بخشی از حافظه‌ی پردازه محسوب نمی‌شود و درواقع داخل کد و با استفاده از مشخصه یکتا دسترسی به آن صورت می‌گیرد این دسترسی در پردازه فرزند هم حفظ خواهد شد.