به نام خدا

سیستم‌های عامل (پاییز ۱۴۰۱)

پاسخ‌نامه تمرین اول

استاد درس:

دکتر جوادی

طراحی وجمع‌آوری سوالات:

خانم ها سروقد و میرزازاده و آقای پولاد

۱- به سوالات زیر در مورد دستگاه های ورودی و خروجی و نحوه‌ی انتقال اطلاعات از آنها به پردازنده و برعکس پاسخ دهید.

الف) وظیفه‌ی کنترلر و درایور دستگاه ها چیست؟ تعامل این دو قسمت با یکدیگر و با دستگاه مربوط به خودشان از آغاز تا پایان یک عملیات I/O به چه صورت است؟

هر دیوایس کنترلر مسئول یک دستگاه خاص است (برای مثال یک صفحه نمایش گرافیکی) و بر اساس نوع کنترلر بیش از یک دستگاه میتوان به آن متصل نمود.
دیوایس کنترلر یک بافر ذخیره‌سازی محلی و تعدادی رجیستر خاص منظوره را نگهداری میکند و مسئول جابجایی داده بین دستگاه‌های جانبی‌ای که کنترل میکند و بافر ذخیره‌سازی است.
معمولا سیستم عامل‌ها به ازای هر دیوایس کنترلر یک دیوایس درایور دارند. این درایور دستگاه در حقیقت دیوایس کنترلر را میفهمد و یک رابط یکپارچه بین بقیه سیستم عامل و دستگاه ایجاد میکند. سی‌پی‌یو و دیوایس کنترلرها میتوانند به صورت موازی فعالیت کنند.
یک عملیات ساده I/O را درنظر بگیرید: برای شروع یک عملیات I/O ، دیوایس درایور رجیسترهای مناسب را در دیوایس کنترلر لود میکند. دیوایس کنترلر در عوض محتوای این رجیسترها را میسنجد تا بفهمد چه اقدامی باید انجام دهد(مثلا یک کاراکتر را از کیبورد بخواند). دیوایس کنترلر سپس انتقال داده از دستگاه به بافر محلی خود را آغاز میکند. دیوایس کنترلر پس از پایان این عملیات انتقال داده، دیوایس درایور را از پایان یافتن عملیات خود مطلع می سازد و این کار را با کمک وقفه‌ها انجام میدهد. حال دیوایس درایور کنترل را به بخش‌های دیگر سیستم عامل میدهد و شاید دیتا یا اشاره‌گر به دیتا را(اگر عملیات خواندن باشد) را برمیگرداند. در سایر عملیات‌ها دیوایس درایور اطلاعات وضعیت نظیر "عملیات نوشتن با موفقیت انجام شد" یا "driver busy" را برمیگرداند.

ب) می‌دانیم یک روش انتقال داده بین دستگاه های ورودی و خروجی و پردازنده، مبتنی بر وقفه هاست. عیب این روش چیست و چگونه در سیستم های کامپیوتری امروزی رفع شده؟

این روش برای انتقال داده‌های با حجم بالا دارای سربار زیادی است و برای رفع این مشکل از DMA یا direct memory access استفاده میکنیم. در حقیقت بعد از تنظیم بافرها، اشاره‌گرها و شمارنده‌ها برای دستگاه‌های ورودی خروجی، دیوایس کنترلر تمام بلوک دیتا را به طور یکجا و بدون مداخله سی‌پی‌یو از دستگاه به حافظه اصلی میفرستد(یا برعکس). در این حالت به جای یک وقفه به ازای هر بایت، تنها یک وقفه در هر کلاک تولید میشود تا به دیوایس درایور اطلاع دهد که عملیات پایان یافته است. در این روش زمانی که دیوایس کنترلر در حال انجام این عملیات‌هاست، پردازنده قادر است به کارهای دیگر بپردازد.

۲ - در مورد فراخوان های سیستمی و API ها به موارد زیر پاسخ دهید:

الف) توضیح دهید چگونه استفاده از API ها به جای فراخوانی مستقیم فراخوان های سیستمی به برنامه‌نویسان کمک می‌کند؟

این api های برای سادگی ارتباط با os طراحی شده اند و استفاده از آن ها روند برنامه نویسی را نسبت به اینکه مستقیما با system call ها کار کنند بسیار راحت تر می کند. همچنین مهم ترین دلیل استفاده از آن ها این است که اگر یک برنامه با تکیه بر این api ها نوشته شود و کامپایل و اجرا شود و تغییراتی در system call ها ایجاد شود، این برنامه همچنان می تواند اجرا شود و نیازی به تغییر نخواهد داشت.(portability)

ب) میدانیم هنگام استفاده از توابع API های مختلف سیستم عامل در برنامه‌هایمان، در حقیقت بعضی فراخوان های سیستمی استفاده می‌شوند. توضیح دهید این تبدیل فراخوانی های API به فراخوانی فراخوان های سیستمی چگونه شکل می‌گیرد؟

با کمک system call interface که یک لینک به فراخوانی‌های سیستمی فراهم شده توسط سیستم عامل ایجاد میکند. واسط سیستم کال در واقع فراخوانی‌های تابعی را در API ها رهگیری کرده و فراخوانی سیستم مربوط در سیستم عامل را فرامیخواند. معمولا به هر سیستم‌کال یک شماره تخصیص داده می شود و واسط سیستم‌کال یک جدول اندیس‌گذاری شده بر اساس این شماره‌ها را نگه‌میدارد. واسط سیستم‌کال سپس فراخوانی سیستم مورد نظر در هسته سیستم عامل را فراخوانی می کند و وضعیت فراخوانی سیستم را برمی گرداند.

پ) روش های کلی انتقال پارامترها به سیستم عامل هنگام استفاده از فراخوان های سیستمی را توضیح دهید.

به طور کلی سه روش برای انتقال پارامترها به سیستم عامل هنگام استفاده از سیستم‌کال‌ها وجود دارد. ساده‌ترین روش ارسال پارامترها به رجیسترها است. اما گاهی ممکن است تعداد پارامترها از تعداد رجیسترها بیشتر شود. در چنین حالتی پارامترها معمولا در یک بلاک یا جدول در حافظه ذخیره میشوند و آدرس محل ذخیره به عنوان یک پارامتر در یک رجیستر ذخیره میشود. اگرچه لینوکس از ترکیبی از این دو روش استفاده میکند. اگر ۵ یا تعداد کمتری پارامتر وجود داشته باشد از رجیسترها و در غیر این صورت از روش بلاکی استفاده میکند. پارامترها همچنین می توانند توسط برنامه در پشته قرار گیرند(push) و توسط سیستم عامل از پشته برداشته شوند(pop). برخی سیستم عامل ها روش بلاکی را ترجیح میدهند زیرا این روش طول و تعداد پارامترهای پاس داده شده را محدود نمیکند.

۳- در مورد فراخوان های سیستمی زیر در سیستم عامل لینوکس ۶۴ بیتی با معماری x86 تحقیق کنید و بنویسید که روش ارسال پارامتر های هر کدام به چه صورت است.

پارامتر های این ۴ فراخوان سیستمی را می‌توانید در جدول زیر مشاهده کنید. در سطر اول، آنهایی که با % شروع می‌شوند نام رجیستر های پردازنده هستند. عدد داخل رجیستر rax، شماره‌ی فراخوان سیستمی را نشان می‌دهد.

%rax	System Call	%rdi	%rsi	%rdx	%r10	%r8	%r9
1	sys_write	unsigned int fd	char *buf	size_t count			
3	sys_close	unsigned int fd					
39	sys_getpid						
99	sys_sysinfo	struct sysinfo *info					

- sys_write

این فراخوان سیستمی ۳ پارامتر می‌گیرد. در rdi یک عدد، در rsi آدرس یک بلاک حافظه و در rdx نیز یک عدد قرار می‌گیرد. پس این فراخوان سیستمی از هر دو روش رجیستر و آدرس بلاک حافظه استفاده می‌کند.

- sys_close

این فراخوان سیستمی یک پارامتر می‌گیرد. در rdi یک عدد قرار می‌گیرد. این فراخوان سیستمی فقط از روش رجیستر استفاده می‌کند.

- sys_getpid

این فراخوان سیستمی پارامتر ورودی ندارد.

- sys_sysinfo

این فراخوان سیستمی یک پارامتر می گیرد. در rdi آدرس یک بلاک حافظه قرار می‌گیرد. این فراخوان سیستمی فقط از روش آدرس بلاک حافظه استفاده می‌کند.

اطلاعات عمومی: برای اینکه بفهمید هر فراخوان سیستمی چه پارامتر هایی می‌گیرد، معنای این پارامتر ها چیست و چه کاری انجام می‌دهد و به طور کلی برای اینکه توضیحات کامل در مورد آن فراخوان سیستمی را بخوانید می‌توانید در سیستم های مبتنی بر Unix مانند لینوکس و مک از دستور man 2 syscall استفاده کنید و به جای syscall اسم فراخوان سیستمی مد نظر خود را قرار دهید. برای مثال برای توضیحات در مورد sys_sysinfo می‌توانید دستور man 2 sysinfo را اجرا کنید. برای مثال با اجرای دستوری که گفتیم می‌توانیم بفهمیم ساختار struct sysinfo به صورت زیر است:

```
Since Linux 2.3.23 (i386) and Linux 2.3.48 (all architectures)
the structure is:

    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 */
    };
```

۴- چگونه می‌توان سیستمی طراحی کرد که اجازه‌ی انتخاب یک سیستم عامل از چند سیستم عامل را هنگام بوت شدن به کاربر بدهد؟ برنامه‌ی bootstrap برای این منظور چه کاری باید انجام دهد؟

در بسیاری از سیستم‌های کامپیوتری، برنامه‌ای به نام bootstrap وجود دارد که وظیفه‌ی آن load کردن هسته‌ی سیستم عامل است. به این صورت که برنامه‌ی هسته‌ی سیستم عامل را به حافظه‌ی اصلی سیستم (main memory) منتقل کرده و اجرای آن را آغاز می‌کند. گاهی نیز شروع اجرای هسته‌ی سیستم عامل در دو مرحله صورت می‌گیرد. بدین صورت که برنامه‌ی ساده‌ی bootstrap برنامه‌ی پیچیده‌ی دیگری را جهت لود کردن هسته‌ی سیستم عمل از دیسک انتخاب کرده و آن را در main memory اجرا می‌کند و این برنامه، کرنل مختص به سیستم عامل خود را لود می‌کند.

۵- یک برنامه‌ی کمکی می‌خواد به گونه‌ای این امکان را فراهم کند که یک برنامه که برای ویندوز کامپایل شده را در لینوکس اجرا کند. اگر معماری پردازنده‌ی دو سیستم عامل یکسان باشد، توضیح دهید این برنامه چه کارهایی باید انجام دهد.

تفاوت هایی که باعث می‌شود برنامه های کامپایل شده برای یک سیستم عامل در سیستم عامل دیگر قابل اجرا نباشند:

1. فرمت باینری برنامه ها و مکان قرار گیری اطلاعات یک برنامه در سیستم عامل های مختلف متفاوت است.
2. ممکن است معماری پردازنده‌ی دو سیستم متفاوت باشد و امکان اجرای دستورات سیستم دیگر وجود نداشته باشد.
3. ممکن است تفاوت هایی در API سیستم کال ها و یا روش ارسال پارامتر به سیستم کال وجود داشته باشد.

چون گفته شده معماری پردازنده ها یکسان است مورد دوم مشکلی برای ما ایجاد نمی‌کند، پس این برنامه ابتدا باید بتواند فایل exe ویندوز با فرمت PE را مطابق فرمت ELF مورد استفاده در لینوکس لود و اجرا کند. سپس باید سیستم کال های ویندوز را در لینوکس پیاده سازی کند تا برنامه‌ی ویندوزی بتواند از آنها استفاده کند. مشخصا برنامه‌ی واقعی از این پیچیده‌تر است و در صورت علاقه می‌توانید برای اطلاعات بیشتر در مورد برنامه‌ی Wine مطالعه کنید.

۶- وقتی یک پردازه با صدا زدن fork پردازه‌ی جدیدی می‌سازد، کدام از موارد زیر بین پردازه‌ی والد و فرزند مشترک خواهد بود؟

- **استک**
- **هیپ**
- **قسمت های حافظه‌ی مشترک**

فقط سگمنت‌های مشترک حافظه میان پردازه‌ی والد و فرزند مشترک خواهند بود. برای پردازه‌ی فرزند استک و هیپ جدیدی با کپی گرفتن از استک و هیپ پردازه‌ی والد ایجاد می‌شود.