Operating systems

**Project2 report**

**رضا چهرقانی 810101401**

**امیر نداف فهمیده 810101540**

**مصطفی کرمانی‌نیا 810101575**

**مخزن گيتهاب اين پروژه‌:**

**https://github.com/mostafa-kermaninia/OS\_LAB\_P2**

horizontal line

* **پرسش 1) کتابخانه‌های سطح کاربر در XV6، برای ایجاد ارتباط میان برنامه‌های کاربر و کرنل به کار می‌روند. این کتابخانه‌ها شامل توابعی هستند که از فراخوانی‌های سیستمی استفاده می‌کنند تا دسترسی به منابع سخت‌افزاری و نرم‌افزاری سیستم‌عامل ممکن شود. با تحلیل فایل‌های موجود در متغیر ULIB در XV6، توضیح دهید که چگونه این کتابخانه‌ها از فراخوانی‌های سیستمی بهره می‌برند؟ همچنین، دلایل استفاده از این فراخوانی‌ها و تأثیر آنها بر عملکرد و قابلیت حمل برنامه‌ها را شرح دهید.**

در xv6، کتابخانه‌های کاربری که در پوشه ULIB قرار دارند، واسطی بین برنامه‌های کاربری و کرنل فراهم می‌کنند که شامل توابعی است که مستقیماً از System Call‌ها استفاده می‌کنند. این توابع با فراخوانی‌های سیستمی، امکان دسترسی به منابع کرنل و در نهایت منابع سخت‌افزاری را به برنامه‌های کاربری می‌دهند. برای استفاده از System Call، برنامه کاربری ابتدا باید به کرنل سیگنالی برای ورود به "حالت کرنل" بدهد. در این حالت، کرنل به اجرای کد مربوط به System Call پرداخته و پس از اتمام، دوباره کنترل را به برنامه کاربری برمی‌گرداند​​.

### دلایل و تأثیرات استفاده از فراخوانی‌های سیستمی

استفاده از فراخوانی‌های سیستمی از طریق کتابخانه‌های کاربر در xv6 دارای مزایای متعددی است:

* کتابخانه‌های کاربر پیچیدگی تعامل مستقیم با سخت‌افزار را پنهان می‌کنند و توسعه برنامه‌های کاربری را ساده‌تر می‌سازند.
* با محدود کردن دسترسی مستقیم به سخت‌افزار، کرنل می‌تواند سیاست‌های امنیتی را اعمال کرده و اطمینان حاصل کند که فرآیندها در محدوده‌های مجاز عمل می‌کنند.
* با ارائه مجموعه‌ای یکپارچه از فراخوانی‌های سیستمی، کتابخانه‌های کاربر در xv6 تعامل با کرنل را استاندارد کرده و برنامه‌ها را قابل حمل‌تر می‌سازند.
* فراخوانی‌های سیستمی امکان مدیریت خطا در سطح کرنل را فراهم می‌کنند، که می‌تواند با شکست‌هایی مثل کمبود حافظه یا خطاهای دسترسی به فایل به صورت کارآمدتری برخورد کند.

### تأثیر بر قابلیت حمل و عملکرد

استفاده از فراخوانی‌های سیستمی از طریق کتابخانه‌های کاربر، با رعایت استانداردهای مشابه سیستم‌های شبه یونیکس، بر قابلیت حمل برنامه‌ها تأثیر مثبتی دارد و مهاجرت برنامه‌های xv6 به سایر سیستم‌های مبتنی بر یونیکس را تسهیل می‌کند. با این حال، انتقال بین فضای کاربر و فضای کرنل در طی فراخوانی‌های سیستمی باعث کاهش عملکرد می‌شود، زیرا نیاز به تغییر وضعیت یا همان context switch دارد. این کاهش عملکرد، بهای امنیت و پایداری است که مکانیزم‌های حفاظتی کرنل فراهم می‌کنند.

### تحلیل فایل های موجود

متغیر ULIB (در makefile) :



این متغیر از 4 تا object file تشکیل می شود. حالا برای تشخیص system call های آنها باید به کد منبع آن ها مراجعه کنیم:

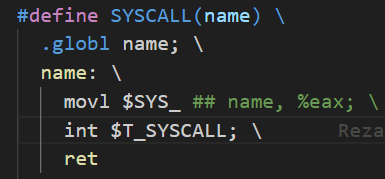
1. **ulib.c**

این فایل کد منبع ulib.o است و شامل توابع کمکی‌ای مانند strcpy، strcmp، strlen، memset، strchr، gets، stat، atoi و memmove است که به عنوان توابع سطح کاربر عمل می‌کنند (پس در h.user اظهار یا declare شده اند) می باشد. از این توابع، تنها دو تابع gets و stat از فراخوانی‌های سیستمی بهره می‌برند:

* تابع gets: این تابع برای خواندن ورودی از stdin، از فراخوانی سیستمی read استفاده می‌کند. هر کاراکتر از ورودی با استفاده از این فراخوانی در یک حلقه خوانده می‌شود و در متغیر یا آرایه مشخصی ذخیره می‌گردد.
* تابع stat: این تابع سه فراخوانی سیستمی open، fstat و close را برای دسترسی به فایل‌ها و خواندن اطلاعات متادیتای آن‌ها (مانند سایز فایل) استفاده می‌کند. ابتدا فایل مورد نظر با open باز می‌شود، سپس با fstat اطلاعات فایل به دست می‌آید و در نهایت با close فایل بسته می‌شود.

1. **usys.S**

این فایل یک کد اسمبلی است و کد منبع usys.o است و شامل پوشاننده‌های (wrappers) فراخوانی‌های سیستمی است. در این فایل، یک ماکرو به نام SYSCALL تعریف شده که وظیفه ایجاد این پوشاننده‌ها را بر عهده دارد:



این ماکرو به ازای هر فراخوانی سیستمی، نام آن را به صورت global اعلام کرده و شماره فراخوانی سیستمی مربوطه را در رجیستر eax قرار می‌دهد، سپس دستور int $T\_SYSCALL را اجرا می‌کند ($T\_SYSCALL برابر با 64 است زیرا شماره تله فراخوانی سیستمی 64 است و برنامه جهت فراخوانی سیستمی دستور 64 int را فراخوانی می کند) که باعث ایجاد یک وقفه نرم‌افزاری شده و کنترل را به کرنل منتقل می‌کند.

1. **printf.c**

فایل printf.c کد منبع printf.o و شامل توابعی مانند putc، printint و printf است که برای چاپ کردن اطلاعات به خروجی مورد استفاده قرار می‌گیرند. تنها تابع putc از فراخوانی سیستمی استفاده می‌کند (در تو تابع دیگر از تابع putc استفاده شده است):

* تابع putc: این تابع از فراخوانی سیستمی write برای ارسال یک کاراکتر مشخص به fd مورد نظر (که معمولاً stdout است) استفاده می کند.

از آنجا که توابع printf و printint در نهایت putc را فراخوانی می‌کنند، به‌طور غیرمستقیم از فراخوانی سیستمی write استفاده می‌کنند تا اطلاعات را به خروجی ارسال کنند.

1. **umalloc.c**

این فایل کد منبع umalloc.o است و شامل توابعی مانند malloc، free و morecore است. تابع malloc برای تخصیص حافظه پویا و free برای آزاد کردن آن به کار می‌روند و هیچکدام مستقیماً از فراخوانی‌های سیستمی استفاده نمی‌کنند. (تابع free یک پوینتر را به عنوان ورودی می گیرد که این پوینتر به یک آدرس از حافظه اشاره می کند و سپس این بخش از حافظه را آزاد می کند و تابع malloc برای تخصیص حافظه ی پویا با اندازه ی مشخص استفاده می شود که مقدار خروجی آن یک اشاره گر void به حافظه ی تخصیص داده شده است.) اما تابع morecore که برای افزایش فضای حافظه مورد نیاز فرآیند استفاده می‌شود، از فراخوانی سیستمی sbrk بهره می‌برد تا فضای حافظه پردازه را به میزان دلخواه افزایش دهد.

* **پرسش 2) فراخوانی‌های سیستمی تنها روش برای تعامل برنامه‌های کاربر با کرنل نیستند. چه روش‌های دیگری در لینوکس وجود دارند که برنامه‌های سطح کاربر می‌توانند از طریق آنها به کرنل دسترسی داشته باشند؟ هر یک از این روش‌ها را به اختصار توضیح دهید.**

می‌توانیم وقایع رخ داده در سیستم را به صورت زیر دسته‌بندی کنیم:

۱- Exception  
۲- Interrupt

* S.W interrupt
* H.W interrupt

به طور کل دسترسی به هسته با یک interrupt (وقفه) رخ می‌دهد.

وقفه سخت‌افزاری: این وقفه توسط دستگاه‌های سخت‌افزاری خارجی به صورت آسینکرون تولید می‌شود که به طور مثال می‌تواند برای ورودی کاربر، تکمیل عملیات I/O و... باشد. در حالتی که چنین وقفه‌ای رخ می‌دهد CPU فعالیت فعلی خود را ذخیره می‌کند و کنترل را به interrupt service routine منتقل می‌کند. بعد از انجام کارهای مربوطه برای رفع interrupt کنترل به CPU باز می‌گردد.

وقفه نرم‌افزاری (trap): این وقفه توسط برنامه به صورت سنکرون ایجاد می‌شود. این وقفه‌ها معمولاً به دلیل درخواست انجام یک کار توسط سیستم عامل ایجاد می‌شود که به طور مثال می‌توان به درخواست یک حافظه اشتراکی، خاتمه دادن به یک برنامه، بازکردن یک فایل و... باشد.

از انواع trap ها می‌توان به:  
1- Signal  
سیگنال های مختلف در لینوکس وجود دارد نظیر SIGKILL  
 System call -2  
که به آن اشاره شده است.

در لینوکس Pseudo-file-systems نیز وجود دارد.

* **پرسش 3) آیا باقی تله‌ها را نمی‌توان در سطح DPL USER فعال نمود؟ چرا؟**

خیر چنین کاری امکان ندارد. سطح ذکر شده سطح دسترسی کاربر است و در این سطح نباید اجازه دسترسی به هسته سیستم عامل (کرنل) و اجرای تله‌ها را داشت. درصورت انجام این عمل protection exception فعال می‌شود.

لازم به ذکر است که درصورتی که این اجازه داده می‌شد به kernel protection ایراد وارد می‌شد.

از دلایلی که این موضوع در انحصار سطح kernel است می‌توان به:

۱- یک برنامه قصد سوءاستفاده از هسته را دارد و با این روش کنترل کل هسته سیستم عامل را در اختیار می‌گیرد.

۲- یک برنامه مخرب در چنین حالتی می‌تواند با آسیب به هسته تمام سطوح H.W & S.W آسیب وارد کند.

۳- اگر برنامه کاربر دچار ایراد باشد در حالتی که در سطح kernel اجرایی شود این مشکل می‌تواند به تمام سطح هسته گسترش یابد.

* **پرسش 4) در صورت تغییر سطح دسترسی، ss و esp روی پشته Push می‌شود. در غیر این‌صورت Push نمی‌شود. چرا؟**
* **پرسش 5) در مورد توابع دسترسی به پارامترهای فراخوانی سیستمی به طور مختصر توضیح دهید. چرا در argptr() بازه آدرس‌ها بررسی می‌گردد؟ تجاوز از بازه معتبر، چه مشکل امنیتی ایجاد می‌کند؟ در صورت عدم بررسی باز در این تابع، مثالی بزنید که در آن، فراخوانی سیستمی read\_sys() اجرای سیستم را با مشکل روبرو سازد.**