

پاسخ:

در سیستم‌عامل 6v، فراخوانی‌های سیستمی معمولاً با استفاده از یک دستور trap یا وقفه‌ی نرم‌افزاری به کرنل ارسال می‌شوند. هر کدام از توابع کتابخانه سطح کاربر که به عملکردهای سیستمی نیاز دارند (مانند خواندن و نوشتن فایل‌ها، تخصیص حافظه، یا مدیریت پروسه‌ها)، یک فراخوانی سیستمی مناسب را اجرا کرده و سپس به کرنل منتقل می‌کنند. این توابع، شناسه‌ای منحصر به فرد (شماره‌ی فراخوانی سیستمی) و پارامترهای مورد نیاز را به کرنل ارسال می‌کنند، که کرنل این شناسه را تفسیر و سرویس مورد نظر را اجرا می‌کند. این نوع طراحی، به توابع کتابخانه‌های سطح کاربر اجازه می‌دهد تا بدون نیاز به دسترسی مستقیم به سخت‌افزار و جزئیات داخلی کرنل، به سرویس‌های سیستمی دسترسی پیدا کنند.



Ulib شامل آبجکت فایل های زیر است.به کد سی هر کدام مراجعه می کنیم و درباره آن ها توضیح می دهیم:

Ulib.c:

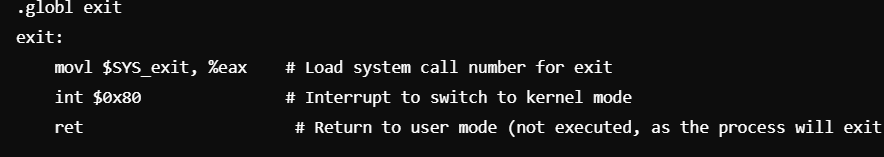
این فایل شامل توابع کاربردی برای کاربر است.این توابع شامل توابع مربوط به تغییر string ها مانند strlrn ، strcpy و strcmp است.گروه دیگر این توابع شامل توابع مربوط به مموری مانند memset و همچنین فانکشن های پرکاربردی برای تبدیل اینت به استرینگ و برعکس مانند atoi است.

در دو تابع gets و stat به ترتیب از syscall های read(برای خواندن از stdin) در اولی،open،close و fstat (اطلاعات فایل ها) استفاده شده.

Usys.s:

شامل یک سری interface برای استفاده از سیستم کال ها در سطح کاربر است.با استفاده از کد سیستم کال و اینتراپت ترنزیشن از سطح یوزر به سطح کرنل اتفاق می افتد.

این تصویر خلاصه ای از کد های در Usys.s است:



Printf.c:

این فایل شامل فانکشن هایی برای نوشتن است.با تابع printint در پروژه ی قبلی برای تغییر کنسول آشنا شدیم.توابعی مانند putc نیز در تابع console برای چاپ کاراکتر استفاده شده است.

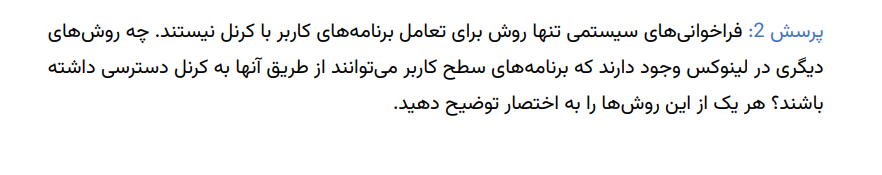
سیستم کال استفاده شده در اینجا سیستم کال write است که برای در فایل دیسکریپتور مورد نظر می نویسد.

Umalloc.c:

در اینجا تابع morecore را داریم که از سیستم کال sbrk برای افزایش سگمنت دیتای پراسس برای مقدار خاصی استفاده می کند.این سیستم کال ترمز برنامه(جایی که دیتا سگمنت تمام شده) را تغییر می دهد تا مموری اضافی فراهم شود. تابع mallocنیز از تابع morecore در مواردی استفاده می کند و غیر مسقیم با این سیستم کال سر و کار دارد.

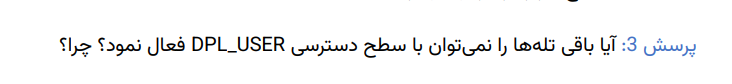
توابعی مانند free نیز وجود دارد که با کرنل کاری ندارد و صرفا حافظه را برای الوکیشن های بعد آزاد می کند.

سیستم کال ها یک اینترفیس فراهم می کنند که مستقل از معماری سخت افزار در صورت ساپورت سیستم عامل از این سیستم کال ها می توان از آن سیستم عامل استفاده کرد.از طرفی برنامه هایی که از سیستم کال ها استفاده می کنند لازم نیست جزییات دقیق مدیریتی سیستم عامل را بدانند و در صورت دانستن اینترفیس سیستم کال ها می توانند به راحتی با هر سیستم عاملی کار کنند.



در سیستم‌عامل لینوکس، روش‌های متعددی برای دسترسی از سطح کاربر به هسته وجود دارد. اگرچه فراخوانی‌های سیستمی (System Calls) یکی از رایج‌ترین این روش‌ها هستند، اما تنها روش نیستند. در ادامه به اختصار به توضیح روش‌های دیگر می‌پردازیم:

1. **فراخوانی‌های سیستمی (System Calls)**:
   * این روش اصلی‌ترین و مستقیم‌ترین راه ارتباط میان برنامه‌های کاربر و هسته است. در این روش، از طریق شماره‌ی فراخوانی سیستمی خاص و یک سری پارامترها، درخواست‌های سطح کاربر به هسته ارسال می‌شود و هسته به آن‌ها پاسخ می‌دهد. این فراخوانی‌ها اغلب شامل عملیات‌هایی مانند خواندن و نوشتن فایل، تخصیص حافظه، مدیریت پردازه‌ها و غیره هستند.
2. **فایل‌های دستگاه (Device Files)**:
   * در لینوکس، بسیاری از دستگاه‌ها و سخت‌افزارها به صورت فایل‌های دستگاه (مانند /dev/null، /dev/sda، و غیره) در سیستم فایل نمایش داده می‌شوند. از طریق خواندن و نوشتن این فایل‌ها، برنامه‌های کاربر می‌توانند با دستگاه‌ها و سخت‌افزارها ارتباط برقرار کنند. این روش امکان دسترسی غیرمستقیم به قابلیت‌های هسته را بدون استفاده مستقیم از فراخوانی‌های سیستمی فراهم می‌کند.
3. **فایل‌های مجازی در /proc و /sys**:
   * سیستم فایل‌های مجازی مانند /proc و /sys حاوی اطلاعات و پارامترهای تنظیمی درباره سیستم، هسته و پردازه‌ها هستند. این فایل‌ها به کاربران اجازه می‌دهند که اطلاعاتی از هسته بخوانند یا تنظیمات خاصی را تغییر دهند. برای مثال، کاربران می‌توانند با خواندن فایل‌های داخل /proc اطلاعاتی در مورد پردازه‌ها، مصرف حافظه، وضعیت CPU و دیگر موارد را بدست آورند.
4. **فراخوانی‌های کتابخانه‌ای (Library Calls)**:
   * در بسیاری از موارد، فراخوانی‌های کتابخانه‌ای (مانند توابع استاندارد libc در C) به عنوان واسطی برای دسترسی به سیستم عمل می‌کنند. این توابع در نهایت از فراخوانی‌های سیستمی استفاده می‌کنند، اما برای برنامه‌نویسان سطح کاربر، این روش به صورت مستقیم با فراخوانی‌های سیستمی درگیر نمی‌شود و از یک واسط سطح بالاتر بهره‌مند می‌شود.
5. **مکانیزم‌های ارتباط بین پردازه‌ای (IPC)**:
   * مکانیزم‌های IPC مانند صف‌های پیغام (Message Queues)، حافظه اشتراکی (Shared Memory) و لوله‌ها (Pipes) روش‌هایی برای ارتباط بین پردازه‌ها فراهم می‌کنند که گاهی به طور غیرمستقیم به هسته دسترسی دارند. هرچند برای استفاده از این مکانیزم‌ها معمولاً از فراخوانی‌های سیستمی نیز استفاده می‌شود اما همچنان برای کاربر ها مانند اینترفیس هستند.
   * . Message Queue (صف‌های پیغام): این روش اجازه می‌دهد که پردازه‌ها پیغام‌هایی را در صف‌هایی ذخیره کرده و پردازه‌های دیگر آن‌ها را دریافت کنند.
   * Shared Memory (حافظه اشتراکی): در این روش پردازه‌ها می‌توانند بخشی از حافظه را به اشتراک بگذارند و به سرعت داده‌ها را بین یکدیگر منتقل کنند.
   * Semaphores (سمافورها): برای هماهنگ‌سازی دسترسی به منابع مشترک بین پردازه‌ها استفاده می‌شود
6. **ioctl** 
   * این فراخوانی خاص برای اجرای عملیات‌های خاص و سفارشی در دستگاه‌ها و فایل‌های خاص استفاده می‌شود و دسترسی‌های ویژه‌ای به هسته و درایورهای دستگاه می‌دهد. فراخوانی ioctl به برنامه‌های کاربر اجازه می‌دهد که عملیات‌های سطح پایین و خاص دستگاه‌ها را اجرا کنند.
7. **سیگنال‌ها (Signals)**:
   * سیگنال‌ها یکی دیگر از روش‌های ارتباط بین کاربر و هسته هستند که معمولاً برای مدیریت اتفاقات ناگهانی و اعلان‌ها استفاده می‌شوند. هسته می‌تواند با ارسال سیگنال به پردازه‌ها، آن‌ها را از رخدادهایی مانند خطاها، وقفه‌ها و تکمیل عملیات‌ها مطلع سازد
   * • برخی از سیگنال‌های رایج عبارتند از:
   * SIGKILL: برای خاتمه دادن به یک پردازه.
   * SIGTERM: برای درخواست خاتمه پردازه به صورت نرم.
   * SIGSEGV: برای اطلاع از دسترسی غیرمجاز به حافظه.
   * • پردازه‌ها می‌توانند سیگنال‌ها را مدیریت کرده یا پردازش کنند، یا سیگنال‌ها می‌توانند به صورت پیش‌فرض در نظر گرفته شوند.
8. **کال‌بک‌ها (Callbacks) و توابع هوکینگ (Hooking Functions)**:
   * برخی مکانیزم‌ها در لینوکس به برنامه‌ها اجازه می‌دهند توابع خاصی را به عنوان "هوک" به وقایع خاصی از سیستم متصل کنند. برای مثال، برنامه‌های مدیریتی ممکن است به گونه‌ای تنظیم شوند که هنگام وقوع یک رویداد خاص (مانند اتصال یا قطع یک دستگاه) فراخوانی شوند.



در سیستم‌عامل‌هایی مانند 6xv که از معماری‌های چند سطحی استفاده می‌کنند، مانند سطح 0 برای هسته (Kernel Mode) و سطح 3 برای کاربر (User Mode)، اصول امنیتی ایجاب می‌کند که دسترسی به هسته برای برنامه‌های سطح کاربر محدود شود. به عبارت دیگر، سطح دسترسی کاربر (DPL\_USER) که معادل سطح 3 است، نباید اجازه دسترسی مستقیم به هسته و اجرا کردن تله‌ها (traps) را بدهد.

اگر یک برنامه سطح کاربر به راحتی می‌توانست تله‌ها را فعال کند و به هسته دسترسی پیدا کند، این امر با اصول حفاظت از هسته در تضاد است. به این ترتیب، در معماری‌های مدرن، هسته سیستم‌عامل از پردازش‌ها و برنامه‌های سطح کاربر به طور منطقی جدا می‌شود تا امنیت سیستم حفظ شود.

برای مثال، اگر یک پردازنده بخواهد تله‌ای را فعال کند که منجر به ایجاد وقفه‌ای در سیستم شود، هسته سیستم‌عامل به طور مستقیم این دسترسی را به پردازش‌های سطح کاربر نمی‌دهد، چرا که چنین درخواست‌هایی باید توسط هسته و در سطح دسترسی بالاتر مدیریت شوند. این کار به این دلیل است که هسته از حملات احتمالی که ممکن است از سوی برنامه‌های سطح کاربر صورت گیرد، محافظت کند.

اگر چنین دسترسی‌هایی ممکن می‌شد، کاربر می‌توانست با بهره‌برداری از این تله‌ها به هسته حمله کند و به تمامی منابع سیستم، از جمله سخت‌افزار و نرم‌افزار، دسترسی پیدا کند. این موضوع باعث آسیب‌پذیری جدی در امنیت سیستم می‌شد و می‌توانست به هسته آسیب بزند یا دستکاری‌هایی در آن ایجاد کند. به همین دلیل، دسترسی مستقیم برنامه‌های سطح کاربر به هسته به طور عمدی مسدود می‌شود تا از این نوع حملات جلوگیری شود.