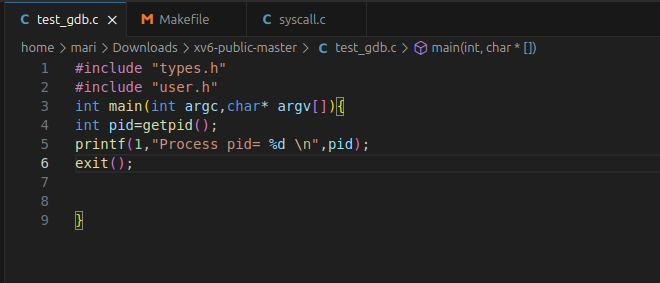
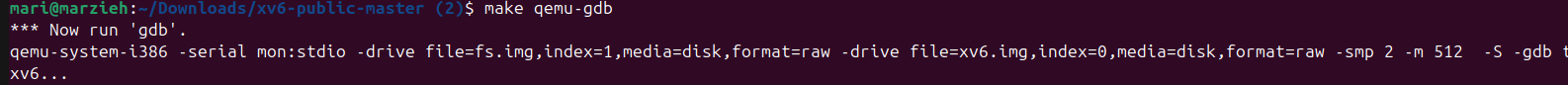
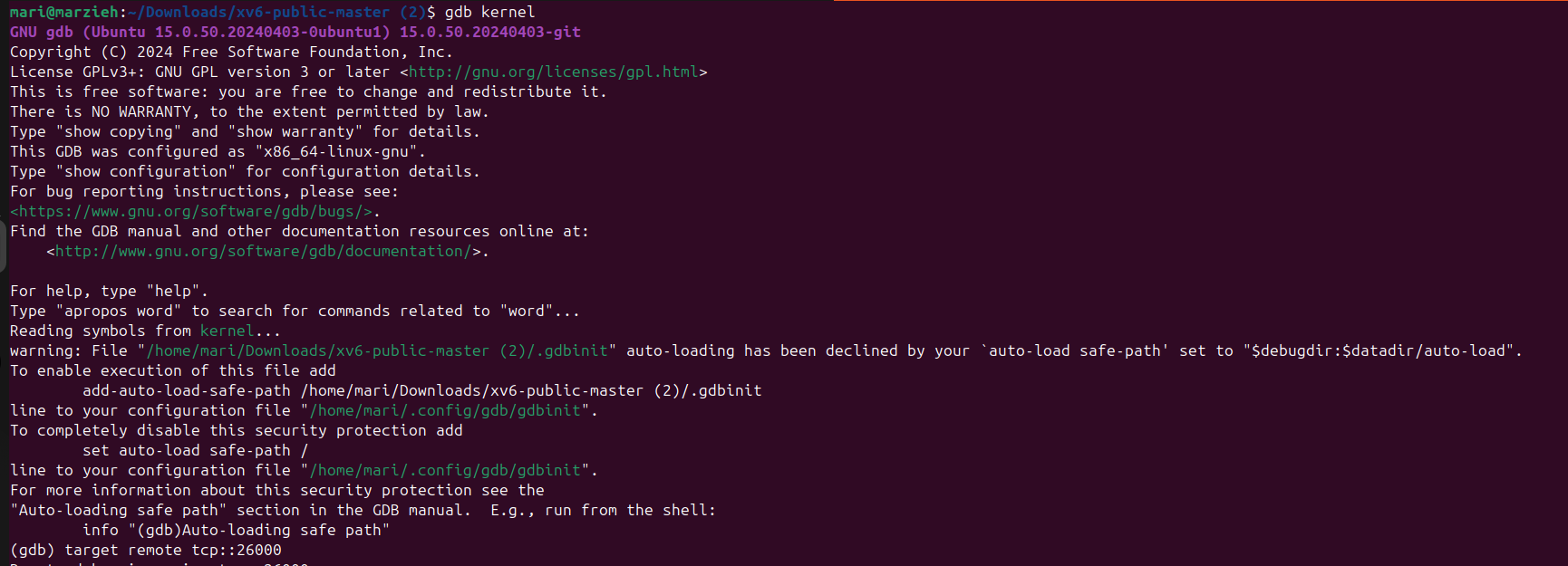


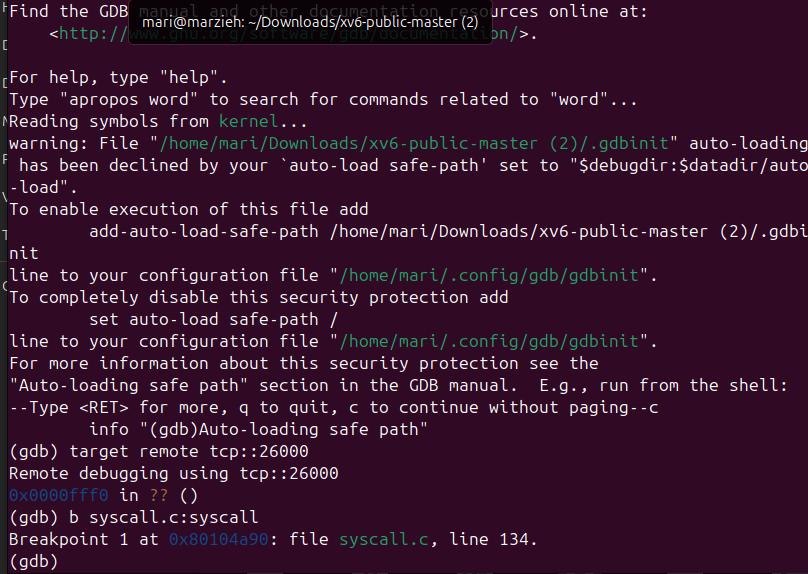
ابتدا برنامه ی سطح کاربر زیر را نوشته و طبق روال به xv6 اضافه می کنیم:  


حال دستور make qemu-gdb را در یک ترمینال ران کرده و در ترمینال دیگر دستور gdb kernel را ران می کنیم:

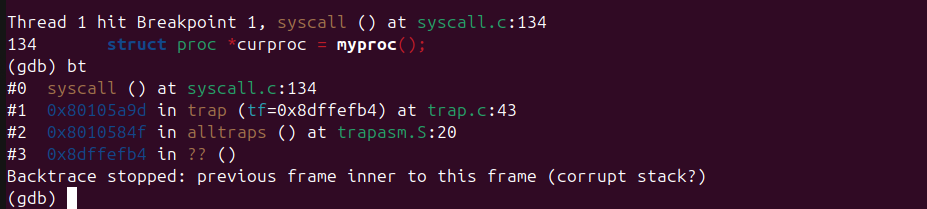




حال طبق روال دستور target remote tcp را برای اتصال به سیستم عامل می زنیم و سپس breakpoint را بر سر تابع syscall در فایل syscall.c می گذاریم:



برای دیدن استک فریم های مربوط به اجرا از دستور bt استفاده می کنیم:



زمانی که دستورbt در gdb اجرا می‌شود، اطلاعات موجود در **پشته فراخوانی توابع** (stack call) را نمایش می‌دهد. این پشته‌ی فراخوانی مانند یک استک (stack) عمل می‌کند و به ما امکان می‌دهد تا روند اجرای برنامه را از نقطه فعلی تا ابتدای اجرای برنامه بررسی کنیم. به عبارت دیگر، این پشته شامل لیستی از توابعی است که به ترتیب فراخوانی شده‌اند تا برنامه به وضعیت فعلی برسد.

**پشته فراخوانی** هر بار که یک تابع در برنامه فراخوانی می‌شود، یک بلاک حافظه‌ای به نام **stack frame** برای آن تابع در بالای استک اضافه می‌کند. این stack frame شامل اطلاعاتی از قبیل:

* **آدرس بازگشت**: آدرسی که پس از اجرای تابع، برنامه به آن بازمی‌گردد.
* **آرگومان‌های تابع**: مقادیری که در هنگام فراخوانی به تابع ارسال شده‌اند.
* **متغیرهای محلی تابع**: داده‌هایی که در درون تابع تعریف و ذخیره شده‌اند.
* **اطلاعات دیباگ**: آدرس تابع در حافظه و مسیر فایل سورس کد.

این فرآیند به ما کمک می‌کند تا مسیر دقیق اجرای برنامه را درک کنیم و بفهمیم کدام توابع و با چه ترتیب و پارامترهایی فراخوانی شده‌اند. با اجرای دستور bt، اطلاعات پشته فراخوانی از جمله:

* **نام توابع فراخوانی شده**
* **آدرس دقیق هر تابع در سورس کد**
* **آدرس حافظه هر تابع**
* **آرگومان‌های ورودی توابع و مقادیر آن‌ها**

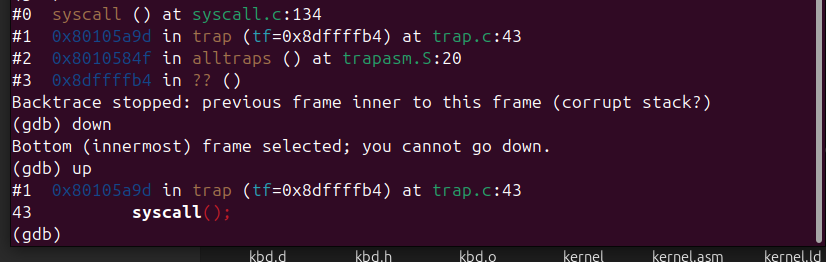
به ترتیب نمایش داده می‌شوند.

برای اضافه کردن یک سیستم کال (System Call) جدید در سیستم‌عامل، مراحل مختلفی طی می‌شود که به شرح زیر هستند:

1. **اختصاص شناسه و شماره‌ی سیستم کال**  
   هر سیستم کال در سیستم‌عامل با یک شناسه‌ی عددی یکتا شناسایی می‌شود. این شماره‌ها در فایل‌های syscall.h تعریف می‌شوند. اگر بخواهیم یک سیستم کال جدید اضافه کنیم، باید آن را در این لیست ثبت کنیم و یک شناسه‌ی یکتا به آن تخصیص دهیم.پرتوتایپ سیستم کال های مربوط به بخش کاربر نیز در user.h موجود است.
2. **تعریف سیستم کال در سطح اسمبلی**  
   سیستم کال‌ها برای تعامل مستقیم با سخت‌افزار به زبان اسمبلی تعریف می‌شوند و این تعاریف در فایل usys.s قرار دارند. پپ
3. **رجیستر eax و ذخیره مقدار بازگشتی**  
   در xv6، رجیستر eax برای ذخیره شماره‌ی سیستم کال استفاده می‌شود. هنگامی که سیستم کال فراخوانی می‌شود، شماره آن در eax ذخیره شده و قبل از اجرای دستور int 64 که سیگنال اجرای سیستم کال را صادر می‌کند، قرار می‌گیرد. پس از اجرای این دستور، مقدار یا وضعیت بازگشتی از سیستم کال نیز در eax ذخیره می‌شود.
4. **اجرای دستور int 64 و ورود به بردار 64**  
   پس از فراخوانی سیستم کال و اجرای دستور int 64، پردازنده به (interrupt vector) هدایت می‌شود. سپس در این مرحله (vector address) به‌صورت خودکار در استک (stack) ذخیره شده و به تابع alltraps که در فایل vector.s تعریف شده، (jump) می‌شود.
5. **ایجاد trap frame توسط alltraps**  
   در alltraps، ابتدا trap frame ساخته می‌شود که شامل اطلاعات لازم از وضعیت سیستم در زمان فراخوانی سیستم کال است. این trap frame در استک قرار می‌گیرد و سپس alltraps تابع trap را که در فایل trap.c قرار دارد، فراخوانی می‌کند.
6. **فراخوانی trap و تشخیص سیستم کال با شماره 64**  
   در تابع trap، trap frame که قبلاً در استک قرار داده شده بود به‌عنوان trap frame فرآیند جاری تنظیم می‌شود. با استفاده از شماره 64، مشخص می‌شود که یک سیستم کال فراخوانی شده است و سپس تابع syscall فراخوانی می‌شود.
7. **اجرای تابع syscall**  
   تابع syscall پردازش سیستم کال را آغاز می‌کند. این تابع، در واقع تمامی مراحل پردازش سیستم کال را انجام می‌دهد و پس از آن نتیجه به فراخواننده بازمی‌گردد.

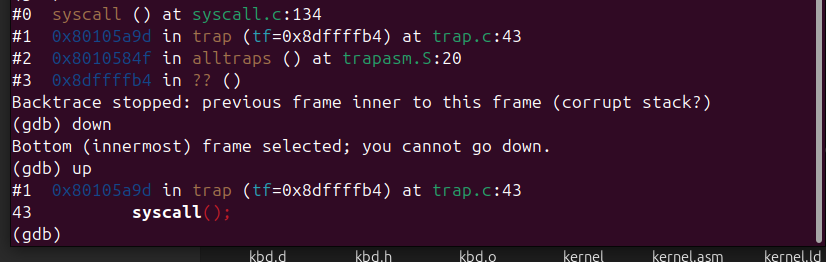
همان‌طور که در تصویر نیز قابل مشاهده است، مراحل مختلف سیستم کال شامل فراخوانی از طریق trap و alltraps هستند که به ترتیب در این مسیر قرار گرفته‌اند و تا اجرای کامل syscall طی می‌شوند.

حال دستور down را می زنیم:

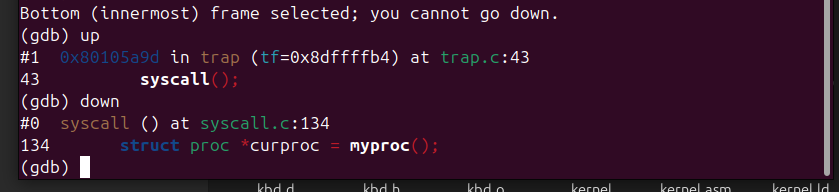


همانطور که مشاهده می کنید ما نمی توانیم به فریم درونی تر برویم.دلیل آن است که فریم درونی تر تابعی است که آخرین تابع ما صدا زده که بدیهتا آخرین تابع ما تابعی را صدا نزده بنابراین نمی توانیم پایین تر برویم.

دستور down و up همانطور که در آزمایش قبل نیز توضیح داده شد برای جابجای بین استک فریم ها می باشد. دستور down به استک فریم callee و دستور up به استک فریم caller می رود.



حال اگر بعد از دستور up دستور down را بزنیم مشکلی پیش نمی آید:

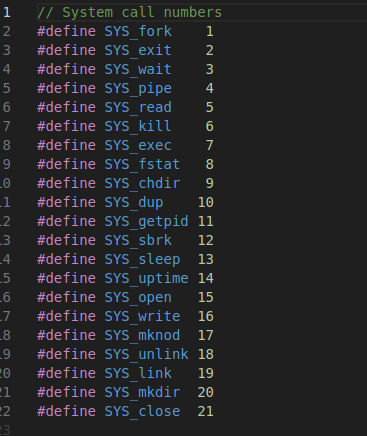


قبلا درباره دستور print که به ما محتوای اکسپرشن ها را نشان می دهد صحبت کردیم.از این دستور برای دیدن مقدار eax که در آن سیستم کال پراسس مورد نظر استفاده می کنیم:

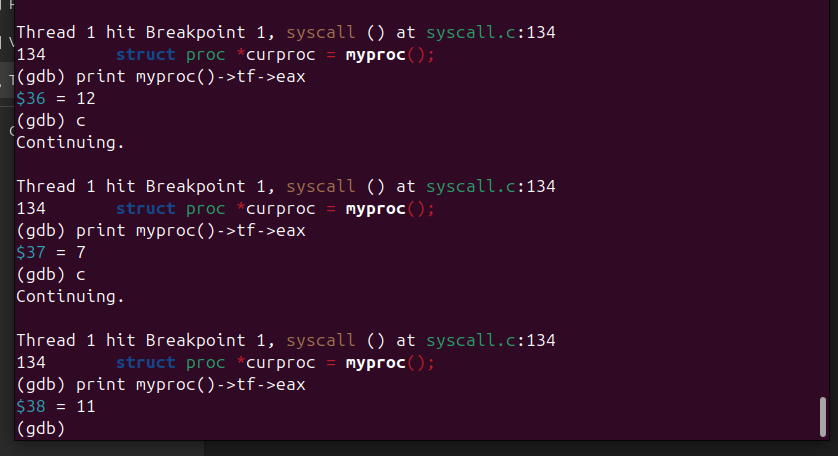


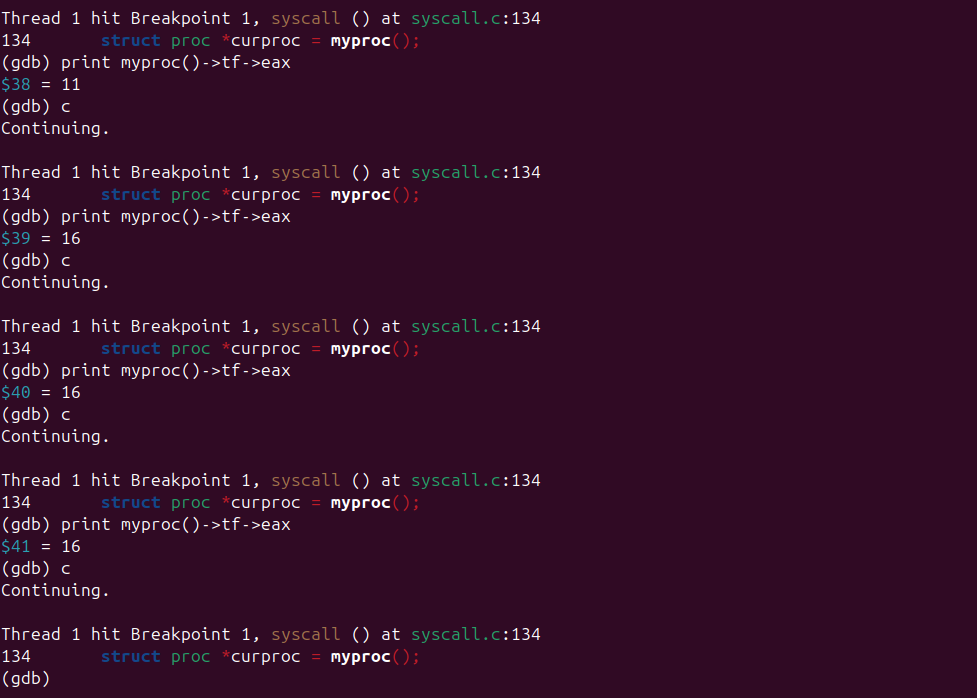
شاید این سوال پیش بیاید که چرا هیچ کدام از کد های سیستم کال ،مقداری برابر 11 ندارد ؟دلیل آن است که ابتدا پراسس ابتدایی که شامل بوت سیستم و پراسس های دیگری باید اجرا شوند تا نوبت به برنامه سطح کاربر ما برسد.

آیدی سیستم کال های ما در سیستم عامل xv6 در فایل syscall.h قرار دارد که به شرح زیر است:



بنابراین آن قدر دستور c که معادل continue است را می زنیم که به ورودی گرفتن از کاربر و برنامه ی سطح کاربر برسیم:



طبق برنامه باید انتظار داشته باشیم بعد از سیستم کال 11 برنامه شروع به نوشتن کند و کد 16 را ببینیم:  


که نتیجه مورد انتظار ماست.

همچنین بعد از نوشتن همه ی کاراکتر ها باید برنامهexit کند که کد 2 را ببینیم:



همچنان نتیجه مورد انتظار است.

16 بعد برای نوشتن $ در خط بعد است.

خروجی نهایی به شرح زیر است:

