|  |
| --- |
| Trường Đại học Khoa học tự nhiên – Khoa Công nghệ thông tin. |
| Đồ án thực hành 02 |
| Operating System – Hệ điều hành. |

|  |
| --- |
| Nhóm 3T  Tháng 11, 2024. |

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**

**📖👨‍💻📖**

**A blue and white logo

Description automatically generated**

**ĐỒ ÁN THỰC HÀNH SỐ 02**

**Bộ môn:** Hệ điều hành.

**Tên đề tài:**

*“System Call”.*

**Tên nhóm:** 3T.

**Thành viên:**

1. 22120384 – Nguyễn Đình Trí.
2. 22120398 – Vũ Hoàng Nhật Trường.
3. 22120412 – Nguyễn Anh Tường.

# **Thông tin chung:**

1. **Bộ môn:** Nhập môn khoa học dữ liệu.
2. **Giảng viên lý thuyết:** Thầy Trần Trung Dũng.
3. **Giảng viên thực hành:** Thầy Lê Thanh Quân.
4. **Mã lớp:** 22\_4.
5. **Tên nhóm:** 3T.
6. **Danh sách thành viên:**
   1. 22120384 – Nguyễn Đình Trí.
   2. 22120398 – Vũ Hoàng Nhật Trường.
   3. 22120412 – Nguyễn Anh Tường.
7. **Link github repository:** [“Click here to go to our github repository.”](https://github.com/richardnguyen0715/OS-Proc-Lap02-SystemCall)

# **Mở đầu:***A blue line drawing of a server Description automatically generated**Giới thiệu chung.*

1. **Bảng phân công công việc:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Họ tên** | **MSSV** | **Câu thực hiện** | **Tiến độ** |
| Nguyễn Anh Tường | 22120412 | Sysinfo | 100% |
| Nguyễn Đình Trí | 22120384 | GDB |  |
| Vũ Hoàng Nhật Trường | 22120398 | Tracing | 100% |

1. **Đánh giá tổng thể:**

**100%**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

# **Câu 1:***A blue line drawing of a server Description automatically generated**GDB*

**I/Cách sử dụng GDB để debug:**

Trong terminal, nhập **make qemu-gdb**. Điều này sẽ chạy QEMU và kích hoạt gỡ lỗi. Ở đây, cổng là 25000. Sau đó mở một terminal khác và chạy **gdb-multiarch -x .gdbinit**. Thao tác này sẽ chạy các lệnh trong .gdbinit, cho phép gỡ lỗi từ xa và đặt Arch thành riscv64.

**II/Giải thích cách thức hoạt động và trả lời các câu hỏi:**

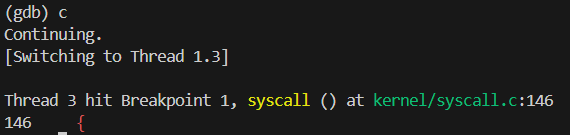
**1/Giải thích một số câu lệnh:**

**1. (gdb) b syscall**



* **Ý nghĩa**: Đặt một điểm dừng (breakpoint) tại hàm syscall.
* **Chi tiết:**
  + GDB sẽ dừng thực thi chương trình ngay khi hàm syscall được gọi.
  + "Breakpoint 1 at 0x80001cf0" cho biết đây là điểm dừng số 1 và địa chỉ bộ nhớ của hàm syscall là 0x80001cf0.
  + Thông tin "file kernel/syscall.c, line 146" chỉ rõ rằng hàm syscall được định nghĩa tại dòng 146 của tệp kernel/syscall.c.

**2. (gdb) c**



* **Ý nghĩa**: Tiếp tục thực thi chương trình (continue).
* **Chi tiết**:
  + Sau khi đặt điểm dừng, chương trình đang tạm dừng. Lệnh c yêu cầu GDB tiếp tục thực thi cho đến khi gặp điểm dừng tiếp theo hoặc chương trình kết thúc.
  + Trong trường hợp này, chương trình tiếp tục chạy cho đến khi chạm tới điểm dừng tại hàm syscall.

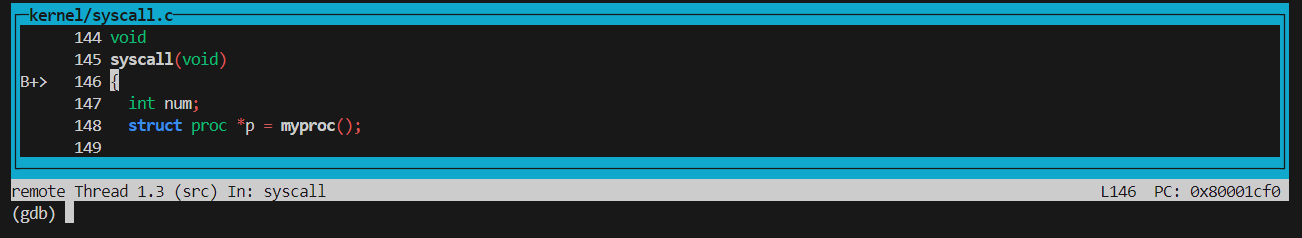
**Thông báo: [Switching to Thread 1.3]**

* **Ý nghĩa**: Chuyển ngữ cảnh sang luồng (thread) 1.2.
* **Chi tiết**:
  + Khi chương trình đang chạy đa luồng, GDB chuyển sang luồng cụ thể có liên quan đến điểm dừng.
  + Luồng 1.2 là luồng kích hoạt điểm dừng.

**Thông báo: Thread 3 hit Breakpoint 1, syscall () at kernel/syscall.c:146**

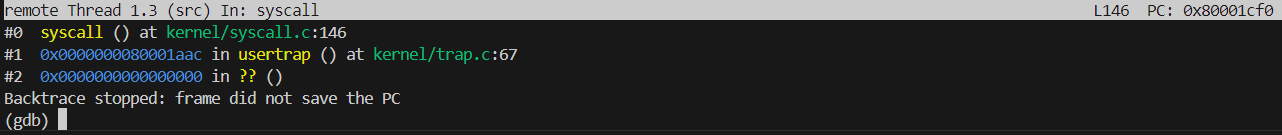
* **Ý nghĩa**: Thông báo rằng luồng 2 đã chạm vào điểm dừng.
* **Chi tiết**:
  + "Thread 2" là luồng đã thực thi đến điểm dừng.
  + "Breakpoint 1" cho biết đây là điểm dừng số 1 đã được đặt trước đó.
  + "syscall () at kernel/syscall.c:146" cho biết chương trình đã dừng tại hàm syscall ở dòng 146 trong tệp kernel/syscall.c.

**3. (gdb) layout src**

****

* **Ý nghĩa**: Hiển thị giao diện dạng văn bản của mã nguồn.
* **Chi tiết**:
  + Lệnh này mở một bố cục (layout) hiển thị mã nguồn trực tiếp trong cửa sổ GDB.
  + Giúp bạn xem mã nguồn ngay trong GDB để dễ dàng theo dõi.

**4. (gdb) backtrace**

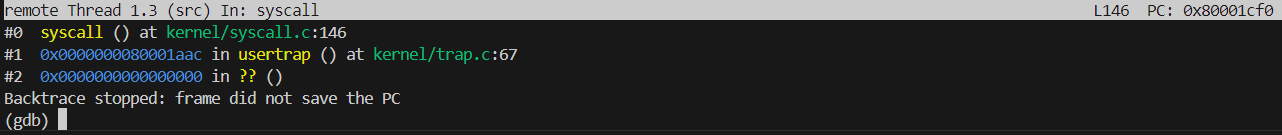
****

* **Ý nghĩa**: Hiển thị ngăn xếp lệnh (call stack).
* **Chi tiết**:
  + Call stack liệt kê các hàm đang được gọi, từ hàm hiện tại đến các hàm gọi trước đó.
  + Thông tin này giúp bạn xác định dòng chảy của chương trình dẫn đến điểm dừng.
  + Mỗi dòng trong backtrace cho biết hàm nào được gọi, các tham số được truyền vào, và dòng mã nơi cuộc gọi xảy ra.

**III/ Trả lời câu hỏi và giải thích**

**1. Looking at the backtrace output, which function called syscall?**

Syscall được gọi bởi usertrap(), được thể hiện trong output của backtrace.

****

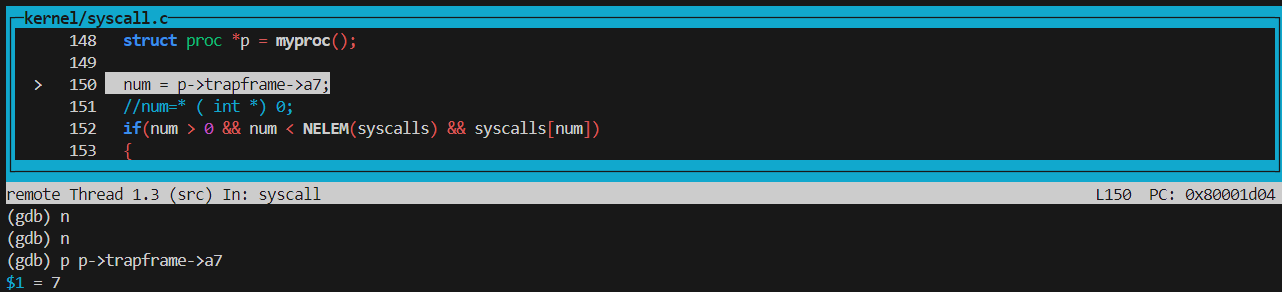
**2. What is the value of p->trapframe->a7 and what does that value represent?**

Ở đây, ta sẽ sử dụng 3 câu lệnh:

Lệnh 1: (gdb) n 🡪 thực hiện câu lệnh hiện tại và di chuyển đến dòng lệnh tiếp theo

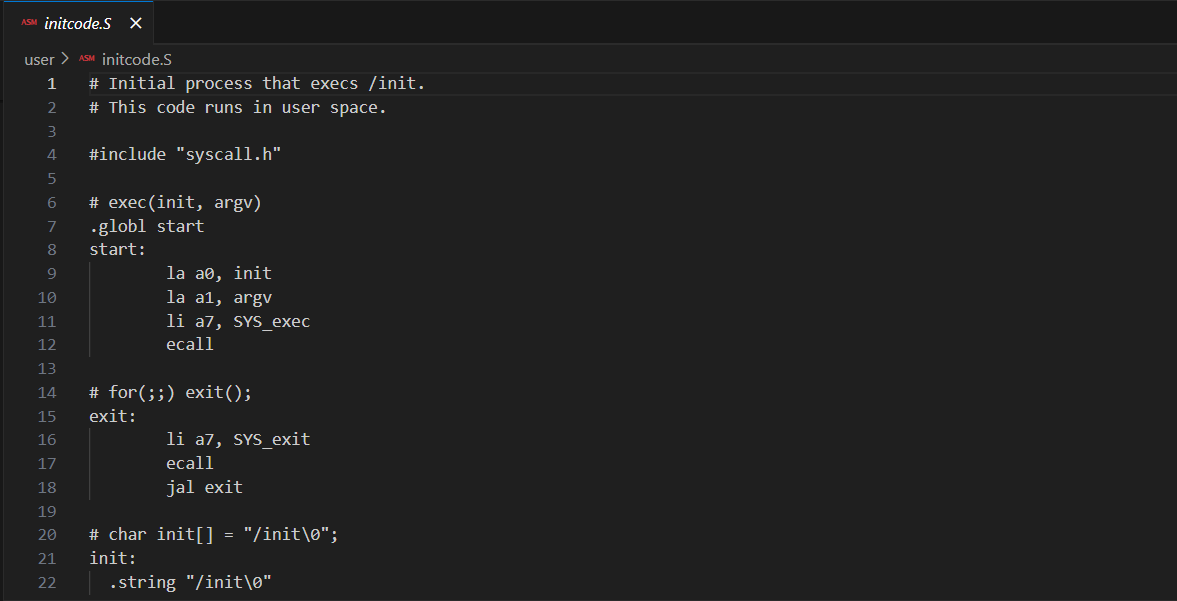
Lệnh 2: (gdb) n

Lệnh 3: (gdb) p p->trapframe->a7 🡪 lệnh in giá trị

****

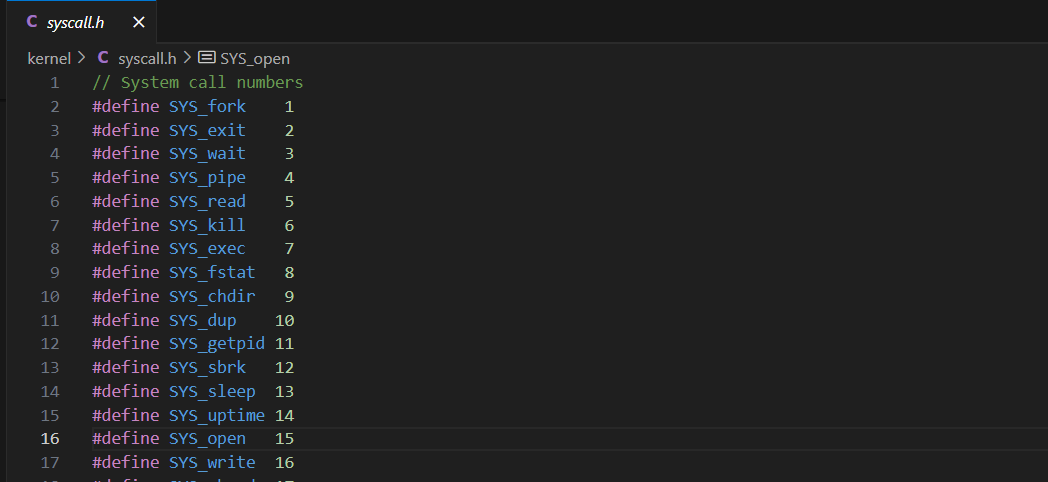
Vậy giá trị của **p->trapframe->a7** là 7.

Mở file **user/initCode.S,** ta thấy number syscall được lưu trong thanh ghi a7.





Mở file **kernal/syscall.h**. Số 7 đại dại cho **SYS\_exec**.





**3. What was the previous mode that the CPU was in?**

Lệnh: p/t $sstatus



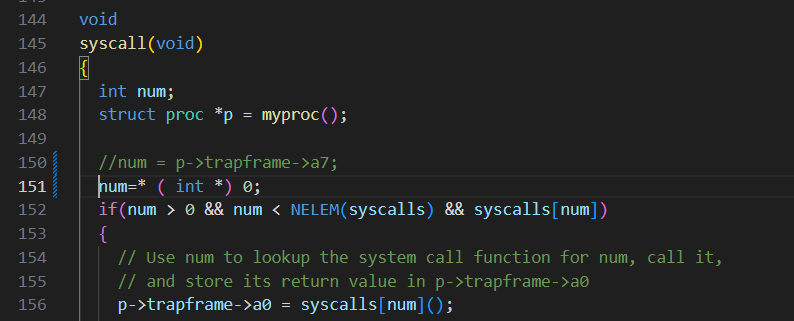
**SPP (Supervisor Previous Privilege)**:  
Theo tài liệu kiến trúc RISC-V, **bit thứ 5 (SPP)** từ phải sang là bit quyết định **privilege mode** trước khi trap xảy ra:

* + SPP = 0: **User Mode** trước trap.
  + SPP = 1: **Supervisor Mode** trước trap.

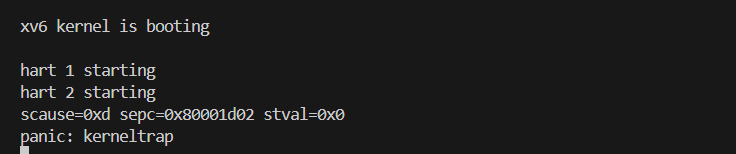
Ở đây SPP = 0, nên previous mode là **User mode.**

**4. Write down the assembly instruction the kernel is panicing at. Which register corresponds to the varialable num?**

Thay thế lệnh **num = p->trapframe->a7;** bởi **num = \* (int \*) 0;**

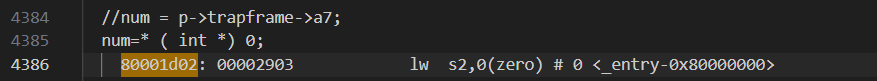


Sau khi thực hiện lệnh **make qemu**:



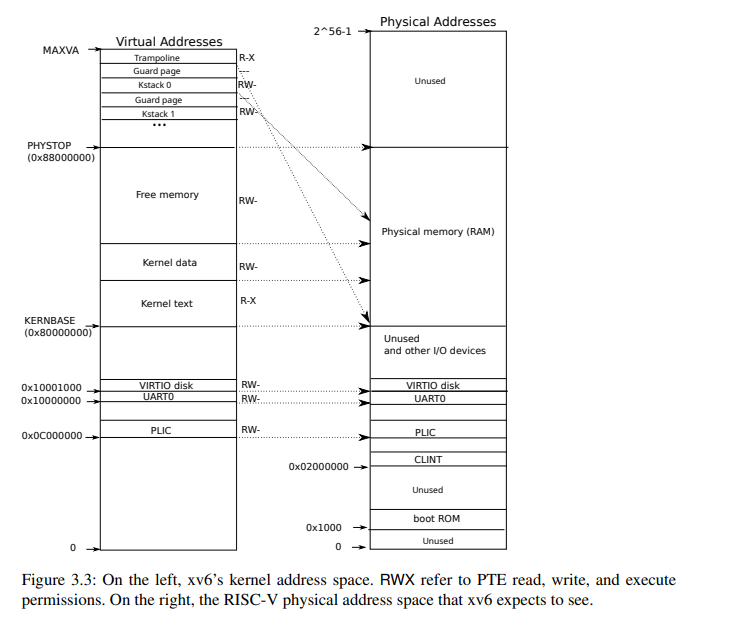
Ở đây, **sepc** đề cập đến địa chỉ của mã trong kernel nơi xảy ra sự cố.

**Sepc= 0x80001d02**, tôi tìm kiếm địa chỉ **80001d02** trong file **kernel/kernel.asm.**



🡪 Câu lệnh lỗi : **lw s2,0(zero),** thanh ghi tương ứng với num là : **s2**

**5. Why does the kernel crash?**

****

Để kiểm tra chúng ta gặp lỗi gì, thực hiện lệnh sau:

****

In giá trị của thanh ghi **scause**, thanh ghi này lưu nguyên nhân gây ra **trap** (ngoại lệ hoặc ngắt) trên kiến trúc RISC-V.

Trên kiến trúc **RISC-V**, giá trị của scause cho biết nguyên nhân gây trap:

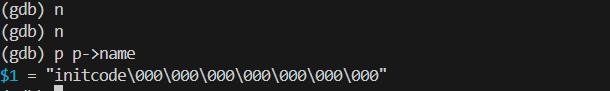
* **13** tương ứng với **Load Page Fault**.

Điều này có nghĩa là:

Kernel đã thực hiện một **lệnh load (đọc dữ liệu)** từ bộ nhớ. Lỗi đó xảy ra   
trong khi tải dữ liệu từ địa chỉ bộ nhớ 0 vào thanh ghi s2, nhưng trong sách giáo trình, địa chỉ 0 không ánh xạ vào không gian kernel.

🡪Trap (ngoại lệ) xảy ra, dẫn đến lỗi.

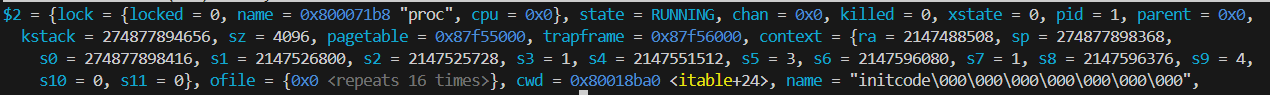
**6. What is the name of the binary that was running when the kernel paniced? What is its process id (pid)?**

****

**🡪** Tên của tệp nhị phân đang chạy khi kernal gặp lỗi: **initcode**

Để xem **pid** của tiến trình, ta thực hiện lệnh **(gdb) p \*p** để in ra chi tiết nội dung tiến trình trên, chính là tiến trình mà con trỏ **p** đang trỏ đến.



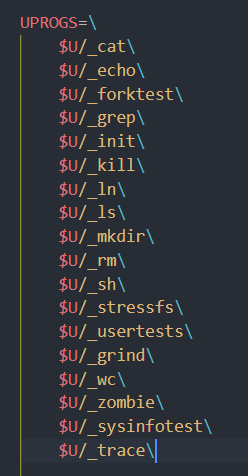


🡪 **pid = 1**

# **Câu 2:***A blue line drawing of a server Description automatically generated**Tracing.*

**Phần khai báo:**

1. **Makefile:**



**Mục đích:** khai báo dùng để biên dịch file trace.c trên make.

1. **user/user.h:**



**Mục đích:** khai báo cấu trúc của systemcall **trace**, với tham số **mask** là kiểu số nguyên.

1. **user/usys.pl:**



**Mục đích:** Tạo liên kết trong file usys.S cho systemcall tương ứng trace (ánh xạ hàm trace trong user -> sys\_trace trong kernel).

**Ghi chú:** usys.S là file đóng vai trò là lớp liên kết kernel và user mode. Giúp chuyển lời gọi từ user\_mode sang kernel\_mode. Nói cách khác thì đây là file dùng để giao tiếp với kernel.

1. **kernel/syscall.h**



**Mục đích:** Tạo một ID cho systemcall trace với một mã định danh là 22.

**Ghi chú:** ID này dùng để ánh xạ hàm đến hàm xử lý tương ứng của nó trong kernel thông qua một hàm ánh xạ trong file syscall.c

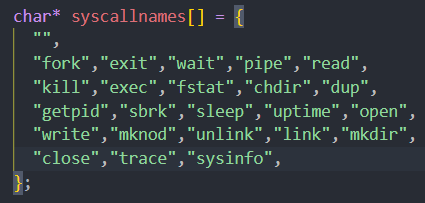
1. **kernel/syscall.c:**



**Mục đích:** khai báo một hàm có tên là sys\_trace(void) đã được định nghĩa ở một nơi khác ( thông qua từ khóa extern).



**Mục đích:** Thêm một khai báo ánh xạ ID trong hàm (\*syscalls[])(void) đã được định nghĩa trong syscall.h (ID = 22).



**Mục đích:** Ánh xạ số ID của systemcall với tên của systemcall ở dạng string. Mảng được dùng để kernel in tên của systemcall cần theo dỗi hoặc ghi log, dùng để truy vết hoạt động của các hàm.

**Ví dụ:** như đang thực thi thì kernel sẽ dùng mảng này để ghi tên systemcall ra log.

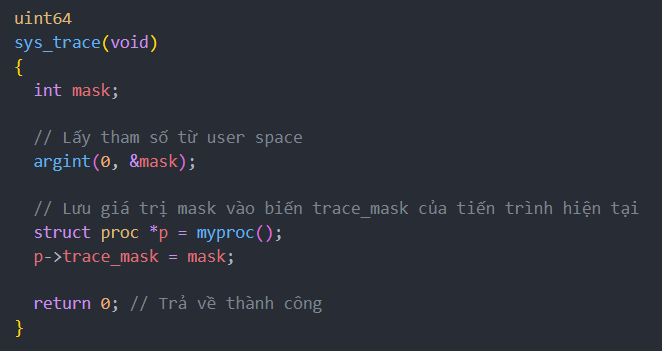
1. **kernel/proc.h**



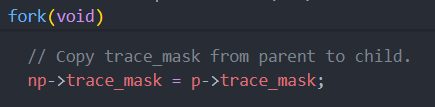
**Mục đích:** Tạo 1 biến **trace\_mask** trong struct **proc** để lưu giá trị số nguyên của mask.

**Phần cài đặt:**

1. **kernel/sysproc.c**

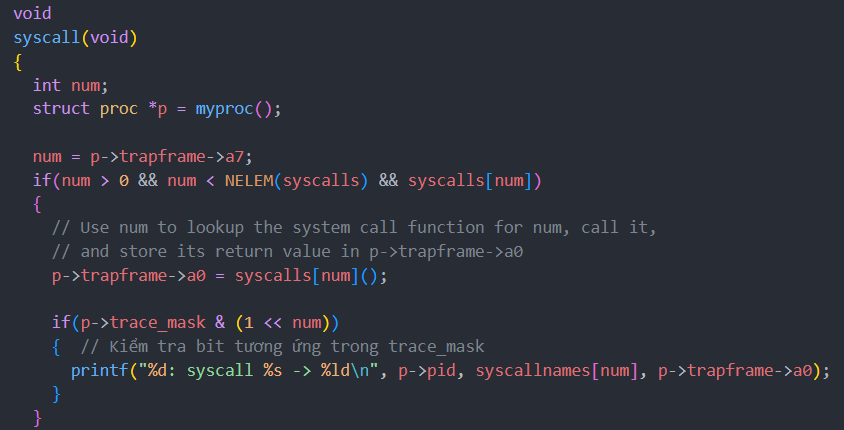
**Mục đích**: Lấy giá trị mask trong tầng user và lưu vào trong biến **trace\_mask** của struct **proc** trong tầng kernel. Hàm trả về 0 nếu thành công.

1. **kernel/proc.c**



**Mục đích**: Khi 1 tiến trình **fork()** để tạo ra tiến trình con, tiến trình con đó cũng có nhu cầu được biết giá trị mask để trace, nên ta sẽ phải sao chép **trace\_mask** của cha vào con (Vì ta không thể chắc chắn rằng khi trace một process, process đó có gọi **fork()** để sinh ra process con hay không, do đó ta phải liệt kê hết).

1. **kernel/syscall.c**

**Mục đích**: Hiệu chỉnh hàm syscall để trả ra thông tin trace được, gồm **process ID**, **tên của syscall** và **giá trị trả ra của syscall** đó (trong thanh ghi a0).

**Lưu ý**: Ở đây ta so sánh **p->trace\_mask & (1 << num)**, với **num** là số thứ tự của syscall đã được khai báo trong file syscall.h. Ta sẽ kiểm tra **bit thứ num** từ **phải sang trái** của **trace\_mask** có được bật hay không, bằng cách **bitwise trace\_mask** với **(1<<num)** (dịch chuyển số 1 trong hệ nhị phân 4 bytes sang trái **num** lần), nếu bật thì xuất ra thông tin, nếu tắt thì không cần xuất ra.

# **Câu 3:***A blue line drawing of a server Description automatically generated**Sysinfo.*

1. **Phần khai báo:**
2. **Makefile:**

A black rectangular object with white text

Description automatically generated

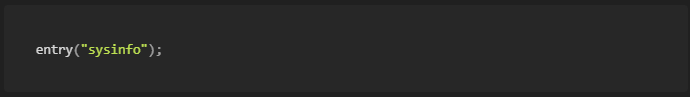
**Mục đích:** khai báo dùng để biên dịch file sysinfotest.c trên make.

1. **User/User.h:**



**Mục đích:** khai báo cấu trúc của systemcall có ý nghĩa là sysinfo.

1. **User/Usys.pl:**

****

**Mục đích:** Tạo liên kết trong file usys.S cho systemcall tương ứng sysinfo.

**Ghi chú:** usys.S là file đóng vai trò là lớp liên kết kernel và user mode. Giúp chuyển lời gọi từ user\_mode sang kernel\_mode. Nói cách khác thì đây là file dùng để giao tiếp với kernel.

1. **Kernel/Syscall.h**

A black rectangular object with white text

Description automatically generated

**Mục đích:** Tạo một ID cho systemcall sysinfo với một mã định danh là 23.

**Ghi chú:** ID này dùng để ánh xạ hàm ( ví dụ: sysinfo ) đến hàm xử lý tương ứng của nó trong kernel thông qua một hàm ánh xạ trong file syscall.c

1. **Kernel/Syscall.c:**

A black rectangular object with white text

Description automatically generated

**Mục đích:** khai báo một hàm có tên là sys\_sysinfo(void) đã được định nghĩa ở một nơi khác ( thông qua từ khóa extern ).

A black rectangular object with white text

Description automatically generated

**Mục đích:** Thêm một khai báo ánh xạ ID trong hàm (\*syscalls[])(void) đã được định nghĩa trong syscall.h ( ID = 23 ).

A screen shot of a computer code

Description automatically generated

**Mục đích:** Ánh xạ số ID của systemcall với tên của systemcall ở dạng string. Mảng được dùng để kernel in tên của systemcall cần theo dỗi hoặc ghi log, dùng để truy vết hoạt động của các hàm.

**Ví dụ:** như đang thực thi thì kernel sẽ dùng mảng này để ghi tên systemcall ra log.

1. **Kernel/defs.h:**

**A black rectangular object with white text

Description automatically generated**

**Mục đích:** Khai báo hàm tính num of process và free memory.

**Ghi chú:** defs.h là file khai báo các hàm, macro và các cấu trúc dữ liệu được sử dụng trên toàn bộ hệ thống. Giúp liên kết các khai báo giữa các file trong kernel.

1. **Kernel/sysproc.c:**

A black rectangular object with white text

Description automatically generated

**Mục đích:** Khai báo file header sysinfo vì trong file này có một số định nghĩa quan trọng.

**Ví dụ:** struct sysinfo, hàm systemcall int sysinfo(struct sysinfo \*info), …

1. **Phần cài đặt:**
2. **Kernel/Proc.c:**

**A screen shot of a computer

Description automatically generated**

* **proc:** Là một mảng các cấu trúc tiến trình (struct proc) lưu trữ thông tin về tất cả các tiến trình trong hệ thống.
* **NPROC:** Là số lượng tối đa các tiến trình mà hệ điều hành có thể quản lý cùng một lúc.

Vì trong XV6, mỗi tiến trình được quản lý bởi một phần tử trong mảng proc.

* **Trạng thái UNUSED:**
  + Mỗi tiến trình trong hệ thống có một thuộc tính state để biểu thị trạng thái hiện tại (ví dụ: RUNNING, SLEEPING, UNUSED,...).
  + UNUSED là trạng thái của các slot tiến trình chưa được sử dụng (chưa được cấp phát cho bất kỳ tiến trình nào).
  + Chỉ những tiến trình không ở trạng thái UNUSED mới được tính vào procCount.

**Hoạt động**:

* Duyệt qua từng phần tử của mảng proc.
* Kiểm tra trạng thái của mỗi tiến trình.
* Tăng biến đếm nếu tiến trình không ở trạng thái UNUSED.

**Kết quả**:

* Trả về tổng số lượng tiến trình đang tồn tại hoặc hoạt động.

**Ý nghĩa**:

* Hỗ trợ giám sát hệ thống.
* Tích hợp trong system call hoặc dùng để log trạng thái kernel.

1. **Kernel/kalloc.c:**

**A screenshot of a computer

Description automatically generated**

**Cách làm:** đếm số lượng các khối bộ nhớ tự do trong danh sách kmem.freelist.

**Hoạt động**:

* + Duyệt danh sách liên kết để đếm số khối bộ nhớ tự do.
  + Tính tổng dung lượng trống bằng cách nhân số khối với kích thước của một trang (PGSIZE).
  + Trả về kết quả dưới dạng số byte.

**Ý nghĩa**:

* Cung cấp thông tin về trạng thái bộ nhớ hệ thống.
* Được sử dụng trong các chức năng giám sát và báo cáo trạng thái kernel.

1. **Kernel/sysproc.h:**

**A computer screen shot of a black screen

Description automatically generated**

**Cách hoạt động:**

* Nhận địa chỉ từ chương trình người dùng.
* Thu thập thông tin trong kernel.
* Sao chép kết quả ra không gian người dùng.

**Kết quả:**

* Trả về 0 nếu thành công, -1 nếu xảy ra lỗi.

**Mục đích**:

* Cung cấp thông tin về trạng thái hệ thống (bộ nhớ trống và số tiến trình) cho chương trình người dùng.