

Steps to Execute: Custom System Calls in xv6 (MIT RISC-V)

Overview

This document provides the complete step-by-step procedure to compile, run, and test the custom system calls implemented on the **MIT xv6 (RISC-V)** operating system. It also explains how each system call should be invoked and validated.

Environment Used

- Operating System Kernel: xv6 (MIT RISC-V version)
- Emulator: QEMU
- Toolchain: riscv64-linux-gnu (where applicable)

Step 1: Prepare the Build Environment

Navigate to the root directory of the xv6 project.

```
cd xv6-labs-2024
```

Ensure the repository is clean before building.

Step 2: Clean Previous Builds

Always clean before compiling to avoid conflicts.

```
make clean
```

Step 3: Compile and Launch xv6

Compile and launch xv6 using QEMU:

```
make qemu
```

If you are using a specific RISC-V toolchain:

```
make clean
```

```
make TOOLPREFIX=riscv64-linux-gnu-
```

```
make TOOLPREFIX=riscv64-linux-gnu- qemu
```

On success, xv6 boots and presents the shell prompt.

Step 4: Execute Implemented System Calls

Once xv6 is running in the QEMU terminal, the following commands can be executed.

1. Scheduler Statistics (schedtest)

Displays number of context switches since boot.

`schedtest`

Expected Output:

Total context switches since boot: <number>

2. Process Information (psinfo)

Prints all active processes with PID, state, and name.

`psinfo`

Expected Output:

PID	STATE	NAME
1	SLEEP	init
2	SLEEP	sh
3	RUN	psinfo

3. Memory Usage Statistics (memtest)

Executed through the memtest program.

`memtest`

Displays memory layout including:

- Code Size
 - Heap Size
 - Stack Size
 - Allocated Pages
 - Page Faults
-

4. Priority Based Scheduling (setpriority)

Modifies a process's execution priority.

```
setpriority <pid> <priority>
```

Examples:

```
setpriority 2 1
setpriority 2 10
setpriority 2 20
```

Test programs:

```
testpriority
prioritytest
```

Lower number means higher priority.

5. Process State Statistics (procstatetest)

Displays number of processes in major states.

```
procstatetest <pid>
```

Expected Output:

```
Process State Statistics:
RUNNING:    X
RUNNABLE:   Y
SLEEPING:   Z
```

Implementation Checklist

Component	Status	
Syscall wiring	✓	
Kernel logic	✓	
User wrappers	✓	
Output verified	✓	
Clean build	✓	

Important Notes

- Do not change syscall numbers unless required.
- Always run `make clean` before recompiling.
- Forgetting syscall registration may cause xv6 to freeze.
- All syscalls must be declared in:

-
- kernel/syscall.c
 - kernel/syscall.h
 - user/usys.pl
 - user/user.h
-

Conclusion

All five system calls were successfully compiled, executed, and verified on xv6 (MIT RISC-V). This document serves as a step-by-step execution guide and user reference for operating the implemented features.