1 Explanation

1.1 Trace

Trace is a system call that lets you monitor the calls a process makes to the kernel, providing insight into program behavior and helping to detect bugs or inefficiencies. It works as follows:

1. Whenever a system call is called, the OS stores the mask in main process structure.
2. On every syscall, if the corresponding bit is set, OS prints the process ID along with the details before returning to user mode.
3. It is then inherited by child process during fork.

1.2 Backtrace

A backtrace shows the function call chain at a specific point in execution. In debugging, it is critical to know how you reached a certain point in code.

1. Copying of s0 register into C variable is done by assembly code given.
2. From the frame pointer, the **return address** of the current function is found at `(fp - 8)`, and the **previous frame pointer** is found at `(fp - 16)`.
3. By repeatedly following the chain of frame pointers, the kernel can walk back through the call stack.
4. The traversal stops once the frame pointer goes outside the current kernel stack page.
5. Finally each return address is printed.

2 Implementation

2.1 Trace

1. Added trace in UPROGS in Makefile.
2. Made a new file trace.c and implemented the trace function.
3. Added `int trace(int mask)` as a new syscall in user.h.
4. Added an entry for trace in usys.pl.
5. Assigned number 22 for trace syscall in kernel/syscall.h.
6. Created a new variable trace_mask in proc.h.
7. Implemented the sys_trace function in sysproc.c which stored the value of trace_mask in mask.
8. Copied the trace_mask to child process in fork() int proc.c.
9. In syscall() in syscall.c, if current syscall is enabled, then print.

2.2 Backtrace

1. Read the frame pointer in riscv.h.
2. Implemented backtrace() in printf.c.
3. Added backtrace() in defs.h.
4. Calling backtrace in sys_sleep and panic() in printf.c
5. Adding bttest in UPROGS in Makefile and bttest.c in user.