Abanoub David Awad -- ada80

Nick Girardo -- nag82

Operating Systems Homework 1

Stage 1

We started by extending the xv6 kernel to support signal's so that we could create signals from the kernel, using mostly `sysproc.c` and `proc.h` as well as of course `trap.c`. Writing a `signal.h`.

Specifically we added an array of process specific signal handlers to the proc struct in addition to the definitions of **SIGFPE** and the `sighandler_t` function pointer.

Stage 2

The next part of the assingnment was written mostly in assembly as the task was simple enough to not need the luxuries of a high-level language like C. In fact writing in assembly eliminates the need to switch between C and assembly to fix the stack. We didn't need to worry about gcc eliminating our division by zero as well.

The specific implementation can found in the `stage2_timing.S` file. We modified the makefile to include a rule to build `stage2_timing.o` from `stage2_timing.S` and a rule to override the implicit rule for `stage2_timing.c`.

Stage 3

We had to edit our trap.c to deal with pushing the volatile registers unto the stack before we executed the user level signal handler as we needed a way to get those values back when restorer.h executes. We then popped the values in `restorer.h` after the user level handler has finished.

The minor changes can be found in `proc.h` and `sysproc.c` as well as changes to `trap.c`. With the new file being `restorer.h`.

Challenges

"One of my biggest challenges was all of the complications for adding functionality for signals. As we had to modify a lot of different files in a lot of specific ways, as well as some pointer casts that kept giving us warnings." - **David**

"We kept getting page faults when trying to pop the volatile registers in `restorer.h`"

Thank you