barkhausen institut

Debugging

Microkernel-based Operating Systems—Exercise

Barkhausen Institut, Dresden (Germany)

Viktor Reusch, Martin Küttler

WiSe 26/27

Fiasco Kernel Debugger

JDB

- Make sure Qemu is started with `-serial stdio` (the default in this repository).
- Add `default-kernel fiasco -serial_esc` to the top of your `modules.list`
- You can enter JDB by
  - Pressing escape at any time during the execution
  - Including this code:

    ```
    #include <l4/sys/kdebug.h>

    // somewhere in your code
    l4_kd_enter("message");
    ```

    For that your process needs the JDB capability (`jdb = L4.Env.jdb` in Lua). This will also name the process in the debugger.
- It is normal for one CPU to run at 100% in JDB (it polls for input).

JDB

Commands

- Most importantly: **h** – help
- **JS** – resize JDB to match terminal size
- **Q** – list kernel objects
  - Navigate with cursor keys
  - Select an object with enter for more information
  - For tasks & threads: S = address space, C = cpu, R = ref count
  - For IPC gates: L == label, D = owning thread
- **Esc** – Leave menus like the above
- **g** – Continue running

JDB

Commands continued

- **s** – list all tasks
- **lp**/**lr** – list all/ready threads
- In detailed thread view (after selecting a thread in Q, lp, lr):
  Space – disassembly
- **dt<task-id><address>** – memory dump
  - Space switches modes (big endian, little endian, ASCII)
  - e allows to edit the memory
  - u gives disassembly

IPC logging



- JDB can log all IPCs, i.e., log system calls
- **I*** – turn on IPC log
- **IR+** – turn on result log
- **O** – even more kernel-related logging
- **T** – view trace buffer (after running your code)
- Output format:

```
ipc: THR_ID TYPE->[C:CAP_DEST] DID=DEST_ID \
        L=LABEL [TAG] (MSG1, MSG2) TO=TIMEOUT
     THR_ID answ [TAG] L=FROM err=ERR.no (ERR.str) (MSG1,MSG2)
```

Here `MSG1` and `MSG2` are the first two words of the message. The `answ` lines are threads receiving (not necessarily answers).

Debugging with GDB



- Launch Qemu with `-s` to start GDB stub
- Consider passing `-S` to Qemu: It'll only boot after you type `continue` in gdb
- You can pass these options via an environment variable:
  `QEMU_OPTIONS="-s -S" make qemu`
- Connect from GDB with `target remote localhost:1234`
- Add symbol files in `obj/l4/amd64/bin/amd64_gen/l4f/.debug` (e.g. moe, my_pkg) and Fiasco:
  `add-symbol-file obj/fiasco/amd64/fiasco.debug`
- Considerations:
  - GDB won't know which address space you are in
  - Addresses of binaries might overlap
  - Consider dispersing binary addresses via `DEFAULT_RELOC`

Assignment 4

- Let’s have a look at the various debugging techniques
- Modify your printing client and service
  - Add the JDB capability to both tasks
  - Enter the kernel debugger start and end of the client
- Enter JDB and find the IPC gate object connecting the tasks
- Trace and observe the IPC messages between the tasks

Assignment 4

- Let’s have a look at the various debugging techniques
- Modify your printing client and service
  - Add the JDB capability to both tasks
  - Enter the kernel debugger start and end of the client
- Enter JDB and find the IPC gate object connecting the tasks
- Trace and observe the IPC messages between the tasks

- Use GDB to add a breakpoint to your server message handler
- Observe the handler’s parameters
- Step through the handler to observe its execution