

MIT OpenCourseWare  
<http://ocw.mit.edu>

6.189 Multicore Programming Primer, January (IAP) 2007

Please use the following citation format:

Phil Sung, *6.189 Multicore Programming Primer, January (IAP) 2007*.  
(Massachusetts Institute of Technology: MIT OpenCourseWare).  
<http://ocw.mit.edu> (accessed MM DD, YYYY). License: Creative  
Commons Attribution-Noncommercial-Share Alike.

Note: Please use the actual date you accessed this material in your citation.

For more information about citing these materials or our Terms of Use, visit:  
<http://ocw.mit.edu/terms>

# 6.189 IAP 2007

---

## Recitation 4

### Cell Debugging Tools

# Preparing for Debugging

---

- Two methods
  - Get program state on crash
  - Attach and step through program
- Compile for debugging
  - Use `gcc -g` or `xlc -g` to generate debugging info
  - or, in our Makefile:  
`CC_OPT_LEVEL = $(CC_OPT_LEVEL_DEBUG)`

# Running Processes Under GDB

---

- `ppu-gdb ./hello-world`  
`(gdb) run [args]`  
...  
`(gdb) quit`
- `export SPU_INFO=1`  
for extra information about threads

# Attaching to Running Programs

---

- `ppu-gdb ./hello -p 1234`  
`(gdb) continue`  
...  
`(gdb) detach`
- Finding the PID
  - `./hello &`  
`[1] 1234`
  - `ps -e | grep hello`  
`1234 pts/2 00:00:01 hello`
  - `top`

# Examining Program State

---

- Stack trace
  - `(gdb) bt`

```
#0  0x0f6a7fc8 in mmap () from /lib/libc.so.6
#1  0x0f2a62e0 in pthread_create@@GLIBC_2.1 () fro...
#2  0x0ff98168 in spe_create_thread () from /usr/l...
#3  0x01801bec in calc_dist () at dist.c:36
#4  0x01801cdc in main () at dist.c:55
```

# Examining Program State

---

- Examine variables
  - `(gdb) info locals`
- Evaluate expressions
  - `(gdb) print VARNAME`
    - `(gdb) print 'FILENAME'::VARNAME`
    - `(gdb) print 'FUNCTION'::VARNAME`
  - `(gdb) print EXPR`
    - Example: `(gdb) print x + 100 * y`
- gdb knows data types and prints values appropriately
  - To show type: `(gdb) whatis VARNAME`

```
id = {0x181e038, 0x1}
i = 1.2
```

# Examining Code

---

- View code at a specific location
  - `(gdb) list LINENUM`
  - `(gdb) list FUNCTION`
  - `(gdb) list FILENAME:FUNCTION`
- Display code above/below previous snippet
  - `(gdb) list`
  - `(gdb) list -`

```
21     calc_dist()  
22     {  
23         speid_t id[2];  
24  
25         // Set up different co...
```



# Controlling Program Execution

---

- Run to first line of main procedure
  - `(gdb) start`
- Next line in current procedure
  - `(gdb) next`
- Descend into function calls
  - `(gdb) step`
- Run until function exit, return to caller
  - `(gdb) finish`
- Resume execution until next breakpoint
  - `(gdb) continue`
- Cease debugging
  - Allow program to continue after gdb exits: `(gdb) detach`
  - Exit gdb: `(gdb) quit`

```
gen_points () at dist.c:67
67          srand(time(NULL));
```

# Breakpoints

---

- Halt program when a certain point is reached in execution
- Setting breakpoints
  - `(gdb) break FUNCTION`
  - `(gdb) break LINENUM`
  - `(gdb) break FILENAME:FUNCTION`
  - `(gdb) break FILENAME:LINENUM`
  - Conditional breakpoints:  
`(gdb) break ... if EXPR`
    - Example expression: `(x == 1 && y == 2)`
- Viewing or removing breakpoints
  - `(gdb) info breakpoints`
  - `(gdb) remove 2`

```
Breakpoint 2 at 0x1801740:  
file dist.c, line 70.
```

# Watchpoints

---

- Halt program when a value changes
- (gdb) watch VAR
  - watch myVar
  - watch myArray[6]

# Examining Memory

---

- (gdb) **x/Ni ADDR**
- **N** = how many units (machine words) to show
  - Default **N** = **1**
- Flag before address controls how to interpret data
  - **i**: machine instructions
  - **x**: hex
  - **d**: decimal
  - **a**: address (calculates offset from nearest symbol)
  - **f**: floating point numbers
  - **s**: string

# Examining Memory: Example

---

- `const char* a = "cell-processor\n";`
- Display as string
  - Note that count ("1") is by strings, not words
  - `(gdb) x/1s a`  
`0x10000bc0 <__dso_handle+4>: "cell-processor\n"`
- Display as hex
  - `(gdb) x/4x a`  
`0x10000bc0 <__dso_handle+4>:`  
`0x63656c6c 0x2d70726f 0x63657373 0x6f720a00`
  - `" c e l l - p r o c e s s o r \n \0"`

# Selecting Frames

---

- View state higher up in the call stack
  - Frame numbers are given by `bt`
  - `(gdb) frame 0`  
`(gdb) frame 1`  
`(gdb) frame 2`  
...
  - `(gdb) up`  
`(gdb) down`

# Debugging From emacs

---

- **M-x gdb** invokes gdb
  - Replace 'gdb' with 'ppu-gdb' when prompted
  - Specify executable path relative to *current buffer's directory*
  - Enter gdb commands in \*gud-...\* buffer
  - Active line in current frame is highlighted in editor
- Keyboard shortcuts available in source code files
  - Set breakpoint: C-x SPC
  - Print value of selected expression: C-x C-a C-p
  - Step: C-x C-a C-s
  - Next: C-x C-a C-n
  - Down frame: C-x C-a >
  - Up frame: C-x C-a <

# Exercise 1 (5 minutes)

---

- Find the value of control block (**cb**) in SPU thread
  - Get the recitation tarball
    - See example code in recitations section.
    - `tar xzf rec4.tar.gz`
  - Build the program
    - `cd rec4/dma-alignment/`
    - `make`
  - Run to the error with `ppu-gdb`
  - Debug



# Debugging Threaded Programs

---

- When a new thread is entered, gdb prints `[New Thread 123 (LWP 6041)]`
- List threads
  - `(gdb) info threads`
- gdb maintains 'current thread', used for `bt`, etc.
  - Switch threads: `(gdb) thread 2`
- On breakpoint or signal, gdb makes the triggered thread current

```
3 Thread 4151747792 (LWP 6042) 0x0f6ac0c8 in clone (...
* 2 Thread 4160398544 (LWP 6041) 0x000002f8 in main
(speid=25288760, argp=25269760, envp=0) at dist_spu.c:16
1 Thread 4160663552 (LWP 6038) 0x0f6ac0c8 in clone (...
```

## Exercise 2 (10 minutes)

---

- Verify that `cb` in the first SPU thread is the same as `cb[0]` in the PPU program
  - **You will need to qualify names**
  - **Build the program**
    - `cd rec4/lab1/`
    - `make`
  - **Set breakpoints, run and debug**
- Also examine the PPU thread state in Exercise 1 when the bus error occurs

```
typedef struct {
    uintptr32_t a_addr;
    uintptr32_t b_addr;
    uintptr32_t c_addr;
    uint32_t padding;
} CONTROL_BLOCK;
```

# Exercise 2

---

```
(gdb) break dist_spu.c:19
```

```
(gdb) run
```

```
(gdb) print cb
```

```
$1 = {a_addr = 25286272, b_addr = 25269248,  
res_addr = 25269888, padding = 0}
```

```
(gdb) thread 1
```

```
(gdb) print 'dist.c'::cb
```

```
$2 = {{a_addr = 25286272, b_addr = 25269248,  
res_addr = 25269888, padding = 0}, {a_addr =  
25286528, b_addr = 25269248, res_addr = 25278080,  
padding = 0}}
```

# Exercise 2

---

- Types are consistent with source code

```
(gdb) whatis cb
type = CONTROL_BLOCK
(gdb) whatis 'dist.c'::cb
type = CONTROL_BLOCK [2]
```

# Debugging Threaded Programs

---

- gdb can get confused by SPU threads
  - gdb removes breakpoint after first thread exits
  - gdb may complain about source files for SPU program
    - "No source file named dist\_spu.c. Make breakpoint pending on future shared library load? (y or [n])"
    - Choose "y" and continue, source should be visible later

# Debugging SPU Threads Alone

---

- Use spu-gdb to debug individual SPU threads
  - `SPU_DEBUG_START=1 ./hello &`
    - Prints PIDs of threads; threads wait for debugger to attach  
"Starting SPE thread 0x181e038, to attach debugger  
use: spu-gdb -p 1234"
  - `spu-gdb ./spu-hello -p 1234`
    - Attach gdb to SPU thread

# Troubleshooting Common gdb Issues

---

- Problem: gdb examines wrong variable when names are ambiguous
  - Use spu-gdb or rename variables
- Problem: breakpoints are deleted prematurely
  - Use spu-gdb or keep threads alive for as long as possible
- Error: "Thread Event Breakpoint: gdb should not stop!"
  - Use spu-gdb

# Errors that Debugger Can Help With

---

- "Bus error"
  - DMA transfer problem
  - Memory misalignment
- "Segmentation fault"
  - Invalid address
- Deadlock
  - Attach and examine state