Using IDA's GDB debugger with QEMU emulator

Copyright 2009 Hex-Rays SA

QEMU is a processor emulator which can emulate a handful of processors, including Intel x86 and ARM architectures. It includes a GDB stub which can be used with new GDB debugger plugin in IDA 5.4.

Getting QEMU

QEMU's home page is at <u>http://bellard.org/qemu/</u>. Win32 builds can be downloaded from Takeda Toshiya's page at <u>http://homepage3.nifty.com/takeda-toshiya/qemu/</u>. This primer assumes you downloaded QEMU 0.9.1 for Win32 or later. We will debug the small Linux included with QEMU.

Enabling GDB stub

After unpacking QEMU, make a copy of the <code>qemu-win.bat</code> file, for example <code>qemu-win-gdb.bat</code> and edit it. Add <code>-s -S</code> to the <code>qemu.exe</code> call (-s enables GDB stub and -S instructs QEMU to stop at the system start):

Ď qemu-win-gdb.bat - Notepad	
<u>File E</u> dit F <u>o</u> rmat <u>V</u> iew <u>H</u> elp	
SET QEMU_AUDIO_DRV=dsound	
REM QEMU_AUDIO_LOG_TO_MONITOR=1 displays log messages in QEMU monitor. SET QEMU_AUDIO_LOG_TO_MONITOR=0	
REM PCI-based PC(default): -M pc REM ISA-based PC : -M isapc REM -M isapc is added for NE2000 ISA card.	
qemu.exe -s -S -Lm 128 -hda linux.img -soundhw all -localtime -M p	c %* _

Run the .bat file. QEMU will stop and wait for the debugger.



Debugging with IDA

Start IDA.



If you get the welcome dialog, choose "Go".

	The interactive disassembler									
File	<u>E</u> dit	<u>J</u> ump	Searc	<u>h V</u> iew	Deb <u>ugg</u> er	Options	<u>W</u> indows	Help		
					Run	- +				
					Attach	•	Local Bochs	s debugger		
							Local Windo	ows debugger		
							Remote GD	B debugger		
							Remote Lin	ux debugger		
							Remote Ma	ic OS X debugge	r	
							Remote Syr	mbian debugger		
							Remote Wir	nCE debugger		
							Remote Wir	ndbg debugger		
							Remote Wir	ndows debugger	r	
							Remote iPh	ione v1.x debug	ger	nble
						_				
Ē										
Ind										
Auto				Diek						
Matte	,	P	own ji	Diak						

Choose Debugger | Attach | Remote GDB debugger.

Debug application setup: gdb		×
Debugger specific options		
Hostname localhost	▼ Port 1234	•
Save network settings as default		
0 <u>K</u> Cancel	Help	

Enter "localhost" for hostname and 1234 for the port number. Click "Debugger specific options".

GDB configurat	ion	×
Max packet size	2000 💌	
Timeout	1000 💌	
x86 options		
Use CS:IP	reakpoints at EIP+1 in real mode	
Processor	7	
ⓒ Intel x86 C ARM		
0 <u>K</u>	Cancel	Help

QEMU needs special configuration because it behave slightly differently from other GDB stubs. Uncheck "Software breakpoints at EIP+1" and check "Use CS:IP in real mode". Make sure Processor is set to "Intel x86". Click OK, then click OK in "Debug application setup" dialog.

Choose h	Choose process to attach to							
ID	Name	Γ						
0	<attach on="" process="" started="" target="" the="" to=""></attach>	<attach on="" process="" started="" target="" the="" to=""></attach>						
1	<enter a="" attach="" pid="" to=""></enter>							
OK	Cancel Help Search							

Choose <attach to the process started on target> and click OK.

Debugging the BIOS

	ID	A View-EIP									×
	•	MEMORY:000	FFFEC	db	0						
	•	MEMORY:000	FFFED	db	0						
	•	MEMORY:000	FFFEE	db	0						
	•	MEMORY:000	FFFEF	db	0						
	.	MEMORY:000	DFFFFØ	; -							
EIP	Ŀ	MEMORY:000	OFFFFØ	jmp)	fa	r ptr	3131h:(0F 0 0 0E 05	Bh	
		MEMORY:000	OFFFF0	ş							
	•	MEMORY:000	FFFF7	db	2Fh	Ç.	1				
	•	MEMORY:000	OFFFF8	db	3 Øh	;	0				
	•	MEMORY:000	FFFF9	db	31h	ş	1				
	•	MEMORY:000	FFFFA	db	2Fh	ş	1				
	•	MEMORY:000	FFFFB	db	3 Øh	ş	0				
	•	MEMORY:000)FFFFC	db	36h	ş	6				
	•	MEMORY:000	OFFFFD	db	0						
	•	MEMORY:000	FFFFE	db	ØFCh	Ş.	ь				
	•	MEMORY:000	FFFFF	db	ØF7h	Ş.	ч				
	•	MEMORY:001	00000	db	0						-
		•									▶
٠ŧ		UNKNOWN	00FFFF0:	MEM	IORY:00	OFF	FF0				

The BIOS entrypoint is displayed but the disassembly is wrong since the default MEMORY

segment is 32-bit.

9	ID	A - C	:\WIN	DOWS\1	TEMP\i	da74	4146.id	b (<gdb< th=""><th>remote p</th><th>rocess>)</th></gdb<>	remote p	rocess>)
Fi	le	Edit	Jump	Search	View	De	bugger	Options	Windows	Help
General registers					(Debugg Breakpo Watche Tracing	ger windov oints es	vs	* * * *	
	c I: id	le	C	Down D	isk: 419	Þ	Continu Attach	ie process to process	; 5,,,	F9
] ID •	A Vie MEN	w-EIP	000FF	FEC d	11	Process Pause p	orocess		
	:	MEN MEN	10RY : 10RY : 10RY :	000FF 000FF 000FF	FED d FEE d FEF d		Termina Detach	from proces	iess iess	Ctrl+F2
EIF			IORY :	000FF	FF0 ; FF0 j		Send a Parse N	command lach-O sy	to GDB mor mbols	nitor
	•	MEN	IORY :	000FF	FF5 d		Manual	memory r	egions	
		MEN MEN	IORY : IORY :	000FF 000FF	FF6 d FF7 d	B3 BD	Step ini	to /er		F7 58
	•	MEN MEN	IORY : IORY :	000FF 000FF	FF8 d FF9 d		Run un	til return		Ctrl+F7
		MEN MEN MEN	10RY : 10RY : 10RY :	000FF 000FF 000FF	FFA d FFB d FFC d	<u>n</u>	Run to	cursor per options	s	

Let's create some manual memory regions to reflect the real memory map.

Go to Debugger | Manual memory regions.

Manual memory regions			×
Edit Search			
Start End	Base Name	Class X B	lits
	Insert Ins		
	Delete Del		
	Pafresh Ctrl+1		
	Copy Ctrl+Ins		

Right-click and choose "Insert..." (or press Ins).

Add debugge	r memory region	×			
Please specify	a new memory region				
<u>S</u> tart address	F0000	•			
<u>E</u> nd address	100000	-			
<u>B</u> ase address	F000	-			
<u>N</u> ame	BIOS	•			
<u>C</u> lass	CODE	-			
 ○ 16-bit seg ○ 32-bit seg ○ 64-bit seg 	gment gment gment				
	0 <u>K</u> Cancel				
Enter the following details: Start address: F0000					

Start address: F0000 End address: 100000 Base address: F000 Name: BIOS Class: CODE select "16-bit segment" Click OK.



Now the disassembly is correct and you can trace the BIOS code.

Debugging the kernel

To debug the Linux kernel, we first need to create memory regions where it will execute. The usual kernel entrypoint is at address 100000, and it can use memory almost to the maximum 4G address.

Open memory regions list (Debugger | Manual memory regions) and add a new region:

Edit debugge	Edit debugger memory region						
Start address	100000	.					
<u>End</u> address	F0000000						
<u>B</u> ase address	0	•					
<u>N</u> ame	LINUX	-					
<u>C</u> lass	CODE	•					
 ○ <u>1</u>6-bit seg ○ <u>3</u>2-bit seg 	ment Executable						
	0 <u>K</u> Cancel						

Start address: 100000 End address: F0000000 Base address: 0 Name: LINUX Class: CODE select "32-bit segment" Click OK.

We now have two regions:

Manual memory regions							
Edit Search							
Start	End	Base	Name	Class	X	Bits	
000F0000	00100000	F000	BIOS	CODE	Х	16	
00100000	F0000000	0000	LINUX	CODE	Х	32	
1							
Line 2 of 2							///

Double-click the LINUX region to go to its beginning.

IDA View-EIP	
LINUX:00100000 ; LINUX:00100000 L	; Segment permissions: Read/Execute INUX segment byte public 'CODE' use3
LINUX:00100000 a LINUX:00100000 ;	org 100000h
* LINUX:00100000 c	Ib Otol Byte 0
* LINUX:00100002 c * LINUX:00100003 c	Ib (DAT Word 0 Ib (DAT Double word 0
LINUX:00100004 c LINUX:00100005 c LINUX:00100005 c	Ib Synchronize with Ib G Jump to IP
• LINUX:00100007 c	1b (∰+ Set IP Ctrl+N 1b (🖁 Add breakpoint F2
* LINUX:00100009 c * LINUX:0010000A c	1b 47 W Add write trace 1b 47 W Add read/write trace
• LINUX:0010000C c	ib and Add watch
UNKNOWN 00100000:	LINUX:00100000

Press F2 or choose "Add breakpoint" from the context menu.

Breakpoint settings			
Address 0x00100000			
☑ Enabled ☑ Hardware breakpoint			
Hardware breakpoint settings			
Modes : C <u>R</u> ead/Write C <u>W</u> rite C E <u>x</u> ecute			
Condition			
Actions			
🔽 Break 🔲 Trace			
<u>Cancel</u>			

Check "Hardware breakpoint" and select "Execute" in "Modes". Click OK.

IDA View-EIP		×
LINUX:00100000	; Segment permissions: Read/Execute	
LINUX:00100000	LINUX segment byte public 'CODE' use3	
LINUX:00100000	assume cs:LINUX	
LINUX:00100000	;org 100000h	
LINUX:00100000	assume es:LINUX, ss:LINUX, ds:LINUX,	
LINUX:00100000	db <mark>8</mark>	
* LINUX:00100001	db <mark>0</mark>	
* LINUX:00100002	db <mark>0</mark>	
* LINUX:00100003	db <mark>0</mark>	
* LINUX:00100004	db <mark>0</mark>	
* LINUX:00100005	db <mark>0</mark>	
* LINUX:00100006	db <mark>0</mark>	
* LINUX:00100007	db <mark>0</mark>	
* LINUX:00100008	db <mark>0</mark>	
* LINUX:00100009	db <mark>0</mark>	
* LINUX:0010000A	db <mark>0</mark>	
* LINUX:0010000B	db <mark>0</mark>	
* LINUX:0010000C	db <mark>0</mark>	-
•		▶
UNKNOWN 0010000	0: LINUX:00100000	

Now press F9 or choose Debugger | Continue process. You should see BIOS and LILO messages on the screen, and the execution will stop at the "Loading Linux...." message.

📑 ID	A View-EIP		
	LINUX:0010000	0	_
EID.	LINUX:0010000	0 loc_100000:	; DAT 💻
EIP •	LINUX:0010000	0 cld	
•	LINUX:0010000	1 cli	
	LINUX:0010000	2 mov eax, 18h	
	LINUX:0010000	7 mov ds, ax	
	LINUX:0010000	9 assume ds:nothing	
	LINUX:0010000	y moves, ax	
	LINUX:0010000	B assume es:nothing	
1 1	LINUX:0010000	B mov +s, ax	
	LINUX:0010000	v assume +s:notning	
	LINUA:0010000	v mov gs, ax E accura accorthing	
	LINUA:0010000	Flassume ys:nutniny Flass acc. dc:100000b	
	LINUA:0010000	F 155 esp, us: 1030200	
	LINUX-0010001	o xur eax, eax	
	LINUX-0010001	0 9 loc 100019•	• con
	I TNUX - 0010001		,
	I TNUX - 0010001	9 mou large ds:8 eax	
	L TNUX : 0010001	F cmn ds:100000b, eax	
•	L TNUX : 0010001	4 iz short loc 100018	
	L TNUX : 0010002	6 nush	
•	LINUX:0010002		
•	LINUX:0010002	9 xor eax. eax	
•	LINUX:0010002	B mov edi, offset unk 1B69CF	-
	•	· · ·	
		007: TNUX:00100007	

This is the initial loader which decompresses the kernel. If you press F9 once more, you'll see "Uncompressing Linux..." and then "Ok, booting the kernel" messages in QEMU and IDA will stop at the decompressed kernel entrypoint:

📑 ID	A View-EIP			×
	LINUX:00100000	assume	es:LINUX, ss:LINUX, ds:LINUX, fs:LINUX	•
EIP 🔸	LINUX:0010000	cld		
•	LINUX:00100001	mov	eax, 18h	
•	LINUX:00100006	mov	ds, ax	
	LINUX:00100008	assume	ds:nothing	
•	LINUX:00100008	mov	es, ax	
	LINUX:0010000A	assume	es:nothing	
•	LINUX:0010000A	MOV	fs, ax	
	LINUX:0010000C	assume	fs:nothing	
•	LINUX:0010000C	MOV	gs, ax	
	LINUX:0010000E	assume	gs:nothing	
•	LINUX:0010000E	MOV	edi, offset unk_102000	
•	LINUX:00100013	MOV	eax, 7	
	LINUX:00100018			
	LINUX:00100018	10c_10	9018: ; COD	
	LINUX:00100018	stosd		
	LINUX:00100019	add	eax, 1000h	
	LINUX:0010001E	cmp	edi, offset unk_104000	
L -	LINUX:00100024	jnz	short loc_100018	
	LINUX:00100026	MOV	eax, offset off_101000	
1 1	LINUX:0010002B	MOV	cr3, eax	
1 1	LINUX:0010002E	MOV	eax, cru	
	LINUX:00100031	or	eax, offset unk_80000000	
	LINUX:00100030	mov	cru, eax	
	LINUA:00100039	յար	SHUFL \$+2	
	LINUA:00100038	imp	eax, offset unk_90100042	
	LINUA:00100040	Tub	eax	
	LTNUX - 00100040	, db 0El		
				•
	•			F
	UNKNOWN 001000	A: LINUX:00	010000A	

This code sets up paging table, enables paging, and then jumps to the "real" kernel entrypoint.

Adding symbols

Kernel symbols are available as /proc/ksyms or /proc/kallsyms pseudo-file after booting. If you get that file from the VM to the host, you can add the symbols to your disassembly. Go to File | Python command... and enter the following short script:

```
ksyms = open(r"D:\ksyms") #path to the ksyms file
for line in ksyms:
   addr = int(line[:8], 16)
   name = line[9:-1] # use line[11:-1] in case of kallsyms
   idaapi.set_debug_name(addr, name)
   MakeNameEx(addr, name, SN_NOWARN)
   Message("%08X: %s\n"%(addr, name))
```

Click OK and wait a bit until it finishes. After that you should see the symbols in the dissassembly and name list:

Choose a name	<u>_ </u>
Name	Address P., 🔺
D gdt	9010022C
D empty_zero_page	90104000
D disable_hlt	90107020
D enable_hlt	90107030
D default_idle	90107040
D machine_real_restart	90107110
D machine_restart	901071D0
D machine_halt	90107250
D machine_power_off	90107260
D dump_thread	90107670
D get_wchan	90107990
Ddown_failed	90107D9C
Ddown_failed_interruptible	90107DA8
Ddown_failed_trylock	90107DB4
D_up_wakeup	90107DC0
D dump_stack	901094E0
D disable_irq	9010A650
OK Cancel Help	Search
Line 1 of 1212	11.

Happy debugging!

Copyright 2009 Hex-Rays SA