Debug for GDB Users

Basic Process Control

To be useful, a debugger must be capable of basic process control. This functionally allows the user to create a debugging session and instruct the process what to do next.

Action	Description	Debug	GDB
Creating	Creating a debug session for a program	$debug <program> <args> >create <program> <args> (note that the program arguments are provided here on process creation)	$gdb <program> >file <program>
Grabbing	Grabbing an already executing program	$debug <pid> >grab <pid>	$gdb program <pid> >attach <pid> (use the file command to load the program if attach cannot find the program in your search path)
Releasing	Release a debugged program	>release	>detach
Core	Analyzing a core file generated by a buggy program	$debug -c <core> <program> >grab -c <core> <program>	$gdb <program> <core> >target core <core>
Executing	Executing a program	>run	>run <args> (note that ther progran arguments are given now)
Halting	Halting a program that is executing under the control of the debugger	<Ctrl-C> >halt	<Ctrl-C>
Continuing	Continuing a program execution after it has been halted	>run	>cont
Stepping	Continue program execution to the next line of source and follow a function call if necessary	>step	>step

Next	Continue executon to the next line of source and do not follow a function call	>step -o >next	>next
Return	Continue execution until the end of the current function	>run -r	>finish
Machine code step	Continue execution to the next machine code instruction and follow a function call if necessary	>step -i >si	>stepi >si
Machine code next	Continue execution to the next machine instruction and do not follow a function call	>step -io >ni	>nexti >ni
Terminating	Stopping an program and then terminating its execution	<Ctrl-C> (if no command prompt) >kill	>kill
Quitting	Exiting the debugger	>quit	>quit

Basic Process Manipulation

Process manipulation, the core of a debugger, indicates when a process is to stop execution. This break in execution allows the user to examine the process state at known locations or when data is altered or accessed.

Action	Description	Debug	GDB
Breakpoints	Setting a breakpoint at a specific program location, halt program execution at that point	>stop <expression> >stop <filename@line>	break <expression> >break <filename:line>
Temprary breakpoints	Setting a breakpoint that is cleared after being reached once	N / A	>break <expresson>
Regular expression breakpoint	Setting a breakpoint on all functions that match the regular expression	N / A	>break <regex>
Watchpoint	Set a breakpoint that is activated when the value of a variable ot memory area is changed	>stop *(expr)	>watch <expression>
Breakpoint list	Provide a list of set	>stop	>info breakpoints >info break

	breakpoints		>info watchpoint

Events and Signals

Program execution can generate events a debugger can recognize. Some of these events are forks, execs, throw, catch and signals. Signals all a program to respond via a handler.

Action	Description	Debug	GDB
Signal list	List the debugger's approach to handling signals	>signal	>info signals
Signal ignore	Ignore a signal and pass it straight to the program	>signal -io <signal>	>handle <signal> nostop
Signal intercept	Intercept the signal before it is passed onto the program and halt program execution	>signal <signal>	>handle <signal> stop

Process Thread Control

A process can sometimes have simultaneous, different paths of execution. This capability is particularly useful for multi-processor machines. Thread control allows the user to deal with these different execution paths.

Action	Description	Debug	GDB
Thread list	Provide information about the available threads	>ps	>info threads
Thread switch	Switch to a selected thread	>set %thread <thread no.>	>thread <thread no.>
Thread command	Applying a debugger command to a thread, list of threads or all	>command -p <thread no.> [,<thread no.>]* >command -p all	>thread apply <thread no>. <args> >thread apply all <args>

Multiple Processes

Debug is able to debug several processes at once. This feature is particularly useful if a program undergoing debugging consists of several independently running processes. Unfortunately GDB can debug onl one process at a time.

Action	Description	Debug	GDB
Process list	List the current processes being debugged	>ps	N / A
	Execute a debugger		

Process specific command	command for a specific process only	><command> -p <proc no>. | all <args>	N / A
Process current	Make a particular process the current process	>set %proc <proc no.>	N / A

The Stack

To determine the function call path to the current position in the program, the user needs to examine the stack. A stack consists of frames, and each frame is associated with a function call. Typically, the user needs to view the full stack and possibly navigate through the stack. A user is able to examine each stack frame's register values, function call arguments, and local variables.

Action	Description	Debug	GDB
Stack trace	Display the framces of the stack	>stack >t	>backtrace >bt
Change current frame	Change the current frame to another in the stack	>set %frame <frame no.>	frame <frame no.>
Up	Move up the stack	>set %frame %frame - 1	>up
Down	Move down the stack	>set %frame %frame + 1	>down
Local Variables	Display the values of the local variables	>symbols -l	>info local
Functionarguments	Display the vlaues of the function arguments	>stack -c 1	>info args

Additional Commands

The commands shown next can help you complete the debugging commands mentioned previously.

Action	Description	Debug	GDB
List Source	List the program source	>list >list <expr>	>list >list <expr>
Register values	Display the values of the registers of the current stack frame	>regs	>info registers >info all-registers
Register names	The names of the registers that can be used in debugging expressions	%<register>	$<register>
Disassembly	List machine code	>dis	>assemble

	instructions		