Application Debugging for Itanium: HP WDB

Carl Burch
Software Engineer
Hewlett-Packard

© 2004 Hewlett-Packard Development Company, L.P.
The information contained herein is subject to change without notice

Overview of debugging techniques...

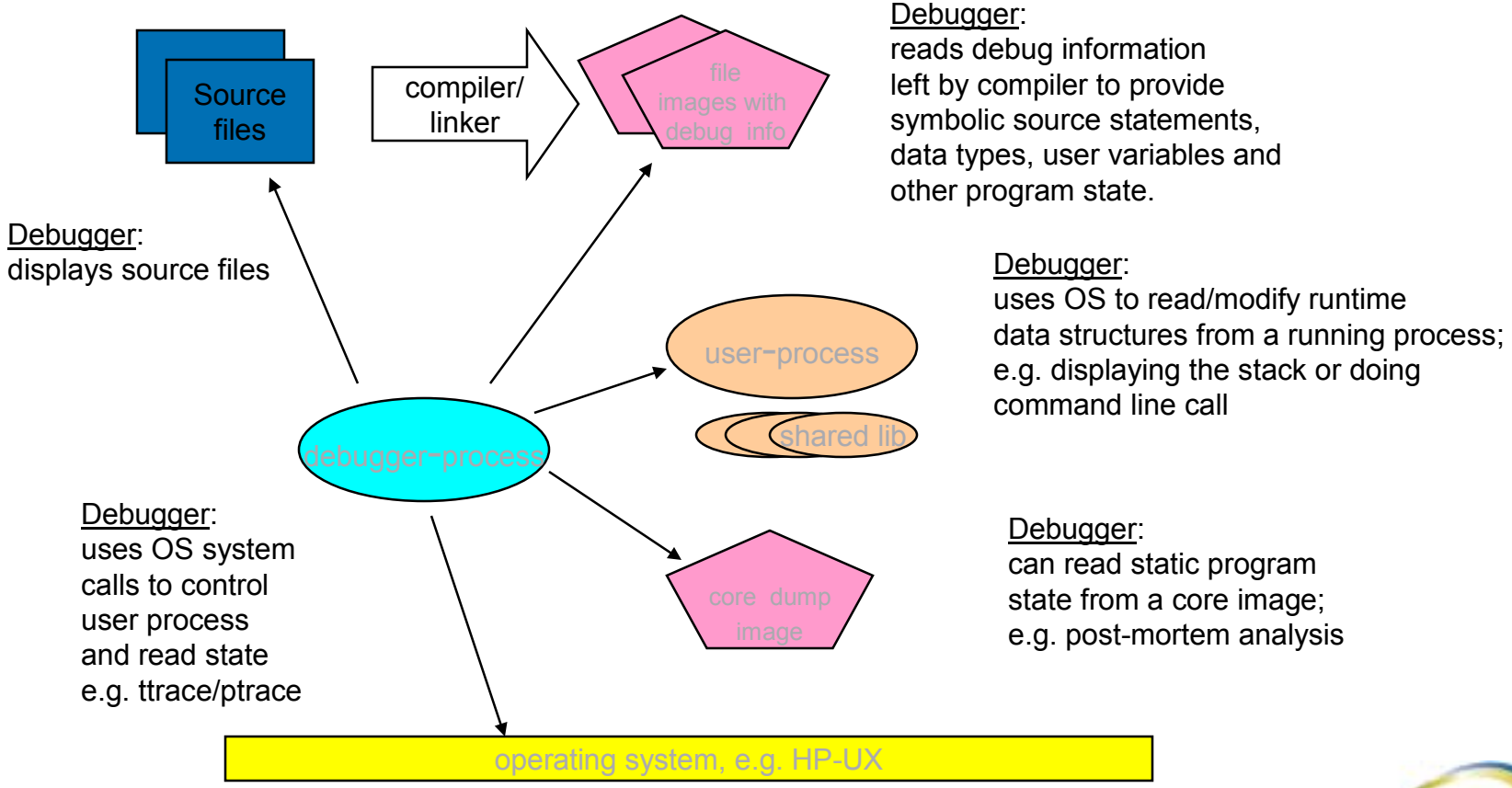
- Assertions: add consistency checks within the code, check for cases that should never happen.
- Diagnostics: add logging and dump facilities, pretty print data structures; use tools that help create diagnostics, e.g. Purify™ to look at memory issues
- Comparisons: use a case that works and one that doesn't and eliminate differences that don't matter
- Post-mortem analysis: start with the point of failure, work backwards to the cause.
- Divide and conquer: break problem into small parts, check the state after individual steps complete.

A debugger can assist with these techniques.

WDB Availability

- Recommended debugger on HP-UX (PA & IPF)
- Based on open source GNU debugger (gdb) with lots of HP value adds.
- Best spot to pick up most recent wdb is from the web:
  http://www.hp.com/go/wdb
- Majordomo lists to keep up to date with wdb
  wdb-announce@cxx.cup.hp.com – used to announce new versions
  hpux-devtools@cxx.cup.hp.com – used for discussion of tools
  cxx-dev@cxx.cup.hp.com is for discussion of C++ development
- wdb is bundled with C, C++ and Fortran compilers
- Web releases include both pre-built binaries and source code (gdb)

Debugger overview



WDB Interfaces: GDB command line

```
Terminal
anteater:mev[130]$ gdb hello
HP gdb 1.2.01
Copyright 1986 - 1999 Free Software Foundation, Inc.
Hewlett-Packard Wildebeest 1.2.01 (based on GDB 4.17-hpwdb-980821)
Wildebeest is free software, covered by the GNU General Public License, and
you are welcome to change it and/or distribute copies of it under certain
conditions.  Type "show copying" to see the conditions.  There is
absolutely no warranty for Wildebeest.  Type "show warranty" for details.
Wildebeest was built for PA-RISC 1.1 or 2.0 (narrow), HP-UX 10.20.
..
(gdb) break main
Breakpoint 1 at 0x321c: file hello.c, line 5.
(gdb) run
Starting program: /tmp_mnt/home/vobadm/mev/hello

Breakpoint 1, main () at hello.c:5
5         printf("hello new world\n");
(gdb) n
hello new world
6         for (i=0;i<100;i++){
(gdb)
```

Gdb: underlying debugger engine

WDB Interfaces: Vdb

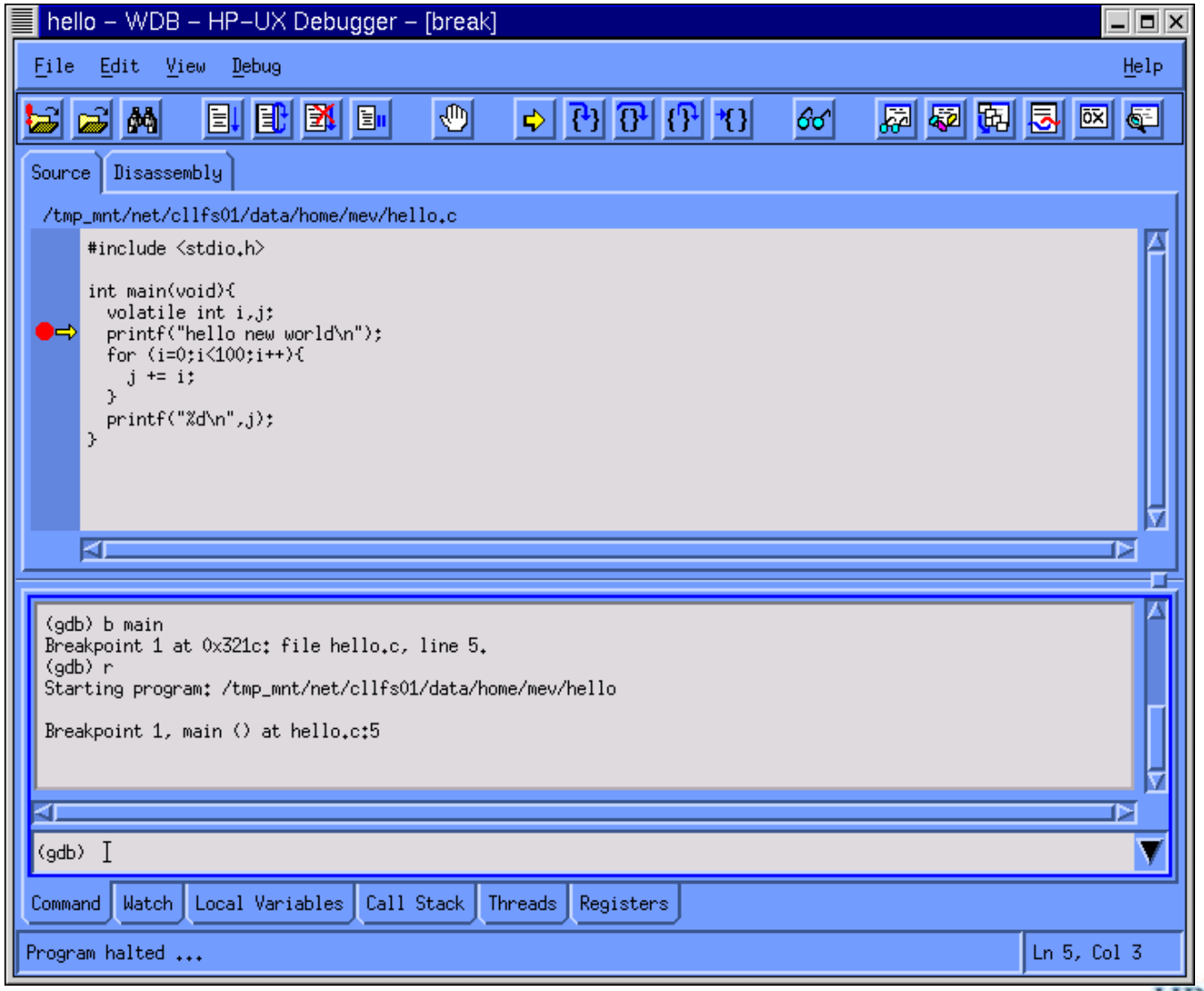
```
vdb : hp-ux debugger                                   [F1 for help]
    102 VimMain
    103 # else
    104 main
    105 # endif
    106 (argc, argv)
    107     int         argc;
    108     char        **argv;
    109 {
    110     char_u      *initstr;               /* init string from environment */
*>  111     char_u      *term = NULL;           /* specified terminal name */
    112     char_u      *fname = NULL;          /* file name from command line */
    113     char_u      *tagname = NULL;        /* tag from -t option */
    114     char_u      *use_vimrc = NULL;      /* vimrc from -u option */
    115 #ifdef FEAT_QUICKFIX
    116     char_u      *use_ef = NULL;         /* 'errorfile' from -q option */
    117 #endif
    118 #ifdef FEAT_CRYPT
    119     int         ask_for_key = FALSE;    /* -x argument */
    120 #endif
    121     int         n_commands = 0;         /* no. of commands from + or -c */
File: main.c    Function: main    Line: 111    Pc: 0x9e360
Copyright 1986 - 2001 Free Software Foundation, Inc.
Hewlett-Packard Wildebeest 3.3 (based on GDB) is covered by the
GNU General Public License. Type "show copying" to see the conditions to
change it and/or distribute copies. Type "show warranty" for warranty/support.
..here
Breakpoint 1 at 0x9e360: file main.c, line 111.

Breakpoint 1, main (argc=3, argv=0x7aff05bc) at main.c:111
111         char_u      *term = NULL;           /* specified terminal name */
(wdb)

Run   Resume  Step  Up   vdb   Finish  Print   Type   List
Faq   Stop    Next  Down       Prompt  Print*  Edit   Credits
```

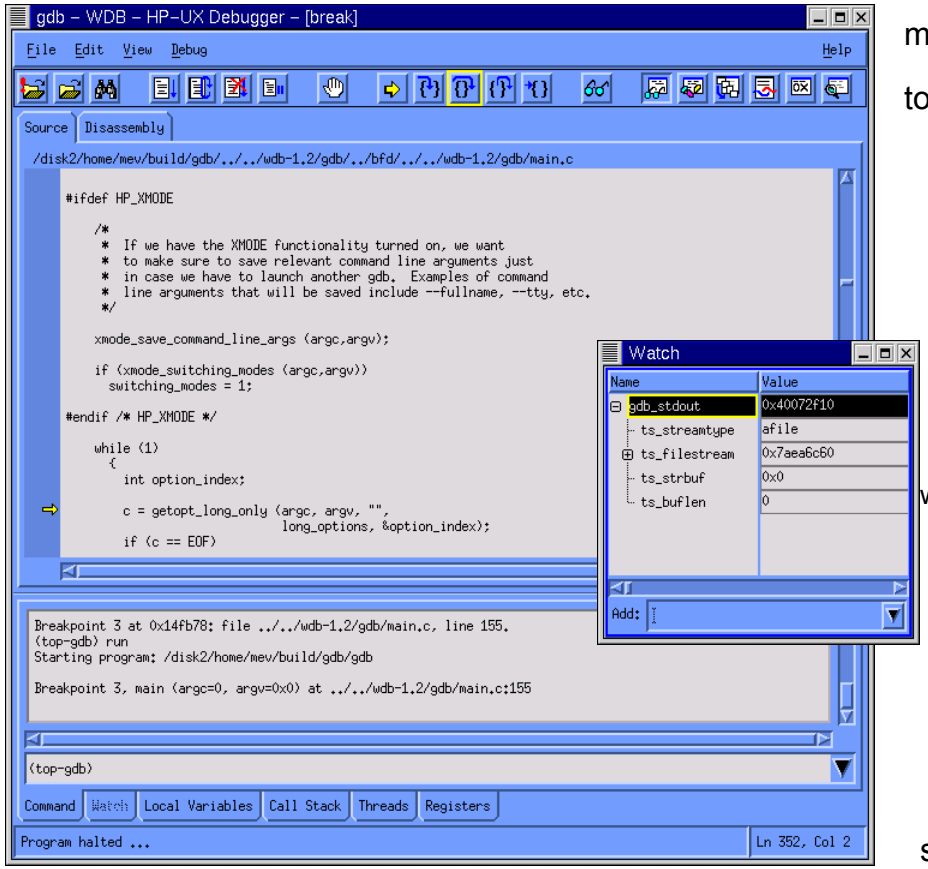
Vdb: replacement for –tui on IPF

WDB Interfaces: WDB



WDB: HP supported GUI

WDB GUI Interface



menu bar

tool bar

assembly display

source display

watch window

command prompt and transcript

views on tabs or popups:
commands, watch, locals, stack, threads, registers

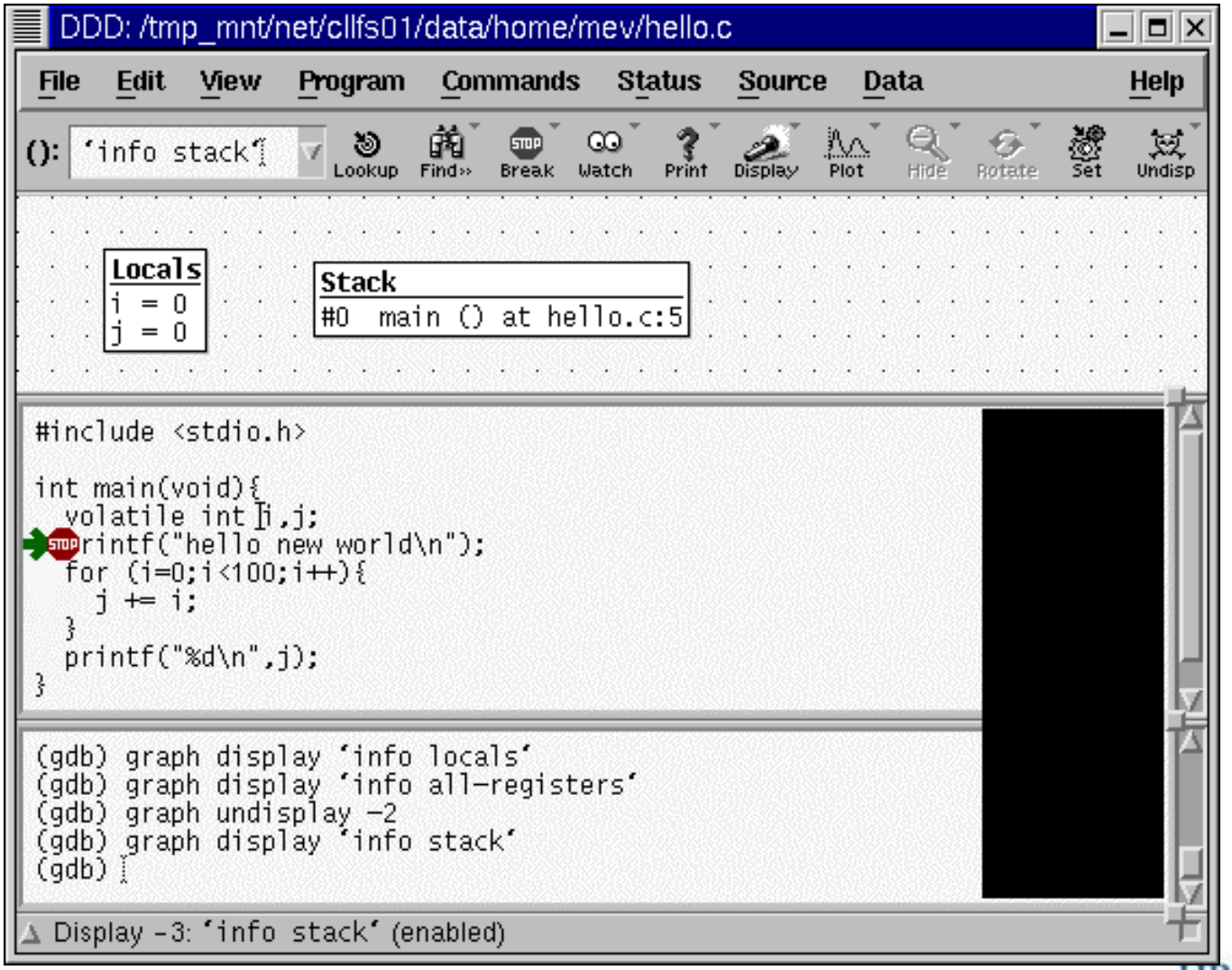
status line

wdb gui highlights

- integrated with wdb plans:
  e.g. fix 'n continue, memory check
- supported by HP

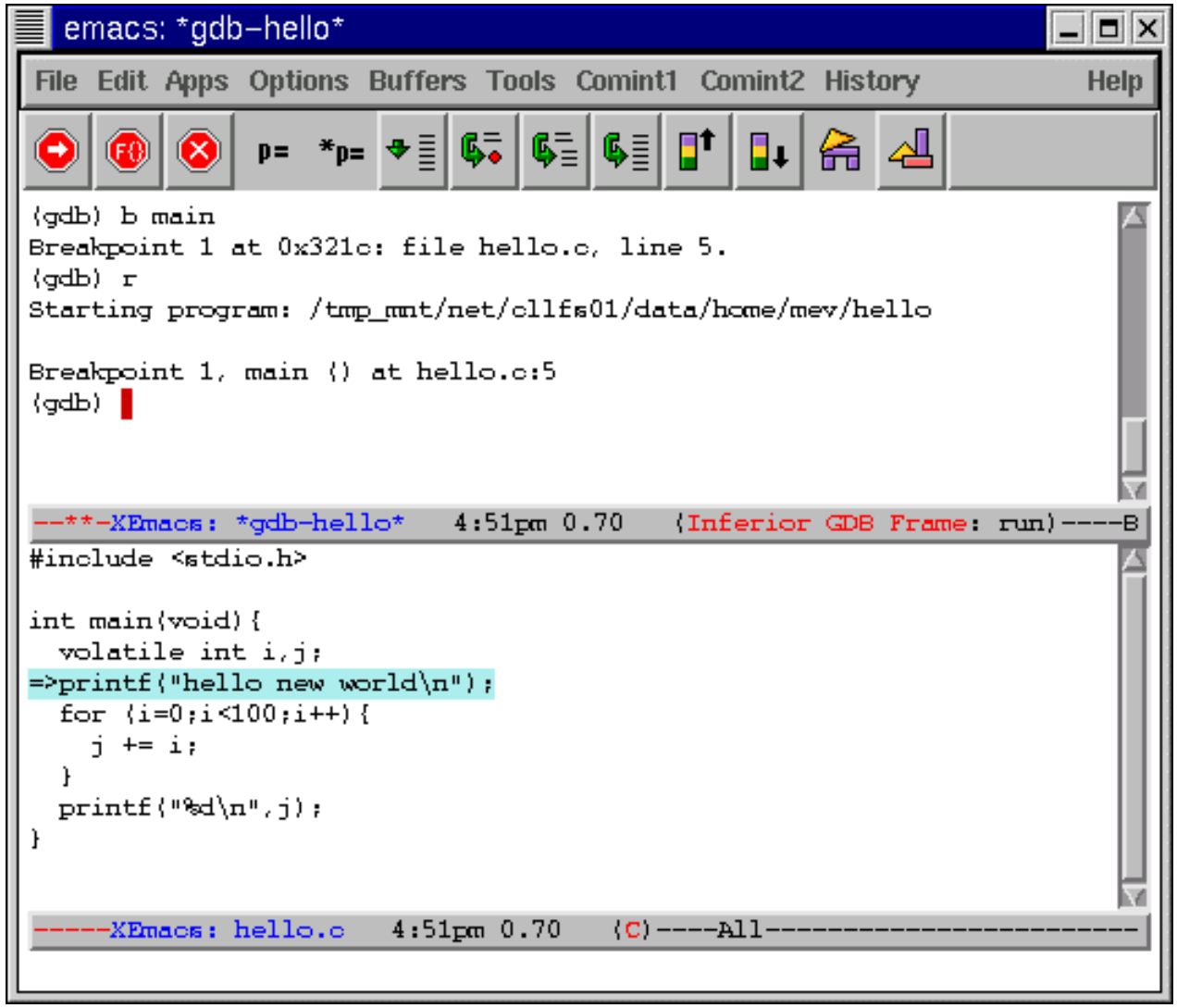
- PC-like look
- configurable, sessions

WDB Interfaces: Ddd



Ddd: popular gdb GUI

WDB Interfaces: Emacs



Emacs: M-x gdb

WDB Interfaces: Firebolt

Firebolt 2.3 development environment for hp-ux.

File Edit Tools Buffers Window Debug Clearcase Help

Lookup Functions

Source Assembly

```
2833            else
2834                gui_mch_stop_blink();
2835            focus = gui.in_focus;
2836        }
2837
2838        /*
2839         * Don't use gui_mch_update() because then we will spin-lock until a
2840         * char arrives, instead we use XtAppProcessEvent() to hang until an
2841         * event arrives.  No need to check for input_buf_full because we are
2842         * returning as soon as it contains a single char.  Note that
2843         * XtAppNextEvent() may not be used because it will not return after a
2844         * timer event has arrived -- webb
2845         */
2846        XtAppProcessEvent(app_context, desired);
2847
2848        if (!vim_is_input_buf_empty())
2849        {
2850            if (timer != (XtIntervalId)0 && !timed_out)
2851                XtRemoveTimeOut(timer);
2852            return OK;
2853        }
2854    }
2855    return FAIL;
2856 }
2857
2858 /*
2859  * Output routines.
```

File: gui_x11.c Function: gui_mch_wait_for_chars Line: 2846 Pc: 0x15cfb4

```
(wdb) up
#5  0x7ad1d290 in XtDispatchEvent+0x108 () from /lib/libXt.3
(wdb) up
#6  0x7ad2baa0 in XtAppProcessEvent+0x320 () from /lib/libXt.3
(wdb) up
#7  0x15cfcc in gui_mch_wait_for_chars (wtime=4000) at gui_x11.c:2846
2846            XtAppProcessEvent(app_context, desired);
(wdb) down
#6  0x7ad2baa0 in XtAppProcessEvent+0x320 () from /lib/libXt.3
(wdb) up
#7  0x15cfcc in gui_mch_wait_for_chars (wtime=4000) at gui_x11.c:2846
2846            XtAppProcessEvent(app_context, desired);
(wdb)
```

Debug Tear-off

Step
Next
Resume
Up
Down
Finish
Print
Print*
Type
List
Goto Prompt

Firebolt – Vim based Edit-compile-debug tool

Common invocation modes

- gdb, wdb and related binaries are stored in /opt/langtools/bin

  $ gdb program → Load program into debugger

  $ gdb program core → Post mortem analysis of the core image

  $ gdb program 1234 → attach to running process 1234

- Invoking the above with wdb instead of gdb, brings up the GUI

- Useful command line options
  - -xdb → XDB compatibility mode; many <u>but not all</u> XDB commands accepted
  - -dbx → dbx compatibility mode; many <u>but not all</u> dbx commands accepted
  - -version → display version information and quit.

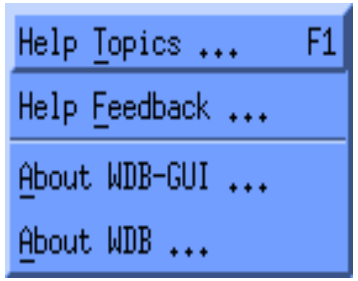
Getting help

(gdb) help

provides a list of commands

$gdb --help

provides a list of arguments



Online reference information shipped with wdb in /opt/langtools/wdb/doc:

- quick reference card & quick start card
- gdb manual
- online help for GUI
- tutorials and xdb transition guide
- emacs info files

WDB Debugger Basics

Most important simple commands to know:

- Commands for wdb control [run] [quit] [attach]
- Commands for breakpoints [break] [step] [next] [continue]
- Commands for watching data [watch]
- Commands for printing values [print] [x] [bt] [call]
- Commands for getting help [help] [info][set][show]

- Knowing just these commands, one can get remarkably far in using wdb
- Commands are common in all interfaces, though GUIs often have other (mouse, menus, panes) ways of doing these basic operations.

Specifying target

(gdb) run *[program arguments…]*

Tip #1: Use (gdb) set args *[arguments…]* to set program arguments

Tip #2: Use (gdb) set env *variable value* to set environment variables

Tip #3: Use a .gdbinit file if you want to repeat the same arguments or environment; create one per application and/or per user

Note #4: Arguments are processed using a shell, usually csh

(gdb) attach *pid*

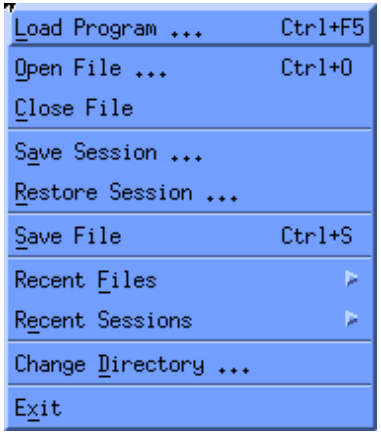
Tip #5: Use "(gdb) file *filename*" to set the name of the executable image

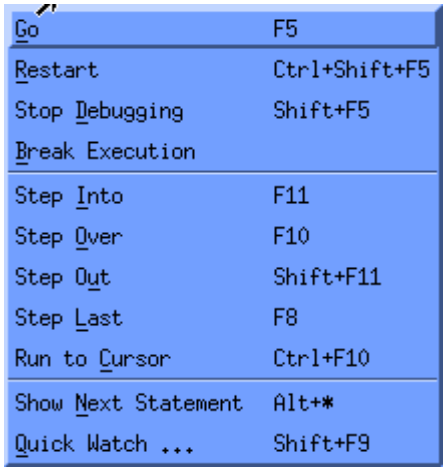
Tip #6: HP-UX doesn't allow attaching on an NFS file system, see workarounds

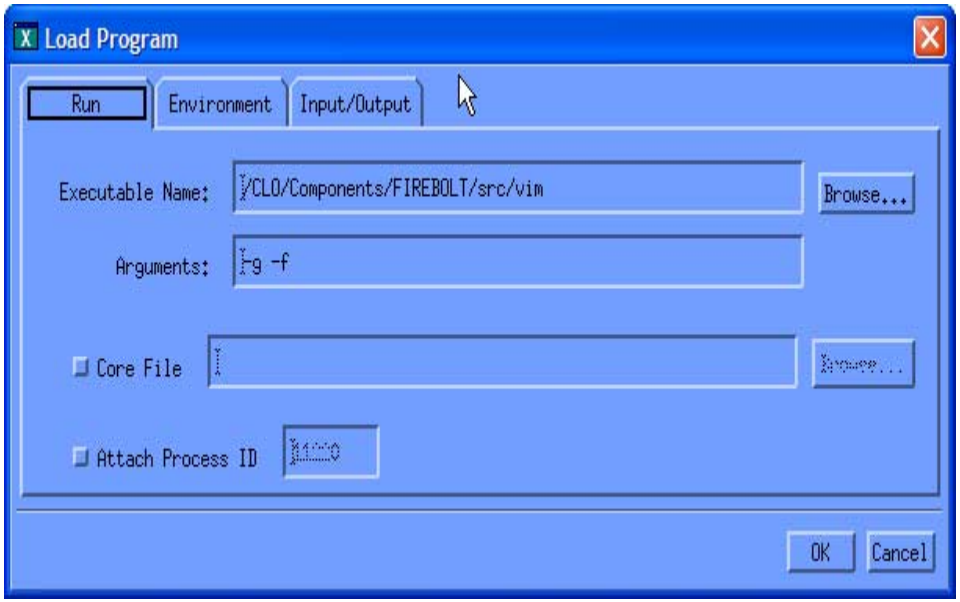
(gdb) core *filename*

Note #7: core files or process id may also be given on command line

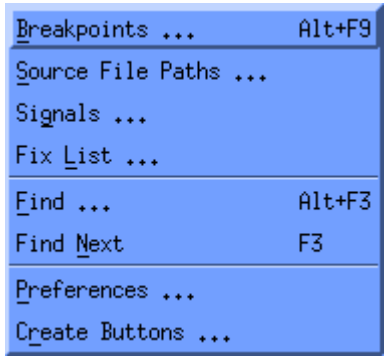
gdb *exec-filename [pid | corefile]*







Breakpoints



(gdb) break *routine*

(gdb) break *file:lineno*

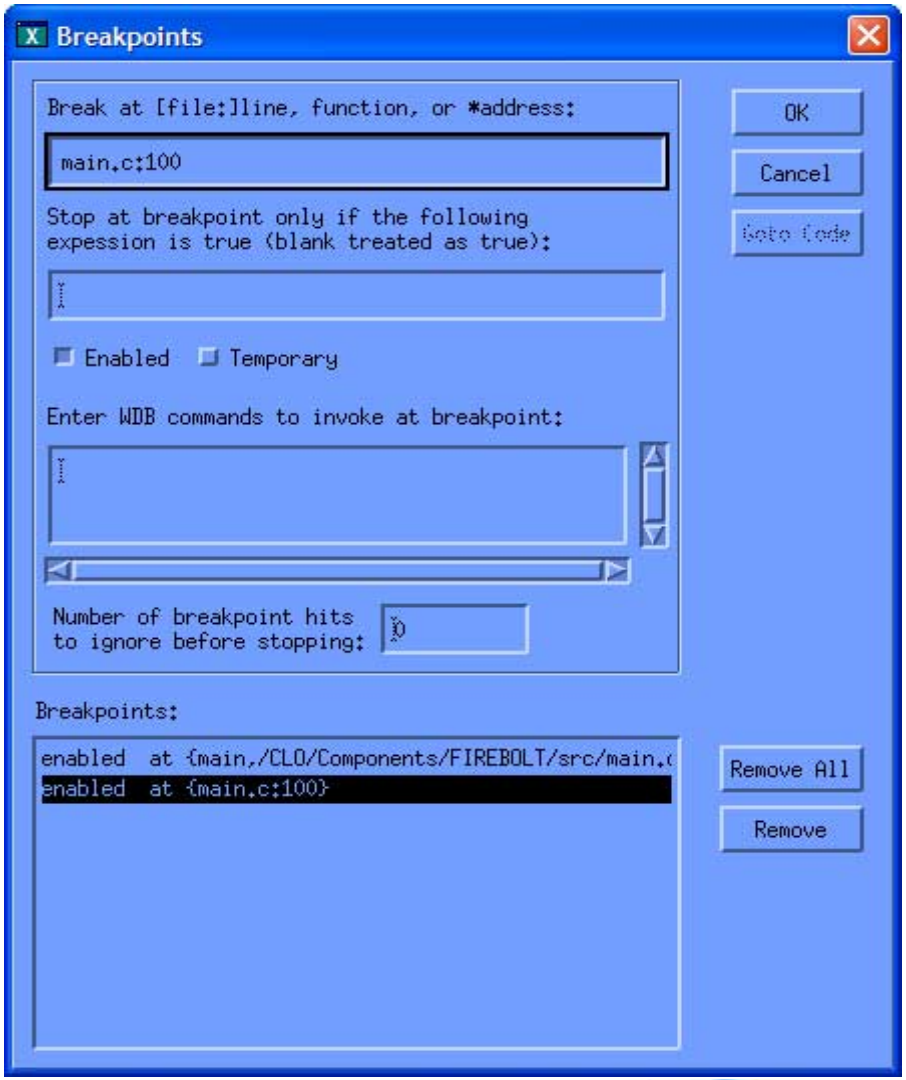
(gdb) break **address*

Note #8: Response tells you what was set :

Breakpoint 1 at 0x31b0: file hello.c, line 4.

Breakpoint 2 (deferred) at “mamba” (“mamba” was not found. Breakpoint deferred until a shared library containing “mamba” is loaded.

Note #9: Deferred breakpoints are set when the debugger can’t find the symbol; useful for shared library debugging.



Breakpoints

Tip #10: Make a breakpoint conditional with the condition command, e.g.

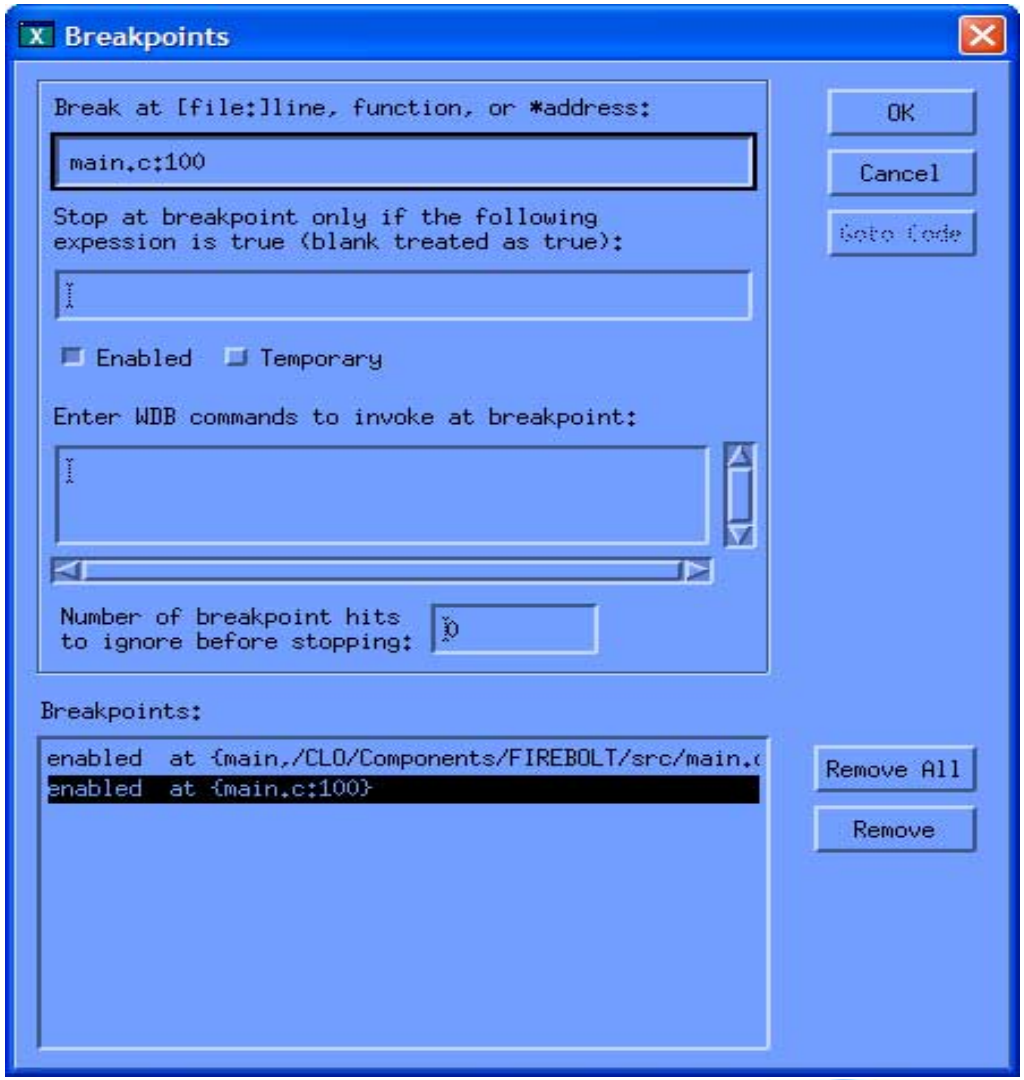
(gdb) condition 1 (x > 5)

Tip #11: Use the commands command to execute commands when a breakpoint is hit, e.g.:

```
(gdb) commands 1
        printf "%d\n",x
        end
```

Tip #12: Use the rbreak command to set a regular expression breakpoint, e.g.:

(gdb) rbreak myfun



Common breakpoints chores

(gdb) tbreak location → set temporary breakpoint at location

(gdb) info breakpoints → view breakpoints list

(gdb) ignore bkpt-num count → ignore count occurrences

(gdb) disable bkpt-num → temporarily suspend breakpoint

(gdb) enable bkpt-num → reactivate a disabled breakpoint.

(gdb) delete bkpt-num → permanently delete a breakpoint.

(gdb) clear → delete breakpoint @ current position

(gdb) clear location-spec → delete breakpoint at the location

(gdb) xbreak function → break at the exit of function

The xbp and xdp commands are useful to set/delete breakpoints at exit of all procedures

Watchpoints (Data Breakpoints)

(gdb) watch *expression*

- Use (gdb) watch *0x*address* to watch an address - otherwise the expression is evaluated repeatedly.
- HP-UX 11.x allows the debugger to implement hardware watchpoints, much much faster
- Itanium supports yet another fast way to watch locations.
- Watchpoints may be modified with the “condition”, “command” and “ignore” commands.
- Recent WDB versions support “deferred” watchpoints, useful to watch as yet unallocated addresses
- Use watchpoints to monitor changes in variable values. Use Display to track variable values.

Deferred Breakpoints

Debugger doesn't know symbols until they are loaded ...

- Solution: deferred breakpoints. This is why wdb reports:
  Breakpoint 1 (deferred) at "routine" ("foo" was not found).
  Breakpoint deferred until a shared library containing "foo" is loaded.
- Deferred breakpoints automatically activated upon library load.

- Note: use "info shared" to see what shared libraries are loaded
- Note: also works with main in a shared library
- Note: supported for "break" or "tbreak" but not for other variations, e.g. xbreak, rbreak. (tbreak sets a temporary breakpoint, xbreak sets breakpoint at exit and rbreak sets a breakpoint on a regular expression).
- Recent versions support deferred watchpoints also

Execution Control

(gdb) cont *[count]*

(gdb) step *[count]*

(gdb) next *[count]*

(gdb) return

(gdb) finish

(gdb) jump *line*

Note: Step steps into a called function, next steps over the call. The optional repeat count tells how many times to do this.

Tip: Many common commands are available by their one letter abbreviations: c, s, n

steplast → useful when call arguments contain other calls : e.g., steplast at foo(goo(),boo()) steps into foo(), not boo()/goo()

Navigating the call stack

- Viewing the current thread's call stack
  - bt, where, info stack
- To switch to a different frame use
  - (gdb) frame <number> → for random access
  - (gdb) up
  - (gdb) down → for sequential navigation
- Viewing local variables of a frame
  - (gdb) info args
  - (gdb) info locals
- To step out of current frame
  - (gdb) finish
- To abruptly return from current frame
  - (gdb) return

Printing values

(gdb) print */format expression*

- The expression can be a command line call e.g.

  (gdb) print function (argument)

- To alter a variable, use a print command, e.g.

  (gdb) print x = 4

(gdb) x */format address*

Formats can include a count, format type and size:

format types include: o - octal; x - hex; d - decimal; u - unsigned decimal; t - binary; f - float; a - address; I - instruction; c - char; s - string.

Format sizes include: b - byte, h - 2bytes, w - 4bytes, g - 8bytes

e.g.

(gdb) print /x variable (gdb) x /20i main

Printing state

(gdb) info

(gdb) show

(gdb) set

Note: “info” tells you about your program;

“show” tells you about the debugger;

“set” changes the things displayed by show.

Things info can display include:

args, breakpoints, files, frame, locals, registers, scope, sharedlib, signals, stack, threads

How to debug a multi-threaded program ?

- Problems troubleshooting multi-threaded and multi-process programs:
  - locking issues: deadlock
  - locking issues: starvation
  - non-deterministic behaviors; non-repeatable
  - overall complexity
- wdb has basic support for user-space and kernel threads, but not explicit support for areas listed above.

WDB support for threaded programs

- Kernel threads, user threads & MxN threads are supported:

(gdb) info threads
  - lists the id numbers of currently known threads
  - Displays both utid and ktid for all user threads
  - Doesn't display kernel threads which are not associated with a user thread (for mxn threads).

(gdb) thread <number>
  - switch to another thread

(gdb) thread apply [number... | all] <command>
  - apply a command to a list of threads

(gdb) break function thread <number>
  - Create a thread specific breakpoint.

(gdb) thread [disable | enable] [number | all]
  - Freeze/thaw threads specifically.

How to debug a multi-process program ?

- wdb has no support (yet) for multi-process programs.
  - attach multiple debuggers, one per process
  - build troubleshooting techniques on top of multiple debuggers
- Following forks:
  - follow-fork-mode decides the identity of the target after fork
  - (gdb) set follow-fork-mode parent | child | ask
  - Default behavior is to stay with the original target.
  - (gdb) catch fork → stop the program on a fork event.

Debugging Shared libraries

(gdb) catch load

- Get control when a shared library is loaded

(gdb) catch unload

- Get control when a shared library is unloaded

(gdb) info shared

- Use this command to list all the shared libraries that are currently loaded.

$ chatr +dbg enable a.out

- Enable shared library debugging during attach
- Loads the shared libraries private

Debugging a running process

- Remember to chatr the executable to debug shared libraries

$ gdb a.out

(gdb) attach 1234

$ gdb a.out 1234

- Stops the process after attach.
- Place required breakpoints and continue

(gdb) detach

Use detach command to detach gdb from the process

Debugging core files

$ gdb *a.out*

(gdb) corefile *core*

$ gdb *a.out core*

- Print local and global variables
- Backtraces
- Thread information
- Examine memory and registers
- Cannot place breakpoints and continue the program

How to ignore or trap signals ?

Signal Disposition

<u>stop</u> - gdb stops program if set

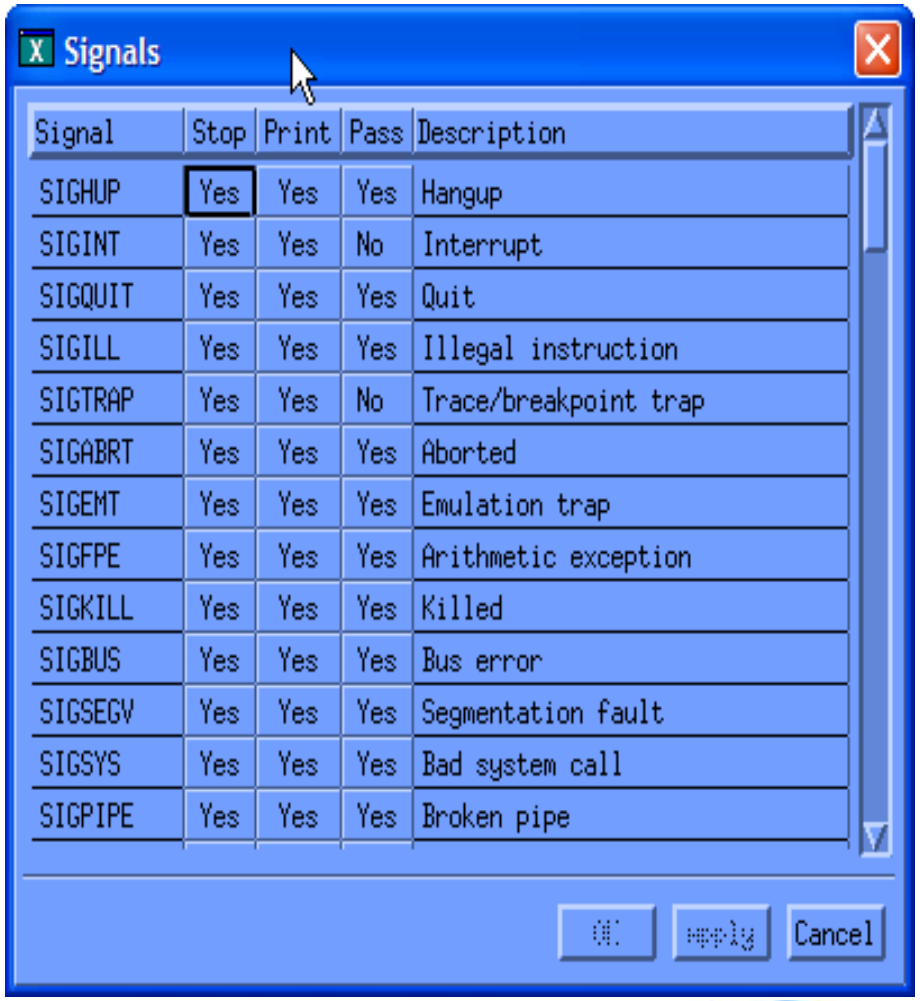
<u>print</u> - gdb prints a message

<u>pass</u> - gdb passes signal to program

(gdb) info signals

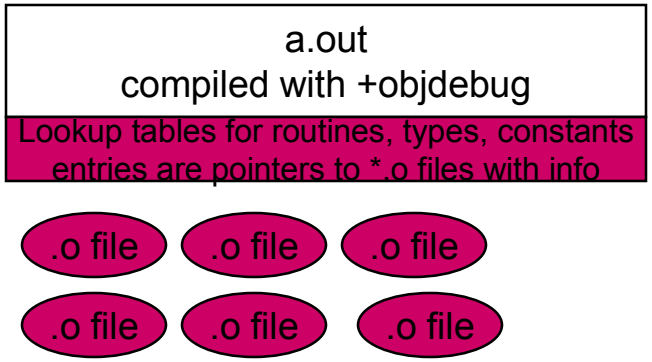
(gdb) handle SIGBUS *[pass | nopass | print | noprint | stop | nostop]*

(gdb) signal SIGBUS



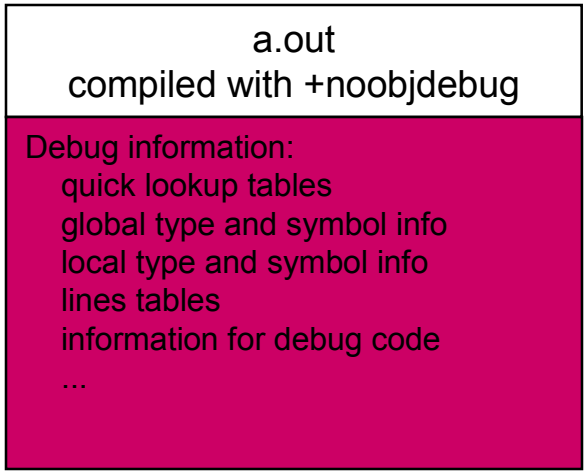
Edit-compile-debug cycle speedup

- Compilation speed for -g links; +objdebug
  - keeps debug information only in object files...



faster link times, no need to run pxdb
smaller executable files, debug info can be huge
different memory profile: smaller to start out

need to load at debug time, often don't care, but can take time.
default behavior on Itanium, new model



no need to find *.o files at debug time
different memory profile: pxdb removes duplicates

default behavior on PA-RISC, old model

How do I find memory leaks?

(gdb) set heap-check [on | off]

(gdb) set heap-check [leaks | bounds | free | scramble] [on | off]

(gdb) set heap-check frame-count *number*

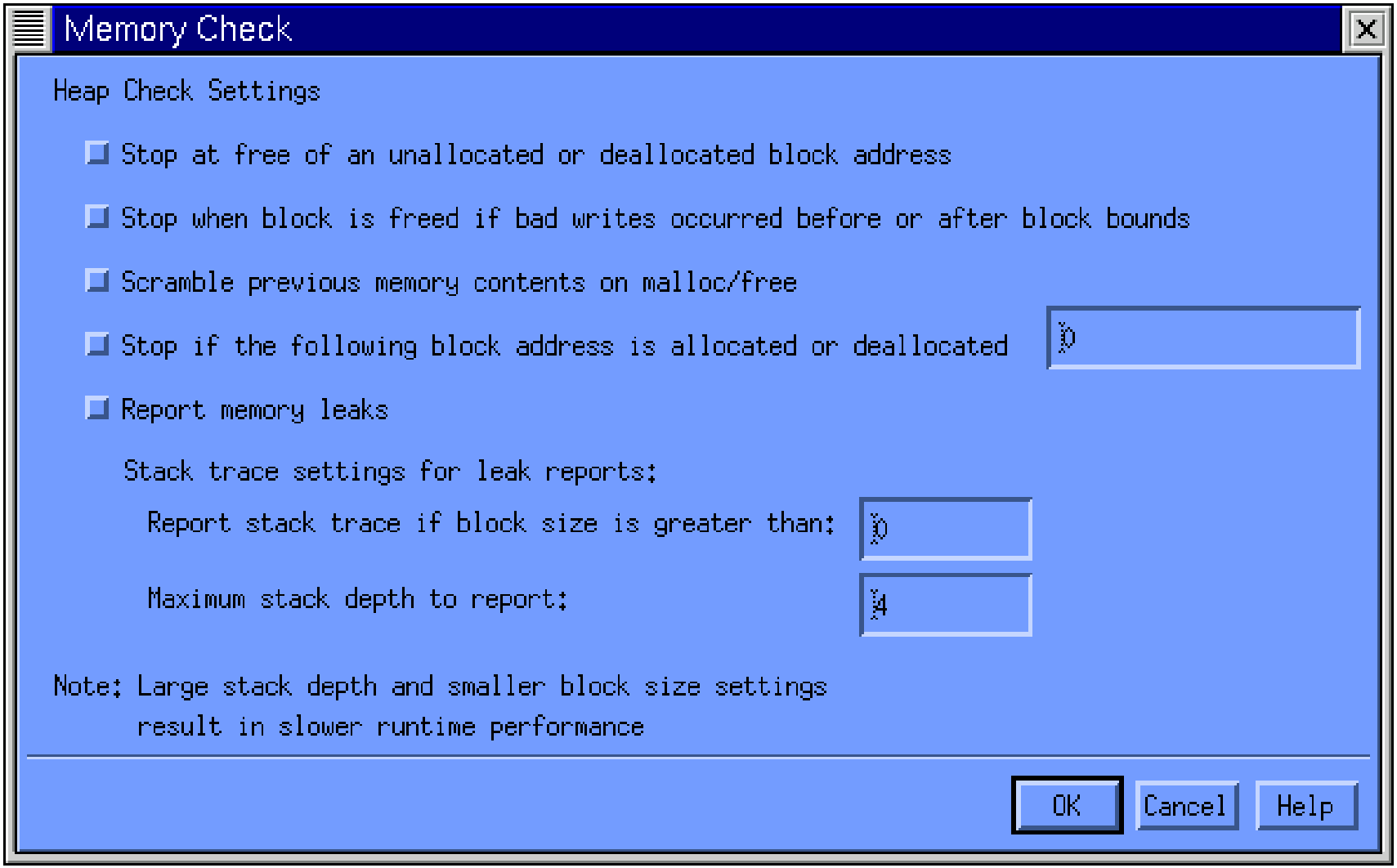
(gdb) set min-leak-size *number*

(gdb) info leaks <file>

(gdb) info heap <file>

Recent versions allow analysis after attach and “Batch mode”

How do I find memory leaks?



Debugging Optimized Code

- Problems with debugging optimized code:
  - register allocation; variables move around and may not even exist
  - source lines combined, eliminated; no longer 1:1 mapping from source code to object code
  - side effects happen in scrambled order; some effects done before others
- Approaches:
  - turn optimization off; either wholesale or with #pragma
  - wdb has some additional basic support:
    - range record information; to keep track of variable locations; tells you truthfully location or if variable is not found
    - better following of scrambled source lines
- Incremental support in future; complex, research problem.

Source level debugging without –g (Itanium)

- Possible on IPF because WDB leverages “minimal line table” added for PBO.
- Useful to debug optimized code, production binaries & dumps from field.
- What works :
  - Breakpoints, step, next, stack traces, disassembly will have source information available
  - Global variables can be printed as usual
- What does not :
  - Type information <u>will not</u> be available
  - Local variables <u>cannot</u> be queried.

Source level debugging without –g: Usage

- Invoking at command line:

  gdb -src_no_g=no_sys_libs <other gdb options>
- Or, start gdb and at the prompt use a set command

  $ gdb <options> ← <u>don't specify file name</u>

  (gdb) set src-no-g no_sys_libs | all | none

  (gdb) file <executable>
- Once src-no-g is enabled, sources are automatically available
- no_sys_libs is the recommended mode.
- Help is available at the following command

  (gdb) help set src-no-g

Support for debugging assembly code

- Facilities for assembly level debugging:
  - wdb GUI, firebolt, ddd have explicit pane/tab
  - If compiled with –g, source and assembly interleaved.
  - info registers to show register state, register window in GUI.
  - si and ni instead of step & next
  - (gdb) disass <function | address> - to disassemble
  - disass works only with statically compiled/linked code addresses.
  - Examine “memory as instruction stream” for dynamic code
    Example : (gdb) x/16i <address>
  - (gdb) b *address → plant a breakpoint at a raw address.

Support for debugging C++ programs

- C++ facilities include
  - Breakpoint menus for overloaded functions
  - use rbreak to set on all members of a class
  - use conditional breakpoints to create instance breakpoints
  - set print object; useful setting to know
  - Exception handling support – catch throw, catch catch

Support for debugging Fortran

- Facilities to debug Fortran include support for
  - Array descriptors
  - Common blocks
  - Case sensitivity
  - Derived types and VMS records
  - Fortran expression types
  - Cray pointers, compiler limitation here

Customizing gdb

- Tip: Create a .gdbinit file for the application; add some of :
  - break fatal
  - dir /path/to/my/sources
  - set args ...
  - set env var value
  - define dump_data
    print data
    end
  - set print object on
- Global preferences in ~/.gdbinit, project specific in ./.gdbinit
- Note: Use “help set” to list the things one might customize
- Note: Use the –nx command line option to ignore .gdbinit
- Note: Use “source <file>” to read in any gdb commands file

Saving and restoring WDB sessions

- Target information
- Breakpoints and watchpoints
- Signal settings
- Source Paths
- Current directory

- Debugger settings
- User-defined buttons
- Positions and sizes of windows
- Command history

Java Unwind Support

- gdb support for Java supplied as a shared library
  - Library supporting Java unwind must be specified
  - Do this by export GDB_JAVA_UNWINDLIB=<path>/libjunwind.sl
  - With latest versions of gdb, you can bypass this step.
- Location of library in the J2SE release (1.3.1+):
  - jre/lib/PA_RISC2.0[W]/libunwind.sl
- No support to debug Java code per se;
- Useful for debugging mixed mode Java/C/C++ code.

Starting Java from gdb

- Built-in mechanism with “java” command:
  - Set DEBUG_PROG in your shell:

    export DEBUG_PROG=/opt/langtools/bin/gdb
    Export GDB_JAVA_UNWINDLIB=<libjunwind.sl>

- Problem using arguments to the java command
  - You need to remove arguments else you get:

    /opt/langtools/bin/gdb: unrecognized option ‘-Xmn500m’
- Once in gdb:
  - Type in all of the arguments:

    (gdb) *r -Xmn500m -Xms1024m -Xmx1024m COM.volano.Mark -port 8000 -count 5000 -rooms 5*
  - Alternatively, use the “set args” feature in your ~/.gdbinit file

Before Java Support

```
#0  0x6ffad8d0 in __ksleep () from /usr/lib/libc.2
#1  0x6fc73298 in _lwp_cond_timedwait () from /usr/lib/libpthread.1
#2  0x6fc72fd4 in pthread_cond_wait () from /usr/lib/libpthread.1
#3  0x6f8de384 in ObjectMonitor::wait ()
   from /opt/java1.3/jre/lib/PA_RISC2.0/server/libjvm.sl
#4  0x6f9182ac in ObjectSynchronizer::wait ()
   from /opt/java1.3/jre/lib/PA_RISC2.0/server/libjvm.sl
#5 ??   → stack trace stops here !!!
```

With Java Support

#0 0x6ffad8d0 in __ksleep () from /usr/lib/libc.2

#1 0x6fc73298 in _lwp_cond_timedwait () from /usr/lib/libpthread.1

#2 0x6fc72fd4 in pthread_cond_wait () from /usr/lib/libpthread.1

#3 0x6f8de384 in ObjectMonitor::wait ()

from /opt/java1.3/jre/lib/PA_RISC2.0/server/libjvm.sl

#4 0x6f9182ac in ObjectSynchronizer::wait ()

from /opt/java1.3/jre/lib/PA_RISC2.0/server/libjvm.sl

#5 0x6f859d30 in JVM_MonitorWait () **– interpreted transition to native**

from /opt/java1.3/jre/lib/PA_RISC2.0/server/libjvm.sl

#6 0xc8b7c in **interpreted frame**: java/lang/Object::wait {(J)V} ()

#7 0x6d41aaf4 in **c2i_adapter frame** () **- compiled to interpreted adapter**

#8 0x6d41a1b8 in **compiled frame**: COM/volano/mcf::x{()[Ljava/lang/Object;} ()

#9 0x6d40517c in **i2c_adapter frame** () **- interpreted to compiled adapter**

#10 0xc4b68 in **interpreted frame**: COM/volano/mca::run {()V} ()

#11 0xc4dcc in **interpreted frame**: java/lang/Thread::run {()V} ()

#12 0x6fe911b8 in Java entry frame ()

Tips & Tricks

- Unable to debug shared libraries upon attach
  - On HP-UX shared libraries truly share a global virtual address
  - No copy on write policy
  - Work around by
    - /opt/langtools/bin/pxdb -s on a.out (or)
    - chatr +dbg on a.out (preferred, works on both PA & IPF)

- Unable to attach to a process
  - Process started over NFS ? Interruptible NFS mount ?
  - mount -o nointr
  - use a local file system
  - Fixed in recent 11.x kernels.

Tips & Tricks (more)

- Debugging C++ inline functions
  - Must use +d switch to aCC
- Debugger as well as debuggee hang
  - Some programs disable all signals when in a critical region
  - Debugger depends on SIGTRAP for breakpoints!!!
  - Best not to muck around with SIGTRAP and SIGINT

Tricks & Tips (more)

- Classes appear partial or empty
  - aCC -g ?
  - Only part of program compiled -g ?
  - Use -g0 for partial debug compiles.
- Long link times with –g
  - Consider +objdebug for compiles
  - Improves link time
  - Higher overhead for gdb bring up.

Tips & Tricks (more)

- Debugger performance
  - If happy, let sleeping dogs lie, go to the next slide.
  - If program composed of shared libraries :
    - set auto-solib-add 1 in .gdbinit file
    - Use share commands in .gdbinit to load relevant libraries
    - Gdb will load libraries as needed in many (not all) situations.
    - Gdb users from other platforms: <u>never</u> set auto-solib-add to 0 on hp !!!
- If watchpoints are very slow
  - 10.20 uses S/W watchpoints :-(

Tips & Tricks (more)

- Unable to view source code
  - Source paths in the image stale ?
  - Use dir command
  - Complied +objdebug?
  - Use objectdir command to let gdb know where the object files are.
  - Use the “pathmap” command in WDB 4.2 and later.

- Backtraces don’t look right on core
  - Analyze core files on the same machine if at all possible.
  - Otherwise environment must be faithfully recreated.
  - ‘packcore’ command available in the WDB 4.5 release.
  - Use GDB_SHLIB_PATH or GDB_SHLIB_ROOT variables.

Tips & Tricks (more)

- Gdb doesn't like the core file
  - "Large" core file ? (>2GB)
  - Recent versions contains support for mega core files.

- Deployment Issues
  - Consider shipping un-stripped
  - Preserve a –O –g version and ship the –O version
  - Install signal handlers and generate stack trace on crash using U_STACK_TRACE() (link with -lcl)

Tips & Tricks (more)

- Command line completion
  - Use <TAB>
- Command line editing and history
  - set editing-mode vi in ~/.inputrc
- Using a different gdb with wdb/firebolt/vdb
  - export GDB_SERVER=<gdb-path>
- Output redirection

  (gdb) ***set redirect-file <filename>***

  (gdb) ***set redirect on|off***

HP WORLD 2004

Solutions and Technology Conference & Expo

Co-produced by:

interex

shared knowledge • shared power

encompass

AN HP USER GROUP

RECOMMENDED TRAINING VENUE FOR THE

HP Certified Professional