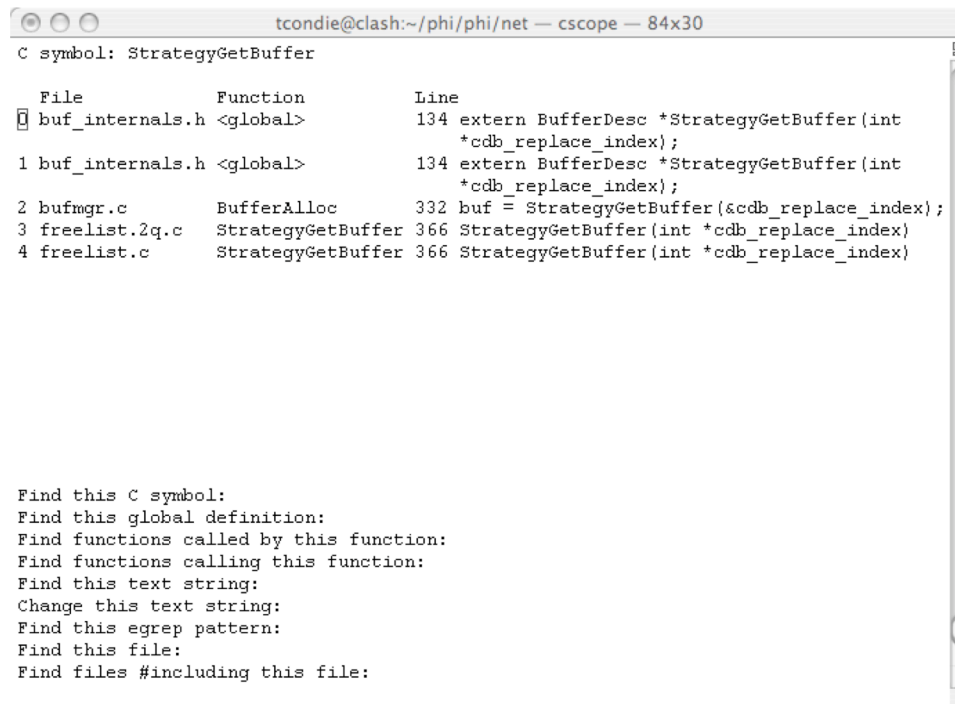


Debugging PostgreSQL

CS186 Staff

Source Browsing with Cscope

- Cscope is a developer's tool for browsing source code
 - <http://cscope.sourceforge.net/> for all the gory details
 - Command line mode
 - In your top level postgres-8.0.3 directory simply run
`cscope -R`
 - This will open a self-explanatory menu of the cscope commands



```
tcondie@clash:~/phi/phi/net -- cscope -- 84x30
C symbol: StrategyGetBuffer

  File          Function          Line
  buf_internals.h <global> 134 extern BufferDesc *StrategyGetBuffer(int
    *cdb_replace_index);
1 buf_internals.h <global> 134 extern BufferDesc *StrategyGetBuffer(int
    *cdb_replace_index);
2 bufmgr.c      BufferAlloc       332 buf = StrategyGetBuffer(&cdb_replace_index);
3 freelist.2q.c StrategyGetBuffer 366 StrategyGetBuffer(int *cdb_replace_index)
4 freelist.c    StrategyGetBuffer 366 StrategyGetBuffer(int *cdb_replace_index)

Find this C symbol:
Find this global definition:
Find functions called by this function:
Find functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
```

Cscope with Emacs

- Building `cscope` index (for use with `vi` or `emacs`):
 - `cscope -bR`
 - Build your `cscope.out` index in the top level directory
 - This generates a `cscope.out` file
 - Rerun when new files/functions are added
- `.emacs` configuration:

```
(setq cscope-do-not-update-database t)
(load-file "<path>/xcscope.el")
(require 'xcscope)
```

`xcscope.el` available at:
www-inst.eecs.berkeley.edu/~cs186/xcscope.el
- Emacs will search for `cscope.out` by looking in the file's directory, then its parent directory, etc...
 - Thus, a single `cscope.out` at top-level source directory suffices

Cscope Keybindings (Emacs)

- All keybindings use the "C-c s" prefix
 - C-c s s Find symbol.
 - C-c s d Find global definition.
 - C-c s g Find global definition (alternate binding).
 - C-c s G Find global definition without prompting.
 - C-c s c Find functions calling a function.
 - C-c s C Find called functions
(list functions called from a function)
 - C-c s t Find text string.
 - C-c s e Find egrep pattern.
 - C-c s f Find a file.
 - C-c s l Find files #including a file.

Setup

- Modify postgres-8.0.3/src/Makefile.global
 - Remove flag '-O2' from CFLAGS variable
- Use postmaster to setup your test database
- Data generation and sample queries can be found in ~cs186/gendata
 - Use .c file to generate massive amounts of rows (stress test your buffer manager)
 - Use postgres backend executable to debug your queries

Setup



Debugging

`gdb`: the Gnu DeBugger

- Debugs C, C++, and Modula-2
- Text and graphical interfaces
- Program crashes
 - Leaves a “core” file
 - Memory dump of the program
 - Run `gdb` on binary and core, explore state of program at crash
- In PostgreSQL you need to specify debug options in the configuration file
 - configure `--enable-debug --enable-cassert`
 - We have done this for you

GDB, the GNU DeBugger

- Text-based, invoked with:

```
gdb [<programfile>] [<corefile>|<pid>]
```

- Argument descriptions:

<i><programfile ></i>	executable program file
<i><corefile></i>	core dump of program
<i><pid></i>	process id of already running program

- Example:

```
gdb ./hello
```

- Compile *<programfile>* with *-g* for debug info

Basic GDB Commands

- **General Commands:**

<i>file</i> [<i><file></i>]	selects <i><file></i> as the program to debug
<i>run</i> [<i><args></i>]	runs selected program with arguments <i><args></i>
<i>attach</i> <i><pid></i>	attach gdb to a running process <i><pid></i>
<i>kill</i>	kills the process being debugged
<i>quit</i>	quits the gdb program
<i>help</i> [<i><topic></i>]	accesses the internal help documentation
<i>shell</i> [<i><stmt></i>]	execute statement in your shell

- **Stepping and Continuing:**

<i>c</i> [<i>ontinue</i>]	continue execution (after a stop)
<i>s</i> [<i>tep</i>]	step one line, entering called functions
<i>n</i> [<i>ext</i>]	step one line, without entering functions
<i>finish</i>	finish the function and print the return value

GDB PostgreSQL

```
emacs@clash-pb.gateway.2wire.net
File Edit Options Buffers Tools Gud Complete In/Out Signals Help

Current directory is /Users/tcondie/Workspace/CS186/pgsql/bin/
GNU gdb 6.1-20040303 (Apple version gdb-384) (Mon Mar 21 00:05:26 GMT 2005)
Copyright 2004 Free Software Foundation, Inc.
GDB 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 GDB.  Type "show warranty" for details.
This GDB was configured as "powerpc-apple-darwin"...Reading symbols for shared library
ies ..... done

(gdb) b StrategyBufferLookup
Breakpoint 1 at 0x1faed4: file freelist.c, line 254.
(gdb) break StrategyGetBuffer
Breakpoint 2 at 0x1fbad0: file freelist.c, line 367.
(gdb) b StrategyReplaceBuffer
Breakpoint 3 at 0x1fc100: file freelist.c, line 501.
(gdb) shell ps
  PID  TT  STAT      TIME COMMAND
12097  p1  S        0:00.10 -bash
12132  p1  R        0:03.81 emacs
12151  p1  S+       0:00.25 postgres test
12144  p2  Ss+     0:00.38 /usr/libexec/gdb/gdb-powerpc-apple-darwin --annotate=
(gdb) attach 12151

--:** *gud-postgres* All L22 (Debugger:run)-----
C-x 1
```

GDB Breakpoints

- Useful breakpoint commands:

`b[reak] [<where>]`

sets breakpoints. `<where>` can be a number of things, including a hex address, a function name, a line number, or a relative line offset

`[r]watch <expr>`

sets a watchpoint, which will break when `<expr>` is written to [or read]

`catch <event>`

breaks on `<event>`, which can be used to catch many events, including the throwing and catching of C++ exceptions

`info break[points]`

prints out a listing of all breakpoints

`clear [<where>]`

clears a breakpoint at `<where>`

`d[etele] [<nums>]`

deletes breakpoints by number

`cond[ition] <bp>`

remove a condition to a given breakpoint

`cond[ition] <bp> <expr>`

Add a condition to a given breakpoint

Playing with Data in GDB

- Commands for looking around:

<i>list</i> [<i><where></i>]	prints out source code at <i><where></i>
<i>search</i> <i><regexp></i>	searches source code for <i><regexp></i>
<i>backtrace</i> [<i><n></i>]	prints a backtrace <i><n></i> levels deep
<i>info</i> [<i><what></i>]	prints out info on <i><what></i> (like local variables or function args)
<i>p[rint]</i> [<i><expr></i>]	prints out the evaluation of <i><expr></i>

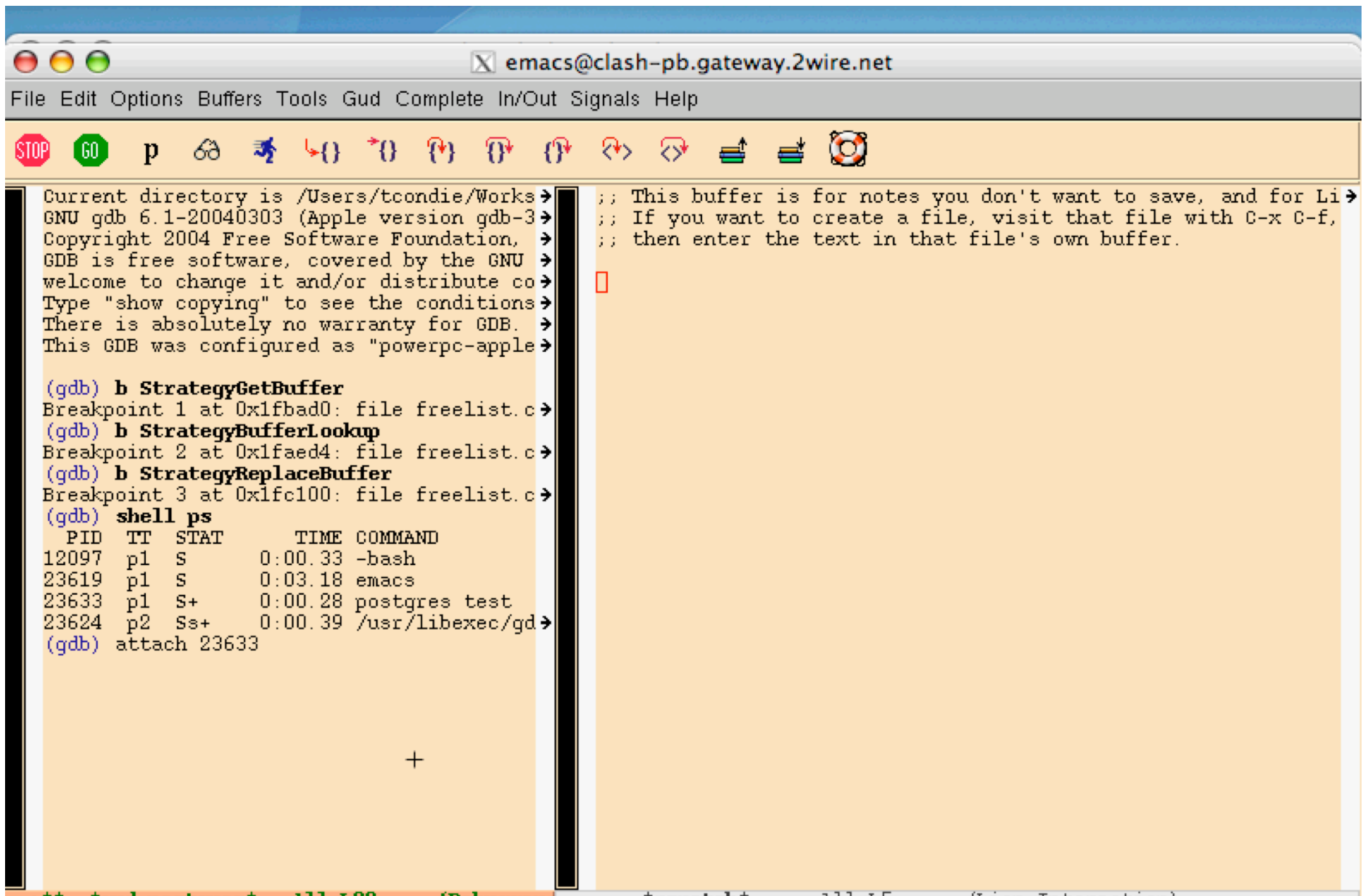
- Commands for altering data and control path:

<i>set</i> <i><name></i> <i><expr></i>	sets variables or arguments
<i>return</i> [<i><expr></i>]	returns <i><expr></i> from current function
<i>jump</i> <i><where></i>	jumps execution to <i><where></i>
<i>call</i> <i><expr></i>	Call a function within debugger

The Basics

- Breakpoint and basic navigation commands will cover most needs
 - A breakpoint halts the execution at the specified routing or file location
 - The primary navigation commands are *next* and *step*
 - The finish and continue commands are also very useful
- The backtrace will print out your call stack

The Basics



```
Current directory is /Users/tcondie/Works>
GNU gdb 6.1-20040303 (Apple version gdb-3>
Copyright 2004 Free Software Foundation, >
GDB is free software, covered by the GNU >
welcome to change it and/or distribute co>
Type "show copying" to see the conditions>
There is absolutely no warranty for GDB. >
This GDB was configured as "powerpc-apple>

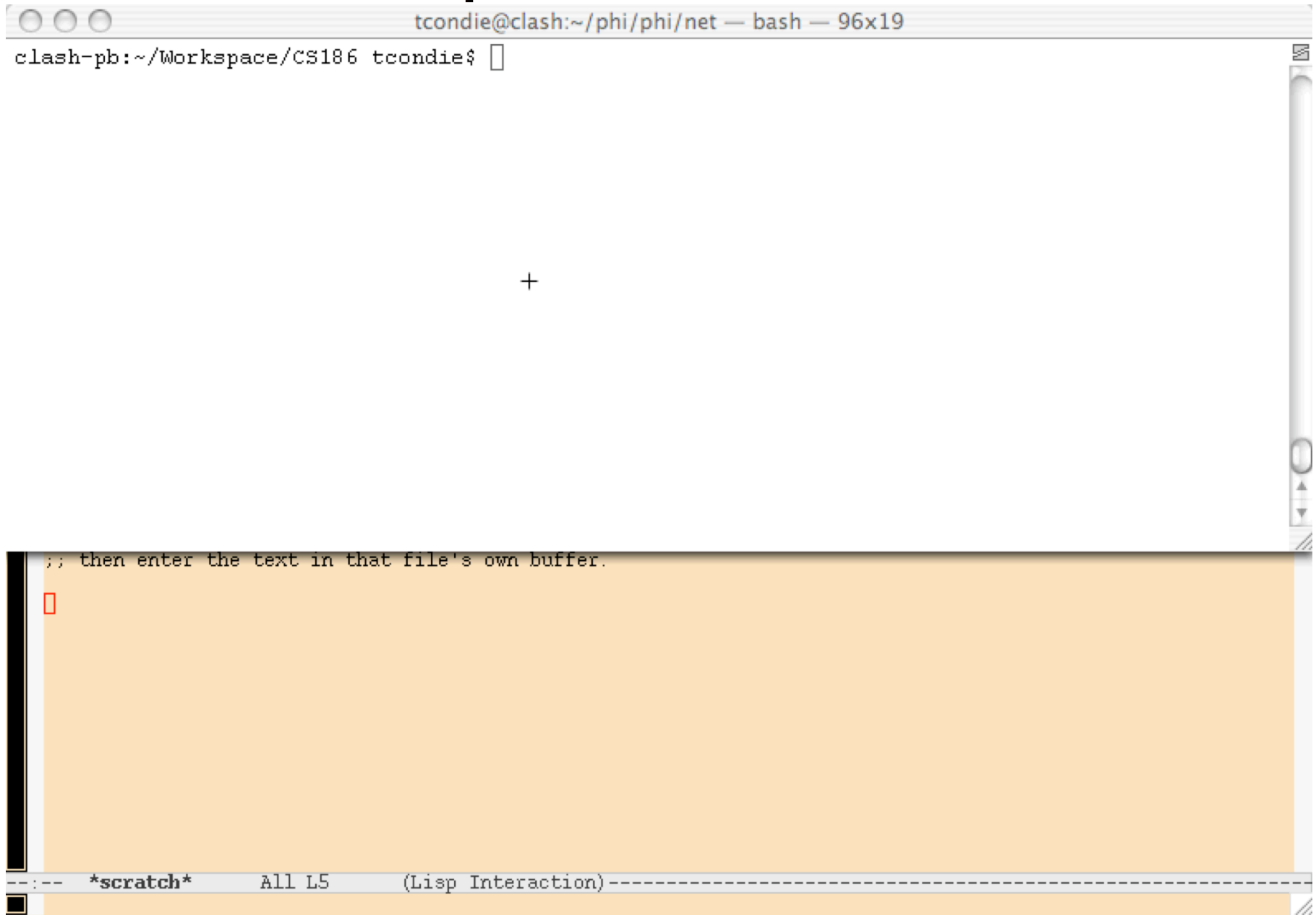
(gdb) b StrategyGetBuffer
Breakpoint 1 at 0x1fbad0: file freelist.c>
(gdb) b StrategyBufferLookup
Breakpoint 2 at 0x1faed4: file freelist.c>
(gdb) b StrategyReplaceBuffer
Breakpoint 3 at 0x1fc100: file freelist.c>
(gdb) shell ps
  PID  TT  STAT      TIME COMMAND
12097  p1  S        0:00.33 -bash
23619  p1  S        0:03.18 emacs
23633  p1  S+       0:00.28 postgres test
23624  p2  Ss+     0:00.39 /usr/libexec/gd>
(gdb) attach 23633

;; This buffer is for notes you don't want to save, and for Li>
;; If you want to create a file, visit that file with C-x C-f,
;; then enter the text in that file's own buffer.
```

Breakpoint Conditions

- You can attach conditions (or predicates) to breakpoints
 - The execution will halt when the condition becomes true
 - Be careful when placing conditions on pointers!
 - You may accidentally dereference a NULL pointer

Breakpoint Conditions



Watchpoints

- A watchpoint is like an unbounded condition on some variable
 - The execution halts when the condition on the watch variable becomes true
 - If the variable goes out of scope then the watch point is automatically deleted!
 - E.g., watch on a variable local to a function is deleted when the function returns
 - As with breakpoints, take care when ‘watching’ pointer variables

Watchpoints

```
emacs@clash-pb.gateway.2wire.net
File Edit Options Buffers Tools Gud Complete In/Out Signals Help

STOP GO p [other icons]

PID  TT  STAT      TIME COMMAND
12097 p1  S         0:00.38 -bash
23659 p1  S         0:06.33 emacs
23681 p1  S+        0:00.25 postgres -B 16 test
23661 p2  Ss+       0:00.91 /usr/libexec/gdb/gdb-powerpc-apple-darwin --annotate=
(gdb) attach 23681
Attaching to program: `/Users/tcondie/Workspace/CS186/pgsql/bin/postgres', process 23681.
0x900137a4 in read ()
(gdb) info b
Num Type           Disp Enb Address      What
1  breakpoint      keep n   0x001faed4 in StrategyBufferLookup at freelist.c:254
   breakpoint already hit 1 time +
2  breakpoint      keep y   0x001fc100 in StrategyReplaceBuffer at freelist.c:501
   stop only if cdb_found_index >= 0
   breakpoint already hit 2 times
(gdb)
--: ** *gud-postgres* Bot L73 (Debugger:run)
Dump of assembler code for function read:
0x900137a0 <read+0>:  li    r0, 3
▶ 0x900137a4 <read+4>:  sc
0x900137a8 <read+8>:  b     0x900137b0 <read+16>
0x900137ac <read+12>:  blr
0x900137b0 <read+16>:  mflr r0
0x900137b4 <read+20>:  bcl- 20, 4*cr7+so, 0x900137b8 <read+24>
0x900137b8 <read+24>:  mflr r12
0x900137bc <read+28>:  mtlr r0
0x900137c0 <read+32>:  addis r12, r12, 4095
0x900137c4 <read+36>:  lwz   r12, 10448(r12)
0x900137c8 <read+40>:  mtctr r12
0x900137cc <read+44>:  bctr
0x900137d0 <read+48>:  .long 0x0
0x900137d4 <read+52>:  .long 0x0
0x900137d8 <read+56>:  .long 0x0
0x900137dc <read+60>:  .long 0x0
```

More and More GDB

- For the most complete and up-to-date documentation on GDB, use the GDB *help* subsystem from within *gdb*. A close second is the GDB info node (*info gdb*).
- There is also an *emacs* interface to GDB, which allows you to keep the source code open in a separate frame while debugging (documentation for how to use this is available in the GDB info node)

Profiling

- “90% exec time spent in 10% of the code”
- Popular tools for profiling:
 - gprof
 - Flat profile: time spent in functions and how often called
 - Call graph: Which functions called which and how often
 - Annotated source: source w/ #times each line executed
 - gcov
 - Test coverage
 - SimpleScalar
 - Program performance analysis
 - Micro-arch modeling
 - Hardware-software co-verification

Profiling: gprof

- We recommend using gprof
 - Others not really important for your purposes
- Use gcc/g++ options: -pg
 - Add to configure file with the --enable-debug flag
- Run the binary (e.g., postgres) as normal, outputs gmon.out
- Run gprof to analyze
 - gprof <binary> <path>/gmon.out
 - binary The binary file (e.g., postgres)
 - gmon.out dynamic call graph and profile (produced by execution).
 - See man page for further details

```
tcondie@clash:~/phi/phi/net -- vim -- 79x20
index  %time  self  descendents  called/total  parents  index
called+self  children
-----
[68]   0.0   0.00   0.00   5429/5429    _ReadBufferInternal [65]
          0.00   0.00   5429        _StrategyBufferLookup [68]
          0.00   0.00   5429/5429  _BufTableLookup [64]
-----
[201]  0.0   0.00   0.00   126/126     _ReadBufferInternal [65]
          0.00   0.00   126        _StrategyGetBuffer [201]
-----
[202]  0.0   0.00   0.00   126/126     _ReadBufferInternal [65]
          0.00   0.00   126        _StrategyReplaceBuffer [202]
          0.00   0.00   126/126    _BufTableInsert [198]
```