1 Debugger Overview

The iWarp debugger, `iwdb`, is a symbolic source-code debugger for debugging cell programs running on iWarp arrays. The iWarp debugger is based on the public domain GNU debugger, `gdb`, with extensions to handle the additional functionality of the iWarp array.

The iWarp debugger provides a line-oriented, interactive tty interface that allows debugging on any iWarp array connected to any server in the network. The iWarp debugger supports the following operations:

- start a program
- make the program stop on specified conditions
- examine program results to identify bugs
- change the program to correct the effects of one bug and proceed to another without having to recompile

The iWarp debugger also supports a window-based user interface, `iwdb_tool`, which supports the full set of features of `iwdb`. You can invoke `iwdb` or `iwdb_tool` on any node in the network that has RPC access to an iWarp array server node. In addition, `iwdb_tool` requires a graphic display running X11, release 5 or later. Figure 1-1 illustrates the window-based debugger.

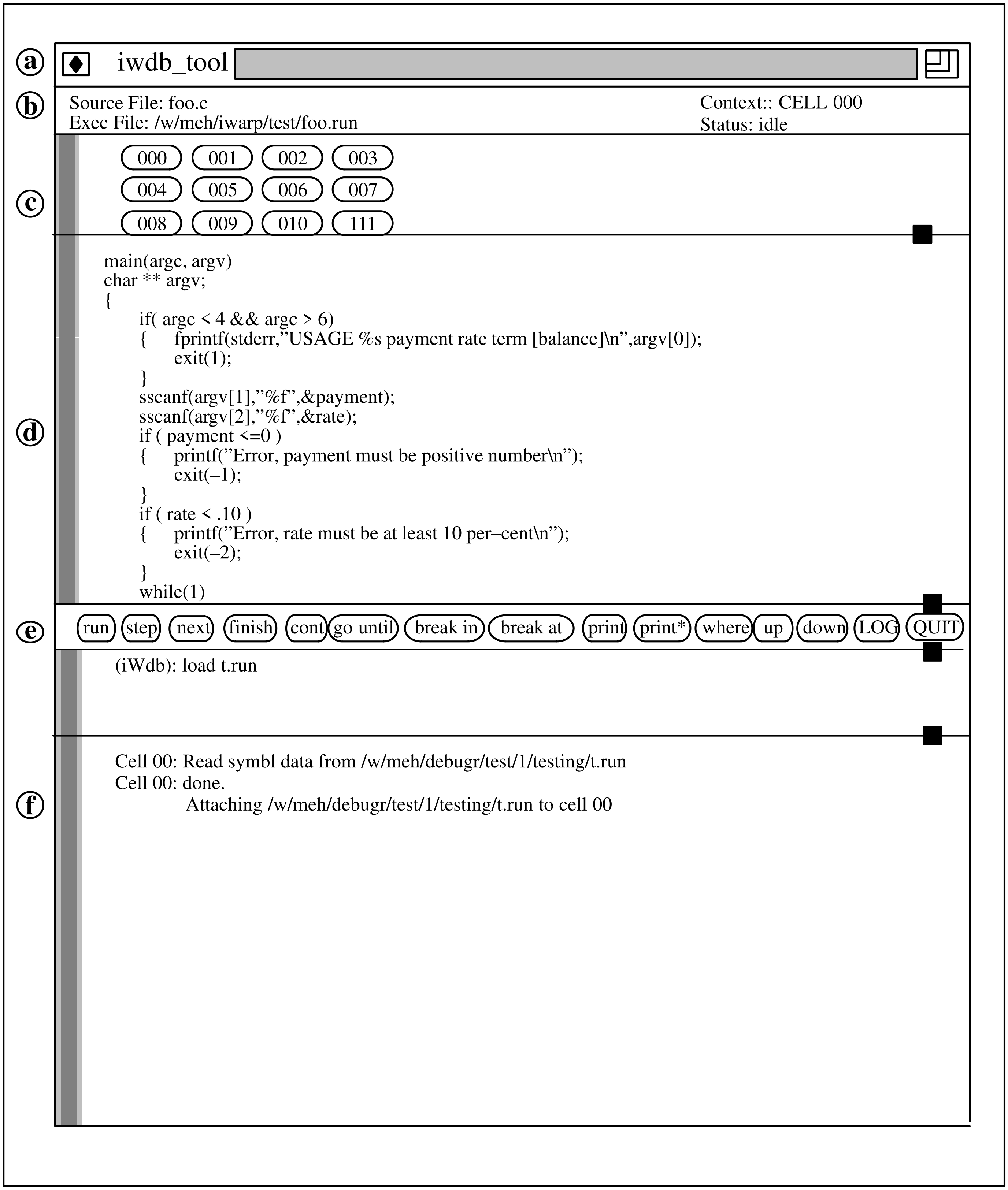

Figure 1-1: Window-Based Debugger

The following describes the frames in the debugger window. The identifying letters correspond to the circled letters shown to the left of each frame of the debugger window in Figure 1-1.

a Title frame: Gives the tool name and version number.

b Context frame: Names the context (either group name or cell_id list), the program currently associated with that context, the line numbers displayed from the current source file, the source file name, and the context status.

c Cell icon frame: Contains one icon for each cell in the array. The frame can be scrolled up, down, left, and right to accommodate very large arrays. The state of each cell is indicated by its shading. A white cell indicates that the cell is stopped. A shaded cell indicates the cell is the current selected context, but not running. A black cell indicates that the cell program is running. You can select a cell to be the current context by clicking on the corresponding icon in this window. If the icon is a member of a group, the group becomes the context.

d Source frame: Displays the source code for the current context. You can move the cursor to select lines, which then become the basis for commands. The source code displayed changes automatically. The cursor indicates the line of the current program counter, whenever the program counter changes due to an `up` or `down` command, or a breakpoint is hit. You can change the source code displayed by using the `list` command.

e Command icon frame: Represents selected commands with icons. By placing the cursor on one of the icons and clicking the LEFT mouse button, that command will be performed in the current array context and source line number indicated by the cursor. Command completion using the keyboard is also allowed.

f Input/Output frame: You can type command input into this frame. This is also where output from the debugger appears. There is a scroll bar on the right-hand side which allows paging back for up to 200 lines.

Getting Help

The iWarp debugger provides the following two commands to get help and additional information:

- `help`
- `info`

The `help` command provides on-line documentation about any debugger command. If you type `help`, the debugger will list the classes of debugger commands. The classes are:

- running running the program
- stack examining the stack
- data examining data
- breakpoints making program stop at certain points
- files specifying and examining files
- status status inquiries
- support support facilities
- user user-defined commands
- alias aliases of other commands

- obscure obscure features

If you type `help`, followed by a class name, the debugger will list all commands in that class. If you type `help`, followed by a command name, the debugger will list complete on-line documentation for that command.

The `info` command provides status information about the current state of the argument given. For example, `info group` causes the debugger to list the current group names followed by their associated cell ids. Uses of the `info` command are included in sections throughout this manual.

Debugger Input and Output Conventions

You invoke the iWarp debugger with the shell command `iwdb`. Once started, the debugger reads commands from the terminal until you tell it to quit.

The debugger input has the following properties:

- A debugger command is a single line of input with no limit on line length.
- A blank line as input to the debugger repeats the previous command.
- A line of input starting with # is a comment. This is useful in command files.
- The `quit` command exits the debugger.
- `Ctrl-c` terminates the cell program under the current context and returns to debugger command level.

You can also control elements of the display using the `set` command. This includes:

- changing the debugger prompt
- setting the screen size
- suppressing information messages

To change the debugger prompt, use the `set prompt` command as follows:

`set prompt` *`newprompt`*

This command replaces the default debugger prompt with the string specified by *newprompt*.

Certain debugger commands output large amounts of information to the screen. To help you read all of it, the debugger pauses at the end of each page of output until you press the return key. The debugger determines the size of the screen from the termcap data base and the value of the TERM environment variable. If the screen size is not correct, you can override it with the `set screensize` command as follows:

`set screensize` *`height`* `[`*`width`*`]`

This command specifies a screen height of *height* lines and a width of *width* characters. If you specify a height of zero lines, the debugger will not pause regardless of the output length. This is useful if output is sent to a file or to an editor buffer.

To suppress certain informational messages, use the `set verbose` command as follows:

```
set verbose off
```

The messages controlled by `set verbose` are those that announce that the symbol table for a source file is being read. To re-enable these messages, use the `set verbose` command as follows:

```
set verbose on
```

Compiling Your Program for Debugging

In order to debug a program, you must ask for debugging information when you compile the program. To generate debugging information, use the -g option to the `iwcc` command.

By using this option, you generate an object file that describes the data type of each variable or function and the correspondence between source line numbers and addresses in the executable code.

The iWarp C compiler is unable to handle the -g and -O options together. This means that you cannot ask for optimization if you ask for debugging information. It is generally more effective to compile using the -g option until all debugging is complete, and then compile the debugged program using the -O option.

Example Debugger Session

This section provides an example iWarp debugger session. The commands and techniques used in the example session are described in detail in the chapters that follow. Refer back to the example as you go through the chapters to see how the commands are used in a debugger session. In the example, (iwdb) is the debugger prompt. All command lines are shown in boldface type for readability.

```
% iwdb

GDB 3.4, Copyright (C) 1989 Free Software Foundation, Inc.
There is ABSOLUTELY NO WARRANTY for GDB; type "info warranty" for
details. GDB is free software and you are welcome to distribute
copies of it under certain conditions; type "info copying" to see
the conditions.
Type "help" for a list of commands.

Connecting to RPC server on iwarpu ..[1] 5815
.Cell 00: Reading symbol data from /iw/debugr/tests/data_types/global...
Cell 00: done.
   Attaching /iw/debugr/tests/data_types/global to cell 00
Cell 00: 9    unsigned char op;     /* the opcode, from 0x00 to 0xff*/
Cell 00: 10   unsigned char signum;  /* signal number, 0 - 256*/
Cell 00: 11   unsigned char len;     /* length , for short DM */
Cell 00: 12   char data[17];
Cell 00: 13   };
Cell 00: 14 struct smsg s1;
Cell 00: 15  main()
Cell 00: 16  {
Cell 00: 17
Cell 00: 18            printf("starting");

(iWdb) info context
The current context is cell: 00
      assigned cell program: /iw/debugr/tests/data_types/global

(iWdb) cell-numbering cellid.2

(iWdb) info context
The current context is cell: 0X0
      assigned cell program: /iw/debugr/tests/data_types/global
```

```
(iWdb) context 0X1

(iWdb) load local
Cell 0X1: Reading symbol data from /iw/debugr/tests/data_types/local...
Cell 0X1: done.
      Attaching /iw/debugr/tests/data_types/local to cell 0X1

(iWdb) context df
iWarp cell id or name of a iWarp cell group

(iWdb) info context
The current context is cell: 0X1
      assigned cell program: /iw/debugr/tests/data_types/local

(iWdb) info group
The currently defined groups are:

(iWdb) help group
Specify a name for a group of iWarp cells. This string can
be used instead of a single cell id. Then all commands apply to the
set of cells. This is only possible if all cells are running the same
program and are in the same state.

(iWdb) group 0X2 0X3 0X4
The first letter of the name cannot be a digit!

(iWdb) group foo 0X2 0X3 0X4

(iWdb) info group
         foo
                with cells:
                   0X4 0X3 0X2

(iWdb) group foo
group name does already exist! Do you want to delete it? (y or n) n
command ignored

(iWdb) context foo

(iWdb) load global
Cell 0X4: Reading symbol data from /iw/debugr/tests/data_types/global...
Cell 0X4: done.
Cell 0X3: Reading symbol data from /iw/debugr/tests/data_types/global...
Cell 0X3: done.
Cell 0X2: Reading symbol data from /iw/debugr/tests/data_types/global...
Cell 0X2: done.
      Attaching /iw/debugr/tests/data_types/global to cell 0X4
      Attaching /iw/debugr/tests/data_types/global to cell 0X3
      Attaching /iw/debugr/tests/data_types/global to cell 0X2

(iWdb) list
Cell foo: 9      unsigned char op;      /* the opcode, from 0x00 to 0xff*/
Cell foo: 10     unsigned char signum;   /* signal number, 0 - 256*/
Cell foo: 11     unsigned char len;  /* length , for short DM */
Cell foo: 12     char data[17];
Cell foo: 13     };
Cell foo: 14  struct smsg s1;
Cell foo: 15  main()
```

```
Cell foo: 16  {
Cell foo: 17
Cell foo: 18       printf("starting");
```

(iWdb)

```
Cell foo: 19       s1.addr = (char*) 0x1234;
Cell foo: 20       s1.op = 'a';
Cell foo: 21       s1.signum = 'b';
Cell foo: 22       s1.len = 16;
Cell foo: 23       strcpy(s1.data , "iw was here");
Cell foo: 24       sub(&s1);
Cell foo: 25       printf("back from sub() \n");
Cell foo: 26  }
Cell foo: 27  sub(s)
Cell foo: 28  struct smsg *s;
```

(iWdb) break 24

```
Cell 0X4:  Breakpoint 1 at 0x1e8a8: file global.c, line 24.
Cell 0X3:  Breakpoint 1 at 0x1e8a8: file global.c, line 24.
Cell 0X2:  Breakpoint 1 at 0x1e8a8: file global.c, line 24.
```

(iWdb) run

```
Cell 0X4:  Starting program: /iw/debugr/tests/data_types/global
Cell 0X3:  Starting program: /iw/debugr/tests/data_types/global
Cell 0X2:  Starting program: /iw/debugr/tests/data_types/global
```

(iWdb)

```
Cell 0X4:  Bpt 1, main () (global.c line 24)
Cell 0X4:  24         sub(&s1);
```

(iWdb)

```
Cell 0X3:  Bpt 1, main () (global.c line 24)
Cell 0X3: 24          sub(&s1);
```

(iWdb)

```
Cell 0X2:  Bpt 1, main () (global.c line 24)
Cell 0X2: 24          sub(&s1);
```

(iWdb) where

```
Cell 0X4:  #0  main () (global.c line 24)
Cell 0X3:  #0  main () (global.c line 24)
Cell 0X2:  #0  main () (global.c line 24)
```

(iWdb) context 0X0

(iWdb) where

(iWdb) list

```
Cell 0X0: 19       s1.addr = (char*) 0x1234;
Cell 0X0: 20       s1.op = 'a';
Cell 0X0: 21       s1.signum = 'b';
Cell 0X0: 22       s1.len = 16;
Cell 0X0: 23       strcpy(s1.data , "iw was here");
Cell 0X0: 24       sub(&s1);
Cell 0X0: 25       printf("back from sub() \n");
Cell 0X0: 26  }
Cell 0X0: 27  sub(s)
Cell 0X0: 28  struct smsg *s;
```

```
(iWdb) run
Cell 0X0: Starting program: /iw/debugr/tests/data_types/global

(iWdb) context 0X1
Cell 0X0: Program exited normally.

(iWdb) run
Cell 0X1: Starting program: /iw/debugr/tests/data_types/local

(iWdb) context foo
Cell 0X1: Program exited normally.

(iWdb) info break
Cell 0X4: Breakpoints:
Cell 0X4: Num Enb   Address    Where
Cell 0X4: #1   y  0x0001e8a8  in main (global.c line 24)
Cell 0X3: Breakpoints:
Cell 0X3: Num Enb   Address    Where
Cell 0X3: #1   y  0x0001e8a8  in main (global.c line 24)
Cell 0X2: Breakpoints:
Cell 0X2: Num Enb   Address    Where
Cell 0X2: #1   y  0x0001e8a8  in main (global.c line 24)

(iWdb) context 0X0

(iWdb) info break
Cell 0X0: No breakpoints.

(iWdb) context foo

(iWdb) delete
Delete all breakpoints? (y or n) y
Delete all breakpoints? (y or n) y
Delete all breakpoints? (y or n) y

(iWdb) cont
Cell 0X4: Continuing.
Cell 0X3: Continuing.
Cell 0X2: Continuing.

(iWdb)
Cell 0X2: Program exited normally.

(iWdb)
Cell 0X3: Program exited normally.

(iWdb)
Cell 0X4: Program exited normally.
(iWdb) quit
The program is running.  Quit anyway? (y or n) y

%
```