Debugging under TOS

Objectives

- Explain the TOS testing system
- Explain some debugging techniques when a program error typically crashes the whole system
- Explain symbolic debugging of TOS

Test Cases

- TOS comes with many test cases that test the behavior of your implementation
- All these tests are located in `~/tos/test`
- Each test case has a name and it tests one particular feature of your implementation, e.g., `test_mem_1` tests the peek and poke functions
- Each test case is stored in a separate file, e.g., `~/tos/test/test_mem_1.c`
- If a test fails, the system will print an error code

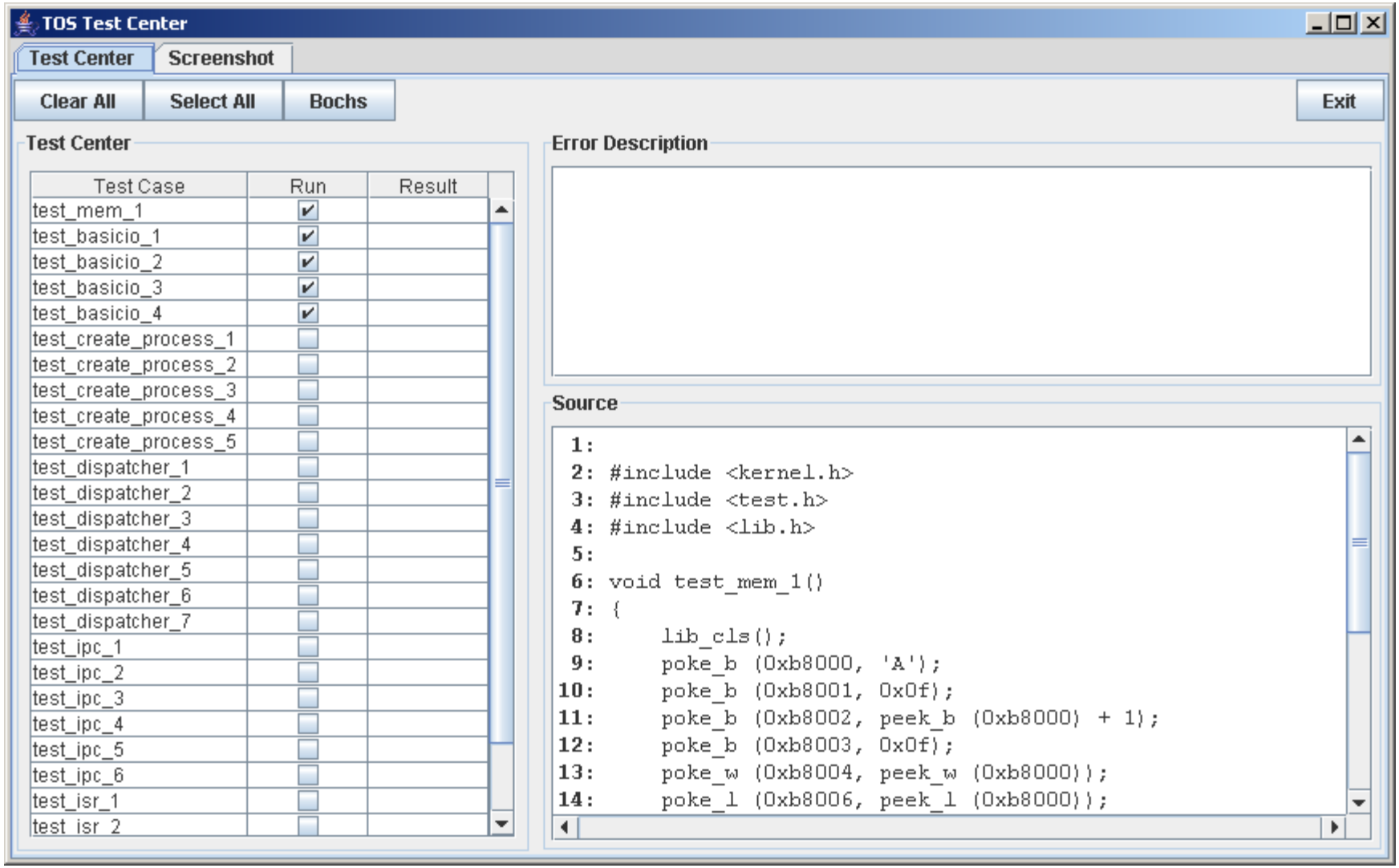
TOS Test Center (TTC)

- The TTC is a Java application that simplifies the execution of test cases.
- TTC allows to select which test cases to run.
- TTC can launch Bochs to execute the test cases within the emulation.
- Shows which test cases succeeded and which failed.
- Provide some hints about common mistakes.
- Screenshots of successful executions for each test case.

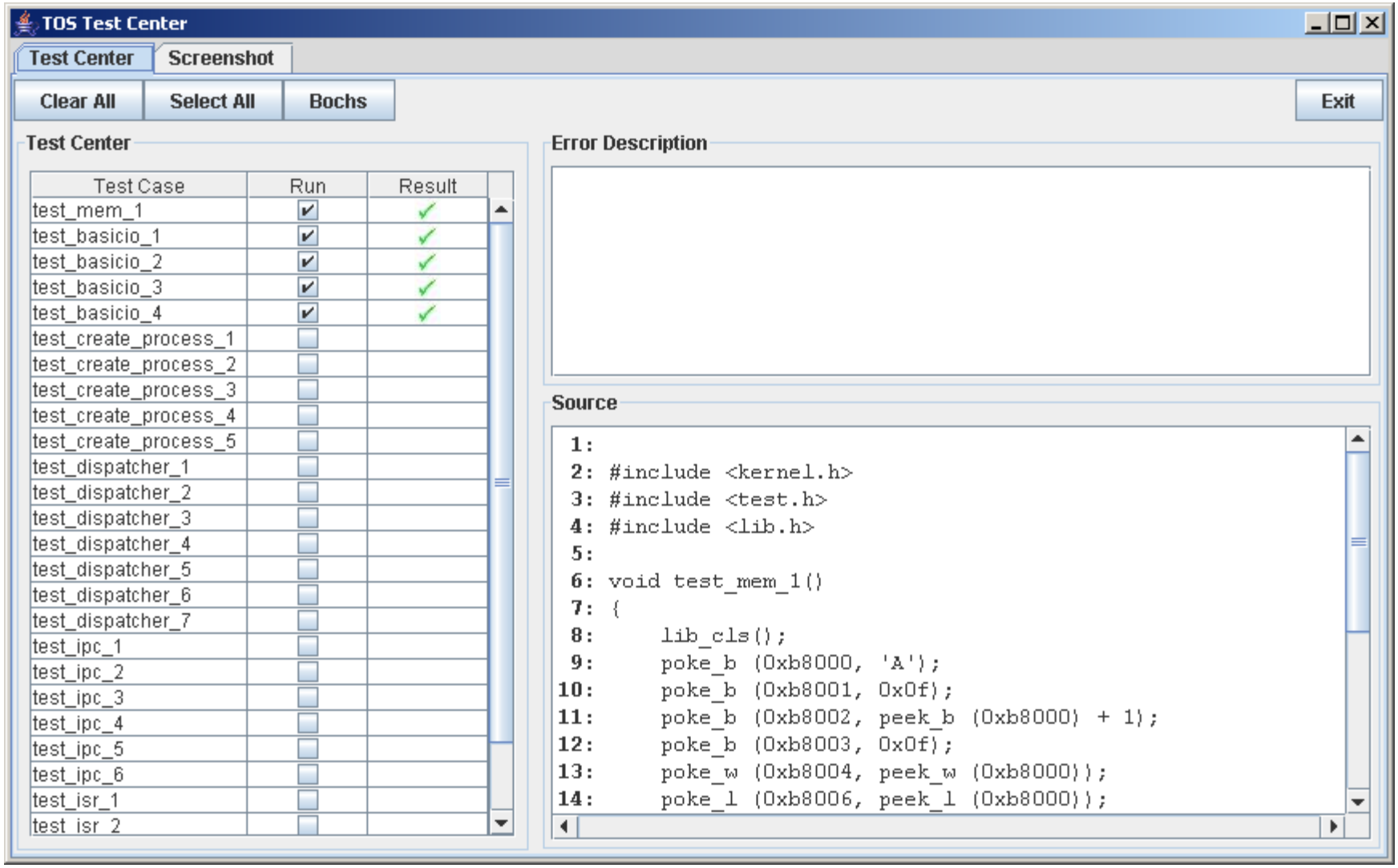
Running Test Cases

- Compile TOS with test cases enabled. This can be accomplished by typing:
  ```
  cd ~/tos
  make tests
  ```
- Run the TOS Test Center (TTC) via:
  `./run-ttc.sh`
- Once the TTC launched, select the test cases you want to run (e.g., `test_mem_1`)
- Next click the “Bochs” button at the top of the TTC. This will launch Bochs.
- Once the tests completed, the TTC will show which tests failed and which succeeded.

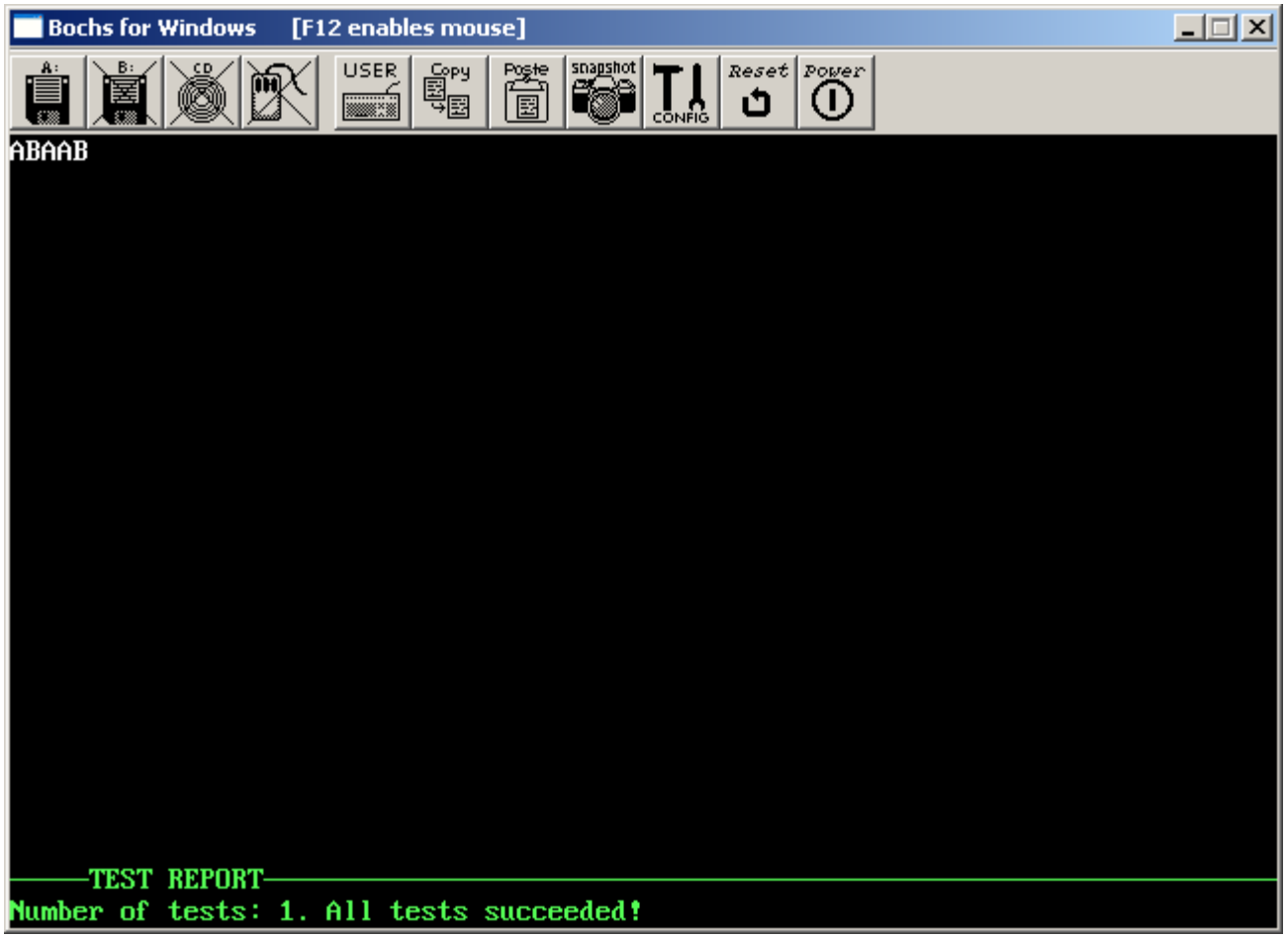
Sample Screenshot



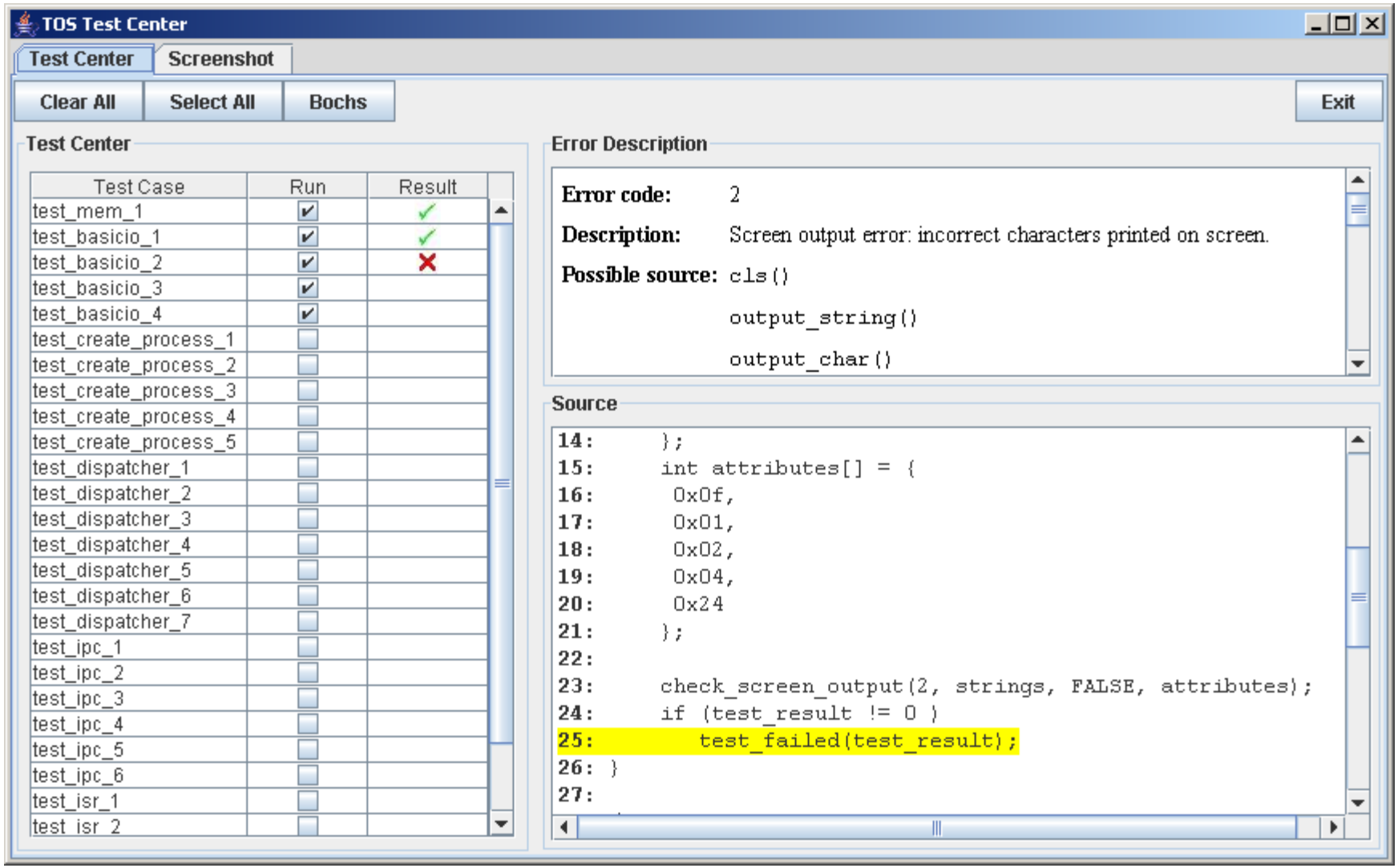
Successful Run



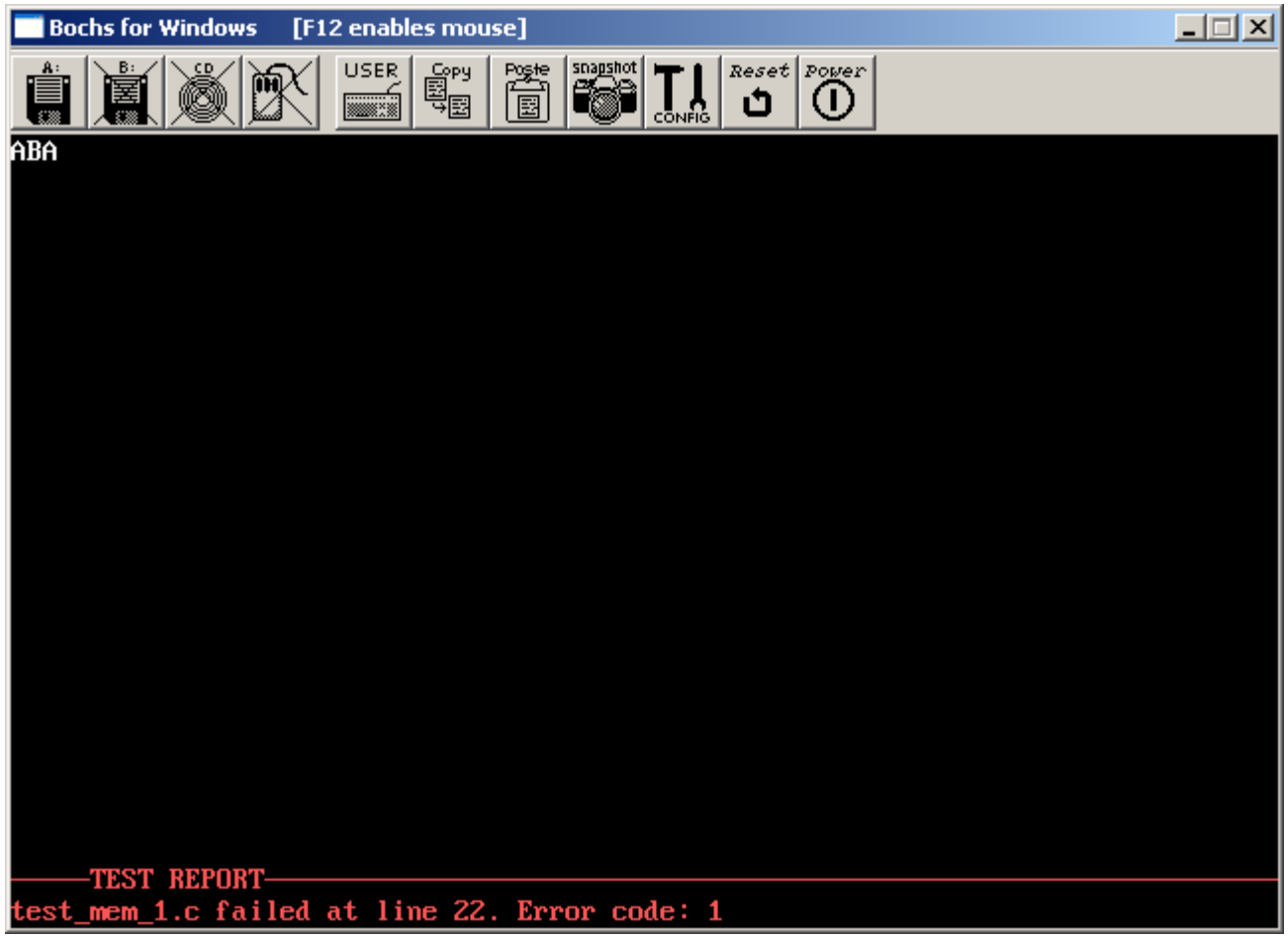
Successful Test



Unsuccessful Run



Unsuccessful Test



Multiple Test Cases

- If a test case is passed and there are more selected, then Bochs and TTC continue top-down to the next test case without stopping. This allows multiple test cases to be tested with one run.
- The TTC will stop at the first erroneous test case and will display an error message.

Notes on TOS Test Cases

- The assignment slides indicate which test cases should be run for that particular assignment.
- If a test cases fails, it will print out an error code. The HTML page `~/tos/test/messages.html` explains all the error codes.
- If a test case fails, it often helps to study the implementation of the test case to understand what it is doing. Note that some helper code is located in `~/tos/test/common.c`
- If all test cases succeed, it doesn't necessarily mean that the implementation is bug free (testing vs. verification)
- If TOS crashes (without printing any error codes), you'll have to employ a debugging technique explained on the following slides.
- Always run previous test cases. If one test succeeds today, it may fail tomorrow due to some changes you made (called a regression)

Debugging hints

- If something goes wrong in TOS, the whole machine usually crashes.
- In that case, the first priority is to locate the line in your program that causes the crash.
- This can be done by carefully inserting an endless loop into your program:

```
statement_1;
statement_2;
Crash_causing_statement;
statement_3;
statement_4
while(1);
```

⟹ System crashes

```
statement_1;
statement_2;
while(1);
Crash_causing_statement;
statement_3;
statement_4;
```

⟹ System does not crash

```
statement_1;
statement_2;
 Crash_causing_statement;
while(1);
statement_3;
statement_4;
```

⟹ System crashes

Debugging hints

- Once the statement that causes the crash has been isolated, the next step is to understand why it crashes.
- This requires us to know the values of C-variables.
- Use `kprintf()` to print the value of C-variables.

Debugging hints

- Another powerful debugging tool are assertions.
- An assertion defines a condition that you expect to be true at a certain program location.
- The assertion is tested at runtime.
- If the assertion evaluates to FALSE, a detailed error message is given.
- TOS provides a courtesy implementation of assertions (i.e., you don't have to implement it).
- However: the assertions provided in TOS assume a working `output_string()` function. This means you can only use assertions once you have implemented this basic output function.
- Assertions are implemented through function `assert()` defined in `~/tos/include/assert.h`
- `assert.h` is automatically included if you include `kernel.h`

Assertion Example (1)

```
Node* elem;
elem = alloc_data_item();
assert(elem != (Node*) 0);
```

- This piece of code is based on the example given earlier for dynamic memory management techniques
- This assertion will fail if there should not be any more free data items

Assertion Example (2)

```
void move_cursor(WINDOW *wnd, int x, int y)
{
      assert(x >=0 && x < wnd->width);
      …
}
```

- It is often useful to check input parameters.
- The code above checks that input parameter ‘x’ is within the allowed boundaries.

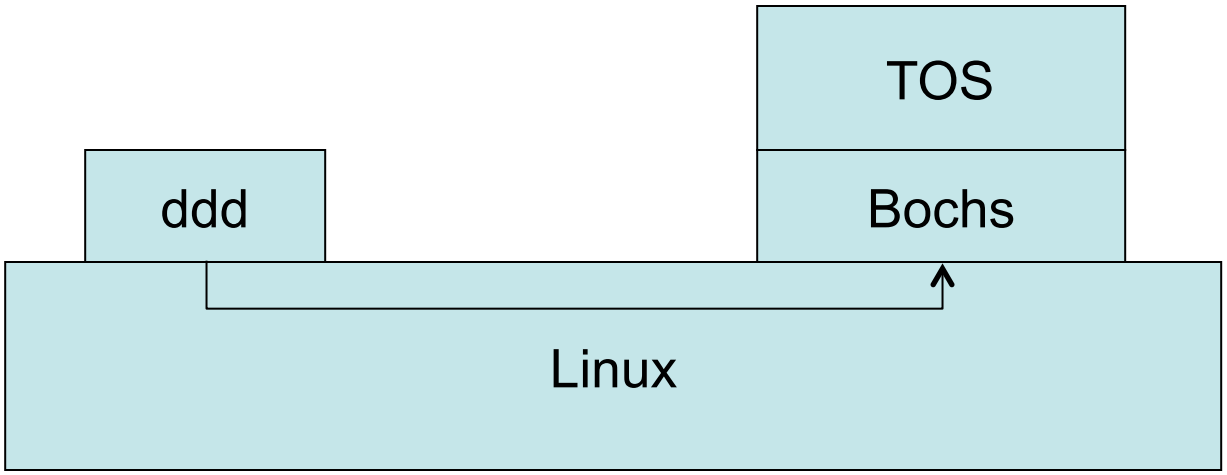
Assertion Tips

- Use many assertions (assertions are your friend!).
- Never remove assertions once you have added them to your program.
- `assert()` is very useful in testing the validity of input parameters of functions.
- `assert(0)` always fails. Useful to mark locations in your program that should never be reached (e.g. default case of a switch-statement).

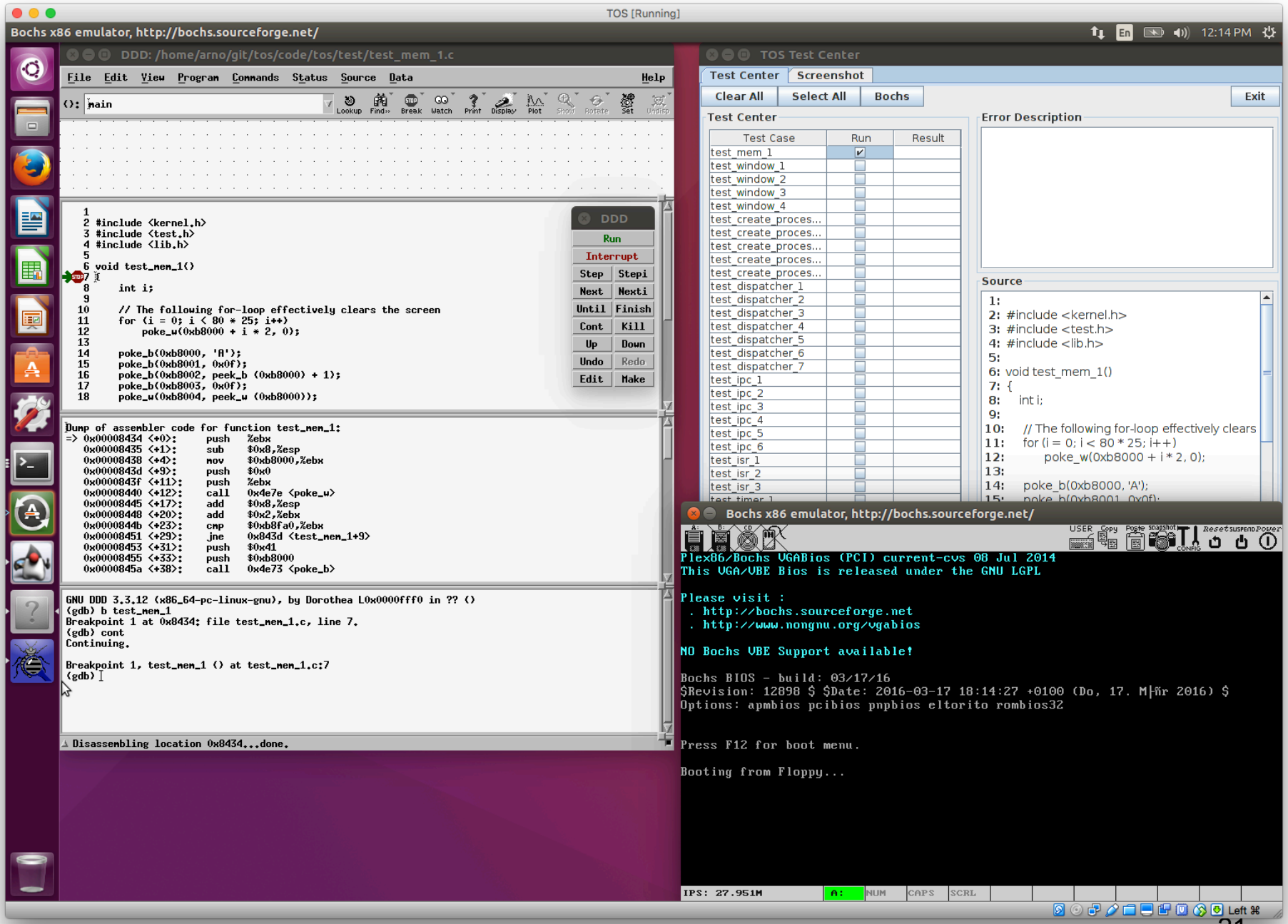
Debugging with gdb

- The latest version of TOS supports debugging via GNU's gdb.
- This requires a special version of Bochs that enables gdb debugging. The TOS installation script will automatically generate this special version.
- TOS's Makefile will generate a TOS kernel image with debug information called tos-debug.img.
- This also requires a special version of the .bochsrc file 'gdb-bochsrc' that the installation script will download.
- The recommended GUI frontend for gdb is a debugger called ddd. See this link for a user manual: https://www.gnu.org/software/ddd/manual/

Remote Debugging with ddd



- Bochs will wait on TCP port 1234 for a remote debugger (such as ddd) to connect.
- While Bochs waits for ddd, TOS is stopped.
- Running ddd will establish a TCP connection to Bochs and ddd remote-controls the execution of TOS.
- ddd uses a specially compiled version of the kernel called tos-debug.img in order to extract the symbol table.


TOS [Running]
Bochs x86 emulator, http://bochs.sourceforge.net/
12:14 PM
DDD: /home/arno/git/tos/code/tos/test/test_mem_1.c
File Edit View Program Commands Status Source Data Help
main
Lookup Find Break Watch Print Display Plot Show Rotate Set Undisp
1
2 #include <kernel.h>
3 #include <test.h>
4 #include <lib.h>
5
6 void test_mem_1()
7 {
8 int i;
9
10 // The following for-loop effectively clears the screen
11 for (i = 0; i < 80 * 25; i++)
12 poke_w(0xb8000 + i * 2, 0);
13
14 poke_b(0xb8000, 'A');
15 poke_b(0xb8001, 0x0f);
16 poke_b(0xb8002, peek_b (0xb8000) + 1);
17 poke_b(0xb8003, 0x0f);
18 poke_w(0xb8004, peek_w (0xb8000));
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
Dump of assembler code for function test_mem_1:
=> 0x00008434 <+0>: push %ebx
0x00008435 <+1>: sub $0x8,%esp
0x00008438 <+4>: mov $0xb8000,%ebx
0x0000843d <+9>: push $0x0
0x0000843f <+11>: push %ebx
0x00008440 <+12>: call 0x4e7e <poke_w>
0x00008445 <+17>: add $0x8,%esp
0x00008448 <+20>: add $0x2,%ebx
0x0000844b <+23>: cmp $0xb8fa0,%ebx
0x00008451 <+29>: jne 0x843d <test_mem_1+9>
0x00008453 <+31>: push $0x41
0x00008455 <+33>: push $0xb8000
0x0000845a <+38>: call 0x4e73 <poke_b>
GNU DDD 3.3.12 (x86_64-pc-linux-gnu), by Dorothea L0x0000fff0 in ?? ()
(gdb) b test_mem_1
Breakpoint 1 at 0x8434: file test_mem_1.c, line 7.
(gdb) cont
Continuing.
Breakpoint 1, test_mem_1 () at test_mem_1.c:7
(gdb)
Disassembling location 0x8434...done.
TOS Test Center
Test Center Screenshot
Clear All Select All Bochs Exit
Test Center
Test Case Run Result
test_mem_1
test_window_1
test_window_2
test_window_3
test_window_4
test_create_proces...
test_create_proces...
test_create_proces...
test_create_proces...
test_create_proces...
test_dispatcher_1
test_dispatcher_2
test_dispatcher_3
test_dispatcher_4
test_dispatcher_5
test_dispatcher_6
test_dispatcher_7
test_ipc_1
test_ipc_2
test_ipc_3
test_ipc_4
test_ipc_5
test_ipc_6
test_isr_1
test_isr_2
test_isr_3
Error Description
Source
1:
2: #include <kernel.h>
3: #include <test.h>
4: #include <lib.h>
5:
6: void test_mem_1()
7: {
8: int i;
9:
10: // The following for-loop effectively clears
11: for (i = 0; i < 80 * 25; i++)
12: poke_w(0xb8000 + i * 2, 0);
13:
14: poke_b(0xb8000, 'A');
Bochs x86 emulator, http://bochs.sourceforge.net/
Plex86/Bochs VGABios (PCI) current-cvs 08 Jul 2014
This VGA/VBE Bios is released under the GNU LGPL
Please visit :
. http://bochs.sourceforge.net
. http://www.nongnu.org/vgabios
NO Bochs VBE Support available!
Bochs BIOS - build: 03/17/16
$Revision: 12898 $ $Date: 2016-03-17 18:14:27 +0100 (Do, 17. Mär 2016) $
Options: apmbios pcibios pnpbios eltorito rombios32
Press F12 for boot menu.
Booting from Floppy...
IPS: 27.951M
A:
NUM
CAPS
SCRL
Left ⌘

Debugging with ddd

1. **`make tests`**
2. **`./run_ttc.sh`**
3. Select **`test_mem_1`** test
4. Open new terminal
5. **`bochs -q -f gdb-bochsrc`**
6. Open new terminal
7. **`ddd`**
8. In the bottom portion of ddd, type "**`b test_mem_1`**". This will set a breakpoint in function test_mem_1.
9. In ddd, click on "Continue" in the floating dialog.

Note: all the above commands need to be run in the 'tos' directory.