

# Debugging by Discipline

## 3. Keep a logbook

```
$ mkdir logs
$ vim logs/short-description
Start by describing what was
seen when the problem first
appeared.
To reproduce: the steps that
can be taken to reproduce
it. Bonus: make a script, put
it in the logs/ directory and
give it the same name as
this file.
Hypothesis: what I suspect
Experiment: how I can test it
Observation: what did I see?
Conclusion: what I learned.
```

## 4. Use mercurial (or git):

Start for the first time:

```
$ hg init # in project dir
$ vim .hgignore
syntax: glob
*.o
*~
*.out
*.err
```

Add main.c:

```
$ hg add main.c
```

Save changes:

```
$ hg commit
```

See changes:

```
$ hg log
```

Go back to version 4:

```
$ hg update 4
```

Go to latest version:

```
$ hg update -C
```

1. Make simple scripts to compile, reproduce and test whether a bug occurs or not:

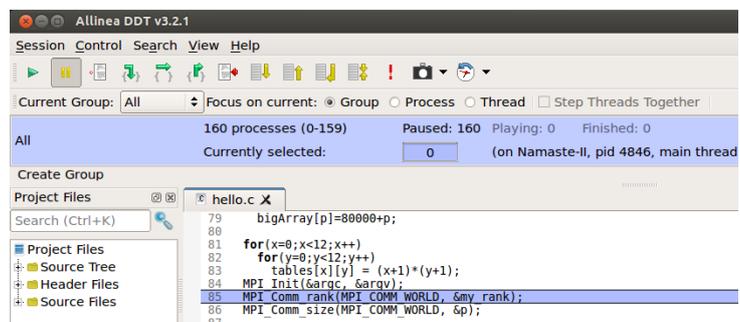
```
$ wget http://content.allinea.com/downloads/test-script.sh
$ vim test-script.sh # instructions are in the file
```

Example of such a script in use:

```
$ ./test-script.sh
Compiling hello with debug information
Submitted to the queue with id srv03-ib.20374
Waiting for srv03-ib.20374 to start
Job started at Sun Sep 30 20:23:03 CEST 2012
Waiting for srv03-ib.20374 to finish
Job finished at Sun Sep 30 20:37:42 CEST 2012
Program crashed: FAIL
```

2. Set up your debugger and check it works before you need it:

```
$ module load ddt
$ cp -r /path/to/ddt-directory/examples .
$ cd examples
$ mpicc -g -O0 hello.c -o hello
$ ddt -np 160 -start ./hello
```



Problem? Email [support@allinea.com](mailto:support@allinea.com) - we will fix it!

Bonus: keep a DDT session file for each logbook file:  
Session -> Save session... -> logs/short-description.ddt

# Debugging by Inspiration

## Sources of inspiration

### 1. Search your logbook

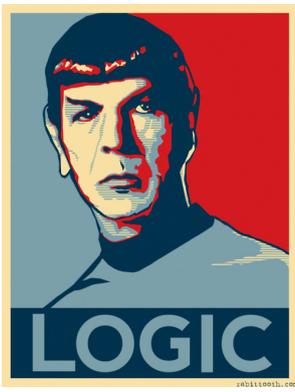
```
$ grep -ir r_send logs/*
segfault-at-64: Conclusion:
increase buffer size in r_send
to stop MPI buffering it
deadlock-at-512: Conclusion:
reduce buffer size in r_send
and hope MPI buffers it
```

### 2. Talk to a duck



"So I'm sure it can't be a mutex problem, because it's locked in both the send and the reciev-oh, wait! That might be it!"

### 3. Be blessed with a logical mind



The rest of us may have to try harder.

Also be lucky enough to have over two decades of some of the finest education in the history of our species!

Listen to your instincts and test them  
 Suspect the bounds are incorrect somewhere in the loop? Make Allinea DDT log them all:

```
$ ddt -offline log.html -n 64 -trace-at watchpoint.c:
46,i,j,k,numprocs ./watchpoint
```

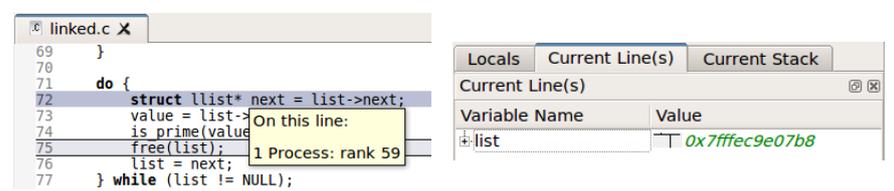
#	Time	Tracepoint	Processes	Values
1	00:11.029	main (watchmatrix.c:46)	0-63	i: 0 numprocs: 64 j: 0 k: 0
2	00:11.029	main (watchmatrix.c:46)	0-63	i: 0 numprocs: 64 j: 0 k: 1
3	00:11.029	main (watchmatrix.c:46)	0-63	i: 0 numprocs: 64 j: 0 k: 2
4	00:11.029	main (watchmatrix.c:46)	0-63	i: 0 numprocs: 64 j: 0 k: 3

Not true? Add it to your log and try something else!

Just look at the problem (for 10 minutes)

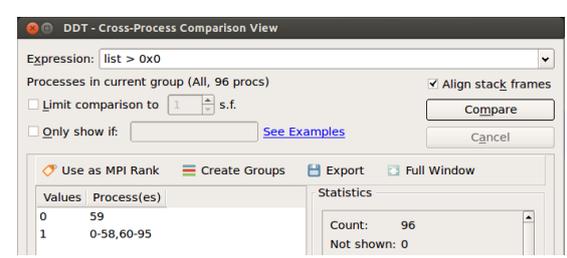
```
$ ddt -n 160 -start programs/linkedlist
```

The code view shows where processes stop and current Line and Locals let you explore the state:



The screenshot shows a code editor with a linked list loop. A tooltip indicates 'On this line: 1 Process: rank 59'. The Locals panel shows the 'list' pointer at address 0x7ffec9e07b8.

That's interesting - why does one process have a much lower address for "list"? Click on the graph to see a more detailed comparison:



The screenshot shows the 'Cross-Process Comparison View' for the expression 'list > 0x0'. It displays a comparison across 96 processes, with a 'Count: 96' and 'Not shown: 0'.

Create groups, add tracepoints and explore freely:



The screenshot shows the DDT interface with process groups defined for 'list > 0x0=0' (rank 59) and 'list > 0x0=1' (ranks 0-58, 60-95). The Locals panel shows variables like 'next' at address 0x7ffec9e07c8 and 'rank' at 0.

Problem? Email [support@allinea.com](mailto:support@allinea.com) - we will fix it!

# Debugging by Magic

## Magical ingredients

1. Keep a logbook in your project directory, also in source control. Use one text file for each bug investigated.

2. Set up Allinea DDT

```
$ module load ddt
```

User guide also available online:

<http://www.allinea.com/products/ddt-support/>

3. Try static analysis

```
$ /path/to/ddt/libexec/cppcheck
$ /path/to/ddt/libexec/ftnchek
```

On by default in Allinea DDT.

4. MPI checker tools

Marmot: <http://www.hlrs.de/organization/av/spmt/research/marmot/downloads/>

MUST is in beta until SC12

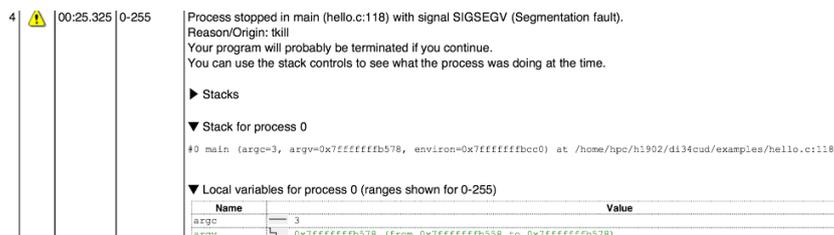
5. Make Mercurial find the bug for you:

```
$ hg bisect --reset
$ hg bisect --bad # broken now
$ hg bisect --good 4 # was ok
$ hg bisect -c ./test-script.sh
...
The first bad revision is:
changeset: 6:a06eae2cf30
```

```
$ hg log -pr 6 # shows the bug!
```

## Program crashes (segfault):

```
$ ddt -offline log.html -n 256 examples/hello arg1 arg2
```



## Program gets stuck (deadlock):

```
$ ddt -n 128 -start programs/loop arg1 arg2
```

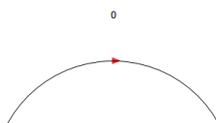
Press play, wait for the deadlock, then hit pause



Check the parallel stack view and variables:

Processes	Function	Variable Name	Value
1	main (loop.c:48)	send_buffer	0xe90af008
1	passitOn (loop.c:27)	to	1
1	PMPI_Send		
127	main (loop.c:51)		
127	PMPI_Barrier		

Or use the MPI message queues window:



Text	Communicator	Queue	Pointer	From (local)	From (global)	To (local)
1 Send: 0x8...	MPI_COMM_WORLD	Send	0x0	0	0	1

## Suspected memory errors:

```
$ ddt -n 256 examples/hello arg1 arg2
```

**Memory Debugging:** Balanced, No guard pages, Backtraces, Preload Details...

Heap Overflow/Underflow Detection

Add guard pages to detect out of bounds heap access

Guard pages: 1 Add guard pages: After



Problem? Email [support@allinea.com](mailto:support@allinea.com) - we will fix it!

# Debugging by Science

## An example logbook

Seen: "Signal: Segmentation fault(11). Failing at address: 0x8". Reproduce: `mpirun -n 64`  
 Debugger: Memory error detected in main (linked.c:75) - "a previous write overwrite the reserved memory."

Hypothesis: Classic off-by-one.  
 Prediction: Adding guard pages will show where the bad write takes place.

Experiment: Run DDT again with guard pages set to "After", 1 page.

Observation: 61 procs stop at line 65, with an invalid "list" pointer (0x7fff3cb5f00). last and list->next are both 0x0.  
 Conclusion: The value of list is \*already\* invalid here!

Hypothesis: Allocation failed  
 Prediction: The call to calloc doesn't assign enough memory.  
 Experiment: Run to line 65 and examine the "list".

Observation: View pointer details claims the size allocated is 0 bytes.  
 Conclusion: Confirmed. The man page for calloc shows we are allocating an array of size 0.

Hypothesis: Calloc is the wrong function to use here.  
 Prediction: Using malloc instead fixes the bug.  
 Experiment: Replace both calls.  
 Observation: 1 proc segfaults at line 72 with list = 0x0.  
 Conclusion: Confirmed, this crash is from another bug!

## 1. Suggest a falsifiable hypothesis

The line "list->next = last" crashes because the call to calloc isn't allocating enough memory here:

```
64 list = calloc(sizeof(struct llist), 0);
65 list->next = last; // <- list->next crashes
66 list->value = value;
```

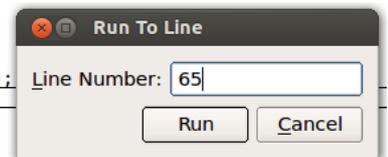
## 2. Make a testable prediction

View pointer details on line 65 says "invalid memory"

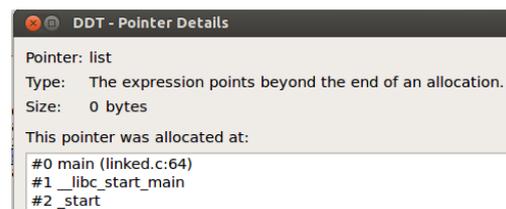
## 3. Experiment

Run to line 65:

```
61 while ((value = rand() % size) != 0) {
62     struct llist* last = list;
63     list = calloc(sizeof(struct llist), 0);
64     list->next = last;
65     list->value = value;
66 }
67
68
```



Use "View pointer details" on "list":



## 4. Observe and record the results

Calloc is returning a valid pointer to 0 bytes of memory, which isn't enough for list->next to be valid.

## 5. Form a conclusion

Confirmed. The calloc man page tells us:

If nmemb or size is 0, then calloc() returns either NULL, or a unique pointer that can later be successfully passed to free().

We are passing size = 0 (parameter 2) - a mistake!

## Repeat until solved

A new hypothesis - is calloc the wrong function here?