Session Outline

- How to diagnose job or program failures on SHARCNET systems
- How to identify and correct common programming bugs
- The use of *gdb* for debugging serial programs
- The use of *DDT* for debugging parallel programs
Identifying Bugs and Errors

• Typical signs that your program is buggy include:
  
  • It fails to complete *(crashes)*
  • It produces incorrect output (!*#%?)
  • It fails to progress *(hangs)*
Diagnosing Job Failures

- Job (process) exit status (code)
- Job Scheduling and output handling
  - LSF vs. Torque/Maui/Moab, SQ
- Job Identifier (jobid)
- Web Portal Jobs Table
- The system's view of a job:
  - sqjobs -l || bhist || qstat
- Inspecting running jobs
  - sqjobs -L || SHARCNET Ganglia
# Common Error Signals

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>OS signal #</th>
<th>OS signal name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating point exception</td>
<td>8</td>
<td>SIGFPE</td>
<td>The program attempted an arithmetic operation with values that do not make sense (e.g. divide by zero)</td>
</tr>
<tr>
<td>Segmentation fault</td>
<td>11</td>
<td>SIGSEGV</td>
<td>The program accessed memory incorrectly (e.g. accessing an array beyond its declared bounds)</td>
</tr>
<tr>
<td>Aborted</td>
<td>6</td>
<td>SIGABRT</td>
<td>Generated by the runtime library of the program or a library it uses, after having detected a failure condition</td>
</tr>
<tr>
<td>Kill</td>
<td>9</td>
<td>SIGKILL</td>
<td>The job management system terminates a job when it exceeds resource limits (e.g. Runtime or memory)</td>
</tr>
</tbody>
</table>

Note: job exit status is typically 0 (success), negative (system issue) or 128+"OS signal #" when there is a failure
# Web Portal Jobs Table

## Jobs

No queued jobs [configure]
No current jobs [configure]
Jobs finished [hide] [configure]

<table>
<thead>
<tr>
<th>System</th>
<th>Job ID</th>
<th>Num CPUs</th>
<th>State</th>
<th>Started</th>
<th>Completed</th>
<th>% CPU</th>
<th>Alloc. Time</th>
<th>User Time</th>
<th>Exit Status</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>whale</td>
<td>3,229,987</td>
<td>1</td>
<td>killed</td>
<td>2010-05-27 12:11</td>
<td>2010-05-27 12:12</td>
<td>94%</td>
<td>1.3m</td>
<td>1.2m</td>
<td>14</td>
<td>input_sequences</td>
</tr>
<tr>
<td>whale</td>
<td>3,229,846</td>
<td>1</td>
<td>failed</td>
<td>2010-05-27 12:08</td>
<td>2010-05-27 12:08</td>
<td>1%</td>
<td>2.0s</td>
<td>0.0s</td>
<td>256</td>
<td>NC_005945.faa</td>
</tr>
<tr>
<td>whale</td>
<td>3,229,784</td>
<td>1</td>
<td>failed</td>
<td>2010-05-27 12:07</td>
<td>2010-05-27 12:07</td>
<td>0%</td>
<td>2.0s</td>
<td>0.0s</td>
<td>256</td>
<td>mafft --auto input_sequences</td>
</tr>
<tr>
<td>hound</td>
<td>194,242</td>
<td>1</td>
<td>failed</td>
<td>2010-05-27 11:19</td>
<td>2010-05-27 11:19</td>
<td>0%</td>
<td>0.0s</td>
<td>0.0s</td>
<td>1</td>
<td>date</td>
</tr>
<tr>
<td>whale</td>
<td>3,224,233</td>
<td>1</td>
<td>done</td>
<td>2010-05-27 10:19</td>
<td>2010-05-27 10:19</td>
<td>0%</td>
<td>2.0s</td>
<td>0.0s</td>
<td>0</td>
<td>date</td>
</tr>
<tr>
<td>brown</td>
<td>8,606,033</td>
<td>1</td>
<td>done</td>
<td>2010-05-27 10:13</td>
<td>2010-05-27 10:13</td>
<td>0%</td>
<td>1.0s</td>
<td>0.0s</td>
<td>0</td>
<td>date</td>
</tr>
<tr>
<td>hound</td>
<td>194,002</td>
<td>1</td>
<td>done</td>
<td>2010-05-27 10:08</td>
<td>2010-05-27 10:08</td>
<td>0%</td>
<td>0.0s</td>
<td>0.0s</td>
<td>0</td>
<td>date</td>
</tr>
<tr>
<td>whale</td>
<td>3,109,154</td>
<td>1</td>
<td>done</td>
<td>2010-05-25 12:53</td>
<td>2010-05-25 13:23</td>
<td>99%</td>
<td>30m</td>
<td>30m</td>
<td>0</td>
<td>mafft --auto input_sequences</td>
</tr>
<tr>
<td>whale</td>
<td>3,107,969</td>
<td>1</td>
<td>failed</td>
<td>2010-05-25 12:29</td>
<td>2010-05-25 12:30</td>
<td>0%</td>
<td>4.0s</td>
<td>0.0s</td>
<td>256</td>
<td>NC_005945.faa</td>
</tr>
<tr>
<td>whale</td>
<td>3,107,953</td>
<td>1</td>
<td>failed</td>
<td>2010-05-25 12:29</td>
<td>2010-05-25 12:29</td>
<td>0%</td>
<td>3.0s</td>
<td>0.0s</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>narwhal</td>
<td>1,173,359</td>
<td>2</td>
<td>failed</td>
<td>2010-05-17 16:23</td>
<td>2010-05-17 16:26</td>
<td>0%</td>
<td>4.8m</td>
<td>0.1s</td>
<td>3,840</td>
<td>MPI_DT.x</td>
</tr>
<tr>
<td>narwhal</td>
<td>1,173,150</td>
<td>2</td>
<td>failed</td>
<td>2010-05-17 16:14</td>
<td>2010-05-17 16:17</td>
<td>0%</td>
<td>5.3m</td>
<td>0.4s</td>
<td>256</td>
<td>MPI_DT.x</td>
</tr>
<tr>
<td>narwhal</td>
<td>1,172,561</td>
<td>2</td>
<td>failed</td>
<td>2010-05-17 14:49</td>
<td>2010-05-17 14:52</td>
<td>0%</td>
<td>6.0m</td>
<td>0.1s</td>
<td>3,840</td>
<td>MPI_DT.x</td>
</tr>
<tr>
<td>narwhal</td>
<td>1,171,939</td>
<td>2</td>
<td>failed</td>
<td>2010-05-17 14:24</td>
<td>2010-05-17 14:27</td>
<td>0%</td>
<td>5.2m</td>
<td>0.4s</td>
<td>256</td>
<td>MPI_DT.x</td>
</tr>
<tr>
<td>requin</td>
<td>629,409</td>
<td>2</td>
<td>killed</td>
<td>2010-05-17 14:24</td>
<td>2010-05-17 14:24</td>
<td>0.0s</td>
<td>0.0s</td>
<td>0</td>
<td>a.out</td>
<td></td>
</tr>
</tbody>
</table>
The First "Bug"

9/9

0800  Auton started
1000  Auton stopped - auton ✓

1300 (033) MP-MC 13057249.5 (5) 4.015925059 (-2)
(033) PRO 2 2.120476415
cond 2.130576455

Relays 6-2 in 033 failed special speed test
No relay.
Relays changed.

1100  Started Cosine Tape (Sine check)
1525  Started Multi-Adder Test.
1545  Relay #70 Panel F (moth) in relay.

First actual case of bug being found.

9 September 1947
Diagnosing the situation

- pay attention to compiler warnings
- inspect the job exit code in the web portal
- look at the job output file
  - may indicate a problem with the state of the program or a lack of progress
  - may contain a runtime error message, signal from the operating system or error from the job management system that helps identify the problem
Common Bugs

- **Arithmetic**
  - infinities, out of range
- **Logic**
  - infinite loop
- **Syntax**
  - wrong operator, arguments
- **Resource starvation**
  - memory leak
- **Parallel**
  - race conditions
  - deadlock
- **Misuse**
  - wrong initial conditions / insufficient checking / variable initialization
Floating Point Exceptions

- compilers/runtimes handle floating point exceptions differently
  - Some allow turning this on/off during compilation
  - Pathscale: -TENV:simd_*mask=OFF
  - Intel (fortran only!): -fpe0
  - gfortran: -ffpe-trap=invalid,zero,overflow
- Can also trap exceptions via library functions
  - glibc: feenableexcept()
    - compile and link to trapfpe.c code
Correcting Bugs

- if no error message is generated or if the message is insufficient to identify the problematic code one can use a debugger
- A debugger is a program that allows one to manipulate and inspect a second program as it is running
- Typically, the program should be compiled to include a symbol table (often `-g`) if you are going to run it in a debugger
gdb

- GNU Project Debugger
- Freely available, runs on most *nix systems, open source
- Works with a wide variety of languages
- Demonstration loosely following this tutorial in our help wiki:
  - Using gdb in the Online Training Centre
Debugging tips

• If your bug isn't repeatable:
  • Race condition? Randomness?
  • If a bug only appears with certain configurations / initial conditions it may be due to resource starvation or incorrect usage

• When reporting problems with the underlying system/software, provide a simple (and quick) test case, if possible

• Incorrect validation of input can result in many different errors!
Debugging tips

- Most Fortran compilers support runtime checking for out-of-bound array accesses, eg.
  - $ f90 -ffortran-bounds-check outbounds.f90

- Ensure that variables are defined with sufficient precision (overflow/underflow)

- Some MPIs support reporting more diagnostic information (eg. linking with hp-mpi's `-ldmpi`)

- Functionality in SHARCNET job submission to automatically generate a backtrace (LSF only):
  - `sqsub -backtrace ...`
DDT

- In addition to features in gdb:
  - GUI debugger (tabbed interface)
  - Shows multiple source files w/ syntax highlighting
  - Support for MPI, threaded and GPGPU debugging
    - Independant and group process/thread control
      (breakpoints, syncronization, comparisons)
  - inspection of variables (visualization, watches, checking pointers)
  - Visualization of MPI message queues
  - memory debugging!