A GDB Tutorial

Stojche Nakov

CS Departement
Princeton University

Sub-Group meeting, November 8, 2021

GDB: The GNU Project Debugger

GDB overview

- First release in 1986.
- Supports variouse languages: **C, C++, Fortran, Objective-C** etc.
- Multiple GUI extensions (DDD, kDgb, Nemiver ...).

GDB: The GNU Project Debugger

GDB overview

- First release in 1986.
- Supports variouse languages: **C, C++, Fortran, Objective-C** etc.
- Multiple GUI extensions (DDD, kDgb, Nemiver ...).

GDB's Purpose

- Allows you to see what is going on 'inside' another program while it executes.
- Makes the porgram stop on specific conditions.
- Examine what has happened.
- Change things in your program.

GDB Basics

Compiling

Must be compiled using the flag "**-g**". Also it is recomended that the optimization flags are removed and the "**-Og**" flag is used.

To run the program use "**gdb –args ./exe arg1 arg2**"

GDB Basics

Compiling

Must be compiled using the flag "**-g**". Also it is recomended that the optimization flags are removed and the "**-Og**" flag is used.

To run the program use "**gdb –args ./exe arg1 arg2**"

Basic commands

- **run / r** – Begins running the program. If the program is already active, it restarts it.
- **coninue / c** – Continues the execution of the program.
- **break / b** – Sets a breakpint.
- **print / p item** – Prints an item (value of variable, function, structure, class etc).
- **backtrace / b** – Shows the calling squence.
- **list / l** – Prints the code.
- **frame / f #** – Changes to the given frame.

Parallel Debugging

Multi-Threading

- **info threads / t** – Prints information for each thread.
- **thread / t #** – Switches to the thread.
- **thread apply all "cmd" / t a a "cmd"** – Applies "cmd" to all threads (usually **backtrace** is used).

Parallel Debugging

Multi-Threading

- **info threads / t** – Prints information for each thread.
- **thread / t #** – Switches to the thread.
- **thread apply all "cmd" / t a a "cmd"** – Applies "cmd" to all threads (usually **backtrace** is used).

Distributed memory

MPI is **SIMD** parallel model.

Parallel Debugging

Multi-Threading

- **info threads / t** – Prints information for each thread.
- **thread / t #** – Switches to the thread.
- **thread apply all "cmd" / t a a "cmd"** – Applies "cmd" to all threads (usually **backtrace** is used).

Distributed memory

MPI is **MIMD** parallel model.

Parallel Debugging

Multi-Threading

- **info threads / t** – Prints information for each thread.
- **thread / t #** – Switches to the thread.
- **thread apply all "cmd" / t a a "cmd"** – Applies "cmd" to all threads (usually **backtrace** is used).

Distributed memory

MPI is **MIMD** parallel model.

```
mpirun -np 2 ./exe1 : -np 2 ./exe2 : -np 2 ./exe3
```

Parallel Debugging

Multi-Threading

- **info threads / t** – Prints information for each thread.
- **thread / t #** – Switches to the thread.
- **thread apply all "cmd" / t a a "cmd"** – Applies "cmd" to all threads (usually **backtrace** is used).

Distributed memory

MPI is **MIMD** parallel model.

mpirun -np 1 **gdb ./exe1** : -np 1 ./exe1 : -np 2 ./exe2 : -np 2 ./exe3

Parallel Debugging

Multi-Threading

- **info threads / t** – Prints information for each thread.
- **thread / t #** – Switches to the thread.
- **thread apply all "cmd" / t a a "cmd"** – Applies "cmd" to all threads (usually **backtrace** is used).

Distributed memory

MPI is **MIMD** parallel model.

Alternatively, open one xterm per process:
mpirun -np 4 **xterm -e gdb ./exe**

GDB Advance

Launching gdb

‘

- Execute **gdb**, and then specify the exec file using the command **file**.
- Arguments can be supplied to the **run** command.
- By setting **ulimit -c unlimited**, and then use **gdb core_file exec_file**.
- Attaching to a running process: **gdb attach $pid**.
- **vgdb**: Valgrind + GDB. Launch valgrind with the following arguments: **–vgdb=yes –vgdb-error=0**.
  Then launch gdb with the executable and copy paste the proposed commands in the first terminal.

GDB Advance

- Conditional breakpoints: **break if condition — break if not condition**
- **watch, rwatch, awatch** commands.
- **printf** is availble.
- **call** allows to call functions.
- **Defining helpers** in the .gdbinit file.
- **info breakpoint** prints the breakpoints. **command # cmdS end**, will execute all cmdS on each breakpooint.
- **set var=value** sets the value of **var**.
- **Convenience Functions**, like **$_caller_is**, for example.
- **Reverse debugging** with the **record** command. Hint: **set can-use-hw-watchpoints 0**, if planning to use watchpoints in reverse debugging.