GDB (The GNU DeBugger)

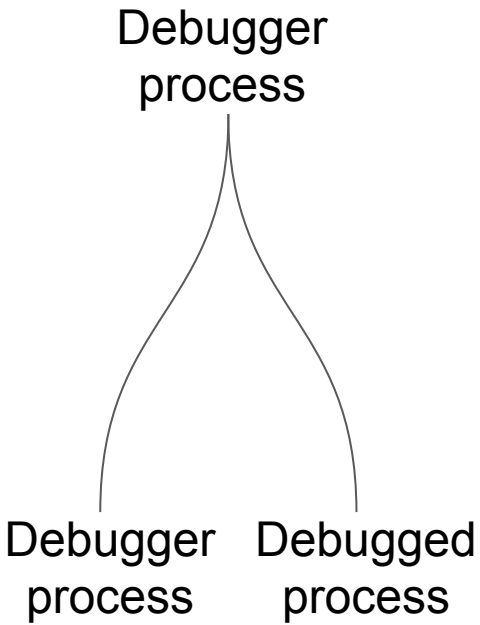
Amitabha Sanyal

Starting GDB

- **The GDB model**
  - The debugger process and the debugged process
- **Basic usage (now)**
  - Starting GDB
  - Setting breakpoints
  - Stepping over the program: next, step, finish
  - Examining data
  - Watchpoints
  - Examining the call stack
- **Advanced gdb (when GDB returns)**
  - Working with core dumps
  - Writing printers
  - Discovering memory leaks through valgrind.

The GDB model

Reads user input, and executes it:
a. Show information about the debugger state
b. Show information about the debugged process.
c. Sets breakpoints (highly non-trivial)
d. Passes control to debugged process.



Executes till breakpoint or end.
Forced to give up control which passes to debugger.

Compiling the program for debugging

- Assume that the program being debugged is `buggyprog.cpp`
  - Compile the program as

```
$ g++ -g buggyprog.cpp -o buggyprog
```

Why `-g`

- While the debugger works with the executable, it needs information about the C++ program.
  - What are the 10 C++ program statements centered around line number 21?
  - What are the addresses of the machine instructions for the statement:

    ```
    seriesValue += xpow/ComputeFactorial(k);
    ```
  - What is the type of the variable `seriesValue`?
  - In which memory address is `seriesValue` stored?

The `-g` flag equips the executable with such information.

Starting the Debugger

- Start the debugger as:

  ```
  $ gdb buggyprog
  $ gdb buggyprog core (more about this later)
  $ gdb --args buggyprog  <args…> (if the
    program takes arguments from the command
    line)
  ```

- Run the debugger as:

  ```
  $ run
  ```

  And the program runs to completion (or abortion) with bugs and everything. Not very useful.

The debugging cycle

- The cycle of activities during debugging:

  - Setting one or more breakpoints. Setting breakpoints requires judgement.
  - Running the program to stop at one of the breakpoints
  - Examining the data around these breakpoints
  - Continuing the execution to the next breakpoint

Setting Breakpoints

- Setting breakpoints

  - at line 43 of the current file

    (gdb) break 43
  - Other variations

    (gdb) break <function>

    (gdb) break <filename:linenum>

    (gdb) break <filename:function>

    (gdb) info breakpoints

Reaching a breakpoint

- Reaching breakpoints
  - To reach the first breakpoint

    (gdb) run
  - To reach subsequent breakpoints

    (gdb) continue

Stepwise execution

- Stepping over the program line by line
  - (gdb) next execute the current instruction and go to the next line.
  - (gdb) step step into a function
  - (gdb) finish Continue to the end of the current function

Locating the current point of execution

- Locating the current point
  - `where` show the call stack
  - `bt` show the call stack
- Listing statements around the current point
  - `list`

Examining values

- Printing
  - `print <exp>` any C expression
  - `print <function::variable_name>`
  - `print <file_name::variable_name>`
- Display
  - `display <exp>` display at every step

Setting watchpoints

- break when the value of an expression changes
  - `watch <exp>`
  - `watch <exp> if <cond>`