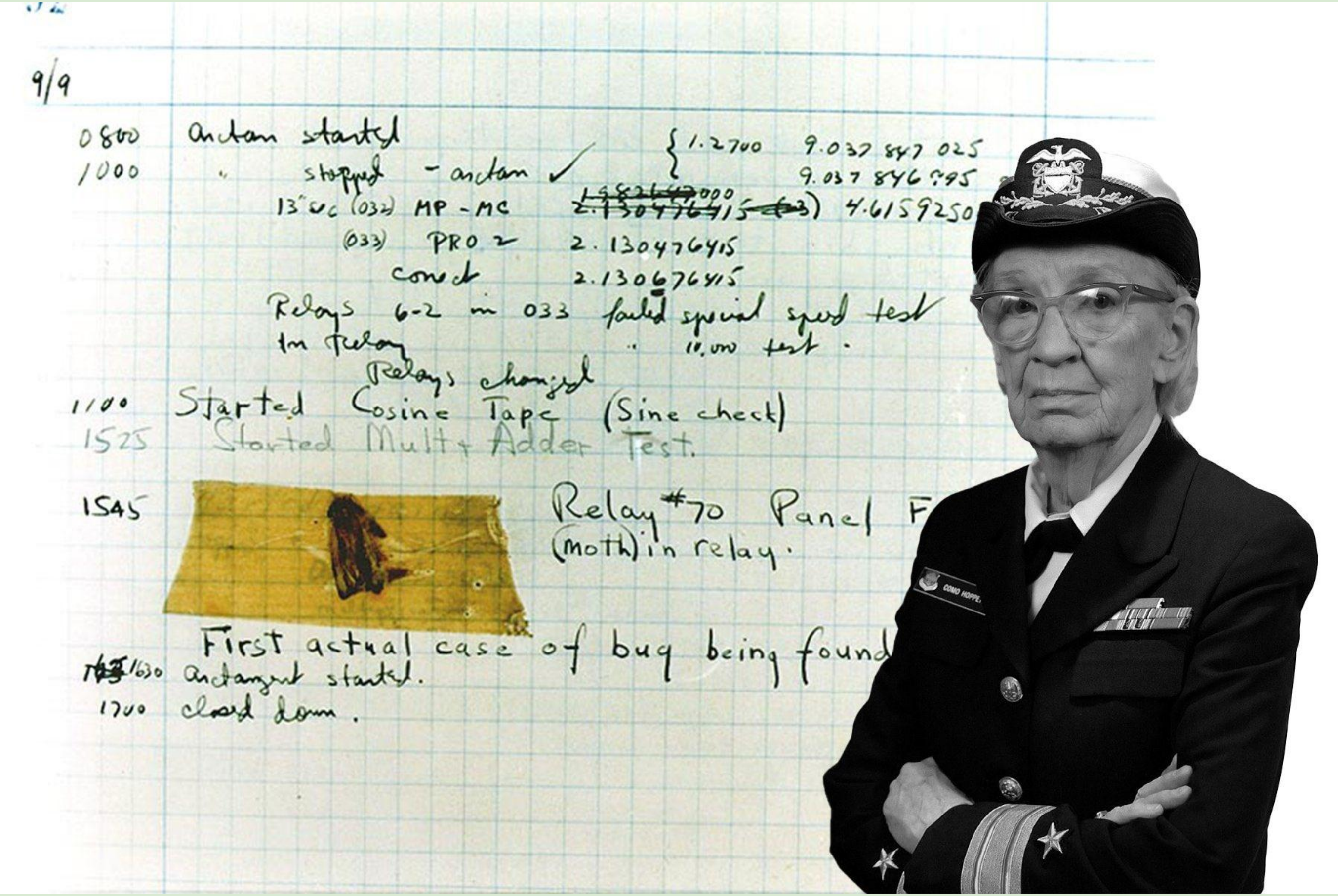
Advanced Programming - HPC

Tool : Debugger

Monojyotsna Koch

Date : 29.10.25

What is Debugging?


9/9
0800 Anctan started
1000 " stopped - anctan ✓
(033) PRO 2 2.130476415
correct 2.130676415
Relays 6-2 in 033 failed special speed test
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.
1545 Relay #70 Panel F (moth) in relay.
First actual case of bug being found.
1630 anctangent started.
1700 closed down.

What is Debugging?

- **Definition**:
  Debugging is the process of identifying, analyzing, and fixing bugs or errors in a program.
- **Importance**:
  - Ensures software correctness, reliability, and efficiency.
  - Saves time and resources in the long run.
- **Types of Bugs**:
  - Syntax errors, logic errors, runtime errors, concurrency issues.

What’s wrong here?

```
int main()
{
        int a = 10, b = 0;
        int c = a / b;
        return 0;
}
```

GDB

The GNU Project Debugger

GNU Debugger (GDB)

What is GDB?

- A powerful, open-source debugging tool for programs written in C, C++, Fortran, etc.
- Provides the ability to examine the internal workings of a program during execution.

Key Features:

- Breakpoints: Stop program execution at specific points.
- Variable Inspection: Examine or modify variable values.
- Call Stack Analysis: Understand the sequence of function calls.
- Multithreading and parallel debugging support.

Why GDB is Crucial?

- Seamlessly integrates into development workflows for faster bug resolution.

GDB Commands

Basic Commands:

- gdb program: Start GDB with a program.
- break: Set a breakpoint.
- run: Run the program.
- next, step: Navigate through code.
- list : print source code
- print: Display variable values.
- continue: Resume execution.
- quit: Exit GDB.

Intermediate Commands:

- backtrace: Show function call stack.
- info threads: List threads.
- thread: Switch between threads

Advanced Features:

- Watchpoints, conditional breakpoints, scripting capabilities.

Using the GNU Debugger.

1. **Compiling.**

- Add a –g option to enable built-in debugging support (which gdb needs)
- –g which tells the compiler to generate symbolic information required by any debugger.

For example:

```
gcc –g –o code code.c
```

Using the GNU Debugger.

2. Starting GDB

Just try “gdb” or “gdb program.x” to access the GDB prompt

- **(gdb)**

To execute the program

- **(gdb) run**

Using the GNU Debugger.

3. Setting breakpoints

Breakpoints can be used to stop the program run in the middle, at a designated point.

- Useful for investigating issues or examining specific section of code
- Setting Breakpoints - File and Line:
  - Syntax: **(gdb) break file.c:6**
- Set a breakpoint at line 6 in the file "file.c."
- Setting Breakpoints - Function:
  - Syntax: **(gdb) break my_func**
- Set a breakpoint at the beginning of the function "my_func."

After setting a breakpoint, use the *run* command to execute the program until it reaches the specified breakpoint.

- Stepping to the Next line

  $(gdb) next

- Stepping into functions

  $(gdb) step

- Listing Source code

  $(gdb) list

- Inspect variable values during the paused state.

  $(gdb) print <variable>

- Continue execution after inspection.

  $(gdb) continue

- Examining Stack Status:

  $(gdb) backtrace

Example : Basic program

Code Overview

- **A program to hello world.**
- Initial gdb command demo.

Debugging Focus

- Setting breakpoints (e.g., `main`).
- Single stepping through code.
- Inspecting variable values (`print`).

Demonstration

1. Compile with debug information: g++ -g hello.cpp -o hello
2. Debug steps:
    - Start GDB: `gdb ./hello`
    - Break at `main`: `break main`
    - Step through: `next`
    - Inspect variables: `print i`

Hello : Code Snippet

```
#include<iostream>
using namespace std;

int main()
{
        int j=3;
        int k=7;

        j+=k;
        k=j*2;

        std::cout<<"Hello everyone!!"<< std::endl;

}
~
```

Example : Segmentation Fault

Code Overview

- **A program that results in a segmentation fault.**
- Logical error for demonstration.

Debugging Focus

- Setting breakpoints in functions (e.g., `crash`).
- Inspecting stack .

Demonstration

1. Compile with debug information: gcc -g -o crash crash.c
2. Debug steps:
    - Start GDB: `gdb ./crash`
    - Break at `main`: `break main`
    - Step up and down through stack: `up, down`
    - Inspect full stack: `backtrace`

Crash : Code Snippet

```c
void crash(int *i) {
    *i = 1;  // Will cause segmentation fault if i == NULL
}

void f(int *i) {
    int *j = i;
    j = sophisticated(j);
    j = complicated(j);
    crash(j);  // This will crash due to j == NULL
}

int main() {
    int i;
    f(&i);
    return 0;
}
```

Example : Factorial of a number

Code Overview

- **A sequential program to calculate the factorial of a number.**
- Logical bugs for demonstration (e.g., incorrect loop condition).

Debugging Focus

- Setting breakpoints in functions (e.g., `factorial`).
- Stepping through loop iterations.
- Inspecting variable values (`n` and `result`).

Demonstration

1. Compile with debug information: gcc -g factorial.c -o factorial
2. Debug steps:
    - Start GDB: `gdb ./factorial`
    - Break at `factorial`: `break factorial`
    - Step through: `step`, `next`
    - Inspect variables: `print result`

Factorial : Code Snippet

```c
#include <stdio.h>

// Function to calculate factorial
int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;  // Multiplying to calculate factorial
    }
    return result;
}

int main() {
    int number;

    printf("Enter a positive integer: ");
    scanf("%d", &number);

    if (number < 0) {
        printf("Error: Factorial of a negative number doesn't exist.\n");
        return 1;
    }

    int fact = factorial(number);
    printf("Factorial of %d is %d\n", number, fact);

    return 0;
}
```

OpenMP Debugging

Commands helpful for OpenMP debugging:

- Start up GDB to debug a program

  $ gdb ./application.exe

Thread-specific Commands in GDB

- ***$(gdb)* info thread**- Prints out information about all current threads.
  - Helpful for understanding the status of threads during program execution, especially in parallel sections.

- ***$(gdb) thread*** -Prints the current thread

- ***$(gdb) thread <thread_no>*** - switches to specific thread.
  - Allows the user to switch between threads and focus debugging efforts on specific threads of interest.

Example : OpenMP Application Debugging

Code Overview

- Matrix multiplication program parallelized using OpenMP.
- Demonstrates how to optimize computational workloads using threads.

Debugging Focus

- Debugging nested loops in parallel regions.
- Inspecting thread-specific computations.
- Identifying race conditions or synchronization issues.

Demonstration

1. **Compile the Program**: gcc -g -fopenmp matrix_multiply.c -o matrix_multiply

Matrix Multiplication : Code Snippet

```
#pragma omp parallel for collapse(2)
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        int sum = 0;
        for (int k = 0; k < N; k++) {
            sum += A[i][k] * B[k][j];
        }
        C[i][j] = sum;
    }
}
```

Start GDB: `gdb ./matrix_multiply`

Break at the parallel region: `break 20` (or specific line number).

Step through code:

- `step` (to enter functions).
- `next` (to step over code).

Check thread details: `info threads`
Switch threads: `thread [id]`

Inspect variables:

- print A[i][k]
- print B[k][j]
- print C[i][j]

THANK YOU

QUIZ

What does the bt command do?

What command sets a breakpoint at main?

What command steps into a function?

Which command continues execution until the next breakpoint?

What happens if you run a binary without -g in GDB?