Tổng quan

Đây là tài liệu hướng dẫn sử dụng GDB để thay thế cho các cách debug giao diện trên IDE.

Lưu ý:

Tài liệu này không phải là tài liệu đầy đủ và hoàn chỉnh về GDB, mà chỉ tập trung vào các lệnh và quy trình cần thiết để debug trong môi trường phát triển nhúng với vi điều khiển và các unit test.

Tài liệu này sẽ trình bày với giả định đã có cài đặt GDB và có hiểu biết về mã máy.

Quy trình debug với GDB

1. **Biên dịch mã nguồn với tùy chọn debug**: Đảm bảo rằng mã nguồn của bạn được biên dịch với các tùy chọn debug (thường là `-g` trong GCC) để GDB có thể truy cập thông tin về biến, hàm, và dòng mã.

   Ví dụ:

   ```
   gcc -I<[/path/to/include]> -s -g [my_program.c] -o [my_program]
   # Trong đó -g là tùy chọn để biên dịch với thông tin debug
   # -I<> là đường dẫn đến thư mục chứa file header nếu cần include
   # -s là tùy chọn để tối ưu kích thước file thực thi
   # -o my_program chỉ định tên file thực thi được tạo ra
   ```

2. **Khởi động GDB**: Mở terminal và chạy lệnh `gdb <tên_tập_tin_thực_thi>` để khởi động GDB với tập tin thực thi của bạn.

3. **Setup giao diện debug**: Sử dụng lệnh `lay next` để chuyển sang giao diện debug nếu cần.

4. **Thiết lập điểm dừng đầu**: Sử dụng lệnh `break <tên_hàm>` để đặt điểm dừng tại hàm bạn muốn bắt đầu debug. Ở đây, khi bắt đầu chương trình thường sẽ chạy ở hàm `main()`, do đó thực hiện gọi lệnh `break main`.

5. **Chạy chương trình**: Sử dụng lệnh `run` để bắt đầu chạy chương trình. Chương trình sẽ dừng lại tại điểm dừng đã thiết lập.

6. **Điều khiển quá trình thực thi**:

   - Sử dụng lệnh `next` để thực thi dòng mã hiện tại và dừng lại ở dòng tiếp theo. (`next` ở đây cũng tương đương với `step over` trong các IDE)
   - Sử dụng lệnh `nexti` để thực thi dòng mã máy hiện tại và dừng lại ở lệnh máy tiếp theo.
   - Sử dụng lệnh `step` để bước vào bên trong hàm được gọi.
   - Sử dụng lệnh `continue` để tiếp tục chạy chương trình cho đến khi gặp điểm dừng tiếp theo hoặc kết thúc.
   - Sử dụng lệnh `ref` để làm mới giao diện debug, tránh rối mắt khi có nhiều thông tin.

Xử lý thông tin khi có lỗi

Khi chương trình gặp lỗi hoặc dừng lại đột ngột, GDB sẽ cung cấp thông tin về vị trí lỗi với dạng mẫu như sau:

```
Program received signal SIGSEGV, Segmentation fault.
0x0800045a in some_function () at my_program.c:25
```

Trong đó:

- `SIGSEGV` là tín hiệu báo lỗi phân đoạn bộ nhớ (segmentation fault).
- `0x0800045a` là địa chỉ bộ nhớ nơi lỗi xảy ra.
- `some_function` là tên hàm nơi lỗi xảy ra.
- `my_program.c:25` chỉ ra rằng lỗi xảy ra tại dòng 25 trong file `my_program.c` hoặc file tương ứng xảy ra lỗi.

Bạn có thể sử dụng các lệnh sau để kiểm tra trạng thái chương trình khi gặp lỗi:

- `x/i $pc`: Hiển thị lệnh máy tại con trỏ chương trình (program counter).
- `info registers`: Hiển thị trạng thái các thanh ghi hiện tại.
- `backtrace`: Hiển thị ngăn xếp cuộc gọi (call stack) để xác định chuỗi hàm đã được gọi trước khi lỗi xảy ra.
- `list`: Hiển thị mã nguồn xung quanh vị trí hiện tại của con trỏ chương trình.
- `print <tên_biến>`: In giá trị của biến cụ thể để kiểm tra trạng thái dữ liệu.

Các lệnh hữu ích khác

- `info breakpoints`: Hiển thị danh sách tất cả các điểm dừng đã thiết lập.
- `clear <số_điểm_dừng>`: Xóa điểm dừng cụ thể.
- `disable <số_điểm_dừng>`: Vô hiệu hóa điểm dừng cụ thể mà không xóa nó.
- `watch <tên_biến>`: Thiết lập điểm dừng khi giá trị của biến thay đổi.
- `quit`: Thoát khỏi GDB khi hoàn thành việc debug.
- `<enter>`: Nhấn phím Enter để lặp lại lệnh trước đó.
- `continue`: Tiếp tục thực thi chương trình cho đến khi gặp điểm dừng tiếp theo hoặc kết thúc.

Tài liệu bổ sung

1. Learn GDB in 60s
2. Low Level Debugging with GDB