GDBデバッガー入門

使用レポジトリ:
git@github.com:masaruo/gdb-for-dummies.git

前提：Macは対象外

- GDBはmacでは使えないが、同じようなデバッガーとしてLLDBが入っている
- コマンドは似てますが、違いもあるので、本日のGDBのコマンドが直接動くわけではない
- gdb to lldbにGDBからLLDBへのコマンド一覧がある
- dockerなどでubuntu仮想環境つかえるなら問題なし

本日のアジェンダ

- みんな大好き**Printf**をGDBで出来るようになろう
- Printfを**越えて**いこう
- 課題別**お役立ち**GDB機能
- GDB以外の、**あなたの味方**紹介

たった5つのコマンドで、あなたも Printf()@GDB をマスター

実践GDBでPrintf(): 覚えるのはたった5つのコマンドだけ！

- -g
  - cc -g <file_to_compile>：-g flagをつけてコンパイル
- gdb
  - gdb <executable_file_name>：GDBを起動
- (b)reak
  - break <func_name || line_num>：プログラムを止めるポイントを指定
- (r)un
  - run <if_any_args>：デバッグを開始
- (p)rint
  - print <var_name>：値を確認

使用ファイル
：gdb_for_dummies/basic/basic.c

なんか、ものすごく、情報が
見辛くないか？

デバッグ情報を見やすくしよう

- TUIモード
  - tui enable
  - tui disable
- Ctrl + l（エル）
  - 画面が乱れた時に表示を直す

使用ファイル：gdb_for_dummies/basic/basic.c

Printf()の限界を 越えていこう

printf()を可動にしよう

- (n)ext
  - 次の行へ
- (s)tep
  - 次の行へ（関数呼び出しがあれば、入る）
- (c)ontinue
  - 次のブレークポイントまで進む

使用ファイル：gdb_for_dummies/basic/basic.c

ループなしで配列データをさくっと調べてみよう

- print arr
- print &arr
- print *arr
- **print *arr@<length>**
  - print *arr@4

使用ファイル：gdb_for_dummies/basic/basic.c

ここはどこ？バックトレースでどこにいるか調べよう

- スタックフレームとは？
- backtrace
  - 自分はいまどこ？
  - 自分はなぜここに連れてこられたの？
  - どんなデータを渡されてきたの？
- frame <frame_number>
  - 調べるフレームを選択

使用ファイル：gdb_for_dummies/backtrace/bt.c

無限ループにハマる

- 私は、どこでハマっているの？
  - `ctrl + c`で割り込み
  - `backtrace`でハマっている場所

使用ファイル：gdb_for_dummies/backtrace/bt.c

あの課題にはこの機能がおすすめ！

void*型の使いにくさに 😭libft: 連結リスト

- 連結リストのデータ型　＝　なんでも格納できる万能`void*`
- ただし型の情報がないから、教えてあげなければいけない
- いちいち型情報教えるのはめんどくさいから、`define`しよう
- `source`コマンドで自分のコマンドを読み込もう

使用ファイル：gdb_for_dummies/voidPointer/voidPointer.c

子に翻弄されるminishell: Child Process

- デフォルトは親プロセスに留まる
- 子プロセスに行きたい
  - set follow-fork-mode child
  - break <func_in_child_proc>

使用ファイル：gdb_for_dummies/fork/fork.c

言うことを聞かない哲学者たち：thread

- info thread
- thread apply all <command>
- ただ、レースコンディションなどのデバッグはGDBより
  - -fsanitize=thread
  - `valgrind --tool=helgrind`
    - false positive / false negativeあり

使用ファイル：gdb_for_dummies/thread/thread.c
出所：Undo.io

CPP課題のクラスターPC問題: std::stringが・・・

- -fno-limit-debug-info （クラスターのPCでは）
  - clangのフラグ
  - 42のPCにはライブラリ関数にデバッグシンボルが入っていないようだ。

```
(gdb) s
RequestFactory::createRequest (fd=4096, config_factory=...) at src/http/utility/RequestFactory.cpp:14
(gdb) n
(gdb) p raw_request
$1 = Python Exception <class 'gdb.error'>: There is no member named _M_dataplus.

(gdb)
```

例外で右往左往のWebserv: exception

- catch throw && catch catch
- STL迷いの森にステップイン
  - skip <file>などの設定もあるが、設定ファイルが必要
  - tbreakで一時ブレークをつけてcontinueが楽かも

使用ファイル：gdb_for_dummies/cpp/exception.cpp

make

- GDBのターミナルの中でもmakeが出来る
  - GDBからexitしなくても、makeして、runして、デバッグ継続
    - 指定したブレークポイントなどは生きたまま
      - make
      - run

GDB以外のあなたの味方

fsanitize && valgrind

- -fsanitize=addressつけてコンパイル
- -g flagつけておけ
- Macでも使える（fsanitize=leakはだめ）
- Macのリークはleaks ＜実行ファイル＞

- valgrind ＜実行ファイル＞
- -g flagつけておけ
- お互いの仲は、悪いようだ

使用ファイル：gdb_for_dummies/overflow/overflow.c

参照

- Undo Univ GDB Course
- Undo WatchPoint
- Undo@youtube
- Debugging with GDB
- 実践デバッグ技法