第 10 回 Unix ゼミ
C プログラム(デバッグ編)

川島研 B4 高木 空

2024 年 07 月 04 日

デバッガ

- デバッグ (debug)
  - バグ(bug)を取り除く(de-)こと
- デバッグの手法
  - print デバッグ
  - コードを読む
  - デバッガを使う
    - 今回の主題

デバッグの手法

- print デバッグ
  - ソースコードに print を埋め込む
  - 利点
    - 気軽に実行できる
    - 欲しい出力を欲しい形式で得られる
  - 欠点
    - ソースコードを改変する必要がある
    - バグの箇所を検討してからしかできない
    - 得られる情報が少ない

デバッグの手法

- デバッガを使う
  - デバッガ - デバッグを補助するツール
  - 利点
    - プログラム全体を観察できる
    - プログラムの変更が(一般には)不要
    - スタックやメモリの監視もできる
  - 欠点
    - 使い方を知っている必要がある

C 言語のデバッガ

- C 言語プログラムのデバッガ
- GDB
  - Gnu Project のデバッガ
  - gcc を使うならコレ
  - Linux に標準搭載されている
- LLDB
  - LLVM のデバッガ
  - clang を使うならコレ

デバッガの起動

```
$ gdb [options] [<program>]
```

で GDB を起動

- `options`：起動時のオプションを指定
  - `--help`：簡単な使い方を表示
  - `--tui`：TUI モード(後述)で起動
- `program`：デバッグ対象の実行可能ファイルを指定

デバッガの終了

- GDB が起動すると先頭に(gdb)と表示される

```
(gdb) quit [<expr>]
(gdb) exit [<expr>]
```

で GDB を終了(ctrl-d でも可)

引数:

- expr : GDB の終了コードを指定

デバッガ起動中のシェルコマンド

```
(gdb) shell <command>
(gdb) ! <command>
```

で GDB 起動中にシェルコマンドを実行

引数:

- `command` : 実行するシェルコマンド

補足:

- パイプ等も使える

コマンド概要

- GDB はコマンドで操作
  - `quit` や `shell` もコマンド

```
(gdb) <command> [<args>...]
```

の形で入力

- コマンドが区別できれば省略できる
  - 例：`quit` → `q`
- `TAB` キーによる補完が可能
  - 候補が唯一の場合自動入力
  - 複数の場合 2 回押すと候補を表示

コマンド補助

```
(gdb) help [<class>|<command>]
```

コマンドの一覧や使い方を表示

引数:

- `class` : コマンド群を指定するクラス
- `command` : ヘルプを見たいコマンドを指定

補足:

- 引数無しで `help` を実行すると `class` の一覧が表示される

プログラムの開始

```
(gdb) run [<args>...]
```

でプログラムを GDB の下で実行

- `args` : プログラムのコマンドライン引数として渡される

チェックポイントとリスタート

特定の場所でのプログラムの状態を保存して再開できる

```
(gdb) checkpoint
```

で現在の状態を保存

```
(gdb) info checkpoints
```

で保存したチェックポイントの一覧を表示

```
(gdb) restart <id>
```

で指定したチェックポイントから再開

プログラムの停止

- GDB を使うとプログラムを中断できる
- 停止する条件
  - ブレークポイント
  - ウォッチポイント
  - キャッチポイント
- 実行の再開
  - 継続実行
  - ステップ実行

ブレークポイント

- プログラム上の指定場所に到達したら中断

```
(gdb) break [<loc>] [if <cond>]
```

でブレークポイントを設置

引数:

- `loc`：位置指定。以下の形式で指定:
  - `[<filename>:]<linenum>`：行番号指定
  - `<offset>`：行オフセット指定
  - `[<filename>:]<function>`：関数名指定
- `cond`：条件式。満たすときだけ中断

ウォッチポイント

式の値が変更したら中断

```
(gdb) watch [-location] <expr>
```

でウォッチポイントを設置

引数:

- `-location`：`expr` の参照するメモリを監視
- `expr`：監視対象の式

ブレークポイントの削除

```
(gdb) clear [<locspec>]
```

<locspec>にあるブレークポイントを削除

```
(gdb) delete [breakpoints] [<list>...]
```

<list>で指定したブレークポイント、ウォッチポイントを削除

```
(gdb) info breakpoints
```

設置されたブレークポイント、ウォッチポイントを表示

継続実行

次の停止場所まで実行する

```
(gdb) continue [<count>]
(gdb) fg [<count>]
```

で継続実行

引数:

- `count` : 停止箇所を無視する回数

ステップ実行

次の停止箇所を指定しつつ再開

```
(gdb) step [<count>]
(gdb) nexti [<count>]
```

で次の行まで実行。

補足:

- `step` は関数呼び出しの場合中に入る
- `next` は関数呼び出しの場合中に入らない

引数:

- `count` : 無視する行数

```
(gdb) until <locspec>
```

`locspec` で指定した位置まで実行

バックトレース

関数呼び出しのトレース

```
(gdb) backtrace
(gdb) where
(gdb) info stack
```

でバックトレースを表示

フレームの選択

```
(gdb) frame [<spec>]
```

でフレームを選択

引数:

- spec: フレームを指定。以下の形式が可能
  - <num>: フレーム番号を指定
  - <function-name>: 関数名を指定

```
up <n>
down <n>
```

で一つ上または下のフレームを指定

ステップ実行

```
(gdb) finish
```

で選択中のフレームが返るまで実行

ソースコード情報の表示

```
(gdb) list [<line>|<function>|+|-]
```

でソースコードを表示

引数:

- `line` : 行番号を指定してそこを中心に表示
- `function` : 関数名を指定して開始地点を中心に表示
- `+`, `-` : 前に表示した部分の後/前を表示

```
(gdb) list <start>, <end>
```

で指定部分を表示

プリント

```
(gdb) print [[<options>...] --] [/<fmt>] <expr>
```

でフォーマットを指定して expr の値を表示

引数:

- options : オプション
- fmt :
- expt : 表示する値

1 デバッガ 0
1.1 デバッガとは 0
1.1.1 概要 0
1.2 デバッグの手法 1
1.2.1 print デバッグ 1
1.2.2 デバッガ 2
1.3 デバッガの具体例 3
1.3.1 GDB と LLDB 3
1.4 デバッガの起動、終了 4
1.4.1 起動 4
1.4.2 終了 5
1.4.3 シェルコマンド 6
1.5 コマンド 7
1.5.1 コマンド概要 7
1.5.2 ヘルプ 8
1.6 プログラムの開始 9
1.6.1 スタート 9
1.6.2 チェックポイントとリスタート 10
1.7 プログラムの停止 11
1.7.1 プログラム中断の概要 11
1.7.2 ブレークポイント 12
1.7.3 ウォッチポイント 13
1.7.4 ブレークポイントの削除 14
1.8 プログラムの再開 15
1.8.1 継続実行 15
1.8.2 ステップ実行 16
1.9 スタックの調査 17
1.9.1 バックトレース 17
1.9.2 フレームの選択 18
1.9.3 フレーム関連のステップ実行 19
1.10 ソースコードの調査 20
1.10.1 リスト 20
1.11 データの調査 21
1.11.1 プリント 21
1.11.2 ディスプレイ 22
1.11.3 人工配列 22
1.11.4 レジスタ 22
1.12 (トレースポイント) 22
1.13 (TUI) 22