目次

1 デバッガ

プログラムのバグ(bug)を取り除く(de-)ことをデバッグといいます。デバッグを行う手法はいくつかあり、例えばプログラム中に標準出力を行う命令を追加してデバッグを行う print デバッガと呼ばれる方法があります。デバッガはデバッグを支援するツールで、プログラムの任意箇所での停止や、変数の値の表示や変更、スタックトレースやメモリ内容の監視など高度な機能によりデバッグを支援します。

C 言語で書かれたプログラムに対応するデバッガはいくつか存在しており、有名なものに GDB と LLDB が存在します。このドキュメントではこの二つのデバッガについて基本的な使用方法の解説を行います。

2 GDB

GDB は Gnu Project のデバッガです。

2.1 GDB の起動

GDB を起動するには以下のいづれかのコマンドを使用します。起動後はコマンドを受け付けます。

```
gdb [options] [executable-file [core-file or process-id]]
 gfb [options] --args <executable-file> [inferior-arguments ...]
```

`--args` を指定する場合、実行可能ファイルの後の引数(inferior-arguments) が実行時に渡されます。例えば `gdb --args gcc -O2 -c foo.c` は `gcc -O2 -c foo.c` の実行にデバッガをアタッチします。

options に指定できるオプションは `gdb -h` で確認できます。

2.1.1 ファイルオプション

GDB が起動すると、options 以外の引数は実行ファイルとコアファイル(またはプロセス ID)を指定するものとして読まれます。つまり`-se`、`-c` としてです。

`-symbols <file>, -s <file>`

file からシンボルテーブルを読み取ります。

`-exec <file>, -e <file>`

file を実行ファイルとして読み込みます。

`-se <file>`

file からシンボルテーブルを読み取り、実行ファイルとして使用します。

`-core <file>, -c <file>`

file をコアダンプとして検査します。

`-pid <number>, -p <number>`

プロセス ID が number のプロセスにアタッチします。

`-command <file>, -x <file>`

file からコマンドを実行します。

`-eval-command <command>, -ex <command>:`

単一の GDB コマンドを実行します。複数回指定可能です。

`-init-command <file>, -ix <file>`

下位ファイルをロードする前かつ gdbinit ロード後に file からコマンドを実行します。

`-init-eval-command <command>, -iex <command>`

下位ファイルをロードする前かる gdbinit ロード後に GDB コマンド `command` を実行します。

-early-init-command <file>, -eix <file>

出力生成前にファイルからコマンドを実行します。

-early-init-eval-command <command>, -eiex <command>

出力生成前に GDB コマンド command を実行します。

-directory <directory>, -d <directory>

directory をソースファイルとスクリプトファイルを検索するパスに追加します。

-readnow, -r

各シンボルファイルのシンボルテーブル全体を起動時に読み取ります。デフォルトではこの機能はオフになっています。

--readnever

各シンボルファイルのシンボルテーブルを読み取らないようにします。このオプションをつけるとシンボリックデバッグが実行できなくなります。

2.1.2 モードオプション

GDB はさまざまなモードで実行できます。

-nx, -n

初期化ファイルにあるコマンドを実行しません。

-nh

ホームディレクトリ初期化ファイルにあるコマンドを実行しません。システム全体及びカレントディレクトリの初期化ファイルは実行されます。

-quiet, -silent, -q

起動時のメッセージを表示しません。これらメッセージはバッチモードでも表示されません。コマンドによりこのオプションを有向化することもできます。

-batch

バッチモードで実行します。-x で指定したコマンドファイルのコマンドがすべて実行された後、終了コード 0 を返して終了します(-n が指定されていない場合は初期化ファイルのコマンドも実行されます)。ファイル内のコマンド実行中にエラーが発生した場合は 0 以外のステータスコードを返して終了します。

-batch-silent

バッチモードで実行し、かつ全く標準出力への出力を行いません。

-return-child-result

GDB の終了ステータスをデバッグ中のプロセスの終了コードにします。ただし (1)GDB が異常終了した場合、(2)ユーザが明示的に終了ステータスを指定した場合、(3)子プロセスが実行されないか終了しない場合(終了ステータスは-1 になる)の三つの場合を除きます。

-nowindows, -nw

GDB に GUI インターフェースがある場合、CUI のみを使用するように指定します。

windows, -w

GDB に GUI インターフェースがある場合、GUI インターフェースを使用します。

-cd <directory>

作業ディレクトリを directory に移動して実行します。

-data-directory <directory>, -D <directory>

directory をデータディレクトリ(GDB が補助ファイルを検索する場所)として実行します。

-fullname, -f

スタックフレーム表示時およびプロセス停止時に完全なファイル名と行番号を出力します。

-annotate <level>

GDB 内の注釈レベルを設定します。これはプロンプト、式の値、ソース行、その他の出力とともに GDB が出力する情報の量を制御します。レベル 0 が通常、1 が Gnu Emacs で使用され非推奨、レベル 3 は最大の注釈です。

--args

実行ファイル以降の引数をすべて下位のコマンドライン引数として渡します。

--baud <bps>, -b <bps>

GDB がリモートデバッグに使用するシリアルインターフェイスの回線速度を設定します。

-l <timeout>

GDB がリモートデバッグに使用する通信のタイムアウト(単位:秒)を設定します。

-tty <device>, -t <device>

プログラムの標準入力と出力に device を使用して実行します。

-tui

TUI(Text User Interfaec) モードをアクティブにします。TUI はターミナル上の複数のテキストウィンドウを管理し、ソース、アセンブリ、レジスタ、およびコマンド出力を表示します。

-interpreter <interp>

制御プログラムまたはデバイスとのインターフェイスにインタープリター interp を使用します。このオプションは GDB をバックエンドとして GDB と津伸によって設定されることを目的としています。

-write

実行ファイルとコアファイルを読み取り書き取りの両方で開きます。

-statistics

GDB は各コマンドを完了してプロンプトに戻った後、時間とメモリ使用量に関する統計情報を表示します。

-configuration

GDB はビルド時の構成パラメータの詳細を出力し、終了します。

2.1.3 起動に GDB が行う動作

セッション起動時に GDB が行う処理を以下に示します。

1. 基本的な内部状態を初期化します。
2. ホームディレクトリにある初期初期化ファイルが存在する場合、コマンドを読み取ります。
3. `-eiex` と `-eix` で指定されたコマンドとコマンドファイルを指定された順番に実行します。
4. コマンドラインで指定されたコマンドインタープリターを設定します。
5. システム全体の初期化ファイルと初期化ディレクトリからファイルを読み取ります。
6. ホームディレクトリ内の初期化ファイルを読み取り、ファイル内のすべてのコマンドを実行します。
7. `-iex` および `-ix` で指定されたコマンドとコマンドファイルを指定された順番に実行します。通常 `-ex` および `-x` を代わりに使用します。この方法では GDB 初期化ファイルが実行される前および inferior がロードされる前に設定を適用できます。
8. コマンドラインオプションとオペランドを処理します。
9. 現在の作業ディレクトリにある初期化ファイルを読み込んで実行します。
10. デバッグするまたはアタッチするプログラムまたはコアファイルが指定されている場合、GDB はプログラムまたはそのロードされた共有ライブラリ用に提供された自動ロードスクリプトをロードします。
11. `-ex` および `-x` で指定されたコマンド及びコマンドファイルを読み込んで実行します。
12. *history file* に記録されたコマンド履歴を読み取ります。

2.1.4 初期化ファイル

GDB 起動時に GDB はいくつかの初期化ファイルからコマンドを実行します。これらの初期化ファイルはコマンドファイルと同じ構文を使用し、同様に処理されます。

起動時にロードされる初期化ファイルのリストをロードされる順番で表示するには `gdb --help` が使用できます。

初期初期化ファイルは初期化プロセスの非常に速い段階でロードされます。ここでは `set` または `source` コマンドのみを配置できます。

ほかの一般の初期化ファイルは任意のコマンドを実行できます。

2.1.4.1 ホームディレクトリの初期初期化ファイル

GDB は最初にこれを探します。GDB がホームディレクトリ内を検索する場所はいくつかあり、これらの場所は順番に検索され、最初に見つかったファイルのみをロードします。MacOS 以外では以下の場所が検索されます。

- `$XDG_CONFIG_HOME/gdb/gdbealyinit`
- `$HOME/.config/gdb/gdbealyinit`
- `$HOME/.gdbealyinit`

`-nx, -n` オプションでこれらの初期初期化ファイルを読むことを阻止できます。

2.1.4.2 システム全体の初期化ファイル

以下の二か所が検索され、これらは常にチェックされます。

system.gdbinit

単一のシステム全体初期化ファイルです。 `--with-system-gdbinit` オプションで設定できます。

system.gdbinit.d

ディレクトリです。

2.1.4.3 ホームディレクトリ初期化ファイル

システム全体初期化ファイルを読んだ後、これを探します。以下の場所を検索し、最初に見つかったファイルのみをロードします。 MacOS 以外では以下の場所が検索されます。

- `$XDG_CONFIG_HOME/gdb/gdbinit`
- `$HOME/.config/gdb/gdbinit`
- `$HOME/.gdbinit`

2.1.4.4 ローカルディレクトリ初期化ファイル

カレントディレクトリで`.gdbinit` ファイルを検索します。`-x, -ex` で指定したコマンドを除いて最後にロードされます。すでにホームディレクトリ初期化ファイルとして読み込まれている場合は再度ロードされることはありません。

2.2 GDB の終了

GDB を終了するには `quit [expression]`, `exit [expression]` または `q` または `ctrl+d` で終了できます。`expression` に指定した値は終了コードとして帰ります。`ctrl+c` は実行中の GDB コマンドアクションを終了します。

2.3 シェルコマンド

GDB 起動中にシェルコマンドを使用することができます。

```
shell <command-string>
!<command-string>
```

`pipe` 命令を使用して gdb の出力を他のプログラムに繋ぐことができます。

```
pipe [command] | <shell_command>
| [command] | <shell_command>
  pipe -d <delim> <command> <delim> <shell_command>
  | -d <delim> <command> <delim> <shell_command>
```

`command` が `|` を含むときには -d で別の記号(列)を指定します。

2.4 ロギング出力

GDB の出力をファイルに行うことができます。GDB にはロギングを制御するコマンドがいくつか用意されています。

set loggging enabled [on|off] ロギングのオンオフ切り替え

set logging file <file> 現在のログファイルの名前を変更。デフォルト値は gdb.txt

set logging overwrite [on|off] 上書きか書き足しか(on で上書き)。デフォルト値は `off`

set logging redirect [on|off] on にすると GDB の出力がログファイルにのみ行われる。デフォルト値は `off`

set logging debugredirect [on|off] on にすると GDB デバッグの出力がログファイルにのみ行われる。デフォルト値は `off`

show logging ロギングの設定を表示する

3 GDB コマンド

GDB コマンドは曖昧性がなければコマンド名の最初の数文字のみで使用できます。また、`ret`(エンター)を入力すると特定の GDB コマンドを繰り返し実行できます。また、TAB キーによる補完機能が有効です。

3.1 コマンド構文

GDB コマンドは一行の長さ無制限の入力です。`command [args]` の形をしています。

`run` など一部コマンドを除いて空白行を入力すると直前のコマンドを繰り返します。`list` 及び `x` コマンドでは引数が変わります(???)。

3.2 コマンド設定

多くのコマンドは変数及び設定で動作が変わります。これらの設定は `set` コマンドで変更できます。

gdbinit ファイルに書き込むことで初期化時に設定できますし、対話中にコマンドを実行して設定することもできます。

`with` コマンドを使用して、コマンド呼び出しの期間中一時的に設定を変更することもできます。

```
with <setting> [value] [-- command]
 w <setting> [value] [-- command]
```

3.3 コマンド補完

GDB では TAB キーによる補完が有効です。候補が唯一の場合は自動で入力が保管され、複数ある場合は候補が表示されます。TAB を二回押して候補を表示する代わりに `esc ?`で表示することもできます。

以下のコマンドで補完候補の最大数を設定できます。デフォルト値は 200 です。

```
set max-completions <limit>
```

`limit` には整数値または `unlimited` が指定できます。

```
show max-completions
```

で現在の設定を確認できます。

3.4 ファイル名引数

ファイル名をコマンドの引数として渡す場合、ファイル名に空白、ダブルクォート、シングルクォートが含まれていない場合は単純な文字列として記述できます。これらが含まれている場合、いくつか方法があります。

- GDB に任せる
- エスケープを使う
- クオートで囲う

3.5 コマンドオプション

一部コマンドは先頭に-がついたオプションを受け付けます。コマンド名と同様に、明確な場合は省略形を使うことができます。また、補完も効きます。

一部コマンドの引数にハイフンを含む場合は--を使うことでそれ以降の引数をオプションとして解釈しなくなります。

3.6 ヘルプ

help コマンドを使用してコマンドのヘルプを閲覧できます。

```
help, h
```

引数なしの help コマンドはコマンドのクラスのリストを表示します。

```
help <class>
```

ヘルプクラスを指定するとそのクラスの個々のコマンドのリストを表示します。

```
help <command>
```

コマンドを指定するとそのコマンドの短い使用方法を表示します。

```
apropos [-v] <regexp>
```

コマンド、エイリアス及びそのドキュメントを検索し、引数で指定した正規表現を検索します。見つかったすべてを表示します。-v オプションをつけるとドキュメントの一致部分をハイライトして表示します。

```
complete <args>
```

コマンドの先頭部分の一致候補を表示します。

info, show. set コマンドを使用して、プログラムの状態や GDB の状態を設定および照会することができます。

```
info, i
```

プログラムの状態を表示します。help info でサブコマンドの一覧を閲覧できます。

```
set
```

式の結果を環境変数に割り当てます。

```
show
```

GDB の状態を表示します。set できるものは大体 show できます。

show にあって set できないものを以下に示します。

- version

バージョン情報を表示します。

- copying

著作権表示を行います。

- warranty

保証情報を表示します。

- configuration

GDB のビルド情報を表示します。

4 GDB でプログラムを実行する

GDB でプログラムを実行するにはコンパイル時にデバッグ情報を付与する必要があります。

任意の環境で、引数を指定して GDB を起動できます。ネイティブデバッグではプログラムの IO をリダイレクトしたり、実行中プロセスをデバッグしたり、子プロセスを強制終了したりできます。

4.1 デバッグのためのコンパイル

プログラムを効果的にデバッグするにはコンパイル時にデバッグ情報を生成する必要があります。この情報はオブジェクトファイルに保存され、各変数、関数のデータ型と実行可能コード内のソース行番号とアドレスの対応関係が記述されます。

デバッグ情報の生成は`-g` オプションで行うことができます。

GCC では`-g` オプションは`-O`(最適化オプション)と併用できます。

4.2 プログラムの開始

`run, r`

GDB でプログラムを実行するには `run` コマンドを使用します。このコマンドを使用するには GDB 起動時またはコマンドでプログラムを指定する必要があります。

引数

`run` コマンドの引数はそのままプログラムのコマンドライン引数として渡されます。

環境

プログラムは GDB から環境を継承します。`set` コマンドで環境を変更することもできます。

作業ディレクトリ

`set cwd` でプログラムの作業ディレクトリを設定できます。設定しない場合、GDB の作業ディレクトリを引き継ぎます。リモートデバッグの場合にはリモートサーバの作業ディレクトリを引き継ぎます。

標準入出力

通常、プログラムの標準入出力は GDB と同じになります。`tty` コマンドで別のデバイスを設定することもできます。

`run` コマンドで実行したプログラムは直ちに実行を開始します。

GDB はシンボルファイルの変更を検出し、再読み込みを行います。

`start`

`start` コマンドはメインプロシージャにブレークポイントを設置して `run` します。引数の扱いは `run` と同様です。

`starti`

start と同様ですが、ブレークポイントの位置は最初の命令です。

```
 set exec-wrapper <wrapper>
 show exec-wrapper
  unset exec-wrapper
```

を使用してデバッグ用プログラムの起動します。つまり Shell コマンド exec wrapper program を実行します。

```
 set startip-with-shell
 set startip-with-shell on
  set startip-with-shell off
 show startip-with-shell
```

プログラムをシェルで実行します。

4.3 プログラムの引数

4.4 プログラムの環境

4.5 プログラムの作業ディレクトリ

4.6 プログラムの入出力

4.7 すでに実行中のプロセスのデバッグ

4.8 子プロセスの終了

4.9 複数の下位接続とプログラムのデバッグ

4.10 複数スレッドのプログラムのデバッグ

4.11 フォークのデバッグ

4.12 チェックポイント、再起動