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。

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。

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。
```
set auto-connect-native-target
set auto-connect-native-target on
set auto-connect-native-target off
show auto-connect-native-target
```
現在の下位プロセスがターゲットに接続されていないときにローカルマシンで実行します。
```
set disable-randomization
set disable-randomization on
```
プログラムの仮想アドレス空間のネイティブランダム化をオフにします。

プログラムの引数

プログラムへの引数は `run` コマンド実行時に指定します。指定しなかった場合は以前の `run` 実行時の引数または `set args` で指定した引数を使用します。
```
set args
show args
```

プログラムの環境

環境は、環境変数とその値のことを指します。
```
path <directory>
```
環境変数の先頭にディレクトリを追加します。GDB が使用する環境変数の値は変化しません。
```
show paths
```
実行可能ファイルの検索パスのリストを表示します。
```
show environment [varname]
```
環境変数 `varname` の値を表示します。指定しない場合はすべての環境変数とその値を表示します。
```
set environment varname [=value]
```
環境変数 `varname` の値を設定します。値はプログラムに対してのみ変更され、GDB が読む変数の値は変わりません。

プログラムの作業ディレクトリ

```
set cwd [directory]
```
下位の作業ディレクトリを `directory` に変更します。
```
show cwd
```
下位プロセスの作業ディレクトリを表示します。
```
cd
```
GDB の作業ディレクトリを変更します。
```
pwd
```
GDB の作業ディレクトリを表示します。

プログラムの入出力

デフォルトでは GDB が実行するプログラムは GDB と同じターミナルに入出力を行います。

```
  info terminal
```

プログラムが使用している端末モードについての情報を表示します。

`run` コマンドでシェルのリダイレクト機能を使えます。

```
  run > <output file>
```

`tty` コマンドで入出力の場所を指定できます。

```
  tty <file>

  set inferior-tty <tty>
  show inferior-tty
```

デバッグ対象のプログラムの tty を設定、表示します。

すでに実行中のプロセスのデバッグ

```
  attach <pid>
```

すでに実行中のプロセスのプロセス ID を指定してデバッガをアタッチします。

```
  set exec-file-mismatch 'ask|warn|off'
  show exec-file-mismatch
```

GDB がロードした実行ファイルとアタッチしたプログラムの実行ファイルが一致するか確認した際に不一致だった場合の挙動を設定します。 ask は警告を出しプロセスの実行ファイルをロードするか確認します。warn は警告を表示のみします。off は不一致確認を行いません。

GDB はプロセスにアタッチするとそのプロセスを停止します。続行するには `continue` をします。

```
  detach
```

プロセスからデタッチします。

子プロセスの終了

```
  kill
```

GDB で実行されている子プロセスを終了します。

複数の下位接続とプログラムのデバッグ

GDB では 1 セッションで複数のプログラムを実行、デバッグできます。一部システムでは同時に行えます。

GDB では各プログラムの状態を inferior と呼ばれるオブジェクトで管理します。

```
  info inferior
```

現在存在する inferior を表示します。

```
  inferior
```

現在の inferior の情報を表示します。

```
  info connection
```

現在開いているターゲット接続を表示します。

```
  inferior <infno>
```

現在の inferior を infno に変更します。

```
  add-inferior [-copies <n>] [-exec <executable>] [-no-connection]
```

実行ファイルを `executable` とする inferior を n 個追加します。

```
  clone-inferior [-copies <n>] [infno]
```

inferior を n 個コピーします。
```
remove-inferiors infno...
```
inferior を削除します。
```
detach inferior infno...
kill inferior infno...
```
inferior を指定して detach, kill します。
```
set print inferior-events [on|off]
show print inferior-events
```

inferior のプロセスが開始または終了したときに通知を受け取ります。
```
maint info program-spaces
```
GDB によって管理されているすべてのプログラムスペースのリストを出力します。

inferior 固有のブレークポイント

複数の inferior プロセスをデバッグする場合、すべての inferior プロセスにブレークポイントを設定するか、個別にするか選択できます。
```
break <locspace> inferior inferior-id
break <locspace> inferior inferior-id if ...
```

複数スレッドのプログラムのデバッグ

GDB はマルチスレッドデバッグを行うために以下の機能を提供します。

- 新しいスレッドの自動通知
- スレッドを切り替えるコマンド
- 既存のスレッドの情報を表示するコマンド
- スレッドのリストにコマンドを適用するコマンド
- スレッド固有のブレークポイント
- スレッド開始、終了時のメッセージ設定

GDB では複数スレッドを観察できます。今見えているスレッドをカレントスレッドといいます。デバッグコマンドはカレントスレッドに適応されます。

新しいスレッドを検出するとスレッド識別子を表示します。

```
info threads [-gid] [thread-id-list]
```
スレッドの情報を表示します。引数を指定しなければすべてのスレッドの情報が表示されます。`-gid` を指定するとグローバルスレッド番号も表示します。
```
thread <tid>
```
カレントスレッドを `tid` のスレッドに変更します。
```
thread apply [thread-id-list | all [-ascending]] [flag]... <command>
```
指定したスライド全体にコマンドを適用します。フラグに設定できるのは以下のとおりです。

`-c` コマンド内のエラーを表示してその後のコマンドは続行されます。
`-s` コマンド内のエラーを無視します。
`-q` スレッド情報を表示しません。

```
taas [option]... <command>
```
`thread apply all -s [option]... <command>`のショートカット。
```
tfaas [option]... <command>
```

`thread apply all -s -- frame apply all -s [option]... <command>`のショートカット。

```
thread name [name]
```

カレントスレッドに名前をつけます。名前を指定しない場合は名前を削除します。

```
thread find [regexp]
```

指定した正規表現と一致する名前またはシステムタグを持つスレッドを検索して表示します。

```
set print thread-event [on|off]
show print thread-event
```

スレッド開始、終了時のメッセージを有効または無効にします。

フォークのデバッグ

```
set follow-fork-mode <mode>
```

fork の呼び出しに対する応答を設定します。

- `parent` 元のプロセスは fork のあとデバッグされます。子プロセスは妨げられずに実行されます。デフォルト。
- `child` 新しいプロセスが fork のあとにデバッグされます。親プロセスは妨げられずに実行されます。

```
set detach-on-fork 'on|off'
```

- `on` 子プロセスは切り離され、独立して実行されます。(デフォルト)
- `off` 両方のプロセスが GDB の制御化に置かれます。片方のプロセスは中断されます。(別 inferior)

チェックポイント、再起動

デバッグ中の状態にチェックポイントをおいて戻ることができます。

```
checkpoint
```

現在の状態にチェックポイントを起きます。

```
info checkpoint
```

設置されたチェックポイントの一覧を表示します。

```
restart <checkpoint-id>
```

チェックポイントに戻ります。

```
delete checkpoint <checkpoint-id>
```

チェックポイントを削除します。