JPUG しくみ分科会 勉強会

PostgreSQL ソースコードの読み方

～ vi + ctags, gdb, strace ～

2003年12月02日(火)

NTTデータ先端技術(株)
井久保 寛明

もくじ

- vi + ctags によるソースの読み方
  - 準備
  - ソースコードの読み方
  - ソースを読むためだけに必要な vi のコマンド
  - vi に必要な設定
  - grep
- gdb の使い方
  - gdb の使い方
  - gdb の使い方2 (プロセスへのアタッチ)
  - gdb の使い方3 (コアファイルの調査)
  - その他の gdb の便利な機能
- strace の使い方
  - strace の使い方
  - strace の使い方2 (プロセスへのアタッチ)

Copyright(C)2003 NTT DATA Corporation
Copyright(C)2003 NTT DATA Intellilink

準備

■ ソースの展開

```
frisk(3)% cd ~/work/
frisk(4)% tar zxvf postgresql-7.3.4.tar.gz
```

■ コンパイルとインストール

- ◆ コンパイル時に作成されるヘッダなども読めた方が便利なので、一度コンパイルする

```
frisk(5)% ./configure --enable-syslog --enable-debug --prefix=/home/ikubo/work/
pg734dbg
frisk(6)% make
frisk(7)% su
frisk# make install
frisk# exit
```

- ◆ 環境変数やパスの設定を行ったら、DBを初期化
- ◆ 必要に応じて postmaster を起動

```
frisk(12)% initdb --no-locale --encoding=EUC_JP
frisk(13)% pg_ctl start
```


Copyright(C)2003 NTT DATA Corporation
Copyright(C)2003 NTT DATA Intellilink

vi + ctags によるソースコードの読み方

■ ctags によるタグファイルの作成

```
frisk(14)% cd ~/work/postgresql-7.3.4/src/backend
frisk(15)% ctags */*.[chyl] */*/*.[ch] ../include/*.h ../include/*/*.h
```

■ vi での読み方

- ◆ ファイルを開くときは、必ず、「src/backend」から開く

```
frisk(16)% vi main/main.c
```

- ◆ 移動したい関数や変数などにカーソルをあてて、Ctrl +] で移動する
- ◆ Ctrl + t で元の位置に戻る

■ 探したいものを見つけるときは、

- ◆ ひたすら grep
- ◆ または、次のようにいきなりタグファイルを開く方法もあり

```
frisk(17)% vi tags
```


Copyright(C)2003 NTT DATA Corporation
Copyright(C)2003 NTT DATA Intellilink

ソースを読むためだけに必要な vi のコマンド

■ コマンドモード(参照)と編集モードを意識する

- ◆ 今回は参照だけなので、おかしいなと思ったら、とにかく Esc キーを押す
- ◆ なるべく参照モード (view or vi -v) でファイルは開く

■ 参照でよく使うコマンド

- ◆ 終了: :q! ← 続けて入力する
- ◆ 1文字移動 左下上右: h j k l ※ カーソル(←↓↑→)でも移動可能
- ◆ 一画面下／上へ移動: Ctrl + f , Ctrl + b
- ◆ 検索: / を入力した後、探したい文字列を入力してEnter を入力
  - 繰り返す場合は、n 。逆向きなら N
- ◆ 目的の行番号にジャンプする 行番号を入力した後: G
- ◆ ファイルの先頭／最後にジャンプする: 1G , G
- ◆ 該当の関数にジャンプする: ジャンプしたい関数名にカーソルを合わせてCtrl +]
  - 元の位置に戻るには: Ctrl + t

Copyright(C)2003 NTT DATA Corporation
Copyright(C)2003 NTT DATA Intellilink

vi に必要な設定

■ 設定の方法

- ◆ $HOME 以下に、.vimrc または .exrc を作成する

■ tabstop

- ◆ PostgreSQL のソースコードは、4文字タブであることを前提に書いてあるので、タブの文字数を設定する必要がある
- ◆ set tabstop=4　(set ts=4 と省略して書くことも可能)

■ ビープ音を鳴らさないようにする

- ◆ set noerrorbells

■ バックアップファイルを作らないようにする (編集する場合)

- ◆ set nobackup

Copyright(C)2003 NTT DATA Corporation
Copyright(C)2003 NTT DATA Intellilink

grep

■ grep の使い方

- ◆ 定義されている場所を探す場合や、参照元を探す場合によく使う
- ◆ おおよその位置の検討がついている場合

  例えば、BufferSync() を呼び出しているものを知りたい

```
frisk(24)% grep BufferSync() */*.c */*/*.c
```

- ◆ どこにあるか分からない場合、include も含めて調べる

  例えば、FormData_pg_class の定義が知りたい

```
frisk(25)% grep FormData_pg_class */*.[ch] */*/*.[ch] ../include/*.h \
../include/*/*.h
```

```
access/transam/xlogutils.c:     _xlpgcarr = (Form_pg_class) malloc(sizeof(FormDa
ta_pg_class) * _xlcnt);
access/transam/xlogutils.c:     memset(_xlpgcarr, 0, sizeof(FormData_pg_class) *
 _xlcnt);
                                        :
                                        :
../include/catalog/pg_class.h:} FormData_pg_class;
../include/catalog/pg_class.h:   (offsetof(FormData_pg_class,relhassubclass) + s
izeof(bool))
../include/catalog/pg_class.h:typedef FormData_pg_class *Form_pg_class;
```


Copyright(C)2003 NTT DATA Corporation
Copyright(C)2003 NTT DATA Intellilink

vi + ctags の感想

- ■ 好きなところ
  - ◆ 軽い
  - ◆ キーの入力が少なくて済む
  - ◆ どこの環境でも、ほぼ確実に使える

- ■ 不便に感じるところ
  - ◆ 一部のヘッダファイルのタグが作れない
    - カタログのヘッダなど、マクロを使ってあるところ (include/catalog/*.h など)
  - ◆ 呼び出し元を探すときは、grep を使う

Copyright(C)2003 NTT DATA Corporation
Copyright(C)2003 NTT DATA Intellilink

gdb の使い方

■ PostgreSQLのコンパイル時

```
frisk(5)% ./configure --enable-debug
```

必ず、デバッグを有効にする

■ 使い方

1. gdbの実行
2. ブレークポイントの設定
3. プログラムの実行
4. あとは、ステップ実行したり、変数の中を確認したりする

bootstap を追いかけるときの例

```
frisk(36)% gdb /home/ikubo/work/pg734dbg/bin/postgres
(gdb) break main
(gdb) run -boot -x1 -F -D /home/ikubo/data template1 < /home/ikubo/data/postgres.bki

[bootstrap の break Pointの例]
        b heap_create ; リレーション(テーブル)作成のエントリポイント
        b InsertOneValue ; insertのエントリポイント
        b InsertOneTuple ; タプルを生成する部分
```


Copyright(C)2003 NTT DATA Corporation
Copyright(C)2003 NTT DATA Intellilink

gdb の主なコマンド

gdbを終了する	q	quit
ヘルプの表示	h	help
プログラムの実行	r	run [引数]
プログラムを止める		Ctrl + C
プログラムを続行する	c	continue
プログラムを止めたい場所を指定する(※1)	b	break 関数名
次の行を実行する (関数の場合、関数の中に入る)	s	step
次の行を実行する (関数の中に入らない)	n	next
現在行の前後を表示	l	list
スタックを見る	bt	backtrace
変数の表示	p	print 変数名
メモリのダンプ	x	x 変数名
文字列の表示	x/s	x/s 変数名

※1 動的にリンクされる関数の場合、最初から指定できない場合がある。そのときは別のブレークポイントを設定して、ライブラリがロードされるところまで実行した後に指定すると、指定が可能になる。

Copyright(C)2003 NTT DATA Corporation
Copyright(C)2003 NTT DATA Intellilink

gdb の使い方２(プロセスへのアタッチ)

■ 使い方2

◆ 別のターミナルからクライアント(psqlなど)で接続する

```
frisk(1)% psql template1
```

◆ どのバックエンドプロセスか確認して、そのプロセスにアタッチする

```
frisk(41)% ps -ax | grep postgres
18373 pts/0    S      0:00 postgres: stats buffer process
18375 pts/0    S      0:00 postgres: stats collector process
18396 pts/0    S      0:00 postgres: ikubo template1 [local] idle
18399 pts/0    S      0:00 grep postgres
```

```
frisk(45)% gdb /home/ikubo/tmp/pg734/bin/postgres 18396
```

■ あとは、ソースコードのエントリポイントを頼りに中を追う

◆ 第1回目の勉強会資料「DBMSアーキテクチャの概要とソースツリーの概要」などを参照

Copyright(C)2003 NTT DATA Corporation
Copyright(C)2003 NTT DATA Intellilink

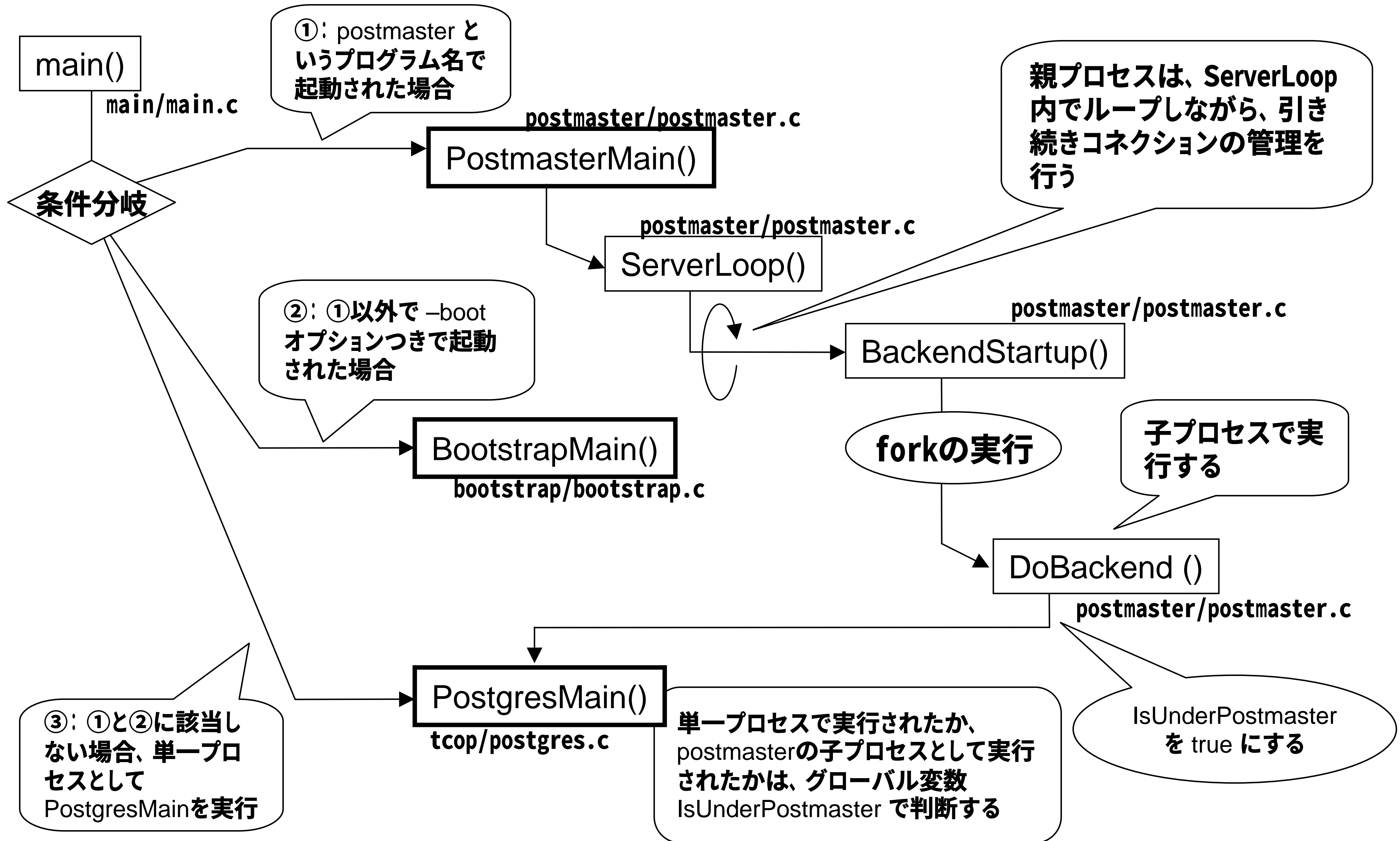
gdb の使い方２(プロセスへのアタッチの例)

```
(gdb) bt
#0  0x401c64e2 in __libc_recv () from /lib/i686/libc.so.6
#1  0x080dc164 in secure_read (port=0x8240388, ptr=0x81f22e0, len=8192)
    at be-secure.c:307
#2  0x080dedc9 in pq_recvbuf () at pqcomm.c:463
#3  0x080dee0d in pq_getbyte () at pqcomm.c:500
#4  0x08119549 in SocketBackend (inBuf=0x826d7a8) at postgres.c:247
#5  0x081195bf in ReadCommand (inBuf=0x826d7a8) at postgres.c:304
#6  0x0811aa8d in PostgresMain (argc=4, argv=0xbffecc10,
    username=0x82404b9 "ikubo") at postgres.c:1930
#7  0x08102b6c in DoBackend (port=0x8240388) at postmaster.c:2310
#8  0x081024be in BackendStartup (port=0x8240388) at postmaster.c:1932
#9  0x081016d1 in ServerLoop () at postmaster.c:1009
#10 0x08101292 in PostmasterMain (argc=1, argv=0x82287e0) at postmaster.c:788
#11 0x080df86b in main (argc=1, argv=0xbffed5a4) at main.c:210
#12 0x400f4657 in __libc_start_main (main=0x80df68c <main>, argc=1,
    ubp_av=0xbffed5a4, init=0x806a8a0 <_init>, fini=0x8171560 <_fini>,
    rtld_fini=0x4000dcd4 <_dl_fini>, stack_end=0xbffed59c)
    at ../sysdeps/generic/libc-start.c:129
(gdb)
```

クライアントのリクエストを待っているところなので、クライアントからSQLを実行すると、バックエンド側でステップ実行などができる

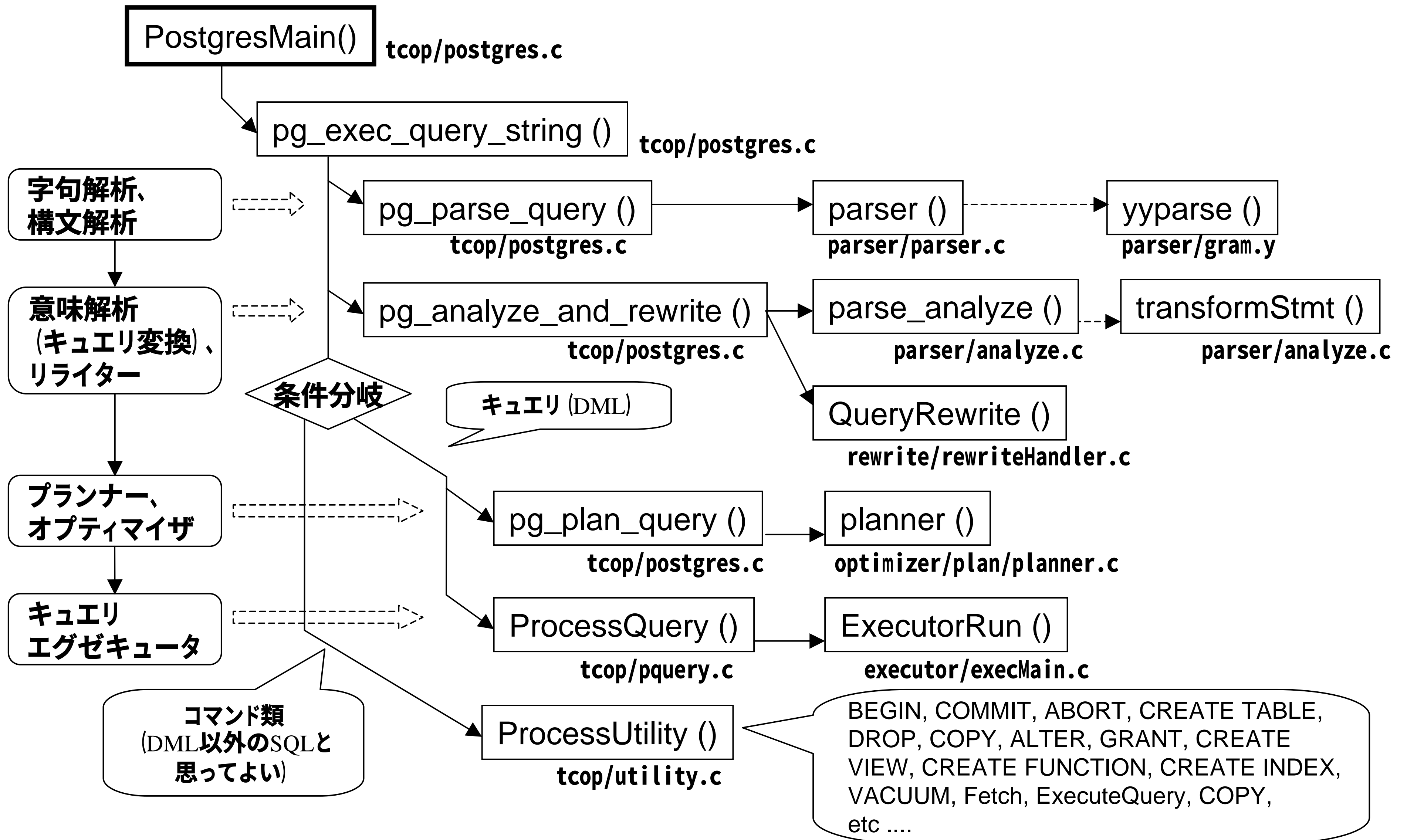
Copyright(C)2003 NTT DATA Corporation
Copyright(C)2003 NTT DATA Intellilink

参考 [第1回目の資料より]：main() からの処理



Copyright(C)2003 NTT DATA Corporation
Copyright(C)2003 NTT DATA Intellilink

参考 [第1回目の資料より]：PostgresMain() からの処理



Copyright(C)2003 NTT DATA Corporation
Copyright(C)2003 NTT DATA Intellilink

gdb の使い方３(コアファイルの調査)

- core ファイルがある場合、core の調査も可能
  - コアファイルを指定して、gdbを起動する

```
frisk(47)% gdb a.out core.10865
```

  - コアファイルの調査の場合の主な操作
    - 落ちた場所は、起動時に分かる

```
frisk(47)% gdb a.out core.10865
GNU gdb Red Hat Linux (5.1.90CVS-5)
Copyright 2002 Free Software Foundation, Inc.
              :
#0  0x08048454 in func_2 () at main.c:25
25                i = *p;
(gdb)
```

    - backtrace で、core dumpした場所を確認する

```
(gdb) bt
#0  0x08048454 in func_2 () at main.c:25
#1  0x08048410 in main () at main.c:5
#2  0x42017499 in __libc_start_main () from /lib/i686/libc.so.6
```

    - 変数の値を確認する

```
(gdb) print p
$1 = (int *) 0x0
```

core を出力するためには、coredumpsize の制限を取り除いておく必要がある。
csh 系の場合、unlimit coredumpsize と実行しておけばよい。
bash の場合、 ulimit -c unlimited で制限がなくなる。(ただし、強い制約 -H で変更されている場合、元の値より増やせないのでエラーになる)

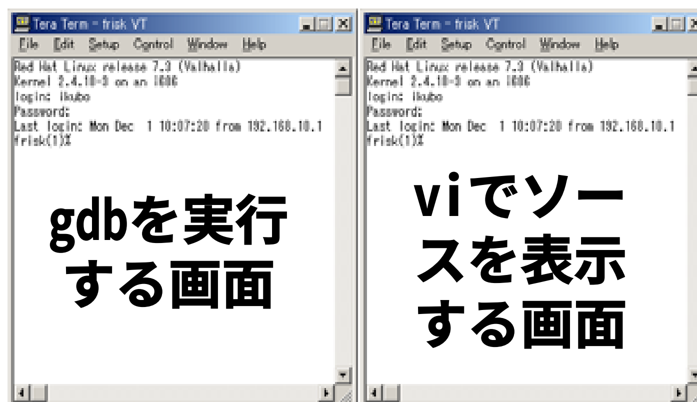
main() から呼び出された、func_2() の中であることが分かる

変数 p を見てみると 0 が入っているので、0番地参照のために落ちたということが分かる。

Copyright(C)2003 NTT DATA Corporation
Copyright(C)2003 NTT DATA Intellilink

gdb の使い方に関して

- **本当は、**emacs **や** GUI **環境と一緒に使うと便利なはずだが**…。
- vi + ctags **を好んで使用してるので、ターミナルを並べて使う。**



- **基本的に** vi **でソースを追いながら、進めたいところに**break point **を設定して、**gdb **の方を進める**

- **このような使い方をしているので、基本的には面倒**….。

Copyright(C)2003 NTT DATA Corporation
Copyright(C)2003 NTT DATA Intellilink

その他の gdb の便利な機能

- **変数を監視**
  - 「watch **変数名**」**で、変数が書き換えられたところで止まってくれる**
- print **の省略**
  - 「display **変数名**」**で、ブレークポイントで止まるたびに、その変数を表示する**
- **まとめて何バイトかダンプさせる**
  - 「x/20x」**で、4バイト単位で20個分の出力をしてくれる**

- **環境設定（例：構造体の表示で、項目ごとに改行を入れる）**
  - set print pretty on
  - show **で、設定を確認できる**（ show print pretty ）
  - show **で引数を省略すると、設定できる項目の状態が確認できる**
  - **いつも使うものは、**~/.gdbinit **に書いておけば、毎回読み込んでくれる**

Copyright(C)2003 NTT DATA Corporation
Copyright(C)2003 NTT DATA Intellilink

strace の使い方

■ 機能

- ◆ システムコールを監視する

■ 基本的な使い方

- ◆ コマンドラインから指定して、プロセスを監視する
- ◆ 実行中のプロセスにアタッチして、プロセスを監視する

■ コマンドラインから指定する

strace.sql の中身は、実行したいSQL。
例えば、select username from pg_user; など

```
frisk(57)% strace postgres template1 < strace.sql > & log1
```

ファイルの内容

```
execve("/home/ikubo/work/pg734dbg/bin/postgres", ["postgres", "template1"], [/* 30 vars */]) = 0
uname({sys="Linux", node="frisk", ...}) = 0
brk(0)                                  = 0x8227964
open("/etc/ld.so.preload", O_RDONLY)    = -1 ENOENT (No such file or directory)
open("/home/ikubo/work/pg734dbg/lib/i686/mmx/libz.so.1", O_RDONLY) = -1 ENOENT (No such file or directory)
stat64("/home/ikubo/work/pg734dbg/lib/i686/mmx", 0xbfffeba0) = -1 ENOENT (No such file or directory)
open("/home/ikubo/work/pg734dbg/lib/i686/libz.so.1", O_RDONLY) = -1 ENOENT (No such file or directory)
                                                :
                                                :
unlink("/home/ikubo/data/postmaster.pid") = 0
munmap(0x4001f000, 4096)                = 0
_exit(0)                                = ?
```


Copyright(C)2003 NTT DATA Corporation
Copyright(C)2003 NTT DATA Intellilink

strace の使い方２（プロセスへのアタッチ）

- psql を例に挙げて説明
  - psql で接続
  - ps –ax で、プロセス番号を確認
  - strace でプロセスをアタッチする

クライアントのリクエスト待ちで止まっている

```
frisk(61)% strace –p 1234
recv(7,
```

  - psql でコマンドを実行

psql

```
test=# begin;
```

strace

```
"Qbegin\0", 8192, 0)           = 7
gettimeofday({1070246135, 980824}, NULL) = 0
close(38)                               = 0
rt_sigaction(SIGPIPE, {SIG_IGN}, {SIG_IGN}, 8) = 0
send(7, "CBEGIN\0Z", 8, 0)              = 8
rt_sigaction(SIGPIPE, {SIG_IGN}, {SIG_IGN}, 8) = 0
recv(7,
```


Copyright(C)2003 NTT DATA Corporation
Copyright(C)2003 NTT DATA Intellilink

strace の使い方２（プロセスへのアタッチ）のつづき

psql

```
test=# insert into foo values ( 1 );
```

strace

```
"Qinsert into foo values ( 1 );\0", 8192, 0) = 31
open("/home/ikubo/data/base/16976/16977", O_RDWR|O_LARGEFILE) = 38
_llseek(38, 0, [0], SEEK_END)           = 0
_llseek(38, 0, [0], SEEK_END)           = 0
write(38, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 8192) = 8
192
rt_sigaction(SIGPIPE, {SIG_IGN}, {SIG_IGN}, 8) = 0
send(7, "Pblank\0CINSERT 16979 1\0Z", 24, 0) = 24
rt_sigaction(SIGPIPE, {SIG_IGN}, {SIG_IGN}, 8) = 0
recv(7,
```

foo のテーブルは、今 fd = 38 が割り当てられている

psql

```
test=# commit;
```

strace

```
"Qcommit;\0", 8192, 0)          = 9
time(NULL)                              = 1070246318
_llseek(39, 8536064, [8536064], SEEK_SET) = 0
write(39, "Z\320\1\0\22\0\0\0\0\0\0\0\0@\202\0\222\n\0\0\0\0\0\0\0"..., 8192) = 8192
fdatasync(39)                           = 0
rt_sigaction(SIGPIPE, {SIG_IGN}, {SIG_IGN}, 8) = 0
send(7, "CCOMMIT\0Z", 9, 0)             = 9
rt_sigaction(SIGPIPE, {SIG_IGN}, {SIG_IGN}, 8) = 0
recv(7,
```

xlogファイルは、fd = 38 が割り当てられていると予想される

wal_sync_method が fdatasync なので、ここでは、fdatasync が実行されている

Copyright(C)2003 NTT DATA Corporation
Copyright(C)2003 NTT DATA Intellilink