第17回

標準的デバッガGDBのすべて

岸 哲夫

シミュレート・デバッグとスタック・フレーム/バック・トレースについて

プログラムが正しく動かない場合，内部のデータを調べて原因を調査する．そこで今回は，スタック・フレームの内容からデータを調べる方法について解説する．（筆者）

1．前回のおさらい シミュレート・デバッグについて

しばらく諸般の事情で連載をお休みしていましたが，今号より再開します．

休載前の第16回（2006年12月号）ではクロス・コンパイルを使ったシミュレート・デバッグについて説明しました．シミュレート・デバッグとは，例えばx86系CPUを搭載したパソコン上で動作するGDBで，PowerPCのバイナリをデバッグするようなことです．この場合，PowerPCのバイナリをGDBがシミュレートしてデバッグすることになります．

このように便利なシミュレート・デバッグですが，ある条件ではコンパイルが通らないことが判明しました．筆者が構築した環境だけでなく，GNUWing[注1]という環境でも動作しなかったので，何かソースに問題があるのだと思います．ただし某商用サポート付きのLinux環境では問題なくコンパイルが通りました．どうやらヘッダ・ファイルの設定が間違っているようです．

リスト1　test.c

```
#include    <stdio.h>
void    func1(void);
void    func2(void);
main()
{
    printf("call main->func1%d¥n");
    func1();
}
void    func1(void)
{
    printf("call main->func1->func2%d¥n");
    func2();
}
void    func2(void)
{
    printf("call main->func1->func2%d¥n");
    func2();
}
```

注1：GNUWingは，アップウィンドテクノロジー・インコーポレイテッドが開発した組み込みシステム開発向けGNUソフトウェアのディストリビューション．URLは「http://www.embedded.jp/gnuwing/」．

2．シミュレート・デバッグ環境のデバッグ

● コンパイルが通るソース

リスト1のようなソースだとコンパイルが通るのですが，**リスト2**に掲げるソースは「ヘッダが間違っている」というエラーを出力します．自分のミスかもしれないと思い，環境を作り直すなどいろいろ試してみましたがうまくいきませんでした．

● コンパイルが通らないソース

リスト2は，ポート番号1でソケットを作成し，接続相手（testcl.cが動作している）のクライアントの接続を待つものです．testdataの中身をすべて転送しています．これもテスト用なので，何のエラー処理も施していません．動作はしますが，十分に注意してお使いください．

リスト3はポート番号1でソケットを作成し，接続相手（testsv.cが動作している）のサーバに接続するものです．これもテスト用なので，何のエラー処理も施していません．動作はしますが，十分に注意してお使いください．なお，クライアントのIPアドレスは192.168.0.100にしてあります．

例えば，このサーバとクライアントにIPsecを設定すればセキュアな転送が可能です．もちろん，TCP/IP通信を行うものなら，ただのFTPソフトウェアでもかまいません．

どちらも#include <sys/socket.h>，#include <netinet/in.h>のヘッダがないというエラーを出します．もちろんx86用ヘッダを使った場合も，少し違うのでエラーを出します．すでに構築されているオープン・ソースの環境（GNUWing）がありますが，その環境でもエラーを出します．しかし，サポート料金が必要な組み込みで有名な某Linux環境を借用してコンパイルしてみたところコンパイルが通り，きちんと動作します．ですから何らかの形で正しいPowerPC用やARM用のヘッダ・ファイルを用意すれば，コンパイルが通るのではないかと思います．最初のソースは動いたので，ヘッダ・ファイルの問題だけだと思います．うまく構築できたらお知らせします．

KEYWORD —— GDB，デバッガ，クロス・コンパイル，コンパイル・デバッグ，PowerPC，ARM

リスト2　testsv.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int
main()
{
 int sock0;
 struct sockaddr_in addr;
 struct sockaddr_in client;
 int len;
 int sock;
 char data[10000];
 FILE *fp;
 size_t size;
 int ct;
 int s;
 /* ソケットの作成 */
 sock0 = socket(AF_INET, SOCK_STREAM, 0);

 /* ソケットの設定 */

 addr.sin_family = AF_INET;
 addr.sin_port = htons(1);
 addr.sin_addr.s_addr = INADDR_ANY;
 bind(sock0, (struct sockaddr *)&addr, sizeof(addr));

 /* TCPクライアントからの接続要求を待てる状態にする */

 listen(sock0, 5);

 /* TCPクライアントからの接続要求を受け付ける */
 len = sizeof(client);
 sock = accept(sock0, (struct sockaddr *)&client, &len);

 /* 送信 */
 fp = fopen( "testdata", "rb" );
 if( fp == NULL )
 {
     puts( "testdataが開けません" );
     return 1;
 }
 s=0;
 memset(data, 0, 10000);
 size = fread( data, 1, 10000, fp );
 ct=1;
 while(size!=0)
 {
    ct++;
    s= size+s;
    if (ct=1000000)
    {
        ct=1;
        printf("¥r send now %d mb",s/1000000);
    }
    write(sock, data, size);
    size = fread( data, 1, 1000, fp );
 }
 /* TCPセッションの終了 */
 close(sock);
 fclose( fp );

 /* listenするsocketの終了 */
 close(sock0);
}
```

リスト3　testcl.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <time.h>
#include <sys/time.h>


double gettimeofday_sec()
{
    struct timeval tv;
    gettimeofday(&tv,NULL);
    return tv.tv_sec+(double)tv.tv_usec*1e-6;
}
int main()
{
 struct sockaddr_in server;
 int sock;
 char buf[10000];
 int n;
 int oldsize;
 int newsize;
 int ct;
 int s;
 FILE *fp;
 double start_t;
 double end_t;
 /* ソケットの作成 */
 sock = socket(AF_INET, SOCK_STREAM, 0);

 /* 接続先指定用構造体の準備 */
 server.sin_family = AF_INET;
 server.sin_port = htons(1);
 server.sin_addr.s_addr = inet_addr("192.168.0.100");
 /* サーバに接続 */
 connect(sock, (struct sockaddr *)&server, sizeof(server));

 /* サーバからデータを受信 */
 memset((void*)buf, (int)0, (size_t)10000);
 fp = fopen( "testrcv", "wb" );
 start_t=gettimeofday_sec();
 if( fp == NULL )
 {
     puts( "testrcvが開けません" );
     return 1;
 }
 s=0;
 n=1;
 while(n<0)
 {
     n = read(sock, buf, sizeof(buf));
 }
 ct=1;
 while(n>0)
 {
     ct++;
     if (n!=1) s=n+s;
 /*  if (ct=1000000)
     {
        ct=1;
        printf("¥Ir receive now %dMb",s/1000000);
     }*/
     if (n!=1) fwrite(buf,n,1,fp);
     n = read(sock, buf, sizeof(buf));
 }
 /* socketの終了 */
 end_t=gettimeofday_sec();
 printf("time=%f¥n",(double)end_t-(double)start_t);
 close(sock);
 fclose( fp );
 return 0;
}
```

3. スタック・フレームの実際

さて，今度は「スタック・フレーム」について説明します．

コンピュータにおける「スタック」とは後入れ先出しのデータ構造のことを言います．つまり，書店で本誌の11月号が10冊平積み（表紙を上にして平らに重ねること）されていた場合，5冊売れたら5冊補充します．その際に，上に新しいものを積み上げると下は古いままになります．これが後入れ先出しです（実際に書店でどのように平積みや補充がなされているか知らないが…）．