Binary2.0 Conference 2006 発表資料

Hello, binary world!

さとう ゆうすけ <ads01002 @ nifty.com>

自己紹介

さとう ゆうすけ (d:id:yupo5656)

- ソフトウェア エンジニア
- Hello World愛好家

- Binary Hacks の執筆に参加
  - 主な担当ハック:
    #25 「glibcを使わないでHello Worldを書く」

自己紹介

- 好きな休日の過ごし方
  - コードを読む、書く
  - 読書
    - The Single UNIX Specification
    - ISO/IEC 9899:1999　(C)
    - ISO/IEC 14882:2003（C++)
  - SICPの問題をC++で解く
  - Z80マイコン製作
  - Hello World

自己紹介 + 本日の話のレベル

- user-space バイナリアン
  - kernel-space バイナリアン
    - リアルバイナリアン

本日の内容

- バイナリアンの分類
- Hello World愛好家の分類
- GCC拡張入門
- GCC拡張を使ったHello World ５連発
- まとめ

目標：
いつのまにかGCC拡張機能をマスター

Hello world 愛好家の分類

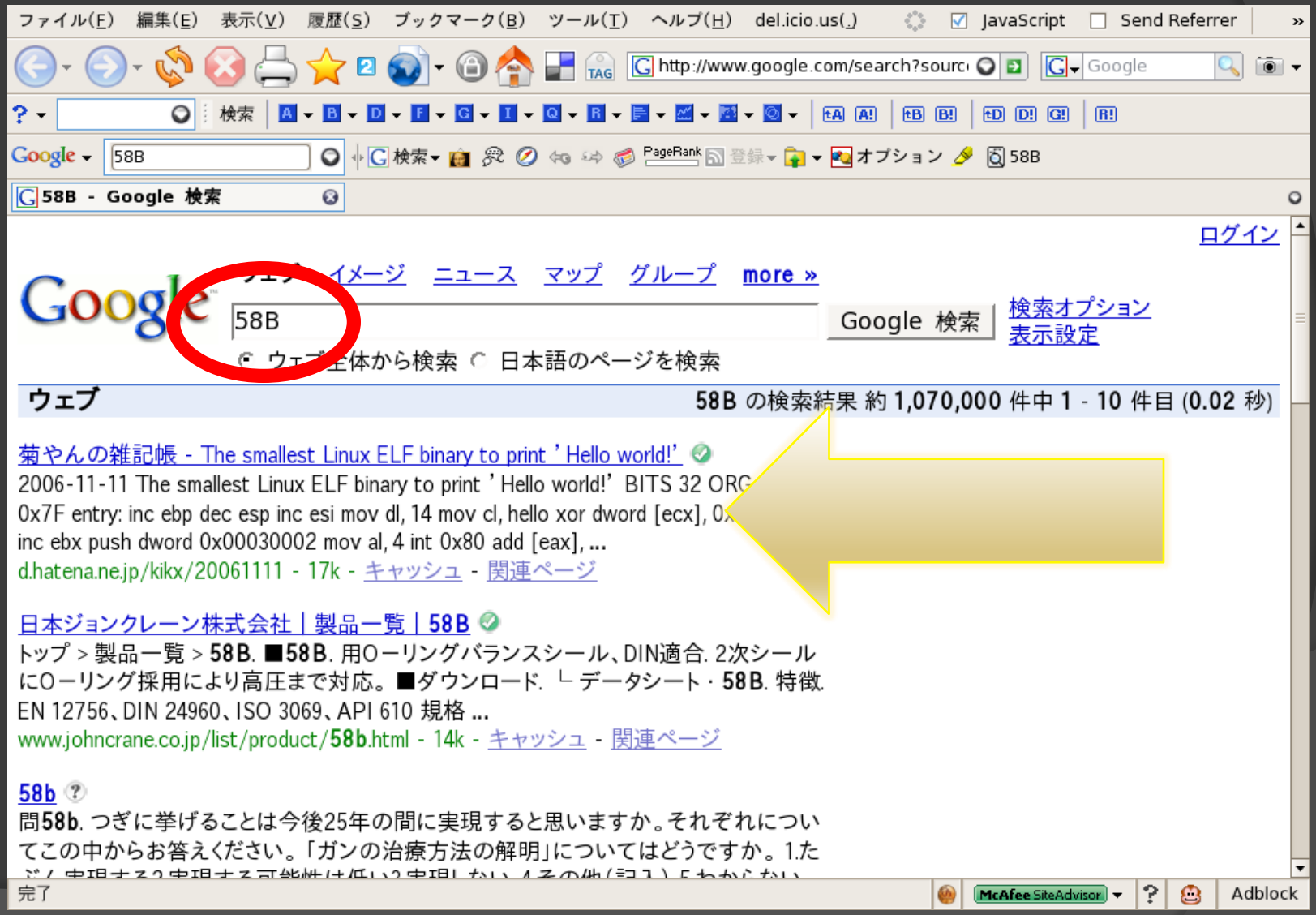
GCC拡張すごいよ派

ELF Golf派

1. ELF Golf派

- “Hello, world!” を出力する、世界最小最軽量のELFバイナリを追求
- 深追い対象:
  - ELFフォーマット
  - ローダ（linux-2.6.xx/fs/binfmt_elf.c）
- 会話例:
  「アリアリで58Bですか。それはすごい」
  - アリアリ: **Hello, World!**
  - アリナシ: **Hello, World**
  - ナシアリ: **Hello World!**
  - ナシナシ: **Hello World**

58B


ファイル(F) 編集(E) 表示(V) 履歴(S) ブックマーク(B) ツール(T) ヘルプ(H) del.icio.us(J) JavaScript Send Referrer
http://www.google.com/search?sourc
Google
検索
58B
58B - Google 検索
ログイン
Google
イメージ ニュース マップ グループ more »
58B
Google 検索
検索オプション
表示設定
ウェブ全体から検索 日本語のページを検索
ウェブ
58B の検索結果 約 1,070,000 件中 1 - 10 件目 (0.02 秒)
菊やんの雑記帳 - The smallest Linux ELF binary to print 'Hello world!'
2006-11-11 The smallest Linux ELF binary to print 'Hello world!' BITS 32 ORG
0x7F entry: inc ebp dec esp inc esi mov dl, 14 mov cl, hello xor dword [ecx], 0
inc ebx push dword 0x00030002 mov al, 4 int 0x80 add [eax], ...
d.hatena.ne.jp/kikx/20061111 - 17k - キャッシュ - 関連ページ
日本ジョンクレーン株式会社 | 製品一覧 | 58B
トップ > 製品一覧 > 58B. ■58B. 用Oーリングバランスシール、DIN適合. 2次シールにOーリング採用により高圧まで対応。■ダウンロード. └ データシート・58B. 特徴. EN 12756、DIN 24960、ISO 3069、API 610 規格 ...
www.johncrane.co.jp/list/product/58b.html - 14k - キャッシュ - 関連ページ
58b
問58b. つぎに挙げることは今後25年の間に実現すると思いますか。それぞれについてこの中からお答えください。「ガンの治療方法の解明」についてはどうですか。1.たぶん実現する2.実現する可能性は低い3.実現しない 4.その他(記入) 5.わからない
完了
McAfee SiteAdvisor
Adblock

2. GCC拡張すごいよ派（私）

- 本日のメインテーマ
- 無駄に華麗なCのコードを追及
- GCC拡張機能を無意味に活用

- 一見意味のわからないコード
- 実はHello world
- [これはすごい] [これは便利] [lifehack] 1024 users

- 実はELF Golfについていけないだけ

予習: GCC拡張入門

```
__attribute__((constructor))
void before_main(){
  puts("mainの前に呼ばれるよ");
}

int main(){ puts("mainだよ"); }

__attribute__((destructor))
void after_main(){
 puts("mainの後に呼ばれるよ");
}
```

GCC拡張機能

- constructor
- destructor
- section
- cleanup

ctor/dtor関数が呼ばれるしくみ（概要）

```
_start()
    _libc_start_main()
        _do_global_ctors_aux()
            コンストラクタ関数1()
            …
        main()
            exit()
                _do_global_dtors_aux()
                    デストラクタ関数1()
                    …
                _exitシステムコール
```

libcが呼んでくれる！

mainからreturnで戻った場合は、_libc_start_main()がexit()を呼ぶ

本日のハローワールド５種類

- 二度呼ばれるmain（によるハローワールド）
- 蹂躙されるmain
- 呼ばれるのに関数にしてもらえないmain
- スルー力の高いmain
- 何もしないmain

- 二度呼ばれるmain
- 蹂躙されるmain
- 呼ばれるのに関数にしてもらえないmain
- スルー力の高いmain
- 何もしないmain

（ネタ1/5）二度呼ばれるmain

```
__attribute__((constructor)) int main() {
  static int i = 0;
  if (i) puts("world!");
  else i = printf("hello, ");
}
```

- mainの前に好きな関数を呼べる
- じゃあ、mainの前にmainを呼んでみるのはどう？

d:id:shinichiro_h:20060809より、一部改変

デモ

大切なお知らせ

- 以降のネタにつきましては、出力がいつも同じであるため、デモを省略させていただきます

main2度呼びのコールツリー

```
_start()
    _libc_start_main()
        _do_global_ctors_aux()
            main()
        main()
        exit()
```

- mainを二度呼んでも全く問題なし！
- どう見ても単なる関数です。本当に(ry
- 次ネタ以降、~~偉そうな~~mainの地位をどんどん危うくしていきますよ（main蹂躙芸）

- 二度呼ばれるmain
- 蹂躙されるmain
- 呼ばれるのに関数にしてもらえないmain
- スルー力の高いmain
- 何もしないmain

（ネタ2/5）蹂躙されるmain

```
__attribute__((constructor, destructor))
void x(){
  static int i = 0;
  if (i) puts("world!");
  else i = printf("hello, "), exit(0);
}
```

- mainを（二度呼ぶどころか）一度も呼ばないハローワールド
- constructor関数内でexit

コールツリー

```
_start()
    _libc_start_main()
        _do_global_ctors_aux()
            x() ←コンストラクタ
              exit()
                  _do_global_dtors_aux()
                    x() ←デストラクタ
                  _exitシステムコール
```

リンク失敗..

- (.text+0x18): undefined reference to `main‘
- main関数がないと実行ファイルが作成できない
- どうせ呼ばれないのに…

[Q] カラのmain関数を書けばよいのでは？
[A] だが断る。

ソリューション: main変数

```
int main = 0;
__attribute__((constructor, destructor))
void x(){
  if (main) puts("world!");
  else main = printf("hello, "), exit(0);
}
```

- リンカ的には関数も変数もただの「シンボル」
- 「main変数」を書いとけばよい
- 変数にしたからには働いてもらう！
  - main様を、x()の動作を変えるためのフラグ扱い

コンパイル風景

```
% gcc iyana.c
iyana.c:5:warning: ’main’is usually a function
% ./a.out
Hello, World!
```

知ってます

- 二度呼ばれるmain
- 蹂躙されるmain
- 呼ばれるのに関数にしてもらえないmain
- スルー力の高いmain
- 何もしないmain

（ネタ3/5）
呼ばれるのに関数にしてもらえないmain

- mainをフラグとしてのみ使うのは失礼。存在を無視しすぎ。ゆとり教育の弊害。
- ちゃんとmainを呼んであげる


main無視は失礼だって言うんだろ？
なら一応呼んでやるよ！
変数のままだけどな！
まさに外道
まさに外道ジェネレータ
http://gedo-style.com

呼ばれるのに関数にしてもらえないmain

```
main; // 変数（フラグ）兼 関数
__attribute__((constructor, destructor))
void x(){
  if (main) puts("world!");
  else printf("hello, "), main = 195;
}
```

- exitする代わりに、mainに謎の数値を代入
  - 195 = 0xC3 = x86のRET命令
- このmain変数は「RET命令だけの関数」と同じ

d:id:shinichiro_h:20060829より、一部改変

コールツリー

```
_start()
    _libc_start_main()
        _do_global_ctors_aux()
            x() ←コンストラクタ，mainを195に
        main変数() ← 一瞬でRET
        exit()
            _do_global_dtors_aux()
              x() ←デストラクタ
            _exitシステムコール
```

実行

```
% ./a.out
Hello, World!
```

微妙に使いみちがあったりする

- データ実行許可（NXなし）環境

```
% ./a.out
hello, world!
```

- データ実行禁止（NXあり）環境

```
% ./a.out
zsh: segmentation fault  ./a.out
```

- /proc/sys/kernel/exec-shield を見るまでもなくOSの設定状況がわかる!（見たほうが早いけど）

- 二度呼ばれるmain
- 蹂躙されるmain
- 呼ばれるのに関数にしてもらえないmain
- スルー力の高いmain
- 何もしないmain

(ネタ4/5)スルー力の高いmain

```
__attribute__((section(".text"))) main = 2425393296;

_() {
  __attribute__((destructor)) _(){ puts("world!"); }
  printf("hello, ");
}
```

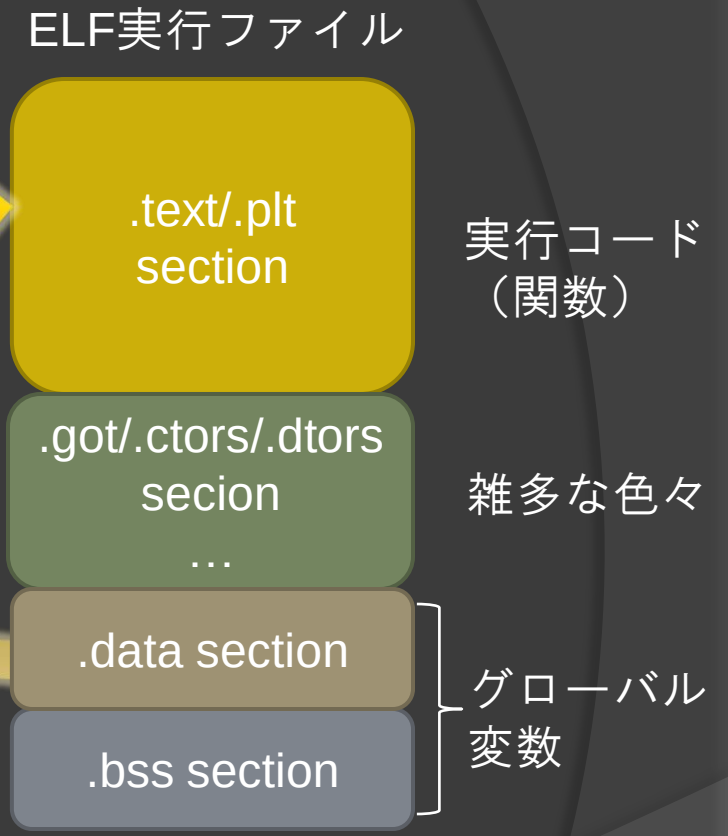
- 今回はちゃんと(?) main変数から実行開始
  - constructor関数なし
- 関数名が _ だけなのは、Haskellへの対抗心
- 関数の中に関数を書けるのもGCC拡張
  - 今回はオシャレで使っただけ、深い意味はなし

見やすく整形

```
// 同じ動作をするコード
__attribute__((section(".text")))
int main = 0x90909090;

void f() { printf("hello, "); }

__attribute__((destructor))
void g() { puts("world!"); }
```



- 0x90 = NOP命令
- 2425393296 = 0x90909090 = NOP命令４つ
- main変数のELF上の位置を変更（.data → .text）
  - ELFバイナリ上、関数fのすぐ上にmainが配置される

新概念: 関数フォールスルー

NOP４つだけ

- main変数
- エントリポイント

Hello出力

- 関数f

World出力

- 関数g
- デストラクタ属性

実行

```
% ./a.out
Hello, World!
```

逆アセンブルするとわかりやすい

```
% objdump -D a.out -j .text
08048384 <main>:
 8048384:       90                      nop
 8048385:       90                      nop
 8048386:       90                      nop
 8048387:       90                      nop

                                隙間なし
08048388 <f>:
 8048388:       55                      push   %ebp
 8048389:       89 e5                   mov    %esp,%ebp
 804838b:       83 ec 08                sub    $0x8,%esp
 804838e:       c7 04 24 80 84 04 08    movl   $0x8048480,(%esp)
 8048395:       e8 1e ff ff ff          call   80482b8 <printf@plt>
 804839a:       c9                      leave
 804839b:       c3                      ret

0804839c <g>:
 804839c:       55                      push   %ebp
```

① main変数実行開始

② 関数fに
フォールスルー

③ Hello 出力

④ main変数からリターン

- 二度呼ばれるmain
- 蹂躙されるmain
- 呼ばれるのに関数にしてもらえないmain
- スルー力の高いmain
- 何もしないmain

(ネタ5/5)何もしないmain

- 自動変数にcleanup属性付与
  - 自動変数がスコープから外れたとき
  - 指定した１引数関数を呼べる
- 関数には、その自動変数へのポインタが渡る

- 脊髄反射的にハローワールドに応用

```
// 変数しかないように見えるが・・
int main() {
  __attribute__((cleanup(puts)))
    const char hoge[] = "hello, world!";
}
```

解説

ソースコード

```
int main() {
  __attribute__((cleanup(puts)))
    const char hoge[] = "hello, world!";
}
```

コンパイル

GCCが出力するコードのイメージ

```
int main() {
    const char hoge[] = "hello, world!";
    puts(&hoge);  // GCCが自動で追加
}
```

何もしないmain その2（エキスパート・ハローワールダ向け）

```
main;
__attribute((constructor, destructor)) _() {
  if (main++) {
    __attribute((cleanup(puts)))   char _[] = "world!";
  } else {
    __attribute((cleanup(exit)))   int  __;
    __attribute((cleanup(printf))) char _[] = "hello,";
  }
}
```

- 積極exit型main蹂躙スタイル
- _() → printf() → exit() → _() → puts()

応用: C言語でRAIIイディオム

- cleanup属性を使うと、CでC++のRAIIイディオムの模倣が可能
- RAIIイディオムとは！

```
// C++
void bar() {
  scoped_lock lk(mutex);
  ...
            return; // ロック自動解放
  ...
  return; // ロック自動解放
}
```

普通のCだと手動解放、超面倒

```
// C
void bar() {
  mutex_lock(mutex);
  ...
            mutex_unlock(mutex); // 手動解放
            return;
  ...
  mutex_unlock(mutex); // 手動解放
  return;
}
```

もの作るってレベルじゃねーぞ！

GCCならラクラク

- ほぼC++相当、RAIIもどき
- マクロの中身は d:id:yupo5656:20060609
  - GCCのcleanup属性を使ってます

```
// GCC
void bar() {
  SCOPED_LOCK(mutex);
  ...
            return; // ロック自動解放
  ...
  return; // ロック自動解放
}
```

まとめ

GCCの拡張機能便利！

ご清聴ありがとうございました

参考になる資料

1. Binary Hacks （宣伝）

2. ELF: プログラマの視点から

   http://www.globe.to/~oka326/archive/elf_doc_sgml_ja/elf_doc.html

3. GCCのオンラインマニュアル
   - Function Attributes

     http://gcc.gnu.org/onlinedocs/gcc/Function-Attributes.html
   - Variable Attributes

     http://gcc.gnu.org/onlinedocs/gcc/Variable-Attributes.html