急がば回れ、一歩ずつ。

USP MAGAZINE

for the sophisticated shell scripters

日本で唯一のシェルスクリプト総合誌

Update版

2014 8 August Vol. 16 ¥500

特集

IT美女特集

女の生き方とこれから

漢のUNIX

「シェル芸」に効く AWK処方箋

Artwork created by Koji Ishizuka. © 2012 Zie's Bank .Co,. LTD All rights reserved.

USP MAGAZINE

2014 August vol.16

Contents

04 特集
IT美女特集 女の生き方とこれから

14 **漢のUNIX 第13回**
後藤大地

19 未来に活きる！ 現場で使える！
データモデリング 第6回
熊野憲辰

23 **ユニケージ開発手法 コードレビュー 第5回**
大内智明

28 **IPv6新時代を体感しよう！ 第3回**
波田野裕一

32 組織文化を変える汗と涙の物語
アジャイル改善塾 第4回
山海一剛

34 **りゅうちの工作員日記 第5回**
りゅうちてつや

38 「シェル芸」に効く
AWK 処方箋 第5回
斉藤博文

42 シェル芸勉強会後追い企画
Haskellでやってはいかんのか？ 第5回
上田隆一

46 教えて先輩♡
サーバー運用お助けTips 第3回
濱田康貴

50 **中小企業手作りIT化奮戦記 第12回**
菅雄一

53 **スズラボ通信 第8回**
すずきひろのぶ

58 **人間とコンピュータの可能性 第16回**
大岩元

60 **法林浩之のFIGHTING TALKS 第3回**

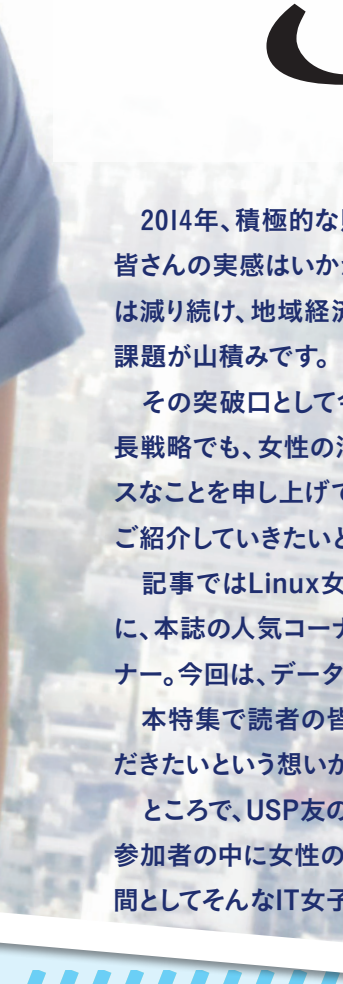
63 自転車の上にも30年　シェル魔人／編集後記

こちらの美女は誰？
答えは特集記事の中に……

（特集企画：かまた ひろこ）

IT美女特集

女の生き方とこれから



2014年、積極的な財政出動、大胆な金融緩和政策を背景に、国内の景気が上向きつつあるそうです。皆さんの実感はいかがですか？ ただ、長いスパンで眺めると——少子高齢化で日本全体の労働力人口は減り続け、地域経済の低迷、過疎化、限られた財源の中で若者世代の負担増が懸念されるなど難しい課題が山積みです。

その突破口として今、あらためて女性の存在が注目されています。アベノミクス第三の矢と言われる成長戦略でも、女性の活躍が期待されています。それは、IT業界でも例外ではありません……などと、シリアスなことを申し上げてしまいましたが、本特集では強くたくましく美しく花咲くIT女子の姿を、明るく楽しくご紹介していきたいと思います。

記事ではLinux女子部とのコラボで開催したイベント、TechGIRLの盛り上がりをお伝えします。さらに、本誌の人気コーナー「IT美女図鑑」が復活！ “IT美女”といえども女性の中身を重視しているこのコーナー。今回は、データセンターをこよなく愛する女性です。

本特集で読者の皆様に、IT業界の女性達のありのままの姿を知ってほしい、身近な存在に感じていただきたいという想いが、私たちにはあります。

ところで、USP友の会が開催するシェル芸勉強会（シェルのワンライナー勉強会）には毎回40名ほどの参加者の中に女性の姿がちらほら見られます。USP友の会会員メンバーの一人として、IT業界の一人の人間としてそんなIT女子を心から応援しています。

女の生き方の現実と描く未来

TechGIRL開催報告

2014年5月24日(土)に、USP研究所、GMOインターネット、Linux女子部の共催で「TechGIRL」を開催致しました。当日MCを担当したLinux女子部 部長の平愛美がレポートいたします。

TechGIRL 発足のきっかけ

「TechGIRL」とは、女子部から派生した、IT 業界を明るく、そしてワクワクさせる為の交流イベントです。

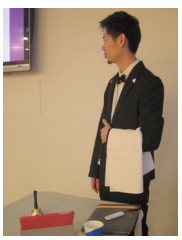
IT 業界で働く女性の悩み…「もっとプレゼンする機会が欲しい」「女性エンジニア同士の交流を深めたい」といった声に応えるべく、TechGIRL を立ち上げました。

そして記念すべき第一回目は、GMO インターネットグループが運営するコミュニケーションスペース「GMO Yours」にて開催し、男女合わせて 100 名を超える参加者が集結しました。

女性スピーカーだらけの LT 大会

第一部は、IT 業界を支える女性スピーカーによる LT 大会を行いました。ちなみに、IT 系イベントの LT は、銅鑼を叩く女性 (どら娘) が配置されているのが一般的ですが、本イベントでは増田さんが執事役となりました。LT の制限時間 (5 分間) を過ぎると、呼び鈴を鳴らして「お嬢様、お時間です」と優しくお知らせしました。

第一部では計 10 名のスピーカーによるLTを行いました(本誌では誌面の関係上、5 名のスピーカーをご紹介いたします)。



【執事役】サイバーエージェント
増田謙一さん

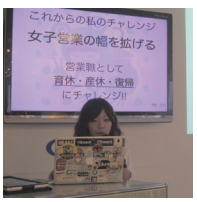
●「ワーキングマザーエンジニアの三種の神器」
たかはし るみさん

独立行政法人 情報通信研究機構 (NICT)

ベンダー資格 CCIE を保有するエキスパートなネットワークエンジニアであり、一児の母でもある高橋さんは、仕事と子育ての両立をするための「三種の神器」は、「保育園」「家電」「パートナー」であると説明。さらに、パートナーから掛けてもらいたい魔法の言葉は「私が○○みているから、あなたは△△しておいで」──相手をいたわる心の余裕が重要であることを伝えました。今回、出産しても働きたい女性や、イクメンパパが多数参加していたこともあり、多くの参加者が共感し、参考にしていました。

●「新卒がClerの営業になってそれなりに時が経った話」永淵 恭子さん

株式会社サーバーワークス / JAWS-UG AWS User Group - Japan
クラウド女子会 コアメンバー



Cler（クラウドインテグレーター）の法人営業である永淵さんは、これまで男社会といわれていた営業職の世界に飛び込んで経験したことを本音で語り、会場を共感と笑いの渦に巻き込みました。女性ならではのポジティブさ、周囲への気配り、行動力は、営業職に向いていることをアピール。今後の目標は、女性営業職として産休・育休・復帰にチャレンジすることを宣言。これからの永淵さんの活躍から目が離せません。

●「Apache CloudStack のおはなし」大削 緑（おおげ みどり）さん

NTT コミュニケーションズ / 日本 CloudStack ユーザー会 副会長

パブリック・クラウドサービス、Cloudn（クラウド・エヌ）のアーキテクト、大削さんよる CloudStack の紹介プレゼン。大削さんはインプレス R&D 出版「CloudStack 実践ガイド」の著者でもあります。

CloudStack は無料で入手でき、システム要件を満たしている Linux マシンさえあれば、自分の手でクラウド環境を作ることができます。

女性エンジニアがクラウドの最前線で活躍していることに、多くの参加者が元気をもらいました。エンジニアの皆さんはぜひこの機会に、CloudStack でクラウド環境を作って、スキルアップを目指してみましょう。

●「女性エンジニアが長く働くコツ」石川 博子さん

株式会社エイチアイ・シス 代表取締役 / 日本プロジェクトマネジメント協会 Women's SIG リーダー

石川さんは理系大学を卒業後、大手ゲーム会社に就職。入社当初は「女性は結婚したら退職するから、女性には技術は教えられない」という理由で企画開発のお茶汲みからスタート。でも、エンジニアになることを諦めたくなくて他部署の上司に頼み込んだ結果、技術部に転部することができ所属していたゲーム会社で初の女性エンジニアになることができたそうです。

結婚後ゲーム会社を退職し、フリーランスを経て会社を設立。女性起業家となった今でも現役エンジニアを貫き通す理由は、もの作りで完成したときの達成感が嬉しかったり、IT 系コミュニティに参加することにより仲間が増えたり、新しい知識を吸収できて楽しいことが多いからだそうです。

仕事で無理をせず、「細く長く」続けることで、人生を楽しみながら仕事をするという働き方は、参加者に多くの気付きを与えたようです。

●「時短主婦の味方の紹介」ゆうこさん

株式会社 ISAO

インフラエンジニアの ゆうこさんによるプレゼン。ゆうこさんは一児の母で、Linux 女子部の勉強会に子連れでよく参加されています。

仕事と育児の両立のため、選択肢の一つとして育児短時間勤務制度（時短勤務）を利用する方法があります。ゆうこさんは時短勤務を選択していますが、限られた勤務時間の中で、インフラストラクチャ自動化フレームワークである「Chef」を活用し、業務の効率化を図っています。帰宅後も最速で家事をするために、家電をフル活用しています。

もちろん、両立できるのは心身の健康があってこそ。無理をしないこと、無理が出てきそうだったら実家や職場の上司に相談して一人で抱え込まないことが大事だと伝えました。

プレゼンで紹介された家電はどれも欲しくなる逸品ばかりで、中には購入を真剣に検討する参加者も。バランスを取って、無理せずにお仕事を続けることが大事ですね。

スピーカー公開済みのスライドは、Linux 女子部公式サイトにて紹介しております。

http://www.linux-joshi.net/2014/06/techgirl.html

その他スピーカー
- 「RHEL7 リリース間近!?systemdで焦らないための Fedora のススメ」Linux女子部 平 愛美
- 「ConoHa中の人」 GMOインターネット GRACEさん
- 「新入社員でもこわくない、やさしい世界のご案内」USP研究所 かまた ひろこ
- 「ますだっく執事になります」サイバーエージェント 増田 謙一さん
- 「Windows女子部のご紹介」 Windows女子部

第二部の懇親会では食事をしながら参加者同士で歓談をし、交流を深めることができました。また、希望者による LT を実施。こちらも大盛況でした！ 最後に全員で集合写真！

TechGIRL では、IT 業界で働く女性が主役のイベントを今後も企画する予定です。お楽しみに！

No.8 スズラボ通信

Alloc, Free and Die Hard

Author : **すずきひろのぶ**
Email: suzuki.hironobu@gmail.com

プログラマはその生涯で三度メモリーアロケーションについて考えるという古くからの言い伝えがあります。これは、その三度目のお話です※1。

古典的 UNIX の malloc

malloc は C 言語の標準ライブラリで、メモリーを割り当てます。

リスト 1

```
void *malloc(size_t size);
```

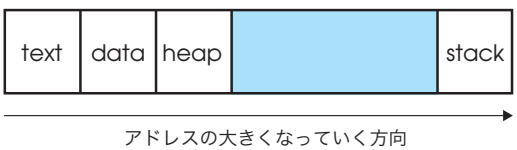
C 言語の標準ライブラリ、とあっさりと言っていますが、C 言語は UNIX のカーネルコードをアセンブラから高級言語へ書き換えるために作られた言語ですから、そもそもの malloc は UNIX のアーキテクチャーにどっぷりと依存しています。

man で見ればわかるように malloc は man3 のカテゴリ、つまりユーザー空間で動作するライブラリです。malloc の仕組みはざっくりと次のようなものです。

まずシステムコール brk/sbrk でヒープエリアを拡大します。そこで得られたメモリー空間から、malloc で求められたサイズに合わせて切り出していきます。ちなみにシステムコール brk/sbrk は POSIX.1-2001 では、すでに外されているシステムコールで、レガシー扱いになっています。

UNIX のプログラムが割り当てられるメモリーは図 1 のように、テキスト領域があって、次にデータ領域※2 があって、ヒープ領域（青色の部分を含む）があります。その右側には、スタック領域があります。さてシステムコール brk/sbrk によってヒープ領域が成長（拡大）します。その領域が個々の malloc の呼び出しの都度、切り出され、提供されていく「原資」となります。

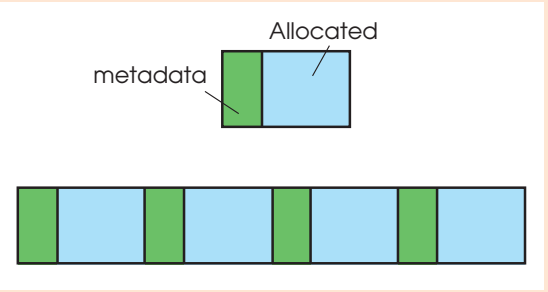
図 1：メモリー空間の割り当て



だったら malloc じゃなくシステムコール brk/sbrk でいいじゃないか、という話になりますが、malloc を使うと 2 つ大きな利点が得られます。1 つは、いちいち呼び出しコストのかかるシステムコールを呼ばないこと、もう 1 つは free をかけた領域を再利用することができる点です。現実問題として昔の UNIX は brk/sbrk を呼び出すと大変遅かったのです。

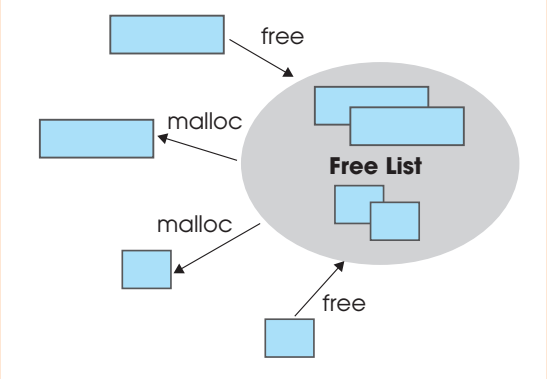
いったん大きなエリアで取ったメモリー空間は次に細分化された上で使われます。切り出す時は、アロケーションで呼び出したエリアの他に、頭の部分にメタデータ（制御情報）を入れます。まず切り出された時のメモリー空間を見てみましょう。こんな風に切り出されたデータが並んでいます（図 2）。

図 2：アロケーションされたエリア



アロケーションされたメモリー空間で、使われなくなったら、free を呼び出せば、それが再利用のリストに登録され、次回 malloc が呼び出された時に、そのリストの中に適したサイズのものがあれば、それを再利用します（図 3）。

図 3：Free List を使って再利用する



処理コストは実質的に数カ所のポインタの書き換えで済みます。それらのリストなどを管理するための情報がメタデータです。実装方法によって違いはありますが、こんな感じで色々と重要な情報が格納されています。

ソースコード 1：linux の /usr/include/malloc.h にあるメタデータ

```
struct mallinfo
{
  int arena;    /* non-mmapped space allocated from sys
tem */
  int ordblks;  /* number of free chunks */
  int smblks;   /* number of fastbin blocks */
  int hblks;    /* number of mmapped regions */
  int hblkhd;   /* space in mmapped regions */
  int usmblks;  /* maximum total allocated space */
  int fsmblks;  /* space available in freed fastbin blo
cks */
  int uordblks; /* total allocated space */
  int fordblks; /* total free space */
  int keepcost; /* top-most, releasable (via malloc_tri
m) space */
};
```

次にソースコード 2 を見てください。思いっきりバグっていますが、これで動いてしまいます。これは次に続く領域のメタデータを破壊しているはずです。元々は自分の持ち物になった 1 つのメモリー空間ですから、その範囲であればデータを書き込めてしまうので、このような大問題が、問題なくできてしまうのです。もちろん隣接しているアロケーションされたデータ領域がどこで使われているかなど、神のみぞ知るです。領域をオーバーしてデータを書き込むバグなど日常茶飯事なはずです。この時、実際にデータを不正に書き込むバグの部分と、その結果として何らかの現象を起こす部分には関連がありません。

ソースコード 2: 不正な領域を書き込む

```
#include <stdlib.h>
main(){
  char *a;
  int i;
  a = (char *)malloc(8);
  for( i = 0 ; i < 128 ; i++) {
    a[i] = 1;
  }
}
```

そこでいくらデバッグしてもまるでアロケーションしたデータが魔法のように書き換えられる、あるいは、malloc でデータを取ってこようとすると、free のリストにあったメタデータが壊れていて、malloc がまともに動作しないといった不思議な現象が現れます。

80 年代のまだ UNIX が普及し始めて C 言語を使い慣れない、そして malloc のソースコードなど一度も眺めたことがない人にとって malloc は魔法の、あるいは泥沼のブラックボックスだったと思います。いったんこのあたりでバグり始めると収集がつかなくなり、面倒くさいものでした。80 年代も終わりの頃、とあるソフトウェア生産関連の研究会の論文では「malloc にはバグがある云々、よって UNIX は安定していない OS である」ということが堂々と書かれていました。

それ以外にもメモリーリークという名の、アロケーション後、使われなくなった領域を free するのを忘れて、どんどん使用メモリー量が増えていくバグなど、様々な問題を引き起こすものがありました。そもそもメモリーが漏れて（リーク）、などという表現そのものが変な話です。設計レベルにおいて、利用するメモリー空間をどこで割り当て、どこで解放するのかをきちんと考慮していないために、リークじゃなく、延々と溜め込んでいるというのが正しい表現でしょう。

90 年代に入ると、そんなデバッグがむずかしい malloc に使える Purify というツールが出てきます。これは、今でも IBM Rational Purify として販売されています。

時代は mmap

これまでは brk/sbrk のシステムコールを使ってユーザーの使えるメモリー空間を確保していましたが、mmap が使えるようになって、mmap を使った

malloc 実装が出てきました。glibc の malloc 実装では mmap を使っています。

こうなると mmap で取れるメモリー領域が理論上の最大メモリ領域なので、glibc の malloc で使うことができるメモリ領域の総量が、size_t が 32 ビットだと 2^{32}-2 ページ分、size_t が 64 ビットだと、2^{64}-2 ページ分になります。

ソースコード 3 は、1GB のメモリー領域を確保しています。今時のメモリー管理は copy-on-write ですから、何か書き込まないと実メモリー領域は確保されないので、ここでは 1 を代入しています。

ソースコード 3

```
#include <sys/mman.h>
main() {
  char *mm;
  int i;
  mm = (char *)mmap(0, 1024*1024*1024, (PROT_READ | PRO
T_WRITE),
MAP_ANONYMOUS|MAP_PRIVATE, -1, 0);
  for(i=0;i<1024*1024*1024;i++){
    mm[i]=1;
  }
}
```

リスト 2：ソースコード 3 の実行結果

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT STA
RT   TIME COMMAND
hironobu 15369 99.1  6.8 1052736 1048948 pts/11 S+  13:
18   0:08 ./a.out
```

mmap は基本的にファイルをメモリー領域にマップして、メモリー空間もファイルも単一のアクセス方法、つまりメモリー空間と同じアクセスで扱えるようにするものです。このメカニズムのことを単一レベル記憶（Single Level Storage）といいます。mmap が現れるまでは、単一レベル記憶は、IBM のミニコン OS ぐらいにしか用意されていなかったので、IBM 独自のアイデア、あるいは機能かと思われるかもしれませんが、アイデアと実装はとても古く、UNIX より以前に作られた（作ろうとした）OS である Multics でも使われました。

UNIX の mmap は、カーネギーメロン大学で作った OS である Mach のメモリー管理を経由して 4.3BSD で入ってきました。

mmap は MAP_ANONYMOUS オプションを使うとファイルがなくてもメモリー領域を返してくれます（MAP_PRIVATE ですので自分以外はこの領域へアクセスすることはできません）。この時のメモリー領域は仮想記憶空間から、そのまま割り当てられます。この時、ファイルディスクリプタを（Linux では）「-1」とします。

これである大きさのメモリー空間を mmap で確保し、ユーザーライブラリである malloc で、データを細分化、あるいは使いまわしをする、ということになります。mmap は POSIX.1-2001 で定義されているので、POSIX 互換性である OS ならば問題はありません。あとは malloc の実装の問題になります。

アクセスによるデータの破壊や、セキュリティだけではなく、その他にも並列処理やスケーラビリティ、あるいはデバッキングの簡便さなど、色々な利点をもった malloc 置き換えのためのアロケーションのライブラリが出てきます。Google はツールとして tcmalloc を公開していますし、FreeBSD は jemalloc を用意しています。Mozilla プロジェクトではシステム標準ライブラリの malloc を使わずに、Mozilla Firefox のソースツリーの中に jemalloc を用意し、それを使っています。

他にもセキュリティ強化された secmalloc や smalloc、OpenBSD プロジェクトの Otto Moerbeek 氏が作った通称 otto malloc など多数の malloc 実装が現れます。ちなみに otto malloc は OpenBSD プロジェクトでは単に malloc、あるいは malloc for OpenBSD のような呼び方をしています。

OpenBSD の otto malloc は、アロケーションするメモリー空間のサイズがページより大きい場合は mmap で直接アロケーションします。OpenBSD の mmap はランダムなアドレスに割り当てられる実装なので他のアドレスから、次に割り当てられたアドレスを推測することができません。

mmap で生成した個々のメモリー空間は基本的に独立であり、書き出しが領域からはみ出してアクセスしていくと、セグメンテーション・フォルトを起こしてプログラム自体が不正終了することになります。

メモリー空間のサイズがページサイズ以下の小さな場合のみ、これまでのように切り出して使いますが、なるべくランダムな領域を取ろうとするので、これまでの問題を最小限に留めようとしています。

このような新しい試みがなされたmallocが徐々に増えました。ですが古典的な、というよりスタンダードな実装ともいえるglibcのmallocでは、やっぱり、ユーザーがどこでもアクセスできてしまう（そして、いくらでも関係のないデータを壊してしまう）という昔からの問題が依然として持ち越されたままです。

そしてプログラマーは、あまりにも増えたmalloc実装のバリエーションに頭を悩ませるのでした。

Heartbleedとメモリーアロケーション

最近話題のHeartbleedです。TLS/SSLの実装の1つであるOpenSSLにバグがあって、秘密にしていなければならない情報がダダ漏れになるアレです。

TLSにはHeartbeatという拡張機能があって、相手が生きているか死んでいるかを確認する、またはネットワーク接続を継続させる、一種の死活監視を行います。HeartbeatRequestのパケットを投げると、その中身をコピーしたHeartbeatResponseのパケットを戻してくれます。

HeartbeatRequestを送った側はそれを受け取り、自分の投げたものと同じかどうかを確認します。

HeartbeatRequestを投げるのがクライアント、HeartbeatRequestを受け取り、HeartbeatResponseを送るのがサーバー、としてモデル化するとこんな流れ（図4）になります。

図4：処理のおおよその流れ：

1）HeartbeatRequest送信
●データの長さは本当は128Bなのに64KBと偽の情報を送る。
●データバッファの実際の内容は128B（ここでは仮の値）

2）HeartbeatRequest受信
●データの長さは64KBだと通知されているので、本当に64KBだと思っている。
●データバッファを128Bだけ読み込みそのサイズのメモリーアロケーションを行い、そこにコピーしておく。

3）HeartbeatResponse送信の準備
●データの長さに関する情報を見て64KB分を確保する。
●128Bだけメモリアロケーションした所から64KB分コピーする。

4）HeartbeatResponse送信
●128Bだけ本物で、64KBから128Bを引いた分は本来アクセスしてはいけない秘密情報

5）HeartbeatResponse受信
●（何かはわからないけど）本来秘密である情報を得る。

これをモデルにしてプログラムを書くと、こんな感じになります。変数*a*がHeartbeatRequest、変数*b*がHeartbeatResponseだと思ってください。HeartbeatResponseには本来見えないはずの秘密の情報がダダ漏れです。

ソースコード4

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
void allocdata (void)
{
  char *d[3];
  /* 秘密のデータをmallocのエリアに書いておく */
  d[0] = (char *) malloc (8);
  strncpy (d[0], "abcdef", 7);
  d[1] = (char *) malloc (8);
  strncpy (d[1], "ghijkl", 7);
  d[2] = (char *) malloc (8);
  strncpy (d[2], "mnopqr", 7);
}
void main (void)
{
  struct { int len; char *buf;} a,b;
  int i;
  a.len=8096 ; /* 長さ8KBにみせかけて */
  a.buf = (char *)malloc(3); /* 本当の長さは3Bのみ */
  allocdata (); /* 秘密のデータ生成中 */
  b.len=a.len; /* 嘘の長さをコピー */
  b.buf=(char *)malloc(b.len); /* エリアが8KBになってしまっている */
  memcpy (b.buf, a.buf, b.len); /* バッファ オーバーリード */
  for (i = 0; i < 128; i++){ /* 先頭128Bのみダンプ */
    putchar (b.buf[i]);
  }
}
```

リスト3：ソースコード4の実行結果

```
% cc sample.c
% ./a.out | cat -v
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@!
^@^@^@^@^@^@abcdef^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
@!
^@^@^@^@^@^@ghijkl^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
@!
^@^@^@^@^@^@mnopqr^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
@M-1^_^@^@^@^@^@
```

OpenSSLはデバッグのためにmallocに一皮かぶせています。デフォルトでは、標準ライブラリのmalloc、realloc、freeを使っています。プログラム中では、CRYPTO_malloc_init()というマクロをCRYPTO_set_mem_functions(malloc, realloc, free)に展開し、このインタフェースを使ってmalloc、realloc、freeの関数をデフォルトで使うメモリーアロケーションの関数にセットしています。これが各々

CRYPTO_malloc、CRYPTO_realloc、CRYPTO_free という関数で包まれて（ラップされて）います。

実際にプログラム中でメモリーアロケーションをしているのは OPENSSL_malloc() というマクロで、これも CRYPTO_malloc() へ展開しているだけです。

これらのコードが含まれる src/crypto/mem.c や mem_dbg.c をみると、山ほどデバッグ用のデータダンプのためのコードが組み込まれています。

メモリーをフリーする時は OPENSSL_free() ですが、これも free を呼び出しているだけです。このエリアをクリアにはしていません。つまり、HeartbeatResponse で戻されている秘密情報の中には、すでにアクティブではない、free されている (が内容は必ずしも消されていない)、といった、かなり古い秘密情報まで入っている可能性があります。

Heartbleed Bug のために OpenSSL のソースコードを読んでいて、特にメモリーアロケーションまわりのやたらと汚いデバッグ用ダンプコードや、あるいは、システムのデフォルトの malloc をそのまま使うコードだということがわからないほど関数ポインターの嵐で、何がどこでセットされているのか、かなりじっくり読まないと見過ごします。このソースコードの見通しの悪さはこれから先もトラブルを起こす原因となることでしょう。

プログラムで、既存の malloc や free や realloc を使わず、別の malloc を使いたい時は、CRYPTO_set_mem_functions() を使えばセットアップできると Makefile には書いています。

図 5 : CRYPTO_set_mem_functions() を使えば

```
# MFUNC - Make all Malloc/Free/Realloc calls call
#       CRYPTO_malloc/CRYPTO_free/CRYPTO_realloc which
can be setup to
#       call application defined callbacks via CRYPTO_
set_mem_functions()
```

ですが、これを理解できている人は世界中でも何人もいないでしょう。

malloc について考えるのは人生で三度目

Heartbleed の問題点を追っかけていて、すぐに、あーこれは、と理解できたのは、実は 80 年代に UNIX プログラマーに成り立ての頃、この手のバッファー・オーバー・ライト（リード）のバグに散々悩まされたからです。この時が一回目です。当時はデバッグ文を挟むしか手だてがありませんでした。ちょうど今の OpenSSL の mem.c にあるようなコードです。2014 年に大騒ぎした話題も元をたどれば 80 年代から変わらないのか、と不思議な感覚になります。

動的なメモリーアロケーションに劇的な変化を生んだのは mmap 実装の登場でしょう。MAP_ANONYMOUS を使えば仮想記憶側からメモリー空間を持ってこれますが、やろうと思えばプログラムレベルでローカルな仮想記憶モドキを作ってしまえるほどの柔軟性です。そして mmap という標準化したシステムコールを利用できることになって、色々な malloc の実装が進みました。malloc と free の間を行き来する細切れ（チャンク）にされたリストをどう効率よく管理するか、そしてデータ侵害・セキュリティ侵害からの保護、あるいはデバッグライブラリの充実など、いったいどれを選べばいいのか、ほとほと困り果て、の、二回目です。色々な資料を読んでいって、思ったことは「信ずるものは救われる」です。ただし glibc の malloc では、もう機能不足のように思えるのは確かです。

そして、今回が三回目。バッファ・オーバー・リードなんですが、それは malloc で使っているメモリー空間をガリガリと読んでしまう点が、本質的な問題です。どんな風にメモリーアロケーションがなされているのかについて今回は論じませんが、先ほど言ったように OpenSSL はメモリーアロケーションのデバッグのために大量のデバッグ用コードを入れています。ですから、（どんな秘密がアロケーションされているのか）実際の影響度を図るのには、実は十分なコードが用意されています。

というわけで、真面目に malloc のことを考えたのは、人生で三度目。もうこれ以上、malloc を振り返ることのない人生を歩めるよう、日々精進していくことを心に決めて今回は筆を措きたいと思います。

※1 本気に取らないように。
※2 bss セグメントもデータ領域ですが、図1 ではそこに含めています。

いま私たちは何を学ぶべきか

人間とコンピュータの可能性

第16回

生体システムはあり合わせ材料の再利用で進化した

written by 慶應義塾大学 名誉教授 **大岩 元**

今回は、我々の身体を生涯通じて思い通りに動かすための健康法を紹介しつつ、生体の持つ合理的な動作について考えてみたい。

自律分散協調系としての筋肉組織

100万年を越える人類の歴史の中で、わずか200年前に始まった産業革命による工業化によって、人間は労働から解放されて筋肉を使う時間と量が圧倒的に少なくなってしまった。情報化はこの傾向をさらに助長しているが、使わなくなった筋肉は劣化し、その事が健康に大きな影響を与え出している。

要らなくなった筋肉を使わなくても困らないように思えるが、そうではない。筋肉組織は人工物と違って、目的に応じて設計されたものではなく、既存の筋肉を新たな目的のために利用することを通じて、複雑な筋肉組織を形成してきたのである。

例えば、首の筋肉が弱まると、噛めなくなる、頭がはっきりしない、唾液が出ずらい、聴力が落ちる、頚椎がずれたり減ったりする、猫背になる、ひざや腰が痛んだりする、といった症状が現れる。首を支える筋肉は、その他の目的にも使われており、劣化すると、上記のように関連する機能も同時に劣化してしまうのである。

「きくち体操」

菊池和子氏が開発した「きくち体操」では、「筋肉を鍛える」のではなく、「筋肉を育てる」という新しい考え方の体操を行っていて、女性を中心に広く支持を集めている。私がこの体操を自分の生活に取り入れたのは、菊池氏の著書『はじめての「きくち体操」』（講談社プラスアルファ新書）に紹介されていた事例に感激したからである。

ある時、菊池さんの体操教室に70代半ばの女性が訪れた。「腰が曲って足が上げられなくなり、バスにも乗れなくなった。体操をしてバスに乗れるようにしたい」と相談を持ちかけられたという。その女性は以来14年間、毎日熱心に体操を続けた結果、開脚で上体を前に倒して床にピタリとつけられるまで、体の柔軟性を取り戻すことができたそうである。彼女が90歳近くで亡くなった時、担当された医師が「こんな見事な亡くなり方をされた方は見たことがありません。是非解剖させて欲しい」と希望されて、解剖を行うことになった。その結果、この医師は「すべての筋肉、骨、臓器をここまで使い切った方は見たことがありません」と驚いたそうである。私も是非このような状態で逝きたいと思う。

「きくち体操」では体操を行なう時に、筋肉と脳の関係を意識することが重視される。学校で教わる体育の時間では、前屈して手が地面につくかどうかというような、動作の結果として得られる「形」だけを問題にすることが多い。しかし、これでは筋肉は「育たない」のである。

両脚を前に出して座る「長座」でも、筋肉を普段から育てていない人は、後ろに手をつかないと座れない。背中に力を入れて上体を前に持ってこようとしても、うまくいかない。「きくち体操」では、まず太ももに力を入れて、膝を床に押しつけ、お腹を引いて上体を起こすように指導する。この時、ふくらはぎや膝の裏側がどのような感じがするか、に気を配って、長座に必要な筋肉を動

かすように意識させるのである。うまく座れたかどうかを問題にするのではない。筋肉を意識して動かすことを毎日続けることで、筋肉は育っていくのである。

太ももの筋肉は、前はお腹、後ろはお尻につながっていて、上体を支えている。太ももが弱くなると、お尻の筋肉も弱って、お尻が横にずり落ちて股関節や腰も弱りだすのである。

協調作業が合理的な仕事を可能にする

タイピングを行なう場合、キーを叩くのは指であるが、熟練タイピストは指をあまり使わない。キーの上に指の先をもっていく作業は、指を動かすのではなく、腕全体を動かすことで、指の負担を軽減している。叩く作業自体も、腕の重さを使って筋肉はできるだけ使わないようにする。

指は、キーを叩くだけでなく、多様な使い方をする。同様に、腕も色々な働きをする。このように、多様な働きをする指と腕に協調作業させることで、それぞれの負担は小さくても、大きな仕事をできるようになる。

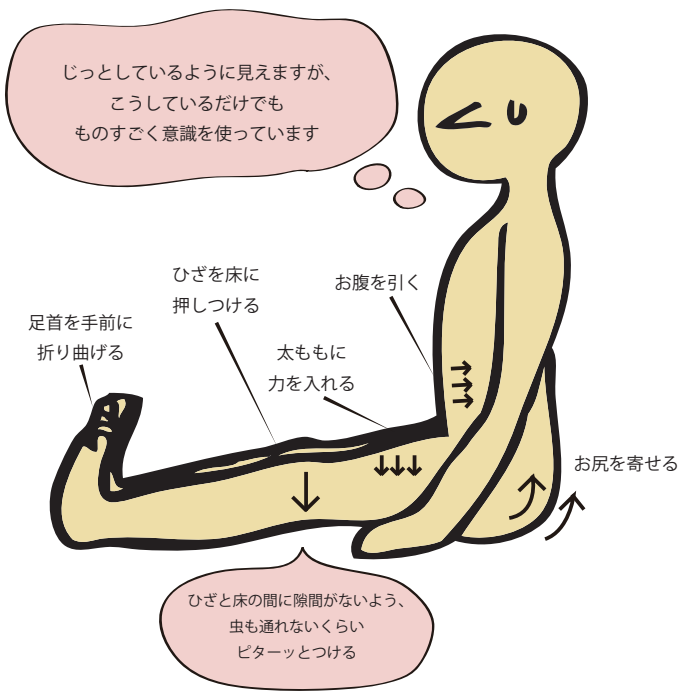
手、指、腕といった、人間の機能モジュールは、魚のひれから足へと進化したものが、直立歩行にともなって、巨大化した脳とともに生まれたものである。手と腕の協調動作が脳によって飛躍的に強力に発展したことで、ことばの発達をうながし、現生する人類を生み出したように思われる。

生体の形成とユニケージ技法

コンピュータ上に作る情報システムは、実現したい目的に到達するべく合理的な設計を伴って作られる。目的を実現するのに必要となる機能モジュールを洗い出し、そこへの情報の出し入れはできるだけ小さくして、定められた出入り口だけを通じてモジュール間で情報交換をするようにして、巨大なシステムを構築する方法が一般的である。

これに対して情報システムよりはるかに複雑で巨大な生体システムは、全く違う方式で構成されている。脳もその例外でない。ゲアリー・マーカス『脳はあり合せの材料から生まれた』（早川書房）は、題名自体がその事を示しているが、原題は "Kluge: The Haphazard Construction of the Human Mind" である。kluge という用語を、私がこの本で初めて知ったが、「エレガントにはほど遠く、無様であるのにもかかわらず、驚くほど効果的な問題解決法」を表わす技術用語であるという。生体は、人工物とは全く違う原理で作られたにもかかわらず、はるかに複雑で強力な仕組みを何憶年という長い時間をかけて、形成したのである。

ユニケージは、生体的なシステム構築法であるように見える。商品管理のために、1000 以上のコマンドを作り、結局数十個のコマンドだけが使われているそうであるが、進化の歴史で淘汰された種のことが思い起こされる。



菊池和子著
『はじめての「きくち体操」』
（講談社プラスアルファ新書）

HANDS LAB

ユニケージ®エンジニア数 最大級!

ハンズラボはユニケージ®開発手法に特化したITソリューション企業です。東急ハンズの営業システムを刷新したノウハウを駆使し、小売業における「オーダーメイド」のシステム開発を行います。

中途採用 大募集!詳しくはHPで!

› http://www.hands-lab.com/

「ユニケージ®」は有限会社ユニバーサル・シェル・プログラミング研究所の登録商標です。

これであなたもユニケージエンジニア!

ユニケージ開発手法教育講座

「ユニケージ開発手法教育講座」は、ユニケージ開発手法におけるデータ管理の方法や、オリジナルコマンドの使用方法などをハンズオン形式で具体的に学べる講座です。UNIX の基礎からユニケージ開発手法による開発プロジェクトの進め方まで、ユニケージエンジニアとしてのトータルスキルを習得できます。

http://www.usp-lab.com/LECTURE/CGI/LECTURE.CGI

- K1 コマンド学習編
- K2 シェルスクリプト学習編（帳票・バッチ処理）
- K3 シェルスクリプト学習編（ウェブアプリケーション処理）
- K4 ユニケージアーキテクチャ編
- K5 ユニケージ開発環境セットアップ編
- K6 システム運用・管理編
- K7 プロジェクトマネジメント・人材育成編
- KSQL SQL エンジニアのためのユニケージ活用編
- KSTAT ユニケージにおける統計コマンド編

続々と新講座も充実中!

UNIX 初心者のための講座などもご用意しています。

TechLION For Independent Engineers

技術の草原で百獣の王を目指す
エンジニアたちの新感覚トークライブ!

http://techlion.jp/

上記のサービス内容は 2014 年 7 月現在のものです。
最新の情報は弊社ホームページにてご確認ください。

ISBN 978-4-9048-0710-1
C3455 ¥500

9784904807101

USP 研究所
定価（本体 500 円+税）

1923455005002