Sambaのソースを読もう

source code readingへのおさそい

日本Sambaユーザ会/日本電気

太田俊哉

日本samba ユーザ会

http://www.samba.gr.jp

今日の内容

- nmbdのソースを読みます
  - 本当に読みます
  - 機能の紹介、ノウハウのような話ではありません
- 読むのはごく一部、さわりです。
  - ソースリーディングの雰囲気をつかんで頂ければ幸いです。
- 目がよくないとつらいです

ソースリーディングって大事

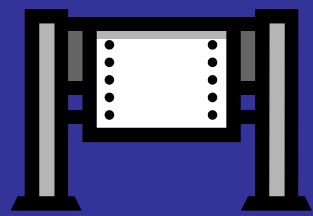
- オープンソースはソースが決め手
  - そこに全部書いてある
  - 中を見られるのがオープンソースの利点
  - ソースを使え、ルーク！
    (http://itpro.nikkeibp.co.jp/free/ITPro/OPINION/20040203/139269/)
  - 規模が大きいものは読むのが大変
    - ~~チーズ~~仕様書はどこへ消えた?
      - そもそもあるんだっけ?
    - 伽藍方式だったらあるはず....
      - 最終更新日はいつだ?

読む理由

- 動機(目的)
  - プログラムがおかしいので調べたい
  - もうちょっと機能追加が出来ないか調べたい
  - すげー機能があるので方式を調べたい
  - 上司の指示(?)
  - その他
- いろいろあるけどとりあえず読んでみよう
  - 今回の趣旨

手段を考える(1)

- ラインプリンタで全ソースプリントアウト
  - いつの時代だ?
    - 資源浪費、肉体労働
  - 全体を見渡しやすい



- more
  - どうやって前に戻る? → less
- grep hogehoge ソース
  - その行しか見付からない。効率悪い。
  - 小さなものなら当たりをつけられる

手段を考える(2)

- ed,edlin,ノートパッド、vi,emacs
  - 検索機能が使える。前後も分かる。
  - 大きくて、複数のファイルがあるものは大変
- タグファイル+エディタ
  - ctags (see man ctags)
- global
  - webベースのctagsか
  - 検索も出来るし、自在に手繰れる
  - 今風

GNU GLOBAL(1)

- タグ生成ソフト。C言語だけではなく、C++、Yacc、Java、PHP4に対応
- 巨大ソースに対応(#ifdefとか)
- 入手
  - http://tamacom.com/global-j.html
  - portsもRPMもあり(portsはちょっと古い)
  - OS依存性少、コンパイル等は比較的楽

GNU GLOBAL(2)

- 設定方法
  - portsかRPMで放り込んでしまえばおしまい
  - 対象のOS向けのPRMがない場合には、他のものを持ってきてちょちょいといじれば大丈夫。
  - 後はweb サーバの設定(この辺は省略)
- タグファイル作成
  - gtags -v
  - htags -vgsn などなど
    - コンテンツは静的に作成

実際に読んでみる

- 今回の対象
  - nmbd
- 軽いものは他にもあるが
  - Windowsネットワークの動作を知りたいというのが理由
    - winsの登録の動作などなど
- ここから先は細かな字が多いので、見づらい場面が多いです

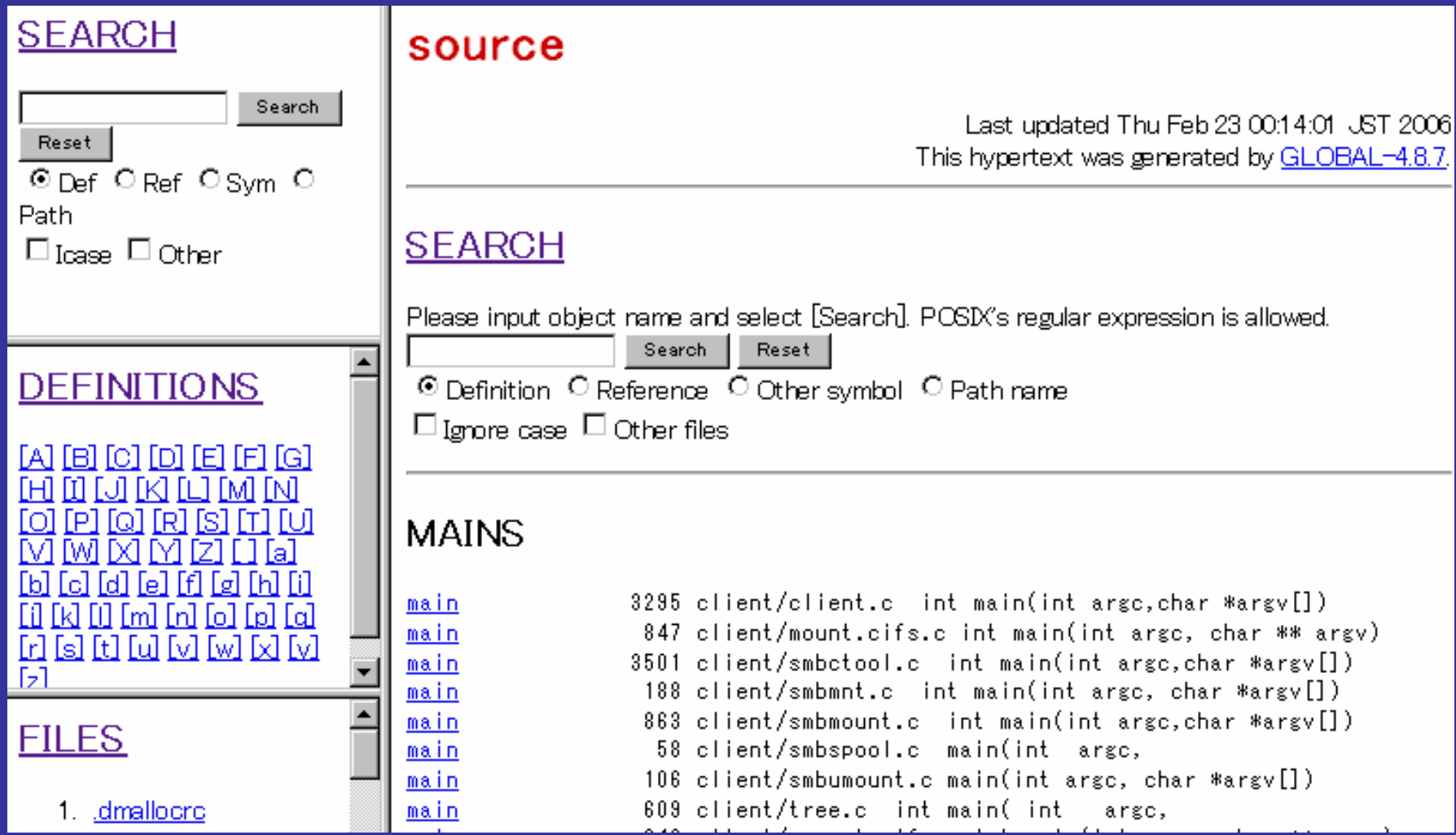
戦略

- 当たりをつける
  - wins関連なのでnmbdに関連したところに違いない
  - キーワード wins
  - cd source/nmbd
  - ls *wins*
    - nmbd_winsproxy.c と nmbd_winsserver.c が見付かる
- globalでのぞく
- 基礎知識は必要
  - C言語の基礎とか、そのソフト固有の機能とか

そのほかの方法

- ログメッセージから辿る
  - log.nmbdとかlog.smbdに出るログメッセージ
  - ソース中に文字列として定義
  - それを grep で探す
  - エラーに関する情報を探すときに便利
- main()から辿る
  - おおざっぱな流れを知るときに便利
  - あとは引数の動作とか

GLOBALの起動画面



ツアーの開始

- 右下のペインから　28. nmbd/ をクリック
- 変わった画面で 26. nmbd_winsserver.c
- 右側の画面にnmbd_winsserver.c が表示
  - 上に関数一覧、その下にソースコード本体
  - 各関数、定義名は全てリンクされている

initiate_wins()

- どこから見ていくか
  - このソースにはmain()がない
    - メインプログラムではないから→main()から辿る戦略は使えない
  - 最初に呼ばれそうなところを探してみる
    - initiate_wins がよさそう
    - 関数一覧の initiate_wins をクリック
    - ちゃんとしているソースならば、変数名等に意味があるように書いてあるので、細かく見なくてもだいたいはわかる....はず

```
235 BOOL initialise_wins(void)
236 {
237         time_t time_now = time(NULL);
238         XFILE *fp;
239         pstring line;
240
241         if(!lp_we_are_a_wins_server()) {
242                 return True;
243         }
244
245         add_samba_names_to_subnet(wins_server_subnet);
246
247         if((fp = x_fopen(lock_path(WINS_LIST),O_RDONLY,0)) == NULL) {
248                 DEBUG(2,("initialise_wins: Can't open wins database file %s.
249                         WINS_LIST, strerror(errno) ));
250                 return True;
251         }
252
253         while (!x_feof(fp)) {
254                 pstring name_str, ip_str, ttl_str, nb_flags_str;
255                 unsigned int num_ips;
256                 pstring name;
257                 struct in_addr *ip_list;
258                 int type = 0;
259                 int nb_flags;
260                 int ttl;
261                 const char *ptr;
262                 char *p;
263                 BOOL got_token;
264                 BOOL was_ip;
265                 int i;
266                 unsigned int hash;
267                 int version;
268
```

こいつはなんだ?

調べてみよう

(クリックする)

```
WINS_LIST          25 nmbd/nmbd_winsserver.c #define WINS_LIST "wins.dat"
WINS_LIST          33 wrepld/process.c #define WINS_LIST "wins.tdb"
```

nmbd_winsserver.c に定義されていた

```
 25 #define WINS_LIST "wins.dat"
 26 #define WINS_VERSION 1
 27
 28 /*****************************************************************************
 29  Change the wins owner address in the record.
 30 *****************************************************************************/
```

wins.dat のファイル名の定義であることがわかる
(こんなところにあるわけだ)

続いて、initialize_wins()の続きを読んでみる

```
253         while (!x_feof(fp)) {
254                 pstring name_str, ip_str, ttl_str, nb_flags_str;
255                 unsigned int num_ips;
256                 pstring name;
257                 struct in_addr *ip_list;
258                 int type = 0;
259                 int nb_flags;
260                 int ttl;
261                 const char *ptr;
262                 char *p;
263                 BOOL got_token;
264                 BOOL was_ip;
265                 int i;
266                 unsigned int hash;
267                 int version;
```

wins.dat を読むループがあるわけね

```
269                 /* Read a line from the wins.dat file. Strips whitespace
270                         from the beginning and end of the line.  */
271                 if (!fgets_slash(line,sizeof(pstring),fp))
272                         continue;
```

ふむふむ、単に行を読むだけでなく、加工しながら読んでいるわけね。
(もちろんここでgets_slashを読みに行ってもよし)
この後wins.datのバージョンチェックをしているところは省略。

```
289                 /*
290                  * Now we handle multiple IP addresses per name we need
291                  * to iterate over the line twice.  The first time to
292                  * determine how many IP addresses there are, the second
293                  * time to actually parse them into the ip_list array.
294                  */
```

え"、2回読むの?まあwins.dat は1回しか読まないから多少効率悪くてもいいわけだけど。

```
if (!next_token(&ptr,name_str,NULL,sizeof(name_str))) {
        DEBUG(0,("initialise_wins: Failed to parse name when parsing line %s\n", line ));
        continue;
}

if (!next_token(&ptr,ttl_str,NULL,sizeof(ttl_str))) {
        DEBUG(0,("initialise_wins: Failed to parse time to live when parsing line %s\n", line ))
        continue;
}

/*
 * Determine the number of IP addresses per line.
 */
num_ips = 0;
do {
        got_token = next_token(&ptr,ip_str,NULL,sizeof(ip_str));
        was_ip = False;
```

NETBIOS名読んで、TTL読んで、それからIPアドレス読んでいるわけね。
(next_tokenは読みにってもいいけど、名前から動作を推測してもいい)

```
336                 /* Reset and re-parse the line. */
337                 ptr = line;
338                 next_token(&ptr,name_str,NULL,sizeof(name_str));
339                 next_token(&ptr,ttl_str,NULL,sizeof(ttl_str));
340                 for(i = 0; i < num_ips; i++) {
341                         next_token(&ptr, ip_str, NULL, sizeof(ip_str));
342                         ip_list[i] = *interpret_addr2(ip_str);
343                 }
344                 next_token(&ptr,nb_flags_str,NULL, sizeof(nb_flags_str));
345
```

二回目の構文解析。name_str**に**NETBIOS**名が、**ttl_str**に**TTL**が、**ip_list[]**に**IP**アドレスが、**nb_flags_str**に**NETBIOS**フラグが入る。**

```
346                 /*
347                  * Deal with SELF or REGISTER name encoding. Default is REGISTER
348                  * for compatibility with old nmbds.
349                  */
350
351                 if(nb_flags_str[strlen(nb_flags_str)-1] == 'S') {
352                         DEBUG(5,("initialise_wins: Ignoring SELF name %s¥n", line));
353                         SAFE_FREE(ip_list);
354                         continue;
355                 }
356
357                 if(nb_flags_str[strlen(nb_flags_str)-1] == 'R') {
358                         nb_flags_str[strlen(nb_flags_str)-1] = '¥0';
359                 }
```

NETBIOSフラグの末尾が"S"のものは処理されないのね。"S"が何を意味するかは後にしておく。
(Samba**の全てにも説明ないので**)
この後は、name**に名前そのものが、**type**に**NETBIOS**タイプがデコードされて入る。**

```
373                 /* add all entries that have 60 seconds or more to live */
374                 if ((ttl - 60) > time_now || ttl == PERMANENT_TTL) {
375                         if(ttl != PERMANENT_TTL) {
376                                 ttl -= time_now;
377                         }
```

60秒未満の場合には別処理になりそうだ(下の方に問題ありTTLで処理しないコードがある)。
TTL=0の場合はPERMANENT_TTLになる(定数をクリックすると 0 が定義されている)

```
382                         (void)add_name_to_subnet( wins_server_subnet, name, type, nb_flags,
383                                                   ttl, REGISTER_NAME, num_ips, ip_list );
```

どうやらここでwins.datのデータを内部データ構造に登録しているようだ。
ここから先は別途見るとして、先にコードの終わりまで見ておこう。

```
385                         DEBUG(4, ("initialise_wins: not adding name (ttl problem) "
386                                 "%s#%02x ttl = %d first IP %s flags = %2x\n",
387                                 name, type, ttl, inet_ntoa(ip_list[0]), nb_flags));
388                 }
389
390                 SAFE_FREE(ip_list);
391         }
392
393         x_fclose(fp);
394         return True;
```

TTLが現在の時間より60秒未満の時にはエラー処理。複数のipアドレスを保持する変数は使い捨て。

ここまでで分かったこと

- wins.dat は固定定義
- 各行は2回構文解析される
- NETBIOSフラグの"S"と"R"に意味あり
- TTLは現在時刻と60秒離れていないとだめ
- ポイントポイントでDEBUGメッセージ出力が入っている

ちょっと考えてみよう

ソースの二回構文解析

- 一回での解析はやってやれないことはない
- 解析した結果が、IPアドレスでなければNETBIOSフラグだから
- hackすべきか
  - エレガントなコード/エレファントなコード
  - 時間効率/空間効率
  - 木を見るか、森を見るか
  - 怠惰、短気、傲慢
  - いじるな危険

基礎知識は必要

- WINSの機能を見る
  - WINSの基礎的な知識は必要　たとえば
    - NetBIOS名の最後につくのはタイプ
    - その名前がなんの機能をサポートするか
      - 00:Workstationサービス
      - 03:Messengerサービス
      - 20:Serveサービス　　などなど
  - 「アンドキュメンテッドMicrosoftネットワーク」と「Sambaのすべて」は必須
    - ガイドブックですね
- 次いってみましょう

add_name_to_subnet()

- nmbd_namelistdb.cの中
  - ファイルをまたがってジャンプ
  - １００行弱
  - 冒頭は **struct** name_record

```
180 struct name_record *add_name_to_subnet( struct subnet_record *subrec,
181                                         const char          *name,
182                                         int                  type,
183                                         uint16               nb_flags,
184                                         int                  ttl,
185                                         enum name_source     source,
186                                         int                  num_ips,
187                                         struct in_addr      *iplist)
```

  - でも引用しているところは
    (**void**)add_name_to_subnet(.....
    となっている。→戻り値は使わない

```
192         namerec = SMB_MALLOC_P(struct name_record);
193         if( NULL == namerec ) {
194                 DEBUG( 0, ( "add_name_to_subnet: malloc fail.¥n" ) );
195                 return( NULL );
196         }
```

何これ？

確かに記述量は減るけど..

```
SMB_MALLOC_P      346 include/smb_macros.h #define SMB_MALLOC_P(type) (type *)malloc_(sizeof(type))
SMB_MALLOC_P      364 include/smb_macros.h #define SMB_MALLOC_P(type) (type *)malloc(sizeof(type))
```

所で namerec は何?

```
219 struct name_record {
220         ubi_trNode              node[1];
221         struct subnet_record *subnet;
222         struct nmb_name         name;    /* The netbios name. */
223         struct nmb_data         data;    /* The netbios data. */
224 };
```

nameserv.hの中
1つの変数と3つの構造体

```
825 #define ubi_trNode    ubi_btNode
```

```
352 typedef struct ubi_btNodeStruct {
353   struct ubi_btNodeStruct *Link[ 3 ];
354   char                    gender;
355   char                    balance;
356   } ubi_btNode;
```

っと思いきや、typedefだった
構造体をリンクするためのポインタ構造体

```
1604 struct nmb_name {
1605         nstring      name;
1606         char         scope[64];
1607         unsigned int name_type;
1608 };
```

名前をしまう構造体と
データをしまう構造体

```
202 struct nmb_data {
203         uint16 nb_flags;          /* Netbios flags. */
204         int num_ips;              /* Number of ip entries. */
205         struct in_addr *ip;       /* The ip list for this name. */
206
207         enum name_source source; /* Where the name came from. */
208
209         time_t death_time; /* The time the record must be removed (do not remove if 0). */
210         time_t refresh_time; /* The time the record should be refreshed. */
211
212         SMB_BIG_UINT id;                /* unique id */
213         struct in_addr wins_ip; /* the adress of the wins server this record comes from */
214
215         int wins_flags;          /* similar to the netbios flags but different ! */
216 };
```

```
198         memset( (char *)namerec, '¥0', sizeof(*namerec) );
199         namerec->data.ip = SMB_MALLOC_ARRAY( struct in_addr, num_ips );
200         if( NULL == namerec->data.ip ) {
201                 DEBUG( 0, ( "add_name_to_subnet: malloc fail when creating ip_flgs.¥n" ) );
202                 ZERO_STRUCTP(namerec);
203                 SAFE_FREE(namerec);
204                 return NULL;
205         }
206
207         namerec->subnet = subrec;
208
209         make_nmb_name(&namerec->name, name, type);
210         upcase_name(&namerec->name, NULL );
211
212         /* Enter the name as active. */
213         namerec->data.nb_flags = nb_flags | NB_ACTIVE;
214         namerec->data.wins_flags = WINS_ACTIVE;
215
216         /* If it's our primary name, flag it as so. */
217         if (strequal( my_netbios_names(0), name )) {
218                 namerec->data.nb_flags |= NB_PERM;
219         }
220
221         /* Copy the IPs. */
222         namerec->data.num_ips = num_ips;
223         memcpy( (namerec->data.ip), iplist, num_ips * sizeof(struct in_addr) );
224
225         /* Data source. */
226         namerec->data.source = source;
227
228         /* Setup the death_time and refresh_time. */
229         if (ttl == PERMANENT_TTL) {
230                 namerec->data.death_time = PERMANENT_TTL;
231         } else {
232                 namerec->data.death_time = time_now + ttl;
```

この辺では淡々とデータを構造体に格納している。

make_nmb_nameとかupcase_name とかは関数名が意味を表わしていると考えて詳細は省略。

普通数字を直接使うことは余りやらない。実は、my_netbios_namesはsmb_my_netbios_namesという配列のn番目を返すだけ。で、それは、グローバルな変数(文字列へのポインタの配列)

```
231         } else {
232                 namerec->data.death_time = time_now + ttl;
233         }
234
235         namerec->data.refresh_time = time_now + MIN((ttl/2), MAX_REFRESH_TIME);
236
237         /* Now add the record to the name list. */
238         update_name_in_namelist( subrec, namerec );
239
240         DEBUG( 3, ( "add_name_to_subnet: Added netbios name %s with first IP %s ¥
241 ttl=%d nb_flags=%2x to subnet %s¥n",
242                 nmb_namestr( &namerec->name ),
243                 inet_ntoa( *iplist ),
244                 ttl,
245                 (unsigned int)nb_flags,
246                 subrec->subnet_name ) );
247
248         subrec->namelist_changed = True;
249
250         return(namerec);
251 }
```

リフレッシュ間隔は、ttlの半分又はMAX_REFRESH_TIME(60*20秒)の短い方

名前登録時に出るログがここで出力される。よくみるやつ。

```
nmbd.log.2:  add_name_to_subnet: Added netbios name WORKGROUP<1d> with first IP
192.168.2.22 ttl=0 nb_flags=60 to subnet UNICAST_SUBNET
```

ここでどうやらレコードを登録するらしい

詳細は分からないけど、木に登録するような感じだねえ

```
76 /*****************************************************************************
77  Remove a name from the namelist.
78 *****************************************************************************/
79
80 static void update_name_in_namelist( struct subnet_record *subrec,
81                                      struct name_record    *namerec )
82 {
83         struct name_record *oldrec = NULL;
84
85         ubi_trInsert( subrec->namelist, namerec, &(namerec->name), &oldrec );
86         if( oldrec ) {
87                 SAFE_FREE( oldrec->data.ip );
88                 SAFE_FREE( oldrec );
89         }
90 }
```

ここまでで分かったこと(2)

- namerec構造体にwins.datの情報が格納
- subrec(呼び出し元のwins_server_subnet)にリンクが追加される
- subrecは木構造
- リフレッシュ間隔は20分またはttl/2の短い方
  - ttl/2はWindowsと同じ
- じゃ、逆にwins.datに書くには?
  - よく見ると、wins_write_databaseという関数があるではないか

```
1823 void wins_write_database(BOOL background)
1824 {
1825         struct name_record *namerec;
1826         pstring fname, fnamenew;
1827
1828         XFILE *fp;
1829
1830         if(!lp_we_are_a_wins_server())
1831                 return;
1832
1833         /* We will do the writing in a child process to ensure that the parent doesn't block while this is
1834         if (background) {
1835                 CatchChild();
1836                 if (sys_fork()) {
1837                         return;
1838                 }
1839         }
1840
1841         slprintf(fname,sizeof(fname)-1,"%s/%s", lp_lockdir(), WINS_LIST);
1842         all_string_sub(fname,"//", "/", 0);
1843         slprintf(fnamenew,sizeof(fnamenew)-1,"%s.%u", fname, (unsigned int)sys_getpid());
1844
1845         if((fp = x_fopen(fnamenew,O_WRONLY|O_CREAT,0644)) == NULL) {
1846                 DEBUG(0,("wins_write_database: Can't open %s. Error was %s\n", fnamenew, strerror(errno)));
1847                 if (background) {
1848                         _exit(0);
1849                 }
1850                 return;
1851         }
```

引数が真ならばバックグラウンドで動作するような設定

プロセスIDを付けたファイル名をオープンしている。

```
1857        for( namerec = (struct name_record *)ubi_trFirst( wins_server_subnet->namelist ); namerec; namer
1858                int i;
1859                struct tm *tm;
1860
1861                DEBUGADD(4,("%-19s ", nmb_namestr(&namerec->name) ));
1862
1863                if( namerec->data.death_time != PERMANENT_TTL ) {
1864                        char *ts, *nl;
1865
1866                        tm = localtime(&namerec->data.death_time);
1867                        ts = asctime(tm);
1868                        nl = strrchr( ts, '¥n' );
1869                if( NULL != nl ) {
1870                                *nl = '¥0';
1871                }
1872                        DEBUGADD(4,("TTL = %s  ", ts ));
1873                } else {
1874                        DEBUGADD(4,("TTL = PERMANENT                 "));
1875                }
1876
1877        for (i = 0; i < namerec->data.num_ips; i++) {
1878                        DEBUGADD(4,("%15s ", inet_ntoa(namerec->data.ip[i]) ));
1879        }
1880                DEBUGADD(4,("%2x¥n", namerec->data.nb_flags ));
1881
1882                if( namerec->data.source == REGISTER_NAME ) {
1883                        unstring name;
1884                        pull_ascii_nstring(name, sizeof(name), namerec->name.name);
1885                        x_fprintf(fp, "¥"%s#%02x¥" %d ", name,namerec->name.name_type, /* Ignore scope. */
1886                                (int)namerec->data.death_time);
1887
1888                        for (i = 0; i < namerec->data.num_ips; i++)
1889                                x_fprintf( fp, "%s ", inet_ntoa( namerec->data.ip[i] ) );
1890                        x_fprintf( fp, "%2xR¥n", namerec->data.nb_flags );
1891                }
1892        }
1893
```

上半分でDEBUGメッセージを、下半分でファイルに書き込みを行なっている。

```
1894          x_fclose(fp);
1895          chmod(fnamenew,0644);
1896          unlink(fname);
1897          rename(fnamenew,fname);
1898          if (background) {
1899                  _exit(0);
1900          }
1901  }
1902
1903  #if 0
1904          Until winsrepl is done.
1905  /****************************************************************************
1906   Process a internal Samba message receiving a wins record.
1907  ****************************************************************************/
1908
1909  void nmbd_wins_new_entry(int msg_type, struct process_id src,
1910                           void *buf, size_t len)
1911  {
```

最後の部分
クローズして、
ファイル名を変更して
おしまい。

winsの複製機能を実装するコードがまだできていない。
そのためその下の関数が全部コメントアウト。

書き込むタイミング

- wins_write_databaseで書き込むことは分かった
- でも、いつ書き込むか
  - wins_write_databaseを呼び出す所を探す
  - これは、wins_write_databaseの関数定義部分をクリックすればよい
  - 呼び出されているのは2か所

```
wins_write_database   61 nmbd/nmbd.c            wins_write_database(False);
wins_write_database 1814 nmbd/nmbd_winsserver.c                wins_write_database(True);
```

  - それぞれを確認してみる

```
52 /**************************************************************************** **
53  Handle a SIGTERM in band.
54  **************************************************************************** */
55
56 static void terminate(void)
57 {
58         DEBUG(0,("Got SIGTERM: going down...¥n"));
59
60         /* Write out wins.dat file if samba is a WINS server */
61         wins_write_database(False);
62
63         /* Remove all SELF registered names from WINS */
64         release_wins_names();
65
66         /* Announce all server entries as 0 time-to-live, 0 type. */
67         announce_my_servers_removed();
68
69         /* If there was an async dns child - kill it. */
70         kill_async_dns_child();
71
72         exit(0);
73 }
```

引数は'偽'
この場合、write_wins_database はforegroundで実行される。
terminate処理中だから待ち合わせても問題はないわけだ

ちなみにこのコードはnmbd.c の中

```
1794                                            /* that's not as MS says it should be */
1795                                            namerec->data.wins_flags&=~WINS_STATE_MASK;
1796                                            namerec->data.wins_flags|=WINS_TOMBSTONED;
1797                                            namerec->data.death_time = t + EXTINCTION_TIMEOUT;
1798                                            DEBUG(3,("initiate_wins_processing: tombstoning %s¥n",
1799                                    case WINS_TOMBSTONED:
1800                                            DEBUG(3,("initiate_wins_processing: deleting %s¥n", nmb
1801                                            remove_name_from_namelist( wins_server_subnet, namerec
1802                                            break;
1803                                    case WINS_RELEASED:
1804                                            DEBUG(0,("initiate_wins_processing: %s is in released s
1805 we are not the wins owner !¥n", nmb_namestr(&namerec->name)));
1806                                            break;
1807                                    }
1808                            }
1809
1810                    }
1811            }
1812
1813            if(wins_server_subnet->namelist_changed)
1814                    wins_write_database(True);
1815
1816            wins_server_subnet->namelist_changed = False;
1817 }
1818
1819 /****************************************************************************
1820  Write out the current WINS database.
1821 ****************************************************************************/
```

こちらの方の引数は'真'
この関数は、initiate_wins_processing
namelist_changed**が真の場合、書き込みが行なわれる。**
ということは、名前が変更になったときと考えてよさそうだ。

ツアーの終了

- wins.datに注目して読んでみた
  - 最初に当たりをつけるのに少々苦労するが、あとは比較的すんなり
  - 構造体が何で、何をしているかを把握するのは少々面倒
  - 読んでみると分かるが、多重定義がいくつかある
    - 項目を辿ると単に別の名前に再定義されているもの
    - これがあると読みづらい
  - Samba固有のマクロも多い
    - 敷居が高くなる遠因

本日のまとめ

- Sambaのソースは巨大
- 全部読むのはそれこそ業務でもないと無理
- なにかあったときに、キーワードから探す
- ツールを使って効率化
- 本家svnwebなども使うとよい

- 今後チャンスがあれば、ソースコード読書会をしましょう

ご静聴ありがとうございました

日本 samba ユーザ会

http://www.samba.gr.jp