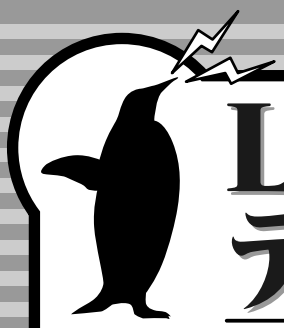

Linux対応無線LANのデバイス・ドライバ詳解

赤松 徹

第6回（最終回） wpa_supplicantが利用可能な無線LANカード

Linux用に開発されたIntel社の無線LANデバイス・ドライバについて6回にわたって解説してきました．最終回である今回は，オープン・ソースを読むときの助けとなるデバッグ・レベルの修正方法と，複数の会社から発売されている無線LANカードをwpa_supplicantがどのように動作させるかという2点に着目して解説します．（筆者）

wpa_supplicantを動作させる手順は，Webサイトにはあまり公開されていませんでした．そこでソース・プログラムを読んでみた結果を以下に記します．ソース・プログラムを読むとき，デバッグ情報を表示するレベルを確認して，レベルの設定を1か所修正することにより，多くのデバッグ情報を表示させることができます．この手法を理解して，ほかのソース・プログラムを読むときにも役立ててください．

複数のLANカードを動作させるためには，上位層と下位層の二つの階層構造を構築します．上位層にあたるwpa_supplicantが下位層にあたる複数種類のデバイス・ドライバを操作するのです．そのための重要事項は，異なるデバイス・ドライバの実操作関数を共通利用できるように構造体で定義することです．上位層から呼び出す関数名は一つの共通関数名だけですが，接続されるデバイスによって異なる実操作関数を実行します．このような階層構造で処理する手法は，カーネル内部でも数多く使用しているので，ぜひとも理解してください．

デバッグ・レベルを変更する

先月号ではwpa_supplicantを設定してアクセス・ポイントにTKIP暗号方式で接続しました．このとき使用した設定ファイルは，動作させることを最優先とした，単純なものです．アプリケーション・ソフトウェアでソース・コードを読まなければならないのは，エラーが発生したときはもちろんのこと，設定ファイルを詳細にチェックしなければならないときがいちばん多いと思います．そこで，設定ファイルを内部で処理する部分を確認しましょう．

リスト1　wpa_printf( )関数

```
221  void wpa_printf(int level, char *fmt, ...)
222  {
223          va_list ap;
224
225          va_start(ap, fmt);
226          if (level >= wpa_debug_level) {
227                  wpa_debug_print_timestamp();
228                  vprintf(fmt, ap);
229                  printf("¥n");
230          }
231          va_end(ap);
232  }
```

具体的に設定ファイルを処理している部分は，config.cファイルにまとめられています．誌面のつごうでwpa_supplicantバージョン0.3.9の設定ファイル処理の全リストは掲載できませんが，設定ファイルを処理する主要部分は，824行目のwpa_config_read()関数です．この関数の引き数で指定した設定ファイル（規定値は/etc/wpa_supplicant.conf）を開いて，設定情報を入手します．

```
824  struct wpa_config * wpa_config_read (
                        const char *config_file)
```

wpa_config_read()関数に分岐してきたとき，840行目のwpa_printf()関数に着目しましょう．このwpa_printf()関数を実行しても，MSG_DEBUGレベルでは何も表示しませんが，このデバッグ情報を表示できれば，ソース・プログラムの内容を確認しやすくなります．

```
840     wpa_printf(MSG_DEBUG,
            "Reading configuration file '%s'",
841                 config_file);
```

そこで，wpa_printf()実関数を検索すると，common.cファイルの221行目以降にあることがわかります（**リスト1**）．226行目のif文でレベルを判定しています．引き数から取得したレベルがwpa_debug_level以上の値ならばデバッグ情報を表示します．

次にwpa_debug_levleに設定される値を確認すると，同じcommon.cファイルの29行目でMSG_INFOと定義しています．

```
29 int wpa_debug_level = MSG_INFO;
```

表示レベルを検索すると，common.hファイルの135行目で定義しています．この表示レベルを参考にして，29行目のMSG_INFOをMSG_DEBUGに変更して，再度make，make installし直します．これで再実行するときには，MSG_DEBUGレベルのデバッグ情報まで表示するようになります．ソース・プログラム内部の確認を完了したのち，具体的に運用を開始するときは，デバッグ・レベルを元のMSG_INFOに再修正することを忘れないでください．

```
132  /* Debugging function - conditional
     printf and hex dump. Driver wrappers can
```


KEYWORD ――無線LAN，デバイス・ドライバ，wpa_supplicant，LANカード，ipw，hostap，TKIP暗号方式

表1　使用する無線LANカードが異なっても共通利用できる定義

.name	"ipw"	"hostap"
.desc	"Intel ipw2100/2200 driver"	"Host AP driver (Intersil Prism2/2.5/3)"
.get_bssid	wpa_driver_ipw_get_bssid	wpa_driver_hostap_get_bssid
.get_ssid	wpa_driver_ipw_get_ssid	wpa_driver_hostap_get_ssid
.set_wpa	wpa_driver_ipw_set_wpa	wpa_driver_hostap_set_wpa
.set_key	wpa_driver_ipw_set_key	wpa_driver_hostap_set_key
.set_countermeasures	wpa_driver_ipw_set_countermeasures	wpa_driver_hostap_set_countermeasures
.set_drop_unencrypted	wpa_driver_ipw_set_drop_unencrypted	wpa_driver_hostap_set_drop_unencrypted
.scan	wpa_driver_ipw_scan	wpa_driver_hostap_scan
.get_scan_results	wpa_driver_ipw_get_scan_results	wpa_driver_hostap_get_scan_results
.deauthenticate	wpa_driver_ipw_deauthenticate	wpa_driver_hostap_deauthenticate
.disassociate	wpa_driver_ipw_disassociate	wpa_driver_hostap_disassociate
.associate	wpa_driver_ipw_associate	wpa_driver_hostap_associate
.set_auth_alg	wpa_driver_ipw_set_auth_alg	wpa_driver_hostap_set_auth_alg
.init	wpa_driver_ipw_init	wpa_driver_hostap_init
.deinit	wpa_driver_ipw_deinit	wpa_driver_hostap_deinit

リスト2　表1を作成する元となるプログラム・リスト（Intel社のipwの例）

```
434  struct wpa_driver_ops wpa_driver_ipw_ops = {
435          .name =                  "ipw",
436          .desc =                  "Intel ipw2100/2200 driver",
437          .get_bssid =             wpa_driver_ipw_get_bssid,
438          .get_ssid =              wpa_driver_ipw_get_ssid,
439          .set_wpa =               wpa_driver_ipw_set_wpa,
440          .set_key =               wpa_driver_ipw_set_key,
441          .set_countermeasures =   wpa_driver_ipw_set_countermeasures,
442          .set_drop_unencrypted =  wpa_driver_ipw_set_drop_unencrypted,
443          .scan =                  wpa_driver_ipw_scan,
444          .get_scan_results =      wpa_driver_ipw_get_scan_results,
445          .deauthenticate =        wpa_driver_ipw_deauthenticate,
446          .disassociate =          wpa_driver_ipw_disassociate,
447          .associate =             wpa_driver_ipw_associate,
448          .set_auth_alg =          wpa_driver_ipw_set_auth_alg,
449          .init =                  wpa_driver_ipw_init,
450          .deinit =                wpa_driver_ipw_deinit,
451  };
```

```
133  *  use these for debugging purposes. */
134
135  enum { MSG_MSGDUMP, MSG_DEBUG, MSG_INFO,
                   MSG_WARNING, MSG_ERROR };
136
```

ここでwpa_supplicant固有の設定内容の解説は省略します．ただし，デバッグ・レベルを変更する方法は，アプリケーション・プログラムによって異なりますが，最初にデバッグ情報を多く出力するように変更してください．この手法を身につけると，オープン・ソースを読むときの力強い助けとなります．

異なる無線LANカードを共通利用する

無線LANカードが異なれば，それを動作させるデバイス・ドライバは当然異なります．しかし，wpa_supplicantが複数の無線LANカードを利用可能にするためには，一つの共通関数名で，具体的に組み込んでいるデバイスの実操作関数を動作させる定義が必要です．この手法を学習しましょう．

最初に，Intel社のipwとIntersil社のhostapの2種類の無線LANカードを利用する定義例を**表1**にまとめました．なぜIntersil社のhostapを取り上げたかというと，この無線LANカードはアクセス・ポイントにする手法が公開されているからです．

wpa_supplicantが指定する共通関数名は，**表1**の左端に書かれているドット(.)で始まる関数名です．デバイス固有の実操作関数名は，中程がIntel社のipwで，右端に書かれているのがIntersil社のhostap用の実操作関数名です．一目でどの無線LANの実操作関数であるかを識別できるように，デバイス固有の実操作関数名は，wpa_driver_デバイス_共通関数名と名付けています．このように関数の名前の付け方を統一しておくと，デバッグ時に役立ちます．

表1に関する定義は，パッケージに含まれるdeiver_デバイス名.cファイルの最後の部分に記述されています．

たとえばIntel社のipwは**リスト2**に示すように，driver_ipw.cファイルに定義されています．等号をはさんで，左辺が共通関数名で，右辺がデバイス固有の実操作関数名です．読者