ＤＴ－９００／９１０／９３０
アプリケーション作成上の注意点

Rev １．００

カシオ計算機株式会社
国内営業統轄部
ＩＴ推進部部

はじめに

この解説書は、アプリケーションの組み方が原因となるトラブルについて、過去の事例と共にその内容と対策を述べたものです。
プログラムを作成する上で、この解説書を十分ご活用下さるよう、お願いいたします。

注意点一覧

１．プログラミング上の注意点

２．特定関数の使用上の注意点

１．プログラミング上の注意点

１－１．配列変数のオーバーフロー

【内容】

指定したサイズ以上の文字を配列変数に代入してしまう、もしくは配列に代入するつもりが、エリア外に代入を行ってしまう。
これによって、配列外のエリアを破壊してしまい、以降の処理で暴走することがある。

【例】

①UB　abc[10];
strcpy(abc, "1234567890");
（上記の場合、文字数１０とＮＵＬＬ文字の合計１１バイトを要するため、配列は１１バイト以上必要）

②bar_str[len-4]='¥0';
（上記の指定で、len の値が４未満であったり、len-4 が宣言した配列の大きさを超えていた場合、配列外のエリアを破壊）

③memcpy(abc, "1234567890", len-4);
（上記同様、len の値が４未満であったり、コピーデータが配列のエリアを超えていた場合、配列外のエリアを破壊）

【対策】

・sizeof 演算子を利用する。
sizeof(配列変数名)で、配列の大きさが得られるので、その値を使用してエリア外への書き込みを行わないようにする。
・変数の値を事前にチェックする。
添え字や代入文字数に変数を使用する場合、事前にその値をチェックする。

【対策例】

```
①strncpy(abc, "1234567890", sizeof(abc)-1);
  abc[sizeof(abc)-1]='¥0';
  ②if((len>=4)&&((len-4)<sizeof(bar_str)))
        bar_str[len-4] = '¥0';
  ③ if ((len>=4)&&((len-4)<sizeof(abc)))
        memcpy(abc, "1234567890", len-4);
```

１－２．アドレス未設定のポインタの使用

【内容】

ポインタ変数にアドレスを与えないまま、処理に使用した。

これによって、不定のアドレスにアクセスしてしまい、以降の処理で暴走することがある。

【例】

```
UB   *abc;
strcpy(abc, "1234567890");
```

（アドレスを設定しない内に処理に使用するため、どこにアクセスするか分からない）

【対策】

対応策の一つとして、変数宣言時に初期化してしまう。

【対策例】

```
UB c[100], *abc = c;
```

１－３．変数の型の不一致

【内容】

引数として与える変数が、正しい型で取られていない。
この場合、関数に正常な値が渡されなかったり、逆に正常な値を取得できないことがあり、以降の処理が正しく動作しない。

【例】

①関数内で値が格納される変数の型が異なる

```
int  rcd, barlen;
OBR_gets(obrbuf, &rcd, &barlen);
```

（関数仕様上、rcd、barlen はそれぞれ UW、UB だが、両方とも int で宣言している。
rcd の場合、int も UW も４バイトなので、結果的には同じになるが、barlen の場合は、UB が１バイトなので、参照すると全く的外れな値が入っている）

②関数に値を渡す変数の型が異なる

```
int  busy_ch,nonbusy_ch;
c_open(COM0,param,buf,buf_l,&tim_out,&del_cod,busy_ch,nonbusy_ch);
```

（関数仕様上、busy_ch、nonbusy_ch はそれぞれ B だが、両方とも int で宣言している。
本来１バイトの引数を渡すべきところに４バイトの変数を与えた場合、自分で設定したつもりの値が設定されない）

【対策】

コンパイル時に“1016 (W) Argument mismatch”の警告エラーが発生するので、その際はプログラムを再度確認する。

１－４．永久ループの使用

【内容】

ある特定の条件が発生しない限り、ループを抜けないようなロジックを組むと、ループから抜け出せず、プログラムがストップした様に見えることがある。

【例】

```
key_fnc_mode(FNC_MODE_SET,FNC_2,&flg_id,&ptn);
while(1){
   flg_sts(&dumy,&ptn,FL_FK_INT_ID);
   if (ptn & FL_FK_INT_FNC1)
         break;
}
```

（Ｆ１が押されるまでループするが、Ｆ１を通知モードにしていないと永久に抜けない）

【対策】

対策として、「永久ループ」の形を取らない組み方や、必ず成立しうる脱出条件を設定する。

【対策例】

```
key_fnc_mode(FNC_MODE_SET, FNC_2, &flg_id, &ptn);
s_timeget(&tim_dat);
tim_dat.sec += 60;
while(1){
    flg_sts(&dumy,&ptn,FL_FK_INT_ID);
    if (ptn & FL_FK_INT_FNC1)
          break;
    s_timeget(&tim_dat2);
    if (tim_dat.sec - tim_dat2.sec <= 60)
          break;      /* break at 60 seconds later */
}
```

（６０秒後に脱出する制御を付加）

１－５．使用不可のＣライブラリ関数の使用

【内容】

動作保証をしていないＳＨ－Ｃの標準Ｃ関数を使用する。

動作の確認をしていないため、どのような現象が発生するかは不明。

【例】

```
fp=fopen(“test.dat”, “r+”);
fgets(buf, DATLEN, fp);
```

（fgets は、動作保証外の関数）

【対策】

使用可能な標準関数については、Ｃライブラリ解説書の「１．２　標準ライブラリ関数」を参照し、ここに記載されていない標準関数は用いないようにする。

１－６．メモリブロック領域の確保／解放の不一致

【内容】

malloc 等の関数で確保した領域の未解放や、確保していないエリアの解放により、メモリエリアの整合性が取れなくなる。

【例】

①同一変数での複数回のエリア確保

```
unsigned char *area;
while(1){
    area = malloc(AREA_SIZE);
        ・
}
```

（解放せずに、再度エリア確保）

②確保していないエリアの解放

```
unsigned char *area1, *area2;
area1 = malloc(AREA_SIZE);
        ・
free(area1);
free(area2);
```

（確保していないエリア area2 の解放を行う）

【対策】

決定的な対策はないが、malloc と free は、極力同一関数内で実行するようにし、プログラム作成時にセットで組み込むようにする。

１－７．スタックオーバーフロー

【内容】

ローカル変数を大量に確保したり、引数を大量に用いて、アプリケーションで使用可能なスタック領域（２ＫＢ）を超えてしまう。

【例】

```
usr_func(UB *str,struct K_DEF *k_def,int len)
{
    unsigned char s_buf[1024],r_buf[1024];
        ・
}
```

（上記の場合、ローカル変数で２０４８バイト、引数で１２バイトのため、２ＫＢを超えている）

【対策】

・変数を static 型やグローバルで確保し、スタックを使わないようにする。
・関数の引数を極力少なくする（構造体にまとめてしまう、自身の戻り値を用する、等）

☆なお、コンパイル時にリストファイル（－ｌオプションにて作成される）を作成する事で、各関数が使用するスタックサイズが確認できます。
（ソフトウェア解説書の「４．１　スタック領域について」参照）

【対策例】

```
unsigned char s_buf[1024],r_buf[1024];
usr_func(UB *str,struct K_DEF *k_def,int len)
{
        ・
}
```

１－８．レジュームＯＦＦ時のファイルオープン＆電源ＯＦＦ

【内容】

レジュームＯＦＦの状態でファイルをオープンし、クローズしていない状態で電源を切ると、次に電源を入れた際にＩＮＩＴ　ＥＲＲＯＲが発生する。

（下記の画面）

```
INIT ERROR
R4:00000003
R5:00000000
R6:00000003
```

【例】

```
dat_pwr_str.res_md = RESUME_OFF;                    /* resume off */
dat_system(SYSD_FNC_WRITE,SYSD_PWR,&dat_pwr_str); /* mode set */
        ・
fp = fopen("test.dat", "r+");
        ・
fclose(fp);
```

（上記のプログラムで、ファイルオープンの後、クローズを行う前に電源が切られると、次に電源を入れたときにＩＮＩＴ　ＥＲＲＯＲが発生する）

【対策】

・アプリケーションは、レジュームＯＮにする。
それが出来ない場合は、ファイルオープン中の間だけレジュームＯＮ状態にするか、電源ＯＦＦ（ＬＢ５）を通知モードにして電源キーを無効にする。
・ファイルオープン中の状態はなるべく短くする。
（読み込み、書き込みの直前にオープンし、終了直後にクローズする）

【対策例】

```
dat_pwr_str.res_md = RESUME_ON;                     /* resume on */
dat_system(SYSD_FNC_WRITE,SYSD_PWR,&dat_pwr_str); /* mode set */
        ・
strcpy(buf, "ABC");
fp = fopen("test.dat", "r+");
fwrite(buf, 3, 1, fp);
  fclose(fp);
```

１－９．通信処理中の電源ＯＦＦ／ＯＮ

【内容】

通信処理を行っている最中に電源をＯＦＦ／ＯＮすると、受信バッファにゴミが乗ることがある。

【例】

```
c_din(COM0, buf);
```

この関数でデータ受信待ちの最中に電源が切られ、再び電源が入れられると受信バッファ中にゴミが乗ることがある。

【対策】

・受信データのチェックを必ず行う。
手組で通信プログラムを作成する場合、プロトコルによりデータのチェックを行うようにする。

【対策例】

１データのパリティビット（偶数ｏｒ奇数）を入れ、１パケットのデータの後ろに１～２バイトのチェック用データを付加する。

１－１０．Ｘｏｎ／Ｘｏｆｆによるビジー制御の注意

【内容】

通信のビジー制御でＸｏｎ／Ｘｏｆｆによる制御を行う場合、受信バッファのサイズは６７バイト以上で用意する。

【例】

```
UB r_buf[64];
param=B_19200|PARI_EVN|CHAR_8|STOP_1|RTS_ON|ER_ON|SI_OFF|XON_XOFF;
c_open(COM0, param, r_buf, buf_len, &time_out, &del_cod, DC3, DC1);
```

上記の場合、バッファが６４バイトしかないため、Ｘｏｎ／Ｘｏｆｆは正常に機能しません。

【対策】

・バッファは必ず６７バイト以上取る。

１－１１．キー入力バッファの初期化

【内容】

キー入力関数（key_read、key_string、key_num）に使用する入力用バッファを正常に初期化していないと、暴走することがある。

【例】

```
KEY_INPS  key_inps;
UB   abc[21];
     …
key_string(&key_inps, abc);
```

（キー入力関数は、バッファに格納されている値を表示するが、ここにゴミが入っていた場合、暴走することがある）

【対策】

ＮＵＬＬクリアを行うか、妥当な値を入れて置く。

【対策例】

```
UB   abc[21];
  …
memset(abc, '¥0', sizeof(abc));
key_string(&key_inps, abc);
```

２．特定関数の使用上の注意点

２－１．ｆｌｇ＿ｓｔｓ

【内容】

通知モードで設定したフラグの状態を確認する関数ですが、永久ループ内では、この関数のみで脱出条件を設定することは避け、別の脱出条件（キー入力等）を入れるようにして下さい。

ＤＴ－７００では、タイミングによって、内部的にフラグがマスクされるケースがあります。そのため、フラグの状態が変わらず、同関数のみを脱出条件にした永久ループから抜けなくなります。

ただし、待ちの入る関数（key_read、wai_flg、c_din 等）を呼ぶことで、マスク状態は解除されますので、永久ループに入る直前（もしくはループ内）で、これらの関数を呼んでいただくことで永久ループから脱出できなくなる危険性は無くなります。

【プログラム例】

・悪い例

```
t_id=s_settimer(FL_TM2_INT_ID, FL_TM2_INT_ITU0, 3);
while(1){
    flg_sts(&dumy, &ptn, FL_TM2_INT_ID);
    if (ptn & FL_TM2_INT_ITU0)
        break;
}
```

・良い例

```
t_id=s_settimer(FL_TM2_INT_ID, FL_TM2_INT_ITU0, 1);
wai_flg(&ptn, FL_TM2_INT_ID, FL_TM2_INT_ITU0, TWF_ORW);
t_id=s_settimer(FL_TM2_INT_ID, FL_TM2_INT_ITU0, 2);
while(1){
    flg_sts(&dumy, &ptn, FL_TM2_INT_ID);
    if (ptn & FL_TM2_INT_ITU0)
        break;
}
```

２－２．ＯＢＲ＿ｇｅｔｓ

【内容】

読み込んだバーコードリーダをバーコードバッファから取り出す関数です。
この関数は、正常に終了すると与えられた３つの引数（文字列、読み取りコードの種類、読み取り文字数）にデータを設定して返しますが、この関数の実行と同時に電源キーが押されると、この関数はコードの種類に「データ無し」、読み取り文字数に０を入れて返し、文字列のバッファはＮＵＬＬクリアしています。
これは、電源ＯＦＦ処理時の状態で読み取ったデータが不定なためです。

電池の電圧が低下した状態でバーコード読み込みを行った際に、電源が切れることがありますが、この場合にも上記の内容は適用されます。

この関数の実行後、読み取ったコードの種別や文字数のチェックをせずに処理に用いると、不具合が発生することがありますので注意して下さい。

【プログラム例】

・悪い例

```
OBR_gets(buf, &rcd, &obrlen);
buf[obrlen-1] = '¥0';
```

・良い例

```
OBR_gets(buf, &rcd, &obrlen);
if (obrlen>0) {
    buf[obrlen-1] = '¥0';
      ・
      ・
```

最終ページです