RS-LAN 変換器 BLC-100

BIOS
リファレンス

本製品を正しくご利用いただくために
この説明書をよくお読みください。
また、お読みになりました後もすぐ取り出せる
場所に保管して、必要なときにご活用ください。

株式会社 ビッツ

はじめに

このたびは RS-232C/LAN 変換アダプタ「BLC-100」をお求めいただき、誠にありがとうございます。

BLC-100 には任意に制御できるようなユーザプログラムの実行が可能です。また、ユーザプログラムから簡単にかつ、高度に制御できる BIOS を装備しております。
本書では、ユーザプログラムの作成方法や BIOS の使用方法などについて説明しています。

本書は BLC-100 を対象とします。本書では、説明のため「本製品」と称します。

本製品の仕様ならびに外観、各部名称などは取扱説明書をご覧ください。
ユーザプログラムのダウンロードの方法については、セットアップマニュアルをご覧下さい。

ご注意

- 本書の内容の一部または全部を無断転載することは禁じられております。
- 本書に記載された内容は予告なく変更する場合があります。
- 本書の内容については万全を期していますが、万一ご不審な点や誤り、記載漏れなどお気づきのことがありましたら弊社営業までご一報ください。
- 弊社では、製品の運用を理由とする損失、逸失利益などの請求につきましては、本書の不審点や誤り、記載漏れなどに関わらず、いかなる責任も負いかねますのであらかじめご了承ください。
- 本書に記載される会社名および商品名は、各社の商標または登録商標です。

目　次

１．　概要

本書は、本製品上で動作させるユーザプログラムを作成するためのリファレンスマニュアルです。

本製品には、ハードウェアをユーザプログラムから制御するための BIOS を装備します。

ユーザプログラムは本製品の BIOS 機能を、無条件トラップ命令を使って呼び出します。

２．　ユーザプログラム

本製品には標準で FTP プロトコル、および TCP ソケットスループロトコルを搭載しており、接続形態によってはそのまま利用することもできますが、本 BIOS インタフェースを利用したユーザプログラムを作成し、標準搭載の FTP プロトコル・TCP ソケットスループロトコルを呼び出す事により、容易に高度な利用が可能となります。

２.１ ユーザプログラムの構造

ユーザプログラムには次のヘッダを付加し、論理アドレス 0x0c060000 以降にマッピングします。

アドレス	+0	+1	+2	+3
0c060000	プログラム名称			
0c060004				
0c060008				
0c06000c	0	（予約）	バージョン(上位)	バージョン(下位)
0c060010	プログラムサイズ			
0c060014	エントリアドレス			
0c060018 ： ：	プログラムコード・データ・変数・スタックエリア			

プログラム名称　ユーザプログラムの名称を指定します。
英数字と _（アンダーバー）のみ指定可能。大文字と小文字は区分されます。

バージョン　ユーザプログラムのバージョン番号を指定します。

プログラムサイズ　上記ヘッダを含めたプログラムエリア全体のバイト数を 65536 の倍数で指定します。65536 の倍数以外を指定した場合、エラーにはなりませんが、ダウンロード時に 65536 の倍数に切り上げられます。

エントリアドレス　ユーザプログラムのエントリアドレス（エントリ関数のアドレス）を指定します。ユーザプログラムが起動されると、システムはユーザプログラムにプロセス ID を割り当て、これを引数としてエントリ関数に渡して制御を移します。BIOS ファンクションを使用する際には、引数としてこのプロセス ID を指定する必要があります。CD(添付品)内の BIOSIF.C では、「BIOS 初期化」でプロセス ID を退避し、BIOS ファンクションコール時には「BIOS 初期化」で退避したプロセス ID を使用してコールするように実装されています。詳しくは「４．４．１ BIOS 初期化」を参照して下さい。

2.1.1 C言語例

C言語で前ページのヘッダを定義する場合、ソースファイルの先頭で以下のように記述します。

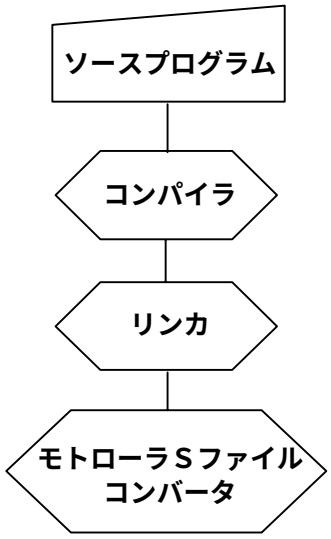
```
void main(int aPid);   /* エントリルーチンのプロトタイプ */

const struct {
    char    Name[13];
    char    dmy;
    char    Ver;
    char    Rev;
    int     size;
    void    (*func)(int);
} PrgHead = {“Sample”, 0, 1, 0, 65536, main};
```

2.2 ユーザプログラムの作成

ユーザプログラムは、

日立超ＬＳＩシステムズ社製の「SuperH RISC engine C/C++コンパイラパッケージ」

を使用してコンパイル・リンクし、最終ファイルとして本製品にダウンロードできるファイル形式である「モトローラＳファイル」を生成します。
本製品のユーザプログラムは、ビッグエンディアンで作成します。コンパイラの標準ライブラリは、「SH3用」「非ポジションインディペンデント」「ビッグエンディアン」のものを選択してリンクします。上記のコンパイラパッケージの場合、「shc3npb.lib」です。



セクション・アドレスの先頭を 0x0c060000 とした、アブソリュートファイルを生成します。

拡張子（.MOT）を生成します。

2.3 ユーザプログラムの本製品への組込み

「モトローラＳファイル」形式で生成したユーザプログラムのオブジェクトファイルは、本製品を設定モード（製品のロータリスイッチを「7」ポジションに設定した状態）にし、RS-232C または FTP により、本製品のユーザプログラム格納エリアへダウンロードします。

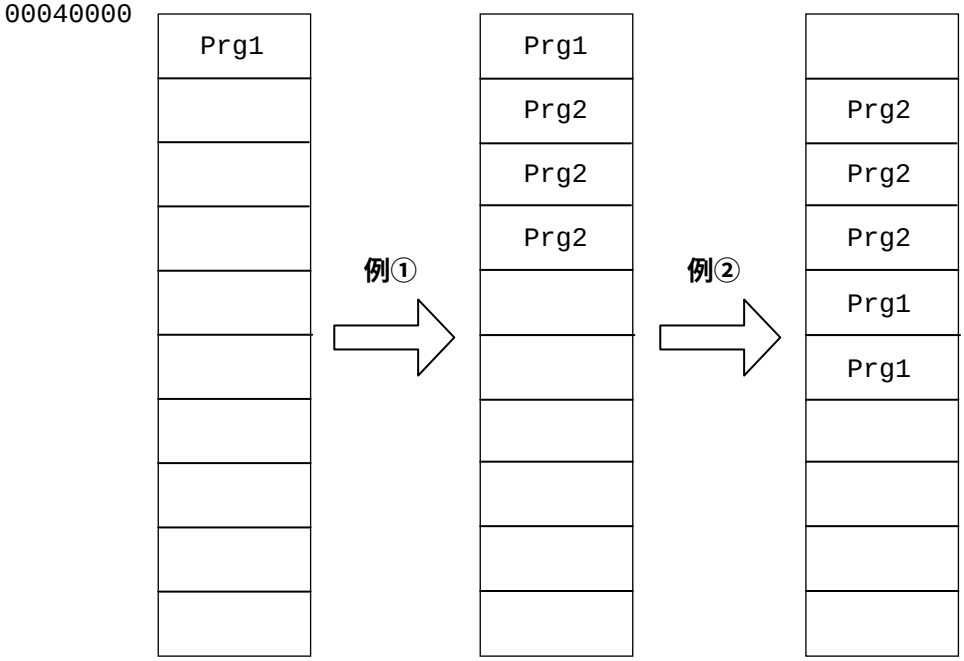
なお 、BIOS ファンクションの「WriteProgram」（プログラム書込み）ファンクションを使用することにより、ユーザプログラム実行中に、別のユーザプログラムを本製品のユーザプログラム格納エリアへダウンロードすることもできます。

本製品には、64 キロバイトを一単位としたユーザプログラム格納エリアを 5 個持っています。
すなわち、各ユーザプログラムが 64 キロバイト以内であれば、最大 5 本のユーザプログラムを本製品に組込み、指定したプログラムを動作させることができます。また、1 本のユーザプログラムで最大 512 キロバイトのプログラムとしても組込み、動作させることができます。

ユーザプログラムは、格納エリアの上位の空きエリアから格納されます。このとき、プログラムヘッダ中のプログラム名称がキーとなり、同じプログラム名称のプログラムをダウンロードすると、既に格納されているプログラムを上書きします。プログラムサイズが 64 キロバイトを超える場合、格納するのに必要な連続したエリアに格納されます。

例①：64 キロバイトのプログラム（Prg1）が 1 本格納されているときに、192 キロバイトのプログラム（Prg2）をダウンロードした場合、下図のように格納されます。

例②：さらに Prg1 のプログラムサイズを 128 キロバイトに変更してダウンロードした場合、下図のように格納されます。



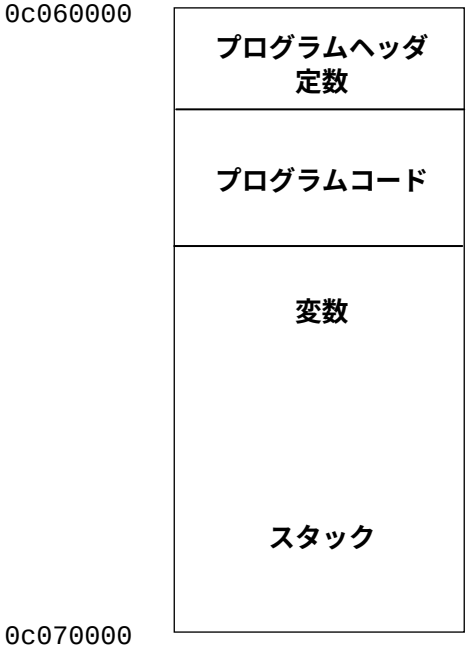
2.4 ユーザプログラムの実行

本製品に格納したユーザプログラムを実行させるには、本製品の動作パラメータ設定において、起動時に実行するプログラムの名称を「起動プログラム」の項目に指定します。

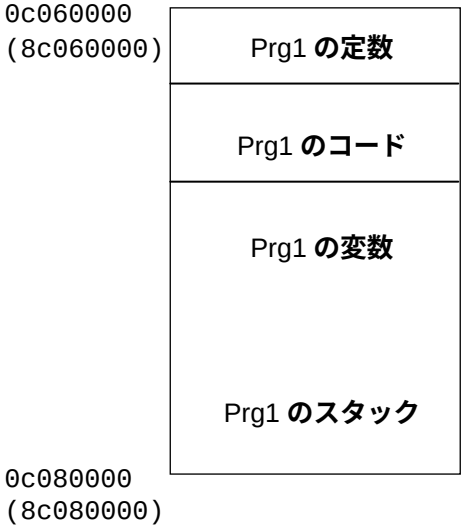
ユーザプログラムは実行時に格納エリアから、0x0c060000 番地から始まる RAM エリアに転送されます。

プログラムサイズに 65536 を指定した場合の実行時のメモリ配置は、サンプルプログラムの場合を例とすると、以下のようになります。

スタックポインタはプログラムサイズが 65536 の場合は 0x0c070000 に、131072（＝65536×2）の場合は 0x0c080000 に初期化されます。



ユーザプログラムの実行エリアは論理アドレスです。複数のユーザプログラムが動作する場合も、各プログラムは 0x0c060000 番地から配置されますが、対応する物理アドレスは互いに重ならないように配置されます。131072 バイトの大きさのプログラム（Prg1）と 196608 バイトの大きさのプログラム（Prg2）が同時に動作する場合、下図のような配置になります。() 内は物理アドレスを表わします。

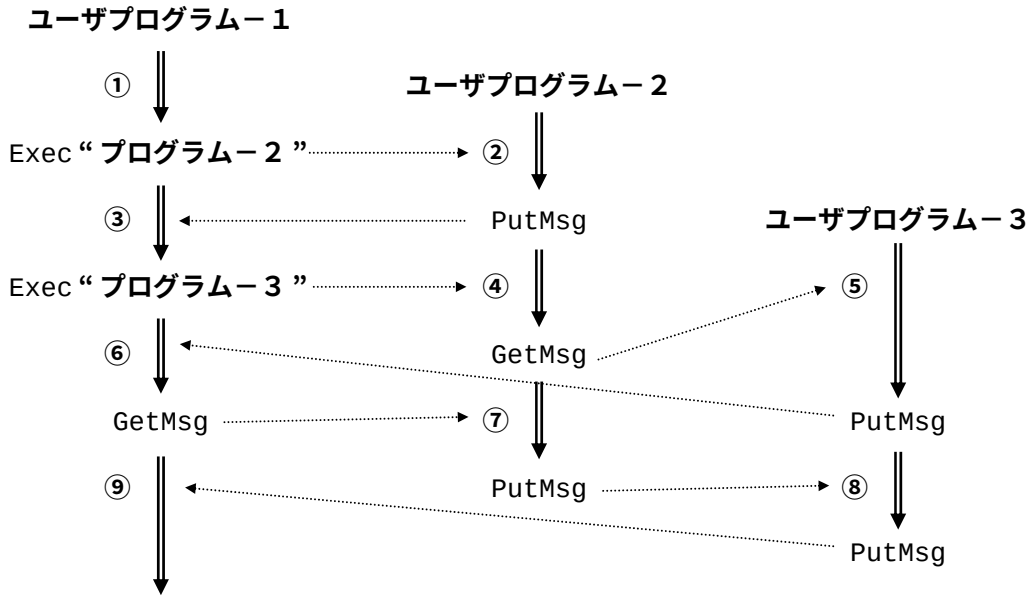


2.5 ユーザプログラムの制御

本製品が起動されるとシステムは、動作パラメータで指定されている「起動プログラム」名称をもとに、該当するユーザプログラムをユーザプログラム格納エリアよりプログラム実行メモリにロードし、プロセス ID の付与と共に、制御をユーザプログラムのエントリアドレスに移します。

ユーザプログラムを複数実行させる場合は、本製品起動後最初に実行するユーザプログラム内において、BIOS ファンクション「Exec」（プログラム実行）ファンクションを利用して、他ユーザプログラムを実行します。
「Exec」ファンクションが呼び出されると、システムは該当プログラムをユーザプログラム格納エリアよりプログラム実行メモリにロードし、新たなプロセス ID の付与と共に、制御をそのユーザプログラムのエントリアドレスに移し、呼び出し元のプログラムは一時停止状態となります。
「Exec」ファンクションは、ユーザプログラムの起動のほか、システムの持つシステムプロセス（FTP サーバプロセス・ソケットクライアントプロセス・ソケットサーバプロセス・TELNET プロセス・RS-232C スループロセス）をユーザプログラムから利用する場合にも使用します。なおシステムプロセスについては「6．システムプロセスについて」を参照して下さい。

なお、呼び出し元プログラムが一時停止状態となる BIOS ファンクションには、「Exec」ファンクションのほか、「PutMsg」（メッセージ追加）ファンクション・「GetMsg」（メッセージ取得）ファンクション・「TimeWait」（指定時間ウェイト）ファンクションがあります。
以下に制御の移り変わりの例を示します。



①～⑨は処理の順番を示します。また、 ┈┈▸ は制御の移り変わりを示します。

３．　BIOS 機能

３.１ BIOS の使用方法

３.１.１ アセンブラから呼び出す場合

本 BIOS ファンクションをアセンブラから呼び出す場合は、以下の手順により実行して下さい。

① R0 レジスタにプロセス ID をセットします。

② R4～R7 レジスタに必要なパラメータをセットします。

③ TRAPA #imm 命令で BIOS を実行します。
#imm には使用する BIOS ファンクションのトラップ番号を指定します。

３.１.２ C 言語から呼び出す場合

本 BIOS ファンクションを C 言語から呼び出す場合は、

・ 組込み用関数の `trapa_svc()`

を使用して下さい。

４．　BIOS ファンクションリファレンス

本書では、各ファンクションの説明を以下の書式で行います。

【機能】

ファンクションの機能概要を示します。

【入力】

ファンクションの入力パラメータについて説明します。

トラップ番号　アセンブラ利用の場合、trapa #imm 命令で#imm に指定する値を示します。
C 言語利用の場合、trapa_svc()の第 1 引数で指定する値を示します。

コード（R0）　アセンブラ利用の場合、R0 にプロセス ID を指定します。
C 言語利用の場合、trapa_svc()の第 2 引数にプロセス ID を指定します。

R4,R5,R6,R7　アセンブラ利用の場合、各レジスタに指定する内容を示します。
C 言語利用の場合、trapa_svc()の第 3～第 6 引数に指定する内容を示します。
なお、指定する値に制限がある場合、その有効範囲も示します。
また、パラメータがポインタの場合は、内容のメモリイメージを示します。

【出力】

リターンコード　アセンブラ利用の場合、BIOS から戻ったときに R0 レジスタに返される値と、その意味を示します。
C 言語利用の場合、trapa_svc()の戻り値と、その意味を示します。

出力パラメータ　BIOS より返されるパラメータがある場合、その内容のメモリイメージを示します。

【解説】

ファンクションの処理内容を説明します。

【C 関数例】

C 言語利用の場合の BIOS 呼び出し関数例を示します。

【関連項目】

関連するファンクション、メッセージ等を示します。

【使用例】

簡単な使用例を示します。

!注意!
本製品内部では、IP アドレスを 4 バイトの符号なし整数として扱います。
たとえば、IP アドレス 1.2.3.4 は 0x01020304 となります。

以下に BIOS ファンクションの一覧を示します。

システム機能

トラップ番号	関数名	機能
0x20	TmStart	タイマ開始
0x21	TmGet	タイマ取得
0x22	TmStop	タイマ停止
0x23	GetSw	スイッチ取得
0x24	GetTime	時計取得
0x25	SetTime	時計設定
0x26	SetAlarm	アラーム時刻設定
0x27	SetAlarmInt	アラーム割込み設定
0x28	GetParam	パラメータ取得
0x29	SetParam	パラメータ設定
0x2a	GetVect	割込みベクタ取得
0x2b	SetVect	割込みベクタ設定
0x2c	Malloc	メモリ取得
0x2d	Free	メモリ解放
0x2e	PutMsg	メッセージ追加
0x2f	GetMsg	メッセージ取得
0x30	Exec	プログラム実行
0x31	Kill	プログラム停止
0x32	TimeWait	指定時間ウェイト
0x33	GetPid	プロセス ID 検索
0x34	PmErase	PM 領域消去
0x35	PmWrite	PM 領域書き込み
0x36	PmRead	PM 領域読み込み
0x37	WriteProgram	プログラム書き込み
0x38	EraseProgram	プログラム消去
0x39	Reset	リセット
0x3a	GetVer	バージョン番号取得
0x3b	LedCtrl	LED 制御
0x3c	LedCtrlAll	LED 一括制御
0x3d	GetUsrPrm	ユーザパラメータ取得
0x3e	SetUsrPrm	ユーザパラメータ設定

RS-232C 機能

トラップ番号	関数名	機能
0x40	RsOpen	RS オープン
0x41	RsClose	RS クローズ
0x42	RsSend	RS 送信
0x43	RsRecv	RS 受信
0x44	RsGetBuff	RS バッファ量取得
0x45	RsFlushRecv	RS 受信バッファフラッシュ
0x46	RsFlushSend	RS 送信バッファフラッシュ
0x47	RsGetCtrl	制御線状態取得
0x48	RsSetCtrl	制御線状態設定
0x49	RsOpenX	RS オープン 2
0x4a	RsCloseX	RS クローズ 2
0x4b	RsSetBreak	ブレーク送信

ネットワーク機能

トラップ番号	関数名	機能
0x50	LanOpen	LAN オープン
0x51	LanClose	LAN クローズ
0x52	Socket	ソケット作成
0x53	Bind	ソケットの名前付け
0x54	Connect	クライアント接続開始
0x55	ConnectRes	クライアント接続確認
0x56	Listen	接続要求受付開始
0x57	Accept	接続要求受付
0x58	Send	ストリーム送信
0x59	SendTo	パケット送信
0x5a	Recv	ストリーム受信
0x5b	RecvFrom	パケット受信
0x5c	LanGetBuff	ｿｹｯﾄ送受信ﾊﾞｯﾌｧ量取得
0x5d	Select	イベントチェック
0x5e	Close	ソケットクローズ
0x5f	Abort	ソケット強制クローズ
0x60	FlushRecvBuff	LAN 受信バッファクリア
0x61	FlushSendBuff	LAN 送信バッファクリア
0x62	Arp	ARP 送信
0x63	Rarp	RARP 送信
0x64	Icmp	ICMP 送信
0x65	DhcpReq	DHCP 要求送信
0x66	GetMacAddr	MAC アドレス取得

4.1 システム機能

4.1.1 タイマ開始

【機能】
1ms タイマのカウントを開始します。

【入力】
トラップ番号	0x20
コード（R0）	プロセス ID

【出力】
リターンコード	0～31	タイマ番号
	ERR_PID	不正プロセス ID
	ERR_MEMOVER	タイマ番号の空きがない

【解説】
本製品は内部に 32 個の独立したタイマカウンタを持ちます。
本ファンクションにより、現在使用されていないタイマカウンタのカウントを 0 から開始し、そのタイマ番号を返します。タイマカウンタの現在値は「タイマ取得」ファンクションで取得できます。

タイマカウンタは、1ms 毎のタイマ割込みにおいて更新されます。
１カウントはおよそ 1ms ですが、「タイマ開始」ファンクションおよび、「タイマ取得」ファンクションを呼び出すタイミングと、タイマ割込みのタイミングによっては、実時間との差が±1ms 程度出る場合があります。

タイマカウンタの最大値は約 49 日（4,294,967,295 カウント）で、最大値を越えるとカウンタは 0 に戻ります。

【C 関数例】
```
int TmStart(void)
{
    return trapa_svc(0x20, pid);
}
```

【関連項目】
タイマ取得、タイマ停止

【使用例】
以下の関数は約 10ms ウェイトします。
```
void wait10ms(void)
{
    int          tmno;
    unsigned int tmval;

    tmno = TmStart();
    for(tmval = 0; tmval < 10; ) {
        TmGet(tmno, &tmval);
    }
    TmStop(tmno);
}
```

4.1.2 タイマ取得

【機能】
　　タイマカウンタの現在値を取得します。

【入力】

トラップ番号	0x21
コード（R0）	プロセス ID
R4	タイマ番号 0～31
R5	タイマカウント値を格納する領域の先頭アドレス

【出力】

リターンコード	OK　　正常終了

	+0	+1	+2	+3
R5+00→	タイマカウント値がセットされます			

ERR_PARAM	パラメータエラー
ERR_PID	不正プロセス ID
ERR_INUSE	指定のタイマは他で使用中

【解説】
　　指定タイマ番号の現在のカウント値を返します。

　　タイマ番号が範囲外の場合、パラメータエラーを返します。

　　タイマカウント値を格納する領域のポインタが NULL の場合、または 4 バイト境界のアドレスを指していない場合は、パラメータエラーを返します。

　　タイマ番号が他のプロセスで使用中の場合、エラーを返します。

　　「タイマ取得」ファンクションで取得したタイマ番号以外を指定した場合、ERR_INUSE を返します。

【C 関数例】

```
int TmGet(int no, unsigned int *value)
{
    return trapa_svc(0x21, pid, no, value);
}
```

【関連項目】
　　タイマ開始、タイマ停止

4.1.3 タイマ停止

【機能】
1ms タイマのカウントを停止します。

【入力】

トラップ番号	0x22	
コード（R0）	プロセス ID	
R4	タイマ番号 0〜31	

【出力】

リターンコード	OK	正常終了
	ERR_PARAM	パラメータエラー
	ERR_PID	不正プロセス ID
	ERR_INUSE	指定のタイマは他で使用中

【解説】
指定タイマ番号のカウント動作を停止します。

タイマ番号が範囲外の場合、パラメータエラーを返します。

指定タイマ番号が他のプロセスで使用中の場合、エラーを返します。

「タイマ取得」ファンクションで取得したタイマ番号以外を指定した場合、ERR_INUSE を返します。

【C 関数例】
```
int TmStop(int no)
{
    return trapa_svc(0x22, pid, no);
}
```

【関連項目】
タイマ開始、タイマ取得

4.1.4 スイッチ取得

【機能】
　　ロータリスイッチの設定値を取得します。

【入力】

トラップ番号	0x23
コード（R0）	プロセス ID
R4	スイッチ区分 0：現在設定値 1：電源投入（リセット）時設定値

【出力】

リターンコード	0～7	ロータリスイッチの設定値
	ERR_PARAM	パラメータエラー

【解説】
　　スイッチ区分が 0 の場合、現在設定されているロータリスイッチの値を返します。

　　スイッチ区分が 1 の場合は、電源投入またはリセット時に設定されていたロータリスイッチの値を返します。

　　スイッチ区分が範囲外の場合、パラメータエラーを返します。

【C 関数例】

```
int GetSw(int flg)
{
    return trapa_svc(0x23, pid, flg);
}
```

4.1.5 時計取得

【機能】
　　リアルタイム・クロックより現在日時を取得します。

【入力】
　　トラップ番号　　0x24
　　コード（R0）　　プロセス ID
　　R4　　現在日時を格納する領域の先頭アドレス

【出力】
　　リターンコード　　OK　　正常終了

	+0	+1	+2	+3
R4+00→	年(BCD)	月(BCD)	日(BCD)	曜日
+04	時(BCD)	分(BCD)	秒(BCD)	1/100 秒(BCD)

　　ERR_PARAM　　パラメータエラー

【解説】
　　年・月・日・曜日・時・分・秒・1/100 秒、それぞれのカウンタ値を返します。

　　曜日の値と意味は以下の通りです。
　　0 = 日曜日　1 = 月曜日　2 = 火曜日　3 = 水曜日　4 = 木曜日　5 = 金曜日
　　6 = 土曜日

　　現在日時を格納する領域のポインタが NULL の場合、パラメータエラーを返します。

【C 関数例】
```
int GetTime(unsigned char *dtime)
{
    return trapa_svc(0x24, pid, dtime);
}
```

【関連項目】
　　時計設定、アラーム時刻設定

4.1.6 時計設定

【機能】
　　リアルタイム・クロックの設定値を変更します。

【入力】
　　トラップ番号　　0x25
　　コード（R0）　　プロセス ID
　　R4　　設定データが格納されている領域の先頭アドレス

	+0	+1	+2	+3
R4+00→	年(BCD)	月(BCD)	日(BCD)	曜日
+04	時(BCD)	分(BCD)	秒(BCD)	

【出力】
　　リターンコード　　OK　　正常終了
　　ERR_PARAM　　パラメータエラー

【解説】
　　年・月・日・曜日・時・分・秒のそれぞれのカウンタ値を、指定された値に変更します。システム内部の秒未満のカウンタは 0 にリセットされます。

　　年は 0～99、月は 1～12、日は 1～31、曜日は 0～6、時は 0～24、分・秒は 0～59 が有効な値の範囲です。
　　ただし、日は年・月により有効範囲が変化します。また、年が 4 の倍数の場合「うるう年」となります。
　　曜日の値と意味は以下の通りです。
　　0 = 日曜日　1 = 月曜日　2 = 火曜日　3 = 水曜日　4 = 木曜日　5 = 金曜日
　　6 = 土曜日

　　設定データエリアのポインタが NULL の場合、パラメータエラーを返します。

【C 関数例】
```
int SetTime(unsigned char *dtime)
{
    return trapa_svc(0x25, pid, dtime);
}
```

【関連項目】
　　時計取得、アラーム時刻設定

4.1.7 アラーム時刻設定

【機能】

リアルタイム・クロックにアラーム時刻を設定します。

【入力】

トラップ番号 0x26
コード（R0） **プロセス** ID
R4 **設定データが格納されている領域の先頭アドレス**

	+0	+1	+2	+3
R4+00→	**月**(BCD)	**日**(BCD)	**時**(BCD)	**分**(BCD)
+04	**秒**(BCD)			

【出力】

リターンコード	OK	**正常終了**
	ERR_PARAM	**パラメータエラー**

【解説】

指定された値をアラーム時刻として設定します。

本ファンクションはアラーム時刻の設定のみです。

「アラーム割込み設定」ファンクションにより、割込み処理アドレスおよび、アラーム割込みを 1 **（許可）に設定する事により、現在時刻がアラーム時刻に達するとアラーム割込みが発生し、 「アラーム割込み設定」ファンクションにより設定された割込み処理アドレスに制御が移ります。**

月は 1～12 **と** 99**、日は** 1～31 **と** 99**、時は** 0～24**、分・秒は** 0～59 **が有効な値の範囲です。**
月ないし日に 99 **を指定すると、その項目はアラーム時刻対象外となります。**
月・日両方に 99 **が指定すると、アラーム時刻は時・分・秒のみが有効となります。**

設定データエリアのポインタが NULL **の場合、パラメータエラーを返します。**

【C **関数例】**

```
int SetAlarm(unsigned char *atime)
{
    return trapa_svc(0x26, pid, atime);
}
```

【関連項目】

アラーム割込み設定

【使用例】

以下は指定時刻になると文字列を RS-232C **に送信します。**

```
void alarm_msg(void)
{
    RsSend("Alarm Interrupt !!¥r¥n", 20);
}

void main(int aPid)
{
    unsigned char alarm[5];
    int rsprm[3];

    B_Init(aPid);
    GetParam(rsprm, 1);
    RsOpen(rsprm);

    alarm[0] = 0x99;  alarm[1] = 0x99;
    alarm[2] = 0x08;  alarm[3] = 0x30;  alarm[4] = 0x00;
    SetAlarm(alarm);
    SetAlarmInt((int *)alarm_msg, 1);
    for(;;) {
    }
}
```

4.1.8 アラーム割込み設定

【機能】
　　アラーム割込みの許可／禁止を指定します。

【入力】

トラップ番号	0x27
コード（R0）	プロセス ID
R4	アラーム割込み発生時の処理アドレス
R5	アラーム割込みの許可／禁止 0 = 禁止 1 = 許可

【出力】

リターンコード	OK	正常終了
	ERR_PARAM	パラメータエラー

【解説】
　　現在時刻が、「アラーム時刻設定」ファンクションにより設定されたアラーム時刻に達した場合の、動作を指定します。

　　本ファンクションにより「アラーム割込み発生時の処理アドレス」、および「アラーム割込みの許可」を指定し、現在時刻がアラーム時刻に達した場合、CPU の制御は指定処理アドレスに移ります。
　　アラーム割込み処理では、ユーザ独自の処理を記述することができます。アラーム割込み処理の最後では、RET 命令を使用します。

　　本ファンクションで「アラーム割込みの禁止」を指定すると、以前のファンクションで指定されていた「アラーム時刻設定」および、「アラーム割込み発生時の処理アドレス」に影響を与えず、割込みのみを禁止します。

　　アラーム割込み発生時の処理アドレスが NULL の場合、パラメータエラーを返します。

　　アラーム割込みの許可／禁止パラメータが範囲外の場合、パラメータエラーを返します。

【C 関数例】

```
int SetAlarmInt(int *vect, int flg)
{
    return trapa_svc(0x27, pid, vect, flg);
}
```

【関連項目】
　　アラーム時刻設定

4.1.9 パラメータ取得

【機能】

システム内に格納されている動作パラメータを取得します。

【入力】

トラップ番号	0x28
コード（R0）	プロセス ID
R4	動作パラメータを格納する領域の先頭アドレス
R5	動作パラメータの種別
	0 = LAN
	1 = シリアル
	2 = システム

【出力】

リターンコード　OK　正常終了

種別に LAN を指定した場合

	+0	+1	+2	+3
R4+00→	自 IP アドレス			
+04	自ポート番号		（空き）	
+08	宛先 IP アドレス			
+0c	宛先ポート番号		（空き）	
+10	デフォルトゲートウェイ			
+14	サブネットマスク			
+18	コネクトタイマ	Client 接続契機	無通信監視タイマ	
+1b	ユーザ名（16 バイト）			
+2b	パスワード（16 バイト）			
+3b	ファイルターミネータ			
+3e	BOOTP 設定	DHCP 設定		

種別にシリアルを指定した場合

	+0	+1	+2	+3
R4+00→	通信速度			
+04	データ長	パリティ	ストップビット	フロー制御
+08	レコードターミネータ			

種別にシステムを指定した場合

	+0	+1	+2	+3
R4+00→	起動プログラム 1 名称（13 バイト）			
+0d	起動プログラム 2 名称（13 バイト）			
	：			
	起動プログラム 5 名称（13 バイト）			
+c3	起動プログラム 6～起動プログラム 16　未使用			
+d0	LAN デフォルトプロトコル名称（13 バイト）			
+dd	RS デフォルトプロトコル名称（13 バイト）			

ERR_PARAM　パラメータエラー

【解説】

動作パラメータの設定値を、指定された動作パラメータ種別にしたがいシステムから取得します。

動作パラメータ格納域のポインタが NULL の場合、または 4 バイト境界にない場合は、パラメータエラーを返します。

動作パラメータ種別が範囲外の場合、パラメータエラーを返します。

以下に動作パラメータ概要について示します。

グループ	項目名	内　容	初期値
LAN	自 IP アドレス	本製品の IP アドレス	192.168.200.254
	自ポート番号	本製品のポート番号	257
	宛先 IP アドレス	宛先の IP アドレス	2.2.2.2
	宛先ポート番号	宛先のポート番号	514
	デフォルトゲートウェイ	デフォルトゲートウェイの IP アドレス	2.2.2.2
	サブネットマスク	サブネットマスク	255.255.255.0
	コネクトタイマ(秒)	クライアント動作時、コネクト要求に対する応答待ちのタイムアウト時間（2～120 秒）	2（2 秒）
	クライアント接続契機	ソケットスルー(Client)モード時の接続タイミング	0（RS 受信で接続）
	無通信監視タイマ(秒)	ソケットスルー(Client)モード時の無通信監視タイマ（0～240 秒）	0（切断しない）
	ユーザ名	FTP のログイン許可ユーザ名	0x00（なし）　※
	パスワード	FTP のログインパスワード	0x00（なし）　※
	ファイルターミネータ	FTP のデータ出力の区切り文字	0x00（なし）
	BOOTP 設定	BOOTP 機能有効／無効	0（無効）
	DHCP 設定	DHCP 機能有効／無効	0（無効）
シリアル	通信速度	1200～230400bps	9600（9600bps）
	データ長	7bit／8bit	8（8bit）
	パリティ	なし'N'／偶数'E'／奇数'O'	N（なし）
	ストップビット	1bit／2bit	1（1bit）
	フロー制御	なし／RTS･CTS／XON･XOFF	0（なし）
	レコードターミネータ	１レコードとするための区切り文字	0x0D+0x0A（CR+LF）
システム	起動プログラム１～１６	ユーザプログラム 1～１６の名前	0x00（なし）　※
	LAN デフォルトプロトコル	使用するネットワークプロトコルの名前	FtpSv（FTP サーバ）
	RS デフォルトプロトコル	使用する RS-232C プロトコルの名前	Serial（RS スルー）

※：NULL 終端文字列で空文字列を初期値とします。

【C 関数例】

```
int GetParam(int *param, int flg)
{
    return trapa_svc(0x28, pid, param, flg);
}
```

【関連項目】

パラメータ設定、RS オープン

4.1.10　　パラメータ設定

【機能】
　　動作パラメータをシステム内に設定します。

【入力】

トラップ番号	0x29
コード（R0）	プロセス ID
R4	動作パラメータの設定値が格納されている領域の先頭アドレス
R5	動作パラメータの種別 0 = LAN 1 = シリアル 2 = システム

【出力】

リターンコード	OK	正常終了
	ERR_PARAM	パラメータエラー
	ERR_FLSHWRT	フラッシュ書込みエラー

【解説】
　　動作パラメータをシステム内に設定します。
　　設定した値は、リセットまたは電源再投入後に有効となります。
　　パラメータのメモリ配置については、「パラメータ取得」ファンクションの【出力】を参照して下さい。

　　動作パラメータ格納域のポインタが NULL の場合、または 4 バイト境界にない場合は、パラメータエラーを返します。

　　動作パラメータ種別が範囲外の場合、パラメータエラーを返します。

【C 関数例】

```
int SetParam(int *param, int flg)
{
    return trapa_svc(0x29, pid, param, flg);
}
```

【関連項目】
　　パラメータ取得

4.1.11　　割込みベクタ取得

【機能】
　　割込みベクタアドレスを取得します。

【入力】
　　トラップ番号　　0x2a
　　コード（R0）　　プロセス ID
　　R4　　ベクタを取得する割込みコード番号
　　　9,15～17
　　R5　　トラップ番号
　　　128～255（0～127 はシステムでリザーブ）
　　R6　　ベクタアドレスを格納する領域の先頭アドレス

【出力】
　　リターンコード　　OK　　正常終了

	+0	+1	+2	+3
R6+00→	割込み・トラップのエントリアドレス			

　　ERR_PARAM　　パラメータエラー

【解説】
　　割込みコード番号に対応するハンドラルーチンのエントリアドレスを返します。
　　割込みコード番号が無条件トラップの場合、トラップ番号に対応するハンドラルーチンのエントリアドレスを返します。

　　割込みコード番号・トラップ番号が範囲外の場合、またはベクタアドレスを格納する領域のポインタが NULL の場合、パラメータエラーを返します。

　　取得可能な、割込みコード番号とその内容は以下の通りです。

割込みコード番号	内　　容
9	無条件トラップ
15	タイマ 0 割込み
16	タイマ 1 割込み
17	タイマ 2 割込み

【C 関数例】
```
int GetVect(int vect, int trp, void *(*func)(void))
{
    return trapa_svc(0x2a, pid, vect, trp, func);
}
```

【関連項目】
　　割込ベクタ設定

4.1.12　　割込みベクタ設定

【機能】
　　割込みベクタアドレスを変更します。

【入力】

トラップ番号	0x2b
コード（R0）	プロセス ID
R4	ベクタを変更する割込みコード番号 9,15～17
R5	トラップ番号 128～255（0～127 はシステムでリザーブ）
R6	変更するベクタアドレス

【出力】

リターンコード	OK	正常終了
	ERR_PARAM	パラメータエラー

【解説】
　　割込みコード番号に対応するハンドラルーチンのエントリアドレスを変更します。
　　割込みコード番号が無条件トラップの場合、トラップ番号に対応するハンドラルーチンのエントリアドレスを変更します。

　　割込みコード番号・トラップ番号が範囲外の場合、パラメータエラーを返します。

　　割込みコード番号とその内容の対応については、「割込みベクタ取得」ファンクションを参照して下さい。

　　割込みコード番号 15～17（タイマ割込み）をユーザプログラムで使用する場合は、あらかじめ変更前のベクタを退避しておき、ユーザ割込み処理終了時に変更前のベクタへジャンプして下さい。

【C 関数例】

```
int SetVect(int vect, int trp, void (*func)(void))
{
    return trapa_svc(0x2b, pid, vect, trp, func);
}
```

【関連項目】
　　割込ベクタ取得

4.1.13　　メモリ取得

【機能】
　　動的メモリを取得します。

【入力】

トラップ番号	0x2c
コード（R0）	プロセス ID
R4	確保するバイト数
R5	確保したメモリ領域の先頭アドレスが格納されている、メモリ管理テーブルへのポインタ

【出力】

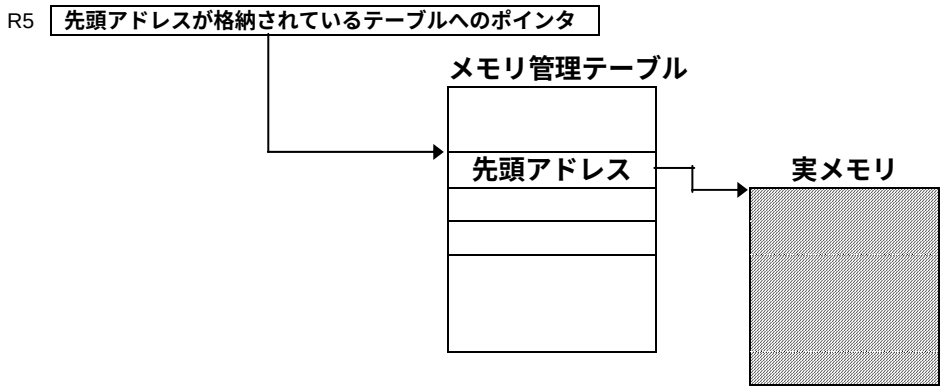
リターンコード	OK	正常終了

	+0	+1	+2	+3
R5+00→	メモリ領域の先頭アドレスが格納されているテーブルへのポインタ			

ERR_PARAM	パラメータエラー
ERR_PID	不正プロセス ID
ERR_MEMOVER	メモリ不足

【解説】
　　指定された容量のメモリを割り当て、その先頭アドレスが格納されている、システム中のメモリ管理テーブルへのポインタを返します。

　　確保するバイト数が 0 の場合、パラメータエラーを返します。
　　テーブルへのポインタを格納するポインタが NULL の場合、または 4 バイト境界にない場合は、パラメータエラーを返します。



【C 関数例】

```
int Malloc(int size, void ***addr)
{
    return trapa_svc(0x2c, pid, size, addr);
}
```

【関連項目】
　　メモリ解放、メッセージ追加、メッセージ取得

【注意】

本ファンクションにより返されるポインタは、実メモリのダブルポインタです。実メモリは、BIOS ファンクション呼び出し時に動くことがありますので、ユーザプログラムでは、このダブルポインタを基点としてアクセスするようにして下さい。

本ファンクションにより割り当てられるメモリは、0c10000～0c1fffff 番地の 1 メガバイトのエリアから確保します。以下の場合には、システムからもこのエリアから動的にメモリを確保します。

確保契機	解放契機	確保容量	最大
TCP の接続確立時	TCP の接続切断時	送受信バッファとして各 4 キロバイト	384 キロバイト（48 ソケット分）
UDP ソケットへのデータ送信時	データ送信完了時	送信するデータサイズ	5100 バイト
UDP ソケットへのデータ受信時	パケット受信ファンクションでデータを取り出した時	受信データサイズ	957 キロバイト（5100 バイト×4 パケット×48 ソケット）
システムプロセスからメッセージを発行する場合で、パラメータサイズが4バイトを超える時	メッセージ受信側で解放した時	通知するデータのサイズ	1 メッセージあたり 64 キロバイト

動的に確保したメモリは、メッセージで他のプロセスに渡した場合を除き、確保したプロセスで解放して下さい。

4.1.14　　メモリ解放

【機能】
　　動的メモリを解放します。

【入力】

トラップ番号	0x2d	
コード（R0）	プロセス ID	
R4	解放するメモリ領域に対するメモリ管理テーブルへのポインタ 「メモリ取得」ファンクションで、システムより返されたポインタをそのまま指定します。	

【出力】

リターンコード	OK	正常終了
	ERR_PARAM	パラメータエラー
	ERR_PID	不正プロセス ID
	ERR_INUSE	指定のエリアは他のプロセスで使用中

【解説】
　　指定されたメモリ領域を解放します。他で確保されたメモリブロックを再配置し、未使用領域が連続したエリアになるようにします。

　　指定されたメモリ管理テーブルへのポインタが NULL の場合、パラメータエラーを返します。

　　指定されたアドレスが使用されていない場合は、パラメータエラーを返します。

【C 関数例】
```
int Free(void **addr)
{
    return trapa_svc(0x2d, pid, addr);
}
```

【関連項目】
　　メモリ確保、メッセージ追加、メッセージ取得

4.1.15　　メッセージ追加

【機能】
　　システムメッセージキューにメッセージを追加します。

【入力】
　　トラップ番号　　0x2e
　　コード（R0）　　プロセス ID
　　R4　　メッセージの先頭アドレス

	+0	+1	+2	+3
R4+00→	宛先プロセス ID		自プロセス ID	
+04	メッセージ種別	コマンドコード	パラメータ長	
+08	パラメータ または パラメータの先頭アドレス			

メッセージ種別	0	要求メッセージ
	1	応答メッセージ
	2	エラーメッセージ
コマンドコード	0〜99	システムで予約
	100〜255	ユーザプログラム間で利用可能

【出力】

リターンコード	OK	正常終了
	ERR_PARAM	パラメータエラー
	ERR_PID	不正プロセス ID
	ERR_QUEFULL	メッセージの追加が行われなかった

【解説】
　　システムのメッセージキューにメッセージを追加し、プロセス制御の切り替えを行います。
　　システムは、各プロセス ID 毎に 15 個のメッセージキューを持ち、このキューとプロセス ID によりプロセス制御を行います。なお本ファンクションからは、本ファンクション発行元プロセスに制御が戻った場合にリターンします。

　　本ファンクションで受け渡すパラメータは、「メモリ取得」ファンクションで確保したメモリ領域を使用して下さい。パラメータの指定方法については次ページの【注意】と「７．メッセージリファレンス」を参照して下さい。

　　メッセージの先頭アドレスが NULL の場合、パラメータエラーを返します。

　　自分宛てのメッセージ追加は、プロセス ID 不正エラーを返します。

【C 関数例】
```
int PutMsg(int *msg)
{
    return trapa_svc(0x2e, pid, msg);
}
```

【関連項目】
　　メモリ取得、メモリ解放、メッセージ取得

【注意】

システムプロセスはメッセージのパラメータ長によって以下のようにパラメータの格納・参照します。

パラメータ長が 4 以下の場合：　メッセージのパラメータ部にはパラメータの値が直接格納されます。

パラメータ長が 4 を超える場合：パラメータの値は「メモリ取得」ファンクションで確保した領域に格納され、メッセージのパラメータ部には「メモリ取得」ファンクションで返されたメモリ管理テーブルへのポインタが格納されます。

ユーザプログラム間で独自に定義するメッセージ（コマンドコードが 100～255）の場合、パラメータ長が 4 以下の場合も「メモリ取得」ファンクションで確保した領域にパラメータ値をセットすることは可能です。

「メモリ取得」ファンクションによって確保したパラメータエリアの解放は、「メッセージ追加」ファンクションが正常に終了した場合は宛先のプロセス（「メッセージ取得」ファンクションを呼び出すプロセス）で行います。「メッセージ追加」ファンクションがエラーを返した場合は呼び出し元プロセス（「メッセージ追加」ファンクションを呼び出したプロセス）で解放して下さい。

4.1.16　　メッセージ取得

【機能】
　　システムメッセージキューよりメッセージを取得します。

【入力】
　　トラップ番号　　0x2f
　　コード（R0）　　プロセス ID
　　R4　　メッセージの先頭アドレスを格納する領域のポインタ

【出力】
　　リターンコード　　OK　　正常終了

	+0	+1	+2	+3
R4+00→	自プロセス ID		メッセージ発行元プロセス ID	
+04	メッセージ種別	コマンドコード	パラメータ長	
+08	パラメータの先頭アドレス			

　　ERR_PARAM　　パラメータエラー
　　ERR_PID　　不正プロセス ID

【解説】
　　自プロセス ID 宛てのメッセージをメッセージキューから取得します。
　　該当するメッセージがキューにない場合、プロセス制御の切り替えを行い、メッセージが他のプロセスから送られるまでウェイトします。他に動作中のプロセスがない場合は、空のメッセージを返します。

　　メッセージのアドレスを格納する領域のポインタが NULL の場合、パラメータエラーを返します。

　　システムプロセスからのメッセージについては、「7．メッセージリファレンス」および「メッセージ追加」ファンクションの【注意】を参照して下さい。

【C 関数例】
```
int GetMsg(int *msg)
{
    return trapa_svc(0x2f, pid, msg);
}
```

【関連項目】
　　メモリ取得、メモリ解放、メッセージ追加、指定時間ウェイト

4.1.17　　プログラム実行

【機能】
　　ユーザプログラムを実行します。

【入力】
　　トラップ番号　　0x30
　　コード（R0）　　プロセス ID
　　R4　　起動するプログラム名の先頭アドレス

	+0	+1	+2	+3
R4+00→	起動するプログラム名（13 バイト）			

【出力】

リターンコード	1 以上	起動したプログラムのプロセス ID
	ERR_PARAM	パラメータエラー
	ERR_MEMOVER	メモリ不足のため起動できない
	ERR_PGNOTFOUND	プログラムが見つからない
	ERR_RUNNING	すでに起動されている

【解説】
　　ユーザプログラム格納エリアに格納されているプログラムを起動し、プロセス制御の切り替えを行います。本ファンクションからは、本ファンクション発行元プロセスに制御が戻った場合にリターンします。

　　起動プログラム名のポインタが NULL の場合、パラメータエラーを返します。

　　システムプロセスをユーザプログラムから利用する場合にも、本ファンクションを利用します。システムプロセスとそのプログラム名は次のとおりです。

システムプロセス	プログラム名
FTP サーバプロセス	FtpSv
ソケットクライアントプロセス	SockCl
ソケットサーバプロセス	SockSv
TELNET プロセス	TelnetSv
RS-232C スループロセス	Serial

　　なおシステムプロセスについては「6．システムプロセスについて」を参照して下さい。

【C 関数例】
```
int Exec(char *prog)
{
    return trapa_svc(0x30, pid, prog);
}
```

【関連項目】
　　プログラム停止、プロセス ID 検索

4.1.18　　プログラム停止

【機能】
　　動作中のユーザプログラムを停止します。

【入力】

トラップ番号	0x31
コード（R0）	プロセス ID
R4	停止するプロセスのプロセス ID

【出力】

リターンコード	OK	正常終了
	ERR_PARAM	パラメータエラー
	ERR_NOPROC	プロセスは存在しない
	ERR_NOOTHER	他にプロセスがないため自プロセスの停止不可

【解説】
　　動作中のユーザプログラムを終了し、「メモリ取得」ファンクションで確保したものも含めてメモリを解放します。
　　自プロセス ID を指定することも可能です。

【C 関数例】

```
int Kill(int prcs)
{
    return trapa_svc(0x31, pid, prcs);
}
```

【関連項目】
　　プログラム実行、プロセス ID 検索

4.1.19　　指定時間ウェイト

【機能】
　動作中のユーザプログラムを指定された時間だけウェイトします。

【入力】

トラップ番号	0x32
コード（R0）	プロセス ID
R4	ウェイトする時間（ms）

【出力】

リターンコード	0 以上	残りのウェイト時間
	ERR_PARAM	パラメータエラー

【解説】
　呼び出し元のメッセージキューに未処理のメッセージがない場合、動作中のユーザプログラムを一時停止し、他のプログラムに制御を切り換えます。指定された時間が経過するか、他のプロセスからメッセージを送られたときに、本ファンクションから復帰します。
　メッセージキューに未処理のメッセージがある場合は、そのまま復帰します。

　ウェイトする時間はミリ秒単位で指定します。負の値を指定した場合、パラメータエラーを返します。

【C 関数例】
```
int TimeWait(int tim)
{
    return trapa_svc(0x32, pid, tim);
}
```

【関連項目】
　メッセージ取得

【使用例】
　以下に典型的な使用方法を示します。
```
while(1) {   /* メインループ */
    if(TimeWait(1000) > 0) {
        GetMsg(&msg);
        switch(msg.cmd) {
        case MSG_SYSMSG:
            /* LAN 側のプロセスはソケットのチェックを行う */
            /* シリアル側のプロセスはシリアルからの受信処理を行う */
            /* 外部インタフェースを持たないプロセスは何もしない */
            break;
        case ...:  /* その他必要なメッセージの処理 */
              :
              :
        }
    } else {
        /* タイムアウトした場合の処理 */
    }
}
```

4.1.20　　プロセス ID 検索

【機能】

動作中のユーザプログラムのプロセス ID **を取得します。**

【入力】

トラップ番号	0x33
コード（R0）	**プロセス** ID
R4	**動作中のプログラム名の先頭アドレス**

	+0	+1	+2	+3
R4+00→	**動作中のプログラム名**（13 **バイト**）			

【出力】

リターンコード	1 **以上**	**該当するプロセス** ID
	ERR_PARAM	**パラメータエラー**
	ERR_NOPROC	**プロセスは存在しない**

【解説】

システムのプロセス管理テーブルから、プログラム名称に該当するプロセス ID **を検索し、そのプロセス** ID **を取得します。**

【C 関数例】

```
int GetPid(char *prog)
{
    return trapa_svc(0x33, pid, prog);
}
```

【関連項目】

プログラム実行、プログラム停止

4.1.21　PM 領域消去

【機能】

パーマネントデータメモリ（電源を落としても消えないメモリ）の指定された 1 セクタを消去します。

【入力】

トラップ番号	0x34	
コード（R0）	プロセス ID	
R4	消去するセクタアドレス 0x0e0000 または 0x0f0000	

【出力】

リターンコード	OK	正常終了
	ERR_PARAM	パラメータエラー

【解説】

パーマネントデータメモリ（電源を落としても消えないメモリ）領域は、64 キロバイトを単位としたセクタとして管理されています。本製品はパーマネントデータメモリ領域を 2 セクタ分持っています。

本ファンクションでは、指定されたアドレスからの 1 セクタ（64 キロバイト）分のパーマネントデータメモリを消去します。

セクタアドレスが範囲外の場合、パラメータエラーを返します。

【C 関数例】

```
int PmErase(int sa)
{
    return trapa_svc(0x34, pid, sa);
}
```

【関連項目】

PM 領域書込み、PM 領域読込み

【注意】

本ファンクション実行中は RS-232C の受信オーバランが発生する可能性があるため、RS-232C をクローズするようにしてください。また、本ファンクション実行中に電源 OFF（またはリセット）を行うとメモリの内容が破壊される場合があります。

4.1.22　　PM 領域書込み

【機能】
　パーマネントデータメモリ（電源を落としても消えないメモリ）領域にデータを書き込みます。

【入力】

トラップ番号	0x35	
コード（R0）	プロセス ID	
R4	書込み開始アドレス 0x0e0000〜0xffffe	
R5	書込みデータの先頭アドレス	
R6	書込みデータのワード数	

【出力】

リターンコード	OK	正常終了
	ERR_PARAM	パラメータエラー
	ERR_FLSHWRT	フラッシュ書込みエラー

【解説】
　パーマネントデータメモリ（電源を落としても消えないメモリ）領域は、64 キロバイトを単位としたセクタとして管理されています。本製品はパーマネントデータメモリ領域を 2 セクタ分持っています。
　本ファンクションでは、パーマネントデータメモリの任意のアドレスに、指定されたデータを書き込みます。

　書込みはワード単位で行われます。システムは書き込み結果をチェックし、書き込んだデータと一致しない場合は、フラッシュ書込みエラーを返します。
　また、一度書込みを行ったアドレスには、そのままでは書込みを行うことはできません。同じアドレスに書き込む場合は、「PM 領域消去」ファンクションでセクタをクリアしてから行ってください。

　書込み開始アドレスが範囲外の場合、また書込み領域がセクタ範囲を越える場合
（書込み開始アドレス + 書込みデータのワード数）は、パラメータエラーを返します。
　第 1 セクタのセクタ範囲は 0x0e0000〜0x0effff 、第 2 セクタのセクタ範囲は 0x0f0000〜0x0fffff です。
　また、書込み開始アドレス、書込みデータの先頭アドレスが 2 バイト境界にない場合も、パラメータエラーを返します。

　書込みデータのポインタが NULL の場合、パラメータエラーを返します。

　書込みデータのワード数が 0 の場合、パラメータエラーを返します。

【C 関数例】

```
int PmWrite(int sa, short *data, int size)
{
    return trapa_svc(0x35, pid, sa, data, size);
}
```

【関連項目】

PM 領域消去、PM 領域読込み

【注意】

本ファンクション実行中は RS-232C の受信オーバランが発生する可能性があるため、RS-232C をクローズするようにしてください。また、本ファンクション実行中に電源 OFF（またはリセット）を行うとメモリの内容が破壊される場合があります。

4.1.23 PM 領域読込み

【機能】

パーマネントデータメモリ（電源を落としても消えないメモリ）領域のデータを読込みます。

【入力】

トラップ番号	0x36	
コード（R0）	プロセス ID	
R4	読込み開始アドレス 0x0e0000〜0xffffe	
R5	読込んだデータの格納アドレス	
R6	読込みデータのワード数	

【出力】

リターンコード	OK	正常終了
	ERR_PARAM	パラメータエラー

【解説】

パーマネントデータメモリ（電源を落としても消えないメモリ）領域は、64 キロバイトを単位としたセクタとして管理されています。本製品はパーマネントデータメモリ領域を 2 セクタ分持っています。

本ファンクションでは、パーマネントデータメモリ中の任意のアドレスからデータを読込み、指定されたアドレスに格納します。

読込みはワード単位で行われます。

読込み開始アドレスが範囲外の場合、また読込み領域がセクタ範囲を越える場合（読込み開始アドレス + 読込みデータのワード数）は、パラメータエラーを返します。

第 1 セクタのセクタ範囲は 0x0e0000〜0x0effff 、第 2 セクタのセクタ範囲は 0x0f0000〜0x0fffff です。

また、読込み開始アドレス、格納アドレスが 2 バイト境界にない場合も、パラメータエラーを返します。

読込みデータのポインタが NULL の場合、パラメータエラーを返します。

読込みデータのワード数が 0 の場合、パラメータエラーを返します。

【C 関数例】

```
int PmRead(int sa, short *data, int size)
{
    return trapa_svc(0x36, pid, sa, data, size);
}
```

【関連項目】

PM 領域消去、PM 領域書込み

4.1.24　　プログラム書込み

【機能】
　　ユーザプログラムをプログラム格納エリアに書き込みます。

【入力】
　　トラップ番号　　0x37
　　コード（R0）　　プロセス ID
　　R4　　書き込むプログラムの先頭アドレス

	+0	+1	+2	+3
R4+00→ +04 +08	プログラム名称			
+0c	0	（予約）	バージョン(上位)	バージョン(下位)
+10	プログラムサイズ			
+14	エントリアドレス			
+18 ：	プログラムコード・データ・変数エリア			

【出力】

リターンコード	OK	正常終了
	ERR_PARAM	パラメータエラー
	ERR_FLSHWRT	フラッシュ書込みエラー
	ERR_PGMEMOVER	プログラム格納エリアに空きがない

【解説】
　本製品は、64 キロバイトを一単位としたユーザプログラム格納エリアを 5 個持っており、ユーザプログラムが 64 キロバイト以内であれば、最大 5 本のユーザプログラムを格納することができます。
　本ファンクションは、ネットワーク等外部機器より受け取ったユーザプログラムを、プログラム格納エリアに格納するためのものです。

　書き込むプログラムの先頭アドレスが偶数番地にない場合、パラメータエラーを返します。

　システムはユーザプログラムを格納する際、書き込み結果をチェックし、書き込んだデータと一致しない場合はフラッシュ書込みエラーを返します。

　ユーザプログラム格納エリアの空き容量が、プログラムサイズ（ヘッダ部を含む）に満たない場合もエラーを返します。

【C 関数例】
```
int WriteProgram(void *prog)
{
    return trapa_svc(0x37, pid, prog);
}
```

【関連項目】

プログラム消去

【注意】

本ファンクション実行中は RS-232C の受信オーバランが発生する可能性があるため、RS-232C をクローズするようにしてください。また、本ファンクション実行中に電源 OFF（またはリセット）を行うとメモリの内容が破壊される場合があります。

4.1.25　　プログラム消去

【機能】
　　プログラム格納エリア内のユーザプログラムを消去します。

【入力】
　　トラップ番号　　0x38
　　コード（R0）　　プロセス ID
　　R4　　消去するプログラム名の先頭アドレス

	+0	+1	+2	+3
R4+00→	消去するプログラム名（13 バイト）			

【出力】
　　リターンコード　　OK　　正常終了
　　ERR_PGNOTFOUND　プログラムが存在しない。

【解説】
　　ユーザプログラム格納エリアに格納されているユーザプログラムを消去します。

　　自プログラムの消去も可能です。この場合、格納されている情報のみが消去され、実行中の動作には影響を与えません。

　　指定されたプログラムが存在しない場合は、エラーを返します。

【C 関数例】
```
int EraseProgram(char *prog)
{
    return trapa_svc(0x38, pid, prog);
}
```

【関連項目】
　　プログラム書込み

【注意】
　　本ファンクション実行中は RS-232C の受信オーバランが発生する可能性があるため、RS-232C をクローズするようにしてください。また、本ファンクション実行中に電源 OFF（またはリセット）を行うとメモリの内容が破壊される場合があります。

4.1.26　　リセット

【機能】
　　CPU リセットを行います。

【入力】

トラップ番号	0x39
コード（R0）	プロセス ID
R4	リセット後の動作モード 0 = 通常動作モード 7 = 設定モード

【出力】

リターンコード	なし	正常終了
	ERR_PARAM	パラメータエラー

【解説】
　　CPU のソフトウェアリセットを行います。
　　本ファンクションを利用するとシステムは再起動され、デバイスの全初期化を行った後、システム内に格納されている動作パラメータ中の起動プログラムを実行します。

　　リセット後の動作モードに設定モードを指定すると、設定モードで起動しますが、通常動作モードに戻るためには、本製品の電源を再投入するかリセットを行う必要があります。

　　リセット後の動作モードが範囲外の場合、パラメータエラーを返します。

【C 関数例】

```
int Reset(int sw)
{
    return trapa_svc(0x39, pid, sw);
}
```

4.1.27　　バージョン番号取得

【機能】
　　カーネルのバージョン番号を取得します。

【入力】
　　トラップ番号　　0x3a
　　コード（R0）　　プロセス ID

【出力】
　　リターンコード　　カーネルのバージョン番号

【解説】
　　カーネルのバージョン番号をリターンコードで返します。カーネルのバージョンが 1.12 の場合、0x010C を返します。

【C 関数例】
```
int GetVer(void)
{
    return trapa_svc(0x3a, pid);
}
```

4.1.28 LED 制御

【機能】
LED の点灯状態を変更します。

【入力】

トラップ番号	0x3b	
コード（R0）	プロセス ID	
R4	LED 番号	
	0 = LED1	1 = LED2
	2 = LED3	3 = LED4
R5	点灯状態	
	0 = OFF	1 = ON
	2 = 1 秒周期の点滅	3 = 400ms 周期の点滅

【出力】

リターンコード	なし	正常終了
	ERR_PARAM	パラメータエラー

【解説】
指定された番号の LED の状態を変更します。

LED 番号、点灯状態が範囲外の場合、パラメータエラーを返します。

【C 関数例】
```
int LedCtrl(int ledno, int flg)
{
    return trapa_svc(0x3b, pid, ledno, flg);
}
```

【関連項目】
LED 一括制御

【注意】
本機能は LED 付き製品に有効です。LED 未搭載製品に本ファンクションを発行した場合、正常終了します。

4.1.29　　LED 一括制御

【機能】
　　全ての LED の点灯状態を変更します。

【入力】
　　トラップ番号　　0x3c
　　コード（R0）　　プロセス ID
　　R4　　点灯状態格納域のポインタ

	+0	+1	+2	+3
R4+00→	LED1 の状態	LED2 の状態	LED3 の状態	LED4 の状態

【出力】

リターンコード	なし	正常終了
	ERR_PARAM	パラメータエラー

【解説】
　　4 個の LED の状態を一括して変更します。

　　各 LED の点灯状態が範囲外の場合、パラメータエラーを返します。

【C 関数例】
```
int LedCtrlAll(unsigned char *flgs)
{
    return trapa_svc(0x3c, pid, flgs);
}
```

【関連項目】
　　LED 制御

【注意】
　　本機能は LED 付き製品に有効です。LED 未搭載製品に本ファンクションを発行した場合、正常終了します。

4.1.30　　ユーザパラメータ取得

【機能】
　　システムに格納されているユーザパラメータを取得します。

【入力】

トラップ番号	0x3d
コード（R0）	プロセス ID
R4	ユーザパラメータ格納域の先頭アドレス
R5	ユーザパラメータの識別番号 0〜255
R6	ユーザパラメータ格納域の大きさ 1〜256

【出力】

リターンコード	OK	正常終了
	ERR_PARAM	パラメータエラー
	ERR_NOTFOUND	指定された識別番号のパラメータがない

【解説】
　本製品のフラッシュメモリに格納されているユーザプログラムに固有のパラメータデータを取得します。格納するデータのフォーマットは、使用するユーザプログラムに依存します。
　ユーザプログラムからデータを書き込む場合、次ページの「ユーザパラメータ設定」ファンクションを使用します。
　また、本製品の設定モードで、FTP によりユーザパラメータの取得・変更が可能です。この場合、ファイル名は「USERxxx.CFG」となります。xxx はユーザパラメータの識別番号で、000〜255 のいずれかを指定します。ユーザパラメータの大きさは 256 バイトまでです。

　実際のデータの大きさが格納域の大きさよりも大きい場合、格納可能なだけのデータを返します。

　格納する領域の先頭アドレスが奇数の場合、識別番号が範囲外の場合および格納領域の大きさが 0 以下の場合、パラメータエラーを返します。

【C 関数例】

```
int GetUsrPrm(short *param, int flg, int size)
{
    return trapa_svc(0x3d, pid, param, flg, size);
}
```

【関連項目】
　　ユーザパラメータ設定

4.1.31　　ユーザパラメータ設定

【機能】
　　ユーザパラメータをシステム内に設定します。

【入力】

トラップ番号	0x3e	
コード（R0）	プロセス ID	
R4	設定するユーザパラメータの先頭アドレス	
R5	ユーザパラメータの識別番号 0～255	
R6	ユーザパラメータの大きさ 1～256	

【出力】

リターンコード	OK	正常終了
	ERR_PARAM	パラメータエラー
	ERR_FLSHWRT	フラッシュ書込みエラー

【解説】
　指定された識別番号でユーザプログラムに固有のパラメータデータを本製品のフラッシュメモリに格納します。格納するデータのフォーマットは、使用するユーザプログラムに依存します。
　ユーザプログラムでデータを読み出す場合、前ページの「ユーザパラメータ取得」ファンクションを使用します。
　また、本製品の設定モードで、FTP によりユーザパラメータの取得・変更が可能です。この場合、ファイル名は「USERxxx.CFG」となります。xxx はユーザパラメータの識別番号で、000～255 のいずれかを指定します。

　設定するユーザパラメータの先頭アドレスが奇数または NULL の場合、パラメータエラーを返します。

　識別番号が範囲外の場合、パラメータエラーを返します。

　ユーザパラメータの大きさが範囲外の場合、パラメータエラーを返します。

【C 関数例】

```
int SetUsrPrm(short *param, int flg, int size)
{
    return trapa_svc(0x3e, pid, param, flg, size);
}
```

【関連項目】
　　ユーザパラメータ取得

4.2 RS-232C 機能

4.2.1 RS オープン

【機能】
RS-232C ポートをオープンします。

【入力】
トラップ番号　0x40
コード（R0）　プロセス ID
R4　通信パラメータが格納されている領域の先頭アドレス

	+0	+1	+2	+3
R4+00→	通信速度			
+04	データ長	パリティ	ストップビット	フロー制御

【出力】
リターンコード　OK　正常終了
ERR_PARAM　パラメータエラー
ERR_RSOPENED　オープン済み

	+0	+1	+2	+3
R4+00→	通信速度			
+04	データ長	パリティ	ストップビット	フロー制御

【解説】
指定された通信パラメータで RS-232C ポートをオープンし、RTS および DTR を On にします。
すでにオープンされている場合は、オープン済みとなり、現在の通信パラメータを返します。

通信パラメータは以下の範囲のみ有効とし、範囲外の項目がある場合はパラメータエラーを返します。

パラメータ	設 定 可 能 範 囲
通信速度	110,300,1200,2400,4800,9600,19200,38400,57600 115200,230400,460800,614400,921600
データ長	7 または 8
パリティ	'N'（なし）'E'（偶数）'O'（奇数） ＊本パラメータのみ ASCII コードで指定のこと。
ストップビット	1 または 2
フロー制御	0（なし） 1（RTS/CTS） 2（XON/XOFF）

正常にオープンされた場合、および、パラメータエラーの場合、通信パラメータ格納領域は変更されません。

制御線を変化させたくない場合は、「RS オープン 2」ファンクションを使用して下さい。

【C 関数例】

```
int RsOpen(int *rsparam)
{
    return trapa_svc(0x40, pid, rsparam);
}
```

【関連項目】

RS **オープン** 2

4.2.2 RS クローズ

【機能】
RS-232C ポートをクローズします。

【入力】
トラップ番号	0x41
コード（R0）	プロセス ID

【出力】
リターンコード	OK	正常終了
	ERR_RSCLOSED	オープンされていない

【解説】
RS-232C ポートをクローズし、RTS および DTR を Off にします。
送信バッファにデータが残っている場合、送信が完了するまでブロックされます。

制御線を変化させたくない場合は、「RS クローズ 2」ファンクションを使用して下さい。

【C 関数例】
```
int RsClose(void)
{
    return trapa_svc(0x41, pid);
}
```

【関連項目】
RS クローズ 2

4.2.3 RS 送信

【機能】
RS-232C ポートからデータを送信します。

【入力】

トラップ番号	0x42
コード（R0）	プロセス ID
R4	送信データの先頭アドレス
R5	送信データのバイト数

【出力】

リターンコード	1 以上	送信したバイト数
	ERR_PARAM	パラメータエラー
	ERR_RSCLOSED	オープンされていない
	ERR_RSFULL	送信バッファが一杯

【解説】
送信データをシステム内部の RS-232C 送信バッファに転送し、転送したバイト数を返します。送信バッファのデータは RS-232C ポートより順次出力されます。
渡されたデータの一部しか転送できない場合、転送可能なバイト数だけバッファに転送し、転送したバイト数を返します。

システム内部の送信バッファは 64 キロバイトですので、最大 64 キロバイトの転送が可能です。なお、システムの持つ送信バッファの状態は、「バッファ量取得」ファンクションで確認することができます。

送信データのポインタが NULL の場合、または送信データのバイト数が 0 の場合、パラメータエラーを返します。

送信バッファの空きがなく、1 バイトも格納できない場合、送信バッファが一杯のエラーを返します。

【C 関数例】

```
int RsSend(char *data, int size)
{
    return trapa_svc(0x42, pid, data, size);
}
```

4.2.4 RS 受信

【機能】
RS-232C ポートからデータを受信します。

【入力】
トラップ番号　0x43
コード（R0）　プロセス ID
R4　受信データを格納する領域の先頭アドレス
R5　格納領域のバイト数がセットされている領域のアドレス

	+0	+1	+2	+3
R5+00→	受信データ格納領域のバイト数			

【出力】
リターンコード　OK　正常終了

	+0	+1	+2	+3
R5+00→	格納した受信データバイト数			

ERR_PARAM	パラメータエラー
ERR_RSCLOSED	オープンされていない
ERR_RSORER	オーバランエラー
ERR_RSFER	フレーミングエラー
ERR_RSPER	パリティエラー

【解説】
システム内部の RS-232C 受信バッファよりデータを取り出し、指定された領域に格納すると共に、格納したバイト数を返します。

システム内部の受信バッファは 32 キロバイトですので、最大 32 キロバイトの転送が可能です。なお、システムの持つ受信バッファの状態は、「RS バッファ量取得」ファンクションで確認することができます。

格納エリアが実際のデータ量よりも大きい場合、格納したデータ量を返して直ちに復帰します。（非ブロッキング）

受信データを格納する領域のポインタが NULL の場合、受信データ格納領域のバイト数が 0 の場合、パラメータエラーを返します。

【C 関数例】
```
int RsRecv(char *data, int *size)
{
    return trapa_svc(0x43, pid, data, size);
}
```

【関連項目】
RS バッファ量取得

4.2.5 RS バッファ量取得

【機能】

システム内部の RS-232C 送受信バッファのデータ量を取得します。

【入力】

トラップ番号	0x44
コード（R0）	プロセス ID
R4	受信バッファのデータ量を格納する領域のアドレス
R5	受信バッファの空き容量を格納する領域のアドレス
R6	送信バッファのデータ量を格納する領域のアドレス
R7	送信バッファの空き容量を格納する領域のアドレス

【出力】

リターンコード　OK　　正常終了

	+0	+1	+2	+3
R4+00→	受信バッファのデータ量			

	+0	+1	+2	+3
R5+00→	受信バッファの空き容量			

	+0	+1	+2	+3
R6+00→	送信バッファのデータ量			

	+0	+1	+2	+3
R7+00→	送信バッファの空き容量			

【解説】

システム内部に持つ RS-232C 送受信バッファの、現在の状態をそれぞれ格納します。
格納する領域のポインタが NULL の場合、その該当項目はセットされません。

なお、システム内部に持つ RS-232C 送受信バッファの容量は、
　送信バッファ　32 キロバイト
　受信バッファ　32 キロバイト
で、それぞれリングバッファ方式で管理されています。

【C 関数例】

```
int RsGetBuff(int *rvex, int *rvfr, int *sdex, int *sdfr)
{
    return trapa_svc(0x44, pid, rvex, rvfr, sdex, sdfr);
}
```

【関連項目】

RS 受信バッファフラッシュ、RS 送信バッファフラッシュ

4.2.6 RS 受信バッファフラッシュ

【機能】
システム内部の RS-232C 受信バッファをクリアします。

【入力】
トラップ番号　　0x45
コード（R0）　　プロセス ID

【出力】
なし

【解説】
システム内部に持つ RS-232C 受信バッファのデータを全て破棄し、バッファを空にします。
フロー制御が有効な場合、受信可能状態になります。

【C 関数例】
```
int RsFlushRecv(void)
{
    return trapa_svc(0x45, pid);
}
```

【関連項目】
RS バッファ量取得、RS 送信バッファフラッシュ

4.2.7 RS 送信バッファフラッシュ

【機能】
　　システム内部の RS-232C 送信バッファをクリアします。

【入力】
　　トラップ番号　　0x46
　　コード（R0）　　プロセス ID

【出力】
　　なし

【解説】
　　システム内部に持つ RS-232C 送信バッファのデータを全て破棄し、バッファを空にします。

　　本ファンクションにより、データ送信を強制的に中止することができます。

【C 関数例】
```
int RsFlushSend(void)
{
    return trapa_svc(0x46, pid);
}
```

【関連項目】
　　RS バッファ量取得、RS 受信バッファフラッシュ

4.2.8 制御線状態取得

【機能】
RS-232C 制御線の状態を取得します。

【入力】
トラップ番号　0x47
コード（R0）　プロセス ID

【出力】
リターンコード　0x00～0x7f　制御状態（以下の論理和データ）

0x01	RTS On
0x02	DTR On
0x04	DSR On
0x08	CTS On
0x10	DCD On
0x20	送信停止中（Xon/Xoff フロー制御時のみ）
0x40	受信停止中（Xon/Xoff フロー制御時のみ）

DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0 固定	受信状態	送信状態	DCD	CTS	DSR	DTR	RTS

【解説】
RS-232C 制御線の状態を取得します。
Xon/Xoff フロー制御でオープンされている場合、そのフロー制御状態は DB6、DB5 にセットされます。

【C 関数例】
```
int RsGetCtrl(void)
{
    return trapa_svc(0x47, pid);
}
```

【関連項目】
制御線状態設定

4.2.9 制御線状態設定

【機能】

RS-232C 制御線の状態を設定します。

【入力】

トラップ番号	0x48
コード（R0）	プロセス ID
R4	制御線の種別 0 = RTS 1 = DTR
R5	設定する状態 0 = OFF 1 = ON

【出力】

リターンコード	OK	正常終了
	ERR_PARAM	パラメータエラー

【解説】

RS-232C の制御線を、指定された状態に設定します。

制御線の種別・設定する状態が範囲外の場合、パラメータエラーを返します。

【C 関数例】

```
int RsSetCtrl(int ctrl, int sw)
{
    return trapa_svc(0x48, pid, ctrl, sw);
}
```

【関連項目】

制御線状態取得

4.2.10 RS オープン 2

【機能】

RS-232C ポートをオープンします。

【入力】

トラップ番号　0x49
コード（R0）　プロセス ID
R4　通信パラメータが格納されている領域の先頭アドレス

	+0	+1	+2	+3
R4+00→	通信速度			
+04	データ長	パリティ	ストップビット	フロー制御

【出力】

リターンコード　OK　正常終了
ERR_PARAM　パラメータエラー
ERR_RSOPENED　オープン済み

	+0	+1	+2	+3
R4+00→	通信速度			
+04	データ長	パリティ	ストップビット	フロー制御

【解説】

指定された通信パラメータで RS-232C ポートをオープンします。RTS および DTR の状態は処理実行前の状態を保持します。
すでにオープンされている場合は、オープン済みとなり、現在の通信パラメータを返します。

通信パラメータは以下の範囲のみ有効とし、範囲外の項目がある場合はパラメータエラーを返します。

パラメータ	設　定　可　能　範　囲
通信速度	110,300,1200,2400,4800,9600,19200,38400,57600 115200,230400,460800,614400,921600
データ長	7 または 8
パリティ	'N'（なし）'E'（偶数）'O'（奇数） ＊本パラメータのみ ASCII コードで指定のこと。
ストップビット	1 または 2
フロー制御	0（なし）　1（RTS/CTS）　2（XON/XOFF）

正常にオープンされた場合、および、パラメータエラーの場合、通信パラメータ格納領域は変更されません。

【C 関数例】

```
int RsOpenX(int *rsparam)
{
    return trapa_svc(0x49, pid, rsparam);
}
```

【関連項目】

RS **オープン**

【注意】

制御線の状態が変化しないことを除いて、「RS **オープン」ファンクションと同じです。**

4.2.11　RS クローズ 2

【機能】
RS-232C ポートをクローズします。

【入力】
トラップ番号　0x4a
コード（R0）　プロセス ID

【出力】

リターンコード	OK	正常終了
	ERR_RSCLOSED	オープンされていない

【解説】
RS-232C ポートをクローズします。RTS および DTR の状態は処理実行前の状態を保持します。
送信バッファにデータが残っている場合、送信が完了するまでブロックされます。

【C 関数例】
```
int RsCloseX(void)
{
    return trapa_svc(0x4a, pid);
}
```

【関連項目】
RS クローズ

【注意】
制御線の状態が変化しないことを除いて、「RS クローズ」ファンクションと同じです。

4.2.12　ブレーク送信

【機能】
RS-232C ポートからブレーク信号を送信します。

【入力】

トラップ番号	0x4b	
コード（R0）	プロセス ID	
R4	ブレーク送信時間（ms）	
	1～2000	

【出力】

リターンコード	OK	正常終了
	ERR_PARAM	パラメータエラー
	ERR_RSSDEXIST	データ送信中

【解説】
RS-232C ポートからブレーク信号を、指定された時間送信します。

ブレーク送信時間は、タイマファンクションと同じタイマを使用して制御します。そのため、送信時間と実時間との間には±1ms 程度の誤差があります。
ブレーク送信中は、ユーザプログラムに制御は戻りません。（ブロッキング処理）

ブレーク送信時間はミリ秒単位で指定します。範囲外の値を指定した場合、パラメータエラーを返します。

RS-232C 送信バッファにデータがある場合、ブレーク信号を送信せずにデータ送信中エラーを返します。

【C 関数例】

```
int RsSetBreak(int tim)
{
    return trapa_svc(0x4b, pid, tim);
}
```

【関連項目】
タイマ開始、RS 送信バッファフラッシュ

4.3 ネットワーク機能

4.3.1 LAN オープン

【機能】
ネットワークインタフェースを初期化します。

【入力】

トラップ番号	0x50	
コード（R0）	プロセス ID	

【出力】

リターンコード	OK	正常終了
	ERR_PID	不正プロセス ID
	ERR_LANOPENED	LAN オープン済み

【解説】
ネットワークインタフェースをリセットし、ソケット関係の資源を全て初期化します。

【C 関数例】

```
int LanOpen(void)
{
    return trapa_svc(0x50, pid);
}
```

4.3.2 LAN クローズ

【機能】
　　ネットワークインタフェースをクローズします。

【入力】
　　トラップ番号　　0x51
　　コード（R0）　　プロセス ID

【出力】

リターンコード	OK	正常終了
	ERR_PID	不正プロセス ID
	ERR_INUSE	他のプロセスが LAN を使用中
	ERR_LANCLOSED	LAN 未オープン

【解説】
　　ネットワークインタフェースをクローズし、ソケット関係の資源を解放します。

　　他のプロセスが LAN を使用している場合、全てのソケット関係の資源は保護され、ネットワークインタフェースのクローズ処理は行われません。

【C 関数例】

```
int LanClose(void)
{
    return trapa_svc(0x51, pid);
}
```

4.3.3 ソケット作成

【機能】
　　ソケットを作成します。

【入力】

トラップ番号	0x52
コード（R0）	プロセス ID
R4	ソケットの種別 0 = TCP 1 = UDP

【出力】

リターンコード	1 以上	作成したソケットの番号
	ERR_PARAM	パラメータエラー
	ERR_PID	不正プロセス ID
	ERR_MEMOVER	メモリ不足
	ERR_LANCLOSED	LAN 未オープン
	ERR_NOBUFS	これ以上ソケットを作成できない

【解説】
　　新たにソケットを作成し、その番号を返します。
　　本製品では、最大 40 個のソケットを作成することができます。

　　ソケット種別に 0 または 1 以外を指定した場合、パラメータエラーを返します。

【C 関数例】

```
int Socket(int flg)
{
    return trapa_svc(0x52, pid, flg);
}
```

4.3.4 ソケットの名前付け

【機能】
ソケットにアドレスを割り付けます。

【入力】

トラップ番号	0x53
コード（R0）	プロセス ID
R4	ソケット番号
R5	IP アドレス 0.0.0.1～255.255.255.254
R6	ポート番号 0～65535

【出力】

リターンコード	OK	正常終了
	ERR_PARAM	パラメータエラー
	ERR_PID	不正プロセス ID
	ERR_LANCLOSED	LAN 未オープン
	ERR_NOSOCKET	ソケット番号が無効
	ERR_BOUND	ソケットはバインド済み
	ERR_ADDRINUSE	指定のアドレスは他で使用中

【解説】
ソケットにアドレスを割り付け、自アドレスを決定します。

IP アドレスとポート番号が範囲外の場合、パラメータエラーを返します。

【C 関数例】

```
int Bind(int sck, unsigned long ip, unsigned short port)
{
    return trapa_svc(0x53, pid, sck, ip, port);
}
```

4.3.5 クライアント接続開始

【機能】
　　指定された接続先にクライアント接続を行います。

【入力】

トラップ番号	0x54
コード（R0）	プロセス ID
R4	ソケット番号
R5	接続先の IP アドレス 0.0.0.1～255.255.255.254
R6	接続先のポート番号 0～65535
R7	ブロッキングフラグ 0 = 接続要求に対する応答を待たない 1 = 接続要求に対する応答を待つ

【出力】

リターンコード	OK	正常終了
	ERR_PARAM	パラメータエラー
	ERR_TIMEDOUT	接続タイムアウト
	ERR_PID	不正プロセス ID
	ERR_LANCLOSED	LAN 未オープン
	ERR_NOSOCKET	ソケット番号が無効
	ERR_NOTTCP	TCP ソケットでない
	ERR_OPENED	ソケットはすでに接続されている
	ERR_NETUNREACH	接続先のノードに到達できない
	ERR_CONNREFUSED	接続が拒否された

【解説】
　　指定された接続先にクライアント接続を行います。
　　ブロッキングする（接続要求に対する応答を待つ）場合、動作パラメータのコネクトタイマで指定された時間が経過するまで、接続要求に対する応答を待ちます。
　　非ブロッキングの場合、「クライアント接続確認」ファンクションで接続結果を確認して下さい。

　　接続先の IP アドレスとポート番号が範囲外の場合、パラメータエラーを返します。

【C 関数例】

```
int Connect(int sck, unsigned long ip, unsigned short port, int flg)
{
    return trapa_svc(0x54, pid, sck, ip, port, flg);
}
```

【関連項目】
　　クライアント接続確認

4.3.6 クライアント接続確認

【機能】
　　クライアント接続の結果を取得します。

【入力】

トラップ番号	0x55
コード（R0）	プロセス ID
R4	ソケット番号

【出力】

リターンコード	OK	正常終了
	ERR_TIMEDOUT	接続タイムアウト
	ERR_PID	不正プロセス ID
	ERR_LANCLOSED	LAN 未オープン
	ERR_NOSOCKET	ソケット番号が無効
	ERR_NOTTCP	TCP ソケットでない
	ERR_NETUNREACH	接続先のノードに到達できない
	ERR_CONNREFUSED	接続が拒否された
	ERR_CONNECTING	接続処理中

【解説】
　　接続要求中のソケットに対し、接続が完了しているか否かを取得します。「クライアント接続開始」ファンクションで、ブロッキングフラグを「応答を待たない」指定にした場合、本ファンクションで接続結果を確認して下さい。
　　ソケットが接続状態になっていた場合に、正常終了します。

　　接続が失敗した場合、要因（タイムアウト、到達不可、接続拒否のいずれか）を返します。

　　接続状態または接続要求中以外のソケットを指定した場合、ソケット番号無効エラーを返します。

【C 関数例】

```
int ConnectRes(int sck)
{
    return trapa_svc(0x55, pid, sck);
}
```

【関連項目】
　　クライアント接続要求

【注意】
　　本ファンクションが接続処理中を返す間は、接続処理が完了していませんので、再度本ファンクションで接続結果を確認して下さい。

4.3.7 接続要求受付開始

【機能】
　　指定ソケットに対する、接続要求の受付を可能にします。

【入力】

トラップ番号	0x56
コード（R0）	プロセス ID
R4	ソケット番号
R5	接続要求受付保留キューの最大数 1〜15

【出力】

リターンコード	OK	正常終了
	ERR_PARAM	パラメータエラー
	ERR_PID	不正プロセス ID
	ERR_LANCLOSED	LAN 未オープン
	ERR_NOSOCKET	ソケット番号が無効
	ERR_NOTTCP	TCP ソケットでない
	ERR_NOTBOUND	ソケットはバインドされていない
	ERR_OPENED	ソケットはすでに接続されている
	ERR_LISTENING	ソケットはすでに接続要求を受付可能

【解説】
　　指定ソケットに対し、接続要求を受付可能な状態にします。

　　指定ソケットに対する接続要求を受信した場合、「接続要求受付保留」状態として、キューへの蓄積を行います。
　　接続キューの最大数が範囲外の場合、パラメータエラーを返します。

　　「接続要求受付」ファンクションを呼び出すことにより、データの送受信が可能になります。

【C 関数例】

```
int Listen(int sck, int backlog)
{
    return trapa_svc(0x56, pid, sck, backlog);
}
```

【関連項目】
　　接続要求受付

4.3.8 接続要求受付

【機能】
　　指定ソケットに対する接続を受け付けます。

【入力】

トラップ番号	0x57
コード（R0）	プロセス ID
R4	接続要求受付用ソケットの番号
R5	接続要求元の IP アドレスを格納する領域のアドレス
R6	接続要求元のポート番号を格納する領域のアドレス

【出力】

リターンコード	1 以上	接続されたソケットの番号

	+0	+1	+2	+3
R5+00→	接続要求元の IP アドレス			

	+0	+1
R6+00→	接続要求元のポート番号	

0	保留中の接続要求がない
ERR_PARAM	パラメータエラー
ERR_PID	不正プロセス ID
ERR_LANCLOSED	LAN 未オープン
ERR_NOBUFS	これ以上ソケットを作成できない
ERR_NOSOCKET	ソケット番号が無効
ERR_NOTTCP	TCP ソケットでない
ERR_NOTBOUND	ソケットはバインドされていない
ERR_OPENED	ソケットはすでに接続されている
ERR_MFILE	ソケットは接続要求を受け付ける状態にない

【解説】
　　「接続要求受付開始」ファンクションで接続要求待ちになっているソケットを指定します。
　　指定ソケットに対しての接続要求に応じて接続ソケットを作成し、接続要求元の IP アドレス・ポート番号を返します。
　　接続要求を受信していない場合は、直ちに 0 を返します。（非ブロッキング）

　　接続要求元の IP アドレス・ポート番号を格納する領域のポインタが NULL の場合、パラメータエラーを返します。

【C 関数例】

```
int Accept(int sck, unsigned long *ip, unsigned short *port)
{
    return trapa_svc(0x57, pid, sck, ip, port);
}
```

【関連項目】
　　接続要求受付開始

4.3.9 ストリーム送信

【機能】
　　指定ソケットに接続されている宛先に対し、指定データを TCP 送信します。

【入力】

トラップ番号	0x58
コード（R0）	プロセス ID
R4	ソケット番号
R5	送信データの先頭アドレス
R6	送信データのバイト数

【出力】

リターンコード	0 以上	送信バイト数
	ERR_PARAM	パラメータエラー
	ERR_PID	不正プロセス ID
	ERR_LANCLOSED	LAN 未オープン
	ERR_NOSOCKET	ソケット番号が無効
	ERR_NOTTCP	TCP ソケットでない
	ERR_NOTBOUND	ソケットはバインドされていない
	ERR_NOTCONN	ソケットは接続されていない

【解説】
　　指定された送信データを、システム内部の TCP ソケット送信バッファに格納します。
　　渡されたデータに対して送信バッファの空きが十分にない場合、格納可能なバイト数だけバッファに格納し、格納したバイト数を返します。

　　システムは、TCP ソケット送信バッファに格納された送信データを含む TCP パケットを作成し、指定されたソケットに接続されている宛先に対して、パケット送信します。

　　送信データのポインタが NULL の場合、または送信データのバイト数が 0 の場合、パラメータエラーを返します。

【C 関数例】

```
int Send(int sck, char *data, int size)
{
    return trapa_svc(0x58, pid, sck, data, size);
}
```

4.3.10　　パケット送信

【機能】
　　指定された宛先に対しデータパケットを UDP 送信します。

【入力】

トラップ番号	0x59
コード（R0）	プロセス ID
R4	ソケット番号
R5	送信データの先頭アドレス
R6	送信データのバイト数
R7	送信先のアドレス情報の先頭アドレス

	+0	+1	+2	+3
R7+00→	送信先の IP アドレス			
+04	送信先のポート番号			

【出力】

リターンコード	0 以上	送信バイト数
	ERR_PARAM	パラメータエラー
	ERR_PID	不正プロセス ID
	ERR_MEMOVER	メモリ不足
	ERR_LANCLOSED	LAN 未オープン
	ERR_NOSOCKET	ソケット番号が無効
	ERR_NOTUDP	UDP ソケットでない
	ERR_NOTBOUND	ソケットはバインドされていない

【解説】
　　指定された送信データを、システム内部の UDP ソケット送信パケットキューに追加します。送信可能なデータの最大サイズは 5094 バイトです。
　　システムは指定された送信データを含む UDP パケットを作成し、UDP ソケット送信パケットキューに格納します。その後システムは、送信キューに格納されているパケットを順次送信します。
　　なお、送信先の IP アドレスにブロードキャストアドレスが指定された場合、システムはブロードキャスト送信を行います。

　　送信データのポインタが NULL の場合、または送信データのバイト数が 0 の場合、パラメータエラーを返します。

　　送信先のアドレス情報のポインタが NULL の場合、パラメータエラーを返します。

【C 関数例】

```
int SendTo(int sck, char *data, int size, int *to)
{
    return trapa_svc(0x59, pid, sck, data, size, to);
}
```

4.3.11　　ストリーム受信

【機能】
　　指定ソケットに対するデータを TCP 受信します。

【入力】

トラップ番号	0x5a
コード（R0）	プロセス ID
R4	ソケット番号
R5	受信データ格納域の先頭アドレス
R6	受信データ格納域の総バイト数

【出力】

リターンコード	0 以上	受信データバイト数
	ERR_PARAM	パラメータエラー
	ERR_PID	不正プロセス ID
	ERR_LANCLOSED	LAN 未オープン
	ERR_NOSOCKET	ソケット番号が無効
	ERR_NOTTCP	TCP ソケットでない
	ERR_NOTBOUND	ソケットはバインドされていない
	ERR_NOTCONN	ソケットは接続されていない

【解説】
　　システムが受信した指定ソケットに対する受信データを、指定されたアドレスに格納します。

　　格納エリアが実際のデータ量よりも大きい場合、格納したデータ量を返して直ちに復帰します。（非ブロッキング）

　　受信データを格納する領域のポインタが NULL の場合、またはデータを格納する領域の総バイト数が 0 の場合、パラメータエラーを返します。

【C 関数例】

```
int Recv(int sck, char *data, int size)
{
    return trapa_svc(0x5a, pid, sck, data, size);
}
```

4.3.12　　パケット受信

【機能】
　　指定ソケットに対するデータパケットを UDP 受信します。

【入力】

トラップ番号	0x5b
コード（R0）	プロセス ID
R4	ソケット番号
R5	受信データを格納する領域の先頭アドレス
R6	受信データを格納する領域のバイト数
R7	送信元のアドレス情報を格納する領域の先頭アドレス

【出力】

リターンコード	0 以上	格納した受信データのバイト数

	+0	+1	+2	+3
R7+00→	送信元の IP アドレス			
+04	送信元のポート番号		パケット種別	

パケット種別	0	ブロードキャスト・パケットを除く UDP パケット
	1	ブロードキャスト・パケット

ERR_PARAM	パラメータエラー
ERR_PID	不正プロセス ID
ERR_LANCLOSED	LAN 未オープン
ERR_NOSOCKET	ソケット番号が無効
ERR_NOTUDP	UDP ソケットでない
ERR_NOTBOUND	ソケットはバインドされていない

【解説】
　指定された UDP ソケットに対する受信パケットデータを、指定されたアドレスに格納し、送信元のアドレス情報・受信パケット種別と共に返します。
　受信したパケットがブロードキャスト・パケットの場合には、パケット種別に１が返されます。

　受信したパケットデータ数が指定格納領域のバイト数よりも大きい場合、指定バイト数を超える受信データは破棄されます。
　受信可能なパケットの最大サイズは、5094 バイトです。

　受信パケットがない場合、リターンコード 0 で直ちに復帰します。（非ブロッキング）

　受信データを格納する領域・送信元のアドレス情報を格納する領域のポインタが NULL の場合、または受信データを格納する領域のバイト数が 0 の場合、パラメータエラーを返します。

【C 関数例】
```
int RecvFrom(int sck, char *data, int size, int *from)
{
    return trapa_svc(0x5b, pid, sck, data, size, from);
}
```

4.3.13　　ソケット送受信バッファ量取得

【機能】
　　指定ソケットの送受信バッファ容量を取得します。

【入力】

トラップ番号	0x5c
コード（R0）	プロセス ID
R4	ソケット番号
R5	送受信バッファ区分 0 = 送信バッファ 1 = 受信バッファ
R6	バッファのデータ量を格納する領域のアドレス(TCP) パケット数を格納する領域のアドレス(UDP) NULL 可
R7	バッファの空き容量を格納する領域のアドレス NULL 可

【出力】
　　リターンコード　OK　　正常終了

	+0	+1	+2	+3
R6+00→	バッファのデータ量（TCP）／バッファのパケット数（UDP）			

	+0	+1	+2	+3
R7+00→	バッファの空き容量（TCP のみ）			

ERR_PARAM	パラメータエラー
ERR_PID	不正プロセス ID
ERR_LANCLOSED	LAN 未オープン
ERR_NOSOCKET	ソケット番号が無効
ERR_NOTUDP	UDP ソケットでない
ERR_NOTBOUND	ソケットはバインドされていない
ERR_NOTCONN	ソケットは接続されていない

【解説】
　　指定されたソケットが TCP ソケットの場合、送受信バッファ中の未処理データバイト数と、空き容量を返します。送受信バッファの総バイト数はそれぞれ 4096 です。
　　指定されたソケットが UDP ソケットの場合は、未処理パケット数のみを返します。

　　バッファのデータ量・空き容量を格納する領域のポインタが NULL の場合、値はセットされません。

【C 関数例】

```
int LanGetBuff(int sck, int flg, int *exist, int *fre)
{
    return trapa_svc(0x5c, pid, sck, flg, exist, fre);
}
```

4.3.14　　イベントチェック

【機能】
　　ソケットに対する受信イベントの状態を取得します。

【入力】
　　トラップ番号　　0x5d
　　コード（R0）　　プロセス ID
　　R4　　ソケット番号の最大値
　　R5　　状態を取得するソケットのリストを表す、バイト配列の先頭アドレス

	+0	+1	+2	+3
R5+00→	ソケット 1 状態取得可否	ソケット 2 状態取得可否	ソケット 3 状態取得可否	ソケット 4 状態取得可否
+04 ：	ソケット 5 状態取得可否	：	：	：

【出力】
　　リターンコード　　0 以上　　受信イベントのあるソケットの個数

	+0	+1	+2	+3
R5+00→	ソケット 1 イベントの有無	ソケット 2 イベントの有無	ソケット 3 イベントの有無	ソケット 4 イベントの有無
+04 ：	ソケット 5 イベントの有無	：	：	：

ERR_PARAM	パラメータエラー
ERR_PID	不正プロセス ID
ERR_LANCLOSED	LAN 未オープン

【解説】
　　バイト配列にはソケット番号をインデックスとし、受信イベントの状態を取得するソケットに 1 を、取得しないソケットには 0 をあらかじめセットして下さい。

　　本ファンクションでは 1 がセットされたソケットに対し、イベント有無をチェックしてイベントがない場合に 0 をセットします。

　　接続要求受付中のソケットの場合、「接続要求受信」のイベントをチェックします。

　　クライアント接続を非ブロッキングで行ったソケットの場合、「接続要求結果受信」のイベントをチェックします。

　　接続状態のソケットの場合、「データ受信」・「切断要求受信」・「強制切断」のイベントをチェックします。イベントの種別は続けて「ソケット送受信バッファ量取得」ファンクションを呼び出すことにより判定して下さい。

　　UDP ソケットの場合、「データ受信」のイベントをチェックします。

　　上記以外の状態のソケットの場合、イベントは発生しません。

【C 関数例】

```
int Select(int cnt, unsigned char *fdset)
{
    return trapa_svc(0x5d, pid, cnt, fdset);
}
```

【関連項目】

ソケット送受信バッファ量取得

【使用例】

以下は TCP 接続状態のソケットのイベントを調べる例です。

```
fdset[sck-1] = 1;  /* 調べるソケットのマーク */
if(Select(sck, fdset) > 0) {
    /* バッファ量取得 */
    if(LanGetBuff(sck, 1, &cnt, NULL) != OK) {
        /* 強制切断 */
        Close(sck);
    } else {
        if(cnt == 0) {
            /* 切断要求受信 */
            Close(sck);
        } else {
            /* データ受信 */
            Recv(sck, data, cnt);
        }
    }
}
```

4.3.15　　ソケットクローズ

【機能】
　　指定ソケットをクローズします。

【入力】

トラップ番号	0x5e
コード（R0）	プロセス ID
R4	ソケット番号

【出力】

リターンコード	OK	正常終了
	ERR_PID	不正プロセス ID
	ERR_LANCLOSED	LAN 未オープン
	ERR_NOSOCKET	ソケット番号が無効
	ERR_EXISTDATA	バッファに未処理データが存在する

【解説】
　　ソケットの送受信バッファが空の場合、ソケットをクローズします。

　　指定ソケットが UDP ソケットの場合、送受信バッファに未処理データが残っていても、指定ソケットをクローズしデータを破棄します。

【C 関数例】

```
int Close(int sck)
{
    return trapa_svc(0x5e, pid, sck);
}
```

4.3.16　　ソケット強制クローズ

【機能】
　　指定ソケットを強制的にクローズします。

【入力】

トラップ番号	0x5f
コード（R0）	プロセス ID
R4	ソケット番号

【出力】

リターンコード	OK	正常終了
	ERR_PID	不正プロセス ID
	ERR_LANCLOSED	LAN 未オープン
	ERR_NOSOCKET	ソケット番号が無効

【解説】
　　指定ソケットの送受信バッファに残っているデータを破棄し、ソケットをクローズします。

【C 関数例】

```
int Abort(int sck)
{
    return trapa_svc(0x5f, pid, sck);
}
```

4.3.17　　LAN 受信バッファクリア

【機能】
　　指定ソケットの受信バッファをクリアします。

【入力】

トラップ番号	0x60
コード（R0）	プロセス ID
R4	ソケット番号

【出力】

リターンコード	OK	正常終了
	ERR_PID	不正プロセス ID
	ERR_LANCLOSED	LAN 未オープン
	ERR_NOSOCKET	ソケット番号が無効

【解説】
　　指定ソケットが TCP の場合、ソケットに対応する受信バッファ内のデータをクリアします。

　　指定ソケットが UDP の場合は、ソケットに対応する受信パケットをクリアします。

【C 関数例】
```
int FlushRecvBuff(int sck)
{
    return trapa_svc(0x60, pid, sck);
}
```

4.3.18　　LAN 送信バッファクリア

【機能】
　　指定ソケットの送信バッファをクリアします。

【入力】

トラップ番号	0x61
コード（R0）	プロセス ID
R4	ソケット番号

【出力】

リターンコード	OK	正常終了
	ERR_PID	不正プロセス ID
	ERR_LANCLOSED	LAN 未オープン
	ERR_NOSOCKET	ソケット番号が無効

【解説】
　　指定ソケットが TCP の場合、ソケットに対応する送信バッファのデータの内、応答待ち以外のデータをクリアします。

　　指定ソケットが UDP の場合は、ソケットに対応する未送信パケットをクリアします。

【C 関数例】

```
int FlushSendBuff(int sck)
{
    return trapa_svc(0x61, pid, sck);
}
```

4.3.19 ARP 送信

【機能】

アドレスリゾリューションプロトコル（ARP）パケットを送信し、指定 IP アドレスに対応する MAC アドレスを取得します。

【入力】

トラップ番号	0x62
コード（R0）	プロセス ID
R4	送信先 IP アドレス
R5	取得 MAC アドレスを格納する領域の先頭アドレス NULL 可

【出力】

リターンコード　OK　正常終了

	+0	+1	+2	+3
R5+00→	応答 MAC アドレス			
+04	応答 MAC アドレス（つづき）			

ERR_PARAM	パラメータエラー
ERR_TIMEDOUT	応答タイムアウト
ERR_PID	不正プロセス ID
ERR_MEMOVER	メモリ不足
ERR_LANCLOSED	LAN 未オープン
ERR_NODEST	宛先なし

【解説】

ARP（アドレスリゾリューションプロトコル）パケットを送信し、指定 IP アドレスに対応する MAC アドレスを取得します。

送信先 IP アドレスが 0.0.0.0 または 255.255.255.255 の場合、パラメータエラーを返します。

MAC アドレスを格納する領域は 6 バイト以上用意して下さい。
MAC アドレスを格納する領域のポインタが NULL の場合、MAC アドレスの格納は行いません。

【C 関数例】

```
int Arp(unsigned long ip, unsigned char *mac)
{
    return trapa_svc(0x62, pid, ip, mac);
}
```

4.3.20　RARP 送信

【機能】

リバースアドレスリゾリューションプロトコル（RARP）パケットを送信し、指定 MAC アドレスに対応する IP アドレスを取得します。

【入力】

トラップ番号　0x63
コード（R0）　プロセス ID
R4　MAC アドレスが格納されている領域の先頭アドレス

	+0	+1	+2	+3
R4+00→	MAC アドレス			
+04	MAC アドレス（つづき）			

R5　取得した IP アドレスを格納する領域の先頭アドレス
NULL 可

【出力】

リターンコード　OK　正常終了

	+0	+1	+2	+3
R5+00→	取得 IP アドレス			

ERR_PARAM	パラメータエラー
ERR_TIMEDOUT	応答タイムアウト
ERR_PID	不正プロセス ID
ERR_MEMOVER	メモリ不足
ERR_LANCLOSED	LAN 未オープン
ERR_NODEST	宛先なし

【解説】

RARP（リバースアドレスリゾリューションプロトコル）パケットを送信し、指定 MAC アドレスに対応する IP アドレスを取得します。

MAC アドレスが FF:FF:FF:FF:FF:FF の場合、または MAC アドレスのポインタが NULL の場合、パラメータエラーを返します。

取得した IP アドレスを格納する領域のポインタが 4 バイト境界のアドレスを指していない場合、パラメータエラーを返します。
また、取得した IP アドレスを格納する領域のポインタが NULL の場合、取得した IP アドレスの格納は行いません。

【C 関数例】

```
int Rarp(unsigned char *mac, unsigned long *ip)
{
    return trapa_svc(0x63, pid, mac, ip);
}
```

4.3.21　　ICMP 送信

【機能】

　インターネット・コントロール・メッセージ・プロトコル（ICMP）のエコーリクエストパケットを送信します。

【入力】

トラップ番号	0x64
コード（R0）	プロセス ID
R4	送信先 IP アドレス
R5	タイムアウト時間（秒）

【出力】

リターンコード	0 以上	応答時間（単位：10ms）
	ERR_PARAM	パラメータエラー
	ERR_TIMEDOUT	応答タイムアウト
	ERR_PID	不正プロセス ID
	ERR_MEMOVER	メモリ不足
	ERR_LANCLOSED	LAN 未オープン

【解説】

　インターネット・コントロール・メッセージ・プロトコル（ICMP）のエコーリクエストパケットを送信します（ping コマンドの実行）。応答を受けた場合、パケット送信から応答までの時間を 10ms を 1 単位とした値で返します。

　送信先 IP アドレスが 0.0.0.0、または 255.255.255.255 の場合、パラメータエラーを返します。

　指定タイムアウト時間は 1～60 秒とし、範囲外の場合は 5 秒とします。

【C 関数例】

```
int Icmp(unsigned long ip, int tmout)
{
    return trapa_svc(0x64, pid, ip, tmout);
}
```

4.3.22　DHCP 要求送信

【機能】

ダイナミック・ホスト・コンフィグレーション・プロトコル（DHCP）のリクエストを送信し、IP アドレスを取得します。

【入力】

トラップ番号	0x65
コード（R0）	プロセス ID
R4	取得した IP アドレスを格納する領域の先頭アドレス NULL 可
R5	タイムアウト時間（ms）
R6	リトライ回数

【出力】

リターンコード　OK　　正常終了

	+0	+1	+2	+3
R4+00→	取得 IP アドレス			
+04	デフォルトゲートウェイ			
+08	サブネットマスク			

ERR_PARAM	パラメータエラー
ERR_PID	不正プロセス ID
ERR_LANCLOSED	LAN 未オープン
ERR_DP_DISOUT	DHCP サーバが存在しない
ERR_DP_RQOUT	応答タイムアウト

【解説】

DHCP 要求を送信し、応答パケットから IP アドレス情報を取得します。

正常に応答を受信した場合、自 IP アドレスを動作パラメータの自 IP アドレス領域にセットすると共に、本ファンクションの出力パラメータにセットして返します。

また、サブネットマスクおよびデフォルトゲートウェイが応答パケットに設定されている場合、これらを動作パラメータのそれぞれの領域にセットすると共に、本ファンクションの出力パラメータにセットして返します。

取得した IP アドレスを格納する領域のポインタが NULL の場合、取得した IP アドレスは動作パラメータへの設定のみを行います。

正常に応答を受信し、自 IP アドレスが変更される場合、本製品内でオープンされている（バインド済みの）ソケットはすべて強制的にクローズされます。

指定されたタイムアウト時間内に応答を受信できなかった場合、指定された回数だけリトライします。リトライアウトした場合、DHCP サーバが存在しないか、応答タイムアウトのいずれかのエラーを返します。このとき、動作パラメータの自 IP アドレス、デフォルトゲートウェイ、サブネットマスクの各設定値は元の値が保持されます。

取得した IP アドレスを格納する領域のポインタが 4 バイト境界のアドレスを指していない場合、パラメータエラーを返します。

タイムアウト時間が 0 以下の場合、パラメータエラーを返します。

リトライ回数が 0 以下の場合、パラメータエラーを返します。

本ファンクションはブロッキングで処理を行います。

【C 関数例】

```
int DhcpReq(unsigned long *ip, int tmout, int retry)
{
    return trapa_svc(0x65, pid, ip, tmout, retry);
}
```

4.3.23　　MAC アドレス取得

【機能】
　　本製品の MAC アドレスを取得します。

【入力】
　　トラップ番号　　0x66
　　コード（R0）　　プロセス ID
　　R4　　MAC アドレスを格納する領域の先頭アドレス

【出力】
　　リターンコード　　OK　　正常終了

	+0	+1	+2	+3
R4+00→	MAC アドレス			
+04	MAC アドレス（つづき）			

　　ERR_PARAM　　パラメータエラー

【解説】
　　本製品に工場出荷時より設定されている MAC アドレスを返します。

　　MAC アドレスを格納する領域のポインタが NULL の場合、パラメータエラーを返します。

【C 関数例】
```
int GetMacAddr(unsigned char *mac)
{
    return trapa_svc(0x66, pid, mac);
}
```

4.4 その他

CD（添付品）に格納されている BIOSIF.C には前述の各ファンクションに加えて、いくつかの関数が格納されています。

4.4.1 BIOS 初期化

【関数名】
B_Init

【機能】
BIOS インタフェースの初期化を行います。

【入力】

int	aPid	ユーザプログラムのプロセス ID

【出力】
なし

【解説】
ユーザプログラムのプロセス ID を BIOSIF.C 内部の領域に退避します。
BIOSIF.C では、各 BIOS ファンクションのコールに必要なプロセス ID を、本関数で退避した領域から参照します。ユーザプログラムの先頭で本関数を呼び出してください。

4.4.2 デバッグインタフェース初期化

【関数名】
TrInit

【機能】
デバッグインタフェースの初期化を行います。

【入力】
なし

【出力】
なし

【解説】
システムプロセスの TELNET サーバを起動し、そのプロセス ID を BIOSIF.C の内部の領域に退避します。TrOut 関数により、ユーザプログラムから任意の文字列を TELNET クライアントの画面に表示します。

4.4.3 デバッグ文字列出力

【関数名】
TrOut

【機能】
デバッグ文字列の出力

【入力】

char	*msgstr	デバッグ文字列

【出力】
なし

【解説】
デバッグ文字列を TELNET サーバプロセスに通知します。TELNET サーバプロセスは、通知された文字列を、最初に接続した TELNET クライアントに送信します。TELNET の接続がない場合、通知された文字列は破棄されます。

５．　エラーコード一覧

BIOS ファンクションが返すエラーコードの一覧を以下に示します。

名称	コード	内容
ERR_PARAM	0xF0000000	パラメータエラー
ERR_PID	0xF0000001	プロセス ID 不正
ERR_INUSE	0xF0000002	資源を他のプロセスで使用中
ERR_RUNNING	0xF0000003	プログラムは起動済み
ERR_NOPROC	0xF0000004	プロセスが存在しない
ERR_PGNOTFOUND	0xF0000005	プログラムが存在しない
ERR_NOOTHER	0xF0000006	他のプロセスが存在しない
ERR_QUEFULL	0xF0000007	メッセージキューに追加不可
ERR_MEMOVER	0xA0000000	メモリ不足
ERR_FLSHWRT	0xA0000001	フラッシュ書込みエラー
ERR_PGMEMOVER	0xA0000002	プログラム格納エリアに空きがない
ERR_RSOPENED	0x90000000	RS-232C オープン済み
ERR_RSCLOSED	0x90000001	RS-232C 未オープン
ERR_RSFULL	0x90000002	RS-232C 送信バッファフル
ERR_RSORER	0x90000003	オーバランエラー
ERR_RSFER	0x90000004	フレーミングエラー
ERR_RSPER	0x90000005	パリティエラー
ERR_RSSDEXIST	0x90000006	RS-232C データ送信中
ERR_LANOPENED	0x81000007	LAN オープン済み
ERR_LANCLOSED	0x81000008	LAN 未オープン
ERR_BOUND	0x84000001	ソケットはバインド済み
ERR_NOBUFS	0x84000002	ソケットをこれ以上作成できない
ERR_NOSOCKET	0x84000003	ソケット番号が無効
ERR_OPENED	0x84050004	ソケットはすでに接続されている
ERR_NOTBOUND	0x84050005	ソケットはバインドされていない
ERR_NETUNREACH	0x84050006	接続先のノードに到達できない
ERR_TIMEDOUT	0x84050007	タイムアウト
ERR_ADDRINUSE	0x84050008	指定のアドレスは他で使用中
ERR_NOTCONN	0x84050009	ソケットは接続されていない
ERR_MFILE	0x8405000A	ソケットは接続要求を受付可能になっていない
ERR_EXISTDATA	0x8405000B	バッファに未処理のデータが存在する
ERR_NOTUDP	0x8405000C	UDP ソケットでない
ERR_CONNREFUSED	0x8405000D	接続が拒否された
ERR_LISTENING	0x8405000E	ソケットは接続要求を受付可能になっている
ERR_CONNECTING	0x8405000F	ソケットは接続要求中
ERR_NOTTCP	0x84060000	TCP ソケットでない

６．　システムプロセスについて

本製品には標準で FTP プロトコル、および TCP ソケットスループロトコルを搭載しており、接続形態によってはそのまま利用することができます。
一方、本 BIOS インタフェースを利用したユーザプログラムにより、本製品の機能をより効果的に利用することが可能で、ユーザプログラムの作成に際しても、本製品が標準で搭載している各種プロトコルを呼び出す事が可能となっています。

ユーザプログラムより呼び出す事のできるシステムプロセスには次のものがあります。

- FTP サーバプロセス
- ソケットクライアントプロセス
- ソケットサーバプロセス
- TELNET サーバプロセス
- RS-232C スループロセス

各プロセスの名称と機能概要は以下の通りです。

システムプロセス名	プログラム名	機能概要
FTP サーバプロセス	FtpSv	FTP プロトコルのサーバ機能をサポートします。 ユーザプログラムからのデータはメッセージとして本プロセスに渡され、FTP クライアントのコマンド要求により転送されます。 FTP クライアントからのデータは、メッセージとしてユーザプログラムに渡されます。
ソケットクライアントプロセス	SockCl	TCP コネクションを使用したクライアントとしての、データの送受信をサポートします。 本プロセスは宛先に対して接続を行い、ユーザプログラムとの間で、メッセージとして受け渡されるデータの送受信を行います。
ソケットサーバプロセス	SockSv	TCP コネクションを使用したサーバとしての、データの送受信をサポートします。 本プロセスはクライアントからの接続要求に対し接続を確立し、ユーザプログラムとの間で、メッセージとして受け渡されるデータの送受信を行います。
TELNET サーバプロセス	TelnetSv	TELNET プロトコルのサーバ機能をサポートします。 デフォルトではコマンドインタプリタとして動作し、TELNET クライアントからの入力に応じて、プロセスの起動、状態の表示等を行います。 ユーザプログラムからのデータはメッセージとして本プロセスに渡され、TELNET クライアントの画面に表示されます。
RS-232C スループロセス	Serial	RS-232C インタフェースに対するデータの送受信をサポートします。 本プロセスは動作パラメータで指定されている通信パラメータにより RS-232C を初期化し、ユーザプログラムとの間で、メッセージとして受け渡されるデータの送受信を行います。

6.1 FTP サーバプロセス

6.1.1 動作

FTP プロトコルのサーバ機能をサポートします。送受信バッファをそれぞれ 64k バイトずつユーザプログラムデータエリアより確保します。

FTP クライアントからのログイン時に、動作パラメータのユーザ名、パスワードを照合します。

ユーザプログラムからメッセージで受け取ったデータは、クライアントからの取得要求により転送されます。動作パラメータのファイルターミネータが設定されている場合、データは複数のファイルに分割されます。

クライアントからのデータは、データ出力先プロセスへメッセージで受け渡されます。

デフォルトのデータ出力先プロセスは RS-232C スループロセスです。ユーザプログラムからデータ要求メッセージにより、出力先プロセスを変更することができます。

ユーザプログラムから、オープン要求メッセージおよびクローズ要求メッセージにより FTP サーバのログインユーザ名・パスワードの変更を行うことができます。また、状態通知要求メッセージにより、FTP サーバの状態を知ることができます。

FTP サーバでは以下の動作パラメータを使用します。詳しくは「BLC-100 セットアップマニュアル」を参照して下さい。

- 自 IP アドレス
- ユーザ名
- パスワード
- レコードターミネータ

6.1.2 入出力メッセージ

FTP サーバプロセスは以下のメッセージにより他のプロセスと連携します。

コード	メッセージ名	入出力	内　　容
MSG_SYSMSG	システムメッセージ	入	FTP サーバのソケットにイベントが発生したときに、システムが通知します。
MSG_PUTFILE MSG_APPENDFILE	ファイル情報通知／応答	入／出	FTP クライアントから STOR,APPE コマンドを受け付けたときに、通知します。 応答メッセージのパラメータにより、ファイル終了通知を省略することができます。ファイル終了通知を省略した場合、FTP クライアントからのデータ受信が終了したときに正常終了を FTP クライアントに通知します。
MSG_GETFILE	ファイル要求／応答	入／出	FTP クライアントから RETR コマンドを受け付けたときに、通知します。
MSG_LSTFILE	ファイル一覧要求／応答	入／出	FTP クライアントから LIST,NLST コマンドを受け付けたときに、通知します。 応答メッセージのパラメータを FTP クライアントに送信します。
MSG_DELFILE	ファイル削除要求／応答	入／出	FTP クライアントから DELE コマンドを受け付けたときに、通知します。
MSG_CH_DIR	ディレクトリ変更要求／応答	入／出	FTP クライアントから CWD コマンドを受け付けたときに、通知します。
MSG_PUTDATA	データ通知	入／出	FTP クライアントから受信したファイルデータを通知します。 入力されたデータは FTP クライアントからの要求により送信します。
MSG_ENDFILE	ファイル終了通知／応答	入／出	FTP クライアントから受信したファイルデータを全て通知した後に出力します。 応答メッセージのパラメータにより、ファイル受信結果を FTP クライアントに送信します。
MSG_GETSTAT	状態取得	入／出	FTP クライアントから STAT コマンドを受け付けたときに、出力します。 「状態取得」が入力されると、サーバの状態が変化したときに、通知メッセージが出力されます。
MSG_GETDATA	データ要求	入	ソケットからの受信データの通知先プロセスを変更します。
MSG_PUTSTAT	状態通知	入	FTP サーバプロセスが出力した「状態取得」に対する応答として入力を受け付けます。入力された情報を FTP クライアントに STAT コマンドの応答として送信します。
MSG_SVOPEN	サーバオープン要求	入	FTP サーバのリッスンソケットをオープンします。 オープンされているか、FTP クライアントから接続されている場合はエラーを要求元に通知します。
MSG_SVCLOSE	サーバクローズ要求	入	FTP サーバの全てのソケットをクローズします。
MSG_CONNECTED	接続完了通知	出	FTP クライアントから接続されたときに通知します。
MSG_SVOPENED	サーバオープン通知	出	FTP サーバのリッスンソケットがオープンしたときに通知します。
MSG_SVCLOSED	サーバクローズ通知	出	FTP サーバの全てのソケットがクローズしたときに通知します。
MSG_BUFFULL	バッファ満杯通知	入／出	FTP サーバの持つ 64 キロバイトの送信データバッファが一杯になったときに通知します。バッファレディ通知が通知されるまでデータ通知をしないようにして下さい。 バッファ満杯通知が入力されると、バッファレディ通知が入力されるまで、データ通知を出力しなくなります。
MSG_BUFREADY	バッファレディ通知	入／出	バッファ満杯通知を出力した後、FTP サーバの送信データバッファが空になったときに通知します。 バッファレディ通知が入力されると、データ通知の出力を再開します。
MSG_PUTLOG	ログ通知	出	内部状態が変化したとき、コマンドの入力およびレスポンスの送信が行われたときに通知します。

6.2 ソケットクライアントプロセス

6.2.1 動作

TCP のクライアントソケットを使用してデータの送受信を行います。
本プロセスは動作パラメータで指定されている宛先 IP アドレス、宛先ポート番号に対して接続を行い、データの受信を待ちます。また、デフォルトの受信データの出力先プロセスを RS-232C スループロセスとします。

ユーザプログラムからメッセージで受け取ったデータは、そのまま接続ソケットから送信されます。接続が確立されていないときにデータを受け取った場合、ソケットの接続動作を開始し、バッファ満杯通知を返します。接続が確立されたときに、バッファレディ通知を出力します。

接続ソケットから受信したデータは、そのまま出力先プロセスへメッセージで受け渡されます。

ユーザプログラムから、接続要求メッセージおよび切断要求メッセージによりソケットの接続をコントロールすることができます。また、状態通知要求メッセージにより、接続状態を知ることができます。

ソケットクライアントでは以下の動作パラメータを使用します。詳しくは「BLC-100 セットアップマニュアル」を参照して下さい。

- 自 IP アドレス
- 宛先 IP アドレス
- 宛先ポート番号
- クライアント接続契機
- 無通信監視タイマ

6.2.2 入出力メッセージ

ソケットクライアントプロセスは以下のメッセージにより他のプロセスと連携します。

コード	メッセージ名	入出力	内　　容
MSG_SYSMSG	システムメッセージ	入	ソケットクライアントのソケットにイベントが発生したときに、システムが通知します。
MSG_PUTDATA	データ通知	入／出	ソケットから受信したデータを通知します。 入力されたデータはソケットの接続先に送信します。
MSG_GETSTAT	状態取得	入	ソケットクライアントの状態が変化したときに、通知メッセージが出力されます。
MSG_GETDATA	データ要求	入	ソケットからの受信データの通知先プロセスを変更します。
MSG_CONNECT	接続要求	入	パラメータで指定された宛先に接続します。
MSG_CONNCLOSE	接続切断要求	入	接続されているソケットをクローズします。
MSG_CONNECTED	接続完了通知	出	宛先に接続できたときに通知します。
MSG_CONNCLOSED	接続切断通知	出	接続が切れたときに通知します。
MSG_BUFFULL	バッファ満杯通知	入／出	ソケットの 4 キロバイトの送信データバッファが一杯になったときに通知します。バッファレディ通知が通知されるまでデータ通知をしないようにして下さい。 バッファ満杯通知が入力されると、バッファレディ通知が入力されるまで、データ通知を出力しなくなります。
MSG_BUFREADY	バッファレディ通知	入／出	バッファ満杯通知を出力した後、ソケットの送信データバッファが空になったときに通知します。 バッファレディ通知が入力されると、データ通知の出力を再開します。
MSG_PUTLOG	ログ通知	出	内部状態が変化したとき、データの送受信が行われたときに通知します。

6.3 ソケットサーバプロセス

6.3.1 動作

TCP のサーバソケットを使用してデータの送受信を行います。

本プロセスは動作パラメータで指定されている自ポート番号でサーバソケットを作成し、クライアントからの接続要求を待ちます。接続要求を受け付けると、サーバソケットをクローズし、データの受信を待ちます。

デフォルトの受信データの出力先プロセスを RS-232C スループロセスとします。

ユーザプログラムからメッセージで受け取ったデータは、そのまま接続ソケットから送信されます。接続が確立されていないときにデータを受け取った場合、エラーメッセージを返します。

接続ソケットから受信したデータは、そのまま出力先プロセスへメッセージで受け渡されます。

ユーザプログラムから、オープン要求メッセージおよびクローズ要求メッセージによりサーバソケットをコントロールすることができます。また、状態通知要求メッセージにより、サーバの状態を知ることができます。

ソケットサーバでは以下の動作パラメータを使用します。詳しくは「BLC-100 セットアップマニュアル」を参照して下さい。

- 自 IP アドレス
- 自ポート番号

6.3.2 入出力メッセージ

ソケットサーバプロセスは以下のメッセージにより他のプロセスと連携します。

コード	メッセージ名	入出力	内　　容
MSG_SYSMSG	システムメッセージ	入	ソケットサーバのソケットにイベントが発生したときに、システムが通知します。
MSG_PUTDATA	データ通知	入／出	ソケットから受信したデータを通知します。 入力されたデータはソケットの接続先に送信します。
MSG_GETSTAT	状態取得	入	ソケットサーバの状態が変化したときに、通知メッセージが出力されます。
MSG_GETDATA	データ要求	入	ソケットからの受信データの通知先プロセスを変更します。
MSG_SVOPEN	サーバオープン要求	入	ソケットサーバのリッスンソケットをオープンします。 オープンされているか、クライアントから接続されている場合はエラーを要求元に通知します。
MSG_SVCLOSE	サーバクローズ要求	入	ソケットサーバの全てのソケットをクローズします。
MSG_CONNECTED	接続完了通知	出	クライアントから接続されたときに通知します。
MSG_SVOPENED	サーバオープン通知	出	ソケットサーバのリッスンソケットがオープンしたときに通知します。
MSG_SVCLOSED	サーバクローズ通知	出	ソケットサーバの全てのソケットがクローズしたときに通知します。
MSG_BUFFULL	バッファ満杯通知	入／出	ソケットの 4 キロバイトの送信データバッファが一杯になったときに通知します。バッファレディ通知が通知されるまでデータ通知をしないようにして下さい。 バッファ満杯通知が入力されると、バッファレディ通知が入力されるまで、データ通知を出力しなくなります。
MSG_BUFREADY	バッファレディ通知	入／出	バッファ満杯通知を出力した後、ソケットの送信データバッファが空になったときに通知します。 バッファレディ通知が入力されると、データ通知の出力を再開します。
MSG_PUTLOG	ログ通知	出	内部状態が変化したとき、データの送受信が行われたときに通知します。

6.4 TELNET サーバプロセス

6.4.1 動作

TELNET プロトコルのサーバ機能をサポートします。通常動作モードで起動した場合、複数の TELNET クライアントからの接続を受け付けます。同時に接続できる TELNET クライアントの本数は 8 本です。

TELNET クライアントからのログイン時に、動作パラメータのユーザ名、パスワードを照合します。

ユーザプログラムからメッセージで受け取ったデータは、そのまま最初に接続されたクライアントに送信されます。接続が確立されていないときにデータを受け取った場合、エラーメッセージを返します。

TELNET クライアントからのデータは本プロセスのコマンドインタプリタへの入力となります。

ユーザプログラムから、オープン要求メッセージおよびクローズ要求メッセージにより TELNET サーバのログインユーザ名・パスワードの変更を行うことができます。また、状態通知要求メッセージにより、TELNET サーバの状態を知ることができます。

TELNET サーバでは以下の動作パラメータを使用します。詳しくは「BLC-100 セットアップマニュアル」を参照して下さい。

- 自 IP アドレス
- ユーザ名
- パスワード

6.4.2 サポートコマンド

TELNET サーバプロセスは TELNET クライアントからの入力として以下のコマンドをサポートします。コマンドの大文字・小文字の区別はありません。

コマンド	内　　容	フォーマット
EX	ユーザプログラムを起動します。	EX プログラム名
LS	格納されているユーザプログラムの一覧を出力します。	LS [-l] [-a]
PS	現在実行中のユーザプログラムの一覧を出力します。	PS
KL	現在実行中のユーザプログラムを停止します。	KL プログラム名
ST	製品の状態情報を出力します。	ST
MD	製品の内部メモリをダンプします。	MD [開始アドレス [{b|w|l}]]
TR	トレースモードに入ります。	TR [プロセス ID1 プロセス ID2...]
QT	TELNET の接続を切断します。	QT

6.4.3 入出力メッセージ

TELNET サーバプロセスは以下のメッセージにより他のプロセスと連携します。

コード	メッセージ名	入出力	内　　容
MSG_SYSMSG	システムメッセージ	入	TELNET サーバのソケットにイベントが発生したときに、システムが通知します。
MSG_PUTDATA	データ通知	入	入力されたデータは TELNET サーバの最初の接続先に送信します。
MSG_PUTSTAT	状態通知	入	TELNET サーバプロセスが出力した状態通知要求に対する応答として入力を受け付けます。入力された情報を TELNET クライアントに ST コマンドの応答として送信します。
MSG_GETSTAT	状態取得	入／出	TELNET クライアントから ST コマンドを受け付けたときに、RS-232C スループロセスに出力します。 状態通知要求が入力されると、以後ソケットサーバの状態が変化したときに、通知メッセージが出力されます。
MSG_SVOPEN	サーバオープン要求	入	TELNET サーバのリッスンソケットをオープンします。 オープンされているか、クライアントから接続されている場合はエラーを要求元に通知します。
MSG_SVCLOSE	サーバクローズ要求	入	TELNET サーバの全てのソケットをクローズします。
MSG_CONNECTED	接続完了通知	出	TELNET クライアントから接続されたときに通知します。
MSG_SVOPENED	サーバオープン通知	出	TELNET サーバのリッスンソケットがオープンしたときに通知します。
MSG_SVCLOSED	サーバクローズ通知	出	TELNET サーバの全てのソケットがクローズしたときに通知します。
MSG_BUFFULL	バッファ満杯通知	入／出	ソケットの 4 キロバイトの送信データバッファが一杯になったときに通知します。バッファレディ通知が通知されるまでデータ通知をしないようにして下さい。 バッファ満杯通知が入力されると、バッファレディ通知が入力されるまで、データ通知を出力しなくなります。
MSG_BUFREADY	バッファレディ通知	入／出	バッファ満杯通知を出力した後、ソケットの送信データバッファが空になったときに通知します。 バッファレディ通知が入力されると、データ通知の出力を再開します。
MSG_PUTLOG	ログ通知	入	入力された情報をトレースモードのクライアントに送信します。

6.5 RS-232C スループロセス

6.5.1 動作

RS-232C インタフェースに対するデータの送受信を行います。

本プロセスは動作パラメータで指定されている通信パラメータにより RS-232C を初期化し、データの受信を待ちます。また、ソケットクライアント→ソケットサーバ→FTP サーバの順にプロセスが起動されているか否かを調べ、受信データの出力先プロセスを決定します。

ユーザプログラムからメッセージで受け取ったデータは、そのまま RS-232C インタフェースに出力されます。

RS-232C インタフェースから受信したデータは、そのまま出力先プロセスへメッセージで受け渡されます。動作パラメータのレコードターミネータが設定されている場合、受け渡されるデータはレコード単位に区切られます。

RS-232C スルーでは以下の動作パラメータを使用します。詳しくは「BLC-100 セットアップマニュアル」を参照して下さい。

- RS-232C パラメータ
- レコードターミネータ

6.5.2 入出力メッセージ

RS-232C スループロセスは以下のメッセージにより他のプロセスと連携します。

コード	メッセージ名	入出力	内　　容
MSG_SYSMSG	システムメッセージ	入	RS-232C の受信バッファにデータがあるときに、システムが通知します。
MSG_PUTDATA	データ通知	入／出	RS-232C から受信したデータを通知します。 入力されたデータは RS-232C に出力します。
MSG_GETSTAT	状態取得	入	状態通知要求が入力されると、制御線状態と送受信バッファのデータ量を状態通知メッセージで返します。
MSG_PUTSTAT	状態通知	出	制御線状態と送受信バッファのデータ量を出力します。
MSG_BUFFULL	バッファ満杯通知	入／出	ソケットの 4 キロバイトの送信データバッファが一杯になったときに通知します。バッファレディ通知が通知されるまでデータ通知をしないようにして下さい。 バッファ満杯通知が入力されると、バッファレディ通知が入力されるまで、データ通知を出力しなくなります。
MSG_BUFREADY	バッファレディ通知	入／出	バッファ満杯通知を出力した後、ソケットの送信データバッファが空になったときに通知します。 バッファレディ通知が入力されると、データ通知の出力を再開します。

7． メッセージリファレンス

システムプロセスが使用しているメッセージコードの一覧を以下に示します。

コード	値	メッセージ名
MSG_SYSMSG	0	システムメッセージ
MSG_PUTDATA	1	データ通知
MSG_GETDATA	2	データ要求
MSG_PUTSTAT	3	状態通知
MSG_GETSTAT	4	状態取得
MSG_SVOPEN	5	サーバオープン要求
MSG_SVCLOSE	6	サーバクローズ要求
MSG_CONNECT	7	接続要求
MSG_CONNECTED	8	接続完了通知
MSG_CONNCLOSE	9	接続切断要求
MSG_CONNCLOSED	10	接続切断通知
MSG_SVOPENED	11	サーバオープン通知
MSG_SVCLOSED	12	サーバクローズ通知
MSG_PUTFILE	13	ファイル情報通知
MSG_GETFILE	14	ファイル要求
MSG_BUFFULL	15	バッファ満杯通知
MSG_BUFREADY	16	バッファレディ通知
MSG_LSTFILE	17	ファイル一覧要求
MSG_ENDFILE	18	ファイル終了通知
MSG_DELFILE	19	ファイル削除要求
MSG_APPENDFILE	20	ファイル追加要求
MSG_CH_DIR	21	ディレクトリ変更要求
MSG_PUTLOG	24	ログ通知

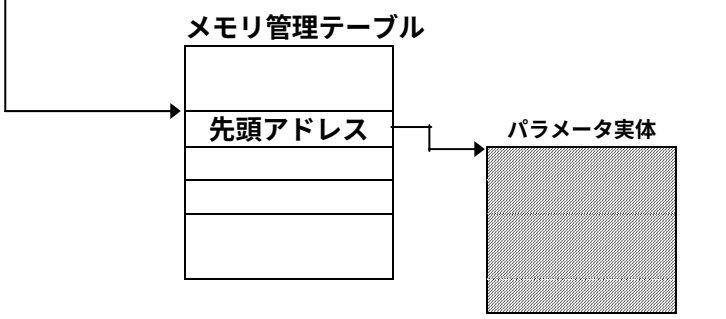
システムプロセスが使用するメッセージではパラメータを以下のように扱います。

・ パラメータ長が 4 以下の場合、メッセージの構造は以下のようになります。

	+0	+1	+2	+3
+00→	宛先プロセス ID		自プロセス ID	
+04	メッセージ種別	コマンドコード	パラメータ長	
+08	パラメータの実体			

・ パラメータ長が 4 より大きい場合、メッセージの構造は以下のようになります。

	+0	+1	+2	+3
+00→	宛先プロセス ID		自プロセス ID	
+04	メッセージ種別	コマンドコード	パラメータ長	
+08	「メモリ取得」ファンクションで取得したポインタ			



7.1 システムメッセージ

【フォーマット】

コード	MSG_SYSMSG
発行元プロセス	**システム（常に** 0**）**
受付プロセス	**全てのプロセス**
メッセージ種別	MSG_REQUEST = 0
パラメータ長	0
パラメータ	NULL

【解説】

RS-232C **受信バッファにデータがある場合、動作中の全てのプロセスに通知します。**

ソケットのイベントが発生した場合、ソケットの所有プロセスにのみ通知します。

【ユーザプログラム側の処理】

RS-232C **のデータ処理を行うプログラムの場合、「**RS **受信」ファンクションでデータを受信して下さい。**

ソケットをオープンしている場合、オープン中のソケットに対して「イベントチェック」ファンクションでソケットの状態をチェックして下さい。

7.2 データ通知

【フォーマット】

コード	MSG_PUTDATA
発行元プロセス	FTP サーバ、ソケットクライアント、ソケットサーバ、RS-232C スルー
受付プロセス	全てのプロセス
メッセージ種別	MSG_REQUEST = 0
パラメータ長	データ長
パラメータ	データ

【解説】

FTP サーバは、データコネクションから受信したデータを本メッセージで通知します。

ソケットクライアント、ソケットサーバは、接続ソケットから受信したデータを本メッセージで通知します。

RS-232C スルーは、RS-232C から受信したデータを本メッセージで通知します。このとき、レコードターミネータの指定があれば、ターミネータ文字で区切られます。

FTP サーバに通知した場合、FTP クライアントからの要求によりデータが送られます。このとき、ファイルターミネータの指定があれば、ターミネータ文字で区切られます。FTP クライアントからの要求があるまでは、内部のバッファ（64 キロバイト）にデータを保持します。

ソケットクライアントに通知した場合、接続ソケットからデータを送信します。接続されていない場合、接続をしてからデータを送信します。

ソケットサーバに通知した場合、接続ソケットからデータを送信します。接続されていない場合、「状態エラー」を発行元に返します。

RS-232C スルーに通知した場合、RS-232C からデータを送信します。

TELNET サーバに通知した場合、最初に接続したクライアントにデータを送信します。

TELNET サーバ以外の各システムプロセスはデータを処理した後、メッセージ種別を MSG_RESPONSE（=1）とした応答メッセージを発行元に通知します。データ長はそのまま返しますが、データは返しません。

データ長が 4 より大きい場合、データは動的メモリのエリアに格納されています。パラメータにはデータの先頭アドレスが格納されている、メモリ管理テーブルへのポインタがセットされます。

データ長が 4 以下の場合、パラメータにはデータがそのままセットされます。

【ユーザプログラムでの処理】

メッセージ種別が MSG_REQUEST の場合、受け取ったデータに対する処理を行って下さい。

データ長が 4 より大きい場合はデータのメモリを解放して下さい。

システムプロセスに対する応答メッセージの通知は必須ではありません。

システムプロセスにデータを渡す場合、データ長によりデータのセット方法を切り換えて下さい。

7.3 データ要求

【フォーマット】

コード	MSG_GETDATA
発行元プロセス	ユーザプログラム
受付プロセス	FTP サーバ、ソケットクライアント、ソケットサーバ、RS-232C スルー
メッセージ種別	MSG_REQUEST = 0
パラメータ長	0
パラメータ	NULL

【解説】

各受付プロセスに通知した場合、以降のデータ通知メッセージの宛先を発行元のユーザプログラムに切り替えます。

【ユーザプログラムでの処理】

システムプロセスがユーザプログラムに発行することはありません。

7.4 状態通知

【フォーマット】

コード	MSG_PUTSTAT
発行元プロセス	RS-232C スルー、ユーザプログラム
受付プロセス	FTP サーバ、TELNET サーバ、ユーザプログラム
メッセージ種別	MSG_REQUEST = 0
パラメータ長	状態を表すテキストデータの長さ
パラメータ	状態を表すテキストデータ

【解説】

FTP サーバは、FTP クライアントから STAT コマンドを受け付けると、「状態取得」を通知して、本メッセージを待ちます。本メッセージのパラメータデータを FTP クライアントに送信します。

TELNET サーバは、TELNET クライアントから ST コマンドを受け付けると、「状態取得」を通知して、本メッセージを待ちます。本メッセージのパラメータデータを TELNET クライアントに送信します。

RS-232C スルーは「状態取得」に対する応答として、制御線の状態、RS-232C 送受信バッファのデータ量をテキストデータで通知します。

【ユーザプログラムでの処理】

受け取ったデータに対する処理を行い、データのメモリを解放して下さい。

7.5 状態取得

【フォーマット】

コード	MSG_GETSTAT
発行元プロセス	FTP サーバ、TELNET サーバ、ユーザプログラム
受付プロセス	FTP サーバ、ソケットクライアント、ソケットサーバ、RS-232C スルー
メッセージ種別	MSG_REQUEST = 0
パラメータ長	4
パラメータ	オプション 0 = 状態要求登録 1 = 現在の状態を要求 2 = 状態を表すテキストデータを要求

【解説】

FTP サーバは、FTP クライアントから STAT コマンドを受け付けると、本メッセージをオプション = 2 で出力します。

TELNET サーバは、TELNET クライアントから ST コマンドを受け付けると、本メッセージをオプション = 2 で出力します。

FTP サーバ、ソケットサーバに通知した場合、「接続完了通知」、「接続切断通知」、「サーバオープン通知」、「サーバクローズ通知」のいずれかを状態に合わせて発行元に通知します。

ソケットクライアントに通知した場合、「接続完了通知」、「接続切断通知」のいずれかを状態に合わせて発行元に通知します。

オプション = 0 の場合、以降の状態変化時に通知を行います。

オプション = 1 の場合、現在の状態を通知し、以降の状態変化時には通知を行いません。

オプション = 2 の場合、FTP サーバ、ソケットクライアント、ソケットサーバはメッセージを無視します。

RS-232C スルーに通知した場合、オプション = 2 であれば、制御線の状態と RS-232C 送受信バッファのデータ量を「状態通知」メッセージで発行元に通知します。オプション = 0 またはオプション = 1 の場合はメッセージを無視します。

【ユーザプログラムでの処理】

システムプロセスがユーザプログラムに発行することはありません。

7.6 サーバオープン要求

【フォーマット】

コード	MSG_SVOPEN
発行元プロセス	ユーザプログラム
受付プロセス	FTP サーバ、ソケットサーバ、TELNET サーバ
メッセージ種別	MSG_REQUEST = 0
パラメータ長	2 + (ユーザ名の長さ + 1) + (パスワードの長さ + 1)
パラメータ	リッスンポート番号 + ユーザ名 + パスワード

【解説】

FTP サーバに通知した場合、FTP のリッスンソケットをオープンします。
リッスンポート番号は無視します。
ログインを許可するユーザ名、パスワードをパラメータのものに切り替えます。

ソケットサーバに通知した場合、パラメータに指定されたポート番号でリッスンソケットをオープンします。

TELNET サーバに通知した場合、TELNET のリッスンソケットをオープンします。
リッスンポート番号は無視します。
ログインを許可するユーザ名、パスワードをパラメータのものに切り替えます。

すでにリッスンソケットがオープンされている場合、「状態エラー」を発行元に返します。

クライアントから接続されている場合、「状態エラー」を発行元に返します。

【ユーザプログラムでの処理】

システムプロセスがユーザプログラムに発行することはありません。

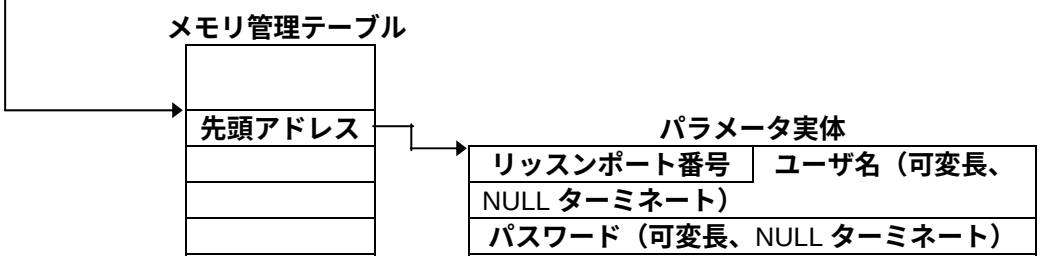
ユーザ名、パスワードはそれぞれ NULL で終わる文字列として下さい。有効な長さは 15 バイトまでで、15 バイトを越える部分は無視されます。

ユーザ名、パスワードをともに「なし」にする場合、メッセージの構造は以下のようにして下さい。

	+0	+1	+2	+3
+00→	宛先プロセス ID		自プロセス ID	
+04	MSG_REQUEST	MSG_SVOPEN	2	
+08	リッスンポート番号			

ユーザ名またはパスワードを指定する場合、「メモリ確保」ファンクションにより必要なメモリを確保し、その先頭から上記のパラメータデータをセットして下さい。メッセージの構造は以下のようにして下さい。

	+0	+1	+2	+3
+00→	宛先プロセス ID		自プロセス ID	
+04	MSG_REQUEST	MSG_SVOPEN	パラメータ長	
+08	「メモリ取得」ファンクションで取得したポインタ			



7.7 サーバクローズ要求

【フォーマット】

コード	MSG_SVCLOSE
発行元プロセス	**ユーザプログラム**
受付プロセス	FTP **サーバ、ソケットサーバ、**TELNET **サーバ**
メッセージ種別	MSG_REQUEST = 0
パラメータ長	0
パラメータ	NULL

【解説】

FTP **サーバ、ソケットサーバ、**TELNET **サーバに通知した場合、クライアントからの接続ソケットも含めてすべてのソケットをクローズします。**

すでにオープンされているソケットがない場合、「状態エラー」を発行元に返します。

【ユーザプログラムでの処理】

システムプロセスがユーザプログラムに発行することはありません。

7.8 接続要求

【フォーマット】

コード	MSG_CONNECT
発行元プロセス	**ユーザプログラム**
受付プロセス	**ソケットクライアント**
メッセージ種別	MSG_REQUEST = 0
パラメータ長	6
パラメータ	**宛先** IP **アドレス** + **宛先ポート番号**

【解説】

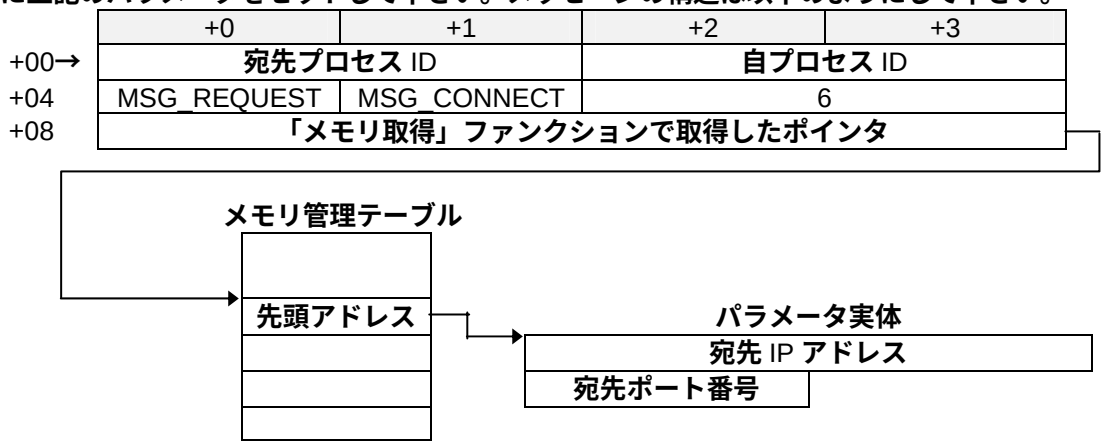
ソケットクライアントに通知した場合、パラメータで指定した宛先に接続します。

すでに接続されている場合、「状態エラー」を発行元に返します。

【ユーザプログラムでの処理】

システムプロセスがユーザプログラムに発行することはありません。

宛先アドレスを格納するための 6 **バイトの領域を「メモリ確保」ファンクションにより確保し、そこに上記のパラメータをセットして下さい。メッセージの構造は以下のようにして下さい。**



7.9 接続完了通知

【フォーマット】

コード	MSG_CONNECTED
発行元プロセス	FTP サーバ、ソケットクライアント、ソケットサーバ、TELNET サーバ
受付プロセス	「状態取得」を発行したユーザプログラム
メッセージ種別	MSG_REQUEST = 0
パラメータ長	0
パラメータ	NULL

【解説】

FTP サーバ、ソケットサーバ、TELNET サーバは、クライアントからの接続を受け付けたときに、本メッセージを通知します。

ソケットクライアントは、接続が確立したときに本メッセージを通知します。

【ユーザプログラムでの処理】

「状態取得」メッセージを通知しない限り、システムプロセスがユーザプログラムに発行することはありません。

7.10 接続切断要求

【フォーマット】

コード	MSG_CONNCLOSE
発行元プロセス	**ユーザプログラム**
受付プロセス	**ソケットクライアント**
メッセージ種別	MSG_REQUEST = 0
パラメータ長	0
パラメータ	NULL

【解説】

ソケットクライアントに通知した場合、接続中のソケットをクローズします。

接続されていない場合、「状態エラー」を発行元に返します。

【ユーザプログラムでの処理】

システムプロセスがユーザプログラムに発行することはありません。

7.11 接続切断通知

【フォーマット】

コード	MSG_CONNCLOSED
発行元プロセス	**ソケットクライアント**
受付プロセス	**「状態取得」を発行したユーザプログラム**
メッセージ種別	MSG_REQUEST = 0
パラメータ長	0
パラメータ	NULL

【解説】

ソケットクライアントは、接続が切れたときに本メッセージを通知します。

【ユーザプログラムでの処理】

「状態取得」メッセージを通知しない限り、システムプロセスがユーザプログラムに発行することはありません。

7.12 サーバオープン通知

【フォーマット】

コード	MSG_SVOPENED
発行元プロセス	FTP サーバ、ソケットサーバ、TELNET サーバ
受付プロセス	「状態取得」を発行したユーザプログラム
メッセージ種別	MSG_REQUEST = 0
パラメータ長	0
パラメータ	NULL

【解説】

FTP サーバ、ソケットサーバ、TELNET サーバは、リッスンソケットのオープンに成功した場合に、本メッセージを通知します。

【ユーザプログラムでの処理】

「状態取得」メッセージを通知しない限り、システムプロセスがユーザプログラムに発行することはありません。

7.13 サーバクローズ通知

【フォーマット】

コード	MSG_SVCLOSED
発行元プロセス	FTP サーバ、ソケットサーバ、TELNET サーバ
受付プロセス	「状態取得」を発行したユーザプログラム
メッセージ種別	MSG_REQUEST = 0
パラメータ長	0
パラメータ	NULL

【解説】

FTP サーバ、ソケットサーバ、TELNET サーバは、すべてのソケットがクローズした場合に、本メッセージを通知します。

【ユーザプログラムでの処理】

「状態取得」メッセージを通知しない限り、システムプロセスがユーザプログラムに発行することはありません。

7.14 ファイル情報通知

【フォーマット】

コード	MSG_PUTFILE
発行元プロセス	FTP サーバ
受付プロセス	RS-232C スルー、「データ要求」を発行したユーザプログラム
メッセージ種別	MSG_REQUEST = 0
パラメータ長	ファイル名の長さ
パラメータ	ファイル名

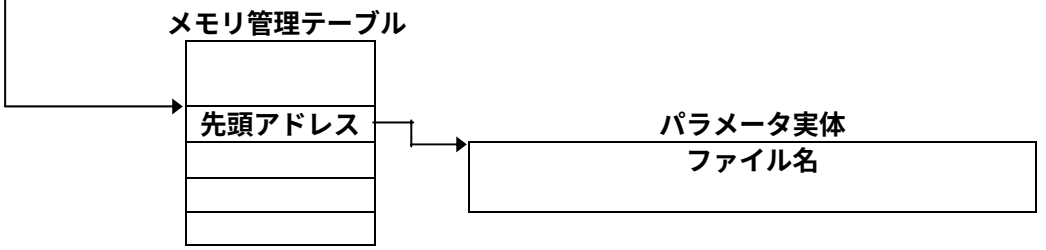
【解説】

FTP サーバは、FTP クライアントから STOR コマンドを受け付けた場合に、本メッセージを通知し、応答待ち状態になります。

RS-232C スルーに通知した場合、「コマンドエラー」を発行元に返します。

ファイル名の長さが 4 より大きい場合、ファイル名は動的メモリのエリアに格納されています。メッセージの構造は以下のようになります。

	+0	+1	+2	+3
+00→	宛先プロセス ID		自プロセス ID	
+04	MSG_REQUEST	MSG_PUTFILE	ファイル名の長さ	
+08	「メモリ取得」ファンクションで取得したポインタ			



ファイル名の長さが 4 以下の場合、パラメータにはファイル名がそのままセットされます。

	+0	+1	+2	+3
+00→	宛先プロセス ID		自プロセス ID	
+04	MSG_REQUEST	MSG_PUTFILE	ファイル名の長さ	
+08	ファイル名			

【ユーザプログラムでの処理】

「データ要求」メッセージを通知しない限り、システムプロセスがユーザプログラムに発行することはありません。

ファイル名の長さが 4 より大きい場合はファイル名のメモリを解放して下さい。

本メッセージを通知された場合、以下の応答メッセージか、「コマンドエラー」を返して下さい。

【応答フォーマット】

コード	MSG_PUTFILE
発行元プロセス	ユーザプログラム
受付プロセス	FTP サーバ
メッセージ種別	MSG_RESPONSE = 1
パラメータ長	4
パラメータ	処理の方法
	1 = 送信確認なし
	0 = 送信確認あり
	-1 = データ受付不能

【解説】

FTP サーバは、上記の応答に応じて次の処理を行います。

「送信確認なし」の場合、FTP クライアントからのデータを「データ通知」で通知します。FTP クライアントからすべてのデータを受信したときに、「ファイル転送成功」のメッセージを FTP クライアントに送信します。

「送信確認あり」の場合、FTP クライアントからのデータを「データ通知」で通知します。FTP クライアントからすべてのデータを受信したときに、「ファイル終了通知」メッセージを通知し、その応答により、ファイル転送の成功か失敗かを FTP クライアントに送信します。

「データ受付不能」の場合、ファイル転送は行われません。FTP クライアントには「ファイル書込み不可」のメッセージを送信します。

コマンドエラーが返された場合、FTP サーバは「送信確認なし」の応答と同じ動作を行います。

7.15 ファイル要求

【フォーマット】

コード	MSG_GETFILE
発行元プロセス	FTP サーバ
受付プロセス	RS-232C スルー、「データ要求」を発行したユーザプログラム
メッセージ種別	MSG_REQUEST = 0
パラメータ長	ファイル名の長さ
パラメータ	ファイル名

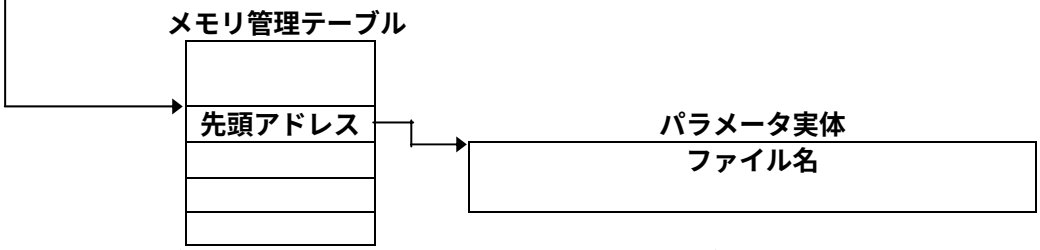
【解説】

FTP サーバは、FTP クライアントから RETR コマンドを受け付けた場合に、本メッセージを通知し、応答待ち状態になります。

RS-232C スルーに通知した場合、「コマンドエラー」を発行元に返します。

ファイル名の長さが 4 より大きい場合、ファイル名は動的メモリのエリアに格納されています。メッセージの構造は以下のようになります。

	+0	+1	+2	+3
+00→	宛先プロセス ID		自プロセス ID	
+04	MSG_REQUEST	MSG_GETFILE	ファイル名の長さ	
+08	「メモリ取得」ファンクションで取得したポインタ			



ファイル名の長さが 4 以下の場合、パラメータにはファイル名がそのままセットされます。

	+0	+1	+2	+3
+00→	宛先プロセス ID		自プロセス ID	
+04	MSG_REQUEST	MSG_GETFILE	ファイル名の長さ	
+08	ファイル名			

【ユーザプログラムでの処理】

「データ要求」メッセージを通知しない限り、システムプロセスがユーザプログラムに発行することはありません。

ファイル名の長さが 4 より大きい場合はファイル名のメモリを解放して下さい。

本メッセージを通知された場合、以下の応答メッセージか、「コマンドエラー」を返して下さい。

【応答フォーマット】

コード	MSG_GETFILE
発行元プロセス	ユーザプログラム
受付プロセス	FTP サーバ
メッセージ種別	MSG_RESPONSE = 1
パラメータ長	4
パラメータ	ファイルサイズ

【解説】

FTP サーバは応答メッセージを受け取ると、以後の「データ通知」で渡されるデータを、応答メッセージのパラメータに指定されているファイルサイズ分 FTP クライアントに送信します。

ファイルサイズが 0 の場合、「ファイルなし」のエラーメッセージを FTP クライアントに送信します。

コマンドエラーが返された場合、FTP サーバは「ファイルなし」のエラーメッセージを FTP クライアントに送信します。

7.16 バッファ満杯通知

【フォーマット】

コード	MSG_BUFFULL
発行元プロセス	FTP サーバ、ソケットクライアント、ソケットサーバ、RS-232C スルー、ユーザプログラム
受付プロセス	FTP サーバ、ソケットクライアント、ソケットサーバ、RS-232C スルー、ユーザプログラム
メッセージ種別	MSG_REQUEST = 0
パラメータ長	0
パラメータ	NULL

【解説】

各システムプロセスは、「データ通知」で渡されたデータのすべてを処理できない場合、本メッセージを「データ通知」の発行元プロセスに返します。受け取ったデータは処理が可能になった時点で処理されます。

【ユーザプログラムでの処理】

本メッセージを通知された場合、「バッファレディ通知」を受け取るまで「データ通知」を発行しないようにして下さい。

本メッセージを他のプロセスに通知する場合、後で「バッファレディ通知」を発行して下さい。

7.17 バッファレディ通知

【フォーマット】

コード	MSG_BUFREADY
発行元プロセス	FTP サーバ、ソケットクライアント、ソケットサーバ、RS-232C スルー、ユーザプログラム
受付プロセス	FTP サーバ、ソケットクライアント、ソケットサーバ、RS-232C スルー、ユーザプログラム
メッセージ種別	MSG_REQUEST = 0
パラメータ長	0
パラメータ	NULL

【解説】

各システムプロセスは、「バッファ満杯通知」を発行した後、再びデータ処理が可能となったときに、本メッセージを通知します。

【ユーザプログラムでの処理】

本メッセージを通知された場合、「データ通知」によるデータの受け渡しを再開して下さい。

「バッファ満杯通知」を通知したプロセスには、後で本メッセージを通知して下さい。

7.18 ファイル一覧要求

【フォーマット】

コード	MSG_LSTFILE
発行元プロセス	FTP サーバ
受付プロセス	RS-232C スルー、「データ要求」を発行したユーザプログラム
メッセージ種別	MSG_REQUEST = 0
パラメータ長	4 + パス名の長さ
パラメータ	一覧オプション 0 = ファイル名のみ 1 = 詳細情報を含む + パス名

【解説】

FTP サーバは、FTP クライアントから LIST または NLST コマンドを受け付けた場合に、本メッセージを通知し、応答待ち状態になります。

NLST コマンドを受け付けた場合、一覧オプションに 0 をセットします。

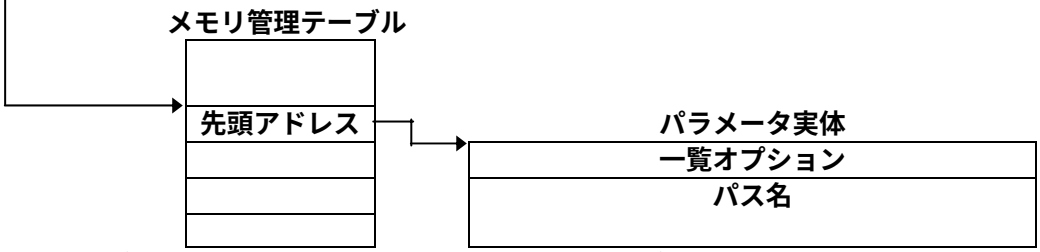
LIST コマンドを受け付けた場合、一覧オプションに 1 をセットします。

パス名は LIST または NLST コマンドの引数が指定されていればセットされます。

RS-232C スルーに通知した場合、「コマンドエラー」を発行元に返します。

パス名の指定がある場合、一覧オプションとパス名は動的メモリのエリアに格納されています。メッセージの構造は以下のようになります。

	+0	+1	+2	+3
+00→	宛先プロセス ID		自プロセス ID	
+04	MSG_REQUEST	MSG_LSTFILE	4 + パス名の長さ	
+08	「メモリ取得」ファンクションで取得したポインタ			



パス名の指定がない場合、パラメータには一覧オプションがそのままセットされます。

	+0	+1	+2	+3
+00→	宛先プロセス ID		自プロセス ID	
+04	MSG_REQUEST	MSG_LSTFILE	4	
+08	一覧オプション			

【ユーザプログラムでの処理】

「データ要求」メッセージを通知しない限り、システムプロセスがユーザプログラムに発行することはありません。

パス名の指定がある場合はパラメータのメモリを解放して下さい。

本メッセージを通知された場合、次ページの応答メッセージか「コマンドエラー」を返して下さい。

【応答フォーマット】

コード	MSG_LSTFILE
発行元プロセス	ユーザプログラム
受付プロセス	FTP サーバ
メッセージ種別	MSG_RESPONSE = 1
パラメータ長	ファイル一覧文字列の長さ
パラメータ	ファイル一覧文字列

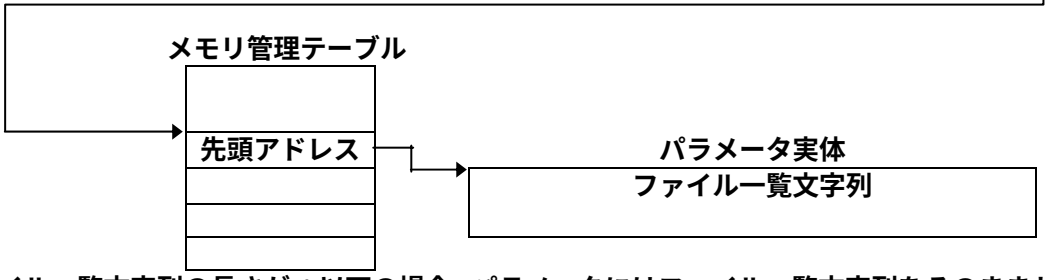
【解説】

FTP サーバは応答メッセージを受け取ると、応答メッセージのパラメータデータを FTP クライアントに送信します。

コマンドエラーが返された場合、FTP サーバは内部バッファの状態を FTP クライアントに送信します。

ファイル一覧文字列が 4 より長い場合、「メモリ確保」ファンクションによりファイル一覧文字列の格納エリアを確保し、その先頭から上記のパラメータデータをセットして下さい。メッセージの構造は以下のようにして下さい。

	+0	+1	+2	+3
+00→	宛先プロセス ID		自プロセス ID	
+04	MSG_RESPONSE	MSG_ LSTFILE	ファイル一覧文字列の長さ	
+08	「メモリ取得」ファンクションで取得したポインタ			



ファイル一覧文字列の長さが 4 以下の場合、パラメータにはファイル一覧文字列をそのままセットして下さい。

	+0	+1	+2	+3
+00→	宛先プロセス ID		自プロセス ID	
+04	MSG_RESPONSE	MSG_ LSTFILE	ファイル一覧文字列の長さ	
+08	ファイル一覧文字列			

7.19 ファイル終了通知

【フォーマット】

コード	MSG_ENDFILE
発行元プロセス	FTP サーバ
受付プロセス	「データ要求」を発行したユーザプログラム
メッセージ種別	MSG_REQUEST = 0
パラメータ長	0
パラメータ	NULL

【解説】

FTP サーバは、FTP クライアントからのファイル転送が終了したときに、本メッセージを通知し、応答待ち状態になります。

「ファイル情報通知」の応答で「送信確認なし」を指定された場合、本メッセージを通知しません。

【ユーザプログラムでの処理】

「データ要求」メッセージを通知しない限り、システムプロセスがユーザプログラムに発行することはありません。

本メッセージを通知された場合、以下の応答メッセージか、「コマンドエラー」を返して下さい。

【応答フォーマット】

コード	MSG_ENDFILE
発行元プロセス	ユーザプログラム
受付プロセス	FTP サーバ
メッセージ種別	MSG_RESPONSE = 1
パラメータ長	4
パラメータ	結果 1 = OK 0 = NG

【解説】

FTP サーバは応答メッセージを受け取ると、応答メッセージのパラメータに対応するメッセージを FTP クライアントに送信します。

コマンドエラーが返された場合、FTP サーバは「ファイル転送成功」を FTP クライアントに送信します。

7.20 ファイル削除要求

【フォーマット】

コード	MSG_DELFILE
発行元プロセス	FTP サーバ
受付プロセス	「データ要求」を発行したユーザプログラム
メッセージ種別	MSG_REQUEST = 0
パラメータ長	削除するファイル名の長さ
パラメータ	削除するファイル名

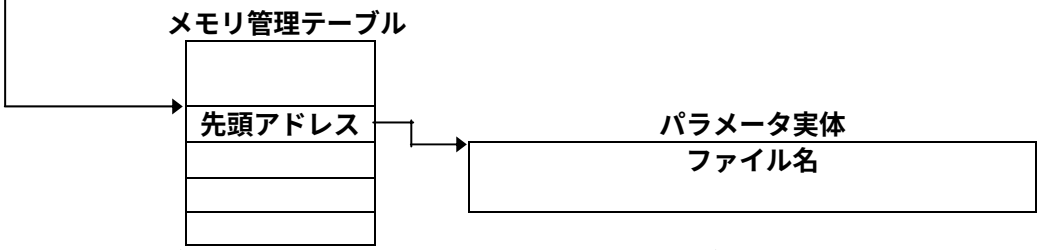
【解説】

FTP サーバは、FTP クライアントから DELE コマンドを受け付けた場合に、本メッセージを通知し、応答待ち状態になります。

RS-232C スルーに通知した場合、「コマンドエラー」を発行元に返します。

ファイル名の長さが 4 より大きい場合、ファイル名は動的メモリのエリアに格納されています。メッセージの構造は以下のようになります。

	+0	+1	+2	+3
+00→	宛先プロセス ID		自プロセス ID	
+04	MSG_REQUEST	MSG_DELFILE	ファイル名の長さ	
+08	「メモリ取得」ファンクションで取得したポインタ			



ファイル名の長さが 4 以下の場合、パラメータにはファイル名がそのままセットされます。

	+0	+1	+2	+3
+00→	宛先プロセス ID		自プロセス ID	
+04	MSG_REQUEST	MSG_DELFILE	ファイル名の長さ	
+08	ファイル名			

【ユーザプログラムでの処理】

「データ要求」メッセージを通知しない限り、システムプロセスがユーザプログラムに発行することはありません。

ファイル名の長さが 4 より大きい場合はファイル名のメモリを解放して下さい。

本メッセージを通知された場合、以下の応答メッセージか、「コマンドエラー」を返して下さい。

【応答フォーマット】

コード	MSG_DELFILE
発行元プロセス	**ユーザプログラム**
受付プロセス	FTP **サーバ**
メッセージ種別	MSG_RESPONSE = 1
パラメータ長	4
パラメータ	**処理結果** 1 = OK 0 = NG

【解説】

FTP **サーバは応答メッセージを受け取ると、応答メッセージのパラメータに対応するメッセージを** FTP **クライアントに送信します。**

コマンドエラーが返された場合、「ファイル削除失敗」のエラーメッセージを FTP **クライアントに送信します。**

7.21 ファイル追加要求

【フォーマット】

コード	MSG_APPENDFILE
発行元プロセス	FTP サーバ
受付プロセス	RS-232C スルー、「データ要求」を発行したユーザプログラム
メッセージ種別	MSG_REQUEST = 0
パラメータ長	ファイル名の長さ
パラメータ	ファイル名

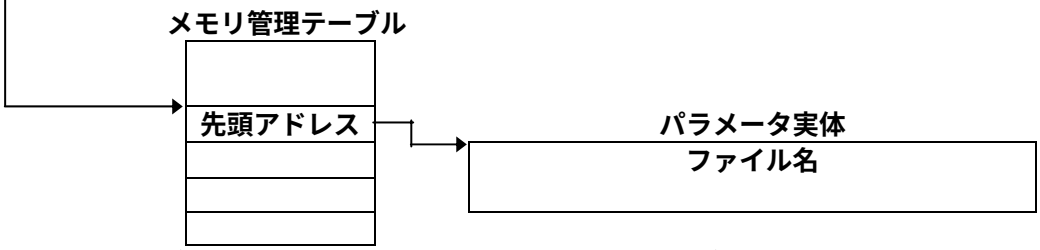
【解説】

FTP サーバは、FTP クライアントから APPE コマンドを受け付けた場合に、本メッセージを通知し、応答待ち状態になります。

RS-232C スルーに通知した場合、「コマンドエラー」を発行元に返します。

ファイル名の長さが 4 より大きい場合、ファイル名は動的メモリのエリアに格納されています。メッセージの構造は以下のようになります。

	+0	+1	+2	+3
+00→	宛先プロセス ID		自プロセス ID	
+04	MSG_REQUEST	MSG_APPENDFILE	ファイル名の長さ	
+08	「メモリ取得」ファンクションで取得したポインタ			



ファイル名の長さが 4 以下の場合、パラメータにはファイル名がそのままセットされます。

	+0	+1	+2	+3
+00→	宛先プロセス ID		自プロセス ID	
+04	MSG_REQUEST	MSG_APPENDFILE	ファイル名の長さ	
+08	ファイル名			

【ユーザプログラムでの処理】

「データ要求」メッセージを通知しない限り、システムプロセスがユーザプログラムに発行することはありません。

ファイル名の長さが 4 より大きい場合はファイル名のメモリを解放して下さい。

本メッセージを通知された場合、以下の応答メッセージか、「コマンドエラー」を返して下さい。

【応答フォーマット】

コード	MSG_APPENDFILE
発行元プロセス	ユーザプログラム
受付プロセス	FTP サーバ
メッセージ種別	MSG_RESPONSE = 1
パラメータ長	4
パラメータ	処理の方法
	1 = 送信確認なし
	0 = 送信確認あり
	-1 = データ受付不能

【解説】

FTP サーバは、上記の応答に応じて次の処理を行います。

「送信確認なし」の場合、FTP クライアントからのデータを「データ通知」で通知します。FTP クライアントからすべてのデータを受信したときに、「ファイル転送成功」のメッセージを FTP クライアントに送信します。

「送信確認あり」の場合、FTP クライアントからのデータを「データ通知」で通知します。FTP クライアントからすべてのデータを受信したときに、「ファイル終了通知」メッセージを通知し、その応答により、ファイル転送の成功か失敗かを FTP クライアントに送信します。

「データ受付不能」の場合、ファイル転送は行われません。FTP クライアントには「ファイル書込み不可」のメッセージを送信します。

コマンドエラーが返された場合、FTP サーバは「送信確認なし」の応答と同じ動作を行います。

7.22 ディレクトリ変更要求

【フォーマット】

コード	MSG_CH_DIR
発行元プロセス	FTP サーバ
受付プロセス	RS-232C スルー、「データ要求」を発行したユーザプログラム
メッセージ種別	MSG_REQUEST = 0
パラメータ長	パス名の長さ
パラメータ	パス名

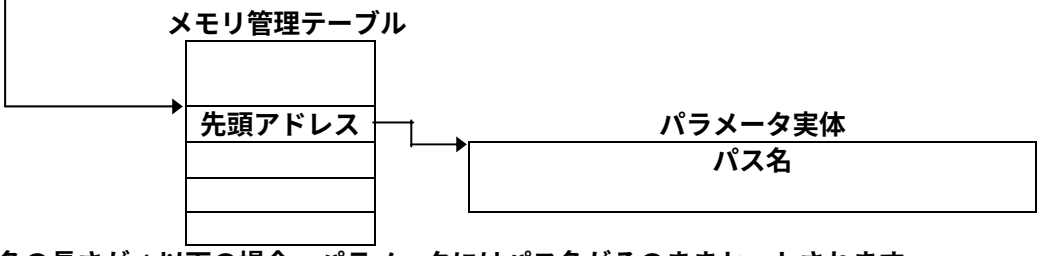
【解説】

FTP サーバは、FTP クライアントから CWD コマンドを受け付けた場合に、本メッセージを通知し、応答待ち状態になります。

RS-232C スルーに通知した場合、「コマンドエラー」を発行元に返します。

パス名の長さが 4 より大きい場合、パス名は動的メモリのエリアに格納されています。メッセージの構造は以下のようになります。

	+0	+1	+2	+3
+00→	宛先プロセス ID		自プロセス ID	
+04	MSG_REQUEST	MSG_CH_DIR	パス名の長さ	
+08	「メモリ取得」ファンクションで取得したポインタ			



パス名の長さが 4 以下の場合、パラメータにはパス名がそのままセットされます。

	+0	+1	+2	+3
+00→	宛先プロセス ID		自プロセス ID	
+04	MSG_REQUEST	MSG_CH_DIR	パス名の長さ	
+08	パス名			

【ユーザプログラムでの処理】

「データ要求」メッセージを通知しない限り、システムプロセスがユーザプログラムに発行することはありません。

パス名の長さが 4 より大きい場合はパス名のメモリを解放して下さい。

本メッセージを通知された場合、以下の応答メッセージか、「コマンドエラー」を返して下さい。

【応答フォーマット】

コード	MSG_CH_DIR
発行元プロセス	ユーザプログラム
受付プロセス	FTP サーバ
メッセージ種別	MSG_RESPONSE = 1
パラメータ長	4
パラメータ	処理の方法
	1 = OK
	0 = NG

【解説】

FTP サーバは応答メッセージを受け取ると、応答メッセージのパラメータに対応するメッセージを FTP クライアントに送信します。

コマンドエラーが返された場合、「コマンド正常終了」のメッセージを FTP クライアントに送信します。

7.23 ログ通知

【フォーマット】

コード	MSG_PUTLOG
発行元プロセス	FTP サーバ、ソケットサーバ、ソケットクライアント、ユーザプログラム
受付プロセス	TELNET サーバ
メッセージ種別	MSG_REQUEST = 0
パラメータ長	ログメッセージの長さ
パラメータ	ログメッセージ

【解説】

TELNET サーバがトレースモード時に本メッセージが通知されると、ログメッセージをクライアントに送信します。

TELNET サーバはトレースモード以外で本メッセージを受けた場合、ログメッセージを破棄します。

ログメッセージは動的メモリのエリアに格納されています。パラメータにはログメッセージの先頭アドレスが格納されている、メモリ管理テーブルへのポインタがセットされます。

【ユーザプログラムでの処理】

システムプロセスが本メッセージをユーザプログラムに発行することはありません。

7.24 コマンドエラー

【フォーマット】

コード	ERR_MSGCMD = 0x81
メッセージ種別	MSG_ERROR = 2
パラメータ長	4
パラメータ	エラーになったコマンドコード

【解説】

未対応のコマンドコードを持つメッセージを受け取った場合に、発行元プロセスに返します。

7.25 状態エラー

【フォーマット】

コード	ERR_INVALIDSTAT = 0x85
発行元プロセス	FTP サーバ、ソケットクライアント、ソケットサーバ、TELNET サーバ
メッセージ種別	MSG_ERROR = 2
パラメータ長	エラーになったメッセージのパラメータ長
パラメータ	エラーになったメッセージのパラメータ

【解説】

内部の状態が、通知されたメッセージを処理できない状態の場合に発行元プロセスに返します。

返されたパラメータは、それぞれのプロセスで処理して下さい。

8. 簡易デバッガ

本製品には、格納したユーザプログラムのデバッグ用に簡易デバッガを装備しています。簡易デバッガを使用する際の手順は次の通りです。

① 簡易デバッガの操作は TELNET 経由で行いますので、動作パラメータの「起動プログラム」に「TelnetSv」を追加設定します。

② デバッグを行うユーザプログラム名の設定を「起動プログラム」から削除します。

本製品の簡易デバッガは以下のコマンドをサポートします。コマンドとパラメータは、ロードの場合のプログラム名を除き、大文字と小文字を区別しません。また、複数のパラメータを指定する場合はスペースで区切ります。

コマンド	パラメータ	内　　容
B	アドレス､ブレークポイント番号	ブレークポイントの設定・参照
D	アドレス範囲、サイズ	メモリダンプ
G	――――	プログラムの実行
L	プログラム名	デバッグプログラムのロード
R	――――	レジスタの表示
S	アドレス、サイズ	メモリの編集
T	――――	シングルステップ実行
U	アドレス範囲	逆アセンブル
Q	――――	簡易デバッガの終了

8.1 起動と終了

本製品を「通常動作モード」で動作させ、TELNET クライアントからログインします。TELNET のプロンプトに対し、

```
EX DEBUG
```

と入力して Enter キーを押すと簡易デバッガのプロンプト（-）を表示します。

プロンプトに対して

```
Q
```

を入力すると簡易デバッガを終了し、TELNET のプロンプトを表示します。デバッグ中のプログラムも終了してメモリを解放します。

8.2 デバッグプログラムのロード

以下のコマンドを入力するとプログラムを RAM にロードします。

```
L プログラム名
```

ロードするプログラムはあらかじめ本製品のフラッシュメモリにダウンロードされている必要があります。

すでにロードされているプログラムは RAM 上から解放され、新たにロードしたプログラムで置換えられます。また、ブレークポイントの設定も初期化されます。

「Sample」というプログラムをロードするには、

```
L Sample
```

と入力して Enter キーを押します。

8.3 ブレークポイントの設定・参照

以下のコマンドを入力すると指定されたアドレスにブレークポイントを設定します。

```
B [Address [0|1]]
```

本デバッガでは、ブレークポイントを最大 2 箇所に指定できます。指定可能なブレークポイント番号は 0 または 1 のいずれかです。ブレークポイント番号の指定を省略した場合、0 を設定したものとみなします。パラメータを省略すると現在設定されているブレークポイントを表示します。アドレスは 16 進数で入力してください。

c060100 番地にブレークポイントを設定したい場合、

```
B c060100
```

と入力して Enter キーを押します。

8.4 プログラムの実行

プロンプトに対して

```
G
```

を入力するとロードされているプログラムの実行を開始します。ブレークポイントが設定されている場合、そのアドレスの命令の実行前で停止します。停止するとその時点のレジスタの値と、停止したアドレスのニーモニックを表示します。

8.5 シングルステップ実行

プロンプトに対して

```
T
```

を入力するとロードされているプログラムを 1 命令だけ実行します。実行後のレジスタの値と、停止したアドレスのニーモニックを表示します。

TRAPA 命令でシングルステップ実行を行った場合、無条件トラップから復帰した時点で停止します。

RTE 命令でシングルステップ実行を行わないでください。RTE 命令でシングルステップ実行を行った場合の結果は予測できません。

8.6 レジスタの表示

プロンプトに対して

```
R
```

を入力すると現在のレジスタの値を以下のように表示します。プログラムがロードされている場合、レジスタの値の後に現在のプログラムカウンタが示すアドレスのニーモニックを表示します。値はすべて 16 進数表示です。

```
R0   = 00000000   R1   = 00000000   R2   = 00000000   R3   = 00000000
R4   = 00000000   R5   = 00000000   R6   = 00000000   R7   = 00000000
R8   = 00000000   R9   = 00000000   R10  = 00000000   R11  = 00000000
R12  = 00000000   R13  = 00000000   R14  = 00000000   R15  = 00000000
GBR  = 00000000   MACH = 00000000   MACL = 00000000   PR   = 00000000
VBR  = 00000000   PC   = 0C060000   SR   = 40000000
```

8.7 逆アセンブル

以下のコマンドを入力すると指定したアドレスの範囲を逆アセンブルして表示します。

```
U [StartAddress [EndAddress]]
```

終了アドレスを省略すると、16 命令分を表示します。開始アドレスと終了アドレスを省略すると、前回の続きの 16 命令分を表示します。アドレスは 16 進数で入力してください。

```
U 0
```

と入力して Enter キーを押すと 0 番地から 16 命令分を逆アセンブルし、以下のように表示します。

```
00000000  DF33  MOV.L  H'000000D0,R15
00000002  D031  MOV.L  H'000000C8,R0
00000004  A006  BRA    H'00000014
00000006  402E  LDC    R0,VBR
00000008  00A0  ?
0000000A  C2FF  MOV.L  R0,@(H'03FC,GBR)
0000000C  FE4C  ?
0000000E  0103  BSRF   R1
00000010  6100  MOV.B  @R0,R1
00000012  0000  ?
00000014  D830  MOV.L  H'000000D8,R8
00000016  480B  JSR    @R8
00000018  0009  NOP
0000001A  D02C  MOV.L  H'000000CC,R0
0000001C  9348  MOV.W  H'000000B0,R3
0000001E  330C  ADD    R0,R3
```

8.8 メモリダンプ

以下のコマンドを入力すると指定した範囲のメモリの内容を表示します。

```
D [StartAddress [EndAddress [B|W|L]]]
```

表示サイズは B または W または L を指定し、それぞれ 1 バイト、2 バイト、4 バイト単位で表示します。表示サイズを省略すると前回の表示サイズで表示します。終了アドレスを省略すると、256 バイト分を表示します。開始アドレスと終了アドレスを省略すると、前回の続きの 256 バイト分を表示します。アドレスは 16 進数で入力してください。

```
D 10000 w
```

と入力して Enter キーを押すと 10000 番地から 256 バイト分のデータをワード単位で以下のように表示します。

```
00010000  4B65 726E 656C 2020 4D4F 5400 8001 0018
00010010  8000 0008 0104 6300 E1FF D021 2012 D221
00010020  D31D 332C 6033 6405 6501 644D 655D E000
00010030  6125 301C 600D 3320 8BFA 340C 350C 644D
00010040  655D 2448 8913 2558 8911 D114 0002 201B
00010050  400E D815 C71B 911B 6403 E000 9219 E32F
00010060  9618 E520 2100 2230 2600 482B 2150 DF0F
00010070  D10F D010 412E 400B 0009 D10F 410B 0009
```

8.9 メモリ編集

以下のコマンドを入力すると指定したアドレスからメモリの内容を変更できます。

```
S [Address [B|W|L]]
```

編集サイズは B または W または L を指定し、それぞれ 1 バイト、2 バイト、4 バイト単位で入力値を書き込みます。編集サイズを省略すると前回の編集サイズで入力を受け付けます。アドレスを省略すると、前回の続きから入力を受け付けます。アドレスおよびデータは 16 進数で入力してください。

```
S c060000
```

と入力して Enter キーを押すと c060000 番地からデータの入力待ちになります。以下のように表示します。

```
0C060000  00 ->
```

データを入力して Enter を押すと次のアドレスに進みます。以下のようになります。

```
0C060000  00 -> 1   ← 「1(01)」を入力した
0C060001  00 ->
```

編集を終了する場合はデータを入力せずに Enter キーを押します。

9. ユーザサポートについて

① 本書に関するお問い合わせ及びユーザサポートは別途有料とさせていただきます。
② ユーザプログラムの不適切なご使用による BLC-100 故障の場合は保障期間内でも有料となります。

株式会社 ビッツ

〒１４１－００３１
東京都品川区西五反田８丁目８番２０号　レナウン本社ビル８階

ＴＥＬ　　：　０３－３７７９－２１９０
ＦＡＸ　　：　０３－３７７９－２１９８
Ｅ－ｍａｉｌ　：　ｍｉｅｃｈａｎ@ｂｉｔｓ．ｃｏ．ｊｐ
ホームページ　：　ｈｔｔｐ：／／ｍｉｅｃｈａｎ．ｊｐ