リアルタイム APIトレーニングテキスト

アドバンスドコース　HDLC

2009/12/01 第3版
コンカレント日本株式会社
プロフェッショナルサービス部

HDLC サーバユーザインタフェース

サーバプロセスへの通信のためのインタフェースライブラリ（全９関数）

hdlc_initialize() ：HDLC 通信を行うための初期化
hdlc_accept() ：接続を受け付ける
hdlc_connect() ：接続要求の送信
hdlc_disconnect() ：切断要求の送信
hdlc_recv() ：データの受信
hdlc_transmit() ：データの送信
hdlc_return() ：接続・接続受付・データ送信・切断の結果取得
hdlc_terminate() ：HDLC 通信の終了処理
hdlc_evtrig() ：AuxClk 入力をイベントとして登録

・ 非同期 I/O

一部の関数は非同期 I/O（ノンブロック）になっており、通信結果を待たずにユーザへ制御が戻ります。非同期 I/O の 4 つの関数については hdlc_return 関数を用いて結果を確認することができます。

hdlc_accept()、*hdlc_connect()*、*hdlc_disconnect()*、*hdlc_transmit()*、*hdlc_evtrig()*

ノンブロック：接続・接続受付・切断・データ送信の成否に関わらず即時にユーザへ制御が戻る。

hdlc_return()

ブロック：接続・接続受付・切断・データ送信の成否を確認。結果が返ってくるまでユーザへ制御を戻さない。ただしタイムアウト時間を指定することは可能。

hdlc_recv()

ブロック：データを受信するまで（または切断されるまで）ユーザへ制御を戻さない。ただしタイムアウト時間を指定することは可能。

hdlc_initialize

int hdlc_initialize(int board, int channel, int loud);

説明

hdlc_initialize は HDLC 通信に使用するリソースを初期化し、以降の通信にて使用するリンク ID を返します。

パラメータ board には、通信に使用するボード番号を指定します。
ボードが 1 枚の場合は board に 0 を設定してください。
パラメータ channel には、通信に使用するチャンネル番号を指定します。
チャンネル番号には 0 から 3 までの 4 つのチャンネルが指定できます。
パラメータ loud は、デバッグ出力用の引数です。
loud に 0 以外の数値を与えると、エラー時に標準出力にエラーメッセージが出力されます。

返り値

HDLC 通信に使用するリンク ID を返します。
エラーが発生した場合には-1 を返し、errno に値が設定されます。

エラー

ENXIO

無効なボード番号またはチャンネル番号が与えられた

EMFILE

プロセスのオープンしているディスクリプタ数が最大値をオーバしている

ENOENT

HDLC 通信のためのサーバプロセスに繋がらない。

サーバプロセスの起動を確認してください。

EAGAIN

サーバプロセスのメッセージ受信バッファがすでにいっぱいである

サーバプロセスの起動を確認してください。

EMSGSIZE

サーバプロセスへ送信するメッセージのサイズエラー

フレームサイズを確認し、サーバプロセス・ライブラリの再コンパイルをしてください

EINTR

シグナルによる割り込み

hdlc_accept

int hdlc_accept(int lid, int *id, int loud);

説明

hdlc_accept は HDLC 通信における接続の受け付けをサーバプロセスへ指示します。
接続を受け付けたかどうかは、hdlc_return を呼び出して結果を確認する必要があります。
hdlc_accept では接続受け付けの成功・失敗確認を行わず、サーバプロセスへの指示が完了すると即座に返ります（ブロックは行いません）。

パラメータ lid は、hdlc_initialize によって得られた HDLC 通信のためのリンク ID です。
パラメータ id には、後で hdlc_return にて結果を確認するための ID が設定されます。
パラメータ loud は、デバッグ出力用の引数です。
loud に 0 以外の数値を与えると、エラー時に標準出力にエラーメッセージが出力されます。

返り値

サーバプロセスへの指示が正常に行われた場合、0 を返します。
エラーが発生した場合には-1 を返し、errno に値が設定されます。

エラー

EBADF
　無効なリンク ID が与えられた
EAGAIN
　サーバプロセスのメッセージ受信バッファがすでにいっぱいである
　サーバプロセスの起動を確認してください。
EMSGSIZE
　サーバプロセスへ送信するメッセージのサイズエラー
　フレームサイズを確認し、サーバプロセス・ライブラリの再コンパイルをしてください
EINTR
　シグナルによる割り込み
ENOBUF

HDLC リンクのリソース不足。

hdlc_initialize を呼び出してリソースの初期化を行う必要があります。

hdlc_initialize の呼び出し後は、呼び出し前の hdlc_connect、hdlc_accept、hdlc_disconnect、hdlc_transmit の id が全て無効になり、結果を確認できなくなることに注意してください。

注意

HDLC 通信における接続受け付けの成功・失敗は、hdlc_accept で設定された id を用いて、hdlc_return を呼び出し確認してください。

hdlc_return は接続を受け付けるまで、またはユーザから hdlc_disconnect が呼び出されるまでプロセスをブロックします。

hdlc_return の呼び出しによって確認できる結果は以下の通りです。

RESULT_OK

相手からの接続要求を受け付け、接続が完了した。

または既に接続状態である。

RESULT_NG

接続待ち受け中に hdlc_disconnect を呼び出された

RESULT_REQ_DUP

connect、disconnect 要求処理中。

または既に他の accept 要求を処理中である。

制限

HDLC の ABM モードでは相手先固定のため、同時に張ることのできるリンクコネクション数は１つだけです。そのため、複数のプロセス・スレッドから同一のチャンネルに対して hdlc_connect・hdlc_accept を呼び出した場合、各コネクションの動作は保障できません。

また、hdlc_accept の呼び出し後は必ず hdlc_return を呼び出して結果を確認してください。

hdlc_return は内部にバッファを持ち、要求に対する結果を保存しますが、ユーザに未返答の結果が溜まるとバッファが足りなくなり、初期化が必要になります。

hdlc_connect

int hdlc_connect(int lid, int *id, int loud);

説明

hdlc_connect は HDLC 通信における接続要求の送信をサーバプロセスへ指示します。接続が成功したかどうかは、hdlc_return を呼び出して結果を確認する必要があります。hdlc_connect では接続の成功・失敗確認を行わず、サーバプロセスへの指示が完了すると即座に返ります（ブロックは行いません）。

パラメータ lid は、hdlc_initialize によって得られた HDLC 通信のためのリンク ID です。
パラメータ id には、後で hdlc_return にて結果を確認するための ID が設定されます。
パラメータ loud は、デバッグ出力用の引数です。
loud に 0 以外の数値を与えると、エラー時に標準出力にエラーメッセージが出力されます。

返り値

サーバプロセスへの指示が正常に行われた場合、0 を返します。
エラーが発生した場合には-1 を返し、errno に値が設定されます。

エラー

EBADF
無効なリンク ID が与えられた

EAGAIN
サーバプロセスのメッセージ受信バッファがすでにいっぱいである
サーバプロセスの起動を確認してください。

EMSGSIZE
サーバプロセスへ送信するメッセージのサイズエラー
フレームサイズを確認し、サーバプロセス・ライブラリの再コンパイルをしてください

EINTR
シグナルによる割り込み

ENOBUF
HDLC リンクのリソース不足。
hdlc_initialize を呼び出してリソースの初期化を行う必要があります。

hdlc_initialize の呼び出し後は、呼び出し前の hdlc_connect、hdlc_accept、hdlc_disconnect、hdlc_transmit の id が全て無効になり、結果を確認できなくなることに注意してください。

注意

HDLC 通信における接続の成功・失敗は、hdlc_connect で設定された id を用いて、hdlc_return を呼び出し確認してください。

hdlc_return は接続要求の応答が帰ってくるまで、またはタイムアウトになるまでプロセスをブロックします。（タイムアウト時間はサーバプロセスによって決められます）

hdlc_return の呼び出しによって確認できる結果は以下です。

RESULT_OK

通信相手が接続要求を受け付け、接続が完了した。
または既に接続状態である。

RESULT_NG

通信相手が接続を拒否し、接続できなかった。

RESULT_TIMEOUT

通信相手から接続要求に対する応答が帰って来ないためタイムアウトした。

RESULT_REQ_DUP

accept 要求処理中。または既に他の connect 要求を処理中である。

制限

HDLC の ABM モードでは相手先固定のため、同時に張ることのできるリンクコネクション数は１つだけです。そのため、複数のプロセス・スレッドから同一のチャンネルに対して hdlc_connect・hdlc_accept を呼び出した場合、各コネクションの動作は保障できません。

また、hdlc_connect の呼び出し後は必ず hdlc_return を呼び出して結果を確認してください。hdlc_return は内部にバッファを持ち、要求に対する結果を保存しますが、ユーザに未返答の結果が溜まるとバッファが足りなくなり、初期化が必要になります。

hdlc_disconnect

int hdlc_connect(int lid, int *id, int loud);

説明

hdlc_disconnect は HDLC 通信における接続切断をサーバプロセスへ指示します。
接続切断が正常に終了したかどうかは、hdlc_return を呼び出して結果を確認する必要があります。hdlc_disconnect では接続切断の成功・失敗確認を行わず、サーバプロセスへの指示が完了すると即座に返ります（ブロックは行いません）。

パラメータ lid は、hdlc_initialize によって得られた HDLC 通信のためのリンク ID です。
パラメータ id には、後で hdlc_return にて結果を確認するための ID が設定されます。
パラメータ loud は、デバッグ出力用の引数です。
loud に 0 以外の数値を与えると、エラー時に標準出力にエラーメッセージが出力されます。

返り値

サーバプロセスへの指示が正常に行われた場合、0 を返します。
エラーが発生した場合には-1 を返し、errno に値が設定されます。

エラー

EBADF
無効なリンク ID が与えられた

EAGAIN
サーバプロセスのメッセージ受信バッファがすでにいっぱいである
サーバプロセスの起動を確認してください。

EMSGSIZE
サーバプロセスへ送信するメッセージのサイズエラー
フレームサイズを確認し、サーバプロセス・ライブラリの再コンパイルをしてください

EINTR
シグナルによる割り込み

ENOBUF
HDLC リンクのリソース不足。

hdlc_initialize を呼び出してリソースの初期化を行う必要があります。
hdlc_initialize の呼び出し後は、呼び出し前の hdlc_connect、hdlc_accept、
hdlc_disconnect、hdlc_transmit の id が全て無効になり、結果を確認できなくなることに注意してください。

注意

HDLC 通信における接続切断の成功・失敗は、hdlc_disconnect で設定された id を用いて、
hdlc_return を呼び出し確認してください。
hdlc_return は切断要求の応答が帰ってくるまで、またはタイムアウトになるまでプロセスをブロックします。(タイムアウト時間はサーバプロセスによって決められます)

hdlc_return の呼び出しによって確認できる結果は以下です。

RESULT_OK
通信相手が接続切断要求を受け付け、切断が完了した。
または既に接続切断状態である。

RESULT_NG
disconnect 要求処理中に hdlc_connect または hdlc_accept が呼び出された。

RESULT_TIMEOUT
通信相手から接続切断要求に対する応答が帰って来ないためタイムアウトした。

RESULT_REQ_DUP
既に他の disconnect 要求を処理中である。

制限

また、hdlc_disconnect の呼び出し後は必ず hdlc_return を呼び出して結果を確認してください。hdlc_return は内部にバッファを持ち、要求に対する結果を保存しますが、ユーザに未返答の結果が溜まるとバッファが足りなくなり、初期化が必要になります。

hdlc_recv

int hdlc_recv(int lid, char *buf, int bufsize, struct timespec timeout, int loud);

説明

hdlc_recv は HDLC 通信におけるデータの受信を行います。
受信するデータがないとき、timeout 時間を過ぎるまで hdlc_recv はプロセスをブロックし、
受信データの到着を待ち受けます。ブロック中に接続が切断された場合は、ブロックを解除し 0 を返します。
パラメータ lid は、hdlc_initialize によって得られた HDLC 通信のためのリンク ID です。
パラメータ buf には､受信したデータを保存するバッファへのポインタを指定します。
パラメータ datasize には、バッファのサイズを指定します。
datasize に 0 または負の値が設定された場合、hdlc_recv はデータの受信を行わず即座に 0 を返します。
パラメータ timeout には、受信するデータフレームが存在しない場合の最大待ち受け時間を指定します。timeout に NULL が設定された場合は、受信するデータフレームが到着するまでプロセスをブロックし続けます。
パラメータ loud は、デバッグ出力用の引数です。
loud に 0 以外の数値を与えると､エラー時に標準出力にエラーメッセージが出力されます。

返り値

正常にデータを受信した場合、hdlc_recv は受信した HDLC フレームのデータサイズを返します。HDLC 通信における接続が切断された場合、およびパラメータ bufsize に 0 以下の数値が指定された場合には 0 を返します。
タイムアウトまたはエラーが発生した場合には-1 を返し､errno に値が設定されます。

エラー

EBADF
無効なリンク ID が与えられた

EAGAIN
サーバプロセスのメッセージ受信バッファがすでにいっぱいである
サーバプロセスの起動を確認してください。

EMSGSIZE

サーバプロセスへ送信するメッセージのサイズエラー

フレームサイズを確認し、サーバプロセス・ライブラリの再コンパイルをしてください

EINTR

シグナルによる割り込み

ENOBUFS

HDLC リンクのリソース不足。

hdlc_initialize を呼び出してリソースの初期化を行う必要があります。

hdlc_initialize の呼び出し後は、呼び出し前の hdlc_connect、hdlc_accept、hdlc_disconnect、hdlc_transmit の id が全て無効になり、結果を確認できなくなることに注意してください。

ETIMEDOUT

パラメータ timeout で指定した時間内にデータを受信できなかった。

注意

HDLC ではフレーム単位にデータを受信するため、hdlc_rcv における受信データの最大長は 1 フレームあたりのデータの最大長になります。

1 フレームのデータ最大長はデフォルトでは 1018Byte になります。

そのため、例えば 2000Byte のデータを受信するためには hdlc_recv を二度呼び出す必要があります。

hdlc_transmit

int hdlc_transmit(int lid, char *data, int datasize, int *id, int loud);

説明

hdlc_transmit は HDLC 通信におけるデータ送信をサーバプロセスへ指示します。
データが正常に送信されたかどうかは、hdlc_return を呼び出して結果を確認する必要があります。hdlc_transmit ではデータ送信の成功・失敗確認を行わず、サーバプロセスへの指示が完了すると即座に返ります（ブロックは行いません）。

パラメータ lid は、hdlc_initialize によって得られた HDLC 通信のためのリンク ID です。
パラメータ data には、送信するデータを指定します。
パラメータ datasize には、送信するデータの長さを指定します。
パラメータ id には、後で hdlc_return にて結果を確認するための ID が設定されます。
パラメータ loud は、デバッグ出力用の引数です。
loud に 0 以外の数値を与えると、エラー時に標準出力にエラーメッセージが出力されます。

返り値

サーバプロセスへの指示が正常に行われた場合、0 を返します。
エラーが発生した場合には-1 を返し、errno に値が設定されます。

エラー

EBADF
　無効なリンク ID が与えられた
EAGAIN
　サーバプロセスのメッセージ受信バッファがすでにいっぱいである
　サーバプロセスの起動を確認してください。
EMSGSIZE
　サーバプロセスへ送信するメッセージのサイズエラー
　フレームサイズを確認し、サーバプロセス・ライブラリの再コンパイルをしてください
EINTR
　シグナルによる割り込み

ENOBUFS

datasize が送信可能な最大サイズを超えている。

ENOBUF

HDLC リンクのリソース不足。

hdlc_initialize を呼び出してリソースの初期化を行う必要があります。

hdlc_initialize の呼び出し後は、呼び出し前の hdlc_connect、hdlc_accept、

hdlc_disconnect、hdlc_transmit の id が全て無効になり、結果を確認できなくなることに注意してください。

注意

HDLC 通信におけるデータ送信の成功・失敗は、hdlc_transmit で設定された id を用いて、hdlc_return を呼び出し確認してください。

hdlc_return は全てのデータフレームの送信成功・失敗を確認するまで、またはタイムアウトになるまでプロセスをブロックします。(タイムアウト時間はサーバプロセスによって決められます。)

hdlc_return の呼び出しによって確認できる結果は以下の通りです。

RESULT_OK

要求されたデータの全てを通信相手が受信した。

RESULT_NG

要求されたデータの全てを通信相手が受信できなかった。

または一部のデータのみ通信相手が受信した。

RESULT_TIMEOUT

通信相手から受信確認応答が帰って来ないためタイムアウトした。

（一部のデータのみ通信相手が受信した場合を含む）

RESULT_PEER_BUSY

通信相手がビジー状態のため、一部のデータを通信相手が受信できなかった。

この場合、通信相手がビジー状態を回復するまでデータの送信はできません。

上記結果に対して、通信相手から受信確認が取れたバイト数が

hdlc_return のパラメータ size に設定されます。

制限

一度に送信可能な最大サイズは以下になります。（デフォルトでは 4169728Byte）

SIO4_HDLC_FRAME_DATA_SIZE * TRANSMIT_MSG_RESOURCE_MAX

また、hdlc_transmit の呼び出し後は必ず hdlc_return で結果を確認してください。

hdlc_return は内部にバッファを持ち、要求に対する結果を保存しますが、

ユーザに未返答の結果が溜まるとバッファが足りなくなり、初期化が必要になります。

hdlc_return

名前

hdlc_return - hdlc_accept、hdlc_connect、hdlc_disconnect、hdlc_transmit,hdlc_evtrig の結果を確認する。

書式

gcc [ption...] file -lsio4hdlc -lpthread -lrt -lccur_rt ...

#include "hdlc.h"
#include "hdlclib.h"
int hdlc_return(int lid, int id, int *result, int *size, struct timespec *timeout, int loud);

説明

hdlc_return を呼び出すことによってhdlc_accept（接続受付要求）、hdlc_connect（接続要求）、hdlc_disconnect（接続切断要求）、hdlc_transmit （データ送信要求)、hdlc_evtrig(イベント要求)の結果を確認することができます。
上記５つの関数はサーバプロセスに各要求を出すだけで、通信相手に各要求 が受 け入れられたかどうかを確認しません。つまり通信相手の応答を待たずに終了します。
hdlc_return では各要求の成否がわかるまでプロセスをブロックし、各要求の結果を返します。

パラメータ lid は、hdlc_initialize によって得られた HDLC 通信のためのリンク ID です。
パラメータ id には hdlc_accept、hdlc_connect、hdlc_disconnect、hdlc_transmit,hdlc_evtrig の呼び出しによって取得した要求 id を設定します。
パラメータ result には、要求 id に対する結果（要求に対する成否）が設定さ れます。
パラメータ size には、要求 id に対する送信成功バイト数が設定されます。
ただし、要求 id が hdlc_transmit の場合のみ正の数値が設定され、 hdlc_evtrig

の場合には、サイズではなくサーバープロセスのプロセス番号がその他の要 求に対しては 0 が設定されます。
パラメータ timeout には、要求結果待ち受け時間の最大値を設定します。
timeout に NULL が設定された場合は、要求の成否がわかるまでプロセスをブロッ
クし続けます。
パラメータ loud は、デバッグ出力用の引数です。
loud に 0 以外の数値を与えると、エラー時に標準出力にエラーメッセージが出力されます。

返り値

要求 id に対する結果を正常に取得できた場合、0 を返します。タイムアウトまたはエラーが発生した場合には-1 を返し、errno に値が設定されます。

エラー

EINVAL	無効な要求 id が与えられた または既にその要求 id に対する結果を返答済みである
EBADF	無効なリンク ID が与えられた
EMSGSIZE	サーバプロセスへ送信するメッセージのサイズエラー。フレームサイズを確認し、サーバプロセス・ライブラリの再コンパイルをしてください
EINTR	シグナルによる割り込み
ENOBUF	HDLC リンクのリソース不足。 hdlc_initialize を呼び出してリソースの初期化を行う必要があります。 hdlc_initialize の呼び出し後は、呼び出し前の hdlc_connect 、hdlc_accept 、 hdlc_disconnect 、hdlc_transmit の id が全て無効になり、結果を確認できなくなることに 注意してください。
ETIMEDOUT	パラメータ timeout で指定した時間内に要求結果が返って こなかった。 再度同じ要求 ID を用いて結果を確認してください。

注意

hdlc_return にて 1 度結果を取得した要求 id を用いて再度 hdlc_return を呼び出しても結果を取得することはできず、エラーが返ります。

result に設定される値は、hdlc_accept、hdlc_connect、hdlc_disconnect、hdlc_transmit によって違います。各関数の説明を参照してください。

hdlc_terminate

int hdlc_terminate(int lid, int loud);

説明

hdlc_terminate は HDLC 通信の接続を切断し、使用しているリソースを開放します。

パラメータ lid には、hdlc_initialize によって得られたリンク ID を設定します。
パラメータ loud は、デバッグ出力用の引数です。
loud に 0 以外の数値を与えると、エラー時に標準出力にエラーメッセージが出力されます。

返り値

サーバプロセスへの指示が正常に行われた場合、0 を返します。
エラーが発生した場合には-1 を返し、errno に値が設定されます。

エラー

EBADF
　無効なリンク ID が与えられた

EAGAIN
　サーバプロセスのメッセージ受信バッファがすでにいっぱいである
　サーバプロセスの起動を確認してください。

EMSGSIZE
　サーバプロセスへ送信するメッセージのサイズエラー
　フレームサイズを確認し、サーバプロセス・ライブラリの再コンパイルをしてください

EINTR
　シグナルによる割り込み

hdlc_evtrig

int hdlc_evtrig(int lid, int *id, int type,int loud);

説明

hdlc_evtrig は AuxClk をイベントとして受け付けるようにサーバプロセスへ指示します。

登録を受け付けたかどうかは、hdlc_return を呼び出して結果を確認する必要があります。

hdlc_evtrig では接続受け付けの成功・失敗確認を行わず、サーバプロセスへの指示が完了すると即座に返ります（ブロックは行いません）。

パラメータ lid は、hdlc_initialize によって得られた HDLC 通信のためのリンク ID です。

パラメータ id には、後で hdlc_return にて結果を確認するための ID が設定されます。

パラメータ type は、イベントの引数です。

type=0 の場合には、サーバープロセスが立ち上がり、立ち下がりの変化を検出し、SIGEVTRIG シグナルを１、０を値として送信します。

type=1 の場合には、サーバープロセスが立ち上がりのみを検出し、

type=2 の場合には、サーバープロセスが立ち下がりのみを検出してシグナルを送信します。

パラメータ loud は、デバッグ出力用の引数です。

loud に 0 以外の数値を与えると、エラー時に標準出力にエラーメッセージが出力されます。

返り値

サーバプロセスへの指示が正常に行われた場合、0 を返します。

エラーが発生した場合には-1 を返し、errno に値が設定されます。

エラー

EBADF

無効なリンク ID が与えられた

EAGAIN

サーバプロセスのメッセージ受信バッファがすでにいっぱいである

サーバプロセスの起動を確認してください。

EMSGSIZE

サーバプロセスへ送信するメッセージのサイズエラー

フレームサイズを確認し、サーバプロセス・ライブラリの再コンパイルをしてください

EINTR

シグナルによる割り込み

ENOBUF

HDLC リンクのリソース不足。

hdlc_initialize を呼び出してリソースの初期化を行う必要があります。

hdlc_initialize の呼び出し後は、呼び出し前の hdlc_connect、hdlc_evtrig、

hdlc_disconnect、hdlc_transmit の id が全て無効になり、結果を確認できなくなることに注意してください。

注意

HDLC 通信における接続受け付けの成功・失敗は、hdlc_evtrig で設定された id を用いて、hdlc_return を呼び出し確認してください。

hdlc_return は接続を受け付けるまで、またはユーザから hdlc_disconnect が呼び出されるまでプロセスをブロックします。

hdlc_return の呼び出しによって確認できる結果は以下です。

RESULT_OK

要求を受け付け、イベントの登録が完了した。

RESULT_NG

要求を受け付けることができなかった。

制限

hdlc_evtrig の呼び出し後は必ず hdlc_return を呼び出して結果を確認してください。

hdlc_return は内部にバッファを持ち、要求に対する結果を保存しますが、ユーザ に未返答の結果が溜まるとバッファが足りなくなり、初期化が必要になります。

AUXCONFIG=が０の場合には、イベントの送受信は、ディセーブルです。

信号は入力状態になります。

イベントの受信は、AUXCONFIG=1 のみ可能です。

信号は入力状態になります。

イベントの送信は、AUXCONFIG=2,AUXCONFIG=3 のみ可能です。

信号は出力状態になります。

この値が２では、出力が Low で出力され、３の場合に High が出力されます。

どの場合 でも、コンフィグレーションファイルに設定する必要があります。

HDLC サンプルプログラム

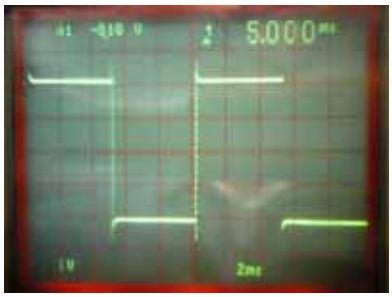
プログラム sio4read.c（イベント送信、データ受信、DTE 設定）

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <errno.h>
#include <string.h>
#include <signal.h>
#include <errno.h>
#include "hdlc.h"
#include "hdlclib.h"

#define TESTBOARD 0
#define TESTLOUD  0
#define TESTMSGSIZE             64
#define MSEC            10                              //100Hz
#define RESOLUTION              5000

void readtime(struct timespec *nowtime)
{
        clock_gettime(CLOCK_REALTIME,nowtime);
}

void adjust(struct timespec *start,struct timespec *finish, float *realtime)
{
        int     sec,nsec;


        sec = finish->tv_sec - start->tv_sec;
        nsec = finish->tv_nsec - start->tv_nsec;
        if (nsec<0)
        {
                sec --;
                nsec += 1000000000;
        }
        *realtime = (float)(nsec/1000)+(float)(sec*1000000);
}

timer_t create_posix_timer(int signum,int sec,int nsec)
{
        static struct sigevent event;
        static timer_t timer;
        static struct itimerspec itspec;

        /* Create the POSIX timer to generate signum */
        event.sigev_notify = SIGEV_SIGNAL;
        event.sigev_signo = signum;
        if (timer_create((clockid_t)CLOCK_REALTIME,&event,&timer)==(-1))
        {
                fprintf(stderr, "create_posix_timer() Cannot create timer - %s¥n",
                strerror(errno));
                return ((timer_t)-1);
        }
        /*      Set the initial delay and period of the timer.
        This also arms the timer */
        clock_gettime(CLOCK_REALTIME,&itspec.it_value);
        itspec.it_value.tv_sec += 3;
        itspec.it_interval.tv_sec = sec;
        itspec.it_interval.tv_nsec =    nsec;
        if (timer_settime( timer, TIMER_ABSTIME, &itspec, NULL)==(-1))
        {
                return ((timer_t)-1);
        }
        return(timer);
```

```
}

static int global_setup_signal(int signo, void (*interrupt_handler)(int,siginfo_t *, void *))
{
        static struct sigaction newact;
        static sigset_t set,oset;

        if (sigprocmask(0,NULL,&set)==(-1))
        {
                return(-1);
        }
        sigdelset(&set,signo);
        if (sigprocmask(SIG_SETMASK,&set,&oset)==(-1))
        {
                return(-2);
        }

        sigemptyset(&newact.sa_mask);
        sigaddset(&newact.sa_mask,signo);
        newact.sa_sigaction = interrupt_handler;
        newact.sa_flags = SA_SIGINFO|SA_RESTART;
        if (sigaction(signo, &newact, 0)==(-1))
        {
                return(-3);
        }
        return(0);
}

static int Stop_flag=0;
static void sigint_handler(int signo, siginfo_t *siginfo, void *misc)
{
        fprintf(stderr,"¥nSIG%d DETECT¥n",signo);
        Stop_flag = 1;
}


int main(int argc, char *argv[])
{
            int   channel;
            char rbuf[TESTMSGSIZE];
            int i,ret;
            int id;
            int lid;
            int result;
            int loud = TESTLOUD;
            struct timespec timeout;
            int pid,signum;
        timer_t timer_id;
        siginfo_t info;
            int loop=0;
        struct timespec w100u={0,100000},t0,t1,T0,T1;
        float real=0.0,REAL=0.0;
        int fail=0;
            union sigval value;
            sigset_t   diset,eiset;
            int his[RESOLUTION];

            if(argc != 2){
                        printf("Usage: %s channel¥n", argv[0]);
                        printf(" channel: 0 - 3¥n");
                        return 1;
            }
            channel = atoi(argv[1]);
            if( channel < 0 || 3 < channel ){
                        printf("Usage: %s channel¥n", argv[0]);
                        printf(" channel: 0 - 3¥n");
                        return 1;
            }

        signum = SIGRTMIN;
        //SIGRTMIN 割り込みからプロセスを保護する
        sigemptyset(&diset);
        sigaddset(&diset, signum);
```

```
sigaddset(&diset, SIGINT);
sigprocmask(SIG_BLOCK, &diset, NULL);
//SIGRTMIN 割り込み待ちのマスクを生成
sigemptyset(&eiset);
sigaddset(&eiset, signum);

timer_id = create_posix_timer(signum,0,MSEC * 1000000); //周期割り込みを生成
if (timer_id<0)
{
        fprintf(stderr, "Cannot set posix timer - %s¥n", strerror(errno));
        exit(1);
}
   global_setup_signal(SIGINT,sigint_handler);

   printf("test start¥n");
   lid = hdlc_initialize(TESTBOARD, channel, loud);
   if(lid == -1){
            printf("hdlc_initialize failure¥n");
            goto term;
   }
   printf("accept start¥n");
   ret = hdlc_accept(lid, &id, loud);
   if(ret == -1){
            printf("hdlc_accept failure¥n");
            goto term;
   }
   timeout.tv_sec = 10;
   timeout.tv_nsec = 0;
   ret = hdlc_return(lid, id, &result, NULL, &timeout, loud);
   if(ret == -1){
            printf("hdlc_return failure¥n");
            goto discon;
   }
   switch(result){
            case RESULT_OK:
                     printf("accept result OK¥n");
                     break;
            case RESULT_NG:
                     printf("accept result NG¥n");
                     goto term;
            case RESULT_REQ_DUP:
                     printf("accept result request dup¥n");
                     goto term;
   }
printf("evtrig start¥n");
ret = hdlc_evtrig(lid, &id,0, loud);
if(ret == -1){
        printf("hdlc_evtrig failure¥n");
        goto term;
}

timeout.tv_sec = 10;
timeout.tv_nsec = 0;
ret = hdlc_return(lid, id, &result, &pid, &timeout, loud);              //プロセス ID を得る
if(ret == -1){
        printf("hdlc_return failure¥n");
        goto discon;
}
switch(result){
        case RESULT_OK:
                printf("evtrig result OK [pid:%d]¥n",pid);
                break;
        case RESULT_NG:
                printf("evtrig result NG¥n");
                goto term;
        case RESULT_REQ_DUP:
                printf("evtrig result request dup¥n");
                goto term;
}
   memset(his,0,sizeof(his));
   fail=0;
   loop =0;
   readtime(&T0);
```

```
		value.sival_int =1;		sigqueue(pid,SIGEVTRIG,value);

		//最初は0これを忘れると割り込みがかからないことがある
		value.sival_int =0;		sigqueue(pid,SIGEVTRIG,value);
		for(loop=0;loop<3600*24*(1000/MSEC);loop++)
		{
			sigwaitinfo(&eiset,&info);					//10ms タイマー割り込み待ち
			if ((fail)||(Stop_flag)) break;
			readtime(&t0); readtime(&T1);adjust(&T0,&T1,&REAL);T0=T1;
			printf("%d recv data start %f¥n",loop,REAL);
			value.sival_int =1; sigqueue(pid,SIGEVTRIG,value);		//High 出力
			ret = hdlc_recv(lid, rbuf, TESTMSGSIZE, &timeout, loud);		//受信待ち
			readtime(&t1); adjust(&t0,&t1,&real);
			if (ret != TESTMSGSIZE)
			{
				printf("error %s¥n",strerror(errno));
				fail++;
			}
			if ((int)real>=RESOLUTION)
			{
				printf("ERROR %d:recv data[%d byte] response time %f¥n"
				,loop, ret,real);fail++;break;
			}
			his[(int)(real)]++;
			if ((real<900)|| (real>1100))
			{
				printf("%d:recv data[%d byte] response time %f¥n",loop, ret,real);
			}
			//50%デューティの波形を作る
		do
		{
		    nanosleep(&w100u,NULL);
		    readtime(&t1); adjust(&t0,&t1,&real);
		} while (real<(float)(MSEC*1000)/2.0);
		value.sival_int =0; sigqueue(pid,SIGEVTRIG,value);		//Low 出力
		}
discon:
printf("Results,TotalNumber %d¥n",loop);
printf("Time(usec),Number¥n");
		for(i=0;i<RESOLUTION;i++)
		{
			if(his[i]) printf("%04d , %d¥n",i,his[i]);
		}
		timer_delete(timer_id);
		printf("disconnect start¥n");
		ret = hdlc_disconnect(lid, &id, loud);
		if(ret == -1){
			printf("hdlc_disconnect failure¥n");
			goto term;
		}

		ret = hdlc_return(lid, id, &result, NULL, &timeout, loud);
		if(ret == -1){
			printf("hdlc_return failure¥n");
			goto term;
		}
		switch(result){
			case RESULT_OK:
				printf("disconnect result OK¥n");
				break;
			case RESULT_NG:
				printf("disconnect result NG¥n");
				break;
			case RESULT_REQ_DUP:
				printf("disconnect result request dup¥n");
				break;
			case RESULT_TIMEOUT:
				printf("disconnect result timeout¥n");
				break;
		}
```



```
term:
        ret = hdlc_terminate(lid, loud);
        if(ret == -1){
                printf("hdlc_terminate failure¥n");
        }

        printf("test end¥n");

        return 0;
}
```

プログラム sio4wrire.c（イベント受信、データ送信、ＤＴＥ設定）

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <errno.h>
#include <signal.h>

#include "hdlc.h"
#include "hdlclib.h"

#define TESTBOARD           0
#define TESTLOUD            0
#define TESTMSGSIZE         64
#define MSEC                10

static int global_setup_signal(int signo, void (*interrupt_handler)(int,siginfo_t *, void *))
{
        static struct sigaction newact;
        static sigset_t set,oset;

        if (sigprocmask(0,NULL,&set)==(-1))
        {
                return(-1);
        }
        sigdelset(&set,signo);
        if (sigprocmask(SIG_SETMASK,&set,&oset)==(-1))
        {
                return(-2);
        }

        sigemptyset(&newact.sa_mask);
        sigaddset(&newact.sa_mask,signo);
        newact.sa_sigaction = interrupt_handler;
        newact.sa_flags = SA_SIGINFO|SA_RESTART;
        if (sigaction(signo, &newact, 0)==(-1))
        {
                return(-3);
        }
        return(0);
}

static int Stop_flag=0;
static void sigint_handler(int signo, siginfo_t *siginfo, void *misc)
{
        fprintf(stderr,"¥nSIG%d DETECT¥n",signo);
        Stop_flag = 1;
}


int main(int argc, char *argv[])
{
    int channel;
    char tbuf[TESTMSGSIZE];
    int ret;
    int i;
    int id;
    int lid;
    int loud = TESTLOUD;
    int result = 0;
    int succeed_size;
    float usec;
    struct timespec before;
    struct timespec after;
    struct timespec diff;
    struct timespec timeout;
    int pid,fail=0,loop=0;
    sigset_t    diset,eiset;
    siginfo_t info;
```

```
	if(argc != 2){
		printf("Usage: %s channel¥n", argv[0]);
		printf(" channel: 0 - 3¥n");
		return 1;
	}
	channel = atoi(argv[1]);
	if( channel < 0 || 3 < channel ){
		printf("Usage: %s channel¥n", argv[0]);
		printf(" channel: 0 - 3¥n");
		return 1;
	}
	    //SIGEVTRIG 割り込みからプロセスを保護する
	    sigemptyset(&diset);
	    sigaddset(&diset, SIGEVTRIG);
	    sigaddset(&diset, SIGINT);
	    sigprocmask(SIG_BLOCK, &diset, NULL);
	    //割り込み待ちのマスクを生成
	    sigemptyset(&eiset);
	    sigaddset(&eiset, SIGEVTRIG);
	global_setup_signal(SIGINT,sigint_handler);
	printf("test start¥n");
	lid = hdlc_initialize(TESTBOARD, channel, loud);
	if(lid == -1){
		printf("hdlc_initialize failure¥n");
		goto term;
	}

	printf("connect start¥n");
	ret = hdlc_connect(lid, &id, loud);
	if(ret == -1)
	{
		printf("hdlc_connect failure¥n");
		goto term;
	}

	timeout.tv_sec = 10;
	timeout.tv_nsec = 0;
	ret = hdlc_return(lid, id, &result, NULL, &timeout, loud);
	if(ret == -1)
	{
		printf("hdlc_return failure¥n");
		goto discon;
	}
	switch(result){
		case RESULT_OK:
			printf("connect result OK¥n");
			break;
		case RESULT_NG:
			printf("connect result NG¥n");
			goto term;
		case RESULT_REQ_DUP:
			printf("connect result request dup¥n");
			goto term;
		case RESULT_TIMEOUT:
			printf("connect result timeout¥n");
			goto term;
	}
	    printf("evtrig start¥n");
	    ret = hdlc_evtrig(lid, &id,1, loud);                                    //立ち上がり(1)だけ割り込みを要
求
	    if(ret == -1){
		printf("hdlc_evtrig failure¥n");
		goto term;
	    }
	    timeout.tv_sec = 10;
	    timeout.tv_nsec = 0;
	    ret = hdlc_return(lid, id, &result, &pid, &timeout, loud);
	    if(ret == -1){
		printf("hdlc_return failure¥n");
		goto discon;
	    }
	    switch(result)
	{
```

```
                case RESULT_OK:
                        printf("evtrig result OK [pid:%d]¥n",pid);
                        break;
                case RESULT_NG:
                        printf("evtrig result NG¥n");
                        goto term;
                case RESULT_REQ_DUP:
                        printf("evtrig result request dup¥n");
                        goto term;
        }


        for(loop=0;loop<3600*24*(1000/MSEC);loop++)
        {
                for(i = 0; i < sizeof(tbuf); i++){
                        tbuf[i] = i % (0x100);
                }
                        do
                        {
                                sigwaitinfo(&eiset,&info);
        //SIGEVTRIG SIGINT だけ割り込める
                        if (info.si_signo == SIGINT) break;
                        if (Stop_flag) break;
                        } while ((info.si_signo != SIGEVTRIG)||(info.si_value.sival_int!=1));          //立ち上が
りだけ
                if ((fail)||(Stop_flag)) break;
                printf("%d:transmit start¥n",loop);
                clock_gettime(CLOCK_REALTIME, &before);
                ret = hdlc_transmit(lid, tbuf, sizeof(tbuf), &id, loud);                    //データ送信
                if(ret == -1){
                        printf("hdlc_transmit failure¥n");
                        goto discon;
                }
                printf("transmit data[%dbyte]¥n", sizeof(tbuf));
                ret = hdlc_return(lid, id, &result, &succeed_size, &timeout, loud);
を確認
                if(ret == -1){
                        printf("hdlc_return failure¥n");
                        goto discon;
                }
                switch(result){
                        case RESULT_OK:
                                printf("transmit result OK¥n");
                                break;
                        case RESULT_NG:
                                printf("transmit result NG¥n");
                                break;
                        case RESULT_PEER_BUSY:
                                printf("transmit result peer busy¥n");
                                break;
                        case RESULT_TIMEOUT:
                                printf("transmit result timeout¥n");
                                break;
                }
                printf("transmit data succeed: %d Byte¥n", succeed_size);
                                clock_gettime(CLOCK_REALTIME, &after);
                printf("data transmit end. and result return¥n");
                diff.tv_sec  = after.tv_sec  - before.tv_sec;
                diff.tv_nsec = after.tv_nsec - before.tv_nsec;
                if(diff.tv_nsec < 0){
                        diff.tv_sec--;
                        diff.tv_nsec += 1000000000;
                }
                printf("Sec: %d.%d sec¥n", (int)diff.tv_sec, (int)diff.tv_nsec);
                usec = (float)(diff.tv_sec*1000000) + (float)(diff.tv_nsec/1000);
                printf("%f bit/sec¥n", sizeof(tbuf) * 8 / usec * 1000000);
        }
discon:
        printf("disconnect start¥n");
        ret = hdlc_disconnect(lid, &id, loud);
        if(ret == -1){
                printf("hdlc_disconnect failure¥n");
                goto term;
```

```
	}
	ret = hdlc_return(lid, id, &result, NULL, &timeout, loud);
	if(ret == -1){
		printf("hdlc_return failure¥n");
		goto term;
	}
	switch(result){
		case RESULT_OK:
			printf("disconnect result OK¥n");
			break;
		case RESULT_NG:
			printf("disconnect result NG¥n");
			break;
		case RESULT_REQ_DUP:
			printf("disconnect result request dup¥n");
			break;
		case RESULT_TIMEOUT:
			printf("disconnect result timeout¥n");
			break;
	}

term:
	ret = hdlc_terminate(lid, loud);
	if(ret == -1){
		printf("hdlc_terminate failure¥n");
	}

	printf("test end¥n");

	return 0
```