HTTPd for NORTi

User's Guide

2012 年 6 月版

株式会社ミスポ

目　次

2012 年 6 月版で改訂された項目

ページ	更新内容
4.5	コンフィグレーションの表内のフォントが乱れていたのを統一
7	T_HTTPD_FILE 構造体の未使用メンバの記載漏れを修正
11,12	HTTP 初期化情報で、使用していない sta_semid,sta_mbxid に「未使用」と明記
12	*myprivkey の説明で「秘密鍵が格納バッファ」を「秘密鍵格納バッファ」に

2012 年 5 月版で改訂された項目

ページ	更新内容
全体	目次書式を MS ゴシックに、ヘッダを HTTPd for NORTi User's Guide に修正 見出しフォーマットの修正、用語の統一、冗長な表現の見直し
1	「はじめに」「特長」「制限事項」の記述を修正
2	動作確認ブラウザを修正(各最新版に、Netscape 削除、chrome,safari 追加)
3	「概要」の記述を修正、使用リソース説明の修正
4	「コンフィグレーション」「MAXSESSION または HTTP_MAXSESSION」の説明を修正
6	「File System を使用する場合」の説明を修正、SSL for NORTi の前に「別売の」を追記
7,8	http_ent_file の説明を見直し
9	「3.7 時計(RTC)」の説明を見直し
11	「3.8 HTTPd の初期化と起動」の説明を修正
12	「httpd_ini(HTTPS 使用時)」に SSL 初期化についての記述を追加
13	「3.9 エラーハンドリング」に説明を追記
14～17	BASIC 認証、ダイジェスト認証の説明を見直し。AUTH_BASIC, FILE_AUTH, AUTH_DIGEST 各マクロについての脚注を追加
18	「3.13 PPP を使用する場合」の位置が不適切だったので、4 章直前に移動
20	multipart/form-data 用の CGI 関数の説明を追加
20	T_CGI_ENV に sockID を追加
21	ポインタ型メンバについての説明を追記。T_CGI_ENV にない HTTP ヘッダの参照についての説明を追記
21～22	「クライアントへ送信するデータについて」として、従来どおりの送信方法と CGI 内で分割送信する方法の説明を追記。その他説明文の見直し
22	「CGI 内で大きなローカル変数を使用する場合について」の説明を修正
23,24	「CGI の使用法」の(1)(2)の項目タイトルを修正
25～30	画面キャプチャを最新のものに差し替え。併せて説明の見直し・修正
26	CGI サンプルソースファイル名と CGI 関数名の誤りを修正
29	cookie.cgi の説明を見直し
31	fsys.cgi の説明を追加

2008 年 3 月版で改訂された項目

ページ	更新内容
全体	禁則処理見直し
1	「はじめに」、「特長」の記述を修正
2	動作確認済みブラウザを修正
3	「概要」の記述を修正
3	使用リソース数訂正、説明の修正
4	MAXSESSION の説明修正、使用メモリ計算を欄外へ
4	MAX_SOCK_BUFFER の説明修正
4	TCP_SND_BUF→TCP_SND_BUFSZ 、 TCP_RCV_BUF→TCP_RCV_BUFSZ に修正
5	SS_MPL_BUFSZ のデフォルト値修正、説明修正
5	SSL_SND_BUFSZ 追加
6	DEFAULT_PATH の説明追加、複数 LAN I/F からの接続待ちを行う場合の説明追加
7	http_ent_file 関数の説明修正
10,11	httpd_ini 関数の戻り値の説明追加
19	T_CGI_ENV のメンバ追加
21	CGI 関数パラメータ説明追加
21	CGI 関数注意事項の説明追加

2007 年 3 月版で改訂された項目

ページ	更新内容
16	「3.12 PPP を使用する場合」を追加

2006 年 11 月版で改訂された項目

ページ	更新内容
1	「使用可能な上位プロトコルは IP Version4 のみです。」を削除
5	MAXSESSION6 を追加
5	「3.4 IPv4/IPv6 デュアルスタックを使用する場合」を追加

2006 年 10 月版で改訂された項目

ページ	更新内容
4	HTTP_MAXSESSION と HTTPS_MAXSESSION を追加
5	「3.3 HTTPS を使用する場合」を追加
5	SSL_RCV_BUFSZ を追加
11	httpd_ini (HTTPS 使用時)を追加

第1章 導入

1.1 はじめに

HTTPd for NORTi(以降「HTTPd」と略す)は、TCP/IP のアプリケーションレイヤとして HTTP(HyperText Transfer Protocol)1.0/1.1 ベースの Web サーバ機能を実現します。
本書では HTTPd の説明のみを行っていますので、カーネルや TCP/IP プロトコルスタックの使用方法については、それぞれのユーザーズガイドを参照してください。

1.2 特長

HTTPd の全ソースコードが付属しています。(暗号化ライブラリを除く)
CGI による関数の呼び出しと応答に対応しており、ファイルアップロードや Cookie を CGI で実現したサンプルが付属しています。
Basic 認証、ダイジェスト認証をサポートしています。
ディスクレスで使用できますが、階層ディレクトリ対応のファイルシステムもサンプルとして付属しています。

1.3 制限事項

「NORTi Professional」、「NORTi Professional II」専用のオプションミドルウェアとして設計されており、他の OS との組み合わせは想定していません。

付属するファイルシステムはロングファイル名に対応していません。製品版との違いは「NORTi File System Version 4 ユーザーズガイド」の「2.9 File System Ver.2 との互換性」を参照してください。

1.4 ファイル構成

HTTPd for NORTi は次のファイルから構成されます。

/inc

w3dsys.h	HTTPd ヘッダファイル
md5.h	ダイジェスト認証で使用する MD5 用ヘッダファイル
w3dcfg.h	HTTPd コンフィグレーションヘッダファイル

/src

base64.c	Base64 処理ルーチン
cgi.c	CGI 処理関数
config.c	ファイルタイプ関連外部変数
headers.c	HTTP ヘッダ関連関数
http09.c	HTTP0.9 処理関数
http10.c	HTTP1.0 処理関数
scodes.c	ステータスコードとメッセージの定義
socket.c	ソケット関連関数
w3d.c	HTTP デーモンメイン
w3dfile.c	ファイル処理関連関数
w3dutil.c	ユーティリティ関数
md5.c	ダイジェスト認証で使用する MD5 用関数

/httpsmp

dcode.c	サンプルコンテンツデータ
cgismp.c	CGI サンプル関数

/smp

補足説明書を参照下さい。

1.5 サポートしている機能

・HTTP version 1.0 または 1.1(対応メソッド GET,POST,HEAD)
・CGI による関数の呼び出しと応答
・BASIC 認証、ダイジェスト認証

1.6 動作確認ブラウザ

Internet Explorer 8.0(Windows XP)/9.0(Windows 7)、Opera 11.51、FireFox 7.0.1
Google Chrome 14.0.835.202、Apple Safari 5.1.1

第2章 HTTPdの構成

本章ではHTTPdの構成と動作について、説明しています。

2.1 概要

HTTPdはNORTi Verison 4のタスクとして動作します。
HTTPdは複数のセッションタスクを生成、起動します。各タスクは同時に複数の接続要求を待機しています。(一般的にブラウザソフトは、ダウンロード速度を向上させるために、同時に複数のセッションを起動します。)
各セッションタスクは非同期に動作し、クライアントからの接続受け入れ後は、相手がクローズするかタイムアウトになるまで接続状態が続きます。(Keep Alive)

本HTTPdは、クライアントからのCGI実行要求に対してあらかじめURIに関連付けられた関数を呼び出す機構を有しており、これをCGI機能としています。
※詳細はCGIの使い方を参照下さい。

HTTPd自体は、デバイスに依存しないように設計されていますが、ファイルシステム部と時計を使う部分は、デバイス依存です。この部分は、使用するハードウェアに合わせてカスタマイズが必要になります。

2.2 使用リソース

〈タスク〉

・タスク数	セッション数と同じ(デフォルト = 4)

〈可変長メモリプール〉

・汎用メモリバッファとして利用	1個(サイズは後述します)

〈セマフォ〉

・ファイル更新同期用セマフォ	1個(付属のファイルシステム内で使用)

〈TCP用受付口〉

・セッション受付口	LAN I/Fのチャネル数分確保されます。 HTTPS使用時はさらにチャネル数分

〈TCP用通信端点〉

・各セッション用に	1個×セッション数

第3章 コンフィグレーション

本章ではHTTPdを使用するために必要なコンフィグレーションを説明します。

3.1 コンフィグレーション

HTTPdのデフォルトコンフィグレーションは、必要最低限の構成で設定されています。使用するコンテンツの規模等により、コンフィグレーションの変更が必要になる場合があります。これらのコンフィグレーションは、ユーザープログラムで#icnlude “w3dcfg.h”の前にマクロ定義を行うことで変更できます。マクロの意味は下記表のとおりです。w3dcfg.hは、ユーザープログラムのどれかのソース1か所で、必ずインクルードしてください。

例： #define MAXSESSION 10
#include “w3dcfg.h”

マクロ名	ﾃﾞﾌｫﾙﾄ値	説　　明
MAXFILENO	35	内部で使用するファイル(コンテンツ)数の上限を指定します。
MAXSESSION または HTTP_MAXSESSION	4	HTTPのセッション(タスク)数を指定します。 HTTPSやIPv6を使用しない場合はMAXSESSIONでセッション数を指定します。 HTTPS使用時は、HTTPのセッション数をHTTP_MAXSESSIONで、HTTPSセッション数をHTTPS_MAXSESSIONで指定してください。 MAXSESSIONは､HTTP_MAXSESSIONとHTTPS_MAXSESSIONから自動的に求められます。 1セッションあたり約12.8KByteのメモリを使用します。 (デフォルト値、計算方法は後述) HTMLファイル中に多数のリンクが記述されている場合は、ブラウザは複数のセッションを起動します。 ブラウザ側に全ての画面が表示されないような場合は、この値を増やしてみてください。(スタックの増加に要注意)
HTTPS_MAXSESSION	4	HTTPS のセッション(タスク)数を指定します。1セッションあたり約18Kbyteのメモリを使用します。 (デフォルト値、計算方法は後述) HTTPSマクロを定義したときのみ使用されます。
MAX_SOCK_BUFFER	2048	内部で受信用ワークバッファのサイズを決定します。
TCP_SND_BUFSZ	1460×3	各セッションが使用するTCPの送信バッファサイズです。

TCP_RCV_BUFSZ	1460×3	各セッションが使用する TCP の受信バッファサイズです。
SS_MPL_BUFSZ	1024×3	可変長メモリプールのサイズを決定するための、1 セッション毎の使用量目安です。この値と欄外の計算式で求めた値の和が、１セッションで使用するメモリプールのサイズです。 CGI 関数で、より多くのメモリを w3d_malloc 関数にて獲得したい場合は、このサイズを増加してください。
SSL_RCV_BUFSZ	4096	SSL 内部で使用する受信バッファのサイズです。 HTTPS マクロが有効時のみ使用されます。
SSL_SND_BUFSZ	4096	SSL 内部で使用する送信バッファのサイズです。 HTTPS マクロが有効時のみ使用されます。
REALM	″HTTPd for NORTi″	BASIC 認証やダイジェスト認証使用時、クライアント側に表示される、入力ウインドウのキャプション文字列を定義できます。
MAXSESSION6	3	IPv6/IPv4 デュアルスタック使用時に、IPv6 の HTTP セッション数を指定します。このマクロは、DUAL_STK が有効時のみ使用されます。

セッションあたりの使用メモリサイズの関係

HTTP のセッション：

(MAX_SOCK_BUFFER×2)+TCP_SND_BUFSZ+TCP_RCV_BUFSZ+8+6 ≒ 12.8KByte

HTTPS のセッション：

(MAX_SOCK_BUFFER×2)+TCP_SND_BUFSZ+TCP_RCV_BUFSZ+
SSL_RCV_BUFSZ + SSL_SND_BUFSZ+
sizeof(T_SSL_CON)+sizeof(T_SSL_CEP)) ≒ 約 18KByte

※T_SSL_CON,T_SSL_CEP は SSL ソース内で定義されています。

3.2 File Systemを使用する場合

コンテンツを格納しておくためにFile Systemを使用する場合は､File Systemにあわせて、NOFILE_VER マクロをプロジェクトファイル全体に一括で定義して、HTTPd のソースファイルをコンパイルしてください。

マクロ名	意味・対応するファイルシステムのバージョン
なし	File System を使用しない
NOFILE_VER=2	NORTi File System Ver.2 (HTTPd for NORTi に添付)
NOFILE_VER=4	NORTi File System Ver.4 (別売)

HTML ファイルのドキュメントルート・ディレクトリを“A:¥”以外に設定したい場合は、“DEFAULT_PATH”マクロを指定して w3dfile.c をコンパイルしてください。。
たとえば、DEFAULT_PATH=“B:¥¥WWW_DIR¥¥”と設定してある状態で、HTTPd が index.htm に対する GET リクエストを受信すると、B:¥WWW_DIR¥index.htm ファイルをオープンし、そのデータをクライアントに返します。
ドキュメントルート・ディレクトリは w3d_set_root_path(“B:¥¥WWW_DIR¥¥”)で設定することも可能です。

3.3 HTTPSを使用する場合

HTTPS の通信は、別売の SSL for NORTi を併せて使用することで実現できます。HTTPS の機能を使用するには、“HTTPS”マクロをプロジェクトファイル全体に一括で定義して HTTPd のソースファイルをコンパイルしてください。そして SSL for NORTi のライブラリおよび暗号化ライブラリとリンクしてください。SSL ライブラリの詳細は SSL for NORTi ユーザーズガイドをご覧ください。

3.4 IPv6/IPv4 デュアルスタックを使用する場合

IPv6/IPv4 デュアルスタックを使用する場合は、“DUAL_STK”マクロをプロジェクトファイル全体に一括で定義して HTTPd のソースファイルをコンパイルしてください。

3.5 複数LAN I/Fからの接続待ちを行う場合

複数の LAN I/F からの待ち受けを行う場合、“MULTI_NIF”を定義して w3d.c をコンパイルしてください。この場合、I/F の数だけ TCP 受付口が必要となります。セッション数は I/F の数で分配し、各 I/F 毎のセッション数は 指定セッション数を I/F 数で割ったものになります。セッション数が I/F 数より少ない場合は E_OBJ エラーとなります。

3.6 HTTPdで使用するコンテンツ(HTMLや画像データ)の設定

http_ent_file

[機　能] HTTPd で使用するコンテンツの設定

[形　式] ER http_ent_file(T_HTTPD_FILE *f, int n)

T_HTTPD_FILE f　コンテンツ情報

int　　n　コンテンツ数

```
typedef struct{
    char    name[MAX_FILE_NAME];/* ① ファイル名 */
    void*   content;            /* ② コンテンツデータへのポインタ */
    W       length;             /* ③ ファイルサイズ */
    struct  tm date;            /* 未使用 */
    BOOL    cgi;                /* ④ CGI フラグ */
    int     cgi_fileno;         /* 未使用 */
} T_HTTPD_FILE;
```

①ファイル名

登録コンテンツのファイル名を設定します。ファイル名は必ず「/」(スラッシュ)から始め、NULL で終端してください。ファイルシステム上のデータだけを使用する場合は、本関数でコンテンツを登録する必要はありません。

②コンテンツデータへのポインタ

メモリ上のコンテンツデータ(HTML や画像ファイルイメージ)を登録するときは、CGI フラグを FALSE とし、コンテンツデータのアドレスを設定してください。CGI 関数を登録するときは、CGI フラグを TRUE とし、関数へのポインタをセットしてください。

③ファイルサイズ

メモリ上のコンテンツデータを登録する際は、そのサイズを設定してください。CGI を登録する場合は、本データは使用されません。(Don't care)

④CGI フラグ

登録するコンテンツが CGI なら TRUE を、そうでない場合は FALSE を設定してください。

［戻り値］

E_OK:	正常終了
E_OBJ:	コンテンツ数が MAXFILENO を超えました
E_PAR:	パラメータエラー

［解　説］

メモリ上のコンテンツや CGI 関数を登録するために使用します。

File System を使用している場合に、本関数を使用するとメモリ上のコンテンツと、ファイルシステム上のコンテンツを併用することができます。File System を使用しない場合は、本関数を使用してコンテンツのデータをメモリ上に設定する必要があります。CGI は必ず本関数を使用して登録してください。

［補　足］　この関数は httpd_ini 呼出し後に使用してください。

3.7 時計(RTC)

クライアントが、以前にアクセスしたコンテンツを再度アクセスした時、コンテンツが更新されていなければ HTTPd からのレスポンスを省略したほうがネットワークのトラフィックを軽減するのに有効な場合があります。
クライアントからのリクエストに“If-Modified-Since”が付加されていれば、リクエストに含まれる日時とコンテンツのタイムスタンプを比較し、更新がなければ 「304 Not Modified」のメッセージを返します。この様にして、クライアント（ブラウザ）に自分でキャッシュしているデータを使用させることができます。
この機能はデフォルトで無効に設定されていますが、**CHK_304** マクロを定義して http10.c をコンパイルすることにより有効化できます。
ただし、CHK_304 マクロを定義しただけでは機能が有効になるだけです。コンテンツデータ(ファイル)にタイムスタンプが記録されるようにしておく必要があります。そのために、時計のハードウェアと、時刻を取得する処理(ソフトウェア)を実装してください。デフォルトではいつも同じ日付を返すダミー関数が組み込まれています。

時計部は搭載するハードウェアに依存しますので、次ページに説明する関数 settime(),gettime()をハードウェアにあわせて実装して下さい。
下記の tm 構造体 は HTTPd 内部で時間を管理するのに利用しています。

```
struct tm {
        int tm_sec;             /* seconds after the minute - [0,59] */
        int tm_min;             /* minutes after the hour - [0,59] */
        int tm_hour;            /* hours since midnight - [0,23] */
        int tm_mday;            /* day of the month - [1,31] */
        int tm_mon;             /* months since January - [0,11] */
        int tm_year;            /* years since 1900 */
        int tm_wday;            /* days since Sunday - [0,6] */
        int tm_yday;            /* days since January 1 - [0,365] */
        int tm_isdst;           /* daylight savings time flag */
};
```

settime

[機　能] 時計の設定

[形　式] BOOL settime(struct tm *time)
　　struct tm *time　設定する時間

[戻り値] TRUE:正常終了, FALSE:エラー終了

gettime

[機　能] 時計の読み出し

[形　式] BOOL gettime(struct tm *time)
　　struct tm *time　取得した時間

[戻り値] TRUE:正常終了, FALSE:エラー終了

3.8 HTTPdの初期化と起動

HTTPd の初期化と起動を行うには以下の関数を呼び出してください。
tcp_ini によるプロトコルスタックの初期化の後に行ってください。

httpd_ini

［機　能］ HTTPd の初期化と起動を行う

［形　式］ ER httpd_ini(T_HTTP_INI* ini)

T_HTTP_INI* ini HTTP 初期化情報

```
typedef struct {
    ID  sta_tskid;      /* タスク開始 ID */
    ID  sta_semid;      /* 未使用 */
    ID  sta_mbxid;      /* 未使用 */
    ID  sta_mplid;      /* 可変長メモリプール開始 ID */
    ID  sta_cepid;      /* 通信端点開始 ID */
    ID  sta_repid;      /* 受付口開始 ID */
} T_HTTP_INI;
```

全ての値を 0 に設定した場合 ID は自動的に設定されます。

［戻り値］ E_OK:正常終了, E_OK 以外:エラー終了
エラー終了の場合、代表的なエラーは下記のようになります。
E_OBJ: セッション用メモリ獲得失敗
　　セッション数不足
　　受付口生成エラー、端点生成エラー
　　セッションタスク生成エラー、起動エラー 等
　　※主にコンフィグレーションによるエラー

httpd_ini (HTTPS 使用時)

[機　能]　HTTPd の初期化と起動を行う

[形　式]　ER httpd_ini(T_HTTP_INI* ini, UB *mycert, UB *myprivkey, UB *keypasswd)

T_HTTP_INI* ini HTTP 初期化情報

```
typedef struct {
    ID  sta_tskid;      /* タスク開始 ID */
    ID  sta_semid;      /* 未使用 */
    ID  sta_mbxid;      /* 未使用 */
    ID  sta_mplid;      /* 可変長メモリプール開始 ID */
    ID  sta_cepid;      /* 通信端点開始 ID */
    ID  sta_repid;      /* 受付口開始 ID */
} T_HTTP_INI;
```

全ての値を 0 に設定した場合 ID は自動的に設定されます。

UB　*mycert　　共通鍵証明書格納バッファへのポインタ

UB　*myprivkey　秘密鍵格納バッファへのポインタ

UB　*keypasswd　秘密鍵のパスワード格納バッファへのポインタ
(秘密鍵が暗号化されている場合に使用します)

[戻り値]　E_OK:正常終了, E_OK 以外:エラー終了

エラー終了の場合、代表的なエラーは下記のようになります。

E_OBJ: セッション用メモリ獲得失敗
セッション数不足
受付口生成エラー、端点生成エラー
セッションタスク生成エラー、起動エラー 等
※主にコンフィグレーションによるエラー

[解　説]　SSL の初期化は、本関数の中で行っています。

3.9 エラーハンドリング

HTTPd のソースのエラー検出箇所には、TroubleShooter マクロを埋め込んであります。
このマクロは w3dutil.c の trouble_shooter 関数を呼び出します。

```
void trouble_shooter(ER er, char* fn, int line)
    ER      er      エラーコード
    char*   fn      ソースファイル名
    int     line    行番号
```

通常は空の関数ですが、DEBUG マクロを定義して w3dutil.c をコンパイルすると、httpd_error という関数を呼び出すようになっていますので、ユーザープログラム内に実装してください。引数は trouble_shooter と同じです。ソースファイル名と行番号より、エラーの確認が容易になります。

3.10 Basic認証機能を使用する場合

許可されたユーザ以外によるコンテンツへのアクセスを制限するためにBASIC認証の機能を使用することができます。BASIC認証の機能を使用する場合は、AUTH_BASICマクロを定義してHTTPdのソースファイルをコンパイルしてください。(*1)
以下に示すパスワードチェック関数はユーザープログラム側で用意してください。

---

httpd_passwd_check

[機　能] HTTPd の BASIC 認証チェックを行う

[形　式] int httpd_passwd_check(const char *name, const char *passwd)
　　const B *name　　ユーザ名
　　const B *passwd　パスワード

[解　説] HTTPd のコンテンツ情報へのアクセス認証を行います。
　　この関数はユーザープログラムで用意してください。
　　クライアント側からコンテンツに最初のアクセスがあったときに HTTPd から
　　コールされます。

[戻り値] ACCESS_OK:アクセス許可,
　　ACCESS_DENIED:アクセス拒否　　クライアントに再入力を促します
　　ACCESS_FAILED:アクセス拒否　　クライアントに 401 Unauthorized を返します

[例]

```
/*****************************************************************************
* BASIC 認証チェック
*****************************************************************************/

int httpd_passwd_check(const char *name, const char *passwd)
{
    if (strcmp(name, "mispo") == 0
     && strcmp(passwd, "norti") == 0)
        return ACCESS_OK;
    else
        return ACCESS_FAILED;
}
```

(*1) AUTH_BASIC マクロの影響を受けるソースファイルは、base64.c, http09.c, http10.c, w3dutil.c です。これらのファイルのコンパイルオプションでマクロ定義を行ってください。プロジェクト全体に一括設定しても構いません。

3.11 個々のコンテンツ毎にBASIC認証機能を使用する場合

前頁のBASIC認証では全てのコンテンツ情報を一組のユーザ名とパスワードによってアクセスの可否が管理されます。
AUTH_BASIC とFILE_AUTHマクロ (*2)を定義してHTTPdのソースファイルをコンパイルすると個々のコンテンツ（ファイル）毎にユーザ名とパスワードの管理を行うことができます。
以下に示すパスワードチェック関数はユーザープログラム側で用意してください。

---

httpd_file_passwd_check

[機　能]　コンテンツ毎に BASIC 認証チェックを行う

[形　式]　int httpd_file_passwd_check(const char *fname,
const char *name, const char *passwd)

const char *fname　ファイル名（コンテンツ名）
const char *name　ユーザ名
const char *passwd　パスワード

[解　説]　個々のコンテンツ情報へのアクセス認証を行います。
この関数はユーザープログラムで用意してください。
クライアント側からコンテンツに最初のアクセスがあったときに HTTPd からコールされます。

[戻り値]　ACCESS_OK:　アクセス許可,
ACCESS_DENIED:　アクセス拒否　クライアントに再入力を促します
ACCESS_FAILED:　アクセス拒否　クライアントに 401 Unauthorized を返します

(*2) FILE_AUTH マクロの影響を受けるソースファイルは、w3dutil.c です。コンパイルオプションでマクロ定義を行ってください。プロジェクト全体に一括設定しても構いません。

[例]

```
/*****************************************************************************
* File(コンテンツ)毎の BASIC 認証チェック
*****************************************************************************/

int httpd_file_passwd_check(const char *fname, const char *user, const char *pass)
{
    if(!stricmp(fname, "/sample.htm")){
        if((!stricmp(user,"mispo1")) && (!stricmp(pass,"12345")))
            return ACCESS_OK;
        else
            return ACCESS_DENIED;
    }
    else if(!stricmp(fname, "/sample.cgi")){
        if((!stricmp(user,"mispo2")) && (!stricmp(pass,"12345")))
            return ACCESS_OK;
        else
            return ACCESS_DENIED;
    }
    else if(!stricmp(fname, "/cookie.cgi")){
        if((!stricmp(user,"mispo3")) && (!stricmp(pass,"12345")))
            return ACCESS_OK;
        else
            return ACCESS_DENIED;
    }
    else
        return ACCESS_OK;
}
```

3.12 ダイジェスト認証

BASIC 認証では、ユーザ名とパスワードをコロン“:”で結合し、BASE64 でエンコードして送信するので、盗聴や改竄されやすいという問題があります。ダイジェスト認証では、クライアントから入力されたユーザ名とパスワードの漏洩を防ぐために、MD5 暗号化方式で暗号化してサーバに送ります。

HTTPd for NORTi ではダイジェスト認証も使用可能で、個々のコンテンツ毎にユーザ名とパスワードでアクセスを制限することができます。

ダイジェスト認証を使用する場合は**AUTH_DIGEST**マクロ(*3)を定義してHTTPdのソースファイルをコンパイルしてください。

パスワード取得関数はユーザプログラム側で用意して下さい。

(*3) AUTH_DIGEST マクロの影響を受けるソースファイルは、http09.c, http10.c, w3dutil.c です。これらのファイルのコンパイルオプションでマクロ定義を行ってください。プロジェクト全体に一括設定しても構いません。

httpd_get_passwd

[機　能]　ダイジェスト認証用 ユーザー名とパスワードの取得

[形　式]　int httpd_get_passwd(const char *fname, char *username, char *passwd)

const char *fname　　ファイル名
char　　*username ユーザ名を設定するバッファのポインタ
char　　*passwd　パスワードを設定するバッファのポインタ

[解　説]　HTTPd 内部から呼び出されます。ダイジェスト認証を使用する場合はこの関数をユーザプログラムで用意してください。
引数 fname に認証の対象となるファイル名が渡された場合は、ユーザ名とパスワードを引数のポインタ(username,passwd)が指す領域へコピーして、戻り値をTRUE(認証が必要)としてリターンしてください。
認証対象外のファイル名が与えられたときは、FALSE でリターンしてください。
※ユーザ名とパスワードの最大文字数はそれぞれ終端文字を含めて 50byte です。
50byte を越えた文字列を設定しないでください。

[戻り値]　TRUE:　ファイルにはユーザ名とパスワードの定義がある(認証必要)
FALSE: ファイルにはユーザ名とパスワードは不要(認証不要)

[例]

```
/*****************************************************************************
* ダイジェスト認証用 ユーザ名とパスワード取得
*****************************************************************************/

int httpd_get_passwd(const char *fname, char *username, char *passwd)
{
    /* If the file needs authentication copy username, passwd and return TRUE */
    if(!stricmp(fname, "/sample.htm")){
        strcpy(username,"mispo");
        strcpy(passwd,"mispo");
    }
    else if(!stricmp(fname, "/sample.cgi")){
        strcpy(username,"MiSPO");
        strcpy(passwd,"12345");
    }
    else
        /* Given file does not require Authentication */
        return FALSE;
    return TRUE;
```

```
}
```

3.13 PPPを使用する場合

PPP for NORTi にはMD5関数が含まれています。PPP for NORTi とHTTPd for NORTi を同時に使用する場合は、ユーザプログラムにHTTPd/SRC/md5.c をリンクしないでください。

第4章 CGIの使い方

本 HTTPd は、クライアントからの CGI 実行要求に対してあらかじめ URI に関連付けられた関数を呼び出す機構を有しています。

4.1 CGIについて

CGI(Common Gateway Interface)は、サーバ上で動作するプログラムとのインターフェースで、WWW クライアントとの間で対話的な処理を行わせる場合や、ページを表示する際にあらかじめなんらかの処理を行いたい場合などに使用します。HTTPd ではこの機能を関数によって実現します。

4.2 CGIの流れ

① クライアントは HTTPd に接続し、要求(リクエスト)をサーバに送る。
② サーバは環境変数を読み込み、クライアントから受け取ったデータを渡す。
③ HTTPd は、任意のパラメータ文字列を設定し CGI 関数を呼び出す。
④ CGI 関数は、実行後、HTTPd に結果(HTTP ヘッダ+HTML)を返す。
⑤ HTTPd はクライアントに応答を送る。

4.3 CGI関数

CGI 関数の例を以下に記します。関数名は任意です。

```
int cgi_app_sample(T_CGI_ENV *envp, void *post, void **outp, int *outlen)
    T_CGI_ENV       *envp    CGI 環境変数の格納テーブル
    void            *post    POST メソッド要求時のパラメータ
    void            **outp   CGI 処理後にクライアントへ返すデータ
    Int             *outlen  outp の長さ
```

enctype 属性が“multipart/form-data”で使用する CGI の場合は以下のように第 2 引数が変わります。関数名は任意です。(ファイルをサーバへ送信するような用途)

```
int upload_file(T_CGI_ENV *envp, ID sockID, void **outp, int *outlen)
    ID              sockID   HTTPd Socket ID

    typedef struct {
        ID          sockID                  /* HTTPd Socket ID */
        char        szServerSoftware[20];   /* サーバソフトウェア */
        char        *szServerName;          /* サーバ名 */
        char        szGatewayInterface[20]; /* GATEWAY インターフェース */
        char        szServerProtocol[20];   /* サーバプロトコル */
        UH          nServerPort;            /* サーバポート番号 */
        char        *szScriptName;          /* スクリプト名 */
        UH          nRequestMethod;         /* リクエストメソッド */
        char        *szQueryString;         /* Query 文字列 */
        char        szRemoteHost[20];       /* リモートホスト名 */
        UW          dwRemoteAddr;           /* リモートホストアドレス */
        char        *szAuthType;            /* 認証タイプ */
        char        *szRemoteUser;          /* リモートユーザ名 */
        char        *szContentType;         /* コンテントタイプ */
        UW          nContentLength;         /* コンテント長 */
        char        *szCookie;              /* Cookie 文字列 */
        Headers     *hInfo;                 /* ヘッダ情報へのポインタ */
    } T_CGI_ENV;
```

上記で重要なのは **nRequestMethod** と **szQueryString** です。nRequestMethod にはクライアントからの POST および GET が設定されます。POST の場合、パラメータは第 2 引数の post に設定されます。GET の場合、パラメータは szQueryString に設定されます。構造体のうち、ポインタ型メンバは、リクエスト中に存在しない場合は NULL ポインタとなっています。必ず NULL チェックを行ってから実体を参照してください。

この構造体に保存されていないヘッダ情報は、Headers *hinfo 経由でアクセスすることにより参照が可能です。Headers 型に保存しているヘッダは以下の通りです。

ヘッダ名	格納先 Headers 構造体メンバ
Accept	char *szAccept
Accept-Charset	char *szAcceptCharset
Accept-Encoding	char *szAcceptEncoding
Accept-Language	char *szAcceptLanguage
Authorization	char *szAuth
Connection	BOOL bPersistent TRUE = KeepAlive FALSE = Close
Content-Length	char *szContentLength
	UW ulContentLength
Content-Type	char *szContentType
Cookie	char *szCookie
Date	char *szDate
From	char *szFrom
Host	char *szHost
If-Modified-Since	char *szIfModSince
	time_t ttIfModSince
Range	char *szRange
Referer	char *szReferer
Transfer-Encoding	char *szTransferEncoding
	BOOL bChunked TRUE = Chunked FALSE = それ以外
Upgrade	char *szUpgrade
User-Agent	char *szUserAgent

※ヘッダの詳細については本書では省略いたします。HTTP の仕様書をご参照願います。

クライアントへ送信するデータについて

以下の 2 つの方法から選択してデータを送信してください。

① CGI からリターンしてから送信

クライアントに送信するレスポンスデータは、CGI の内部で w3d_malloc 関数を使用して動的に領域を獲得し、そこへメッセージをセットしてください。
outp にそのアドレスと、outlen に送信長さをセットしてリターンしてください。この領域は HTTPd がクライアントへ送信後、返却します。

② CGI 内部で送信

長大なデータを送信する CGI では、レスポンス領域のすべてを w3d_malloc 関数で取得するのは、RAM 容量の問題などから困難な場合があります。
その場合は、socket.c の SoSend 関数を使用して、少しずつ送信することが可能です。
(cgismp.c の cgi_snd_dat 関数でも同じです)

```
ER SoSend(ID sockID, char* szBuf, int iLen)
ER cgi_snd_dat( ID sockID, char *snd_buf, int len )
```

CGI に渡される T_CGI_ENV 構造体のメンバである sockID を取得し、上記関数の引数 sockID として渡してください。
全ての送信文字列を上記の関数で送信し、CGI 終了後 HTTPd により送信すべき文字列がない場合は、必ず outp を NULL に、outlen を 0 にしてリターンしてください。この場合は CGI 内で w3d_malloc により獲得した領域は全て返却しておいてください。

CGI内で大きなローカル変数を使用する場合について

CGI 関数内で大きなローカル変数を使用すると、スタックオーバーフローの原因になることがあります。上記の①②以外に大きな領域が必要ならば、その領域も w3d_malloc で動的に獲得してください。
この用途に獲得した領域は、使用終了後に CGI 内部で必ず w3d_free()にて返却しておいてください。
これを忘れると、一見動作しているように見えても、CGI 呼び出しを繰り返しているうちに、やがてメモリープールが枯渇して動作異常となります。

4.4 CGIの使用方法

1) CGI 関数の登録例

```
    T_HTTPD_FILE c_file[3];

    /* CGI の登録 cgi_app_sample1 関数*/
    strcpy(c_file[0].name, "/sample.cgi");
    c_file[0].content = cgi_app_sample1;
    c_file[0].cgi   = TRUE;

    /* CGI の登録 cgi_app_sample2 関数*/
    strcpy(c_file[1].name, "/test.cgi");
    c_file[1].content = cgi_app_sample2;
    c_file[1].cgi   = TRUE;

    ercd = http_ent_file(c_file, 2);

/* CGI 関数 */
int cgi_app_sample1(void *envp, void *post, void **outp, int *outlen)
{
      ......
}
```

クライアントから http://Server/sample.cgi?param=xxxxx
param=xxxxx がパラメータとして、szQueryString で cgi_sample1 に渡されます。

2) CGI を呼び出すメモリ上のコンテンツを登録する例

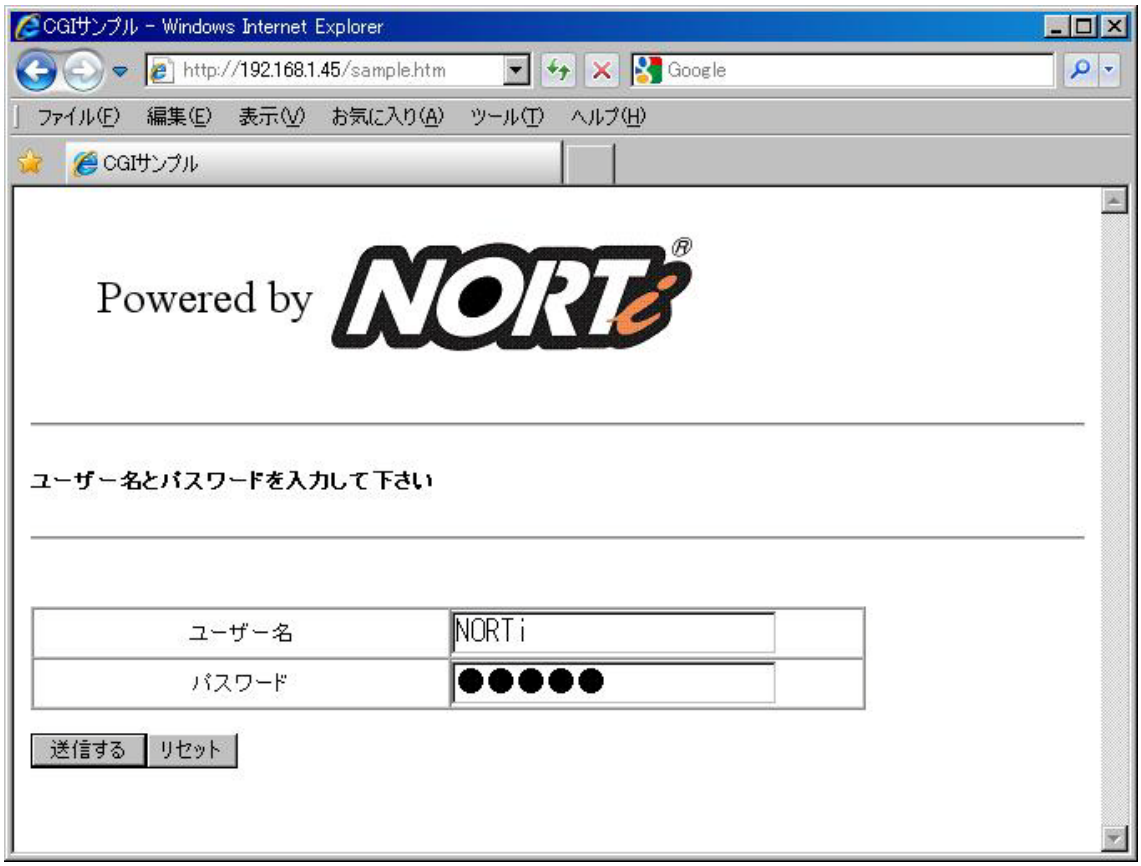
```
/* HTML FORM */
-------------------------
/*コンテンツデータの定義*/
const char settime_htm[]="<html><head><title>時刻設定画面</title></head>¥
    <body><P>時間を設定して下さい。</P>¥
    Form:       YYYY-MM-DD-hh-mm-ss-Wek [Sun,Mon,Tue,Wed,Thu,Fri,Sat] ¥
    <FORM method=¥"POST¥" action=¥"settime.cgi¥">    設定時間：¥
    <INPUT type=¥"text¥" name=¥"date¥" size=¥"37¥"><BR><p>¥
    <INPUT   TYPE=¥"submit¥"VALUE=¥"  送          信  ¥"></FORM></body></html>";
-------------------------
    T_HTTPD_FILE c_file[2];

    strcpy(c_file[0].name, "/settime.cgi");
    c_file[0].content = set_time; /* 関数名 */
    c_file[0].cgi   = TRUE;

    strcpy(c_file[1].name, "/settime.htm");
    c_file[1].content  = settime_htm; /* コンテンツデータ */
    c_file[1].length   = strlen(settime_htm);
    c_file[1].cgi    = FALSE;

/* CGI 関数 */
int set_time(void* envp, void* post, void** outp, int* outlen)
{
      ......
}
```

上記例は時間をサーバへ設定する CGI のサンプル例です。FORM を使用して
時間を設定した後、送信ボタンを押すと settime.cgi が呼び出されます。

第5章 サンプルについて

HTTPd for NORTi のサンプルプログラムでは、いくつかの CGI サンプルとコンテンツデータがデフォルトで組み込まれています。本章ではこれらの使い方について説明します。CGI のサンプルは cgismp.c に、HTML のデータは dcode.c にそれぞれ定義されています。

5.1 sample.htmとsample.cgi

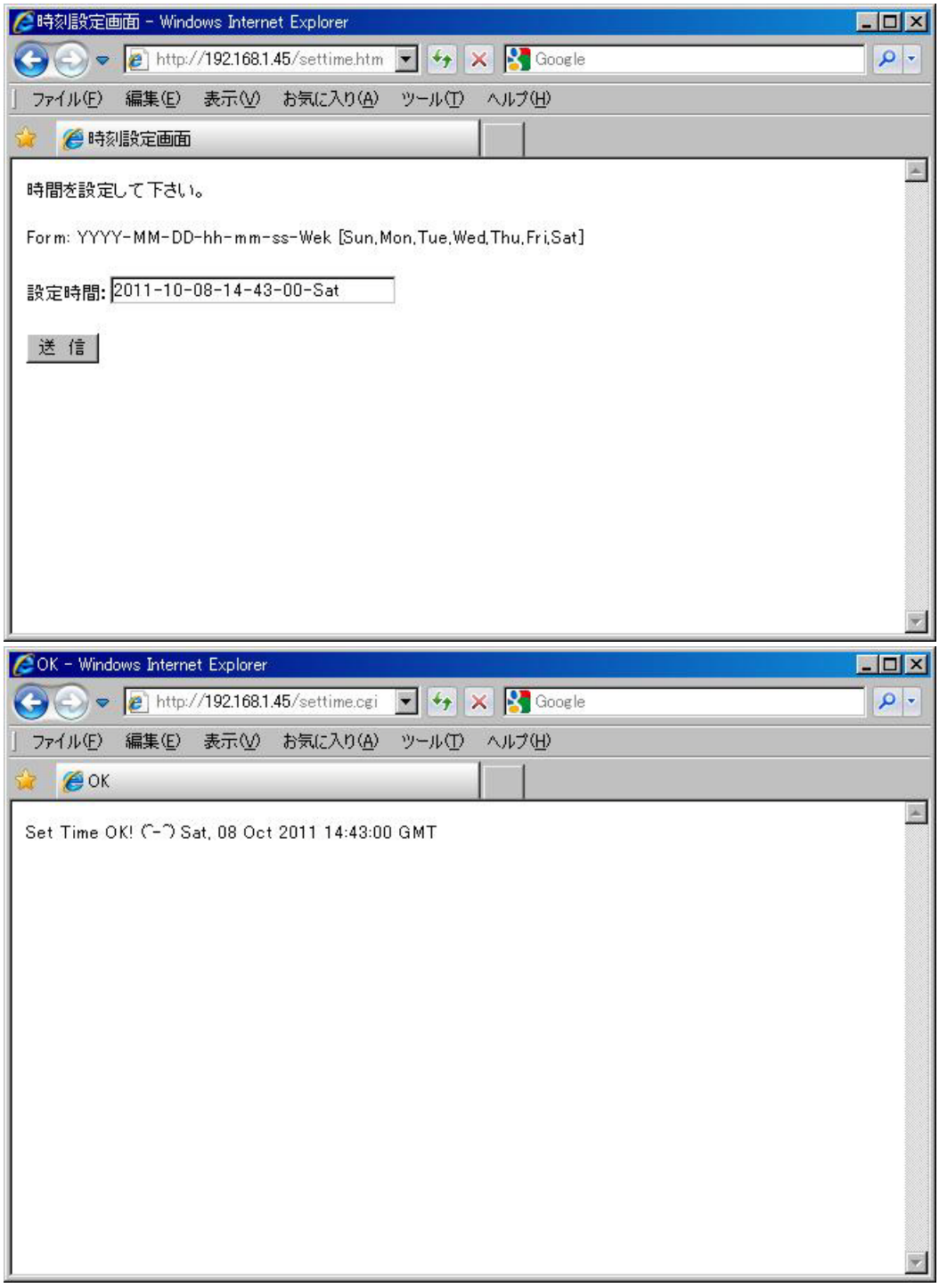


sample.htm を実行すると、上記のような画面がブラウザに表示されます。
ユーザー名、パスワードを設定し送信ボタンを押下します。
HTML はボタンを押下すると **sample.cgi** が実行されるような構文になっています。

sample.cgi が呼び出されると cgismp.c にある cgi_app_sample1 関数が呼び出されます。cgi_sample1 は受け取ったパラメータを元に以下のように表示される HTML データを生成します。

5.2 settime.htmとsettime.cgi

このサンプルも sample.htm と同様に、フォームで設定したパラメータを **set_time.cgi** に渡します。**set_time** 関数はパラメータを元に結果を返します。



直接 「settime.cgi?date=2000-10-15-20-12-22-1」 で CGI を呼び出すことも可能です。

5.3 test.cgi

パラメータなしで呼び出す CGI の例です。関数 cgi_app_sample2 が呼び出されます。

5.4 cookie.cgi

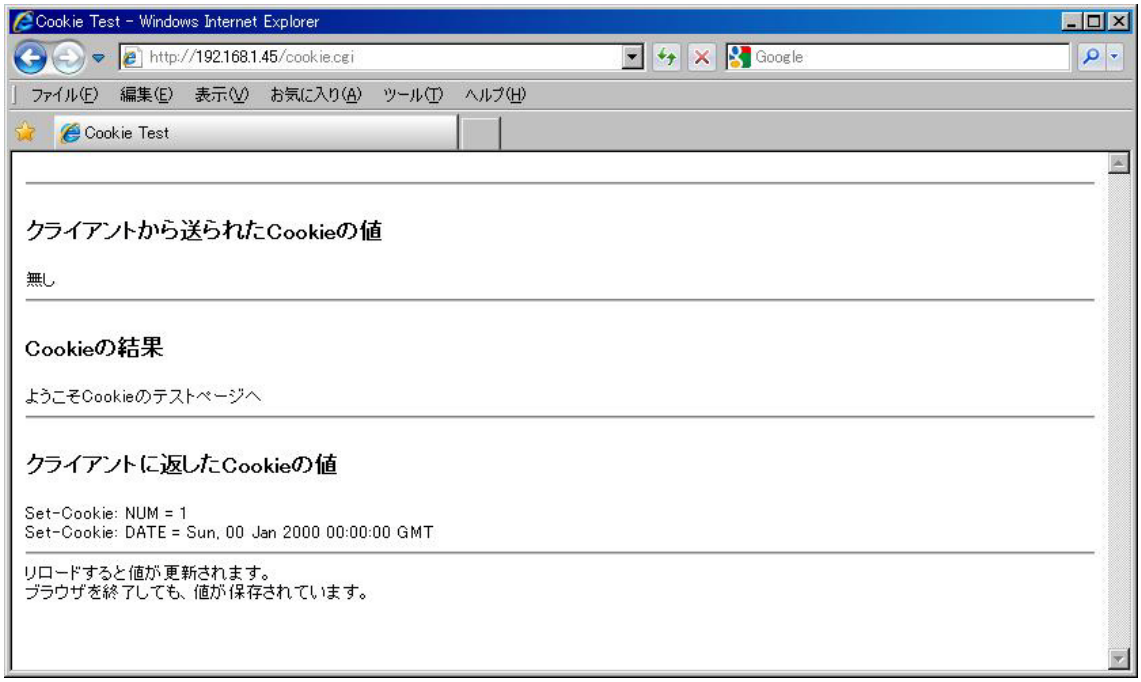
Cookie とはユーザー情報やアクセス履歴などの情報をブラウザと Web サーバ間でやりとりするための仕組みです。本 CGI は、Cookie を表示、更新するサンプルです。関数 **cookie** を使用します。

CGI 内部で HTTP 拡張ヘッダに Set-Cookie：" Cookie 項目名=設定値"と記述し、続いてドメイン名、パス、有効期限などの内容を書いてブラウザに送信します。これらを受け取ったブラウザは、この情報を保存しておき、次回再び同じコンテンツにアクセスする場合に保存されたデータを HTTP 拡張ヘッダに付加します。

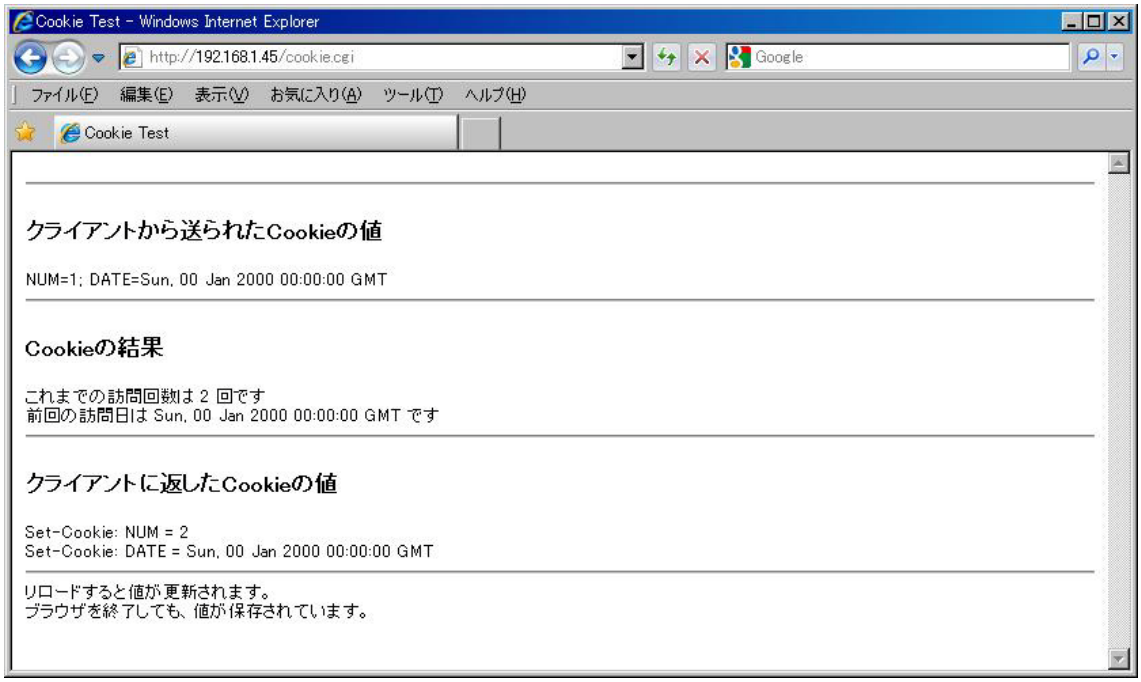
Cookie の値は envp->szCookie に設定されていて、この値を"クライアントから送られた Cookie の値"に表示します。NUM を訪問回数、DATE を前回の訪問日として"Cookie の結果"に表示します。Cookie の値は CGI が更新し、クライアントに更新済みの値を送信するとともに、"クライアントに返した Cookie の値"として表示します。

Cookie の詳細は HTTP プロトコルの専門書等をご覧ください。

なお、HTTPd が Cookie の処理を行っているのではなく、あくまでも CGI が解釈し、更新しているものであることにご注意ください。



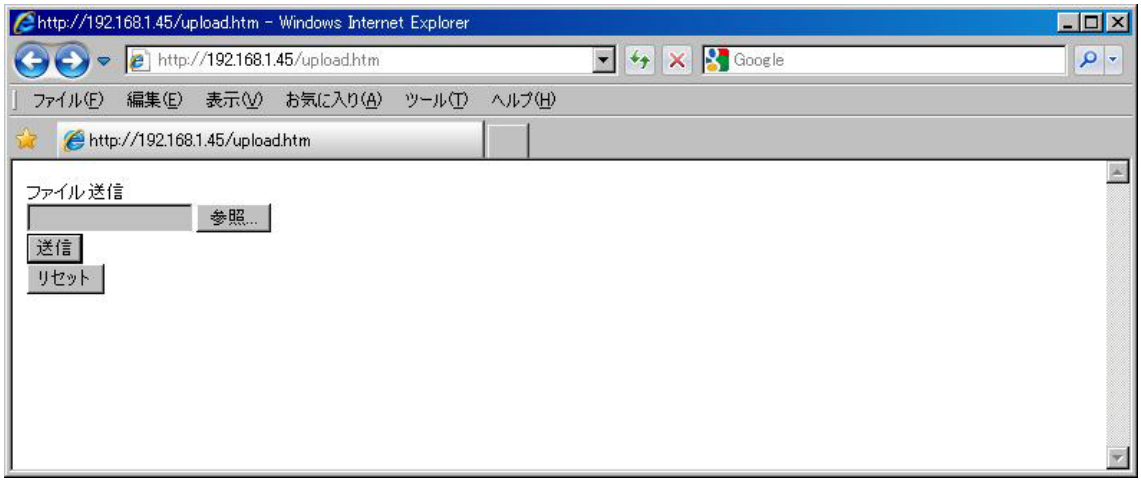
はじめてCGIにアクセスした場合の表示



2 回目以降にCGIにアクセスした場合の表示

5.5 upload.htmとupload.cgi

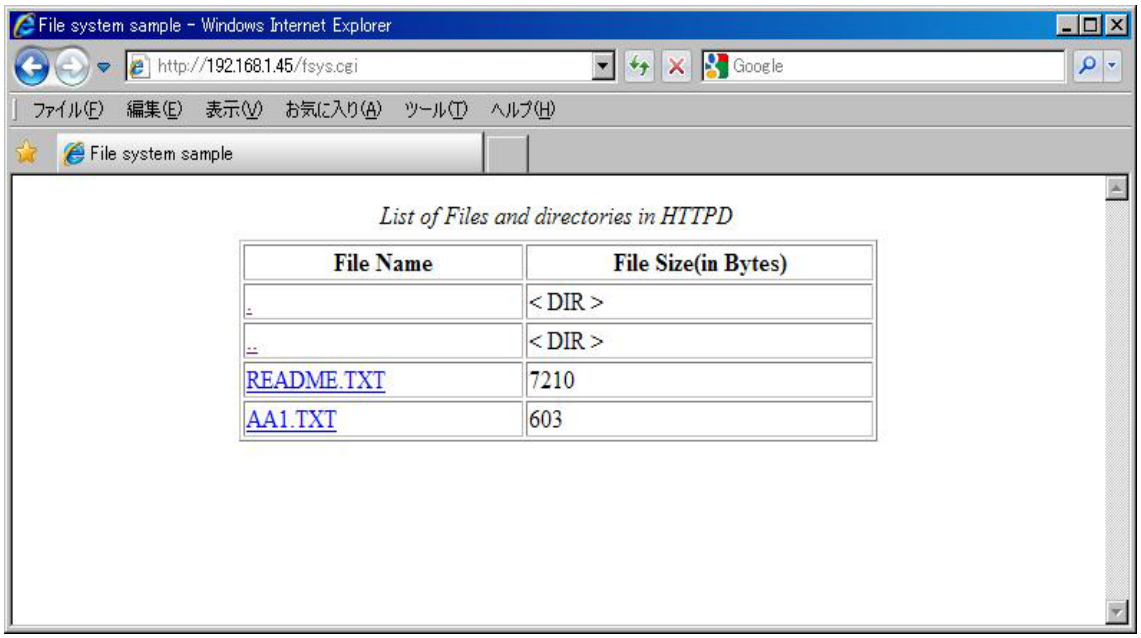
ファイルをアップロードするための CGI サンプルです。



この CGI サンプルは関数 upload_file を呼び出します。送信ボタンを押すと、ファイルの情報が転送されディスクに書き込まれます。(このサンプルを使用する場合は、ターゲット側に File System が組み込まれている必要があります)

5.6 fsys.cgi

ファイルの一覧表示や、ダウンロードを行える CGI のサンプルです。
関数 **file_sys_cgi** を使用します。パラメータなしで本 CGI を呼び出すとルートディレクトリの一覧が表示されます。ファイル名をクリックすると、サーバへは?fname=ファイル名を付加したリクエストが送信されます。この時 CGI はブラウザへ指定ファイルを送信します。ディレクトリ名をクリックすると、サーバへは?dname=ディレクトリ名を付加してリクエストが送信されます。CGI は指定ディレクトリの一覧を送信します。



HTTPd for NORTi ユーザーズガイド

株式会社ミスポ http://www.mispo.co.jp
〒213-0002 神奈川県川崎市高津区二子 5-1-1 高津パークプラザ 3F
一般的なお問い合せ sales@mispo.co.jp
技術サポートご依頼 norti@mispo.co.jp