# オンライン型CBT（Computer Based Testing）システムの作成とWWWの機能的限界について

井関文一*　石井政弘**　松本朋宏***

## 1．はじめに

現在、WWWを利用した教育システムが盛んに作られている。それらのシステムの１つであるオンライン（ネットワーク）型のCBT（Computer Based Testing）システムは、ネットワーク上でコンピュータを使用して試験や自習を行うものである。このシステムを使用することによって、教師側はテスト用紙の節約、自動採点による時間の節約や採点ミスの撲滅、成績の管理・分析などが容易にできるなどの利点を得ることができる。また学生にとっても、自分のテストの得点（成績）やどこを間違えたのかを知ることができ、自習においても自分のレベルにあった学習ができるなどの利点がある。WWWによるCBTシステムは、基盤システム（WWW）が既に幅広く使用されている技術であり、導入や構築のし易さから急速に教育現場で使用されつつある。

しかしながら、WWWを使用したシステムでは、HTTPプロトコル自体の制限により、テスト中にクライアントとサーバ間でセッションを維持することが困難である。HTTPの上位にセッション管理のプロトコルを導入することも可能だが、どうしてもシステムが複雑になるといった欠点があり、セキュリティホールも生じ易くなる。これらの点を考慮すると、WWWがこのようなシステムの基盤として真に有効かどうかは漠然としたところがあるように思われる。

今回我々はWWWを使用したシステムの有効性とその機能的限界を調べるために、WWWサーバをシミュレートするCBTサーバを試作し、実際に学生に対して試験を行った。これによりWWWをネットワーク上のCBTのようなシステムに利用する際の機能的な長所と短所が明らかになった。またこの結果を元にWWWを使用しない独自プロトコルのオンライン型のCBTシステムの開発を行った。

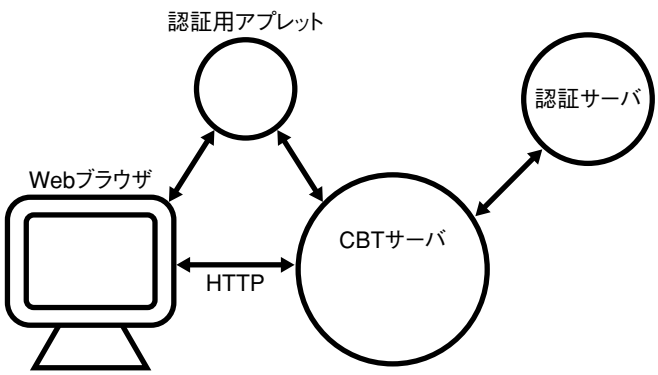


**図1．WWWシミュレーションによるCBTシステムの概要**

*東京情報大学総合情報学部情報システム学科
**東京情報大学総合情報学部情報文化学科
***国際システム(株) eソリューション・ビジネスユニット部

2003年6月11日受理

## 2．WWWシミュレーションによるCBTシステム

### 2-1.システムの概要

通常WWWでは、サーバはWebブラウザにHTMLを送り、Webブラウザがそ のHTMLを解釈し複雑な表示を行う。今回我々はWWWシステム全体を制御・監視するために、WWWサーバの基本機能をシミュレートするCBTサーバを試作した（図1)。WWWサーバをシミュレートするサーバプログラムを作成することにより、Webブラウザをある程度サーバ側からコントロールすることが可能となる。

このシステムのユーザ認証では、通常のBASIC 認証とJavaアプレットを利用した暗号化によるユーザ認証をサポートしている（図2)。WWWで広く使われているBASIC認証はパスワードを暗号化しないので非常に危険である。このシステムでは通常はJavaアプレットを使用してパスワードを暗号化し、Javaアプレットが作動しない環境でのみBASIC認証を使用するようにしている。

ユーザ認証自体はCBTサーバ上で行うのではなく、別に認証サーバを用意しそちらで行う。これはユーザ情報の存在しないサーバ上でもCBTサーバを実行できるようにするためのものである。これにより、認証サーバ上にアカウントがあるユーザであれば、CBTサーバ上にアカウントがなくともシステムを利用することができる。通常認証サーバはその組織のユーザを管理しているマシン上で起動させることになる。認証サーバとのやり取りには簡単なチャレンジ＆レスポンス方式を用いた（独自プロトコル。付録参照)。

CBTサーバはクライアントに対してできる限り接続（セッション）を維持しようと試みる。WWWシステムにおいて接続（セッション）の維持がどれだけ保証されるかを確認するためである。テストには制限時間が設けられており、制限時間を超えてブラウザの操作を行うと強制的に試験は終了する（正常終了)。

また、何らかの理由により、途中で接続（セッション）が切断された場合は試験は異常終了となる。主に試験中に他のページを参照した場合などがこれに相当する（必ずしも常に接続が切断されるわけではない)。これはシステム（HTTP）の問題点ではあるが、今回は逆に学生が試験中に他のページを見ないようにするための制約にもなった。試験が正常終了した場合、学生はその場で自分の合否を知ることができる。図3に実際の問題提示時の画面を示す。

**図2. システムへのログイン画面**

**図3. 問題画面**

WebブラウザとしてWindowsではIE5、IE6、Netscape4.7、Netscape6.0を、LinuxではNetscape4.7、Netscape6.0、Mozilla0.9を使用した。

### 2-2.接続的維持

本来、WWWの転送プロトコルであるHTTPはWebブラウザからのリクエストとそれに対するサーバからのレスポンスで一回の接続（セッション）が終了する。しかし、それではオンライン試験のようなシステムには都合が悪いのは明らかである。

HTTP/1.1からはこのような点を考慮して、接続を維持する動作（接続的維持）がデフォルトの動作となっている[1]が、これは接続の維持を保証するものではなく、どちらかと言えばベストエフォートの動作だと言えることを確認した。

例えば、前述したようにあるページを参照中に他のページを参照したりした場合、タイミングによってはWebブラウザの方から接続が切断されてしまう場合がある。またWebブラウザの中にはサーバに対して一定の時間リクエストを出さない（例えば60秒間ユーザがブラウザを操作しない）とサーバが接続の維持を続けているにも関わらず、ブラウザの方から接続を切断してしまうものもあった。このような動作はWWWの本来の使われ方、すなわちページ単位で情報を得るという使われ方を考えればもっともな動作だとも言える。ただし、オンライン型CBTのようなシステムにおいて、ユーザがWebブラウザの前で何分間か思考している間に勝手にサーバとの接続を断たれてしまうのでは非常に都合が悪い。

今回のシステムでは接続を維持するために、問題文中では画像などの表示は行わず（画像の表示などを行うと、ブラウザによっては新しい接続を作り出す場合があった）、文字のみの非常にシンプルなものとなっている。また、タイムアウトによるWebブラウザからの切断を回避するために、50秒に一回強制的に現在のページをリロードするようにした。ただし、学生が回答を選択し、それを確定する前に画面のリロードが起こると、選択がクリアされてしまうなどの問題があった。

### 2-3.状態の保持

WWWの下層プロトコルであるTCPは、接続を維持するプロトコルであるのに、その上位プロトコルであるHTTPでは接続の維持を保証することができない。したがって、当然サーバはクライアント（Webブラウザ）の状態を保持することができない（ステイトレス）。

この問題を解決するために、Webブラウザが自己の状態をサーバに送信するcookieの仕組みが考え出されたが、cookieの利用においてはいくつかのセキュリティ上の脆弱性が指摘されている。特に今回のオンライン型CBTのようなシステムでは、試験中に他の悪意あるサイトを参照することによって、クロスサイトスクリプティング（XSS）[2]と呼ばれる攻撃を受ける可能性がある（可能性自体は非常に低いと思われる）。この手法を用いられると、cookieの内容をネットワーク上の第3者に知られてしまい、試験妨害などを受ける危険性がある。クライアントから送られてくるデータをサーバがどこまで信用するかという難しい問題も存在する。

また一般的に、cookieを使用するとWebブラウザは自己の状態をテキスト形式で勝手にサーバに送信するため、オンラインプロファイリングに利用されるなど、プライバシーの面でも問題があるとされている。実際にクッキーの使用に関してはいくつかの勧告や規制[3]、[4]も存在する。

cookieはHTTPによって一度破棄されたサービス（TCPのコネクション維持）をもう一度その上位層で実現しようとするものなので、どうしてもシステムが複雑になり、それに伴ってセキュリテ

ィが低下してしまう。

セッションを維持する方法としてはcookieを使用する方法の他に、URLやHTML文章のHIDDENフィールドに特定のセッション維持情報を埋め込む方法があるが、これらの方法はユーザにセッション維持情報を見られてしまう危険性がある。セッション維持情報として推測されにくい可変情報を使用することも可能だが、何度も使用すればセッション維持情報の生成のアルゴリズムを解析される恐れがある（ソースコードを解析されるかもしれない）。

### 2-4.クライアントの制御

オンライン型のCBTのようなシステムでは、ユーザが勝手な行動を取らないようにサーバがクライアントを完全に制御下におくことが必要となる（勝手に他のページを参照しないなど）。当然WWW（HTTP）ではこのようなことは不可能であり、テスト中にユーザが他のサイトのページを参照するといったことが起こり得る。また、ユーザにある一定の解答の手順や制限を要求するような場合は問題が発生する可能性がある。例えば、CCNA試験などのように必ず出題順に問題を解かなければならず、後戻りできないような場合や、一部の問題だけ順番に解かなければならない場合（前の問題の解答により次の問題が変化する場合など）は、ブラウザの「戻る」ボタンを使用されると問題が発生する可能性がある。

WWWシステムであっても、Java Scriptなどを使用すればある程度クライアントを制御可能だが完全に制御することは不可能である。

### 2-5.WWW システムの利点

開発者側にとってWWWシステムを利用することの利点は、システム開発のコストの点にある。今回のWWWをシミュレートしたシステムでも、クライアントにWebブラウザを使用したためクライアントソフトの開発コストは全くかからなかった。一方ユーザ（学生）にとっての利点は、最初からほぼ全てのパソコンにWebブラウザがインストールされており、クライアントプログラムを新たにインストールする必要がなく、ユーザの初期作業が殆ど無い点にある。また、ユーザは通常使い慣れているインターフェイスでシステムを使用することができるので、システムを使い始めるにあたり、その敷居を著しく下げる効果がある。これらは今日WWWによるシステムが数多く作られている要因でもあり、決して無視できない利点でもある。

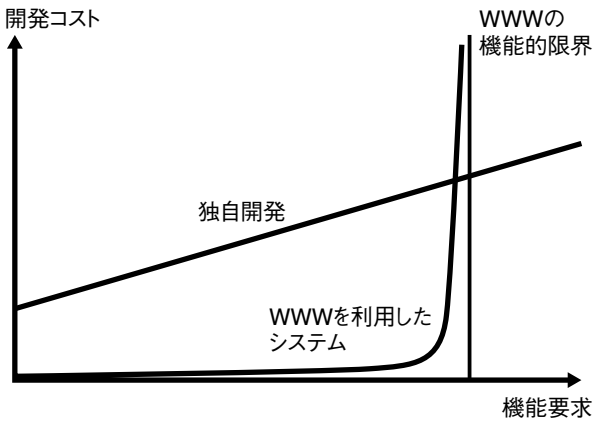


**図 4．開発コストとWWWの機能的限界**

### 2-6.WWW の機能的限界

WWWであっても当然機能的な限界は存在するであろう。接続的維持と状態の保持、及びクライアントの制御は現時点でその代表的なものであると思われる。図4にWWWを使用した場合と独自開発の場合の機能要求と開発コストの関係の予想図を示す。当然、実際の開発では問題はそれほど単純ではなく、グラフの横軸は複数の次元を持つことになるであろう。しかしながら、それらの1つの断面がグラフのようになることは十分予想できる。

機能要求がWWW本来の機能内にあれば、WWWを使用してシステムを作りあげることは十分意味があるし、またそうするべきである。しかしながら、開発しようとするシステムがWWWの機能を超える場合には独自開発を行うべきであると考えられる。WWWの機能を超えるものをWWW上に構築することは、WWWの本来の柔軟性を犠牲にする恐れがあり、見通しの悪いシステムになる可能性が十分にある。WWW上にシステムを構築するか、または新しくシステムを最初から作るか、問題はそのポイントかどこかを見極めることである。

### 2-7.オンラインテストの評価

実際のテストでは、出題する問題は全部で100題用意されており、ランダムにこの中から25題が出題される。各問題は4問択一の形式になっている（図3)。テストの制限時間は30分で、受験者の総数は158名であった。

図5、6に学生に対して試験と共に行ったアンケートの結果を示す。図5はコンピュータを使う試験と通常の紙で行う試験のどちらが良いかの結果である。「どちらでも良い」という意見もかなりあるが、はっきり「紙」の方が良いと答えた学生は「コンピュータ」の方が良いと答えた学生の半数以下となっており、オンラインテストの有用性が示されている。

また、図6はシステムに関する要望である（複数回答可)。「操作性の向上」などの問題もあるが、「家からの接続」、「複数回の受験」や「テスト以外での使用（自習)」などネットワーク上でコンピュータを使った試験に対する期待も感じられる。

「操作性の向上」に関して、Webによるシステムではインターフェイスの構築にいくらか制限が課せられる。ただし、Java Scriptなどのプログラム言語を使用すれば柔軟なインターフェイスを構築することは可能である。また「家からの接続」に関しては、十二分にセキュリティを考慮しなければならない。

### 2-8.WWW シミュレーションによるCBT システムのまとめ

学生へのアンケートにより、このようなシステムの有用性を確認することができた。さらに一部

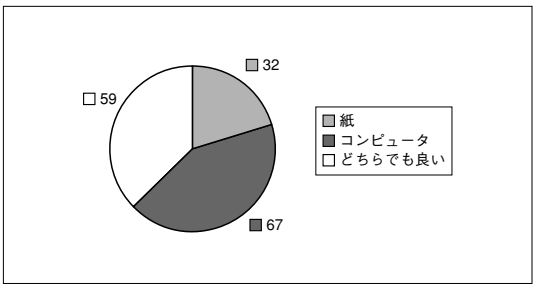


図5. 試験方法に関する意見

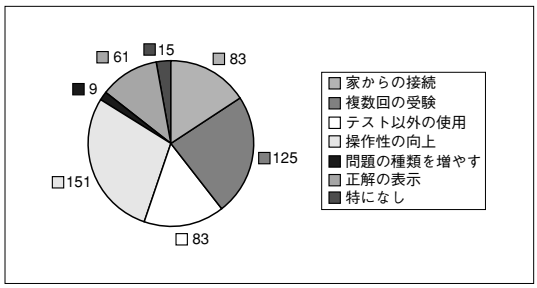


図6. システムに対する要望（複数回答可）

の学生（及び被験者）から「問題がパターン化すると、問題と回答のパターンマッチングになってしまう」との指摘を受けた。このようなシステムでは問題の取り扱いが非常に重要となる。できればデータベースを使用して、既存の問題から新しい問題を（半）自動生成するような機能が必要であると思われる。

また、WWWを使用したシステムでは最初の導入は非常に容易であるが、複雑な機能を追加しようとすると、サーバとクライアントの接続の維持、状態の保持及びクライアントの制御に関する問題点が表面化することを確認した。

## 3．独自プロトコルによるCBT システム

### 3-1.システムの概要

2章での結果を元に、我々はWWWを利用せずに、以下の点を考慮して独自プロトコルによるオンライン型のCBTシステムの開発を行った。

**1）接続の維持が保証されること。**
**2）サーバがクライアントの状態を保持すること（ステイトフル）。**
**3）サーバがクライアントの状態を完全に制御すること。**
**4）セキュリティが十分確保されること。**
**5）クライアントプログラムの汎用性。**
**6）データベースとの接続性。**

1）、2）、3）に関しては、コネクション型サービスのTCP上に直接クライアント・サーバモデルのアプリケーションを構築することにより実現可能である。4）に関しては当面はクライアント・サーバ間の通信にSSL（Secure Socket Layer）のソケットを使用する予定だが、将来的にはプログラム自体にSSLの機能を埋め込みたいと考えている。TCP上に直接SSLを実装したアプリケーションを構築できれば、セキュリティ的にかなり安全なシステムを構築することが可能となるだろう。5）に関してはクライアントソフトをJavaで作成することにより、クライアントマシンでのOS依存性を排除したい。また、JavaアプリケーションはJavaのVMがインストールされていさえすれば、通常コピーするだけで利用することが可能で、複雑なインストール作業を必要としない。6）に関してはフリーのリレーショナルデータベースサーバ（PostgreSQL）を用い、SQLをインターフェイスにして実現する。

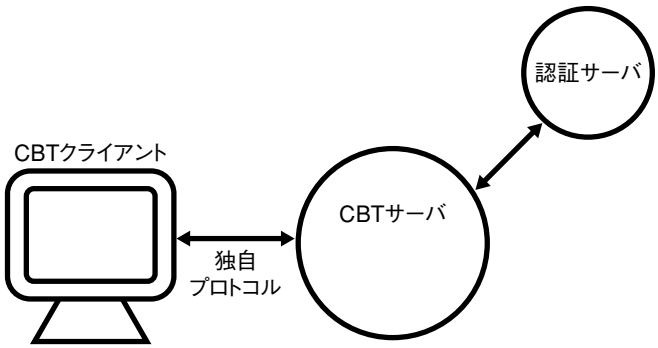


**図7．独自プロトコルによるCBTシステムの概要**

ネットワーク上での構成を図8に示す。WWWシミュレーションを用いた場合の図1とほぼ同じであるが、図1での認証用アプレット相当の機能をクライアントプログラム側に持たせることができるので多少シンプルな構成になっている。

### 3-2.プロトコル

CBTシステムのクライアント・サーバ間のプロトコルは新しく開発された独自プロトコルである。以下にその概要を示す。なお、認証サーバの通信プロトコルはこのプロトコルのサブセット版となっている。

### 3-2-1.応答プロトコル

クライアントからサーバへの要求（コマンド）発行時には、クライアントはサーバからOKまたはERRの応答を期待する。ただし、サーバがダウンしているかビジーの場合は応答待ちがタイムアウトする場合もある。クライアントからの要求（コマンド）に対するサーバからの応答の形式は次の3通りである。

1）正常応答

**OK［オペランド］｜コメント｜**

2）メッセージ（複数行可）を伴う正常応答

**OK**
**メッセージ（複数行可）**
**END**

3）エラー応答

**ERR［オペランド］｜コメント｜**

正常応答1）のOKのオペランドは応答の種類によっては指定しない場合もあるが、複数個になる場合（複数個の項目になる場合）もある。オペランドの項目数は、応答の元になったコマンドの種別に依存する。

正常応答時にサーバからクライアントへメッセージを送りたい場合は、2）の応答形式を使い、OKの後に改行してメッセージを送り、さらに改行してENDで終了する。メッセージは複数行も可能である。3）のエラー応答ERRのオペランドはエラー番号（数字）であり、エラーの内容を示す。1)、3）の応答形式におけるコメントはデバッグ用のものであり、クライアントの動作には影響しない（クライアントプログラムはコメントを無視する)。

応答（OK、END、ERR、オペランド）はケースインセンシティブ（大文字、小文字を区別ない）である。また、OK、ERR、オペランド、コメントの間は少なくとも一個以上の空白が必要である。コメントは複数行に渡ることはなく、CR+LF（0x0d、0x0a）もしくはLF（0x0a）の改行コードで終了し、応答の一行の最大長は1024バイト（改行コードを含む）である。1024バイトを超える部分は切り捨てられる。

### 3-2-2.コマンドプロトコル

クライアントからサーバへ送られる要求（コマンド）の形式は次のようになる。

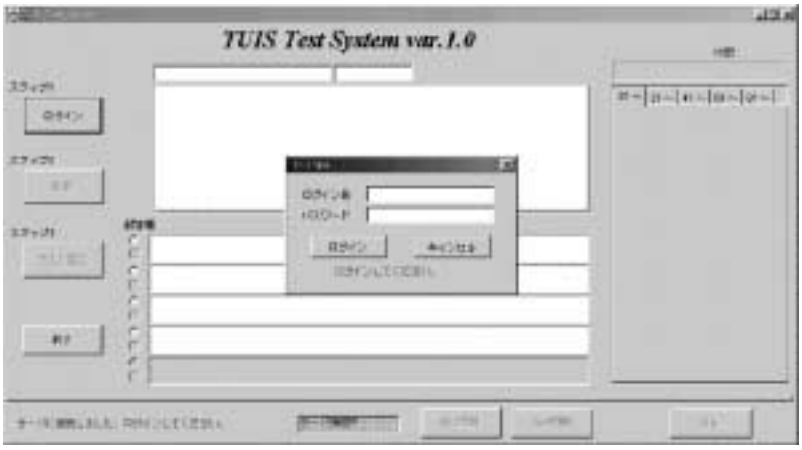


図8. クライアントシステムの全体画面とログイン画面


図9.　テスト項目の選択画面


図10.　テスト実施中のクライアント画面

＜コマンド＞　[オペランド]　｜コメント｜

オペランドはコマンドによっては指定しない場合もあるが、複数個の項目となる場合もある。オペランドの項目数はコマンドの種別によって規定されている。コマンドとオペランドは応答と同様にケースインセンシティブ（大文字、小文字を区別ない）とする。またこの場合のコメントも応答と同様にデバック用であり、サーバの動作には影響しない（サーバプログラムはコメントを無視する)。

コマンド、オペランド、コメントの間は少なくとも一個以上の空白が必要で、コメントは複数個あってもかまわない。コマンド行は複数行に渡ることはなく、CR+LF（0x0d、0x0a）もしくはLF（0x0a）の改行コードで終了し、応答の最大長は1024バイト（改行コードを含む）とする。1024バイトを超える部分は切り捨てられる。

各コマンドの詳細な説明については付録に記載する。

### 3-3.クライアントインターフェイス

クライアントシステムのインターフェイスを図8～10に示す。図8はクライアントシステムの全体画面とログイン画面を示している。図9はテスト項目の選択画面であり、現在は自習用の基本情報処理技術者と初級システムアドミニストレータおよび学内テスト用が用意されている。学生はこの画面により自由にテストを選択できる（学内テストに関しては、該当テスト受験の資格があるかどうかをサーバでチェックする)。

図10は実際にテストを行っている最中の画面である。中央上半分に問題が表示され、中央下半分の選択肢から回答を選択するようになっている。回答を1個選択する場合はラジオボタンによって選択する。また問題によっては回答を複数選択可能であり、複数選択の場合は自動的にラジオボタンが操作不能になり、チェックボックスが操作可能になる。右のタグは問題にマークをつけるためのタグで、分からなかった問題や自信ない問題にマークをつけ、後で改めてチェックするためのものである。右上部には経過時間が表示され、時間切れになると強制的に終了する。試験が終了するとその場で採点が行われる。

### 3-4.　独自プロトコルによるオンライン型CBTシステムのまとめ

独自プロトコルによるオンライン型CBTシステムは現在サーバプログラムと学生用のクライアントプログラムが完成しており、学期末のテストでの実際の運用を目標に準備が進められている。学期末テストで実際に学生に使用させることによって、インターフェイスの使用感などに対するデータを収集する予定である。

ただし現在のシステムでは、教員用（問題提出用）のインターフェイスとそのインターフェイス用の問題自動登録用データベースの機能の一部が未完成であり（手動で問題を登録することは勿論可能である)、問題文の半自動生成なども含めて今後の研究課題となっている。

## 4．全体のまとめ

教育現場において有効に使用できるオンライン型のCBTシステムを構築するために、WWWをシミュレートするシステムと独自プロトコルによるシステムを作成した。WWWをシミュレートする

システムでは、クライアントとサーバの接続の維持やサーバによるクライアントの状態保持と制御に問題があることが分かった。

独自プロトコルによるシステムではこのような問題は発生しないが、インターフェイスの使用感がWWWより劣ることが予想されるので、実際のテストで学生に使用してもらい、そのデータを収集する準備を現在行っている。

また、このようなシステムにおいては問題の内容が画一化しがちであり、学生にとっては問題文と回答のパターンマッチングになってしまう恐れがあるので、それを防止するために問題文を半自動的に生成する仕組みの研究などを行いたい。

## 参考文献

1）RFC2616
2）http://www.ipa.go.jp/security/awareness/vendor/programming/a01_02.html
3）http://www.ftc.gov/opa/2000/07/onlineprofiling.htm（米FTC によるプライバシー勧告）
4）http://www.internetnews.com/IAR/article.php/1154391（欧州連合議会によるプライバシー保護法案に関する記事）

## Appendix　[ ]は省略可能、< >は省略不可を表す

### 1. HELLO

HELLOコマンドはサーバに対する接続要求である。サーバ側は通信準備ができているなら、応答メッセージOKを返し、接続状態を初期化する。オペランドはクライアントのホスト名（省略可）である。このプロトコルは認証サーバのプロトコルと互換性がある。

**使用例**

| | |
|---|---|
| HELLO　[ホスト名] | ：サーバの接続状態を確かめる。 |

**正常応答**

| | |
|---|---|
| OK | ：サーバは応答可能。接続状態を初期化し、次へ進む。 |

**エラー応答**

通常エラー応答はしない。

### 2. USERID

USERIDコマンドはサーバにユーザのユーザIDを送る。オペランドはユーザID。サーバ上にユーザが存在すれば、crypt 関数用のSALTキーが 2 個返される（4octet）。具体的には a-zA-Z0-9./ から選ばれた 4 つの文字である。このプロトコルは認証サーバのプロトコルと互換性がある。

**使用例**

| | |
|---|---|
| USERID　＜ユーザID ＞ | ：サーバにUser ID を送る。 |

**正常応答**

| | |
|---|---|
| OK | |
| チャレンジコード | ：暗号化のためのコード、 4 octet。 |
| END | |

**エラー応答（一部）**

| | |
|---|---|
| ERR 101 User not Exist | ：サーバ上に指定されたユーザは存在しない。 |
| ERR 102 Cannot Get Password | ：サーバ側でユーザのパスワードを得ることができない。 |
| ERR 199 Unknown Error | ：その他のエラー。 |

### 3. PASSWD

PASSWDコマンドはサーバに暗号化されたユーザのパスワードを送る。オペランドは暗号化されたパスワードである。クライアント側はUSERIDコマンドで得た 2 個のSALT（SALT1、SALT2）を使ってcrypt関数でパスワードを

2回暗号化する。サーバ側もパスワードファイルにある暗号化されたパスワード（SALT1によって暗号化済み）をSALT2で暗号化し、クライアント側から送られてきた2重に暗号化されたパスワードと比較する。もし両者が一致するなら、クライアント側にOKを返し、ユーザの認証が完了する。このプロトコルは認証サーバのプロトコルと互換性がある。

**使用例**

PASSWD　<暗号化されたパスワード>　：サーバに暗号化されたパスワードを送る。

**正常応答**

OK　：ユーザの認証が完了した。

**エラー応答（一部）**

ERR 201 UserID Empty　：ユーザID が設定されていない。
ERR 202 Passwd Error　：パスワードが正しくない。
ERR 299 Unknown Error　：その他のエラー。

### 4. GETLIST

GETLISTコマンドはクライアントがサーバに、現在受けることのできるテストの全リストの転送を要求する。サーバはテストの全リストを返す。オペランドはない。

**使用例**

GETLIST

**正常応答**

OK
テストのリスト　：通常は複数行となる。
END

**エラー応答（一部）**

ERR 301 Not Found Server File　：サーバ上にリストデータが存在しない。
ERR 302 Not Authorized　：ユーザー認証がされていない。
ERR 399 Unknown Error　：その他のエラー。

### 5. TEST

TESTコマンドはサーバに選択したテストの種類（番号）を送信する。オペランドは選択したテスト番号。サーバは全問題数とテスト時間を返す。

**使用例**

TEST　<TEST No.>　：サーバにテスト番号を送信。

**正常応答**

OK
テスト時間（分）　全問題数　：テスト時間と問題数の間は一個以上の空白が必要。
END

**エラー応答（一部）**

ERR 401 Not Found Server File　：サーバにファイルが存在しない。
ERR 402 Not Found Test No.　：テスト番号が存在しない。
ERR 403 User Unknown　：指定ユーザはテストに登録されていない。
ERR 499 Unknown Error　：その他のエラー。

### 6. START

STARTコマンドはクライアントがテストサーバに対し、第一問の問題文を要求する。サーバは選択肢の数と種類（複数回答かどうか）と第一問の問題文を返信する。終了応答ENDと間違われないよう、問題文の先頭には必ず一個以上の空白が入る。選択肢はANSERコマンドで要求する。オペランドはない。

**使用例**

START

**正常応答**

OK
選択肢の数　種類
　第一問の問題文　　：複数行可。問題文の先頭には必ず空白が入る。
END

**エラー応答（一部）**

ERR 501 Not Found Server File　：サーバ上に問題文が存在しない。
ERR 502 Not Found Question No.　：問題番号が存在しない。
ERR 599 Unknown Error　　：その他のエラー。

### 7. NEXT

NEXTコマンドはクライアントがサーバに次の問題を要求する。サーバは選択肢の数と種類（複数回答かどうか）と問題文を返信する。終了応答ENDと間違われないよう、問題文の先頭には必ず一個以上の空白が入る。選択肢はANSERコマンドで要求する。オペランドはない。

**使用例**

NEXT

**正常応答**

OK
選択肢の数　種類
　問題文　　：複数行可。問題文の先頭には必ず空白が入る。
END

**エラー応答（一部）**

ERR 601 Not Found Server File　：サーバ上に問題文が存在しない。
ERR 602 Not Found Question No.　：問題番号が存在しない。
ERR 699 Unknown Error　　：その他のエラー。

### 8. BACK

BACKコマンドはクライアントがテストサーバに１つ前の問題文を要求する。サーバは選択肢の数と種類（複数回答かどうか）と問題文を返信する。終了応答ENDと間違われないよう、問題文の先頭には必ず一個以上の空白が入る。選択肢はANSERコマンドで要求する。オペランドはない。

**使用例**

BACK

**正常応答**

OK
選択肢の数　種類
　問題文　　：複数行可。問題文の先頭には必ず空白が入る。
END

**エラー応答（一部）**

ERR 701 Not Found Server File　：サーバ上に問題文が存在しない。
ERR 702 Not Found Question No.　：問題番号が存在しない。
ERR 799 Unknown Error　　：その他のエラー。

### 9. SKIP

SKIPコマンドはクライアントがサーバに任意の問題の問題文を要求する。サーバはその問題の選択肢の数と種類、問題文を返信する。終了応答ENDと間違われないよう、問題文の先頭には必ず一個以上の空白が入る。選択肢の文はANSERコマンドで要求する。オペランドは問題番号。

**使用例**

SKIP　<Question No.>

**正常応答**

OK

選択肢の数　種類
　問題文　　：複数行可。問題文の先頭には必ず空白が入る。
END

**エラー応答（一部）**

ERR 801 Not Found Server File　：サーバ上に問題文が存在しない。
ERR 802 Not Found Question No.　：問題番号が存在しない。
ERR 899 Unknown Error　：その他のエラー。

### 10. ANSWER

ANSWERコマンドはクライアントがサーバにこれから行う問題（直前に要求された問題）の選択肢の文章を要求する。クライアントは選択肢の番号をサーバに通知し、サーバはクライアントに対し選択肢を返信する。一回のANSWERでSelection No.で指定され選択肢を一個返すので、選択肢が4つある場合は4回このプロトコルを使用する。終了応答ENDと間違われないよう、選択肢の先頭には必ず一個以上の空白が入る。オペランドは選択肢の番号。

**使用例**

ANSWER　<Selection No.>

**正常応答**

OK
　選択肢の内容　　：複数行可。選択肢の先頭には必ず空白が入る。
END

**エラー応答（一部）**

ERR 901 Not Found Server File　：ファイルが存在しない。
ERR 902 Not Found Question No.　：問題番号が存在しない。まだ問題が要求されていない。
ERR 903 Not Found Answer　：選択肢が存在しない。
ERR 999 Unknown Error　：その他のエラー。

### 11. RESULT

RESULTコマンドはクライアントがサーバに採点を要求する。サーバはクライアントから送られてきた解答から採点を行い、得点を返信する。オペランドは回答列（回答を繋げた文字列）。

**使用例**

RESULT　<解答列>

**正常応答**

OK
点数
END

**エラー応答（一部）**

ERR 904 Server Not Ready　：サーバの準備ができていない。
ERR 905 Already Transfer　：解答は既に転送済み。
ERR 906 File Not Found　：サーバ上に解答ファイルがない。
ERR 909 Unknown Error　：その他のエラー。

### 12. BYE

BYEコマンドはサーバに切断要求を送信する。オペランドはなし。サーバからOKが返って来た場合、セッションは正常に終了したことになる。このプロトコルは認証サーバのプロトコルと互換性がある。

**使用例**

BYE

**正常応答**

OK　：サーバは接続を切断する。

**エラー応答**

通常エラー応答はしない。