情報工学実験 C ネットワーク実験課題レポート

―クライアントサーバーモデルで動作する名簿管理プログラムの作成―

氏名: 寺岡 久騎 (TERAOKA, Hisaki)
学生番号: 09B22433

出題日: 2024 年 12 月 13 日
提出日: 2025 年 1 月 21 日
締切日: 2025 年 1 月 21 日

1 プログラムの処理の概要および作成方針

本実験では，ネットワーク通信により動作するクライアントサーバーモデルの TCP による相互のデータ送受信を行う名簿管理プログラムの作成を行った．本章ではサーバ・クライアントプログラムのそれぞれの機能と，プログラム全体としての処理の概要について解説する．

1.1 プログラム全体の処理の流れと概要

プログラム全体の処理の流れとして，まずサーバはクライアントからの TCP の接続要求を待ち受け，クライアントはサーバに接続する．コネクションを確立することで名簿管理の処理が開始し，以降はシステムコール関数を用いたソケット通信により以下の一連の処理を繰り返す．

1. クライアントが名簿管理に関する標準入力からの入力をサーバにメッセージとして送信する
2. サーバがクライアントからのメッセージを受信する
3. サーバがメッセージに対応した名簿データに関する処理を行い，その結果をクライアントにメッセージとして送信する
4. クライアントがサーバーから受信したメッセージを標準出力へ出力する

クライアントとサーバは一方が送信処理をする際にもう一方が受信処理を行う様に通信を同期してやりとりを行う．処理の終了時にどちらも接続を切断し，クライアントはプログラムを終了し，サーバは次のクライアントの接続を待ち受ける状態となる．

プログラムで扱う名簿データは「ID，氏名，誕生日，住所，備考」の項目からなる CSV 形式であり，サーバはこの形式のデータを受信するとこれを新たな名簿データとしてメモリに保存する．また，'%' から始まるメッセージは名簿データに対する様々な処理を実行するためのコマンド入力であり，サーバからクライアントへのメッセージは主にこのコマンド処理の結果である．以下，実装したコマンドに対応した処理や機能，及び処理結果の出力について示す．

■%Q コマンド　クライアントプログラムを終了し，サーバープログラムはクライアントとの接続を終了して次のクライアントプログラムの接続を待ち受ける状態となる．

■%C コマンド　登録した名簿データ数の表示を行う．サーバからクライアントへ登録されている名簿データの数と，登録可能な残りのデータ数に関するメッセージを送信する．

■%P コマンド　引数に入力された整数値に応じて登録されている名簿データを表示する．サーバはクライアントへ該当の名簿データ全ての各項目を表示した文字列をメッセージとして送信する．引数の仕様を以下に示す．

- 0 の場合：登録順に全件表示
- 負の場合：登録データの後ろから順に絶対値の数だけ表示
- 絶対値が総データ数以上の場合：登録順に全件表示

■%R コマンド　サーバ側で引数に指定された名前のファイルを読み込み，CSV 形式で記された文字列を名簿データとして登録を行う．

■%W コマンド　サーバ側で引数に指定された名前のファイルへ，登録されている全ての名簿データを CSV 形式で出力する．

■%S コマンド　引数で指定された整数値に対応する項目に関して，サーバプログラムのメモリに登録されている名簿データのソートを行う．

■%F コマンド　引数に入力された文字列と完全に一致する項目を持つ名簿データ全ての表示を行う．%P コマンドと同様に，サーバは該当の名簿データの情報をメッセージとしてクライアントへ送信する．

■%D コマンド　引数に入力された整数値に応じて，サーバがメモリに保存している名簿データを削除する機能を持つ．引数の仕様を以下に示す．

- 0 の場合：全データ削除
- 正の場合：先頭から引数の数だけ削除
- 負の場合：登録データの後ろから順に引数の絶対値の数だけ表示
- 絶対値が総データ数以上の場合：全データ削除

1.2　クライアントプログラムの機能と作成方針

クライアントは始めにサーバとの TCP コネクションの確立を行う処理をする．接続にはサーバが動作する計算機のホスト名と，サーバプログラムのポート番号が必要であり，これらの情報はプログラム実行時の引数として入力し，これをそれぞれ IP アドレス，ポート番号として接続の処理に利用するという設計にした．

サーバとの接続が確立した後は，標準入力からの入力された文字列をサーバへメッセージとして送信し，その後サーバの処理結果を受信して標準出力へ出力するという処理を繰り返す．この繰り返しの実現には，TCP の接続処理の後にループの構文を設け，その中でメッセージの入力と送信，受信と出力の処理を行うという方針を取った．

%Q コマンドによるプログラムの終了は，処理の複雑化を避けるため，上記のサーバとのメッセージの送受信・同期の処理から分岐せず，サーバからのメッセージを受信した後にループから抜けると

いう設計にした.

1.3 サーバプログラムの機能と作成方針

サーバープログラムは，まず通信相手となるクライアントからの接続要求を待ち受ける．この機能ではクライアント側が接続をするために，プロセスのポート番号を設定する必要があり，これをプログラム実行時の引数として入力された文字列を整数値として変換することで実装を行った.

クライアントとの接続後は以下の処理を行う.

1. クライアントの接続を受け付ける
2. クライアントからのメッセージを受信する
3. メッセージに対応した名簿管理の処理を行う
4. 処理の結果をクライアントに送信，2 の処理へ
5. クライアントとの接続終了後，1 に戻る

処理 2–4 はクライアントプログラムと同様に，クライアントとのやり取りが終了するまで繰り返されるため，ループ処理を設けてその中に実装する方針とした．クライアントから %Q コマンドが送信された場合は接続の終了を示すメッセージを送信してクライアントとの接続を終了した後，プログラムはプロセスを終了せず，次のクライアントプログラムの接続を待ち受けるため，処理 1–5 を無限ループの処理の中に設けることで実現した．そのため，サーバプログラムの終了はキーボードの Ctrl-C 入力もしくは `kill` コマンドで行う．Ctrl-C 入力は本来強制的にプロセスを終了させるが，本プログラムは第 1.3.4 項に示す様に安全に終了する機能を持つ.

1.3.1 多重通信受付対応機能 (発展課題 2)

今回作成したサーバプログラムは複数のクライアントのプロセスからの接続を同時に処理することが可能な多重通信受け付け機能を持つ．実現にあたっては，接続の待ち受けをする処理とコネクションを確立し名簿管理のデータ通信を行う処理を分けるという仕組みが有効であると考え，接続要求の待ち受けを主に行う元のサーバプログラムを親プロセスとして TCP コネクションの確立の後にそれぞれのクライアントに対応した通信処理を行うプロセスを複製する方針により実装を行った．機能の確認のため，サーバは複製したプロセスの ID 及びメッセージのやり取りを行ったクライアントとの通信に使用しているソケットのディスクリプタの値を表示する．処理の概要については第 2.1.1 節に示す.

1.3.2 サーバログ機能 (発展課題 3)

社会で利用されるサーバプログラムやシステムでは情報の管理・分析やセキュリティ対策のために接続元の IP アドレス等の情報，アクセスした時刻やイベントをログとして記録することが多い．そのため，本演習で作成したプログラムにおいてもサーバ側で様々な情報をログとして記録する機能を実装した.

ログに記録する情報は接続元の IP アドレスとポート番号，実行されたコマンドとメッセージにより登録された名簿データである．ログファイルの作成はクライアントとの TCP コネクション確立に行い，メッセージの送受信のループ処理の直前でその時点での時刻と接続元の IP アドレスとポート番号をファイルへ書き込む．クライアントから受信したデータに関しては，実行可能なコマンドと登

録可能な形式の名簿データのみを書き込む.

1.3.3 クライアントプログラム停止時の一時データ記録と復帰処理 (発展課題 3)

サーバプログラムは，クライアントとのやりとりと通信の終了を %Q コマンドのメッセージ受信により行う．しかし，クライアントプログラムがヒューマンエラー等の理由により Ctrl-C 入力で終了した場合は扱っていた名簿データが保存されないまま失われてしまう．そのため，サーバ側でクライアントが Ctrl-C 入力で終了したことを検知し，それを異常終了とみなして一時的にファイルへその時点で保存している名簿データを書き出し，以降接続するクライアントにその一時データを読み込む選択ができる機能を実装した.

一時的に記録したファイルがある場合は新たに接続したクライアントにメッセージを送信し，そのデータを読み込むがどうかを選択させ，選択された場合はファイルの内容を名簿データとして読み込んだ上で一時ファイルを削除する．選択されなかった場合はそのファイルを削除せず，データも読み込まない.

1.3.4 Ctrl-C 入力時の安全なプログラムの終了処理 (発展課題 3)

サーバプログラムはクライアントの接続要求を待ち受けるために無限ループの処理を設けており，プロセスの終了には Ctrl-C 入力により強制終了させる方法がある．しかし，この場合では通信で使用している待ち受け用のソケット，クライアント用のソケットの削除，1.3.2 項で示したログファイルのファイルディスクリプタの削除など，本来プロセスを終了する前に行うべき処理が行われない．そのため，サーバプログラムでは Ctrl-C 入力時にこれらの処理を行い安全に終了し，正しくファイルやソケットのディスクリプタを削除したメッセージを標準出力へ標準する．機能を実装した．この機能では，上記の処理に状態のままになることを防ぐために Ctrl-C 入力時にはプロセスにキーボードからの割り込みを示すシグナルが送信されるため，機能の実装にはプログラムで `signal` システムコールを利用して，このシグナル発生時に上記の行うべき処理を行う方針をとった.

2 プログラムの処理の説明

サーバ・クライアントのプロセスは第 1.1 節で示した様に TCP コネクションを確立した後に，相互に送信と受信の対応を同期させてメッセージのやりとりを行う．これら一連の処理の流れについて

1. サーバ・クライアントのコネクションの確立
2. サーバの安全なプロセス終了処理の設定 (発展課題 3)
3. クライアントの標準入力からの入力データ取得・サーバへのメッセージ送信
4. サーバのメッセージ受信
5. サーバの受信メッセージによる名簿データの処理，クライアントへの処理結果のメッセージ送信
6. クライアントのメッセージ受信と標準出力への出力

に分けて解説する.

2.1 サーバ・クライアントのコネクションの確立

2.1.1 サーバプログラム

サーバプログラムは，始めにクライアントの接続要求の受け付け処理を行う．まずソケットの待ち受けの設定を行うために sockaddr_in 構造体の変数を宣言し，これに対して

- ソケットで利用するアドレスタイプに IPv4(AF_INET) を設定する
- プログラム実行時の引数で受け取った文字列をポート番号として atoi 関数により変換した整数値を htons 関数を用いてネットワークバイトオーダに変換して設定する
- どのアドレスからの要求を受け付けるために，受け付ける IP アドレスの項目を INADDR_ANY として htonl 関数によりネットワークバイトオーダに変換して設定する．

という処理を行う．次に socket 関数にソケットで利用するアドレスタイプとして第 1 引数に IPv4 を指定する AF_INET を，第 2 引数に通信形式として TCP を指定する SOCK_STREAM を設定して，ソケットの生成及びその戻り値としてディスクリプタの値を受け取る．このディスクリプタの値と先述の sockaddr_in 構造体の変数を用いて bind 関数によりソケットに待ち受けるための設定を行う．

待ち受け用のソケットを作成した後，そのディスクリプタを引数として listen 関数によりサーバとしてクライアントの接続待ち受けを開始する．第 2 引数に設定する一度に通信する上限のクライアント数は 5 と設定した．次に，クライアントとの接続要求の受付を行うためのループ処理に進む．この中の処理の始めで待ち受け用のソケットのディスクリプタを引数として accept 関数で接続要求を受け取り，クライアントとの通信に使用するソケットを生成する．この時，accept 関数の引数に adddinfo_in 構造体の変数のポインタを sockaddr 構造体のポインタにキャストして渡し，クライアントの情報を取得し，この変数を用いて第 1.3.2 項で示したログ機能により open 関数によるクライアント用のログファイルの作成とクライアントの IP アドレスとポート番号，時刻の記入を行う．

クライアントとの通信を行うソケットの生成を行った後，第 1.3.1 項で示した多重通信受付のために fork 関数によりその時点での処理の進行，メモリ状態が同じであるプロセスの複製を行う．この時の戻り値として受け取ったプロセス ID が 0 より大きい親プロセスとプロセス ID が 0 である複製された子プロセスで処理を if 文により分岐させ，それぞれ以下の処理を行う．

親プロセス　接続したクライアントとのメッセージ送受信の処理は行わず，再びループの始めに戻り，accept 関数で次のクライアントの接続待ち受ける．

子プロセス　始めに第 1.3.3 項で示した一時記録された名簿データの読み込みを選択する処理を行う．その後，クライアントとのメッセージのやりとりを行うループの処理に入る．クライアントとの通信の終了後はクライアントの接続要求を受け付けるループ処理を抜け，クライアントとの通信用ソケットを close 関数で削除してプロセスを終了する．また，この時ログ機能により作成したファイルのディスクリプタも close 関数により削除する．

これによって

- 親プロセスはクライアントの接続要求を受け続ける処理を繰り返す
- 子プロセスはクライアントとのメッセージの送受信を行い，通信終了後にプロセスを終了する

という処理の流れになり，複数のクライアントのプロセスからの接続要求を同時に処理することがで

きる．また，サーバプログラムでは，クライアントとのコネクションを確立してプロセスを複製すると，親プロセスが子プロセスのプロセス ID を標準出力へ表示する．

2.1.2 クライアントプログラム

クライアントプログラムは始めに，接続先のサーバプロセスとの TCP コネクションの確立の処理をする．サーバに接続するために必要なサーバプロセスのホスト名とポート番号はプログラム実行時に引数として文字列データとして受け取るため，こを引数として `getaddrinfo` 関数によって，ソケットの生成に必要な項目の値が設定された `addrinfo` 構造体のデータを取得する．このデータを利用して `socket` 関数によりサーバとの接続に利用するソケットのディスクリプタを得て，この値を引数に `connect` 関数によりサーバとの接続を確立する．コネクションの確立後は `while` 文の中で以降解説するサーバとのやりとりを繰り返し行う．

2.2 サーバの安全なプロセス終了処理の設定 (発展課題 3)

サーバでは，第 1.3.4 項に示した Ctrl-C 入力時の安全なプロセスの終了処理を待ち受け用ソケットを作成した後に設けている．プログラムでは `signal` 関数により，Ctrl-C 入力時にプロセスに送信される `SIGINT` シグナルを第 1 引数として，第 2 引数にプロセス終了前に行う処理をまとめた関数を渡す．この関数では，待ち受け用のソケットのディスクリプタとログファイルのディスクリプタの削除を行う．この時，子プロセスは削除したログファイルのディスクリプタとクライアント通信用ソケットディスクリプタの値を表示し，親プロセスはクライアントんも接続待ち受けに使用していたソケットのディスクリプタの値を表示する．

2.3 クライアントの標準入力からのデータ取得・サーバへのメッセージ送信

標準入力からの入力データの受け取りとサーバへのメッセージの送信は同じ `while` 文中で行い，以下の流れで一連の処理を行う．

まず標準入力からの入力データをシステムコール関数の `read` により取得し，サイズが 10 バイトのバッファー (`char` 型の配列) に格納する．そしてこのバッファーと `read` 関数の戻り値である取得したバイト数を引数として，`send` 関数によりサーバへメッセージを送信する．ユーザからのメッセージ入力の終了は Enter キーの押下による改行文字を目印とし，`read` 関数の処理後に取得バイト数を利用して入力文字列の末尾の改行文字 (`'\n'`) があるかを確認し，あればその要素を送信終了を示す目印としてメッセージで利用しない ASCII コードの整数値 3(`0x03`) に変更し，`send` 関数により送信した後にループ処理から抜ける．その後，サーバからのメッセージを受信する処理へ移行する．

2.4 サーバのメッセージ受信

クライアントからのデータ受信はクライアントの送信処理と同様に `while` 文中でシステムコール関数の `read` により取得し，サイズが 10 バイトの受信用バッファー (`char` 型の配列) に格納することを繰り返す．ここで，名簿管理に使用するメッセージは連結した 1 つの文字列データとして扱うため，予め宣言した最大入力文字数 (1024)+1 を格納できる `char` 型のメッセージ用バッファーへ読み込んだバッファーの内容をコピーを行う．この処理は以下の流れにより行う．

1. `while` 文の直前で名簿管理処理用のメッセージ用バッファーの先頭アドレスをポインタ変数

に格納する

2. while 文の繰り返し処理において，read 関数の処理後，戻り値の取得バイト数だけ strncpy 関数により受信用バッファーの内容を先述のポインタの位置へコピーする
3. 上記ポインタ変数を取得バイト数だけ加算する．

これにより，繰り返し受信したデータを連結したメッセージとして保存を行う．受信処理の終了は，read 関数の処理後に受信用バッファー内にクライアントプログラムにより付与された送信終了の目印である整数値 3(0x03) があるかを確認し，ある場合はその要素を終端文字 ('\0') へ変更し，メッセージ用バッファーへのコピー後に受信のループ処理を抜ける．このメッセージ中の送信終了の目印により，クライアントの send による送信回数に依らず，サーバはメッセージの受信を適切に終了を行う．

また，多重通信受付対応の確認のため，メッセージを送信してきたクライアントとの通信に使用しているソケットのディスクリプタの値を表示する．

2.5 サーバの受信メッセージによる名簿データの処理・クライアントへのメッセージ送信

メッセージの受信後，メッセージ用バッファーに格納した文字列を解析して名簿管理に関する処理を行う．この時，メッセージが実行可能なコマンドまたは登録可能な名簿データである場合は第 1.3.2 項で示したログファイルに書き込む．処理の結果，クライアント側にメッセージを送る場合は，クライアントの送信処理と同様にサイズが 10 バイトの送信用バッファーを用いて，終端文字も含めたメッセージのバイト数を 10 バイトずつ分割して必要な回数だけ send 関数によりクライアントへデータの送信を繰り返す．

この処理の後，受信メッセージが %Q コマンドでない場合はサーバのメッセージ送信終了の目印として，整数値 3(0x03) を代入した 1 バイトのデータを send 関数により送信し，次のクライアントプログラムからのメッセージを受信する処理へ移行する．%Q コマンドであった場合は，このデータをクライアントへ接続終了を示す目印として，メッセージ終了の目印と同様に通常のメッセージで使用されない ASCII コードの整数値 4(0x04) を代入して送信し，クライアントとのメッセージのやりとりを行うループ処理を抜け，クライアントとの通信用ソケットを close 関数により削除する．

以上の様に，コマンドにより名簿管理に関するメッセージの送信をするかどうかに関わらず，必ずクライアント側へサーバの処理が終了したことを表す目印をメッセージと送信を行う．これによりクライアントの受信処理を終了させ，送受信の同期を行う．

2.6 クライアントのサーバからのメッセージ受信・標準出力への出力

サーバからのメッセージの受信と標準出力への出力は第 2.3 項で解説した処理と同様に，while 文中で行い，以下の流れで処理を行う．

まずサーバからの送信されたデータを recv 関数によりサイズが 10 バイトの受信用バッファーに格納し，この受信用バッファーのメッセージをシステムコール関数の write 関数により，標準出力へ recv 関数の戻り値である取得したバイト数だけ出力する．メッセージ出力処理の後はサーバのクライアントメッセージの受信処理と同様にバッファー内にサーバが施したメッセージの終了に関する目印に該当する要素があるかを確認し，あれば以下の処理を行う．

整数値 3(0x03)　メッセージの送信終了を示し，受信処理のループを抜け，再びユーザからの入力とサーバへのメッセージ送信を行う処理へ移行する．

整数値 4(0x04)　%Q コマンド入力判定による通信の接続終了を示し，サーバとのメッセージ送受信処理のループを抜け，ソケットを close 関数により削除してプロセスを終了する．

また，recv 関数による受信データの取得後は，recv 関数の戻り値が 0 であり，接続相手のサーバが通信していたソケットを閉じた場合は上記の %Q コマンド入力判定時の接続終了と同じ処理を行う．

3　プログラムの使用方法と使用例及び動作結果

本実験で作成したサーバ・クライアントのプログラムの，実際に使用する方法，使用例及びそれらの動作結果について以下の操作に関して示す．

- サーバ・クライアントプログラムの起動と通信の接続
- 名簿管理に関するコマンドの実行
- 他学生のサーバプログラムとの通信
- サーバの多重通信受付対応
- Ctrl-C 入力でのサーバプロセスの安全な終了
- Ctrl-C 入力でのクライアント停止時の一時データ記録と復帰処理

3.1　サーバ・クライアントプログラムの実行と通信の接続

サーバプログラムでは，通信相手からの接続要求を受け付けるため，プロセスのポート番号を設定する必要がある．これはプログラム実行時に入力された引数を利用する．

- 実行ファイル ポート番号

例としてポート番号を 60000 として待ち受ける際の使用例を以下に示す．サーバプログラムの実行ファイル名は meibo_server としている．

```
$ ./meibo_server 60000
```

これにより，引数で与えたポート番号でクライアントの接続を受け付ける状態となる．

クライアントプログラムでは，サーバとの接続のために，サーバのホスト名とポート番号を設定する必要がある．これもサーバプログラムと同様にプログラム実行時の引数として与え，第 1 引数がホスト名 (IP アドレスまたはドメイン名)，第 2 引数がポート番号となっている．

- 実行ファイル ホスト名 ポート番号

同一計算機上で上記の例で示したサーバが動作している場合のクライアントの使用例を以下に示す．クライアントの実行ファイル名は meibo_client としている．

```
$ ./meibo_client localhost 60000
[ Server Info ] IP Address: [ 127.0.0.1 ] - Port: [ 60000 ]
```

プログラム実行後，プロセスが終了しない状態であればサーバとの接続が完了し，名簿管理に関する

処理が行えるようになる．

3.2 名簿管理に関するコマンドの実行

以下，サーバと接続後のクライアントプログラムでの各コマンドを実行した際の動作例を示す．サーバは前節の例と同様に同一計算機上でポート番号 60000 で接続要求を待ち受けているものとする．

3.2.1 %Q コマンド

以下，サーバへの接続直後に %Q コマンドを実行した際の動作例を示す．

```
$ ./meibo_client localhost 60000
[ Server Info ] IP Address: [ 127.0.0.1 ] - Port: [ 60000 ]
%Q
See you!
```

コマンド実行後．See you!のメッセージが表示され，プロセスが終了する．

3.2.2 %C，R，P コマンド

以下，%C，R，P コマンドを使用した際の動作例を示す．機能の動作を確認する上で名簿データが必要であったため，そのデータとして実験で配布された sample.csv ファイルを使用している．このファイルには 2886 データが記録されている．コマンド操作は以下の順で行った．

1. %C
2. %P (全データ表示)
3. %R (sample.csv の読み込み)
4. %C
5. %P (先頭から 3 データ)

```
%C
0 profile(s)
Enable to add 10000 data(s)
%P 0
%R sample.csv
%C
2886 profile(s)
Enable to add 7114 data(s)
%P 3
Id    : 5100046
Name  : The Bridge
Birth : 1845-11-02
Addr. : 14 Seafield Road Longman Inverness
Comm. : SEN Unit 2.0 Open

Id    : 5100127
Name  : Bower Primary School
Birth : 1908-01-19
Addr. : Bowermadden Bower Caithness
Comm. : 01955 641225 Primary 25 2.6 Open

Id    : 5100224
Name  : Canisbay Primary School
```

```
Birth : 1928-07-05
Addr. : Canisbay Wick
Comm. : 01955 611337 Primary 56 3.5 Open
```

%R コマンドによるデータの読み込みにより，%C コマンドで表示されるデータ数がファイルに登録されていた数になり，%P コマンドでも指定した数だけ名簿データの表示がされた．

3.2.3 %W コマンド

%W コマンドによる登録データのファイルへの書き出しの動作例を示す．動作確認のため，%W コマンドも含めて以下の操作を順に行った．

1. 適当な名簿データを 3 つ登録する
2. %C コマンドでデータ数を確認する
3. %W コマンドで copy.csv という名前のファイルにデータを書き出す．

```
100,takahashi,2001-1-1,tokyo,no comment.
200,suzuki,2002-2-2,osaka,no comment.
300,honda,2003-3-3,kyoto,no comment.
%C
3 profile(s)
Enable to add 9997 data(s)
%W copy.csv
```

実行後の copy.csv の内容を以下に示す．コマンド実行時点で登録していた名簿データが記録されている．

```
1: 100,takahashi,2001-1-1,tokyo,no comment.
2: 200,suzuki,2002-2-2,osaka,no comment.
3: 300,honda,2003-3-3,kyoto,no comment.
```

3.2.4 %S コマンド

前項で登録した 3 つの名簿データがある状態で %S コマンドによりデータをソートした際の実行結果を示す．操作は以下の順で行った．

1. %P コマンドで全データを表示する
2. %S コマンドで名前 (第 2 項目) についてソートする
3. %P コマンドで全データを表示する

```
%P 0
Id    : 100
Name  : takahashi
Birth : 2001-01-01
Addr. : tokyo
Comm. : no comment.

Id    : 200
Name  : suzuki
Birth : 2002-02-02
Addr. : osaka
Comm. : no comment.
```

```
Id    : 300
Name  : honda
Birth : 2003-03-03
Addr. : kyoto
Comm. : no comment.

%S 2
%P 0
Id    : 300
Name  : honda
Birth : 2003-03-03
Addr. : kyoto
Comm. : no comment.

Id    : 200
Name  : suzuki
Birth : 2002-02-02
Addr. : osaka
Comm. : no comment.

Id    : 100
Name  : takahashi
Birth : 2001-01-01
Addr. : tokyo
Comm. : no comment.
```

名前の項目でアルファベット順にソートされたことが確認できる．

3.2.5 %F コマンド

%W，%S コマンドの動作例で使用した 3 つの名簿データが登録された状態で %F コマンドにより引数で指定した文字列と一致する項目でを持つ名簿データの検索を行った実行結果を示す．操作は以下の順で行った．

1.「100」に一致する項目を持つデータを検索する
2.「no comment.」に一致する項目を持つデータを検索する

```
%F 100
[No.3]
Id    : 100
Name  : takahashi
Birth : 2001-01-01
Addr. : tokyo
Comm. : no comment.

%F no comment.
[No.1]
Id    : 300
Name  : honda
Birth : 2003-03-03
Addr. : kyoto
Comm. : no comment.

[No.2]
Id    : 200
Name  : suzuki
Birth : 2002-02-02
Addr. : osaka
Comm. : no comment.
```

```
[No.3]
Id    : 100
Name  : takahashi
Birth : 2001-01-01
Addr. : tokyo
Comm. : no comment.
```

3.2.6 %D コマンド

%D コマンドによる登録データ削除機能の動作例を示す．動作確認のため，他のコマンドも含めて以下の操作を順に行った．

1. %R コマンドで sample.csv ファイルのデータを読み込む (2886 データ)
2. %C コマンドで登録データ数を確認する
3. %D コマンドで先頭から 2880 データを削除
4. %C コマンドで登録データ数を確認する

```
%R sample.csv
%C
2886 profile(s)
Enable to add 7114 data(s)
%D 2880
%C
6 profile(s)
Enable to add 9994 data(s)
```

3.3 他学生のサーバプログラムとの通信 (発展課題 1)

作成したクライアントプログラムを他学生の作成したサーバプログラムに接続する際の使用例を以下に示す．

サーバプログラムは学生番号：09B22509，氏名：垣本桃弥さんが作成したものを使用した．

動作確認に関する操作は以下の順で行った．

1. 相手方のサーバプロセスが動作している計算機の IP アドレス，ポート番号を引数としてクライアントプログラムを実行する
2. %C コマンドでデータ数を確認する
3. %R コマンドで sample.csv ファイルのデータを読み込む (ファイルはサーバ側のものが対象)
4. %C コマンドでデータ数を確認する
5. %P コマンドで先頭から 3 データを表示する
6. %Q コマンドでサーバとの通信を終了，クライアントプログラムを終了する

```
$ ./meibo_client 61003
%C
0 profile(s)
%R sample.csv
%C
2886 profile(s)
%P 3
Id    : 5100046
```

```
Name  : The Bridge
Birth : 1845-11-02
Addr. : 14 Seafield Road Longman Inverness
Comm. : SEN Unit 2.0 Open

Id    : 5100127
Name  : Bower Primary School
Birth : 1908-01-19
Addr. : Bowermadden Bower Caithness
Comm. : 01955 641225 Primary 25 2.6 Open

Id    : 5100224
Name  : Canisbay Primary School
Birth : 1928-07-05
Addr. : Canisbay Wick
Comm. : 01955 611337 Primary 56 3.5 Open

%Q
$
```

3.4 サーバログ機能 (発展課題 3)

サーバログ機能は，接続したクライアントそれぞれのログファイルを作成する．ログファイルには

- IP アドレス
- ポート番号
- 接続した時刻
- 実行したコマンド・入力により登録した名簿データ

を記入する．ログファイル名は接続時の時刻に基づき，**年-月-日_時-分-秒**.log となる．以下，クライアントが接続し，以下の操作を行った際のサーバプロセスの出力と作成されたログファイルの記録について示す．

1. %C コマンドの実行
2. 無効なコマンドの入力
3. 新たな名簿データの入力
4. %P コマンドで全データを表示
5. %R コマンドによる sample.csv の読み込み
6. %Q コマンドによる接続の終了

■サーバプロセスの出力

```
$ ./meibo_server 60000
log file: log/2025-01-19_16-32-49.log
<connect>         IP [127.0.0.1] PORT [54344] SOCKET: [4]
make process pid: 36692
msg: [%C] #sockfd: 4
msg: [%A] #sockfd: 4
msg: [100,takahashi,2001-1-1,toyko,no comment.] #sockfd: 4
msg: [%P 0] #sockfd: 4
msg: [%R sample.csv] #sockfd: 4
msg: [%Q] #sockfd: 4
```

```
<disconnect>          IP [127.0.0.1] PORT [54344] SOCKET: [4]
```

■ログファイルの内容

```
1: [TIME] 2025/01/19 16:32:49
2: [IP]   127.0.0.1
3: [PORT] 54344
4: %C
5: 100,takahashi,2001-1-1,toyko,no comment.
6: %P 0
7: %R sample.csv
8: %Q
```

仕様の通り，有効なコマンドと新たに登録した名簿データが記録されていることが確認できた．

3.5 Ctrl-C 入力でのサーバプロセスの安全な終了 (発展課題 3)

サーバプロセスが実行時に Ctrl-C 入力された際，サーバは通信に使用していたソケットとログファイルのディスクリプタを close 関数で削除を行い，その値を表示する．

以下，1 つのクライアントプログラムが接続されている状態でサーバプロセスを Ctrl-C で終了させた際のサーバ及びクライアントの出力結果を示す．クライアントがサーバプログラムの終了をメッセージの受信により判別するため，一度何かしらの入力を行いメッセージ送信処理を終える必要があるため，コマンドを入力している．

■サーバプロセスの出力

```
$ ./meibo_server 60000
make process pid: 45549
<connect>          IP [127.0.0.1] PORT [58376] SOCKET: [4]
^C[EXIT] close log fd: 5
[EXIT] close listen socket fd: 3
[EXIT] close client socket fd: 4
```

■クライアントプロセスの出力

```
$ ./meibo_client localhost 60000
[ Server Info ] IP Address: [ 127.0.0.1 ] - Port: [ 60000 ]
%C
[ERROR] Server is terminated.
$
```

サーバ側では

- クライアントの接続待ち受け用のソケットのディスクリプタ
- クライアントとの通信用ソケットのディスクリプタ
- クライアントのログファイルのディスクリプタ

が閉じられた出力が，クライアント側ではサーバ側のプロセスが終了したことを示すメッセージが出力されてプロセスが終了することが確認できた．

3.6 Ctrl-C 入力でのクライアント停止時の一時データ記録と復帰処理 (発展課題 3)

第 1.3.3 項で示した，クライアントプログラム停止時の一時的なデータの記録と，そのデータの以降のクライアント接続時の復元について，サーバに対して 1 つのクライアントプログラムで接続し，以下に示す操作で動作を確認した．この動作確認時点では一時記録ファイルは無い状態である．

1. %R コマンドで sample.csv の 2886 個のデータを読み込む
2. %D コマンドで 2880 個のデータを削除，6 個のデータを残す
3. %C コマンドでデータ数を確認する
4. Ctrl-C 入力でクライアントのプロセスを終了させる
5. 再度クライアントプログラムを実行
6. 直前のデータを復元し読み込むことを選択する
7. %C コマンドでデータ数を確認する

操作を行った際のクライアントプログラムの出力を示す．

```
$ ./meibo_client localhost 60000
[ Server Info ] IP Address: [ 127.0.0.1 ] - Port: [ 60000 ]
%R sample.csv
%D 2880
%C
6 profile(s)
Enable to add 9994 data(s)
^C
$ ./meibo_client localhost 60000
[ Server Info ] IP Address: [ 127.0.0.1 ] - Port: [ 60000 ]
Return to the state before error exit ? [y/n] y
%C
6 profile(s)
Enable to add 9994 data(s)
```

仕様の通り，クライアントが Ctrl-C で終了した際に，その時点の名簿データが記録され，以降接続した際に復元するかどうかの選択処理が行われ，データが正しく読み込まれたことが確認できる．

4 プログラムの作成過程に関する考察

今回のプログラムにおいて重要な機能であるメッセージの送受信とその同期処理の実装にあたり，サーバ・クライアントプログラムのどちらもユーザからの入力や処理結果によって長さが大きいメッセージを扱うことを考慮して，バッファーサイズや処理回数の修正が発生すると考えられる必要な送受信回数を共有するという方法ではなく，メッセージに処理の終了を示す目印を加える実装にしたことで，それぞれのプログラムでの送受信に関する処理の構成が共通化でき，結果として修正や変更が容易になった．加えて，この送受信処理を関数を利用しまとまった構造にすることで，基になっている名簿管理プログラムのコードの変更を少なくし，基本的な仕様が実装しやすく，機能を実装を進めていく上でバグの特定が行い易くなるよう工夫を行った．

また，送受信終了の処理も必ず目印となるメッセージを受け取り，その目印により処理を進めるという仕組みにしてプログラム全体を構成したことで，%Q コマンドによるプロセスの終了とメッセージの受信終了の処理や拡張機能の実装と修正が容易になった．

以上の様に通信処理やそれに関するプログラムの流れに自分なりのルールを決めて作業を進めることで実装と拡張の負担は下がったが，それ以外の，特に元の名簿管理プログラムを変更しないことやその通信関連の部分の処理をより簡単な関数の形にすることを意識しすぎた結果，通信に関わる関数などでプログラム全体で共通で使用するソケットなどの変数の多くをグローバル変数に変更してしまい，実装が進みコードが増えるに伴ってそれに関わる関数の処理や変数名の扱いに関する修正が難しくなってしまったことに苦労した．そのため，今回の様に予め実装箇所が多いプログラムを基にした拡張や改良については，既存のコードの修正をしないことを意識し過ぎることをせず，実装したい機能の構成をある程度固めた上で，それに合わせて適度に修正を加えることが必要であると考えられる．また，今回の通信に使用するソケットディスクリプタやメッセージのバッファーなどのプログラム全体的で必要となる変数を扱う関数の実装も，グローバル変数の増加や複雑化を避けるためにも，これら変数を関連させた構造体として扱い関数やプログラムを考えることも上記の苦労点の 1 つの解決のアプローチと言える．

また，通信関連の処理では特にデータの受信とそのデータに対する処理によるエラーが多く発生したため，`recv` 関数の戻り値を用いたエラーのチェックと本来文字として出力されないメッセージの終了に使用している目印を走査して表示・可視化させる処理によるデータ送受信の確認を行い，加えて受信に使用したデータのバッファーをそのまま次の処理に利用するのではなく，一度別のバッファーにコピーしてそれ以降の名簿管理や通信処理の後のデータを比較・表示できるようにして，`Segmentation Fault` のようなメッセージ操作に関連したエラーがどこで発生したものなのかを `printf` による出力や gdb によるデバッグで特定しやすいよう工夫を施した．

4.1 他学生のサーバプログラムとの通信におけるメッセージ仕様の工夫

今回使用させた頂いた他学生のサーバプログラムは，自身で作成したサーバプログラムと同様に繰り返し処理の中でデータの送受信処理を繰り返し，メッセージの最後に終了を示す目印を加えることで，メッセージの送受信とその同期を行う仕様であった．その目印は NULL 文字 (’\0’) であり，自身のクライアントプログラムでは他の値を利用していた箇所をこの NULL 文字に変更することで正しくメッセージのやり取りができるように目印の修正を行った．また，%Q コマンドによるクライアントプログラムの終了について，自身の仕様ではサーバからのメッセージの末尾の値に応じて終了するかどうかを分岐させていたが，相手方のサーバプログラムでは NULL 文字のみのデータをサーバに送信する仕様であったため，ユーザからの入力メッセージで %Q コマンドである場合にクライアント側のプロセスを終了できるように処理を加えた．

4.2 サーバの多重通信受付対応の機能の確認 (発展課題 2)

今回実装するしたサーバの多重通信受付対応の機能は，クライアントとのコネクションの確立後，`fork` 関数によるプロセスの複製を行う．そのため，サーバプログラムの実行後，複数のターミナルからクライアントプログラムを起動し，サーバへ接続した際のサーバ側の複製したプロセス ID やソケットに関する出力と `ps` コマンドによる実行プロセスの情報により確認を行った．`ps` コマンドでは，パイプで `grep` コマンドを繋ぎ，名簿管理プログラムのサーバ・クライアントのプロセスを調べ，サーバは接続要求待ち受りを行っている親プロセスを除いてサーバプロセスの数がクライアントプロセスの数と同じになるか，そしてサーバプログラムで出力される複製時のプロセス ID と一致しているかどうかで多重通信の受付ができているかをテストした．

5　得られた結果に関する考察

今回，クライアントプログラムにおいて，標準入力からの入力データの取得とメッセージの標準出力への出力は文字列操作に関する標準ライブラリ関数でなく，`open` 関数，`write` 関数をそれぞれ利用した．これにより文字列の終端文字もデータとして扱われ，入力データをまとまった文字列という単位で処理が複雑になり，元の名簿管理プログラムで動作確認で行えていたファイルのリダイレクションによるコマンドの処理ができなくなっていた．そのため，動作確認や実際にユーザが使用する際の利便性も考えて，これらシステムコール関数で取得したデータ中で文字列の区切れとなる改行文字や終端文字を起点として，データを文字列単位で分割して配列やリストでまとめて管理できる処理を可能にすることで，より名簿管理に関する操作が改善され，さらなる機能の拡張に繋がると考えられる．