第1回JST CREST「大規模シミュレーション向け基盤ソフトウェアの開発」プロジェクトワークショップ
平成18年12月27日（水），東京大学山上会館

行列計算ライブラリインタフェース SILC によるアプリケーション開発

梶山民人（JST CREST/東京大学）
〈kajiyama@is.s.u-tokyo.ac.jp〉

はじめに

- 行列計算ライブラリインタフェース SILC
  - Simple Interface for Library Collections
  - 特定のライブラリ、計算環境、プログラミング言語に依存しないインタフェース
    - ターゲット：行列計算ライブラリを呼び出す C や Fortran のプログラム
    - 従来のライブラリ呼出しを置き換えるシステム
  - 使いやすさ、便利さが目的

本講演の目的

- SILC の全体像を示す
- 具体的な利用法についての疑問に答える
  - (a) 基本的な使い方：手軽に行列計算を行なうには
  - (b) 既存のコードに関してどのように便利に使えるか
  - (c) SILC に対応していないライブラリや自作のコードを呼び出すには
- できるだけ多くの実例とデータを提供する
  - プログラム例、実験データ、経験談など

発表のあらすじ

- SILC 開発の背景、設計および実装
- ユーザプログラム（SILC アプリケーション）の作成方法
  - 基本的な使い方
  - 既存のユーザプログラムへの SILC の適用方法
  - 新しいライブラリや自作コードの呼出し方法

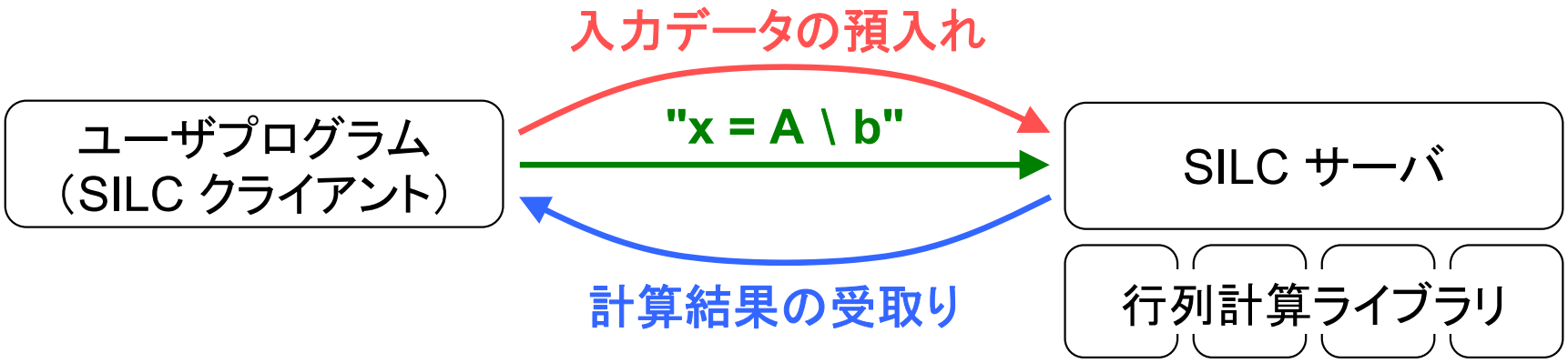
SILC の概要

研究の背景

- 多種多様な行列計算ライブラリ
  - 互換性のないインタフェース（API）
  - 種々の計算環境（共有メモリ並列，分散並列）
    - 特定環境でのみ利用できる専用ライブラリ
- 別のライブラリや計算環境を利用するためにユーザプログラムの修正が必要
  - 解法、行列の格納形式、演算精度の変更
  - 逐次プログラムの並列環境への移植

行列計算ライブラリインタフェースSILC

- **S**imple **I**nterface for **L**ibrary **C**ollections
  - ライブラリ、計算環境、プログラミング言語に依存しないインタフェース
- 次の3ステップで行列計算ライブラリを利用
  1. 入力データの預入れ
  2. 文字列(数式)による計算の指示
  3. 計算結果の受取り



例：連立一次方程式 $Ax = b$ の求解

- ScaLAPACK を利用する従来法のプログラム

```
double *A, *B;
int desc_A[9], desc_B[9], *ipiv, info;
/* 行列 A とベクトル b の作成 */
pdgesv(N, NRHS, A, IA, JA, desc_A, ipiv, B, IB,
  JB, desc_B, &info);
```

- ScaLAPACK を利用する SILC のユーザプログラム

```
silc_envelope_t A, b, x;
/* 行列 A とベクトル b の作成 */
SILC_PUT("A", &A);
SILC_PUT("b", &b);
SILC_EXEC("x = A \\ b"); /* pdgesv の呼出し */
SILC_GET(&x, "x");
```

SILC の特徴

- 用いるライブラリに依存しない
  - 異なるライブラリの解法、前処理、行列の格納形式、演算精度を同じインタフェースで利用できる
- 計算環境に依存しない
  - 逐次、共有メモリ並列、分散並列の各環境をサポート
  - ユーザプログラムの修正なしに他の環境を利用できる
- プログラミング言語に依存しない
  - どの言語でも同じ数式でライブラリを呼出せる
  - ユーザプログラム: C, Fortran, Java, Python で作成可

SILC 構想の全体像


行列計算ライブラリ
Lis
FFTSS
BLAS/LAPACK
ScaLAPACK
行列の格納形式
演算精度
SILC
Simple Interface for Library Collections
データ授受と計算指示の分離
データ入出力
数式による
計算指示
逐次 & 並列 I/O
数式インタプリタ
データ再分散機構
モジュール機構
計算環境
ノート PC
PC クラスタ
ベクトル計算機
大規模分散並列
OpenMP
MPI
ファイル形式
Matrix Market
Harwell-Boeing
ユーザ定義ファイル形式
プログラミング言語
C
Fortran
Java
Python
シミュレーションコードの統合
行列生成
数値シミュレーション
データ可視化

関連研究(1):MATLAB とSILC

- MATLAB
  - 既存のプログラミング言語を置き換える統合開発環境
  - 演算子・関数とライブラリ関数の対応関係が一定
    - \ の解法は行列の構造に応じて一定の選択肢から選ばれる
    - svd, rcond などの関数は LAPACK の呼出しに変換
- SILC
  - 従来のライブラリ利用法に代わるミドルウェア
    - 既存のプログラムの一部のみに SILC の方法を適用できる
  - 演算子・関数とライブラリ関数の対応関係が可変
    - さまざまなライブラリ、計算環境を容易に試せる

関連研究(2):Star-P と SILC

- Star-P
  - MATLAB, Python 等の対話型開発環境にバックエンド並列環境のライブラリをシームレスに組み込むことが目的
    - クライアント側とサーバ側のデータの自動的な授受
    - 演算中に両者のデータを混ぜると結果はサーバ側に置かれる
    - クライアント:逐次、サーバ側のライブラリ:MPI 並列
- SILC
  - C, Fortran 等を対象に、ソースレベルのライブラリ依存性を解消して種々のライブラリ・環境を使いやすくすることに主眼
    - 陽にデータ授受(クライアント側のデータ管理には関与しない)
    - クライアント:逐次, MPI並列、ライブラリ:OpenMP, MPI並列

関連研究(3):Ninf-GとSILC

- Ninf-G
  - グリッド環境下で遠隔手続き呼出し(RPC)を実現するミドルウェア
  - 従来の関数呼出しに似た記法で RPC を記述
  - サーバ側のデータ分散処理をユーザが記述
- SILC
  - 計算指示に数式を利用
  - 数式に基づく高速化の可能性
    - 最適化(例:共通部分式の省略)
    - タスク並列(例:“x1 = A \ b1; x2 = A \ b2”)
  - ユーザがサーバ環境を意識する必要はない

SILC の2種の実装

- OpenMP 版：逐次、共有メモリ並列
  - バージョン 1.2 を公開中
- MPI 版：分散並列
  - 開発中（未公開）
- 対応する計算環境の組合せ

ユーザプログラム（SILC クライアント）	SILC サーバ	対応バージョン
逐次	逐次	OpenMP 版
逐次	共有メモリ並列	OpenMP 版
逐次	分散並列	MPI 版
分散並列	分散並列	MPI 版

主な機能

- データ構造
  - データ型：スカラー、ベクトル、行列、3次元配列
  - 演算精度：整数、実数、複素数（各々単精度または倍精度）
  - 行列の格納形式：密、帯（LAPACK 形式）、疎（CRS, JDS）
- 数式の構成要素
  - 四則演算（+, −, *, /, %）、連立一次方程式の求解（A \ b）
  - 共役転置（A'）、複素共役（A~）
  - 関数、手続き
    - 例：n = sqrt(b' * b)　　ベクトルの2ノルム
  - 添字
    - 例：A[1:5,1:5]　　5×5 の部分行列

モジュール機構

- モジュール
  - ライブラリ依存のコード（ラッパー）をまとめたもの
  - 共有ライブラリとして実現、動的にサーバにリンク
- 演算モジュール
  - 解法ルーチンのラッパーから成るモジュール
  - EXEC リクエストで送られた数式の実行に利用
- 格納形式モジュール
  - 行列の格納形式およびデータ分散方式に依存するコードをまとめたモジュール
  - PUT/GET リクエスト処理に用いる通信ルーチン等を含む

演算モジュール

- 対応する行列計算ライブラリ
  - Lis（反復解法ライブラリ）
  - FFTSS（高速フーリエ変換ライブラリ）
  - BLAS/LAPACK
  - ScaLAPACK
- モジュールの構成
  - モジュール関数（ラッパー）：演算子毎に1つ
  - モジュール関数と演算子を対応付けるテーブル
- 作成方法を後述

格納形式モジュール

- 対応する行列の格納形式
  - 密行列：Fortran 型の2次元配列
  - 帯行列：LAPACK banded storage 形式
  - 疎行列：CRS 形式、JDS 形式
- 対応するデータ分散方式
  - 2次元ブロックサイクリック分割（密）
  - 列ブロック分割（密、帯）
  - 行ブロック分割（疎）

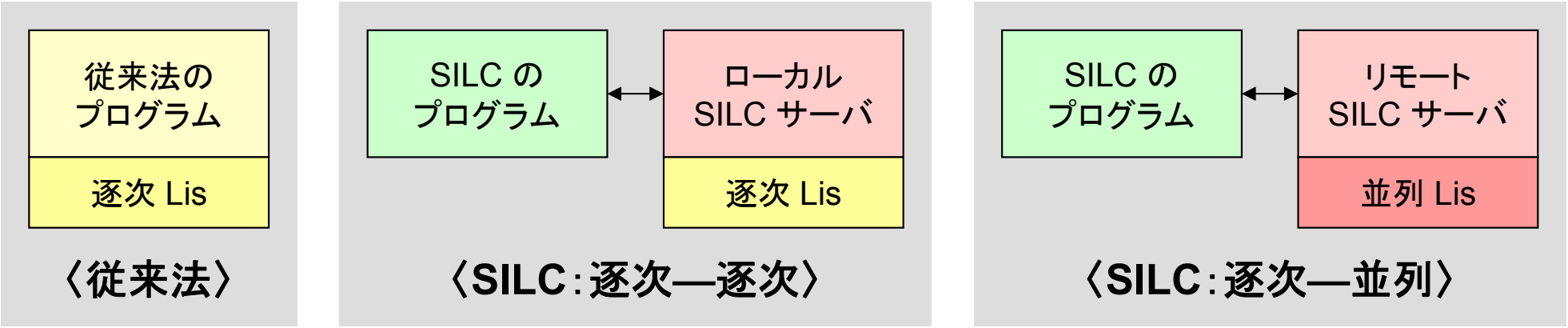
別の解法を利用するには

- 複数の演算モジュールを使い分ける
  - 1つのモジュールにつき1つの解法
  - 利用するモジュールを prefer 文で選択
- 例：LU 分解法と CG 法の比較

```
SILC_EXEC("prefer leq_lu");
SILC_EXEC("x1 = A \\ b"); /* LU 分解法で求解 */
SILC_EXEC("prefer leq_cg");
SILC_EXEC("x2 = A \\ b"); /* CG 法で求解 */
SILC_EXEC("d = b - A * x1");
SILC_EXEC("norm1 = sqrt(d' * d)"); /* ||b − Ax1|| */
SILC_EXEC("d = b - A * x2");
SILC_EXEC("norm2 = sqrt(d' * d)"); /* ||b − Ax2|| */
```

性能評価実験(1)

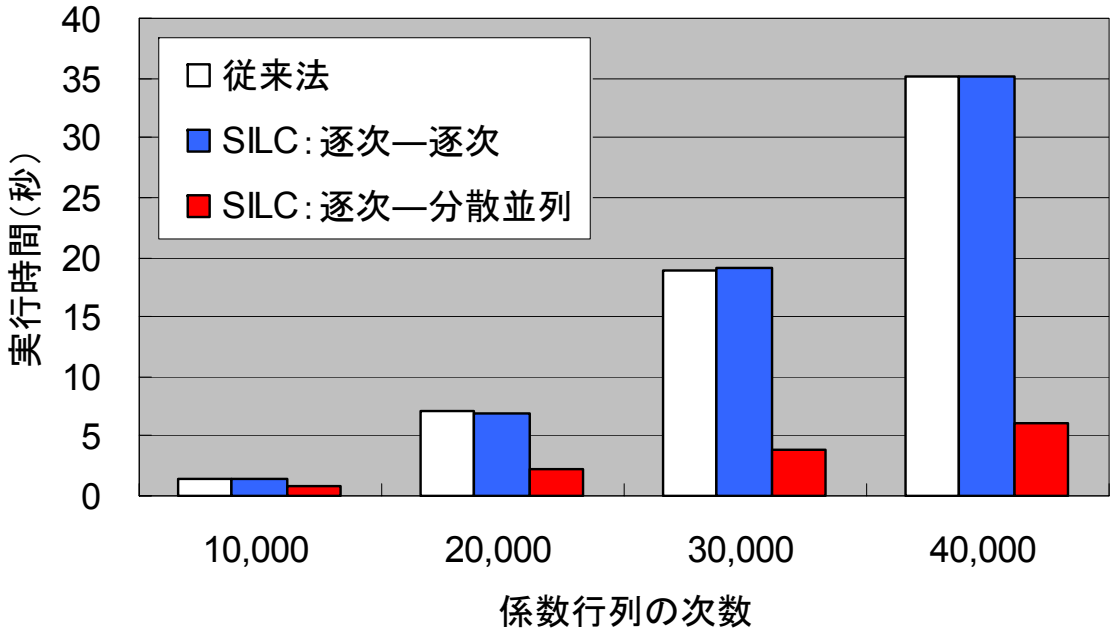
- 従来法および SILC のユーザプログラムの比較
  - 周期境界条件の1次元微分方程式を中心差分で離散化
  - 反復解法ライブラリ Lis の CG 法で求解



  - リモート SILC サーバのみ SGI Altix 3700 上で16プロセスで並列実行
  - その他はすべて同じデスクトップ PC 上で逐次実行

性能評価実験(2)

- 〈従来法〉vs.〈SILC：逐次—逐次〉
  - ローカル SILC サーバとの通信コストは無視できる
- 〈従来法〉vs.〈SILC：逐次—並列〉
  - リモート SILC サーバとの通信コストを上回る1.48倍（次数10,000）～ 5.68倍（40,000）の速度向上



実験環境
- デスクトップ PC：Intel Pentium 4 3.40 GHz, メモリ 1 GB
- SGI Altix 3700：Intel Itanium2 1.3 GHz 32基, ハードウェア分散共有メモリ 32 GB, SGI MPI 4.4
- ネットワーク：ギガビット LAN

ユーザプログラムの作成方法

基本的な作り方

- C, Fortran など従来の言語でユーザプログラムを記述
- 行列計算ライブラリの呼出しを SILC の方法に置き換える
- ユーザプログラムの最小構成
  - 入力データの準備　・・・・・　従来通り
  - ライブラリの呼出し　・・・・・　SILC の方法による
  - 計算結果の利用　・・・・・　従来通り
- 簡単な作例
  - 行列 A の自乗(行列積)を計算する C と Fortran のプログラム

C プログラムの作例（mmul.c）

```c
#include <stdio.h>

#include "client.h"

int main(int argc, char *argv[])
{
  silc_envelope_t object;
  double A[4] = {1.0,2.0,3.0,4.0};
  double X[4];

  SILC_INIT();

  object.v = A;
  object.type = SILC_MATRIX_TYPE;
  object.format = SILC_FORMAT_DENSE;
  object.precision = SILC_DOUBLE;
  object.m = object.n = 2;
  SILC_PUT("A", &object);
  SILC_EXEC("X = A * A");

  object.v = X;
  SILC_GET(&object, "X");

  SILC_FINALIZE();

  printf("A =¥n");
  printf(" %e %e¥n", A[0], A[2]);
  printf(" %e %e¥n", A[1], A[3]);
  printf("A * A =¥n");
  printf(" %e %e¥n", X[0], X[2]);
  printf(" %e %e¥n", X[1], X[3]);
}
```

Fortran プログラムの作例(mmul.f)

```
      PROGRAM MMUL

      INCLUDE 'client.h'

      PARAMETER(N=2)
      REAL*8 A(N,N), X(N,N)
      DATA A /1.0,2.0,3.0,4.0/
      INTEGER*4 IERR

      CALL SILC_INIT(IERR)

      CALL SILC_PUT_MATRIX('A', A,
     &     N, N, SILC_DOUBLE, IERR)

      CALL SILC_EXEC('X = A * A',
     &     IERR)

      CALL SILC_GET_MATRIX('X', X,
     &     IERR)

      CALL SILC_FINALIZE(IERR)

      WRITE(*,*) 'A ='
      WRITE(*,100) A(1,1), A(1,2)
      WRITE(*,100) A(2,1), A(2,2)
      WRITE(*,*) 'X ='
      WRITE(*,100) X(1,1), X(1,2)
      WRITE(*,100) X(2,1), X(2,2)
100   FORMAT(E15.6E3, E15.6E3)

      END
```

Makefile の作例（C の場合）

（1）ヘッダファイルの場所（2ヵ所）

（2）クライアントルーチン（libsilc.a）の場所

（3）追加のライブラリファイル

（4）クライアントルーチンのリンク

```
all: mmul

SILC=       C:/silc-1.2

CC=         gcc
CFLAGS=     -I$(SILC)/test -I$(SILC)                          ····· (1)
LDFLAGS=    -L$(SILC)/test                                   ····· (2)
LIBS=       -lws2_32                                         ····· (3)

mmul: mmul.c
            $(CC) $(CFLAGS) -c mmul.c
            $(CC) $(LDFLAGS) -o $@ mmul.o -lsilc $(LIBS)     ····· (4)
```

Makefile の作例（Fortran の場合）

（1）ヘッダファイルの場所（2ヵ所）

（2）クライアントルーチン（libsilc.a）の場所

（3）追加のライブラリファイル

（4）クライアントルーチンのリンク

```
all: mmul

SILC=       C:/silc-1.2

FC=         g77 -fno-second-underscore
FFLAGS=     -I$(SILC)/test/fortran -I$(SILC)                    ····· (1)
LDFLAGS=    -L$(SILC)/test                                      ····· (2)
LIBS=       -lws2_32                                            ····· (3)

mmul: mmul.f
            $(FC) $(FFLAGS) -c mmul.f
            $(FC) $(LDFLAGS) -o $@ mmul.o -lsilc $(LIBS)        ····· (4)
```

追加のライブラリファイル

- 開発環境によっては通信関連のライブラリファイルの追加が必要
- Windows
  - `LIBS= -lws2_32`
- Solaris
  - `LIBS= -lsocket -lnsl`
- GNU/Linux
  - 特になし
- Mac OS X
  - 特になし

実行方法

- 手順
  （1）サーバの起動
  ```
  > cd ¥silc-1.2
  > server
  ```
  （2）ユーザプログラムの実行
  ```
  > mmul
  ```
- ノート
  - デバッグ情報が標準出力に出力されるため、同じマシンで実行する場合は別のウィンドウで実行するとよい
  - server はカレントディレクトリの modules ディレクトリにあるモジュールを読み込む

実行結果

```
> mmul
connected to kajiyama on port 1639
number of formats = 4
  0: "SILC:dense (column major)"
  1: "SILC:Band"
  2: "SILC:CRS"
  3: "SILC:JDS"
Request: >A
Response: 200 OK
Request: :9:X = A * A
Response: 200 OK
Request: <X
Response: 200 OK
A =
 1.000000e+000 3.000000e+000
 2.000000e+000 4.000000e+000
A * A =
 7.000000e+000 1.500000e+001
 1.000000e+001 2.200000e+001

>
```

クライアントルーチンのデバッグ情報

ユーザプログラムの出力

SILC のユーザプログラム4題

1. SILC の数式による CG 法
2. SILC の数式による2グリッド法
3. 偏微分方程式の初期値問題
4. 特異値分解による画像の圧縮

1. SILC の数式で記述した CG 法

- 反復と条件分岐はユーザプログラムの記述言語で実現
- 行列 *A* の格納形式や精度によらず動作する
  - 複素対称行列なら COCG 法として動作

```
silc_envelope_t object;
double nrm2;

/* A: 係数行列, b: 右辺ベクトル */
SILC_EXEC("rho_old = 1.0");
SILC_EXEC("n = length(b)");
SILC_EXEC("p = zeros(n, 1)");
SILC_EXEC("x = zeros(n, 1)");
SILC_EXEC("r = b");
SILC_EXEC("bnrm2 = 1.0 / norm2(b)");
for (i = 1; i <= maxiter; i++) {
  SILC_EXEC("rho = r' * r");
  SILC_EXEC("beta = rho / rho_old");
  SILC_EXEC("p = r + beta * p");
  SILC_EXEC("q = A * p");
  SILC_EXEC("alpha = rho / (p' * q)");
  SILC_EXEC("r = r - alpha * q");
  SILC_EXEC("nrm2 = norm2(r) * bnrm2");
  SILC_EXEC("x = x + alpha * p");
  /* 収束判定 */
  object.v = &nrm2;
  SILC_GET(&object, "nrm2");
  if (nrm2 <= EPSILON)
    break;
  SILC_EXEC("rho_old = rho");
}
/* x: 解ベクトル */
```

2. 2グリッド法

- スムージング、粗いグリッドでの連立一次方程式の求解に \ 演算子を使用
- split(*A*, *L*, *D*, *U*)

  行列 *A* の下三角部分 *L*、対角部分 *D*、上三角部分 *U* を得る

```
silc_envelope_t object;
double nrm2;

/* A: 係数行列, b: 右辺, R: restriction operator,
    P: prolongation operator */
SILC_EXEC("split(A, L, D, U)");
SILC_EXEC("LD = L + D");
SILC_EXEC("DU = D + U");
SILC_EXEC("Ac = R * A * R'"); /* coarse grid */
SILC_EXEC("bnrm2 = 1.0 / norm2(b)");
for (i = 1; i <= maxiter; i++) {
  /* pre-smoothing on the fine grid */
  SILC_EXEC("x = LD \\ (b - U * x)");
  /* restrict the residual on the coarse grid */
  SILC_EXEC("rc = R * (b - A * x)");
  /* solve Ac uc = rc on the coarse grid */
  SILC_EXEC("uc = Ac \\ rc");
  /* update the solution on the fine grid */
  SILC_EXEC("x += P * uc");
  /* post-smoothing on the fine grid */
  SILC_EXEC("x = DU \\ (b - L * x)");
  /* 収束判定 */
  SILC_EXEC("nrm2 = norm2(b - A * x) * bnrm2");
  object.v = &nrm2;
  SILC_GET(&object, "nrm2");
  if (nrm2 <= EPSILON)
    break;
}
/* x: 解ベクトル */
```

3. 偏微分方程式の初期値問題

- 次の1次元拡散方程式をクランク･ニコルソン法で解く

$$\frac{\partial \varphi}{\partial t} = \frac{\partial^2 \varphi}{\partial x^2} \quad (t \geq 0, 0 \leq x \leq \pi)$$

初期条件 : $\varphi = \sin x \, (t = 0, 0 \leq x \leq \pi)$

境界条件 : $\varphi = 0 \, (t > 0, x = 0), \varphi = 0 \, (t > 0, x = \pi)$

  - タイムステップ毎に連立一次方程式 $A\boldsymbol{x}_k = \boldsymbol{b}$ を解く
    - A は三重対角行列
  - 第 $k-1$ 時刻の解 $\boldsymbol{x}_{k-1}$ から右辺 $\boldsymbol{b}$ を得る
    - 三重対角行列 C を設けて $\boldsymbol{b} = C\boldsymbol{x}_{k-1}$ を計算

3. 偏微分方程式の初期値問題(結果)

- ユーザプログラムの構成

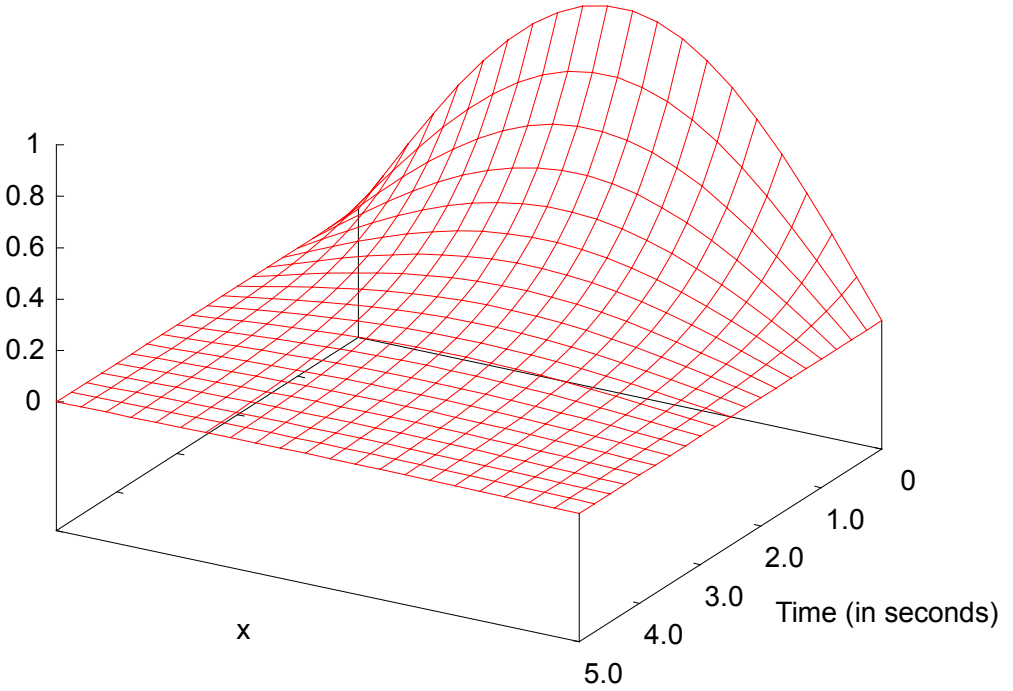
行列 A, C とベクトル x_0 を作成

```
SILC_PUT("A", &A);
SILC_PUT("C", &C);
SILC_PUT("x", &x); /* x0 */
```

時刻 t_k ($k = 1, 2, 3, \ldots$) について {

```
    SILC_EXEC("b = C * x");
    SILC_EXEC("x = A \\ b");
    SILC_GET(&x, "x"); /* xk */
}
```

- 計算結果



4. 画像圧縮

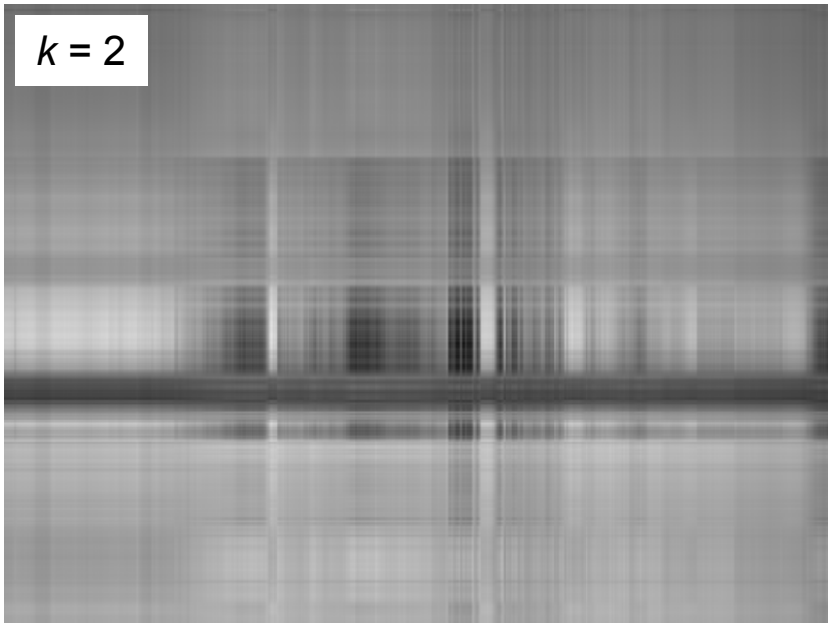
- Python によるユーザプログラムの例
- svd(*X*, *U*, *S*, *V*)

  行列 *X* の特異値分解を求める（$X = USV^H$、*S* の対角要素は X の特異値）
- 添字の利用

  U[:,:k]　*k*列目までを得る

```
from silc import *

def compress(filename, new_filename, k):
  m, n, X = load(filename) # 画像を行列として読み込む

  object = Envelope() # C 版の silc_envelope_t に相当

  object.v = X
  object.type = SILC_MATRIX_TYPE
  object.format = SILC_FORMAT_DENSE
  object.precision = SILC_INT
  object.m = m
  object.n = n
  SILC_PUT("X", object)

  SILC_EXEC("svd(X, U, S, V)") # 特異値分解

  object.v = k
  object.type = SILC_SCALAR_TYPE
  object.precision = SILC_INT
  SILC_PUT("k", object)

  # 画像を圧縮
  SILC_EXEC("Y = U[:,:k] * S[:k,:k] * V[:,:k]'")

  SILC_GET(object, "Y")
  Y = object.v

  save(new_filename, m, n, Y) # 行列を画像として保存
```

4. 画像圧縮(結果)

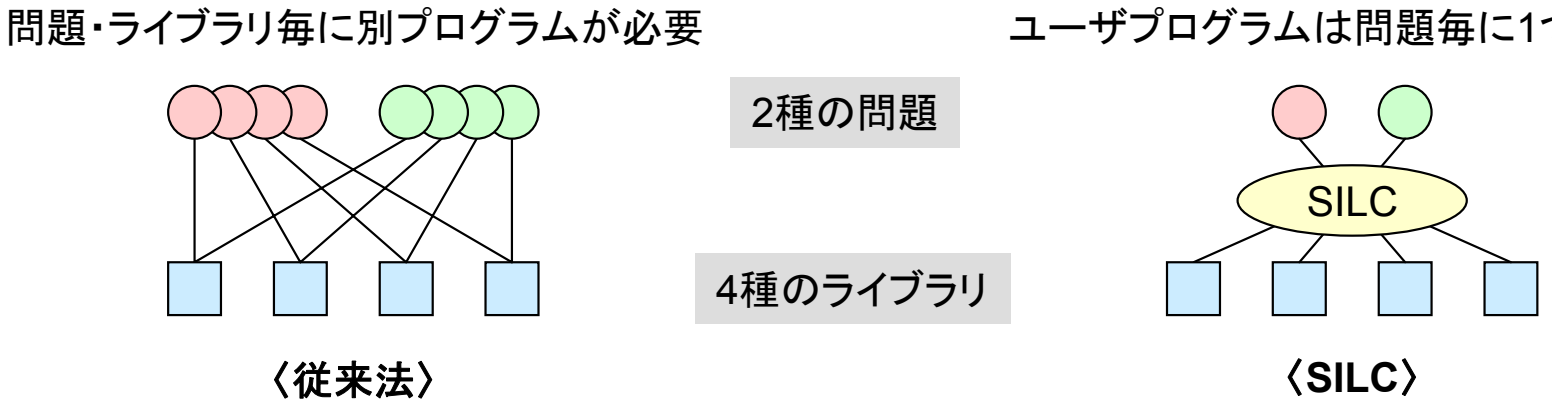


既存コードへの SILC の適用

- アプローチ(1)
  - 主要な行列計算のみを SILC の方法で置き換える
    - 部分的に少しずつ適用可、手間がかからない
    - クライアント側のメモリ量の制約を受けやすい
- アプローチ(2)
  - すべてのデータをサーバ側に移す
  - すべての計算を SILC の数式で行なう
    - ユーザプログラムは制御のみ行なう
    - クライアント側のメモリ量の制約を受けにくい

ライブラリ・自作コードの登録(1)

- どういう場合に登録が必要か
  - 既存の(登録済みの)ライブラリの代わりに利用したい場合
    - 異なる解法(例:連立一次方程式の直接解法と反復解法)
    - 特定の計算環境向けに最適化されたライブラリ
- SILC を通じて利用するメリット
  - 従来法:*M* 個のライブラリ、*N* 個の問題に対して *MN* 個のユーザプログラムが必要
  - SILC:登録ライブラリ *M* 個、ユーザプログラム *N* 個で済む



ライブラリ・自作コードの登録(2)

- 登録方法
  - 演算モジュールにラッパーを追加する
  - 呼出し方法を決める
    - 単項演算子、二項演算子、関数、または手続き
    - ラッパーとの対応関係をモジュール内のテーブルに記述
- 例:BLAS のベクトル内積ルーチンの場合
  - 二項演算子または関数 dot として登録

ラッパーの作例

```
int blasmodule_dot_product(int n_params, silc_mfunc_param_t *param)
{
  silc_object_t *x, *y;

  x = param[0].o;
  y = param[1].o;
  switch (x->precision) {
  case SILC_FLOAT:
    param[2].o = silc_object_new_float(call_sdot(x->d.vector.length,
                                                 x->d.vector.v.s, 1,
                                                 y->d.vector.v.s, 1));
    break;
  case SILC_DOUBLE:
    param[2].o = silc_object_new_double(call_ddot(x->d.vector.length,
                                                  x->d.vector.v.d, 1,
                                                  y->d.vector.v.d, 1));
    break;
  default:
    should_not_reach_here;
  }
  return (param[2].o) ? 0 : -1;
}
```

呼出し方法の決定

- 二項演算子として登録する場合
  - 用例：d = x' * y
    param[0].o　演算子の左辺値
    param[1].o　演算子の右辺値
  - 演算結果を param[2].oに格納する
- 関数として登録する場合
  - 用例：d = dot(x, y)
    param[0].o　第1引数
    param[1].o　第2引数
  - 関数の戻り値を param[2].o に格納する

テーブルの作例

```
static binary_op_table_t _binary_op_table[] = { /* 二項演算子のテーブル */
    :
  {blasmodule_dot_product, "blasmodule_dot_product",
   SILC_MULTIPLY_OP,              /* 二項演算子の種類 */
   SILC_FLOAT|SILC_DOUBLE,        /* 受け付けるデータ精度 */
   SILC_ROW_VECTOR_TYPE, 0,       /* 左辺値のデータ型 */
   SILC_COLUMN_VECTOR_TYPE, 0, /* 右辺値のデータ型 */
   2, /* 0:左辺値を書き換える, 1:右辺値を書き換える, 2:新しいオブジェクトを返す */
   0},
    :
  {NULL} /* sentinel */
};

static function_table_t _function_table[] = { /* 関数のテーブル */
    :
  {blasmodule_dot_product, "blasmodule_dot_product",
   "scalar dot(vector, vector)", /* 関数名、引数、戻り値をC言語風に記述 */
   0},
    :
  {NULL} /* sentinel */
};
```

演算モジュールの初期化

- サーバは起動時にモジュールの初期化関数を呼ぶ
- この初期化関数の中で silc_register_module 関数を用いてモジュール名とテーブルをサーバに渡す
  - モジュール名：prefer 文の処理に利用
  - テーブル：数式の実行（ラッパーの検索）に利用

```
void init_blasmodule() /* 初期化関数名は “init_” + モジュール名でなければならない */
{
  silc_register_module("blasmodule",        /* モジュール名 */
                       _unary_op_table,     /* 単項演算子のテーブル */
                       _binary_op_table,    /* 二項演算子のテーブル */
                       _function_table,     /* 関数のテーブル */
                       _procedure_table); /* 手続きのテーブル */
}
```

SILC v1.2 の主なソースファイル（1）

ファイル名	ハードウェア, OS	コンパイラ	OpenMP
Makefile.ssifs	Sun Fire 3800, Solaris 9 (sparc)	Sun ONE Studio 7	Yes
Makefile.altix	SGI Altix 3700, Red Hat Linux Advanced Server 2.1	Intel C 8.1 Intel Fortran 8.1	Yes
Makefile.ssixc0	IBM eServer xSeries 335, Red Hat Linux 8.0	Intel C 9.0 Intel Fortran 9.0	Yes
Makefile.ibn1	Dell PowerEdge SC 1420, Fedora Core 4	Intel C 9.0 (EM64T) Intel Fortran 9.0 (EM64T)	Yes
Makefile.openpower	IBM OpenPower 710, SuSE Linux Enterprise Server 9 (ppc)	IBM XL C 7.0 IBM XL Fortran 9.1	Yes
Makefile.g5	Apple PowerMac G5, Mac OS X 10.4.2	IBM XL C 6.0 IBM XL Fortran 8.1	Yes
Makefile.sx	NEC SX-6i, SUPER-UX 13.1 SX-6	C++/SX 1.0 for SX-6 FORTRAN90/SX 2.0 for SX-6	No
Makefile.gcc	Panasonic CF-R3, Fedora Core 3	GCC 3.4.2	No
Makefile.mingw	Dell Dimension 84000, Microsoft Windows XP Pro SP2	MinGW (GCC 3.2.3)	No

SILC v1.2 の主なソースファイル(2)

- ヘッダファイル
  - object.h　行列のデータ構造
  - module.h　モジュールのデータ構造
- サーバ本体
  - server.c　メインプログラム
  - cmd_lexer.l　数式の字句解析器
  - cmd_parse.y　数式の構文解析器
  - interp.c　数式インタプリタ
  - net.c　低レベル通信ルーチン
- 演算モジュール
  - coremodule.c　大半の演算子に対応
  - blasmodule.c　BLAS/LAPACK
  - leq_cg.c　CG 法(libleq)
  - leq_gs.c　ガウス・ザイデル法(libleq)
  - leq_lu.c　LU 分解法
  - leq_lis.c　Lis
  - fftss.c　FFTSS
- 格納形式モジュール兼演算モジュール
  - dense.h　密行列
  - dense_alloc.c　〃
  - dense_format.c　〃
  - dense_op.c　〃
  - sparse_band.h　帯行列
  - sparse_band_format.c　〃
  - sparse_band_op.c　〃
  - sparse_crs.h　疎行列(CRS 形式)
  - sparse_crs_alloc.c　〃
  - sparse_crs_format.c　〃
  - sparse_crs_op.c　〃
  - sparse_jds.h　疎行列(JDS 形式)
  - sparse_jds_format.c　〃
  - sparse_jds_op.c　〃

SILC v1.2 の主なソースファイル（3）

- クライアントルーチン
  - test/client.c　　libsilc.a のソース
  - test/client.h　　C 用ヘッダ
  - test/fortran/client.h　　Fortran 用ヘッダ
- Matrix Market 形式（.mtx）の行列データファイルの読込み
  - test/mm/mm_read.c
  - test/mm/mm_read.h
  - test/mm/mm_read_band.c
  - test/mm/mm_read_crs.c
- 上記2点の Python 版
  - test/python/silc.py
  - test/python/mm.py
- 計時ルーチン（UNIX/Linux, Win32 両用）
  - wtime.c
  - wtime.h

まとめ(1)

- SILC によるアプリケーション開発のメリット
  - 少ない手間で実現できる
    - 種々のライブラリを同じ数式で簡単に呼び出せる
    - 既存の入出力コードを活かせる
  - どの計算環境でも修正なしで実行できる
    - 逐次環境で作成、テストして並列環境で実行といった開発スタイルが容易になる
    - 並列計算のメリットを自動的に享受できる

まとめ(2)

- 今後の課題
  - 実装について
    - 対応ライブラリの拡充
    - MPI 版 SILC の開発
  - 応用について
    - 現実味のある問題への SILC の適用
- SILC ホームページ

  http://ssi.is.s.u-tokyo.ac.jp/silc/

付録：Microsoft Visual Studio .NET 2003 によるユーザプログラムのコンパイル

主なポイント

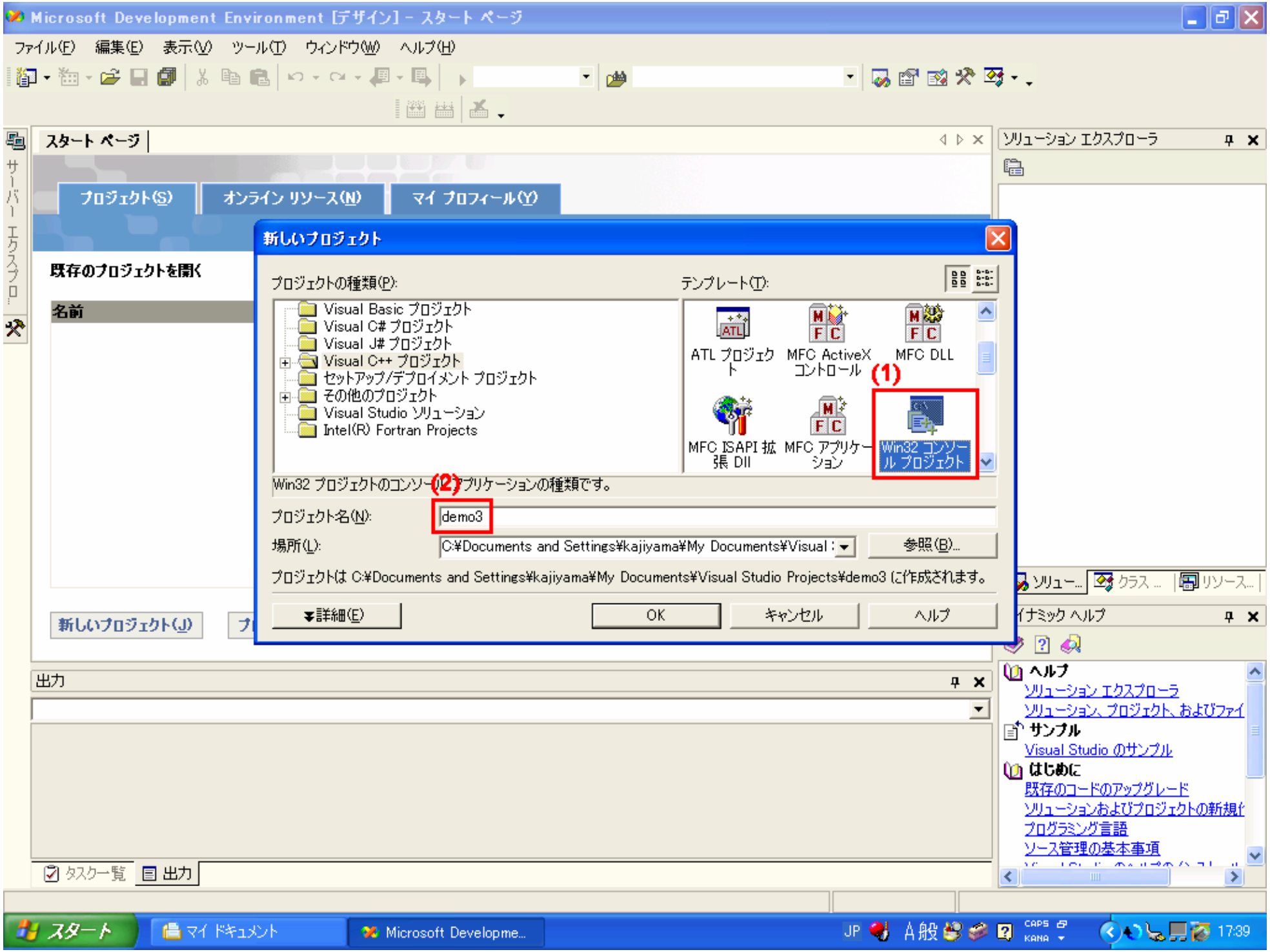
- 以下のファイル（クライアントルーチン）をプロジェクトに追加
  - silc-1.2¥test¥client.c
  - silc-1.2¥test¥client.h
  - silc-1.2¥clist.h
  - silc-1.2¥const.h
  - silc-1.2¥object.h
- 以下のプロパティを設定
  - 〈構成プロパティ〉→〈リンカ〉→〈入力〉
    - 〈追加の依存ファイル〉に ws2_32.lib を追加
  - 〈構成プロパティ〉→〈C/C++〉→〈プリコンパイル済みヘッダー〉
    - 〈プリコンパイル済みヘッダーの作成/使用〉を〈プリコンパイル済みヘッダーを使用しない〉に設定

作成例

- SILC v1.2 に付属するサンプルプログラム demo3.c をコンパイルする場合
- このプログラムは以下のファイルから成る
  - `silc-1.2¥test¥demo3.c`
  - `silc-1.2¥wtime.c`
  - `silc-1.2¥wtime.h`

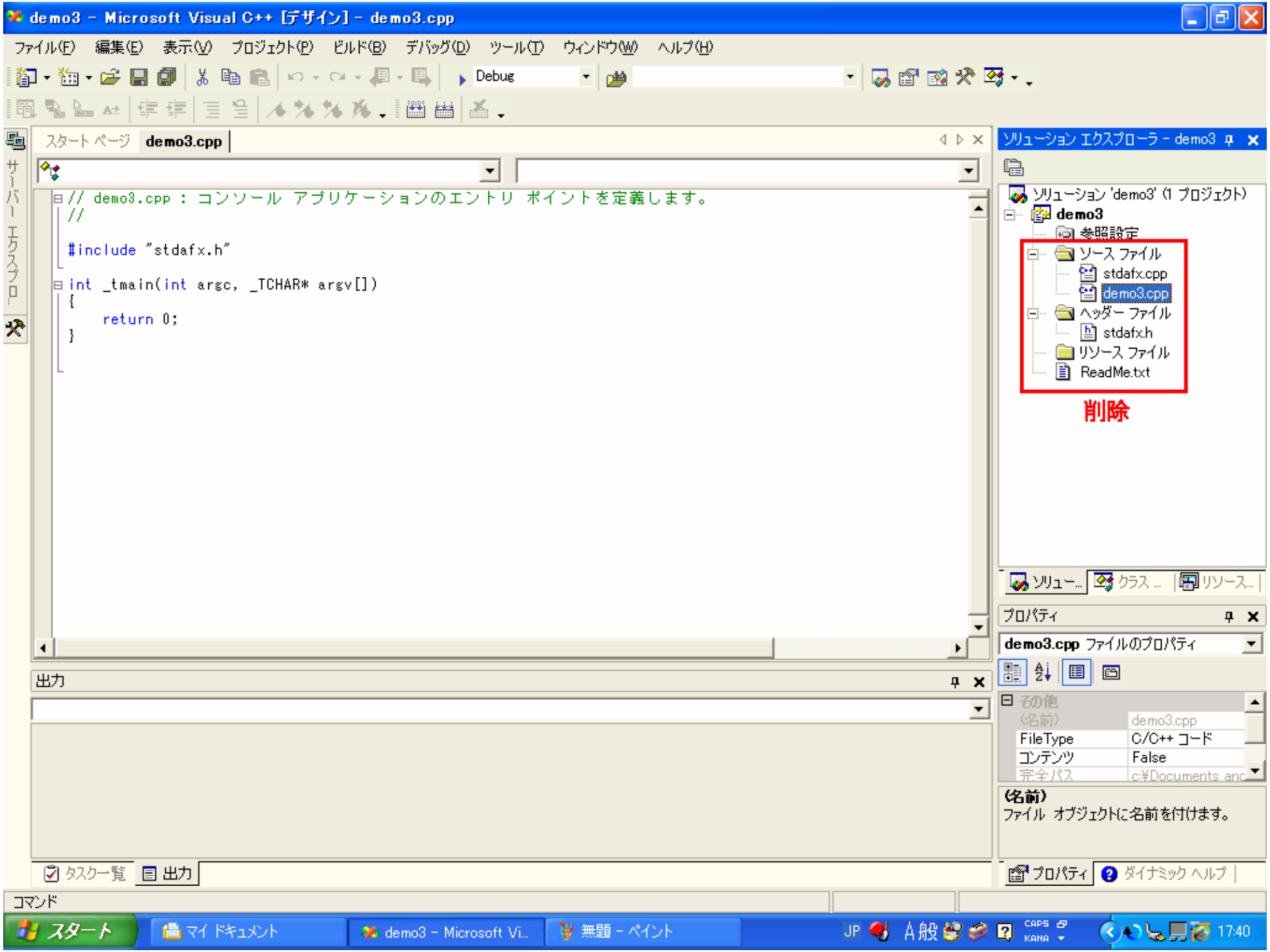
ステップ（1）

- 新しいプロジェクトを作成する
  - 〈Visual C++ プロジェクト〉→〈Win32 コンソールプロジェクト〉を選択
  - プロジェクト名：demo3


Microsoft Development Environment [デザイン] - スタート ページ
ファイル(F) 編集(E) 表示(V) ツール(T) ウィンドウ(W) ヘルプ(H)
スタート ページ
プロジェクト(S) オンライン リソース(N) マイ プロフィール(Y)
既存のプロジェクトを開く
名前
新しいプロジェクト(J)
新しいプロジェクト
プロジェクトの種類(P):
Visual Basic プロジェクト
Visual C# プロジェクト
Visual J# プロジェクト
Visual C++ プロジェクト
セットアップ/デプロイメント プロジェクト
その他のプロジェクト
Visual Studio ソリューション
Intel(R) Fortran Projects
テンプレート(T):
ATL プロジェクト
MFC ActiveX コントロール
MFC DLL
MFC ISAPI 拡張 Dll
MFC アプリケーション
Win32 コンソール プロジェクト
(1)
(2)
Win32 プロジェクトのコンソール アプリケーションの種類です。
プロジェクト名(N): demo3
場所(L): C:¥Documents and Settings¥kajiyama¥My Documents¥Visual
参照(B)...
プロジェクトは C:¥Documents and Settings¥kajiyama¥My Documents¥Visual Studio Projects¥demo3 に作成されます。
詳細(E)
OK
キャンセル
ヘルプ
ソリューション エクスプローラ
出力
タスク一覧
出力
ヘルプ
ソリューション エクスプローラ
サンプル
Visual Studio のサンプル
はじめに
既存のコードのアップグレード
プログラミング言語
ソース管理の基本事項
スタート
マイ ドキュメント
Microsoft Developme...
17:39

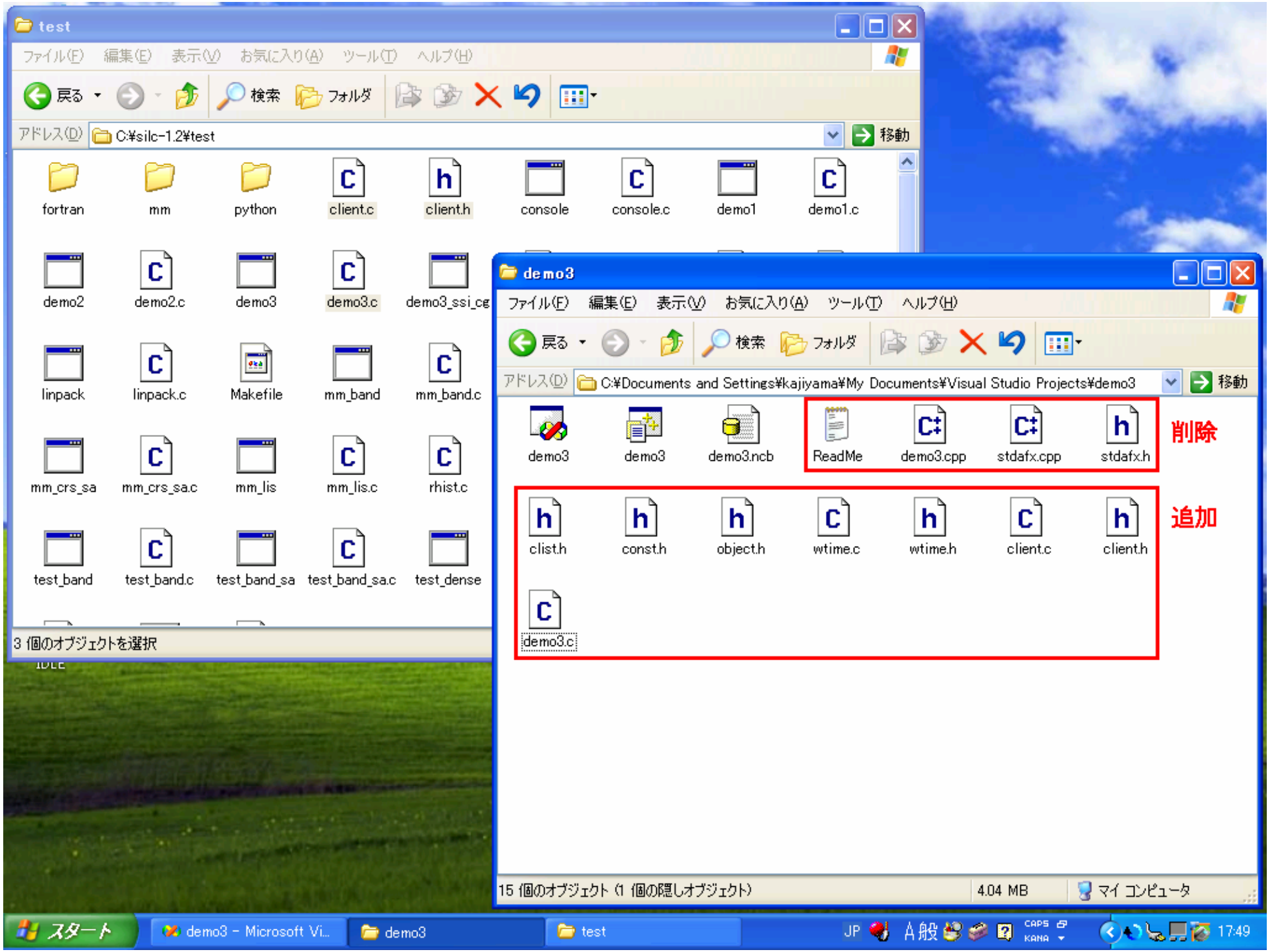
ステップ(2)

- 自動的に作成される以下のファイルをフォルダごと削除する
  - ソースファイル¥demo3.cpp
  - ソースファイル¥stdafx.cpp
  - ヘッダファイル¥stdafx.h
  - ReadMe.txt
- 注:この操作ではファイルの実体は削除されない(次のステップで削除可)


demo3 - Microsoft Visual C++ [デザイン] - demo3.cpp
ファイル(F) 編集(E) 表示(V) プロジェクト(P) ビルド(B) デバッグ(D) ツール(T) ウィンドウ(W) ヘルプ(H)
Debug
スタート ページ
demo3.cpp
// demo3.cpp : コンソール アプリケーションのエントリ ポイントを定義します。
//
#include "stdafx.h"
int _tmain(int argc, _TCHAR* argv[])
{
return 0;
}
ソリューション エクスプローラ - demo3
ソリューション 'demo3' (1 プロジェクト)
demo3
参照設定
ソース ファイル
stdafx.cpp
demo3.cpp
ヘッダー ファイル
stdafx.h
リソース ファイル
ReadMe.txt
削除
プロパティ
demo3.cpp ファイルのプロパティ
その他
(名前) demo3.cpp
FileType C/C++ コード
コンテンツ False
(名前)
ファイル オブジェクトに名前を付けます。
出力
タスク一覧
出力
プロパティ
ダイナミック ヘルプ
コマンド
スタート
マイ ドキュメント
demo3 - Microsoft Vi...
無題 - ペイント
17:40

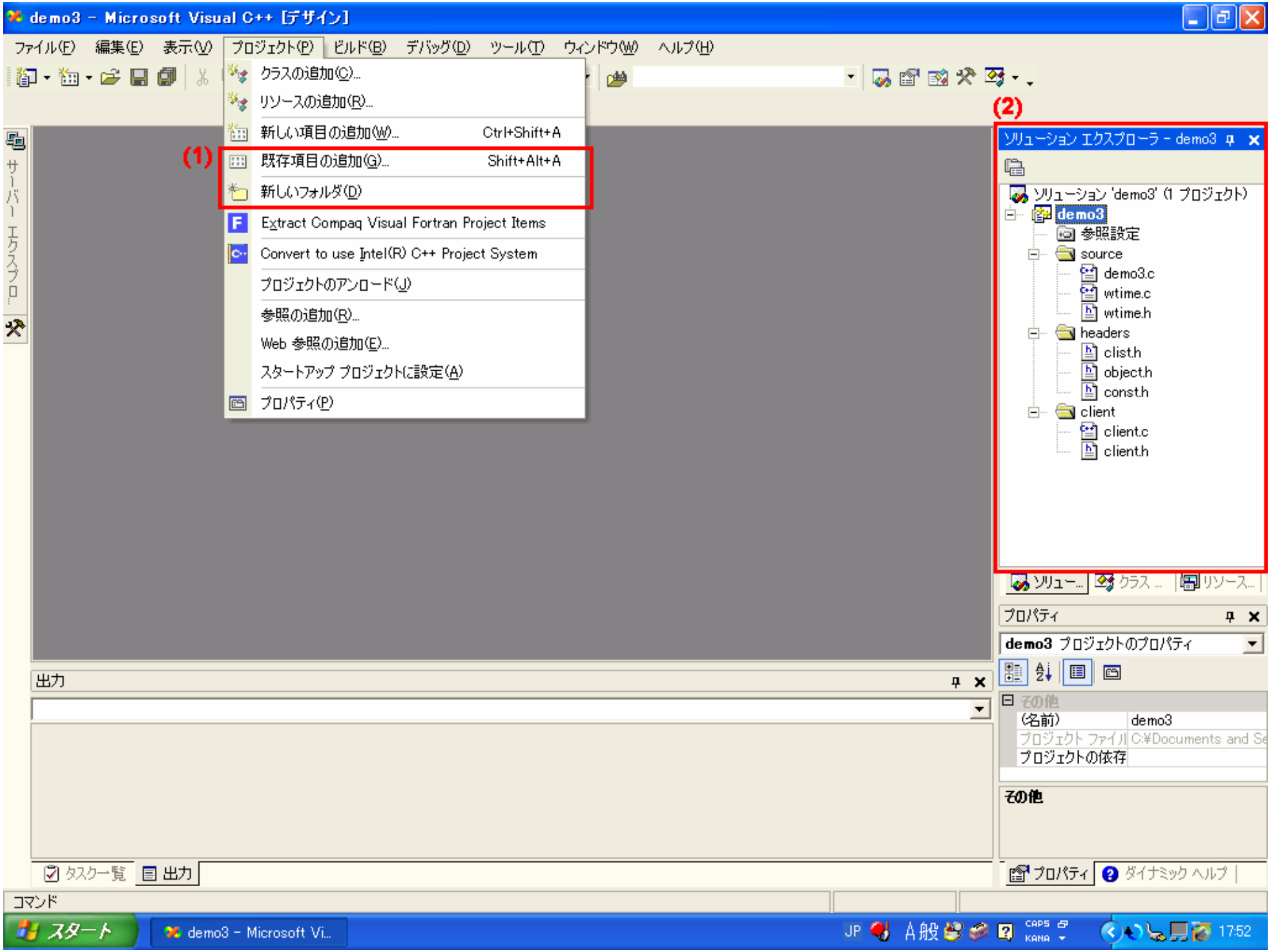
ステップ(3)

- 以下のファイルを〈マイドキュメント〉配下の Visual Studio Project¥demo3 にコピー
  - ユーザプログラム(サンプルプログラム demo3 の場合)
    - silc-1.2¥test¥demo3.c
    - silc-1.2¥wtime.c
    - silc-1.2¥wtime.h
  - クライアントルーチン(共通)
    - silc-1.2¥test¥client.c
    - silc-1.2¥test¥client.h
    - silc-1.2¥clist.h
    - silc-1.2¥const.h
    - silc-1.2¥object.h


test
ファイル(F) 編集(E) 表示(V) お気に入り(A) ツール(T) ヘルプ(H)
戻る 検索 フォルダ
アドレス(D) C:¥silc-1.2¥test
移動
fortran
mm
python
client.c
client.h
console
console.c
demo1
demo1.c
demo2
demo2.c
demo3
demo3.c
demo3_ssi_cg
linpack
linpack.c
Makefile
mm_band
mm_band.c
mm_crs_sa
mm_crs_sa.c
mm_lis
mm_lis.c
rhist.c
test_band
test_band.c
test_band_sa
test_band_sa.c
test_dense
3 個のオブジェクトを選択
demo3
ファイル(F) 編集(E) 表示(V) お気に入り(A) ツール(T) ヘルプ(H)
戻る 検索 フォルダ
アドレス(D) C:¥Documents and Settings¥kajiyama¥My Documents¥Visual Studio Projects¥demo3
移動
demo3
demo3
demo3.ncb
ReadMe
demo3.cpp
stdafx.cpp
stdafx.h
削除
clist.h
const.h
object.h
wtime.c
wtime.h
client.c
client.h
demo3.c
追加
15 個のオブジェクト (1 個の隠しオブジェクト)
4.04 MB
マイ コンピュータ
スタート
demo3 - Microsoft Vi...
demo3
test
JP
A般
CAPS
KANA
17:49

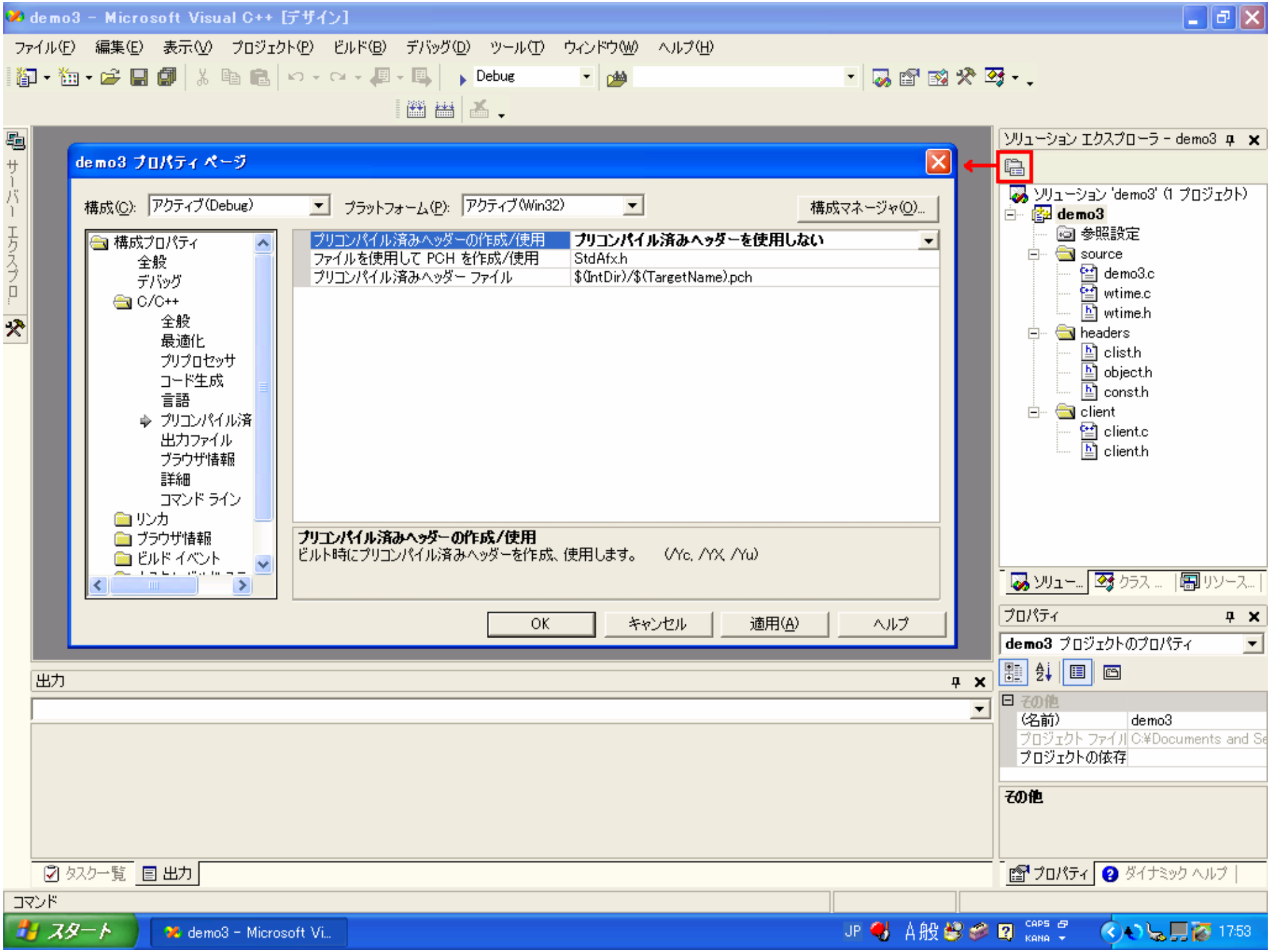
ステップ（4）

- プロジェクトにクライアントルーチンとユーザプログラムのソースファイルを追加する
  - 〈プロジェクト〉→〈既存項目の追加〉でファイルを追加
  - 〈プロジェクト〉→〈新しいフォルダ〉で新しいフォルダを追加
    - フォルダの作成は必須ではないが、図ではフォルダの構成例を示してある


demo3 - Microsoft Visual C++ [デザイン]
ファイル(F) 編集(E) 表示(V) プロジェクト(P) ビルド(B) デバッグ(D) ツール(T) ウィンドウ(W) ヘルプ(H)
(1)
クラスの追加(C)...
リソースの追加(R)...
新しい項目の追加(W)... Ctrl+Shift+A
既存項目の追加(G)... Shift+Alt+A
新しいフォルダ(D)
Extract Compaq Visual Fortran Project Items
Convert to use Intel(R) C++ Project System
プロジェクトのアンロード(J)
参照の追加(R)...
Web 参照の追加(E)...
スタートアップ プロジェクトに設定(A)
プロパティ(P)
(2)
ソリューション エクスプローラ - demo3
ソリューション 'demo3' (1 プロジェクト)
demo3
参照設定
source
demo3.c
wtime.c
wtime.h
headers
clist.h
object.h
const.h
client
client.c
client.h
プロパティ
demo3 プロジェクトのプロパティ
(名前) demo3
プロジェクトの依存
その他
出力
タスク一覧
出力
コマンド
スタート
demo3 - Microsoft Vi...
17:52

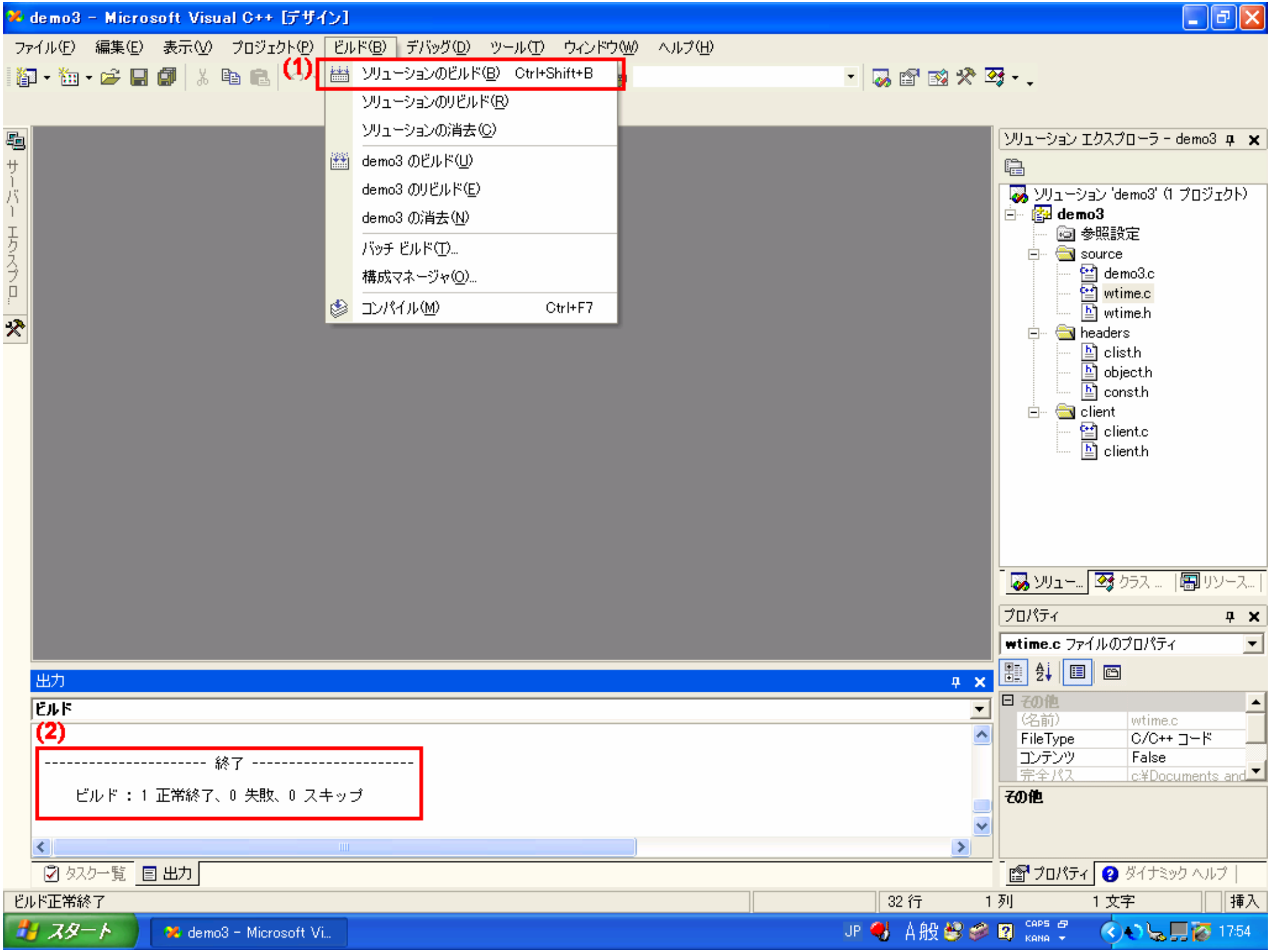
ステップ(5)

- 〈プロジェクト〉→〈プロパティ〉を選択(または図のボタンをクリック)して以下のプロパティを設定する
  - demo3 のプロパティ
    - 〈構成プロパティ〉→〈リンカ〉→〈入力〉
      - 〈追加の依存ファイル〉に ws2_32.lib を追加
    - 〈構成プロパティ〉→〈C/C++〉→〈プリコンパイル済みヘッダー〉
      - 〈プリコンパイル済みヘッダーの作成/使用〉を〈プリコンパイル済みヘッダーを使用しない〉に設定
    - 〈構成プロパティ〉→〈C/C++〉→〈プロプロセッサ〉
      - 〈プロセッサの定義〉に DEBUG を追加
  - wtime.c のプロパティ
    - 〈構成プロパティ〉→〈C/C++〉→〈プリプロセッサ〉
      - 〈プロセッサの定義〉に _TEST を追加


demo3 - Microsoft Visual C++ [デザイン]
ファイル(F) 編集(E) 表示(V) プロジェクト(P) ビルド(B) デバッグ(D) ツール(T) ウィンドウ(W) ヘルプ(H)
Debug
demo3 プロパティ ページ
構成(C): アクティブ(Debug)
プラットフォーム(P): アクティブ(Win32)
構成マネージャ(O)...
構成プロパティ
全般
デバッグ
C/C++
全般
最適化
プリプロセッサ
コード生成
言語
プリコンパイル済
出力ファイル
ブラウザ情報
詳細
コマンド ライン
リンカ
ブラウザ情報
ビルド イベント
プリコンパイル済みヘッダーの作成/使用
プリコンパイル済みヘッダーを使用しない
ファイルを使用して PCH を作成/使用
StdAfx.h
プリコンパイル済みヘッダー ファイル
$(IntDir)/$(TargetName).pch
プリコンパイル済みヘッダーの作成/使用
ビルド時にプリコンパイル済みヘッダーを作成、使用します。 (/Yc, /YX, /Yu)
OK
キャンセル
適用(A)
ヘルプ
ソリューション エクスプローラ - demo3
ソリューション 'demo3' (1 プロジェクト)
demo3
参照設定
source
demo3.c
wtime.c
wtime.h
headers
clist.h
object.h
const.h
client
client.c
client.h
プロパティ
demo3 プロジェクトのプロパティ
(名前) demo3
プロジェクトの依存
その他
出力
タスク一覧
出力
コマンド
スタート
demo3 - Microsoft Vi...
17:53

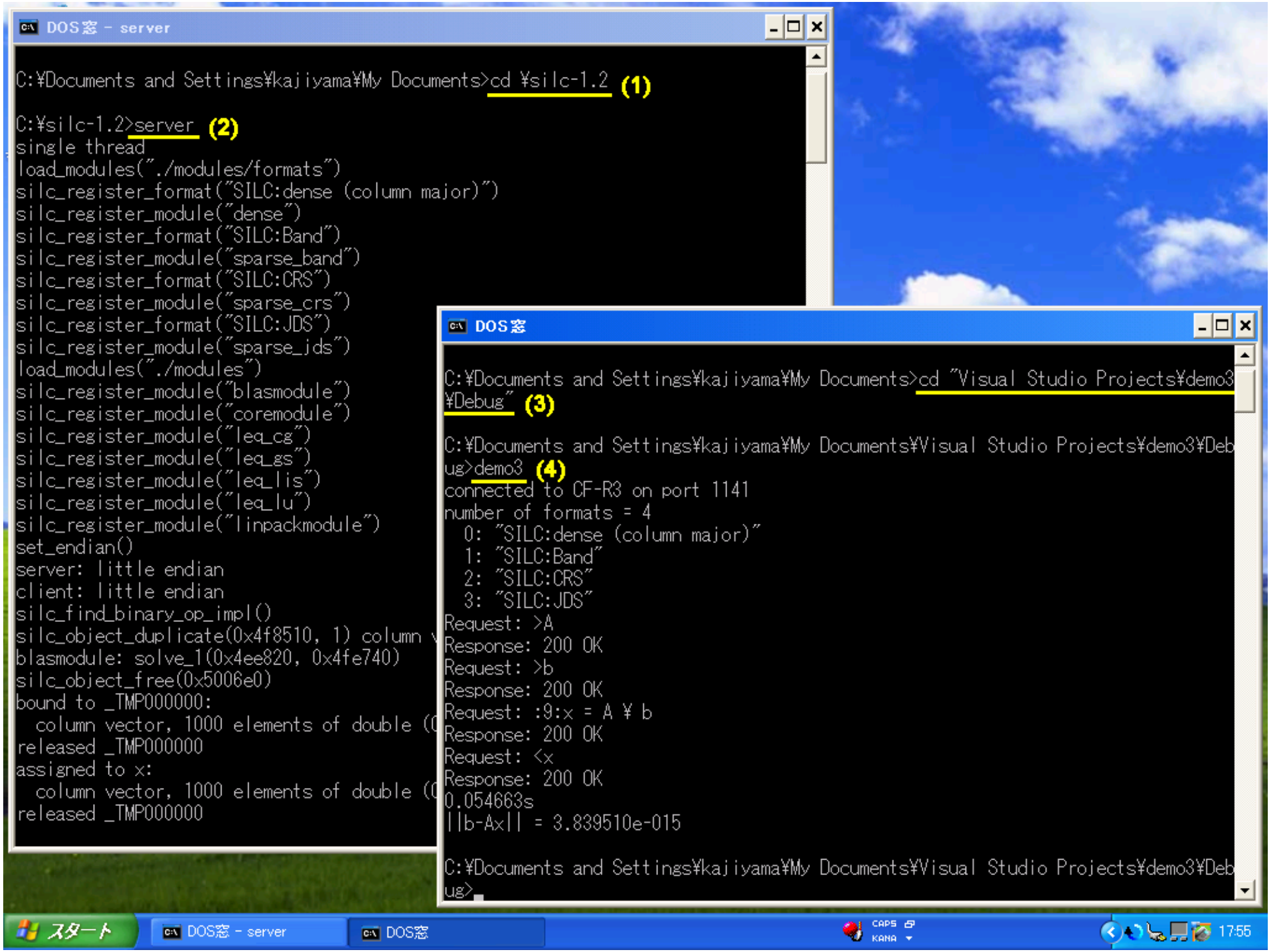
ステップ(6)

- 〈ビルド〉→〈ソリューションのビルド〉を選択してコンパイルとリンクを行なう
  - Visual Studio Project¥demo3¥Debug に demo3.exe が作成される


demo3 - Microsoft Visual C++ [デザイン]
ファイル(F) 編集(E) 表示(V) プロジェクト(P) ビルド(B) デバッグ(D) ツール(T) ウィンドウ(W) ヘルプ(H)
(1)
ソリューションのビルド(B) Ctrl+Shift+B
ソリューションのリビルド(R)
ソリューションの消去(C)
demo3 のビルド(U)
demo3 のリビルド(E)
demo3 の消去(N)
バッチ ビルド(T)...
構成マネージャ(O)...
コンパイル(M) Ctrl+F7
ソリューション エクスプローラ - demo3
ソリューション 'demo3' (1 プロジェクト)
demo3
参照設定
source
demo3.c
wtime.c
wtime.h
headers
clist.h
object.h
const.h
client
client.c
client.h
プロパティ
wtime.c ファイルのプロパティ
その他
(名前) wtime.c
FileType C/C++ コード
コンテンツ False
出力
ビルド
(2)
---------- 終了 ----------
ビルド : 1 正常終了、0 失敗、0 スキップ
タスク一覧
出力
ビルド正常終了
32 行 1 列 1 文字 挿入
スタート
demo3 - Microsoft Vi...
17:54

ステップ(7)

- 〈スタート〉→〈アクセサリ〉→〈コマンドプロンプト〉でDOS 窓を開いて SILC サーバを起動する
  - > cd ¥silc-1.2
  - > server
- 別の DOS 窓を開いてユーザプログラムを実行する
  - > cd "Visual Studio Projects¥demo3¥Debug"
  - > demo3
- 図のような実行結果が得られれば正常に動作している


DOS窓 - server
C:¥Documents and Settings¥kajiyama¥My Documents>cd ¥silc-1.2 (1)
C:¥silc-1.2>server (2)
single thread
load_modules("./modules/formats")
silc_register_format("SILC:dense (column major)")
silc_register_module("dense")
silc_register_format("SILC:Band")
silc_register_module("sparse_band")
silc_register_format("SILC:CRS")
silc_register_module("sparse_crs")
silc_register_format("SILC:JDS")
silc_register_module("sparse_jds")
load_modules("./modules")
silc_register_module("blasmodule")
silc_register_module("coremodule")
silc_register_module("leq_cg")
silc_register_module("leq_gs")
silc_register_module("leq_lis")
silc_register_module("leq_lu")
silc_register_module("linpackmodule")
set_endian()
server: little endian
client: little endian
silc_find_binary_op_impl()
silc_object_duplicate(0x4f8510, 1) column
blasmodule: solve_1(0x4ee820, 0x4fe740)
silc_object_free(0x5006e0)
bound to _TMP000000:
column vector, 1000 elements of double (
released _TMP000000
assigned to x:
column vector, 1000 elements of double (
released _TMP000000
DOS窓
C:¥Documents and Settings¥kajiyama¥My Documents>cd "Visual Studio Projects¥demo3¥Debug" (3)
C:¥Documents and Settings¥kajiyama¥My Documents¥Visual Studio Projects¥demo3¥Debug>demo3 (4)
connected to CF-R3 on port 1141
number of formats = 4
0: "SILC:dense (column major)"
1: "SILC:Band"
2: "SILC:CRS"
3: "SILC:JDS"
Request: >A
Response: 200 OK
Request: >b
Response: 200 OK
Request: :9:x = A ¥ b
Response: 200 OK
Request: <x
Response: 200 OK
0.054663s
||b-Ax|| = 3.839510e-015
C:¥Documents and Settings¥kajiyama¥My Documents¥Visual Studio Projects¥demo3¥Debug>
スタート
DOS窓 - server
DOS窓
CAPS
KANA
17:55