複数CPUのための共有メモリコンポーネント取扱説明書

中央大学理工学研究科 人間機械協調システム研究室 (國井研究室) 小島 隆史

2007年12月1日

目 次

1　はじめに

1.1　本書について

本書では、RT ミドルウェア上で動作する共有メモリ RT コンポーネント「複数 CPU のための共有メモリコンポーネント」に関してその利用のための、機能、仕様手順、仕様等を説明する。

1.2　必要意義

ロボットシステムは、さまざまなタイプのデータが各プログラムで扱われる。このようなデータには、小さいものもあれば大きいものもあるが、特に、大きいものの場合にはオーバヘッドが生じ問題となりうる。一方、分散システムのような効率的な処理できるようなシステムを構築するためには、各プロセス間のデータ共有はかかせず、データの共有に時間がかかれば、分散システムにおいて大きな障害となりうる。このような問題の解決方法として共有メモリの利用が挙げられるが、共有メモリは、ハードウェア的に独立しているパソコンのようなＣＰＵでは、各々のＣＰＵでしか利用できないという制約条件が存在する。

そこで、複数のＣＰＵでも共有メモリを仮想的に１つのメモリとしてみなすことができれば、負荷を分散させるような、分散システムの構築を、CPU を意識することなく容易に分散システムプログラミングを行うことができる。そのため、このような機能を実現するようなコンポーネント、複数 CPU のための共有メモリコンポーネントを作成した。

2　複数 CPU のための共有メモリコンポーネントの機能概要

複数 CPU のための共有メモリコンポーネントは、以下特別に断りがない場合、SHM コンポーネントと表記し、SHM コンポーネントに接続するコンポーネントをクライアントコンポーネントと表記する。本章では SHM コンポーネントの機能について概説する。

2.1　概要

SHM コンポーネントは、複数の CPU を使ったシステムの場合に共有メモリの作成及び、同期機能を持つコンポーネントである。SHM コンポーネントでは、各 CPU に、他コンポーネントから要求されたと共有メモリを作成し、他 CPU にある SHM コンポーネントと同期を行うことで仮想的に、１つの共有メモリを利用しているように扱うためのコンポーネント (Fig.1,Fig.2) である。また、補助機能として、セマフォ領域も共有メモリ１つにつき１つ確保し、提供するため、複数のコンポーネントのクリティカルセクションがある共有メモリでも、排他的制御を容易に行うことができる機能を提供する。

2.2　作成・削除・管理機能

SHM コンポーネントは、作成・削除要求があった場合、その命令を実行する。(Fig.3) 作成される共有メモリ情報から、共有メモリの ID とセマフォの ID を提供するポート、及び更新されたことを SHM コンポーネントへしらせるためのポートを動的にポートの２つを増設する。これによ

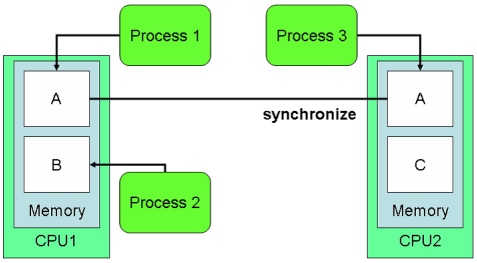

Fig 1: synchronizing shared memorys

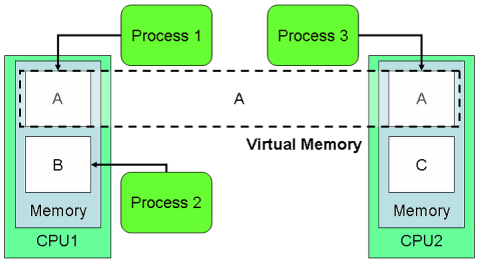

Fig 2: Virtual shared memory

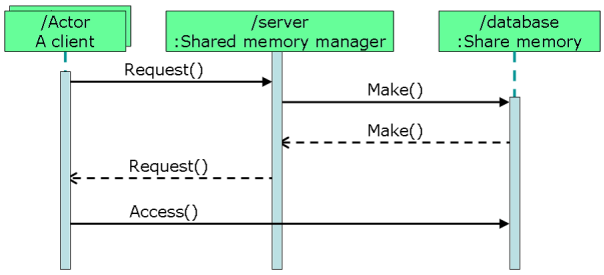

Fig 3: sequence diagram in solving request

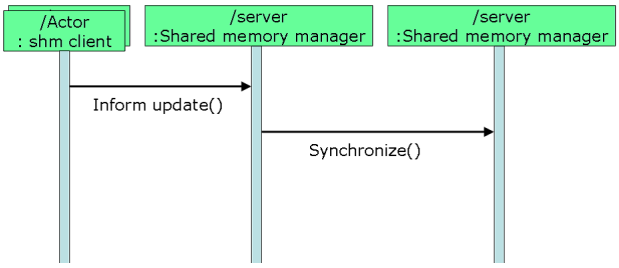

Fig 4: sequencd diagram in synchronization

り、同期させたい共有メモリを利用するコンポーネントから、更新通知を受け取れば、同期させる他 CPU の SHM コンポーネントにデータを送信し同期を図る。(Fig.4

また、複数のコンポーネントで、同じ共有メモリを利用している場合、SHM コンポーネントは作成要求があったコンポーネント数を覚えておき、削除要求の数が等しくなるまで、共有メモリを管理し続ける。これにより、他コンポーネントが共有メモリからデタッチしたとしても共有メモリは削除されずに残り、その CPU 内の最後のコンポーネントがデタッチすると自動的に共有メモリが解放される。

2.3 同期機能

SHM コンポーネントを同期させるためには、自分以外の SHM コンポーネントと入出力同期ポートの接続を行う必要がある。同期ポートを接続すると、自 SHM コンポーネントにおいて、更新通知をクライアントコンポーネントから受け取った場合、出力ポートから、更新通知を受け取った共有メモリ情報と、そのデータが送信されます。他 SHM コンポーネントは、自らが持っている共有メモリと同じ役割、同じサイズの場合には、送られてきたデータを自らの持つ共有メモリにデータをコピーします。更新通知ポートへの接続がない場合、もしくは、更新通知ポートに接続はしているが更新通知を行わないクライアントコンポーネントのみの場合、他 SHM モジュールとの同期を行わない単純な共有メモリとして利用されているのと同じになる。

同期相手が、同じ目的の共有メモリをもっていなくても、更新通知ポートに更新通知が行われると、SHM モジュールは同期送信ポートから送信をはじめるため、余分な通信トラフィックを生じさせてしまう可能性がある。ゆえに、あえて更新通知を行わないような利用の仕方や、同一 CPU でももう 1 つ SHM コンポーネントをインスタンス化するなどすることにより、無駄な通信トラフィックを排除することできる。

SHM コンポーネントの同期方法には、RT ミドルウェアの通信ポートを利用している。ゆえに、同期が高速に行われるためには RT コンポーネント処理速度に依存する。また、すべての共有データを一つのコンポーネントで管理させているため、他 CPU と同期させる共有メモリの量が増えれば増えるほど、完全な同期には時間がずれるなどの問題が生じる可能性も存在する。

3 SHM コンポーネントの仕様

3.1 共有メモリクラス

SHM コンポーネントでは、管理、利用を容易にするために作成した sharedMemoryManager クラスと sharedMemoryClient クラスを利用している。そこで簡単にこれらについての仕様を述べる。より詳細な情報については付録を参照のこと。

3.1.1 sharedMemoryManager

sharedMemoryManager クラスは、共有メモリを作成、管理するためのクラスであり、SHM コンポーネント内部の中核として利用している。ゆえに、クライアント側では利用することはないクラスである。また、Windows 用に拡張する場合、このクラスの共有メモリ作成部分を追加することで、Windows でも利用可能である。

sharedMemoryManager クラスは、ポートからの入力から得る、仕様用途と、サイズから共有メモリを作成し、管理する機能を持つ。また、それらの情報を使ってアタッチ数の増加や、減少、共有メモリの ID を返す役割も担い、アタッチ数が 0 となったとき共有メモリおよびセマフォ領域を削除する。

3.1.2 sharedMemoryClient

sharedMemoryClient クラスは、共有メモリを利用するためのクラスであり、SHM コンポーネントの同期用や、クライアントコンポーネントによる共有メモリアクセスを容易にするためのクラスです。そのための補助機能としてセマフォも同時に領域を確保することが可能であり、セマフォを利用することで排他的なメモリアクセスが可能である。

このクラスでは共有メモリ外の領域にアクセスすることできてしまうため、注意する必要がある。このようなメモリ外の領域にアクセスしてしまうことを防ぐ簡単な利用法として構造体によるアクセスをすることを推奨する。

3.2 通信ポート

クライアントコンポーネントは、最低限サービスポートであるリクエストポートを持つ必要がある。クライアント側ではコルバコンシューマ型のポートを持つ必要があり、クライントは、このポートを介し、共有メモリの要求、ID の取得等をおこなうことができる仕組みとなっている。また、別途更新通知ポートを持つことにより複数 CPU 環境での同期機能の提供や、共有メモリの更新通知をうけとるような入力などを得ることも可能である。

本節では、これらの通信ポートについて解説する。

3.2.1 サービスポート

リクエストポート リクエストポートは、クライアント側の要求により、サービスを開始する。クライアント側から request 関数がよばれると、フラグがセットされ、SHM コンポーネント側では、共有メモリにアタッチした数をカウントアップし、共有メモリの作成が自動でおこなわれる。その作業の後に、作成完了フラグがたつ。そのフラグに対しクライント側では、受信完了フラグを立てることで一連のサービスの終了となる。このとき、他にあいているポートが存在しない場合、ポート数が１つ増え、クライアントコンポーネントをさらに接続させることができる。

簡単にその使用方法について記述する。request(control,byte,explain) 関数の引数は順に、制御変数（１：作成、ー１：削除)、必要バイト数、共有メモリの役割の説明である。必要バイト数及び共有メモリの値は、共有メモリを生成するための鍵として使われ、確保を行う。これにより、説明が同じ文字列であろうとデータ数が違えば別の共有メモリとして確保することは可能である。また、作成された共有メモリは、getShmID() により得ることができる。本コンポーネントでは、同時にセマフォ領域を１つ分容易しているので、getSemID() を使って同様に利用可能である。また、それらが作成されたときの状態を把握する関数として、getRequestFlag() により取得が可能である。この返り値により状態を区別する。返り値の値と、その状態を Table.1 にまとめる。

```
interface requestID
{
  // ************* consumer control ************//
  // request sharedmemory  to consumer from provider (in function set requestFlag = 1)
  void request(in short control ,in long byte, in string explain);

  //get shared memory ID & get sempahore ID
  long getShmID();
  long getSemID();

  // ************* each component using function *****************//
  // get request flagment which tells its phase
  // 0:no request ,  1: requested , 2 : made shm ,3: process finished
  short getRequestFlag();

  // inform finishing from cosumer
  void  setRequestFlag(in short condition);

  // ************* provider control ************//
  //  get request sharedmemory's explain(to make key)
  string getExplain();

  // get request byte (to make key & size)
  long  getByte();

  // get control 1: add , 0:none  ,-1:erase
  short getControl();
```

```
  // shared memory & semaphore ID set (set request flag = 2)
  void setIDs(in long shmID, in long semID);
};
```

Table 1: getRequestFlag() function's return value and condition

返り値	状態
0	初期状態 (共有メモリは作成されていない)
1	共有メモリ作成通知フラグ ON(request 関数によりセット)
2	共有メモリ作成完了フラグ ON(SHM コンポーネントがセット)
3	共有メモリ参照完了フラグ ON(クライアント側が任意でセット)

※注意　リクエストポートは、一つのクライントコンポーネントにつき一つのポートを利用するようにしてください。

3.2.2　出力ポート

更新通知ポート　更新通知ポートは、SHM を他 CPU と同期させたる時に使用する。同一 CPU 内のコンポーネントのみのデータ共有目的の場合持つ必要はないが、拡張性のため持っておくことが望ましい。他の CPU と同期させたいときは、共有メモリの説明と、そのバイト数を送ることで SHM コンポーネントが、同期ポートが繋がっている SHM コンポーネントに対し、同期動作を行う。

```
struct update_inform_connector
{
// Time stamp
RTC::Time updt_tm;

// explanation of roll for shared memory
string explain;

// shared memory's byte
short byte;
};
```

※このポートは、入力ポートとしてクライアントに持たせば、他のクライアントコンポーネントに共有メモリが更新されたことを受け取ることもできます。ただし、共有メモリの ID が送信されているのではないため、共有メモリの特定はクライアント側があらかじめ、知っている必要がある。

3.2.3　同期ポート

同期ポートはクライアント側では使用しないが、SHM コンポーネントがデフォルトでもつ入出力ポートである。これらのポートを他の CPU と接続することで、更新通知が起こったコンポーネント

を同期させるデータを送信し、受信する。各変数は、説明 (syn_explain) と、容量数 (syn_byte)、データ (syn_data) の役割を担っており、共有メモリはデータに 1byte ずつ格納し、送受信する形態を取る。

```
struct synchronize_database_connector
{
// Time stamp
RTC::Time tm;

// explanation of roll for shared memory
string syn_explain;

// shared memory's byte
short  syn_byte;

// shared memory's data
sequence<char> syn_data;
};
```

3.3 一時フォルダ

共有メモリを作成するため、SHM コンポーネントは、一時ファイルを作成しその固有の ID を利用している。そのため、一時的にフォルダを作り、空のファイルを作成する。自動的に削除される仕組みではあるが、削除されないこともある。削除されていない場合は、共有メモリ作成のための一時ファイル等が残っているので、手動で削除しておくことを推奨する。もし、削除しない場合は、一時ファイルにアクセスする権限が足りずに、共有メモリの値が同じになることがある。

4 SHM コンポーネントの使用手順及びシステム作成例

本章では SHM コンポーネントの使用手順例を、単純なサーバクライアントの同期システムを例に、クライアントコンポーネント作成手順を説明する。本章で、１つの CPU におけるコンポーネントのテストと、実際に２つの CPU での同期させるコンポーネントのテストについて解説を行う。

4.1 システム設計

SHM コンポーネントの同期を頻繁に行ったり、大きなデータを送ると通信トラフィックが増加し、通信を圧迫する。そこで、SHM コンポーネントの同期は極力少なくなるように設計することが望ましい。すなわち、大きなデータは、同一 CPU 内で利用し、同期ようの更新通知をモジュールに送らないようにすることで、無駄な通信トラフィックを防止するようなシステム設計を行うことが望まれる。

4.2 同期確認用サンプルシステム

簡単な同期を確認するための、４コンポーネントによるサンプルシステムを作成例をしめす。

- Read only client :共有メモリを参照し、読み込みだけを行うコンポーネント
- Read and write client : 共有メモリを参照し、読み込み及び書き込みを行うコンポーネント
- Server：SHM コンポーネント
- Server：SHM コンポーネント

読み書きを行うコンポーネントからは、同期信号を出力し、SHM コンポーネントに同期を行わせる。読みこみだけを行うコンポーネントが参照する共有メモリは、同期によってのみ値が更新される。

4.3 コンポーネントの動作テスト

サンプルシステムの動作手順を記述する。

1. omniNames を起動
2. すべてのコンポーネントをインスタンス化 (ネームサーバに登録)
3. RTC リンクを立ち上げ、インスタンス化したネームサーバに接続
4. RTC リンクなどで、SHM コンポーネントと、クライントコンポーネントのサービスポートを接続
5. RTC リンクにより、各コンポーネントをアクティブ化

ここまでの作業を終えると、SHM コンポーネントは、共有メモリを作成がおこなわれ、クライアント側は共有メモリの ID を受け取り、クライアントコンポーネントは、読み込みなどの所定の動作を自分の参照する共有メモリに対して行い始める。ここで、SHM コンポーネントの同期ポートの入出力を結び、かつ、SHM コンポーネントの更新通知入力ポートにクライアントコンポーネントの更新通知出力ポートを接続する。すると、共有メモリの同期が SHM コンポーネントを介し始まり、共有メモリの更新が行われる。(Fig.7)

動作環境による違い　このサービスは複数の CPU で動作を確認する場合も、単一の CPU で行う場合も、使い方は基本的には変わりなく動作させる。ただし、インスタンス名が同じになると RTC-Link 上（ネームサーバ上）では、新しい方のみが認識されるため rtc.conf の設定を変更することにより単一 CPU でも動作可能です。

ただし、単一 CPU 内で、２つ以上の SHM コンポーネントが立ち上がっている場合でも、たとえ、request 関数において、引数が全く同じだとしても、全く違う共有メモリが２つ以上作成される仕様となっております。(Fig.5,Fig.6)

異常終了時のメモリ及びセマフォ開放　なんらかの原因で、通常通り共有メモリが削除されない場合、一時フォルダ内に、removeSHM&SEL.sh が作成されています。これを実行することで、簡単に残ってしまった共有メモリを削除できるので利用してください。（正常に、sharedMemoryManager クラスのデストラクタが呼ばれれば、全メモリは解放されます）

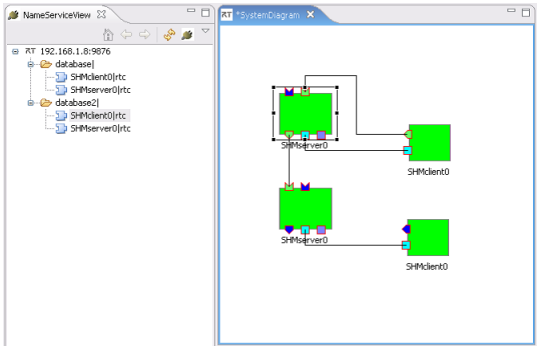

Fig 5: The four test components in single CPU

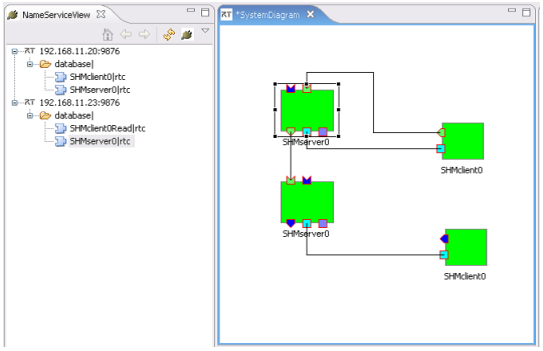

Fig 6: The four test components in two CPUs

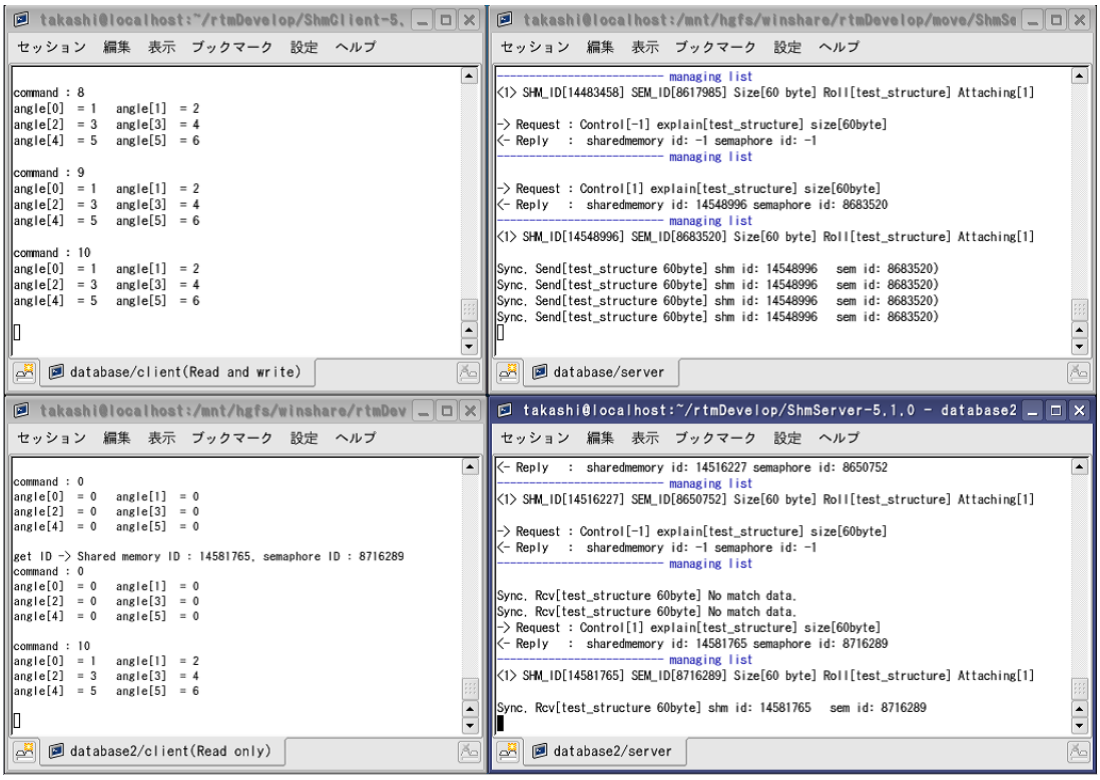

Fig 7: The situation : SEM Client Componet request SEM Component to make shared memory and SEM component is synchoronzing. Left top :SHM Client component(Read and write), Right top:SHM Component Left bottom:SHM Client component(only read), Right bottom:SHM component

4.4 応用例 (同一 CPU における複数のクライアントコンポーネント接続の例)

仮想 H8 コンポーネントとその制御用 GUI コンポーネントによる応用例を紹介する。仮想 H8 コンポーネントは、H8 からの受信データを共有メモリにアップロードし、H8 への送信データを共有メモリからダウンロードするコンポーネントである。制御用 GUI コンポーネントは、共有メモリの内容を確認する機能、及び命令を送信する機能を持つコンポーネントである。これらのクライアントコンポーネントの入出力を Table.2 にまとめる。

各クライアントコンポーネントのサービスポートと SHM コンポーネントのサービスポートを接続し、アクティブ化する。。(Fig.8) さらに仮想 H8 コンポーネントの更新通知出力ポートと、GUI コンポーネントの更新通知入力ポートに接続する。これにより、共有メモリの内容が通信の内容となり仮想的に共有メモリを利用する一つのコンポーネントとして H8 をあつかうことができる。

複数のコンポーネントが、１つの共有メモリに接続しているとき、SHM コンポーネントはその数を認識する。もし、この認識に失敗している場合は、クライアントコンポーネントが、共有メモリへの接続を切断したときに、他のコンポーネントが接続しているにもかかわらず削除してしまう可能性がある。それを防ぐには、Fig.10 のようにコンソールに表示されている情報を確認することで、正常に接続が行われたかどうかを確認できる。応用例の場合には同じ共有メモリに対し、２つのコンポーネントが接続している。そのためコンソール側においても Attatching[2] と表示され、接続している数が等しく、正常に接続が行われていることが確認できる。

Table 2: Each Virtual H8 component's & H8GUI component's property

	仮想 H8 コンポーネント	GUI コンポーネント
役割	H8 の状態・情報を共有メモリに書き込む	共有メモリ情報の提示・コマンド送信
サービス	共有メモリ ID 要求ポート	共有メモリ ID 要求ポート
入力	送信用共有メモリ更新通知	受信用共有メモリ更新通知
出力	受信用共有メモリ更新通知	送信用共有メモリ更新通知

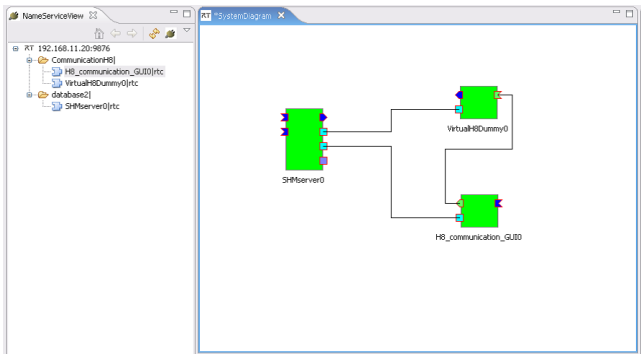

Fig 8: The connection of each component

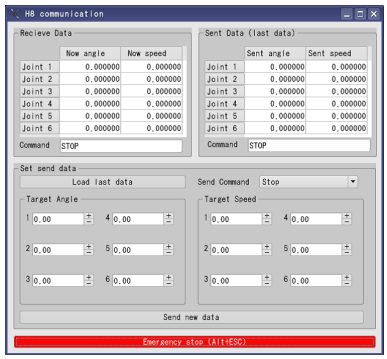

Fig 9: GUI for communicating H8

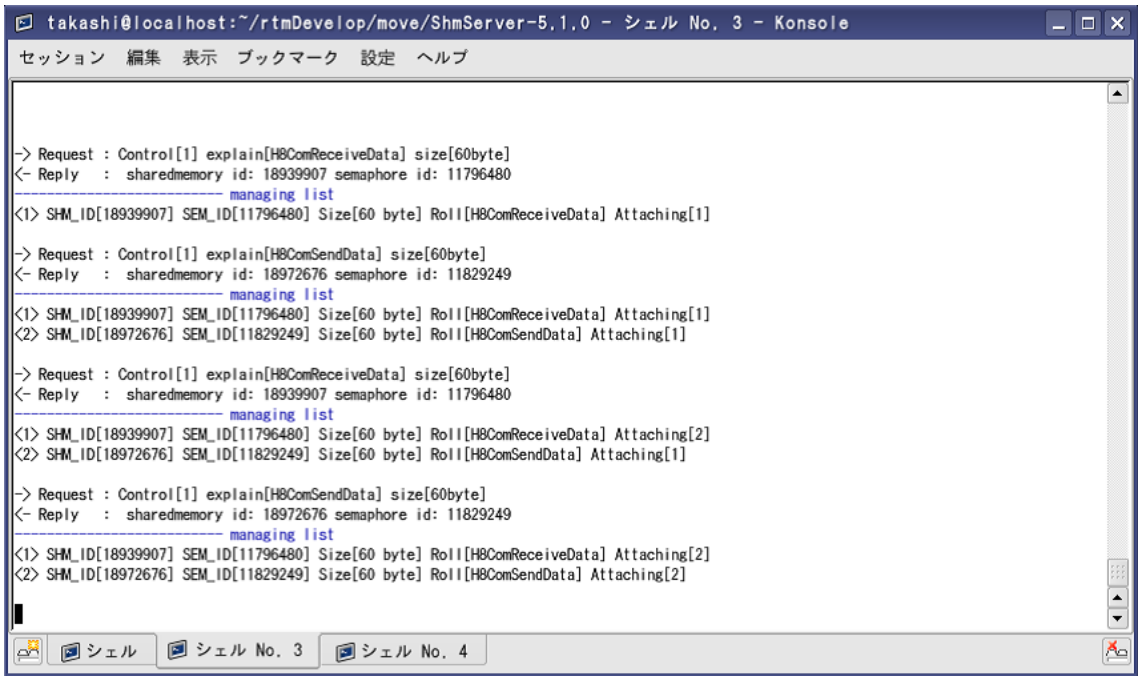

Fig 10: The SHM component's console

5 環境

5.1 開発環境

- OS: Fedora core 6 (kernel:2.6.18-1.2798)
- コンパイラ:gcc 4.1.1-30
- CORBA:omniORB 4.0.7
- ACE: ace 5.5.4
- OpenRTM-aist:OpenRTM-aist-0.4.1-RELEASE

5.2 動作環境

openRTM-aist-0.4.1 がインストールされている Linux 環境であればコンパイルし、実行可能です。windows は sharedMemoryManager,sharedMemoryClient クラスが対応していないため使うことができません。(OpenRTM-aist-0.4.0 では InPort,OutPort がデフォルトが RingBuffer でないため、エラーがでます。このコンポーネントを 0.4.0 で動作させるには、OutPort,InPort の宣言を OutPort<dataport> から OutPort<dataport,RTC::RingBuffer> のように変更し、¡rtm/RingBuffer.h¿ をインクルードすることで使用可能となります)

6　その他

6.1　応用例で用いたGUIコンポーネントについて

基本的にGUIは、独自のメインルーチンをもっており、通常RTミドルウェアのままで実装することは難しい。そこで、実装するために、Pスレッドを利用し実現することができた。今回の応用例では、onActivate内で、GUI用スレッドを立ち上げ、GUIのボタンを押すと、onExecute内にフラグが立ち、更新通知ポートから出力が行われる。また、onDeactivate内で、スレッドをキャンセルし、終わらせてある。その際に、GUIを非表示にしておくことで、もう一度アクティブ化したときに表示させることができる。ただし、その際には、GUIの中を初期化し、適切な状態に設定しなければ望むような結果がえられないので注意が必要である。

6.2　同期ポートについて

同期ポートでは、受信、送信を行う際、ループ間隔が長くなるとオーバフローしてしまう可能性がある。とくに、同期の際、システムでは共有メモリのロックを行うため、他プログラムが長い時間ロックをしていると、受信、送信部分で動かなくなってしまう。そこで、本システムでは、同期を行うための作業は、Pスレッドによる解決を図っている。これにより、送信している間にも受信が行えると言うことが実現できる。また、受信と送信するポートの仕様が重なった場合でもセマフォを使った排他的な制御によって、データ破壊がおこることはない。ただし、必ず後からロックした方がデータを更新するしてしまうため、最新のデータが先にロックをし、古いデータが後からロックし書き込めば、古いデータによって上書きされてしまう可能性がある。

6.3　注意事項

本バージョンのサンプルプログラムでは、同期の際に生じるようなクリティカルセクションについては、サポートしておりません。ゆえに、同時に別々のSHMコンポーネントに作成された同じ共有メモリが書き換えられたときの動作については保証いたしません。しかし、更新通知等にはタイムスタンプを持っているため、ここに、更新時刻を書いておくことで、このような場合でも新しいデータを優先させるようなプログラムとするような拡張性を持っています。

また、異なるCPUにあるSEMコンポーネントとそのクライアントの場合はエラーが起こります。複数のCPUで利用する場合には必ず、同一SEMコンポーネントと接続するようにしてください。

付録

● sharedMemoryManagerクラスリファレンス

int add(int memorySize, const char *explanation)

役割:

共有メモリを作成します。すでに存在する場合はカウントアップして、アタッチしている共有メモリの数をカウントアップします

引数:

memorySize :共有メモリの使用サイズ(byte)
*explain :共有メモリの使用用途(文字列)
返り値:
作成に失敗した場合-1、成功した場合 memory id が返り、アタッチカウントをカウントアップは 0 が返ります。

void erase(int memorySize, const char *explain)

役割:
該当する共有メモリを削除(もしくはアタッチカウントを減算します)
引数:
memorySize :共有メモリの使用サイズ(byte)
*explain :共有メモリの使用用途(文字列)
返り値:
削除に失敗した場合 -1, 削除した場合 1, アタッチ数を減算する場合 0

void eraseAll()

役割:
アタッチしている共有メモリの数にかかわらず強制的に全 shared memory を削除します
(デストラクタでも呼び出されます)

void outputList(FILE *fp)

役割:
出力先を指定して Shared memory list を表示
ファイルポインタを渡せばファイルに書き込みでき、
stdout, stderr などで、ターミナルに表示できます

void display()

役割:
ターミナルに情報を表示させる

int returnShmID(int memorySize, const char *explanation)

役割:
該当する共有メモリ ID を返します
引数:
memorySize :共有メモリの使用サイズ(byte)
*explain :共有メモリの使用用途(文字列)
返り値:
存在しない場合 -1, 存在した場合 共有メモリ ID

int returnSemID(int memorySize, const char *explanation)

役割:
　該当するセマフォID を返します
引数:
　memorySize :共有メモリの使用サイズ (byte)
　*explain　:共有メモリの使用用途 (文字列)
返り値:
存在しない場合 -1, 存在した場合 セマフォID

void write(int memSize, const char *explain)

役割：
　指定された共有メモリ、セマフォの ID を出力 ((共有メモリ名).txt)
※　共有メモリがない場合は出力されません
※　実行ファイルと同じ階層に作成されます

void write(const char* path, const char* filename)

役割：
　共有メモリ、セマフォをまとめて出力 (filename.txt)
引数
 dir : ディレクトリ
 filename : ファイル名
返り値
　エラー時は-1 が返ります。成功時には 0 が返ります。

void write(const char* filepathAndName)

役割：
　共有メモリ、セマフォをまとめて出力 (filename.txt)
引数
 pathAndFilename : ファイルまでの相対パス（or 絶対パス）
返り値
　エラー時は-1 が返ります。成功時には 0 が返ります。

● sharedMemoryClient クラスリファレンス

int read(const char *dir, const char* filename)

役割：
　ファイルから、共有メモリの id とセマフォの id を取得します。
（2 行目にある最初の数字を共有メモリ、次の数字をセマフォの ID として取得します）
引数

dir : ディレクトリ
filename : ファイル名
返り値
エラー時は-1 が返ります。成功時には 0 が返ります。
※ 読み込んだ内容は、public 変数の sharedMemoryID, semaphoreID に代入されています。

int read(const char *pathAndFilename)

役割：
ファイルから、共有メモリの id とセマフォの id を取得します。
（2 行目にある最初の数字を共有メモリ、次の数字をセマフォの ID として取得します）
引数
pathAndFilename : ファイルまでの相対パス（or 絶対パス）
返り値
エラー時は-1 が返ります。成功時には 0 が返ります。

int lock(int sem_id, int target=0)

役割：
指定された id のセマフォをロックします
ロックできるまでプログラムは停止しし続けます
引数
sem_id : セマフォの id を入力します
target : アクセスするセマフォの番号を入力します (省略可能)
デフォルトでは先頭のセマフォにアクセスします。
返り値
エラー時は-1 が返ります。成功時には 0 が返ります。

int lock(int semphore_id, double timeOutSec, int target=0)

役割：
指定された id のセマフォをロックします
規定した時間までプログラムは停止しし続けます
引数
sem_id : セマフォの id を入力します
timeOutSec : タイムアウト時間を規定します (秒)
規定した時間が経過しても確保できない場合 -1 が返ります
target : アクセスするセマフォの番号を入力します (省略可能)
デフォルトでは先頭のセマフォにアクセスします。
返り値
エラー時は-1 が返ります。成功時には 0 が返ります。

int unlock(int sem_id, int target=0)

役割：
　指定された id のセマフォをアンロックします
　アンロックできるまでプログラムは停止しし続けます
引数
 sem_id : セマフォの id を入力します
 target : アクセスするセマフォの番号を入力します（省略可能）
　　デフォルトでは一番最初のものにアクセスします。
返り値
　エラー時は-1 が返ります。成功時には 0 が返ります。

int unlock(int semphore_id, double timeOutSec, int target=0)

役割：
　指定された id のセマフォをアンロックします
　規定した時間までプログラムは停止しし続けます
引数
 sem_id　　: セマフォの id を入力します
 timeOutSec : タイムアウト時間を規定します（秒）
　　規定した時間が経過しても確保できない場合 -1 が返ります
 target　　: アクセスするセマフォの番号を入力します（省略可能）
　　デフォルトでは一番最初のものにアクセスします。
返り値
　エラー時は-1 が返ります。成功時には 0 が返ります。

typename<SHARED_MEMORY_STRUCT> SHARED_MEMORY_STRUCT* returnPointer(int id)

役割　shared memory のポインタを返します
引数
 id : 共有メモリの id
返り値
 領域が存在しない場合は、NULL が返り、
 存在する場合は教諭メモリのポインタが返ります。

int semaphoreCheck(int sem_id)

役割　セマフォがアクセス可能かを調べます
引数
 sem_id : 共有メモリの id
返り値
 セマフォが存在しない場合は、0 が返り
 存在する場合は 1 が返ります。