FORTRAN による物理乱数プログラム例

プログラム実行準備

物理乱数生成システムは、クライアントサーバ方式のアプリケーションになっています。
そのため乱数を使用する側で、クライアント用ライブラリをコンパイルして、プログラムをそれとリンクする必要があります。
クライアント用ライブラリについては、Linux、AIX、Unix、Windows 用の各ライブラリが、以下のディレクトリにソースファイルで提供されています。
またこのディレクトリにある、物理乱数生成システムの仕様書には各ルーチンの実行方法が記述されていますので、そちらも参照してください。

物理乱数生成システムソースファイルディレクトリー覧

各ファイルは/home/doc/ismxc/Random　以下にあります。

・物理乱数配送システムクライアントサイドモジュール用ソースコード

1) Linux/AIX/Unix 系 OS 用 API ライブラリソースコード

application/src/client
application/src/client/RM_Api.o
application/src/client/Makefile
application/src/client/Makefile.aix
application/src/client/RM_Api.c
application/src/client/libRM_Api.a

2) API ライブラリテストプログラムソースコード

application/src/testprog
application/src/testprog/loop_test.csh
application/src/testprog/rmf77test.f
application/src/testprog/RM_Api_test
application/src/testprog/RM_Api_test.c

3) Windows 系 OS 用 API ライブラリソースコード

application/src/Win32
application/src/Win32/RM_Api
application/src/Win32/RM_Api/Debug
application/src/Win32/RM_Api/Release
application/src/Win32/RM_Api/Release/RM_Api.lib
application/src/Win32/RM_Api/Release/RM_Api.obj
application/src/Win32/RM_Api/Release/RM_Api.pch
application/src/Win32/RM_Api/Release/vc60.idb
application/src/Win32/RM_Api/RM_Api.cpp
application/src/Win32/RM_Api/RM_Api.dep
application/src/Win32/RM_Api/RM_Api.dsp
application/src/Win32/RM_Api/RM_Api.dsw

```
application/src/Win32/RM_Api/RM_Api.mak
application/src/Win32/RM_Api/RM_Api.ncb
application/src/Win32/RM_Api/RM_Api.opt
application/src/Win32/RM_Api/RM_Api.plg
application/src/Win32/RM_Api/RM_Api_server.h
application/src/Win32/RM_Api.dsw
application/src/Win32/RM_Api_test
application/src/Win32/RM_Api_test/Debug
application/src/Win32/RM_Api_test/Release
application/src/Win32/RM_Api_test/Release/loop_test.bat
application/src/Win32/RM_Api_test/Release/RM_Api_test.exe
application/src/Win32/RM_Api_test/Release/RM_Api_test.obj
application/src/Win32/RM_Api_test/Release/RM_Api_test.pch
application/src/Win32/RM_Api_test/Release/vc60.idb
application/src/Win32/RM_Api_test/RM_Api_server.h
application/src/Win32/RM_Api_test/RM_Api_test.cpp
application/src/Win32/RM_Api_test/RM_Api_test.dep
application/src/Win32/RM_Api_test/RM_Api_test.dsp
application/src/Win32/RM_Api_test/RM_Api_test.dsw
application/src/Win32/RM_Api_test/RM_Api_test.mak
application/src/Win32/RM_Api_test/RM_Api_test.ncb
application/src/Win32/RM_Api_test/RM_Api_test.opt
application/src/Win32/RM_Api_test/RM_Api_test.plg
```

クライアントのアーキテクチャに従って、必要なファイル（例えば Linux の場合は /home/doc/ismxc/Random/application/src/client　をコピーして、
手元のマシンでライブラリを作成します。Makefile が用意されていますので、ライブラリのインストールパスなどについては、各自の環境に合わせて修正してください。
ライブラリができたら、これをユーザプログラムとリンクします。ismxc ではコンパイル済みのライブラリが、/opt/NTLPrandom/lib/libRM_Api.a に用意されているので、こちらを利用することもできます。

実行例

① クライアント用プログラムをコンパイルするために必要なファイルをコピーします。(例ではカレントディレクトリにコピーしています。AIX を使用する場合は Makefile.aix をコピーした後、ファイル名を Makefile に変更してください)

```
% cp /home/doc/ismxc/Random/application/src/client/Makefile
% cp /home/doc/ismxc/Random/application/src/client/RM_Api.c .
.
```

② クライアントプログラムをコンパイルします。デフォルトでは gcc を使ってコンパイルされます。コンパイラを変えたい場合には、make コマンドの後ろに CC=使用するコンパイラ名（例 CC=icc）のように指定してください。この時使用するコンパイラにあわせて、必要なモジュールをあらかじめロードしておいてください。

% make （gcc を使ってコンパイルする場合）

% module load icc/9.0 （インテルコンパイラの環境を設定）
% make CC=icc (インテルコンパイラを使ってコンパイルする場合)

実行すると RM_Api.o と libRM_Api.a というファイルがカレントディレクトリに作られます。

③ 乱数を使用するプログラムとリンクします。ここでは、後述のサンプルプログラム(ファイル名 test.f)をカレントディレクトリにあるライブラリとリンクしています。ライブラリのある場所によって、-L のディレクトリ名を変更してください（例ではカレントディレクトリを指定するために、`pwd`の出力結果を渡しています）

```
% module load ifort
% ifort -o test test.f -L`pwd` -lRM_Api
```

④ ③までの処理でできたプログラム（test）を会話型環境で実行します。

```
%  ./test
Enter the number ( <= 20000000) of random numbers.
20000000    (20000000０以下の任意の数字を入力します)
 rminit:socket=               3
 rmri ret=      20000000
 rmri rec       1000000 =    477898457
 rmri rec       2000000 = -2064536265
                          :
                          :
 rmrd rec20000000= 0.975987195968627929687500 0
 rmfinish:socket=             3
```

⑤ また LSF の会話型ジョブとしても実行できます。これには bsub に-I オプションを付けて実行します。

```
% module load ifort/9.0   必要なモジュールのロード
% bsub –I –q q1
```

```
> ./test
> [Ctrl-D]
bsub> Job <6977> is submitted to queue <q1>.
<<Waiting for dispatch ...>>
<<Starting on lsfhost.localdomain>>
 Enter the number ( <= 20000000) of random numbers.
20000000      (20000000以下の任意の数字を入力します)
 rminit:socket=              6
 rmri ret=      20000000
 rmri rec       1000000 = -1023868617
 rmri rec       2000000 =   1598126137
 rmri rec       3000000 = -1281631222
 rmri rec       4000000 =   1991995695
                      :
                      :
 rmrd rec18000000= 0.590591788291931152343750 0
 rmrd rec19000000= 0.699936747550964355468750 0
 rmrd rec20000000= 0.013819456100463867187500 0
 rmfinish:socket=              6
```

⑥ 最後に通常のバッチジョブとして実行する例を説明します。サンプルプログラムでは、乱数の範囲を手入力する必要がありました。バッチで動かすために、このソースファイルを修正して、乱数の範囲を定数で参照できるように修正します。バッチジョブとして実行する場合は会話型でなくなるため、bsub コマンドに" -I "オプションをつける必要はありません。代わりに力結果を受け取るために、それが格納されるファイル名を-o オプションを使って指定してください。

ソースの修正点　次ページにあるサンプルプログラムの、10,11 行をコメントアウトし、11 行目の次に定数の設定をします。

```
10   C          print *,'Enter the number ( <= 20000000) of random numbers.'
11   C          read *, numrand
12              numrand=20000000
```

修正したプログラムをコンパイル、リンク後 bsub でキューにサブミットします。実行結果は test.out というファイルに登録されます。

```
% module load ifort/9.0
% bsub -q q1 -o test.out ./test
```

プログラムの概要

値が入力されると、それを最大値とする 3 種類の乱数を生成します。生成される乱数はそれぞれ、32 ビット符号付整数型、32 ビット単精度不動小数点型、64 ビット倍精度浮動小数点型の型を持ちます。

FORTRAN によるプログラム例

```
行番号          ステートメント

01              character*10 ip
02              integer*4           sock, ret, numrand, i
03      C       integer*4           intbuff(20)
04      C       real*4              real4buff(20)
05      C       real*8              real8buff(20)
06              integer*4           intbuff(20000000)
07              real*4              real4buff(20000000)
08              real*8              real8buff(20000000)
09
10              print *,'Enter the number ( <= 20000000) of random numbers.'
11              read *, numrand
12              ip = '172.22.0.129'
13              call rminit(sock, ip)
14              print *,'rminit:socket=',sock
15
16      C       numrand = 20
17
18              call rmri(ret, sock, intbuff, numrand)
19              print *,'rmri ret=',numrand;
20              do i = 1,numrand
21                  if (mod(i, 1000000) .eq. 0) then
```

```
22                      print *,'rmri rec',i,'=',intbuff(i);
23                  endif
24              end do
25
26              call rmrf(ret, sock, real4buff, numrand)
27              print *,'rmrf ret=',numrand;
28      10      format(' rmrf rec',i8,'=',f19.16)
29              do i = 1,numrand
30                  if (mod(i, 1000000) .eq. 0) then
31                      write (6,10) i,real4buff(i);
32                  endif
33              end do
34
35              call rmrd(ret, sock, real8buff, numrand)
36              print *,'rmrd ret=',numrand;
37      20      format(' rmrd rec',i8,'=',f28.25)
38              do i = 1,numrand
39                  if (mod(i, 1000000) .eq. 0) then
40                      write (6,20) i,real8buff(i);
41                  endif
42              end do
43
44              call rmfinish(ret, sock)
45              print *,'rmfinish:socket=',sock
46
47              end
```

プログラムの説明

行番号 13

rminit は物理乱数用ソフトウェアを初期化する際に使用されます。乱数サーバを利用する場合には、必ずこのルーチンをコールする必要があります。この例では 172.22.0.129 の IP アドレスを持つサーバを使用しています。
rminit は以下のような書式になっています。

```
rminit(integer, integer)
```

第１引数は、サブルーチンコールのチェックサムです。
指定された物理乱数生成・配送ソフトウエアとの接続成功によりサーバサイドとの接続識別用ハンドル（１以上の整数値）が設定されます。 エラー発生時は、"-1"が設定されます。
以後、他の API の利用時には、この接続識別用ハンドルが必要となります。

第２引数には、物理乱数生成・配送ソフトウエアが稼動する物理乱数サーバの IP アドレスを指定します。各サーバの IP アドレスについては、巻末の一覧表を参照してください。

行番号 18

rmri は 32 ビット符号付整数型の物理乱数を取得します。この例では intbuff が示すアドレスに、20 個の乱数が格納されます。
rmri は以下のような書式になっています。

```
rmri(integer, integer, integer, integer)
```

第１引数は、サブルーチンコールのチェックサムです。(成功時は、取得した物理乱数の個数を返します。エラー発生時には、"-1"を返します。)
第２引数には、rminit ルーチンから取得した接続識別用ハンドルを指定します。
第３引数には､取得した物理乱数を格納するための 32 ビット符号付整数型配列のアドレスを指定します。
第４引数には、取得する 32 ビット符号付整数型物理乱数の個数を指定します。

行番号 26

rmrf は 32 ビット単精度浮動小数点型の物理乱数を取得します。この例では real4buff が

示すアドレスに、20 個の乱数が格納されます。
ｒｍｒ ｆは以下のような書式になっています。

```
rmrf(integer, integer, real*4, integer)
```

第１引数は、サブルーチンコールのチェックサムです。(成功時は、取得した物理乱数の個数を返します。エラー発生時には、"-1"を返します。)
第２引数には、rminit ルーチンから取得した接続識別用ハンドルを指定します。
第３引数には､取得した物理乱数を格納するための 32 ビット単精度浮動小数点型配列のアドレスを指定します。
第４引数には、取得する 32 ビット単精度浮動小数点型物理乱数の個数を指定します。

行番号 35
rmrd は 64 ビット倍精度浮動小数点型の物理乱数を取得します。この例では real8buff が示すアドレスに、20 個の乱数が格納されます。
rmrd は以下のような書式になっています。

```
rmrd(integer, integer, real*8, integer)
```

第１引数は、サブルーチンコールのチェックサムです。(成功時は、取得した物理乱数の個数を返します。エラー発生時には、"-1"を返します。)
第２引数には、rminit ルーチンから取得した接続識別用ハンドルを指定します。
第３引数には､取得した物理乱数を格納するための 64 ビット倍精度浮動小数点型配列のアドレスを指定します。
第４引数には、取得する 64 ビット倍精度浮動小数点型物理乱数の個数を指定します。

行番号 44
rmfinish は物理乱数ソフトウェア用に確保した領域を開放し、その使用を終了します。
rmfinish は以下のような書式になっています。

```
rmfinish(integer, integer)
```

第１引数は、サブルーチンコールのチェックサムです。(成功時は 0、エラー発生時には、"-1"を返します。)
第２引数には、rminit ルーチンから取得した接続識別用ハンドルを指定します。

乱数サーバ IP アドレス一覧表

乱数サーバの初期化に IP アドレスを下表に示します。XC システムや研究所内から利用する場合は、内部ネットワークアドレスを、研究所外から利用する場合はグローバル IP アドレスを使用してください。

	グローバル IP アドレス	内部ネットワークアドレス
乱数サーバ 1	133.58.105.13	172.22.0.129
乱数サーバ 2	133.58.105.14	172.22.0.130
乱数サーバ 3	133.58.105.15	172.22.0.131
乱数サーバ 4	133.58.105.16	172.22.0.132
乱数サーバ 5	133.58.105.17	172.22.0.133
乱数サーバ 6	133.58.105.18	172.22.0.134
乱数サーバ 7	133.58.105.19	172.22.0.135
乱数サーバ 8	133.58.105.20	172.22.0.136