MULTI-THREAD-READY EPICS-PYTHON INTERFACE

Jun-ichi Odagiri and Noboru Yamamoto
Accelerator Laboratory, High Energy Accelerator Research Organization (KEK)
1-1, Oho, Tsukuba, Ibaraki, JAPAN, 305-0801

Abstract

EPICS (Experimental Physics and Industrial Controls) is a software toolkit for an accelerator control system and now widely used in the various laboratories and accelerators over the world. RARF/RIBF in Riken, KEKB/PF-AR in KEK and J-PARC in JAEK/KEK joint project are instalments of EPICS in Japan. Python, an object oriented script language, is also widely used in these systems for testing a system and for GUI building. Making EPICS-Python Interface multi-thread-ready improves performance of these application and also ease a development of applications.

EPICS-PYTHON INTERFACE のマルチスレッド化

１．はじめに

EPICS1(Experimental Physics and Industrial Control System)は米国LosAlamos国立研究所とArgonne国立研究所の加速器制御グループが共同で開発を始めたネットワーク分散型計算機制御システム構築のためのソフトウエアツールキットである[1]。現在は世界各地の研究所において加速器や大型実験装置の制御システムの基盤として利用され、それらの機関による国際共同研究として現在もなお改良が進められている。日本国内でも、KEKのKEKB,PF-AR, 理研RARF, RIBF, 原研-KEKのJ-PARC等の制御システムに採用されている。

EPICSの提供するツールにはedm/medm[2]などのグラフィカルな操作パネル作成ツールや、snc/seqといった状態遷移ダイアグラムに基づく機器制御プログラム作成ツールが含まれる。これらのツールは制御システムに要求される機能のほぼ全域をカバーするが、ここのシステムに合わせた若干のアプリケーション開発が必要となる場合が多い。

この様な場合には、Python2/Tk/Matlabといったスクリプト言語がしばしば用いられる。これらのスクリプト言語はまた、制御アルゴリズムの検討やシステムの管理ツールとしても有効に用いられている。

KEKBやPF=ARでは、オブジェクト指向スクリプト言語Pythonが利用されている[3]。PythonをEPICS環境化で利用するためのEPICS-PythonインタフェースはKEKB制御グループで最初に開発され[4]これまでKEKB制御アプリケーション開発の一旦を支えてきた。EPICSのバージョンアップに伴い、このEPICS-PythonインタフェースのMulti-thread対応が望まれていた。今回Python-EPICSをMultithread対応とすることで、これらのアプリケーション開発が容易になり、またその実行効率も大きく向上した。この論文ではPython-CAインターフェイスのマルチスレッド化の手順およびマルチスレッド対応化されたPython-CAインタフェースの利用法およびこれまでとの違いについて説明する。

２．EPICS-CAライブラリ

EPICS-CA(Channel Access)ライブラリはEPICSの基幹ソフトウエアの一つである。CAライブラリはシステムを構成する計算機間で情報を交換するためのプロトコル(Channel Access:CA)を実装している[図1]。CAライブラリには、IOC等CAサーバ側で利用される部分とCAクライアント側で使われるCAクライアントライブラリの二つから構成される。制御に使われる操作画面などはこのCAクライアント・ライブラリを使用している。CAライブラリはLinuxを始めとするUnixシステム、Windows, VMS, VxWorks, RTEMS等のOSで動作する。UnixでのCAライブラリは当初シングルスレッドのみをサポートしていたが、2002年にリリースされたEPICS R3.14 以降のバージョンでマルチスレッド化された。これにより、CAのモニター機能を使うCAクライアントの効率化や操作画面のユーザに対する応答性の向上が実現された。

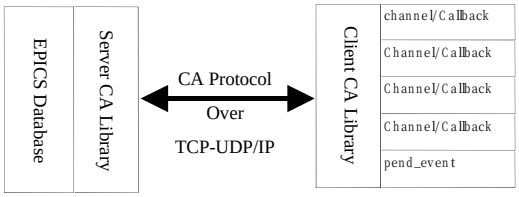

図1 Channel Access **プロトコルとそのライブラリ**

CAプロトコルは非同期通信を基本としている。すなわち、Get/Put等のリクエストをCA libraryのAPIを通じて呼び出すと、これらのAPIはリクエストをキューに積み挙げたところで呼び出し元に制御を戻す。リクエストの完了はサーバからの完了通知を待って指定したコールバック関数を呼び出すことで実現される。

3.13以前のEPICS/CAライブラリではこのコールバックを処理するために、適切な間隔でca_pend_event等のライブラリ関数を呼び出してやる必要があった[図2(a)]。TkなどのFile Descriptorマネー

[1] http://www.aps.anl.gov/epics/
[2] http://www.python.org/ , http://www.python.jp/

ジャ機構をもつアプリケーションではこの処理をFile Descriptorマネージャに任せることも出来た。

バージョン3.13以前のスレッド化されていないEPICS/CAライブラリでは、何らかの理由で、例えばアプリケーションの一部での長時間CPUをとるなど、ca_pend_event等の関数が呼ばれず、callbackのキューが処理されないと、アプリケーションはCAサーバからの応答を適切なタイミングで処理出来なくなってしまう。

スレッド化されたバージョン3.14以降のEPICS CAライブラリでは、ライブラリの初期化オプション

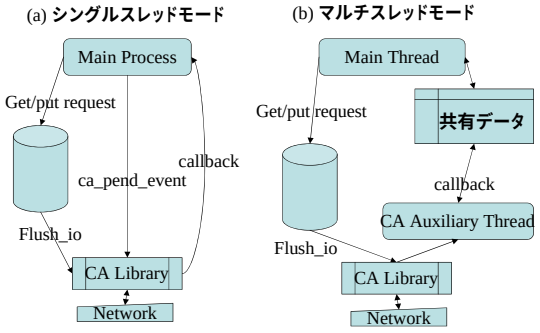

図2 EPICS CA Libraryの二つのモード :(a) シングルスレッドモードおよび (b)マルチスレッドモード

ca_enable_preemptive_callback を指定する3ことでマルチスレッド化されたCAライブラリの機能が利用可能となる。このモードでは、サーバからの応答やエベント通知(モニタ、アーカイブ、アラーム等)はライブラリが起動する別スレッドで処理される。このため、 CAクライアント側では、明示的にca_pend_event等のAPIを呼び出す必要はない[図2(b)]。このとき、コールバック関数が実行されるスレッドはCAのリクエストを送り出したスレッドとは異なっている。したがって、これらのスレッドで共有される資源の制御に注意を払う必要がある。

3. EPICS-PYTHON INTERFACE

EPICSでは標準的なCAクライアント・アプリケーションの一つとしてedm(extensible display manager)等が用意され、簡単に操作画面を構築することが出来る。これらの汎用アプリーケーションでカバー出来ない操作画面に対する解答としてKEKB/PF-AR制御システムではPythonスクリプトを利用している[3]。オブジェクト指向スクリプト言語であるPythonではTk widgetを利用するTkinter, ORACLEやPostgreSQLといったRDB接続のためのモジュール、数値計算のためのNumeric/numarrayなど豊富なモジュールを利用出来る。このPythonスクリプト中でEPICS CAライブラリを利用するためのモジュールは KEK と FNAL で独立に開発された[4,5]。

Python/CA InterfaceとTkinterをつかったPython プログラム(スクリプト)の例を図3に示す。このプログラムではメインのスレッドでTkInterモジュールを使ったGUI部品（Widget）に"jane"と言う名前のEPICSチャンネル4データを表示するとともに、別スレッド

```
#!/usr/local/bin/pythonw

import ca,thread,time
from CaSimple import *

def Sub():
    ch=ca.channel("fred")
    ch.wait_conn()
    ch.autoUpdate()
    ch.flush()
    while 1:
        if ch.val >0.5:
            print ch.name,"exceed 0.5",ch.val
        elif ch.val <-0.5:
            print ch.name,"is smaller than -0.5",ch.val
        time.sleep(1.0)

def main():
    th=thread.start_new_thread(Sub,())
    w=Simple("jane",master=Toplevel())
    w.start_monitor()
    w.mainloop()

if __name__=="__main__":
    main()
```

図3 Python/Tkinter/EPICSのサンプルスクリプト

で 別チャンネル,"fred"の状態を一秒毎に確認し、値に応じた表示を行っている[図4]。このプログラムのように、スレッド対応ライブラリを用いたユーザプログラムでは、制御情報の更新を意識する必要がない。また、GUI部分とその他の部分を別スレッドとすることで、ユーザからの入力受付などの操作がチャンネルの値のチェックなどの作業に及ぼす影響を最小にすることが出来る。

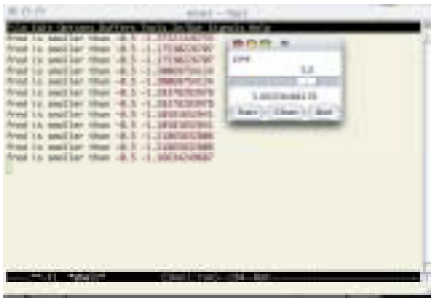

図4 サンプルプログラムの実行例

3.1. Tkinterモジュールの利用上の注意。

PythonインタプリタもTkinterが利用するTcl/Tkインタプリタも複数のスレッドを利用することができる。しかしながら、これらのインタプリタでは、それぞれのインタプリタ本体の実行は一時には一つのスレッドに制限されている。このため、Python中で TkinterモジュールのWidgetを使ったGUIアプリケーションを作成する際には、Tkinter/Widgetの作成・操作はPythonのメインのスレッドにのみ制限されることに注意が必要である。

Python-EPICS CAのコールバック関数の評価の際にも、C関数レベルのコールバック自体はPythonインタプリタの状態とは無関係に実行が開始されるが、コールバック関数のPython部分の実行のためにはこのスレッドがPythonインタプリタのグローバル・インタプリタ・ロック(GIL)を取得する必要がある。このためにコールバック関数の実行が遅延される可能性があることに注意しておく必要がある。 図５に

[3] オプションca_disable_preemptive_callbackを指定すると、CAライブラリーはバージョン3.13以前のライブラリーとほぼ同等に振る舞う。

[4] EPICS/CAでは制御情報は固有の名前で区別される。これをチャンネルと呼んでいる。

Python/Tkinter/EPICS-CAとグローバル・インタプリタ・ロックとの関係を示す。スレッド内部でTkライブラリあるいはEPICS CAライブラリに制御が移る際にはPythonインタプリタのGILはいったん開放される。これによってPythonスクリプトの実行を待っている別スレッドはGILを取得し実行を継続する。これによってスレッド間の並列動作が可能となっている。

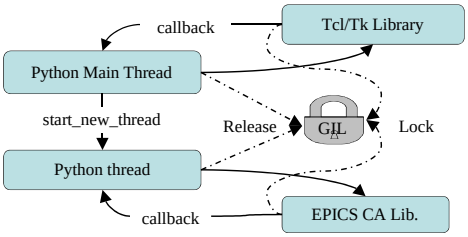

図5．Python**インタプリタのグローバルインタプリタロック。** Tk**および** CA**のライブラリが実行されている間は** Python**のインタプリタロックは一旦リリースされている。** callback**の実行など** Python**インタプリタに制御が戻る際には、再度インタプリタロックを取得する。**

4. Python**スクリプトによる**sequencer**の置き換え**

EPICSの標準ツールの一つにSequencerがある。Sequencerは状態遷移モデルに基づいて機器制御ロジックを実装するためのツールである。このツールは、機器の状態と状態遷移の条件およびその際に実行するべき動作を SNL(Sequence Notation Language) にしたがって記述し、これをSNC(Sequence Notation compiler)で処理することで、実行形式プログラムを生成する。図６にプログラム例を示す。Sequencerは最終的にフロントエンド計算機(IOC)あるいはホスト計算機（OPI）で実行可能なバイナリコードを生成するので、非常に高速なロジックを実現可能である。

Python/EPICS CAのマルチスレッド化により、Pythonスクリプトのチャンネルデータの変化に対する応答はそれまでと比べ大きく改善された。図６のSequencerプログラムとほぼ同等のPythonプログラムは図７に示したような物になる。ホスト計算機上でこれらのプログラムを実行した結果によれば、Python/CAによるプログラムはSequencerによるプログラムと同等あるいはそれ以上の応答を示している。MacOSX10.3.7でのテストでは1000 回の繰り返しに対しsequencer では約2秒、Pythonスクリプトでは約4秒の経過時間が必要であった。Python言語によるプログラム開発の容易さを考慮すると、制御システム開発の初期段階においてはsequencer プログラムを十分に代用することができると考えてよい。

```
program level_check
float v;
assign v to "Input_voltage";
monitor v;

short light;
assign light to "Indicator_light";

ss volt_check {
  state light_off{
    when (v > 5.0) {/* turn light on */
      light = TRUE;
      pvPut(light);
    } state light_on
  }
  state light_on{
    when (v < 5.0) {/* turn light off */
      light = FALSE;
      pvPut(light);
    } state light_off
  }
}
```

図6. SNL**による** Sequencer**プログラムの例**
（ SNL/SNC**マニュアルより**）

```
import ca, time, random
voltage="Input_voltage"
indicator ="Indicator_light"
up = threading.Lock()
dn = threading.Lock()
up.acquire()
dn.acquire()

def light_on():
  while 1:
    up.acquire()
    ca.Put(indicator , 1.0)
    ca.flush_io()

def light_off():
  while 1:
    dn.acquire()
    ca.Put(indicator , 0.0)
    ca.flush_io()

def val_monitor(ch, val):
  if val[0] <= 0.5:
    up.release()
  else:
    dn.release()

def main():
  global lon, loff
  lon = threading.Thread(None,\
        light_on, "Light On", ())
  loff = threading.Thread(None,\
        light_off, "Light Off", ())
  lon.start()
  loff.start()
  ca.Monitor(voltage, val_monitor)
```

図7：Python thread**を用いた状態遷移に基づく制御プログラムの例**

5. 結論

EPICSに基づく制御システムでのスクリプト言語の使用は開発効率の向上に有効である。スクリプト言語PythonをEPICS CAライブラリをマルチスレッド対応とすることで、実行効率の大幅な改善が期待される。実行効率の向上はまた、これまでスクリプト言語以外のツールを使わざるを得なかった分野(EPICS sequencer) をも代替え可能とした。EPICS-Pythonインタフェイスには我々が開発した物の他にもFNALで開発されたバージョンが存在する[5]。このFNALバージョンはPythonスレッドと矛盾はしないが、ca_disable_preemptive_callbackの状態で使われており、EPICS CAライブラリのマルチスレッド化の恩恵を完全には活かしきれていない。

CPU/Networkの高速化と、応用アプリケーションの迅速な開発に対する要求の高まりは今後も制御システム開発に置けるスクリプト言語の重要性を高めて行くだろう。

参考文献

[1] "EPICS Home on WWW", http://www.aps.anl.gov/epics/
[2] "EDM: Extensible Display Manger", John Sinclair, 2002,ORNL; " MEDM Reference Manual", Kenneth Evans, 2005, Argonne National Laboratory, Illinois, USA.
[3] "DEVELOPMENT OF THE PYTHON/TK WIDGETS FOR THE CONTROL SYSTEM BASED ON EPICS", T. T. Nakamura, T. Katoh, N. Yamamoto, Proceedings of EPAC 2000, Vienna, Austria; "DISPLAYING AN EPICSWAVEFORM DATA AS AN IMAGE IN PYTHON/TKINTER", N. Yamamoto, 」ɪProceedings of ICALEPCS2003, Gyeongju, Korea.
[4] A. Akiyama et al., Contributed to IEEE Particle Accelerator Conference (PAC 99), New York, 29 Mar - 2 Apr 1999.
[5]_" Using EPICS Channel Access from Python ", http://www-d0online.fnal.gov/www/groups/ctl/epics/epics_python.html