EPICS

Writing Channel Access Clients

Kazuro Furukawa, KEK

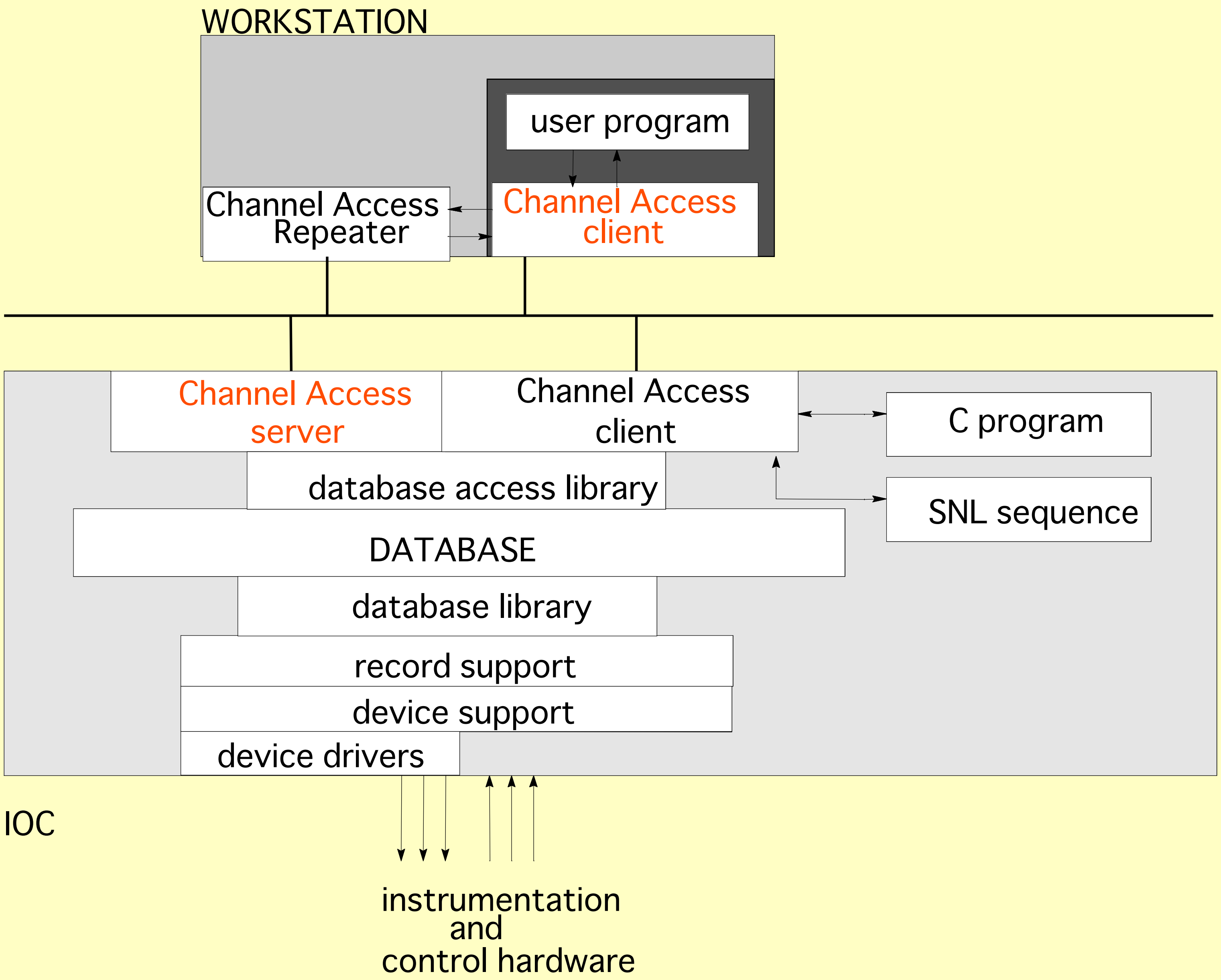
(Marty Kraimer, APS, USPAS1999)

(Bob Dalesio, LANL, USPAS2003)

References

- EPICS R3.12 / R3.14 Channel Access Reference Manual
  詳細なドキュメント
- cadef.h caerr.h - CA の基本インターフェースの記述
- db_access.h
  データの定義、分かりにくいがロバストなソフトウェアを書くためには重要
- LANL にあるチュートリアルも分かりやすい
  EPICS Web ページからたどることができる

CA between IOC and OPI



Overview of Talk

- クライアントソフトウェアの例題の紹介

  CA API/Macro の簡単な使用例

  CA Callback の使用例

  （db_access.h の詳細は省く）

SEVCHK

```
SEVCHK(<function call>,"message")
```

リターンコードを検査する Macro

もしエラーがあればメッセージを表示して終了する

以下の使用例で使う、試験には便利

実用ソフトウェアでは使用するべきではない

Very Simple Example

```
/*caSimpleExample.c*/
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "cadef.h"
main(int argc,char **argv)
{
    double       data;
    chid mychid;
    if(argc != 2) {
         fprintf(stderr,"usage: caExample pvname\n");
         exit(1);
    }
    SEVCHK(ca_task_initialize(),"ca_task_initialize");
    SEVCHK(ca_search(argv[1],&mychid),"ca_search failure");
    SEVCHK(ca_pend_io(5.0),"ca_pend_io failure");
    SEVCHK(ca_get(DBR_DOUBLE,mychid,(void *)&data),"ca_get failure");
    SEVCHK(ca_pend_io(5.0),"ca_pend_io failure");
    printf("%s %f\n",argv[1],data);
    return(0);
}
```

caExample

```
/*from stdin read list of PVs to monitor*/
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <cadef.h>
#define MAX_PV 1000
#define MAX_PV_NAME_LEN 40
typedef struct{
    char         value[20];
    chid         mychid;
    evid         myevid;
} MYNODE;
```

- Channel Access に関する宣言等
- Stdin から Process Variable (Record) 名のリストを読み込み、処理を行う例題

CA macros

```
static void printChidInfo(chid chid, char *message)
{
    printf("\n%s\n",message);
    printf("pv: %s  type(%d) nelements(%d) host(%s)",
       ca_name(chid),ca_field_type(chid),
       ca_element_count(chid),
       ca_host_name(chid));
    printf(" read(%d) write(%d) state(%d)\n",
       ca_read_access(chid),ca_write_access(chid),
       ca_state(chid));
}
```

- chid (Channel ID) を指定すると次の情報が取得できる
    - ca_name - name
    - ca_field_type - type as defined in db_access.h
    - ca_element_count - array size (1 for scalars)
    - ca_host_name - INET name of host
    - ca_read_access - Is read access allowed
    - ca_write_access - Is write access allowed
    - ca_state - connected, not connected, etc.

exception/connection callbacks

```
static void exceptionCallback(
   struct exception_handler_args args)
{
    chid chid = args.chid;
    MYNODE       *pnode = (MYNODE *)ca_puser(chid);
    long type = args.type;/*type of value returned*/
    long count = args.count;
    long stat = args.stat;
    printChidInfo(chid,"exceptionCallback");
    printf("type(%d) count(%d) stat(%d)\n",type,count,stat);
}
static void connectionCallback(struct connection_handler_args args)
{
    chid chid = args.chid;
    MYNODE       *pnode = (MYNODE *)ca_puser(chid);
    printChidInfo(chid,"connectionCallback");
}
```

- exceptionCallback
    - 以下の callback 以外のイベントが起った場合に呼ばれる
    - Ioc で発生した Error など
- connectionCallback
    - connect/disconnect が発生する毎に呼ばれる

accessRightsCallback

```
static void accessRightsCallback(
  struct access_rights_handler_args args)
{
    chid     chid = args.chid;
    MYNODE   *pnode = (MYNODE *)ca_puser(chid);
    printChidInfo(chid,"accessRightsCallback");
}
```

- Connect 時
- access rights が変更になった時

eventCallback

```
static void eventCallback(
   struct event_handler_args eha)
{
    chid         chid = eha.chid;
    MYNODE       *pnode = (MYNODE *)ca_puser(chid);
    long         type = eha.type;
    long         count = eha.count;
    if(eha.status!=ECA_NORMAL) {
      printChidInfo(chid,"eventCallback");
    } else {
      char *pdata = (char *)eha.dbr;
      printf("Event Callback: %s = %s\n",
        ca_name(eha.chid),pdata);
    }
}
```

- Monitor のイベントが発生した時

main - start

```
main()
{
    int         npv = 0;
    MYNODE      *pnode;
    MYNODE      *pmynode[MAX_PV];
    char        *pname[MAX_PV];
    int         i, status;
    char        tempStr[MAX_PV_NAME_LEN];
    char        *pstr;
    while(1) {
      if(npv >= MAX_PV ) break;
      pstr = fgets(tempStr,MAX_PV_NAME_LEN,stdin);
      if(!pstr) break;
      if(strlen(pstr) <=1) continue;
      pstr[strlen(pstr)-1] = '\0'; /*strip off newline*/
      pname[npv] = calloc(1,strlen(pstr) + 1);
      strcpy(pname[npv],pstr);
      pmynode[npv] = (MYNODE *)calloc(1,sizeof(MYNODE));
      npv++;
    }
```

- Stdin から Process Variable (Record) 名のリストを読み込む
  caExample < file

Actual CA calls

```
   SEVCHK(ca_task_initialize(),
        "ca_task_initialize");
   SEVCHK(ca_add_exception_event(
       exceptionCallback,NULL),
       "ca_add_exception_event");
   for(i=0; i<npv; i++) {
      SEVCHK(ca_search_and_connect(
       pname[i],&pmynode[i]->mychid,
       connectionCallback,&pmynode[i]),
       "ca_search_and_connect");
      SEVCHK(ca_replace_access_rights_event(
       pmynode[i]->mychid,
       accessRightsCallback),
       "ca_replace_access_rights_event");
      SEVCHK(ca_add_event(DBR_STRING,
       pmynode[i]->mychid,eventCallback,
       pmynode[i],&pmynode[i]->myevid),
       "ca_add_event");
    }
    /* 正常動作時にはこれ以下には到達しない */
    SEVCHK(ca_pend_event(0.0),"ca_pend_event");
    ca_task_exit();
}
```

Start and End

- **ca_task_initialize**

  ca_repeater との接続などの処理
  (connection management)

- **ca_add_exception_event**

  CA に異常が起こった場合の処理を行うために Callback を指定しておく

- {その他の Code}

- **ca_task_exit**

  CA に関連して確保した資源を解放する

Search

-
  ```
  ca_search_and_connect(name,pchid,
      connectionCallback,userarg)
  ca_replace_access_rights_event(chid,
      accessRightsCallback)
  ```
  - 要求は Buffer がいっぱいになるか ca_pend or ca_flush が実行されるまで Buffer される
  - UDP ブロードキャストで Process Variable を探索要求を出す
    - （EPICS_CA_AUTO_ADDR_LIST が YES または未設定の場合）
    - サブネット上の IOC はその Process Variable を収容していれば、要求に答える
  - Process Variable を収容している IOC に対して TCP 接続を確立する
  - connectionCallback は接続状態が変化すると呼ばれる
  - accessRightsCallback は Access Right が変更されると呼ばれる
  - chid に対して対応する userarg を使うことができる

I/O

- Puts - 多数のバリエーションがある

  -
    ```
    ca_array_put(type,count,chid,pvalues)
    ...
    ca_flush_io()
    ```
    Calls are buffered until: buffer full, ca_flush, or ca_pend.

  -
    ```
    ca_put_callback(type,count,chid,
     pvalue,putCallback,userarg)
    ```
    putCallback called after all records processed because of put complete processing.

- Gets - 同様に多数のバリエーションがある

  -
    ```
    ca_array_get(type,count,chid,pvalues)
    ...
    ca_pend_io(timeout)
    ```

  -
    ```
    ca_array_get_callback(type,count,chid,getCallback,user
    arg)
    ...
    ca_pend_event(timeout)
    ```

I/O continued

- Monitors - 多数のバリエーションがある
  - `ca_add_masked_array_event(type,count,`
    `chid,eventCallback,userarg,`
    `pevid,mask)`

    Call this once for each channel to be monitored.
    Mask allows value changes, alarm changes, archival changes
  - …
  - `ca_pend_event(timeout)`

    Waits at least timeout seconds

Waiting

- `ca_flush_io()`
  通常 ca_pend から呼ばれる
  udp/tcp の Buffer を送り出す
- `ca_pend_io(timeout)`
  ca_flush_io を呼ぶ。未処理の ca_gets や ca_search が終了するか、timeout になるまで待つ
- `ca_pend_event(timeout)`
  イベントが発生するか timeout になるまで待つ
- timeout
  - 0 は無限に待つという意味
  - .0001 などの短い時間を指定すれば、polling を行うことに使える

CA with X

- Channel Access uses select() to wait.
- File Descriptor Manager can be used.
- Channel access provides ca_add_fd_registration
- X provides similar facility

db_access.h

- Describes the data CA can transfer
- Hard to understand and use
- Provides access to
  - data types:
    string, char, short, long, float, double
  - status, severity, time stamp
  - arrays
  - enums (in ioc both menus and DBF_ENUM fields)
  - complete set of enum choices
  - control, display, alarm limits
  - Alarm Acknowledgment

ezCa - Easy Channel Access

- Goals
  - Easy to use.
  - Provide non-callback synchronous model.
- Data Types
  - `ezcaByte, ezcaString, ezcaShort, ezcaLong, ezcaFloat, ezcaDouble`
- Basic Calls
  - `int ezcaGet(pvname,  type,  nelem, buff)`
  - `int ezcaPut(pvname,  type,  nelem, buff)`
  - `int ezcaGetWithStatus(pvname,type, nelem,buff,time,stat,sevr)`
- Synchronous Groups
  - `int ezcaStartGroup(void)`
  - `any combination of get and put`
  - `int ezcaEndGroup(void)`

ezCa continued

- Error Handling
  - `ezcaPerror(message)`
  - `ezcaGetErrorString(message,errorstring)`
  - `ezcaFreeErrorString(errorstring)`
- Other Groupable Functions
  - `int ezcaGetControlLimits(pvname,type,low,high)`
  - `int ezcaGetGraphicLimits(pvname,type,low,high)`
  - `int ezcaGetNelem(pvname,nelem)`
  - `int ezcaGetPrecision(pvname,precision)`
  - `int ezcaGetStatus(pvname,time,stat,sevr)`
  - `int ezcaGetUnits(pvname,units)`
- Monitor Functions
  - `int ezcaSetMonitor(pvname,type)`
  - `int ezcaClearMonitor(pvname,type)`
  - `int ezcaNewMonitor(pvname,type)`
- Others

Starting Your Practice

- mkdir test1
- cd test1
- setenv HOST_ARCH \`$EPICS_BASE/../startup/HostArch\`
  - HOST_ARCH=\`$EPICS_BASE/../startup/HostArch\`
    export HOST_ARCH
- (USER=\`whoami\` ; export USER)
- makeBaseApp.pl -t simple test1
- cd test1App/src
- create source code
- edit Makefile.Host
- gmake (make)
  - Souce file が一つ (xxxx.c) であれば、
    - cd O.linux (or O.solaris, etc)
    - make xxxx

- gmake caExample
- caExample ffred

Practice Explanation 1

-
  ```
  HOST_ARCH=`$EPICS_BASE/../startup/HostArch`
  export HOST_ARCH
  ```

  assigning a platform name for EPICS software

  (backquotes around "$EPICS … HostArch" mean

  "execute it and use the result")

-
  ```
  USER=`whoami` ; export USER
  ```

  assigning a user name for EPICS software

-
  ```
  mkdir test1 ; cd test1
  ```

  making directory for our test

  and going into it

-
  ```
  makeBaseApp.pl -t example test1
  ```

  creating environment (directory and config files)

  for a new EPICS application

  see the manual "EPICS IOC Applications Building and Source Release Control"

Practice Explanation 2

- **`cd test1App/src`**
- **適当な単純な `sample code` を作る**
- **`gmake (make)`**

  makeBaseApp.pl が用意した Makefile に従って EPICS 環境を準備する

- **`cd O.linux ; gmake xxxx`**

  もしも Makefile.Host を適当に変更してあればこの Step は必要ない

- **`xxxx`**

  プログラムを実行してみる

Data Type Conversions in Channel Access

DBR

_STRING, _DOUBLE, _FLOAT, _LONG, _CHAR, _ENUM

Data type conversions are performed in the server

Endian and floating point conversions are done in the client

Polite clients requests data in native type and perform necessary conversion on the client side

Accessing Composite Data Structures

Many fields are fetched from the data store in one access:

```
struct dbr_ctrl_float       data;
struct dbr_ctrl_float       *pdata = &data;
ca_get(DBR_CTRL_FLOAT,mychid,(void *)pdata);
printf(“%d %d\n”,pdata->status, pdata->severity);
printf(“%d %d\n”,pdata->stamp.secPastEpoch, pdata->stamp.nsec);
printf(“%f %f\n”,pdata->high_display_limit,pdata->low_display_limit);
printf(“%f %f\n”,pdata->high_warning_limit,pdata->low_warning_limit);
printf(“%f %f\n”,pdata->high_alarm_limit,pdata->low_alarm_limit);
printf(“%f %f\n”,pdata->high_control_limit,pdata->low_control_limit);
printf(“%f %s\n”,pdata->value, pdata->units);
```

*Refer to db_access.h for structures...

Error Checking

- Error codes and error related macros are in caerr.h
- SEVCHK will exit on errors it deems irrecoverable
- ECA_NORMAL means the exchange was initiated successfully
- SEVCHK exit behavior can be replaced with your own exception handler
  ca_add_exception_event(…..)
- example:
  status = ca_array_put(data_type,channel_id,pvalue);
  SEVCHK(status,”additional info in error message”);

Caching vs. Queuing

- An event handler can either take its actions in the event handler -

  queuing

  all data is handled

  degradation mode is longer delays

- Place data into an intermediate buffer and have an alternate thread handle the data

  caching

  data can be overwritten

  degradation mode is intermediate data is discarded

- note that buffer in IOC will overwrite the last monitor on the queue when a buffer overflows in the IOC

Channel Access Notes

- ca_repeater needs to be run once on each workstation
- in the database,
  - a deadband of 0 posts a monitor on any change
  - a deadband of -1 posts monitors on every scan
- R3.15 may limit monitor events
- read cadef.h, caerr.h and db_access.h before writing a channel access client
- it is most efficient to use native data types and handle data conversions in the client program