Draft of 7/28/2003

IBP Network Function Unit (NFU) API specification

Yong Zheng

Name:

IBP_nfu_op -- Invoke an IBP NFU function at remote depot.

Signature:

```
#include <ibp.h>
#include <ibp_nfu.h>

int  IBP_nfu_op ( depot , opcode , nPara , *paras , timeout)

IBP_depot         depot;
int               opcode;
int               nPara;
PARAMETER         *paras;
IBP_timer         timeout ;
```

Description:

IBP_nfu_op is used to invoke an IBP NFU function at remote depot. *depot* is a structure (defined below) to specify the hostname and port number of the remote server on which the IBP depot is running.

```
typedef struct ibp_depot {
        #define IBP_MAX_HOSTNAME_LEN  256
         char    host[IBP_MAX_HOSTNAME_LEN;
         int      port;
} *IBP_depot;
```

opcode is used to specify the function to be invoked on IBP depot. *opcode* is a globally allocated integer which is assigned to each function supported by IBP depot.

paras argument is a pointer to array of structure PARAMETER (define below) with length of *nPara* . PARAMETER is a generic structure to descript the parameters used by NFU function.

```
 typedef  enum { IBP_REF_RD, IBP_REF_WR,IBP_REF_RDWR
                  IBP_VAL_IN,IBP_VAL_OUT,IBP_VAL_INOUT}  IOTYPE;
typedef struct {
     IOTYPE     ioType;
     void       *data;
     int        offset;
     int        length;
} PARAMETER;
```

There are two groups of parameters: call-by-reference parameters and call-by-value parameters. For call-by-reference parameter, the user uses IBP capabilities as parameters. At the depot side, the NFU function reads/writes the data from/to the capabilities depends on if the parameters are read parameter or write parameters. For call-by-reference parameters, all the capabilities must be local to the depot. For call-by-value parameter, user defined data are sent to the depot for input parameters and results are sent back from the depot for output parameters. It's user's responsibility to interpret immediate parameter correctly, eg. integer data should be passed in the network order. Following are detail descriptions of all parameter types:

IBP_REF_RD: It's call-by-reference parameter. The depot will map the data from the capability specified by *PARAMETER.data* at *PARAMETER.offset* with *PARAMETER.length* to a read only memory and pass the pointer of the mapped region and length to the underneath NFU function. All the modification made by the NFU function does not effect the underneath capability. A read capability must be given for IBP_REF_RD parameter.

IBP_REF_WR: It's call-by-reference parameter. The depot will map the data from the capability specified by *PARAMETER.data* at *PARAMETER.offset* with *PARAMETER.length* to a write only memory and pass the pointer of the mapped region and length to the underneath NFU function. The modification made by the NFU in the mapped region also change the underneath capability. A write capability must be given for IBP_REF_WR parameter.

IBP_REF_RDWR: It's call-by-reference parameter. The depot will map the data from the capability specified by *PARAMETER.data* at *PARAMETER.offset* with *PARAMETER.length* to a read-and-write memory and pass the pointer of the mapped region and length to the underneath NFU function. The modification made by the NFU in the mapped region also change the underneath capability. A write capability must be given for IBP_REF_RDWR parameter.

IBP_VAL_IN: It's call-by-value parameter. IBP_nfu_op sends the user defined data specified by *PARAMETER.data* with *PARAMETER.length* to the depot and the depot just pass it to the underneath NFU function as a read only buffer.

IBP_VAL_OUT: It's call-by-value parameter. The user set *PARAMETER.length* to tell the depot the maximum length of the data he/she wants to receive for this parameter. Upon the success, the function overwrites *PARAMETER.length* to the actual length of the data returned from the depot.

IBP_VAL_INOUT: Like IBP_VAL_IN, user defined data specified by *PARAMETER.data* with *PARAMETER.length* is sent to the depot. Upon the success, the data is returned from the depot with *PARAMETER.length.*

IBP_REF_RD, IBP_REF_WR and IBP_REF_RDWR are used for data access protection. IBP_VAL_IN, IBP_VAL_OUT and IBP_VAL_INOUT are used for data movement optimization between the client and the server.

The user can use the same capability for different type parameters. For example, the same capability can be used for IBP_REF_RD and IBP_REF_WR parameters for one NFU call.

Timeout is used to specify the maximum time that user waits for the result being returned.

Return Value:

On success, IBP_nfu_op will return IBP_OK. On error, the error number is returned and also IBP_errno is set.

Errors:

IBP_E_NFU_UNKNOWN	Unknown NFU function.
IBP_E_NFU_UNKNOWN_PARA_IOTYPE	Invalid NFU parameter type.
IBP_E_NFU_PARA_MISMATCH	Mismatched NFU parameter type.
Function specific error number	(function specific error number is the error number return by Depot NFU function plus 10000) .

Draft of 7/28/2003

IBP Depot Side NFU Function API Specification.

Definition:

```
#include "ibp_nfu_para.h"

typedef  int  (*OP) ( int  nPara, NFU_PARA *paras );
```

Description:

To generalize the depot side NFU function interface, we define a function type *OP* as described above. All the NFU function must comply with the signature of OP. *para* is an array of NFU_PARA structure (defined below) with length *nPara*.

```
typedef  enum {NFU_IN, NFU_OUT,NFU_INOUT} NFU_IO_TYPE;
typedef struct {
     NFU_IO_TYPE type;
     void *data;
     int     len;
} NFU_PARA;
```

When the depot receives a NFU request, first it searches the appropriate NFU function pointer. Upon success, the depot will create an array of NFU_PARA. For each parameter, IBP depot maps data from capability to memory and assigns the pointer of the mapped region to *NFU_PARA.data* (as we mention in NFU API specification). Furthermore it set the *NFU_PARA.len* with the value received from the client and set the appropriate value for *NFU_PARA.type*, then calls the NFU function. After NFU function finishes, the depot unmaps all the mapped memory and sends the return value of the function back to the client. The order of parameters passed to the NFU function is as same as the order that the depot receives from the client.

It's NFU function's responsibility to check if the parameters passed in match the function signature. For IBP_REF_RD and IBP_VAL_IN parameters (client side), the depot will pass the parameters as NFU_IN to the underneath functions; for IBP_REF_WR and IBP_VAL_OUT, it's NFU_OUT; for IBP_REF_RDWR and IBP_VAL_INOUT, it's NFU_INOUT.

Return Value:

On success, 0 is returned; otherwise the error number is returned.

Note:

To add NFU functions into IBP depot, first the developers need to compile and build shared libraries of the NFU functions. Then the admin of depot need to modify NFU config file (nfu.cfg which defined below) to add all new NFU functions. When the depot starts, it will read the nfu config file and load all the NFU functions into memory. Or the admin can send a HUP signal to the running depot. The depot will read the NFU config file again and load all the new added NFU functions.

Each line in the NFU config file defines a NFU function. The format of each line is defined as following :

opcode function-name shared-library-name

opcode is a globally allocated number assigned to each NFU function. *function-name* is the name of NFU function. *shared-library-name* is the shared library name with full path which contains the NFU function . Following is a sample:

0x00000001 do_xor /usr/lib/libnfu.so

Sample:

To illustrate how to use NFU API and how to develop depot side NFU function, we implemented a XOR function as following:

<<< Server Side >>>>

nfu.cfg :

```
    0x00000001  do_xor   /usr/lib/libnfu.so
```

nfu_xor.c:

```
#include <stdio.h>
#include "ibp_nfu_para.h"

/*
 *  Para2 = Para0 xor Para1
 */
int do_xor( int npara , NFU_PARA *paras){

   int i,j;
   char c;
   char *src1, *src2,*dst,*imd;

   /*
    * check parameters
    */
   if ( npara != 3 ){
      return (-1);
   };

   src1 = (char*)(paras[0].data);
   src2 = (char*)(paras[1].data);
   dst = (char*)(paras[2].data);
   for ( j=0, i = 0; i < paras[0].len ; i ++ ){
      dst[i] = src[i]^dst[j];
      j++;
      if ( j >= paras[1].len ){
         j = 0;
      };
   };

   return (0);
};
```

<<<   Client side   >>>>

nfu_test.c:

```
#include “ibp.h”
#include <stdio.h>
#include “ibp_nfu.h”

int main( ) {
    IBP_depot   depot;
   IBP_set_of_caps  ls_caps1, ls_caps2;
   struct ibp_timer   ls_timeout;
   PARAMETER  paras[5];
   char   key[]=”12344554545343432423232”;
   int   op_xor = 0x00000001;

   depot = (IBP_depot)calloc(sizeof(struct ibp_depot),1));
   strcpy(depot->host,”ibp.loci.cs.utk.edu”);
    depot->port = 6714;

   /*
    * allocate ls_caps1 and ls_caps2 and store data to these two capabilities
    *
    .
    .
    */

   /*
    * Test 1:  IBP_REF_RDWR and IBP_REF_RD parameters
    */
    paras[0].ioType = IBP_RED_RD;
    paras[0].data = ls_caps1->readCap;
    paras[0].offset = 0;
    paras[0].len = 600;


    paras[1].ioType = IBP_REF_RD;
    paras[1].data = ls_caps2->readCap;
    paras[1].offset = 0;
    paras[1].len = 50;
    paras[2].ioType = IBP_REF_RDWR;
    paras[2].data = ls_caps2->writeCap;
    paras[2].offset = 0;
    paras[2].len = 50;

    if ( IBP_nfu_run( depot,op_xor,3,paras,timeout) != IBP_OK ){
        fprintf(stderr,"Something Wrong in nfu_test !!!1\n");
        exit(-1);
    };


    /*
     *  Test 2:  IBP_REF_RDWR and IBP_VAL_IN parameters
     */
    paras[0].ioType = IBP_REF_RD;
    paras[0].data = ls_caps1->readCap;
    paras[0].offset = 0;
    paras[0].len = 600;


    paras[1].ioType = IBP_VAL_IN;
    paras[1].data = key;
    paras[1].offset = 0;
    paras[1].len = strlen(key);

    paras[2].ioType = IBP_REF_RDWR;
    paras[2].data = ls_caps1->writeCap;
    paras[2].offset = 0;
    paras[2].len = 600;



    if ( IBP_nfu_run( depot,op_xor,3,paras,timeout) != IBP_OK ){
        fprintf(stderr,"Something Wrong in nfu_test !!!1\n");
        exit(-1);
    };
    fprintf(stderr,"nfu_test is success !\n");

    return 0;
};
```