Simple Communication Buffer for BMDFM

simple_comm_buff.txt

~~~~~~~~~~~~~~~~~~~~~

```
In this example:
~~~~~~~~~~~~~~~~
    We create a C++ object in the Shared Memory Pool. This object
    implements a simple communication buffer:

              .-----------------------------------.
    Producer: | Buffer in the Shared Memory Pool: | Consumer:
      WR(str); |   - str_data                     |   str=RD();
    ---------->|   - WR_Sema4                     |----------->
               |   - RD_Sema4                     |
               |                                  |
               |   - WR(){                        |
               |       wait(WR_Sema4);            |
               |       str_data=str;              |
               |       post(RD_Sema4);            |
               |     }                            |
               |   - RD(){                        |
               |       wait(RD_Sema4);            |
               |       str=str_data;              |
               |       post(WR_Sema4);            |
               |     }                            |
               `-----------------------------------'

    BMDFM producer taskjob writes strings into the simple communication buffer
    and BMDFM consumer taskjob reads them from the simple communication buffer.


Files:
~~~~~~
    simple_comm_buff.txt - This file.

    shmem_stub.c         - Shared Memory Pool stub for fastlisp.

    simple_comm_buff.h   - Interface of the simple communication buffer.
    simple_comm_buff.cpp - Implementation of the simple communication buffer.

    cflp_udf.c           - Extension added to the BMDFM C/C++ interface.

    Makefile             - Makefile to build the example.

    simple_producer.flp  - Example of a simple producer.
    simple_consumer.flp  - Example of a simple consumer.


Output:
~~~~~~~
    $ fastlisp -q simple_consumer.flp

    [RunTimeErrCode=0] No Shared Memory Pool is available!

    $ fastlisp -q simple_producer.flp

    [RunTimeErrCode=0] No Shared Memory Pool is available!

    $ BMDFMsrv -d &
    [2] 11851

    $ ps
      PID TTY          TIME CMD
     2834 pts/0    00:00:00 bash
    11851 pts/0    00:00:00 BMDFMsrv
    11852 pts/0    00:00:00 PROCstat
    11853 pts/0    00:00:00 CPUPROC
    11854 pts/0    00:00:00 CPUPROC
    11855 pts/0    00:00:00 CPUPROC
    11856 pts/0    00:00:00 CPUPROC
    11857 pts/0    00:00:00 CPUPROC
    11858 pts/0    00:00:00 CPUPROC
    11859 pts/0    00:00:00 CPUPROC
    11860 pts/0    00:00:00 CPUPROC
    11861 pts/0    00:00:00 OQPROC
    11862 pts/0    00:00:00 OQPROC
    11863 pts/0    00:00:00 OQPROC
    11864 pts/0    00:00:00 OQPROC
    11865 pts/0    00:00:00 OQPROC
    11866 pts/0    00:00:00 OQPROC
    11867 pts/0    00:00:00 OQPROC
    11868 pts/0    00:00:00 OQPROC
    11869 pts/0    00:00:00 IORBPROC
    11870 pts/0    00:00:00 IORBPROC
    11871 pts/0    00:00:00 IORBPROC
    11872 pts/0    00:00:00 IORBPROC
    11873 pts/0    00:00:00 IORBPROC
    11874 pts/0    00:00:00 IORBPROC
    11875 pts/0    00:00:00 IORBPROC
    11876 pts/0    00:00:00 IORBPROC
    11877 pts/0    00:00:00 ps

    $ BMDFMldr -q simple_producer.flp &
    [3] 11878

    $ BMDFMldr -q simple_consumer.flp
    Hello, world 1 on CPU 7!
    Hello, world 2 on CPU 5!
    Hello, world 3 on CPU 3!
    Hello, world 4 on CPU 2!
    Hello, world 5 on CPU 1!
    Hello, world 6 on CPU 0!
    Hello, world 7 on CPU 7!
    Hello, world 8 on CPU 6!
    Hello, world 9 on CPU 5!
    Hello, world 10 on CPU 3!
    Hello, world 11 on CPU 2!
    Hello, world 12 on CPU 1!
    Hello, world 13 on CPU 0!
    Hello, world 14 on CPU 7!
    Hello, world 15 on CPU 6!
    Hello, world 16 on CPU 5!
    Hello, world 17 on CPU 3!
    Hello, world 18 on CPU 2!
    Hello, world 19 on CPU 1!
    Hello, world 20 on CPU 0!
    Hello, world 21 on CPU 7!
    Hello, world 22 on CPU 6!
    Hello, world 23 on CPU 5!
    Hello, world 24 on CPU 3!
    Hello, world 25 on CPU 2!
    Hello, world 26 on CPU 1!
    Hello, world 27 on CPU 0!
    Hello, world 28 on CPU 7!
    Hello, world 29 on CPU 6!
    Hello, world 30 on CPU 5!
    Hello, world 31 on CPU 3!
    Hello, world 32 on CPU 2!
    Hello, world 33 on CPU 1!
    Hello, world 34 on CPU 0!
    Hello, world 35 on CPU 7!
    Hello, world 36 on CPU 6!
    Hello, world 37 on CPU 5!
    Hello, world 38 on CPU 3!
    Hello, world 39 on CPU 2!
    Hello, world 40 on CPU 1!
    Hello, world 41 on CPU 0!
    Hello, world 42 on CPU 7!
    Hello, world 43 on CPU 6!
    Hello, world 44 on CPU 5!
    Hello, world 45 on CPU 3!
    Hello, world 46 on CPU 2!
    Hello, world 47 on CPU 1!
    Hello, world 48 on CPU 0!
    Hello, world 49 on CPU 7!
    Hello, world 50 on CPU 6!
    Hello, world 51 on CPU 5!
    Hello, world 52 on CPU 3!
    Hello, world 53 on CPU 2!
    Hello, world 54 on CPU 1!
    Hello, world 55 on CPU 0!
    Hello, world 56 on CPU 7!
    Hello, world 57 on CPU 6!
    Hello, world 58 on CPU 5!
    Hello, world 59 on CPU 3!
    Hello, world 60 on CPU 2!
    Hello, world 61 on CPU 1!
    Hello, world 62 on CPU 0!
    Hello, world 63 on CPU 7!
    Hello, world 64 on CPU 6!
    Hello, world 65 on CPU 5!
    Hello, world 66 on CPU 3!
    Hello, world 67 on CPU 2!
    Hello, world 68 on CPU 1!
    Hello, world 69 on CPU 0!
    Hello, world 70 on CPU 7!
    Hello, world 71 on CPU 6!
    Hello, world 72 on CPU 5!
    Hello, world 73 on CPU 3!
    Hello, world 74 on CPU 2!
    Hello, world 75 on CPU 1!
    Hello, world 76 on CPU 0!
    Hello, world 77 on CPU 7!
    Hello, world 78 on CPU 6!
    Hello, world 79 on CPU 5!
    Hello, world 80 on CPU 3!
    Hello, world 81 on CPU 2!
    Hello, world 82 on CPU 1!
    Hello, world 83 on CPU 0!
    Hello, world 84 on CPU 7!
    Hello, world 85 on CPU 6!
    Hello, world 86 on CPU 5!
    Hello, world 87 on CPU 3!
    Hello, world 88 on CPU 2!
    Hello, world 89 on CPU 1!
    Hello, world 90 on CPU 0!
    Hello, world 91 on CPU 7!
    Hello, world 92 on CPU 6!
    Hello, world 93 on CPU 5!
    Hello, world 94 on CPU 3!
    Hello, world 95 on CPU 2!
    Hello, world 96 on CPU 1!
    Hello, world 97 on CPU 0!
    Hello, world 98 on CPU 7!
    Producer done.

    Hello, world 99 on CPU 6!
    Hello, world 100 on CPU 5!
    Consumer done.

    [3]+  Done                    BMDFMldr -q simple_producer.flp

    $ echo command down down >/tmp/.BMDFMsrv_npipe

    $
    [2]+  Done                    BMDFMsrv -d
```

shmem_stub.c

~~~~~~~~~~~~~~

```
/* shmem_stub.c - a stub for the ShMem Pool
                  Shared Memory Pool stub for fastlisp. */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define CHR char
#define UCH unsigned char
```
```

```
#define SCH signed char
#define USH unsigned short int
#define SSH signed short int
#define ULO unsigned long int
#define SLO signed long int
#define DFL double

#ifdef __cplusplus
extern "C" {
#endif

void shmempool_on(void){
  return;
}

void shmempool_off(void){
  return;
}

UCH is_shmempool_on(void){
  return 0;
}

void *reallocpool(void *ptr, ULO size){
  return realloc(ptr,size);
}

void freepool(void *ptr){
  free(ptr);
  return;
}

CHR *shmempoolLUT_add_key_value(CHR **value, const CHR *key){
  return NULL;
}

CHR *shmempoolLUT_del_key_value(CHR **key){
  return NULL;
}

CHR *shmempoolLUT_get_value(CHR **value, const CHR *key){
  return NULL;
}

#ifdef __cplusplus
} // extern "C"
#endif
```

simple_comm_buff.h

~~~~~~~~~~~~~~~~~~~

```
/* simple_comm_buff.h - Simple Communication Buffer
                        Interface of the simple communication buffer. */

#ifndef _SIMPLE_COMM_BUFF_H
#define _SIMPLE_COMM_BUFF_H

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <semaphore.h>

#include "cflp_udf.h"

#ifdef __cplusplus
extern "C" {
#endif

class simple_comm_buff{
private:
  CHR *_simple_comm_buff;
  sem_t wr_sem;
  sem_t rd_sem;
public:
  simple_comm_buff();
  ~simple_comm_buff();
  void *operator new(size_t size);
  void *operator new[](size_t size);
  void operator delete(void *p);
  void operator delete[](void *p);
  CHR write(const CHR *str_data);
  CHR *read(CHR **str_data);
};

#ifdef __cplusplus
} // extern "C"
#endif

#endif /* simple_comm_buff.h */
```

simple_comm_buff.cpp

~~~~~~~~~~~~~~~~~~~~~~

```
/* simple_comm_buff.cpp - Simple Communication Buffer
                          Implementation of the simple communication buffer. */

#include "simple_comm_buff.h"

#ifdef __cplusplus
extern "C" {
#endif

extern void shmempool_on(void);
extern void shmempool_off(void);
extern UCH is_shmempool_on(void);
extern void *reallocpool(void *ptr, ULO size);
extern void freepool(void *ptr);

simple_comm_buff::simple_comm_buff(){
  _simple_comm_buff=NULL;
  sem_init(&wr_sem,1,1);
  sem_init(&rd_sem,1,0);
}

simple_comm_buff::~simple_comm_buff(){
  UCH shmempool;
  shmempool=is_shmempool_on();
  shmempool_on();
  free_string(&_simple_comm_buff);
  sem_destroy(&wr_sem);
  sem_destroy(&rd_sem);
  if(!shmempool)
    shmempool_off();
}

void *simple_comm_buff::operator new(size_t size){
  UCH shmempool;
  shmempool=is_shmempool_on();
  shmempool_on();
  void *ptr=reallocpool(NULL,(ULO)size);
  if(!shmempool)
    shmempool_off();
  return ptr;
}

void *simple_comm_buff::operator new[](size_t size){
  UCH shmempool;
  shmempool=is_shmempool_on();
  shmempool_on();
  void *ptr=reallocpool(NULL,(ULO)size);
  if(!shmempool)
    shmempool_off();
  return ptr;
}

void simple_comm_buff::operator delete(void *ptr){
  UCH shmempool;
  shmempool=is_shmempool_on();
  shmempool_on();
  freepool(ptr);
  if(!shmempool)
    shmempool_off();
  return;
}

void simple_comm_buff::operator delete[](void *ptr){
  UCH shmempool;
  shmempool=is_shmempool_on();
  shmempool_on();
  freepool(ptr);
  if(!shmempool)
    shmempool_off();
  return;
}

CHR simple_comm_buff::write(const CHR *str_data){
  UCH not_error=0,shmempool;
  if(str_data!=NULL){
     shmempool=is_shmempool_on();
     shmempool_on();
     while((not_error=noterror())&&sem_wait(&wr_sem));
     if(not_error){
       equ(&_simple_comm_buff,str_data);
       while(sem_post(&rd_sem));
     }
     if(!shmempool)
       shmempool_off();
  }
  return not_error;
}

CHR *simple_comm_buff::read(CHR **str_data){
  UCH not_error;
  CHR *ret_val=NULL;
  if(str_data!=NULL){
     while((not_error=noterror())&&sem_wait(&rd_sem));
     if(not_error){
       equ(str_data,_simple_comm_buff);
       while(sem_post(&wr_sem));
       ret_val=*str_data;
     }
  }
  return ret_val;
}

#ifdef __cplusplus
} // extern "C"
#endif
```

cflp_udf.c

~~~~~~~~~~

```
/* cflp_udf.c - FastLisp User Defined Functions written in C
                Sancho Mining 07-09-2000 20:51:42.51pm */

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#ifndef _NOT_UNIX_
#include <unistd.h>
#endif
#include <string.h>
#include "cflp_udf.h"
#include "simple_comm_buff.h"

#ifdef __cplusplus
extern "C" {
#endif

const CHR *VERSION_CFLPUDF___="Sancho M. CFLPUDF v.1.0.0.";

extern const ULO INSTRUCTIONS;
```
~~~~~~~~~~

```
/* ____________________________________________________________________ */
/* Functions                                                 SECTION 0 */
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */

class simple_comm_buff *Simple_Comm_Buff=NULL;

void simple_comm_buff_write(const ULO *dat_ptr, struct fastlisp_data *ret_dat){
  CHR *str_data=NULL;
  SLO sync;
  ret_sval(dat_ptr,&str_data);
  ret_ival(dat_ptr+1,&sync);
  if(noterror())
    if(Simple_Comm_Buff==NULL)
      rise_error_info(0,"No Shared Memory Pool is available!");
    else{
      ret_dat->single=1;
      ret_dat->type='I';
      ret_dat->value.ival=(SLO)Simple_Comm_Buff->write((const CHR*)str_data);
    }
  free_string(&str_data);
  return;
}

void simple_comm_buff_read(const ULO *dat_ptr, struct fastlisp_data *ret_dat){
  SLO sync;
  ret_dat->disable_ptr=1;
  ret_ival(dat_ptr,&sync);
  if(noterror())
    if(Simple_Comm_Buff==NULL)
      rise_error_info(0,"No Shared Memory Pool is available!");
    else{
      ret_dat->single=1;
      ret_dat->type='S';
      Simple_Comm_Buff->read(&ret_dat->svalue);
    }
  return;
}

/* ____________________________________________________________________ */
/* FastLisp Callbacks                                        SECTION 1 */
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */

extern void shmempool_on(void);
extern void shmempool_off(void);
extern UCH is_shmempool_on(void);
extern CHR *shmempoolLUT_add_key_value(CHR **value, const CHR *key);
extern CHR *shmempoolLUT_del_key_value(CHR **key);
extern CHR *shmempoolLUT_get_value(CHR **value, const CHR *key);

#define LUT_KEY_FOR_SIMPLE_COMM_BUFF "Simple Comm Buff"

void startup_callback(void){
  CHR *lut_key=NULL,*lut_value=NULL;
  UCH shmempool;
  if(am_I_in_the_BMDFMsrv_module()){
    Simple_Comm_Buff=new simple_comm_buff;
    shmempool=is_shmempool_on();
    shmempool_off();
    get_std_buff(&lut_key,LUT_KEY_FOR_SIMPLE_COMM_BUFF);
    equ_num(&lut_value,(SLO)Simple_Comm_Buff);
    shmempool_on();
    shmempoolLUT_add_key_value(&lut_value,lut_key);
    shmempool_off();
    free_string(&lut_key);
    free_string(&lut_value);
    if(shmempool)
      shmempool_on();
  }
  else
    if(am_I_in_the_CPUPROC_module()){
      shmempool=is_shmempool_on();
      shmempool_off();
      get_std_buff(&lut_key,LUT_KEY_FOR_SIMPLE_COMM_BUFF);
      shmempool_on();
      shmempoolLUT_get_value(&lut_value,lut_key);
      shmempool_off();
      Simple_Comm_Buff=(simple_comm_buff*)atol(lut_value);
      free_string(&lut_key);
      free_string(&lut_value);
      if(shmempool)
        shmempool_on();
    }
  return;
}

void taskjob_end_callback(ULO id_taskjob){
  CHR *lut_key=NULL;
  UCH shmempool;
  if(am_I_in_the_BMDFMsrv_module()){
    delete Simple_Comm_Buff;
    Simple_Comm_Buff=NULL;
    shmempool=is_shmempool_on();
    shmempool_off();
    get_std_buff(&lut_key,LUT_KEY_FOR_SIMPLE_COMM_BUFF);
    shmempool_on();
    shmempoolLUT_del_key_value(&lut_key);
    shmempool_off();
    free_string(&lut_key);
    if(shmempool)
      shmempool_on();
  }
  return;
}

/* The BMDFMldr module is capable of invoking/evaluating VM language
   expressions from C/C++ code (1-Capable;0-Unable).*/
UCH BMDFMldr_capable_call_VMcode_from_C=0;

void user_io_callback(SLO usr_id, CHR **usr_buff){
  /* This is just a stub. Place your own code here. */
  /* The following is a default behavior: */
  CHR *temp=NULL,*temp1=NULL,*temp2=NULL;
  equ(&temp,*usr_buff);
  if(cmp(temp,get_std_buff(&temp1,"PWD"))){
    mk_fst_buff(&temp1,4096);
    if(getcwd((char*)temp1,(size_t)len(temp1)))
      get_std_buff(usr_buff,temp1);
  }
  else
    if(cmp(head(&temp2,temp),get_std_buff(&temp1,"GetEnv"))){
      tail(&temp1,temp);
      get_std_buff(usr_buff,getenv(temp1));
    }
    else{
      lcat(usr_buff,get_std_buff(&temp,", usr_buff=\""));
      lcat(usr_buff,equ_num(&temp,usr_id));
      lcat(usr_buff,get_std_buff(&temp,"USER_IO: usr_id="));
      cat(usr_buff,get_std_buff(&temp,"\"."));
    }
  free_string(&temp);
  free_string(&temp1);
  free_string(&temp2);
  return;
}

/* ____________________________________________________________________ */
/* FastLisp Database Register                                SECTION 2 */
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */

INSTRUCTION_STRU INSTRUCTION_SET[]={
  {"SIMPLE_COMM_BUFF_WRITE",2,'I',(UCH*)"SI",&simple_comm_buff_write},
  {"SIMPLE_COMM_BUFF_READ", 1,'S',(UCH*)"I", &simple_comm_buff_read}
};
const ULO INSTRUCTIONS=sizeof(INSTRUCTION_SET)/sizeof(INSTRUCTION_STRU);

/* ____________________________________________________________________ */
/* Invocation of Function Main                               SECTION 3 */
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */

extern int _Main_(int argc, char *argv[]);

int main(int argc, char *argv[]){
  return _Main_(argc,argv);
}

#ifdef __cplusplus
} // extern "C"
#endif
```

Makefile

~~~~~~~~

```
# Makefile

#============================================================================
# x86-64 Linux, GNU C
#----------------------------------------------------------------------------
CC = g++ -m64
OPTIONS = -Wall -O3 -fPIC
#============================================================================

MATH = -lm
PSM = -lpthread

all: cflp_udf.o shmem_stub.o simple_comm_buff.o \
     fastlisp BMDFMldr BMDFMsrv CPUPROC

cflp_udf.o: shmem_stub.o simple_comm_buff.o cflp_udf.c cflp_udf.h
      $(CC) $(OPTIONS) -c cflp_udf.c

shmem_stub.o: shmem_stub.c
      $(CC) $(OPTIONS) -c shmem_stub.c

simple_comm_buff.o: cflp_udf.h simple_comm_buff.h simple_comm_buff.cpp
      $(CC) $(OPTIONS) -c simple_comm_buff.cpp

fastlisp: fastlisp.o cflp_udf.o shmem_stub.o simple_comm_buff.o
      $(CC) -o fastlisp fastlisp.o shmem_stub.o cflp_udf.o simple_comm_buff.o \
      $(MATH) $(PSM)

BMDFMldr: BMDFMldr.o cflp_udf.o simple_comm_buff.o
      $(CC) -o BMDFMldr BMDFMldr.o cflp_udf.o simple_comm_buff.o \
      $(MATH) $(PSM)

BMDFMsrv: BMDFMsrv.o cflp_udf.o simple_comm_buff.o
      $(CC) -o BMDFMsrv BMDFMsrv.o cflp_udf.o simple_comm_buff.o \
      $(MATH) $(PSM)

CPUPROC: CPUPROC.o cflp_udf.o simple_comm_buff.o
      $(CC) -o CPUPROC CPUPROC.o cflp_udf.o simple_comm_buff.o \
      $(MATH) $(PSM)

clean:
      rm -f cflp_udf.o shmem_stub.o simple_comm_buff.o \
      fastlisp BMDFMldr BMDFMsrv CPUPROC 2>/dev/null
```

simple_producer.flp

~~~~~~~~~~~~~~~~~~~~

```
# Example of a simple producer.


(setq sync 0)

(for i 1 1 100 (progn

  (setq str
    (str_fmt "Hello, world %s!\n" (cat i (cat " on CPU " (id_cpuproc))))
  )

  (setq sync (simple_comm_buff_write str sync))

  (if (! sync)
    (break)
    nil
  )

))
```

```
(setq sync (simple_comm_buff_write "" sync))

(cat "Producer done.\n" (space (& 0 sync)))
```

simple_consumer.flp

```
# Example of a simple consumer.


(setq sync "")

(while 1 (progn

  (setq sync
    (setq str (simple_comm_buff_read sync))
  )

  (if (len str)
    (outf "%s" str)
    (break)
  )

))

"Consumer done.\n"
```

 # <EOF>