IPPC: A LIBRARY FOR
INTER-PROGRAM PIPE COMMUNICATION

J.W. Bandler, Q.J. Zhang, G. Simpson and S.H. Chen

SOS-90-10-U

September 1990

© J.W. Bandler, Q.J. Zhang, G. Simpson and S.H. Chen 1990

No part of this document, computer program, source code, compiled code, related documentation and user manuals, magnetic tape, constituent subprograms, test programs, data and data files may be acquired, copied, reproduced, duplicated, executed, lent, disclosed, circulated, translated, transcribed or entered in any form into any machine without written permission. Address enquiries in this regard to Dr. J.W. Bandler. Neither the authors nor any other person, company, agency or institution make any warranty, express or implied, or assume any legal responsibility for the accuracy, completeness or usefulness of the material presented herein, or represent that its use would not infringe upon privately owned rights. This title page and original cover may not be separated from the contents of this document.

IPPC: A LIBRARY FOR INTER-PROGRAM PIPE COMMUNICATION

J.W. Bandler, Q.J. Zhang, G. Simpson and S.H. Chen

Abstract This document describes inter-program communication in the UNIX environment. Such communication is done with pipes as a vehicle for data transfer. A prototype utility library, IPPC, for such communication is established. The basic form of communication is between two programs, a parent program and a child program. IPPC also permits communication of one parent with several children and grandchildren. Examples are provided. The library has been tested on the Apollo, HP and SUN workstations.

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada under Grant STR0040923 and in part by Optimization Systems Associates Inc.

The authors are with the Simulation Optimization Systems Research Laboratory and the Department of Electrical and Computer Engineering, McMaster University, Hamilton, Canada L8S 4L7. J.W. Bandler, Q.J. Zhang and S.H. Chen are also with Optimization Systems Associates Inc., P.O. Box 8083, Dundas, Ontario, Canada L9H 5E7.

I. INTRODUCTION

The ability to interact between several isolated programs is vital in the development of any large scale software system. Recent software systems, e.g., CHORD [1] and Academy [2], have been made possible only by utilizing several separate programs each performing a specific task.

This report describes a library called IPPC which allows the use of pipes for data transfer between separate programs. The basic form of communication is between two programs, a parent program and a child program. IPPC also permits communication of one parent with several children and grandchildren. Examples are provided. The library has been tested on the Apollo DN3500 Domain/IX SR 9.7, HP 9000/340 HP-UX 6.5 and SPARCstation 1 SunOS 4.0.3c.

The source code list of the IPPC library can be found in [3].

II. LIST OF LIBRARY FUNCTIONS AND THEIR ARGUMENTS

Inclusion File

The user must include "ippc.h" in all files that use the IPPC library module. The following character strings are defined in this inclusion file:

pipe_open, pipe_close

pipe_read, pipe_write, pipe_read2, pipe_write2

pipe_initialize, pipe_initialize2, pipe_loop_stop

All strings listed above have the prefix "pipe_". These character strings are reserved for the library and should not be redefined by the user.

List of Library Functions

The library consists of the following functions (procedures) all of which are accessible by the user.

int pipe_open(char *cmd)

int pipe_close(int cid)

int pipe_initialize(int cid)

int pipe_read(char *buffer, int size, int n_items, int cid)

int pipe_write(char *buffer, int size, int n_items, int cid)

int pipe_initialize2()

int pipe_read2(char *buffer, int size, int n_items)

int pipe_write2(char *buffer, int size, int n_items)

For all the functions listed above, the return value is -1 if an error occurs. Also, for all the functions except pipe_open() and pipe_close(), the return value is 0 if no error is detected.

Description of Library Functions

pipe_open() — is a function to open the pipes and to start a child process. Internally it will open two pipes, one for data transfer from parent to child. The other pipe is for data transfer from child to parent. The return value of this function is the child identifier.

pipe_close() — is a function to close the pipes of a particular child. Internally it will signal the child to stop. It will close the two pipes opened by "pipe_open()". It will wait for the child process to finish and then return. The return value of the function is an integer whose lower-order eight bits encodes the system's idea of the child termination status: zero for normal termination and nonzero otherwise. This return value is identical to the output argument "status" in the wait() function as described in page 23 of [4].

pipe_read() — is to be used in the parent program to read data from the pipe.

pipe_write() is to be used in the parent program to write data to the pipe.

pipe_read2() is to be used in the child program to read data from the pipe.

pipe_write2() is to be used in the child program to write data to the pipe.

pipe_initialize() is to be used in the parent program to initialize the pipe communication. Its primary purpose is to synchronize the parent and child programs. A synchronization signal will be sent from the parent to the child. When a parent uses this function, the corresponding child must use the function pipe_initialize2() to respond.

pipe_initialize2() is to be used in the child program to initialize the pipe communication. It will force the child to wait until a synchronization signal from the parent is received. It will then determine whether the child should continue or exit. The continuation signal and the exit signal (both considered as synchronization signals) are sent from the parent through functions pipe_initialize() and pipe_close(), respectively.

Description of Arguments

cmd is the pointer to a character string. On entry to function pipe_open(), this character string should be the name of the executable file of the child program.

cid is a child identifier. Its value is determined by the return value of function pipe_open(). This value will be used as an input argument for functions pipe_read(), pipe_write(), pipe_close() and pipe_initialize(). Users should NOT alter the value of cid.

buffer is a pointer indicating storage location for data.

size is the size of an (data) item. It must be specified by the user when functions pipe_read(), pipe_write(), pipe_read2() and pipe_write2() are used.

n_items is the number of data items to be transferred. It must be specified by the user when functions pipe_read(), pipe_write(), pipe_read2() and pipe_write2() are used.

III. GENERAL USAGE

Parent Program, Non-iterative Case

The basic form of inter-program pipe communication is between two programs: a parent program and a child program, and the communication is not repetitive (i.e., the child is not called iteratively). The general usage of pipe communication in the parent program is:

```
cid = pipe_open("child_program");          /* open pipe and activate
                                              the child program */
pipe_initialize(cid);                       /* initialization */
pipe_write(buffer, size, n_item, cid);      /* send data to child.
                                              upon receiving all necessary
                                              data, the child program
                                              will start processing */
pipe_read(buffer, size, n_item, cid);       /* get data from child after
                                              data processing in the child
                                              program is finished */
pipe_close(cid);                            /* close the pipes */
```

Parent Program, Iterative Case

Suppose the child program is to be called iteratively by the parent program. Instead of loading the child program in each iteration, we want to activate the child program only once outside the loop in the parent program. The general usage of IPPC in the parent program is:

```
cid = pipe_open("child_program");           /* open pipe and activate
                                               the child program */
for (i = 0; i < 100; i++) {                 /*  iteration  in  the  parent
                                               program, e.g., 100
                                               iterations */
   pipe_initialize(cid);                    /* initialization */
   pipe_write(buffer, size, n_item, cid);   /* send data to child.
                                               upon receiving the data, the
                                               child program will start
                                               processing */
    pipe_read(buffer, size, n_item, cid);   /* get data from child after
                                               a complete iteration of
                                               data processing in the child
                                               program is finished */

   /* data processing in parent program here */

}
pipe_close(cid);  /* tell the child to exit and close the pipes */
```

Child Program, Iterative and Non-iterative Cases

The general usage of pipe communication in the child program is the same regardless of whether the parent calls the child iteratively or non-iteratively. We begin our discussion with the iterative case. The synchronization of iterations in the parent and the child will be realized using the property of pipe communications described as follows.

If a program (either the parent or the child) writes data to a pipe which is full, then the program will automatically wait until new space is available in the pipe. If a program reads from a pipe which has no data yet, the program will automatically wait until data is available in the pipe. The SPARCstation users manual [4] can be referred to for detailed description.

In the child program, the user should artificially set up an infinite loop so that the desired data processing in the child process will be repeated indefinitely as long as required data is available in the pipe. The execution of each repetition will depend upon the availability of a signal sent from the parent through the pipe. On the other hand the parent will decide when to send such a signal in each iteration of the parent program. In this way the two programs are automatically synchronized. The general usage of pipe communication in the child program is:

```
for (i = 0; ; i++) {                      /* set up an infinite loop */
      pipe_initialize2();                 /* initialize (synchronize with
                                          the parent) */
      pipe_read2(buffer, size, n_item);   /* get data from parent */

      /* data processing in child program starts here */
      ......
      /* end of data processing */

      pipe_write2(buffer, size, n_item);  /* send data to parent */
}
```

The above form is also valid for non-iterative case. It should be emphasized that the infinite loop cannot be neglected even when the parent wants to call the child only once. The main reason is that the synchronization signal will be transferred twice (i.e., more than once). The first signal is sent from pipe_initialize() in the parent program. The second one is sent from pipe_close() in the parent program.

Multi-Children and Multi-Generation Case

The IPPC library allows one parent program to communicate with several child and grandchild programs. Each child or grandchild is identified by a child identifier (the integer "cid" as listed in Section II). This identifier is given by the return value of function pipe_open() and used in pipe_read(), pipe_write(), pipe_close() and pipe_initialize(). Also, such identification of children is the sole responsibility of the child's immediate parent. Neither a grandparent nor a child will be asked to provide the parameter "cid". The maximum number of immediate children born from any parent and running concurrently is 127.

IV. EXAMPLES

The computers we used for testing the IPPC library are Apollo DN3500 Domain/IX SR 9.7, HP 9000/340 HP-UX 6.5 and SPARCstation 1 SunOS 4.0.3c.

Example 1: The Speed of Pipe Communication

The purpose of this example is to estimate the speed of data transfer in pipe communication. In the example 160K bytes of data are transferred for 500 iterations. The test programs are listed in Appendix A. "speed0.c" is the parent program. "speed1.c" is the child program.

In order to verify that data is correctly transferred, we added simple arithmetic manipulations into "speed0.c" and "speed1.c". We also performed the same arithmetic manipulations without using pipe communication in another program "speed.c". According to the output from the SPARCstation 1, the CPU times for running "speed.c" and "speed0.c" are 0 and 47 seconds, respectively. Therefore the speed of data transfer in pipe communication is approximately 1.7M bytes per second (160K * 500 iterations / 47 seconds). On the Apollo platform, the CPU time for running "speed0.c" is 433 seconds and the data transfer speed is approximately 0.2M bytes per second. On the HP platform, the CPU time is 104 seconds and the data transfer speed is 0.8M bytes per second.

Example 2: The Overhead CPU Cost of IPPC in Practical Situation

In a practical situation, number crunching is the most CPU intensive part in running a CAD program. In Example 2, some number crunching procedure is included. Therefore pipe communication consumes only a small portion of the overall CPU time. The test programs are listed in Appendix B. The numerical process in the child program is executed for 100 iterations, and in each iteration 80K bytes of data are transferred between the parent and the child. "prog0.c" is the parent program. "prog1.c" is the child program. "prog.c" is the equivalent program without pipe communication. The numerical output from running "prog.c" and "prog0.c" are identical. The CPU times for running "prog.c" and "prog0.c" on the SPARCstation are 257 and 261 seconds, respectively. Therefore the program's overall execution time using pipe communication is only 1.6% more than that without using pipes.

Since the Apollo and the HP computers are much slower than the SPARCstation, we reduced the number of iterations in this example from 100 on the SPARCstation to 10 on the Apollo and HP. The CPU times for running "prog.c" and "prog0.c" are 216 and 221 seconds on the Apollo, and 110 and 112 seconds on the HP, respectively. Therefore the overhead cost of IPPC are 2.3% and 1.9% on the Apollo and HP, respectively.

Example 3: A Multi-Child, Multi-Generation Example

In this example, there are 6 children and grandchildren related to one parent program. The source code of the programs is listed in Appendix C. "grand0.c" is the parent program. "grand1.c" to "grand6.c" are the child programs. Child #1 and Child #2 are called simultaneously. Child #3 is a grandchild, being born from Child #2. Child #4 is born after Children #1, #2 and #3 died. Children #5 and #6 are both grandchildren, both being born from Child #4.

An equivalent program, "grand.c", which does not use pipe communications, performs the same numerical operations as "grand0.c", and its six child programs combined. The numerical results from running "grand.c" and "grand0.c" are identical. This example also illustrates both the iterative and non-iterative cases of using IPPC.

V. REFERENCES

[1] J.R.F. McMacken and S.G. Chamberlain, "CHORD: a modular semiconductor device simulation development tool incorporating external network models", *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 826-836, 1989.

[2] *Academy*, EEsof Inc., 5795 Lindero Canyon Rd., Westlake Village, CA 91362.

[3] J.W. Bandler, Q.J. Zhang, G. Simpson and S.H. Chen, "IPPC: a library for inter-program pipe communication", Department of Electrical and Computer Engineering, McMaster University, Hamilton, Canada, Report SOS-90-10-L, 1990.

[4] *Programming Utilities and Libraries*, SPARCstation 1 Users Manual, Sun Microsystems Inc., 2550 Garcia Ave., Mountain View, CA 94043, pp. 21-26, May 1988.

APPENDIX A

SOURCE CODE LISTING OF EXAMPLE 1

```
/********************************************************
* File: speed0.c
* Date: Sep. 11, 1990
*
* Description: IPPC (inter-program pipe communication).
*              Example 1: test speed of IPPC.
*              Parent program.
*********************************************************/

#include <stdio.h>
#include <math.h>
#include <sys/time.h>
#include "ippc.h"

#define N          20000
#define NO_ERROR       0

struct timeval TpStart, TpFinish;
struct timezone Tzp;

main()
{
   int i, k, kk, error, cid;
   float a[N], b[N];
   long cputime;

   /* initialization */

   for (i = 0; i < N; i++) a[i] = (float) i;
   k = 0;
   gettimeofday(&TpStart, &Tzp);

   if (cid = pipe_open("speed1") != -1) {

      while (k < 500) {

            /* output array a[] and integer k to pipe
               to be read by child program */

            pipe_initialize(cid);
            pipe_write(a, sizeof(float), N, cid);
            pipe_write(&k, sizeof(int), 1, cid);

            /* retrieve array b[] and integer k sent from
               child program through pipe */

            pipe_read(b, sizeof(float), N, cid);
            pipe_read(&k, sizeof(int), 1, cid);

            /* process results */

            a[10000] = b[10000] - k;
            kk = k / 100;
            if((k - kk * 100) == 0)
                printf(" iteration #   %d  b[10000] = %f \n",k, b[10000]);
       }
   }
   else {
       printf ("\n Error in Pipe Communication\n");
   }

   /* stop child process and close pipe */

   pipe_close(cid);
   gettimeofday(&TpFinish, &Tzp);
   cputime = TpFinish.tv_sec - TpStart.tv_sec;
   printf("cpu time = %ld \n", cputime);
}
```

```
/*********************************************************
* File: speed1.c
* Date: Sep. 11, 1990
*
* Description: IPPC (inter-program pipe communication)
*              Example 1: test speed of IPPC.
*              Child program.
*********************************************************/

#include <stdio.h>
#include <math.h>
#include "ippc.h"

#define MAX_N_CALLS  30000
#define N            20000

main()
{
   int i, k, error;
   float a[N], b[N];
   int iteration = 0;

   while (iteration < MAX_N_CALLS) {
      iteration++;

      /* retrieve array a[] and integer k sent from
         parent program through pipe */

      pipe_initialize2();
      pipe_read2(a, sizeof(float), N);
      pipe_read2(&k, sizeof(int), 1);

      /* actual data processing and number crunching in child program */

      k++;
      b[10000] = 2.5 + a[10000];

      /* output array b[] and integer k to pipe
         to be read by parent program */

      pipe_write2(b, sizeof(float), N);
      pipe_write2(&k, sizeof(int),  1);
   }
}


/********************************************************
* File: speed.c
* Date: Aug. 28, 1990
*
* Description: IPPC (inter-program pipe communication)
*              Example 1: test speed of IPPC.
*              This program is functionally the same as
*              "speed0.c" and "speed1.c" combined except that
*              this program does not use IPPC.
********************************************************/

#include <stdio.h>
#include <math.h>
#include <sys/time.h>

#define N         20000
#define NO_ERROR      0

struct timeval TpStart, TpFinish;
struct timezone Tzp;

int subroutine();

main()
{
   int i, k, kk;
   float a[N], b[N];
   long cputime;

   /* initialization */
```

```
   for (i = 0; i < N; i++) a[i] = (float) i;
   k = 0;
   gettimeofday(&TpStart, &Tzp);

   while (k < 500) {
         subroutine(a, b, &k);

         /* process results */

         a[10000] = b[10000] - k;
         kk = k / 100;
         if((k - kk * 100) == 0)
              printf(" iteration #   %d  b[10000] = %f \n",k, b[10000]);
   }

   gettimeofday(&TpFinish, &Tzp);
   cputime = TpFinish.tv_sec - TpStart.tv_sec;
   printf("cpu time = %ld \n", cputime);
}

int subroutine(a, b, k)
float *a, *b;
int *k;
{
   *k = *k + 1;
   b[10000] = 2.5 + a[10000];
}
```

APPENDIX B

SOURCE CODE LISTING OF EXAMPLE 2

```
/*******************************************************
* File: prog0.c
* Date: Sep. 11, 1990
*
* Description: IPPC (inter-program pipe communication)
*              Example 2: Overhead cost of using IPPC.
*              Parent program.
********************************************************/

#include <stdio.h>
#include <math.h>
#include <sys/time.h>
#include "ippc.h"

#define N          10000
#define NO_ERROR       0

struct timeval TpStart, TpFinish;
struct timezone Tzp;

main()
{
   int i, ii, k, kk, cid;
   float a[N], b[N];
   long cputime;

   /* initialization */

   gettimeofday(&TpStart, &Tzp);
   for (i = 0; i < N; i++) a[i] = 2.0 * (float) i;
   k = 0;

   if (cid = pipe_open("prog1") != -1) {

      while (k < 100) {

           /* output array a[] and integer k to pipe */

            pipe_initialize(cid);
            pipe_write(a, sizeof(float), N, cid);
            pipe_write(&k, sizeof(int),  1, cid);

            /* retrieve array b[] and integer k from pipe */

            pipe_read(b, sizeof(float), N, cid);
            pipe_read(&k, sizeof(int), 1, cid);

            /* process results */

            for (i = 0; i < N; i++) a[i] = (b[i] - k) / 1.25 - b[i] /2.0;
            kk = k / 10;
            if((k - kk * 10) == 0)
                printf(" iteration #   %d  b[2000] = %f \n",k, b[2000]);
       }
   }
   else {
       printf ("\n Error in Pipe Communication\n");
   }

   /* stop child process and close pipe */

   pipe_close(cid);

   gettimeofday(&TpFinish, &Tzp);
   cputime = TpFinish.tv_sec - TpStart.tv_sec;
   printf("cpu time = %ld \n", cputime);
}
```

```
/***********************************************************
* File: prog1.c
* Date: Sep. 11, 1990
*
* Description: IPPC (inter-program pipe communication)
*              Example 2: Overhead cost of using IPPC.
*              Child program.
**********************************************************/

#include <stdio.h>
#include <math.h>
#include "ippc.h"

#define MAX_N_CALLS  30000
#define N            10000
void dummy_routine();

main()
{
   int i, k;
   float a[N], b[N];
   int iteration = 0;

   while (iteration < MAX_N_CALLS) {
      iteration++;

      /* retrieve array a[] and integer k from pipe */

      pipe_initialize2();
      pipe_read2(a, sizeof(float), N);
      pipe_read2(&k, sizeof(int),  1);

      k++;
      for (i = 0; i < N; i++) b[i] = 2.5 * a[i] + k;
      dummy_routine();

      /* output array b[] and integer k to pipe */

      pipe_write2(b, sizeof(float), N);
      pipe_write2(&k, sizeof(int), 1);
   }
}


void dummy_routine()
{
  int i, j;
  float a, b;

  b = 123.0;
  for (i = 0; i < 1000; i++)
     for (j = 0; j < 1000; j++)
        a = b * b;
}


/**********************************************************
* File: prog.c
* Date: Aug. 30, 1990
*
* Description: IPPC (inter-program pipe communication)
*              Example 2: Overhead cost of using IPPC.
*              This program is equivalent to "prog0.c" and
*              "prog1.c" combined except that this program
*              does not use pipes.
**********************************************************/

#include <stdio.h>
#include <math.h>
#include <sys/time.h>

#define MAX_N      10000
#define NO_ERROR       0

struct timeval TpStart, TpFinish;
struct timezone Tzp;
```

```
int subroutine();
void dummy_routine();

main()
{
   int i, k, kk;
   float a[MAX_N], b[MAX_N];
   long cputime;

   gettimeofday(&TpStart, &Tzp);

   for (i = 0; i < MAX_N; i++) {
      a[i] = 2.0 * (float) i;
   }
   k = 0;

   while (k < 100) {

        subroutine(a, b, &k);

        /* process results */

        for (i = 0; i < MAX_N; i++) a[i] = (b[i] - k)/ 1.25 - b[i] / 2.0;
        kk = k / 10;
        if((k - kk * 10) == 0)
             printf(" iteration #   %d  b[2000] = %f \n",k, b[2000]);
   }

   gettimeofday(&TpFinish, &Tzp);
   cputime = TpFinish.tv_sec - TpStart.tv_sec;

   printf("cpu time = %ld \n", cputime);
}


int subroutine(a, b, k)
float *a, *b;
int *k;
{
   int i;

   *k = *k + 1;
   for (i = 0; i < MAX_N; i++) b[i] = 2.5 * a[i] + (*k);
   dummy_routine();
}


void dummy_routine()
{
  int i, j, k;
  float a, b, c;
  b = 123.0;
  for (i = 0; i < 1000; i++)
     for (j = 0; j < 1000; j++)
        a = b * b;
}
```

APPENDIX C

SOURCE CODE LISTING OF EXAMPLE 3

```
/*******************************************************
* File: grand0.c
* Date: Sep. 11, 1990
*
* Description: IPPC (inter-program pipe communication)
*              Example 3: multi-children and multi-
*              generation case.
*              Parent program.
********************************************************/

#include <stdio.h>
#include <math.h>
#include <sys/time.h>
#include "ippc.h"

#define NO_ERROR      0

struct timeval TpStart, TpFinish;
struct timezone Tzp;

main()
{
   int i, j, k, m, n, s, cid1, cid2, cid4;
   double a, b, c;
   long cputime;

   /* initialization */

   gettimeofday(&TpStart, &Tzp);
   a = 1.0;
   m = 5;
   n = 10;
   s = sizeof(double);

   cid1 = pipe_open("grand1");
   cid2 = pipe_open("grand2");

   for (i = 0; i < m; i++) {

       /* communication with Child #1 */

       pipe_initialize(cid1);
       pipe_write(&a, s, 1, cid1);
       pipe_read (&b, s, 1, cid1);

       for (j = 0; j < n; j++) {

            /* communication with Child #2 */

            pipe_initialize(cid2);
            pipe_write(&j, sizeof(int), 1, cid2);
            pipe_write(&b, s, 1, cid2);
            pipe_write(&a, s, 1, cid2);
            pipe_read (&a, s, 1, cid2);
            pipe_read (&b, s, 1, cid2);
       }
       printf("intermediate value of a = %f\n", a);
   }
   pipe_close(cid2);
   pipe_close(cid1);

   /* communicate with child #4 */

   cid4 = pipe_open("grand4");
   pipe_initialize(cid4);
   pipe_write(&a, s, 1, cid4);
   pipe_read (&a, s, 1, cid4);
   pipe_close(cid4);

   printf("final value of a = %f\n", a);
```

```
   gettimeofday(&TpFinish, &Tzp);
   cputime = TpFinish.tv_sec - TpStart.tv_sec;
   printf("cpu time = %ld \n", cputime);
}


/*********************************************************
* File: grand1.c
* Date: Sep. 11, 1990
*
* Description: IPPC (inter-program pipe communication)
*              Example 3: multi-children and multi-
*              generation case.
*              Child program -- Child #1.
**********************************************************/

#include <stdio.h>
#include <math.h>
#include "ippc.h"

main()
{
   int i, s;
   double a, b;

   s = sizeof(double);

   for (i = 0; i < 99999; i++) { /* infinite loop */
      pipe_initialize2();
      pipe_read2(&a, s, 1);

      b = sqrt(2.0) * a;         /* task of child #1 */

      pipe_write2(&b, s, 1);
   }
}


/*********************************************************
* File: grand2.c
* Date: Sep. 11, 1990
*
* Description: IPPC (inter-program pipe communication)
*              Example 3: multi-children and multi-
*              generation case.
*              Child program -- Child #2.
**********************************************************/

#include <stdio.h>
#include <math.h>
#include "ippc.h"

main()
{
   int i, j, s, cid3;
   double a, b, c;

   s = sizeof(double);

   for (i = 0; i < 99999; i++) {  /* infinite loop */

      /* read from parent */

      pipe_initialize2();
      pipe_read2(&j, sizeof(int), 1);
      pipe_read2(&b, s, 1);
      pipe_read2(&a, s, 1);

      b += (double) j;          /* task of child #2 */

      /* communication with child #3 */

      cid3 = pipe_open("grand3");
      pipe_initialize(cid3);
      pipe_write(&b, s, 1, cid3);
      pipe_write(&a, s, 1, cid3);
      pipe_read (&a, s, 1, cid3);
```

```
      pipe_close(cid3);

      /* output to parent */

      pipe_write2(&a, s, 1);
      pipe_write2(&b, s, 1);
   }
}


/*******************************************************
* File: grand3.c
* Date: Sep. 11, 1990
*
* Description: IPPC (inter-program pipe communication)
*              Example 3: multi-children and multi-
*              generation case.
*              Child program -- Child #3.
*******************************************************/

#include <stdio.h>
#include <math.h>
#include "ippc.h"

main()
{
   int s, i;
   double a, b;

   s = sizeof(double);

   for (i = 0; i < 99999; i++) {  /* infinite loop */
      pipe_initialize2();
      pipe_read2(&b, s, 1);
      pipe_read2(&a, s, 1);

      a += atan(b);          /* task of child #3 (grand child) */

      pipe_write2(&a, s, 1);
   }
}


/*******************************************************
* File: grand4.c
* Date: Sep. 11, 1990
*
* Description: IPPC (inter-program pipe communication)
*              Example 3: multi-children and multi-
*              generation case.
*              Child program -- Child #4.
*******************************************************/

#include <stdio.h>
#include <math.h>
#include "ippc.h"

main()
{
   int s, i, cid5, cid6;
   double a, b, c;

   s = sizeof(double);

   for (i = 0; i < 99999; i++) {  /* infinite loop */
      pipe_initialize2();
      pipe_read2(&a, s, 1);

      b = 3.0 + a * a;                /* task of child #4 */

      /* communication with child #5 */

      cid5 = pipe_open("grand5");
      pipe_initialize(cid5);
      pipe_write(&b, s, 1, cid5);
      pipe_read (&c, s, 1, cid5);
      pipe_close(cid5);
```

```
      /* communication with child #6 */

      cid6 = pipe_open("grand6");
      pipe_initialize(cid6);
      pipe_write(&c, s, 1, cid6);
      pipe_read (&a, s, 1, cid6);
      pipe_close(cid6);

      /* output to parent */

      pipe_write2(&a, s, 1);
   }
}


/*******************************************************
* File: grand5.c
* Date: Sep. 11, 1990
*
* Description: IPPC (inter-program pipe communication)
*              Example 3: multi-children and multi-
*              generation case.
*              Child program -- Child #5.
********************************************************/

#include <stdio.h>
#include <math.h>
#include "ippc.h"

main()
{
   int s, i;
   double b, c;

   s = sizeof(double);

   for (i = 0; i < 9999; i++) {  /* infinite loop */
      pipe_initialize2();
      pipe_read2(&b, s, 1);

      c = sqrt(b);                    /* task of child #5 (grand child) */

      pipe_write2(&c, s, 1);
   }
}


/*******************************************************
* File: grand6.c
* Date: Sep. 11, 1990
*
* Description: IPPC (inter-program pipe communication)
*              Example 3: multi-children and multi-
*              generation case.
*              Child program -- Child #6.
********************************************************/

#include <stdio.h>
#include <math.h>
#include "ippc.h"

main()
{
   int s, i;
   double a, c;

   s = sizeof(double);

   for (i = 0; i < 99999; i++) {  /* infinite loop */
      pipe_initialize2();
      pipe_read2(&c, s, 1);

      a = c + sqrt(c);                /* task of child #6 (grand child) */

      pipe_write2(&a, s, 1);
   }
}
```

```
/*********************************************************
* File: grand.c
* Date: Aug. 30, 1990
*
* Description: IPPC (inter-program pipe communication)
*              Example 3: multi-children and multi-
*              generation case.
*              This program is functionally the same as
*              "grand0.c" to "grand6.c" combined.
**********************************************************/

#include <stdio.h>
#include <math.h>
#include <sys/time.h>

struct timeval TpStart, TpFinish;
struct timezone Tzp;

main()
{
   int i, j, k, m, n;
   double a, b, c;
   long cputime;

   /* initialization */

   gettimeofday(&TpStart, &Tzp);
   a = 1.0;
   m = 5;
   n = 10;

   for (i = 0; i < m; i++) {

       b = sqrt(2.0) * a;          /* task of child #1 */

       for (j = 0; j < n; j++) {
           b += (double) j;        /* task of child #2 */
           a += atan(b);           /* task of child #3 (grand child) */
       }
       printf("intermediate value of a = %f\n", a);
   }

   b = 3.0 + a * a;                /* task of child #4 */
   c = sqrt(b);                    /* task of child #5 (grand child) */
   a = c + sqrt(c);                /* task of child #6 (grand child) */
   printf("final value of a = %f\n", a);

   gettimeofday(&TpFinish, &Tzp);
   cputime = TpFinish.tv_sec - TpStart.tv_sec;
   printf("cpu time = %ld \n", cputime);
}
```