	FACULTY OF INFORMATION TECHNOLOGY >>> COURSE MATERIAL <<< **INTRODUCTION TO OPERATING SYSTEM** Course ID 502047

Lab Part 6 – Process communication

Nếu như các tiểu trình có thể liên lạc nội bộ bằng các biến số toàn cục, thì các tiến trình cha con lại rất khó khăn trong liên lạc do hoàn toàn không thể dùng phương pháp khai báo biến số toàn cục. Hệ điều hành cung cấp các cơ chế liên lạc bao gồm Vùng nhớ chia sẻ và Gửi thông điệp được giới thiệu trong LAB này.

Mục tiêu	**Lý thuyết liên quan**	**Tài nguyên**
Pipe in Linux	Ch4: Threads and Concurrency.	https://github.com/Trantin84/LAB_IntroOS (mã nguồn ví dụ) Sử dụng image Ubuntu 14 / 16
Share memory		
Message queue		

Yêu cầu nộp bài: các tập tin mã nguồn .c và tập tin khả thực thi .out của các "ví dụ" và bài tập cuối hướng dẫn.

Preferences

[1] Abraham Silberschatz, Peter B. Galvin, Greg Gagne, [2018], Operating System Concepts, 10th edition, John Wiley & Sons, New Jersey.

Programming Problems of Chapter 3.

[2] Nguyễn Hồng Vũ, [2016] Hướng dẫn LAB Hệ điều hành, khoa Công nghệ thông tin, ĐH Tôn Đức Thắng.

[3] Greg Gagne , [2019], GitHub OS-BOOK OSC10e, Westminster College, United States

Access https://github.com/greggagne/osc10e in September 2019.

1. Pipes

- Được sử dụng để truyền dữ liệu giữa các tiến trình theo cơ chế FIFO

- Thư viện:

```
#include <unistd.h>
```

- Hàm ghi / đọc

```
ssize_t write(int fd, const void *buf, size_t count)
ssize_t read(int fd, const void *buf, size_t count)
```

- Kết quả trả về:

  - -1 thất bại
  - Thành công: số byte ghi được hoặc đọc được.

Unnamed pipe:

- Thường được sử dụng cục bộ

- Dành cho các tiến trình có quan hệ cha con

- Hàm tạo unnamed pipe:

```
int pipe(int filedes[2]);
```

- Kết quả

  - Thành công: 0; hai file mô tả tương ứng được trả về trong file filedes[0] và filedes[1].
  - Với hệ thống cũ, filedes[0] được sử dụng để đọc, filedes[1] được sử dụng để ghi
  - Trong các hệ thống mới sử dụng fullduplex, nếu filedes[0] được dùng để đọc thì filedes[1] được dùng để ghi và ngược lại.
  - Thất bại trả về -1.

Ví dụ 1.1:

- Tiến trình con đọc dữ liệu từ đối số truyền, ghi vào pipe.

- Tiến trình cha đọc từ pipe và xuất ra màn hình.

```
1     #include <stdio.h>
2     #include <unistd.h>
3     #include <string.h>

M1    int main(int argc, char* argv[])
02    {
```

```
03      char result[100];
04      int fp[2];
05      int pid;
06      if(argc<2) {
07         printf("Doi so thieu.\n");
08         return -1;
09      }
10      if(pipe(fp)==0) {
11         pid = fork();
12         if(pid<0) {printf("Fork failed\n"); return -1;}
13         else if(pid==0) {
14*           printf("Data from child: %s\n", argv[1]);
15            close(fp[0]);
16            write(fp[1], argv[1], strlen(argv[1]));
17         }
18         else {
19            close(fp[1]);
20*           read(fp[0], result, strlen(argv[1]));
21            printf("Read from child: %s\n", result);
22         }
23      } // of row 14
24*     else {printf("Pipe failed\n"); return -2;}
25   }
```

```
>./vidu1_1.out Hello
Data from child: Hello
Read from child: Hello
```

Named Pipe

- Mang ý nghĩa toàn cục

- Tương tự như unnamed pipe

- Được ghi nhận trên file system

- Có thể sử dụng với các tiến trình không có quan hệ cha con

- Có thể tạo từ dấu nhắc lệnh

- Thư viện:

```
#include <sys/types.h>
#include<sys/stat.h>
```

- Hàm khởi tạo

```
int mknod(const char *path, mode_t mode, dev_t dev)
```

- Đối số:

  - path: đường dẫn đến pipe; kết hợp s_IFIFO với quyền khác.
  - mode: quyền truy cập trên file
  - dev: mặc định là 0

- Kết quả trả về:

  - Thất bại: -1; Thành công: 0

- Hàm sử dụng

```
int mkfifo(const char *pathname, mode_t mode)
```

- Đối số:

  - path: đường dẫn đến pipe
  - mode: quyền truy cập trên file

- Kết quả trả về:

  - Thất bại: -1
  - Thành công: 0

- Tiến trình cha ghi dữ liệu vào pipe

- Tiến trình vào đọc, hiển thị dữ liệu, ghi dữ liệu phản hồi.

- Tiến trình cha đọc dữ liệu phản hồi và hiển thị ra màn hình.

Ví dụ 1.2:

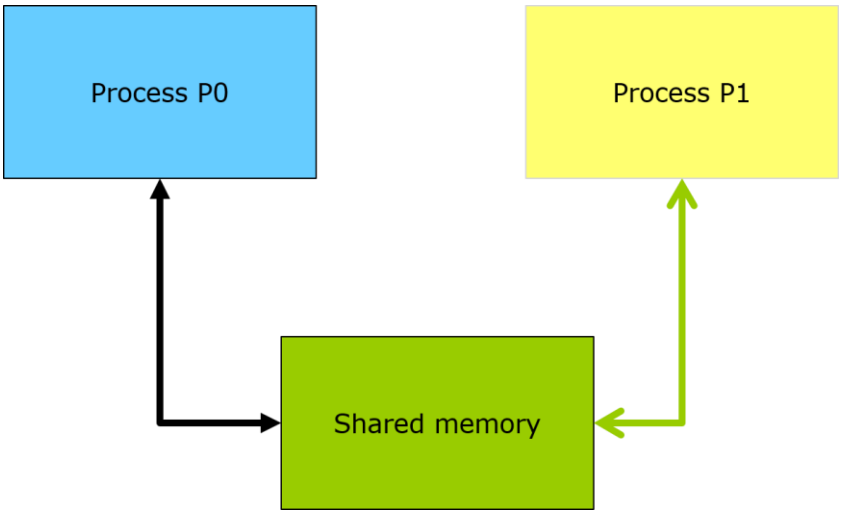
```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <unistd.h>
4   #include <string.h>
5   #include <sys/types.h>
6   #include <sys/stat.h>
7   #include <sys/errno.h>
8
9   #define FIFO1 "/tmp/ff.1"
10  #define FIFO2 "/tmp/ff.2"
11  #define PM 0666
12  extern int errno;
13  #define PIPE_BUF 4096

M1  int main(int argc, char* argv[])
02  {
```

```
03    char s1[PIPE_BUF], s2[PIPE_BUF];
      int childpid, readfd, writefd;
      if((mknod(FIFO1, S_IFIFO | PM, 0)<0)&&(errno!=EEXIST)){
         printf("Fail to create FIFO 1. Aborted.\n);
         return -1;
      }
      if((mknod(FIFO2, S_IFIFO | PM, 0)<0)&&(errno!=EEXIST)){
         unlink(FIFO1);
         printf("Fail to create FIFO 2. Aborted.\n);
         return -1;
      }
      childpid=fork();
      if(childpid==0){   //child
         if((readfd=open(FIFO1, 0))<0)
            perror("Child cannot open readFIFO.\n");
         if((writefd=open(FIFO2, 1))<0)
            perror("Child cannot open writeFIFO.\n");
         read(readfd, s2, PIPE_BUF);
         printf("Child read from parent: %s\n", s2);
         printf("Enter response: ");
         gets(s1);
         write(writefd, s1, strlen(s1));
         close(readfd);
         close(writefd);
         return 1;
      }
      else if(childpid>0) {   //parent
         if((writefd=open(FIFO1, 1))<0)
            perror("Parent cannot open writeFIFO.\n");
         if((readfd=open(FIFO2, 0))<0)
            perror("Child cannot open readFIFO.\n");
         printf("Enter data to FIFO1: ");
         gets(s1);
         write(writefd, s1, strlen(s1));
         read(readfd, s2, PIPE_BUF);
         printf("Parent read from child: %s\n", s2);
         while(wait((int*) 0)!=childpid);
         close(readfd);
         close(writefd);
         if(unlink(FIFO1)<0)
            perror("Cannot remove FIFO1.\n");
         if(unlink(FIFO2)<0)
```

```
            perror("Cannot remove FIFO2.\n");
        return 1;
    }
    else { printf("Fork failed\n"); return -1;}
}
```

2. Share memory



Hình 1. Mô hình Vùng nhớ chia sẻ

- Cho phép các tiến trình sử dụng chung vùng nhớ.

- Kích thước tối thiểu: 1 Byte
- Kích thước tối đa: 4 MB
- Số vùng nhớ chia sẻ tối đa: 4096

- Sử dụng:

- Tạo vùng nhớ chia sẻ
- Các tiến trình phải gắn vùng nhớ chia sẻ vào không gian địa chỉ của mình trước khi sử dụng
- Sau khi sử dụng xong, có thể gỡ vùng nhớ ra khỏi không gian địa chỉ của tiến trình

```
shmget(): sử dụng để tạo SM
shmctl(): sử dụng để điều khiển SM
shmat(): sử dụng để gắn SM vào tiến trình
shmdt(): sử dụng để gỡ SM khỏi tiến trình
```

- Thư viện:

```
#include <sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
```

- Prototype của hàm:

```
int shmget (key_t key,int size, int shmflg);
```

- Được sử dụng để tạo SM.

- Đối số:

- key: key tương ứng với SM
- size: kích thước SM theo Byte
- shmflg: tương tự semflg của semget(). Không có IPC_EXCL

- Thư viện:

```
#include <sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
```

- Prototype của hàm: Gắn SM vào tiến trình.

```
void *shmat (int shmid, void *shmaddr, int shmflg);
```

- Đối số:

- shmid: shmid tương ứng
- shmaddr: địa chỉ gắn SM
- shmflg: SHM_RDONLY hoặc 0

- Thư viện:

```
#include <sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
```

- Prototype của hàm: Gỡ SM khỏi tiến trình.

```
int shmdt (void *shmaddr);
```

- Đối số:

- shmaddr: địa chỉ nhớ gắn SM

- Thư viện:

```
#include <sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
```

- Prototype của hàm: sử dụng để thay đổi quyền trên SM.

int shmctl (int shmid, int cmd, struct shmid_ds *buf);

- Đối số:

- shmid: id of SM.
- cmd: sử dụng để xác định hoạt động cụ thể của shmctl. cmd bao gồm các giá trị sau:
- cmd: sử dụng để xác định hoạt động cụ thể của shmctl. cmd bao gồm các giá trị sau:
    - IPC_STAT: trả về giá trị hiện thời của mỗi thành phần trong shmid_ds. Thông tin trả về được lưu giữ trong đối số thứ 3

- IPC_SET: hạn chế số của những giá trị thành phần của shmid_ds. Các thành phần có thể hiệu chỉnh:shm_perm.uid; shm_perm.gid,..
- IPC_RMID: xóa SM có id tương ứng.
- SHM_LOCK: khóa SM
- SHM_UNLOCK: mở khóa SM

```
struct shmid_ds {
    struct ipc_perm shm_perm;        /* operation permission struct     */
    size_t shm_segsz;                /* size of segment in bytes        */
    __time_t shm_atime;              /* time of last shmat()            */
    unsigned long int __unused1;
    __time_t shm_dtime;              /* time of last shmdt()            */
    unsigned long int __unused2;
    __time_t shm_ctime;                /* time of last change by shmctl()
*/
    unsigned long int __unused3;
    __pid_t shm_cpid;                /* pid of creator                  */
    __pid_t shm_lpid;                /* pid of last shmop               */
    shmatt_t shm_nattch;             /* number of current attaches      */
    unsigned long int __unused4;
    unsigned long int __unused5;
  };
```

Ví dụ 2.1: Tiến trình con đọc vào 2 số nguyên từ đối số truyền, ghi vào SM, tiến trình cha thực hiện tính tổng và ghi lại vào SM. Tiến trình con đọc kết quả và xuất ra màn hình

```
01   #include <stdio.h>
02   #include <unistd.h>
03   #include <limits.h>
04   #include <string.h>
05   #include <stdlib.h>
06   #include <sys/types.h>
07   #include <sys/ipc.h>
08   #include <sys/shm.h>
09
10   #define SIZE 256
11
12   int main(int argc, char* argv[])
13   {
14*     int *shm, shmid, k;
15      key_t key;
16      if((key=ftok(".",'a'))==-1){
17         perror("Key created.\n");
```

```
18        return 1;
19     }
20     if(shmid = shmget(key, SIZE, IPC_CREAT | 0666)) == -1) {
21        perror("Shared memory created.\n");
22        return 2;
23     }
24     shm = (int*) shmat(shmid, 0, 0);
25     pid = fork();
26     if(pid==0) {    // child
27        shm[0] = atoi(argv[1]);
28        shm[1] = atoi(argv[2]);
29        sleep(3);
30        printf("%d + %d = %d\n", shm[0], shm[1], shm[2]);
31        shmdt((void*) shm);
32        shmctl((shmid, IPC_RMID, (struct shmid_ds*) 0);
33        return 0;
34     }
35     else if(pid >0) {  // parent
36        sleep(1);
37        shm[2] = shm[1] + shm[0];
38        shmdt((void*) shm);
39        sleep(5);
40        return 0;
41     }
42     else { perror("Fork failed."); return 4;  }
43     return 0;
44  }
```

```
>./t2_1.out 1 2
1 + 2 = 3
```

3. Message queues

- Các thông tin cần để giao tiếp được đặt trong một cấu trúc thông điệp định nghĩa trước.

- Tiến trình tạo thông điệp phải xác định được kiểu và vị trí của thông điệp.

- Các tiến trình truy xuất hàng đợi của thông điệp thông qua hàng đợi theo cơ chế FIFO

- msgget(): sử dụng để tạo Message queue
- msgctl(): sử dụng để điều khiển MQ
- msgsnd(): sử dụng để gởi thông điệp
- msgrcv(): sử dụng để nhận thông điệp

- Thư viện:

```
#include <sys/types.h>
#include<sys/ipc.h>
```

- Prototype của hàm:

key_t ftok (char *pathname, char proj);

- Được sử dụng để tạo key cho các thao tác trên MQ. Hàm ftok có thể tạo ra cùng key value.

- Đối số:

- pathname: tham chiếu đến một file đã tồn tại. Thường sử dụng "." là tham chiếu đến chính thư mục hiện tại.
- proj: là một ký tự đơn định danh cho project.

Ví dụ 3.1:

```
01  #include <stdio.h>
02  #include <sys/types.h>
03  #include <sys/ipc.h>
04
05  int main(int argc, char* argv[])
06  {
07     key_t key;  char i;
08     for(i='a'; i<'e'; i++) {
09        key = ftok(".", i);
10        printf("Proj = %c key = %d.\n", i, key);
11     }
12     return 0;
13  }
```

```
>./t3_1.out
```

```
vunguyen@vunguyen:~/lab8$ gcc -c t1.c
vunguyen@vunguyen:~/lab8$ gcc -o t1.out t1.o
vunguyen@vunguyen:~/lab8$ ./t1.out
proj = a key = 1627461775
proj = b key = 1644238991
proj = c key = 1661016207
proj = d key = 1677793423
```

msgget – tạo thông điệp

- Thư viện:

```
#include <sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
```

- Prototype của hàm:

```
int msgget (key_t key,int msgflg);
```

- Được sử dụng để tạo thông điệp.

- Đối số:

  - msgflg: các bit thấp xác định quyền truy cập đến MQ.

Ví dụ 3.2

```
01  #include <stdio.h>
02  #include <unistd.h>
03  #include <limits.h>
04  #include <sys/types.h>
05  #include <sys/ipc.h>
06  #include <sys/msg.h>
07
08  #define MAX 5
09
10  int main(int argc, char* argv[])
11  {
12     FILE *fin;
13     char buffer[PIPE_BUF], proj='A';
14*    int i, n, mid[MAX];
15     key_t key;
16     for(i=0; i<MAX; i++, proj++) {
17        key = ftok(".", proj);
18        if(mid[i]=msgget(key, IPC_CREAT | 0666)==-1){
19           perror("Queue created.\n");
20           return 1;
21        }
22     }
23     fin=popen("ipcs","r");
24     while((n=read(fileno(fin), buffer, PIPE_BUF))>0)
25        write(fileno(stdout), buffer, n);
26     pclose(fin);
27     for(i=0; i<MAX; i++)
```

```
28          msgctl(mid[i], IPC_RMID, (struct msqid_ds *)0);
29      return 0;
30  }
```

```
>./t3_1.out
```

Hiệu chỉnh popen("ipcs –q", "r"); popen("ipcs –s", "r"); popen("ipcs –m", "r"); popen("ipcs -q -p", "r"); để xem xem theo Message queue, semaphore, share memory

```
vunguyen@vunguyen:~/lab8$ gcc -c t2.c
vunguyen@vunguyen:~/lab8$ gcc -o t2.out t2.o
vunguyen@vunguyen:~/lab8$ ./t2.out

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x4101188f 163840     vunguyen   666        0            0
0x4201188f 32769      vunguyen   666        0            0
0x4301188f 65538      vunguyen   666        0            0
0x4401188f 98307      vunguyen   666        0            0
0x4501188f 131076     vunguyen   666        0            0

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 294912     vunguyen   600        524288     2          dest
0x00000000 524289     vunguyen   600        16777216   2
0x00000000 491522     vunguyen   600        524288     2          dest
0x00000000 917507     vunguyen   600        524288     2          dest
0x00000000 1409028    vunguyen   600        524288     2          dest
```

msgctl – điều khiển thông điệp

- Thư viện:

```
#include <sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
```

- Prototype của hàm: sử dụng để thay đổi quyền trên MQ.

```
int msgget (int msqid,int cmd, struct msqid_ds *buf);
```

- Đối số:

- msqid: id of MQ.
- cmd: sử dụng để xác định hoạt động cụ thể cua msgctl. cmd bao gồm các giá trị sau:

- Prototype của hàm: sử dụng để thay đổi quyền trên MQ.

```
int msgget (int msqid,int cmd, struct msqid_ds *buf);
```

- Đối số:

  - cmd: sử dụng để xác định hoạt động cụ thể cua msgctl. cmd bao gồm các giá trị sau:
    - IPC_STAT: trả về giá trị hiện thời của mỗi thành phần trong msqid_ds. Thông tin trả về được lưu giữ trong đối số thứ 3
    - IPC_SET: hạn chế số của những giá trị thành phần của msqid_ds. Các thành phần có thể hiệu chỉnh:msg_perm.uid; msg_perm.gid,..
    - IPC_RMID: xóa MQ có id tương ứng.

```
struct msqid_ds {
  struct ipc_perm msg_perm;
  struct msg *msg_first;          /* first message on queue, unused  */
  struct msg *msg_last;           /* last message in queue, unused */
  __kernel_time_t msg_stime;      /* last msgsnd time */
  __kernel_time_t msg_rtime;      /* last msgrcv time */
  __kernel_time_t msg_ctime;      /* last change time */
  unsigned long  msg_lcbytes;     /* Reuse junk fields for 32 bit */
  unsigned long  msg_lqbytes;     /* ditto */
  unsigned short msg_cbytes;      /* current # of bytes on queue */
  unsigned short msg_qnum;        /* number of messages in queue */
  unsigned short msg_qbytes;      /* max number of bytes on queue */
  __kernel_ipc_pid_t msg_lspid;   /* pid of last msgsnd */
  __kernel_ipc_pid_t msg_lrpid;   /* last receive pid */
};
```

```
struct ipc_perm  {
    __key_t __key;                      /* Key                  */
    __uid_t uid;                        /* Owner's user ID.     */
    __gid_t gid;                        /* Owner's group ID.    */
    __uid_t cuid;                       /* Creator's user ID.   */
    __gid_t cgid;                       /* Creator's group ID. */
    unsigned short int mode;    /* Access permission.  */
    unsigned short int __pad1;
    unsigned short int __seq;     /* Sequence number.    */
    unsigned short int __pad2;
    unsigned long int __unused1;
    unsigned long int __unused2;
};
```

```
struct msg {
    struct msg     *msg_next; /* ptr to next message on q */
    long           msg_type;       /* message type */
    ushort         msg_ts;         /* message text size */
    short          msg_spot;      /* address of text message */
};
```

Ví dụ 3.3:

```
01  #include <stdio.h>
02  #include <unistd.h>
03  #include <limits.h>
04  #include <sys/types.h>
05  #include <sys/ipc.h>
06  #include <sys/msg.h>
07
08  #define MAX 5
09
10  int main(int argc, char* argv[])
11  {
12     FILE *fin;
13     char buffer[PIPE_BUF], proj='A';
14*    int i, n, mid[MAX];
15     key_t key;
16     for(i=0; i<MAX; i++, proj++) {
17        key = ftok(".", proj);
18        if(mid[i]=msgget(key, IPC_CREAT | 0666)==-1){
19           perror("Queue created.\n");
20           return 1;
21        }
22     }
23     fin=popen("ipcs","r");
24     while((n=read(fileno(fin), buffer, PIPE_BUF))>0)
25        write(fileno(stdout), buffer, n);
26     pclose(fin);
27     for(i=0; i<MAX; i++)
28        msgctl(mid[i], IPC_RMID, (struct msqid_ds *)0);
29     return 0;
30  }
```

```
>./lab3_3.out
```

Thao tác trên MQ

MQ được sử dụng để gởi và nhận thông điệp. Thông điệp được định nghĩa bởi cấu trúc sau:

```
struct msgbuf  {
    long int mtype;          /* type of received/sent message */
    char mtext[1];           /* text of the message */
  };
```

- Thư viện:

```
#include <sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
```

```
int msgsnd (int msqid, struct msgbuf *msgp, size_t msgsz,     int msgflg);
```

- Giá trị trả về: Thành công 0; thất bại -1

- Đối số:

  - mdqid: id hợp lệ của MQ
  - msgp: tham chiếu đến message sẽ được gởi
  - msgsz: kích thước của message
  - msgflg: hành động sẽ thực hiện nếu hệ thống có giới hạn cho MQ. Các giá trị có thể:
    - IPC_NOWAIT: khi hệ thống đạt đến giới hạn, msgsnd sẽ không gởi, trả lỗi về cho tiến trình gọi
    - 0: msgsnd sẽ khóa cho đến khi giới hạn được gỡ bỏ.

```
int msgrcv (int msqid, struct msgbuf *msgp, size_t msgsz, long msgtyp,
int msgflg);
```

- Giá trị trả về: thành công 0; thất bại -1

- Đối số:

  - mdqid: id hợp lệ của MQ
  - msgp: tham chiếu đến message sẽ được nhận
  - msgsz: kích thước tối đa của message nhận
  - msgtype: loại của message nhận được
  - msgflg: hành động sẽ thực hiện nếu msgtype nhận được Message không có trong queue. Các giá trị có:
    - IPC_NOWAIT: block message type
    - MSG_EXCEPT and msgtyp>0: trả về message đầu tiên.

Ví dụ 3.4:

```
01  #include <stdio.h>
02  #include <unistd.h>
03  #include <limits.h>
04  #include <sys/types.h>
05  #include <sys/ipc.h>
06  #include <sys/msg.h>
07  #include <string.h>
08  #include <stdlib.h>
09
10  struct Message{
11     int mtype;
```

```
12      char* content;
13  }
14*
15  int main(int argc, char* argv[])
16  {
17      int mid, id, n, pid;
18      key_t key;
19      struct Message msg;
20      msg.content=(char*) malloc(BUFFSIZ);
21      if((key = ftok(".", 'a'))==-1){
22          perror("Key created.\n");
23          return 1;
24      }
25      if(mid=msgget(key, IPC_CREAT | 0666)==-1){
26              perror("Queue created.\n");
27              return 2;
28      }
29
30      pid = fork();
31      if(pid==0) {   // child
32          msg.mtype=10;
33          msg.content=argv[1];
34          n=strlen(msg.content);
35          printf("%s\n", msg.content);
36          n+=sizeof(msg.mtype);
37          if(msgsnd(mid, &msg, n, 0)==-1){
38              perror("Message sent.\n");
39              return 4;
40              }
41          sleep(5);
42          printf("Child receive from parent: \n);
43          if(n=msgrcv(mid, &msg, BUFSIZ, 11, 0)==-1){
44              perror("Message received.\n");
45              return 5;
46          }
47          msg.content[strlen(msg.content)]=0;
48          printf("%s\n",msg.content);
49          return 0;
50      }
51      else if(pid >0) {  // parent
52          sleep(1);
53          printf("Child receive from parent: \n);
```

```
54          if(n=msgrcv(mid, &msg, BUFSIZ, 10, 0)==-1){
55             perror("Message received.\n");
56             return 5;
57          }
58          printf("Parent received from child: \n");
59          msg.content[strlen(msg.content)]=0;
60          printf("%s\n",msg.content);
61          printf("------------------- \n");
62          printf("Message from parent:\n");
63          msg.content=argv[2];
64          n=strlen(msg.content);
65          msg.mtype=11;
66          n+=sizeof(msg.mtype);
67          if(msgsnd(mid, &msg, n, 0)==-1){
68             perror("Message sent.\n");
69             return 4;
70          }
71          sleep(10);
72          return 0;
73       }
74       else { perror("Fork failed."); return 3;  }
75       return 0;
76    }
```

```
>./vidu4_1.out Message_from_Child Message_from_Parent
```

```
vunguyen@vunguyen:~/lab8$ gcc -c t4.c
vunguyen@vunguyen:~/lab8$ gcc -o t4.out t4.o
vunguyen@vunguyen:~/lab8$ ./t4.out MQ_from_children MQ_from_Parent
MQ_from_children
Parent receive from children:
MQ_from_children
----------------------------------------
Messsage from parent
children receive from parent:
MQ_from_Parent
```

4. Bài tập

1A. Tiến trình cha chuyển đối số đầu tiên (argv [1]) là một số nguyên lớn hơn 3 cho tiến trình con thông qua đường ống. Tiến trình con nhận, tính giá trị n! = 1 * 2 * … * n và ghi nó vào đường ống. Tiến trình cha nhận và xuất dữ liệu ra màn hình. Sử dụng đường ống vô danh (Unnamed Pipe).

```
>./baitap1A.out 4
4! = 24
```

1B. Giải lại vấn đề 1A với đường ống có tên (Named Pipe).

1C. Giải lại vấn đề 1A với kỹ thuật truyền thông điệp (Message Passing)

2A. Tiến trình cha đọc hai số nguyên và một thao tác +, -, *, / và chuyển tất cả cho tiến trình con. Quá trình con tính toán kết quả và trả về cho tiến trình cha. Quá trình cha mẹ ghi kết quả vào một tệp.

```
>./baitap2A.out 4 6 +
4 + 6 = 10
```

2B. Giải lại vấn đề 2A với đường ống có tên (Named Pipe).

2C. Giải lại vấn đề 1A với kỹ thuật truyền thông điệp (Message Passing)

3. (Shared Memory) Tiến trình con ghi một mảng vào SM_0, với SM[0] chứa số phần tử mảng. Tiến trình cha thực hiện, tính tổng các phần tử của mảng và ghi vào cuối SM_1. Trình con trình nhận và xuất dữ liệu ra màn hình (lưu ý sử dụng 2 SM).