On this page

实验4 进程间通信

实验内容与任务

- 通过共享内存实现进程间通信

相关知识及背景

1. 在进程间通过将共享内存附加到进程地址空间实现两个或多个进程共享一段物理内存空间，这样两个或多个进程就可以通过这段共享的内存空间实现数据交换和通信。
2. 共享内存方式实现进程间通信适合于两个或多个进程间需要高效交换大量数据的场景。
3. 两个或多个进程访问修改共享数据时会存在数据访问冲突的问题，设计进程间通信程序时需要考虑使用互斥锁或信号量实现互斥访问共享数据。

实验要求及过程

实验要求

1. 能够理解共享内存方式实现进程间通信的原理。
2. 能够使用共享内存相关函数实现具有进程间通信功能的应用程序。
3. 能够使用互斥锁和信号量实现互斥访问共享内存或共享数据。

实验过程

示例1. 在一个程序中通过共享内存实现父子进程间通信

1. 文件shm.c

```
#include  <stdio.h>
#include  <stdlib.h>
#include  <sys/types.h>
```

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

void  ClientProcess(int []);

void  main(int  argc, char *argv[])
{
     int    ShmID;
     int    *ShmPTR;
     pid_t  pid;
     int    status;

     if (argc != 5) {
          printf("Use: %s #1 #2 #3 #4\n", argv[0]);
          exit(1);
     }

     ShmID = shmget(IPC_PRIVATE, 4*sizeof(int), IPC_CREAT | 0666);
     if (ShmID < 0) {
          printf("*** shmget error (server) ***\n");
          exit(1);
     }
     printf("Server has received a shared memory of four integers...\n");

     ShmPTR = (int *) shmat(ShmID, NULL, 0);
     if ((int) ShmPTR == -1) {
          printf("*** shmat error (server) ***\n");
          exit(1);
     }
     printf("Server has attached the shared memory...\n");

     ShmPTR[0] = atoi(argv[1]);
     ShmPTR[1] = atoi(argv[2]);
     ShmPTR[2] = atoi(argv[3]);
     ShmPTR[3] = atoi(argv[4]);
     printf("Server has filled %d %d %d %d in shared memory...\n",
            ShmPTR[0], ShmPTR[1], ShmPTR[2], ShmPTR[3]);

     printf("Server is about to fork a child process...\n");
     pid = fork();
     if (pid < 0) {
          printf("*** fork error (server) ***\n");
          exit(1);
     }
     else if (pid == 0) {
          ClientProcess(ShmPTR);
          exit(0);
```

```
    }

    wait(&status);
    printf("Server has detected the completion of its child...\n");
    shmdt((void *) ShmPTR);
    printf("Server has detached its shared memory...\n");
    shmctl(ShmID, IPC_RMID, NULL);
    printf("Server has removed its shared memory...\n");
    printf("Server exits...\n");
    exit(0);
}

void  ClientProcess(int  SharedMem[])
{
    printf("   Client process started\n");
    printf("   Client found %d %d %d %d in shared memory\n",
               SharedMem[0], SharedMem[1], SharedMem[2], SharedMem[3]);
    printf("   Client is about to exit\n");
}
```

示例2. 在两个程序中，通过共享内存Key实现进程间通信

1. 文件shm.h

```
struct Memory
{
    int status;
    int data[4];
};

#define NOT_READY 0
#define FILLED 1
#define TAKEN 2
```

2. 文件server.c

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

#include  "shm-02.h"
```

```
void  main(int  argc, char *argv[])
{
     key_t          ShmKEY;
     int            ShmID;
     struct Memory  *ShmPTR;

     if (argc != 5) {
          printf("Use: %s #1 #2 #3 #4\n", argv[0]);
          exit(1);
     }

     ShmKEY = ftok(".", 'x');

     ShmID = shmget(ShmKEY, sizeof(struct Memory), IPC_CREAT | 0666);

     if (ShmID < 0) {
          printf("*** shmget error (server) ***\n");
          exit(1);
     }

     printf("Server has received a shared memory of four integers...\n");

     ShmPTR = (struct Memory *) shmat(ShmID, NULL, 0);
     if ((int) ShmPTR == -1) {
          printf("*** shmat error (server) ***\n");
          exit(1);
     }
     printf("Server has attached the shared memory...\n");

     ShmPTR->status  = NOT_READY;
     ShmPTR->data[0] = atoi(argv[1]);
     ShmPTR->data[1] = atoi(argv[2]);
     ShmPTR->data[2] = atoi(argv[3]);
     ShmPTR->data[3] = atoi(argv[4]);
     printf("Server has filled %d %d %d %d to shared memory...\n",
            ShmPTR->data[0], ShmPTR->data[1],
            ShmPTR->data[2], ShmPTR->data[3]);
     ShmPTR->status = FILLED;

     printf("Please start the client in another window...\n");

     while (ShmPTR->status != TAKEN)
          sleep(1);

     printf("Server has detected the completion of its child...\n");
     shmdt((void *) ShmPTR);
```

```
     printf("Server has detached its shared memory...\n");
     shmctl(ShmID, IPC_RMID, NULL);
     printf("Server has removed its shared memory...\n");
     printf("Server exits...\n");
     exit(0);
}
```

3.文件client.c

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

#include  "shm-02.h"

void  main(void)
{
     key_t          ShmKEY;
     int            ShmID;
     struct Memory  *ShmPTR;

     ShmKEY = ftok(".", 'x');
     ShmID = shmget(ShmKEY, sizeof(struct Memory), 0666);
     if (ShmID < 0) {
          printf("*** shmget error (client) ***\n");
          exit(1);
     }
     printf("   Client has received a shared memory of four integers...\n");

     ShmPTR = (struct Memory *) shmat(ShmID, NULL, 0);
     if ((int) ShmPTR == -1) {
          printf("*** shmat error (client) ***\n");
          exit(1);
     }
     printf("   Client has attached the shared memory...\n");

     while (ShmPTR->status != FILLED)
          ;
     printf("   Client found the data is ready...\n");
     printf("   Client found %d %d %d %d in shared memory...\n",
               ShmPTR->data[0], ShmPTR->data[1],
               ShmPTR->data[2], ShmPTR->data[3]);

     ShmPTR->status = TAKEN;
```

```
     printf("   Client has informed server data have been taken...\n");
     shmdt((void *) ShmPTR);
     printf("   Client has detached its shared memory...\n");
     printf("   Client exits...\n");
     exit(0);
}
```

示例3. 使用管道实现进程间通信

示例程序 1

程序先写管道，然后再读管道

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int fd;

    // FIFO file path
    char * myfifo = "/tmp/myfifo";

    // Creating the named file(FIFO)
    // mkfifo(<pathname>, <permission>)
    mkfifo(myfifo, 0666);

    char arr1[80], arr2[80];
    while (1)
    {
        // Open FIFO for write only
        fd = open(myfifo, O_WRONLY);

        // Take an input arr2ing from user.
        // 80 is maximum length
        fgets(arr2, 80, stdin);

        // Write the input arr2ing on FIFO
        // and close it
        write(fd, arr2, strlen(arr2)+1);
        close(fd);
```

```
        // Open FIFO for Read only
        fd = open(myfifo, O_RDONLY);

        // Read from FIFO
        read(fd, arr1, sizeof(arr1));

        // Print the read message
        printf("User2: %s\n", arr1);
        close(fd);
    }
    return 0;
}
```

示例程序2

程序先读管道再写管道

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int fd1;

    // FIFO file path
    char * myfifo = "/tmp/myfifo";

    // Creating the named file(FIFO)
    // mkfifo(<pathname>,<permission>)
    mkfifo(myfifo, 0666);

    char str1[80], str2[80];
    while (1)
    {
        // First open in read only and read
        fd1 = open(myfifo,O_RDONLY);
        read(fd1, str1, 80);

        // Print the read string and close
        printf("User1: %s\n", str1);
        close(fd1);
```

```
        // Now open in write mode and write
        // string taken from user.
        fd1 = open(myfifo,O_WRONLY);
        fgets(str2, 80, stdin);
        write(fd1, str2, strlen(str2)+1);
        close(fd1);
    }
    return 0;
}
```

修改以上代码

1. 将发送字符串修改为以二进制形式收发记录

一条记录内容为

```
struct Student
{
int Id;
char Name[80];
ushort Age;
}
```

2. 在前述内容基础之上，修改Name长度为或Student长度为87,继续运行程序，查看程序是否会问题。

3. 进一步修改程序，程序1读取一条录入的Student信息后，将Student信息发送给程序2，程序2将Student的Id和Age加1后发回给程序1， 程序1接收到Student的信息后，再次增长Id和Age并发给程序2，如此往复循环进行，直至Id增长至100,观察两个程序的输出信息。