SYSProg HW7-1

-IPC 기법을 이용한 프로세스 간 데이터 교환-

학 과 컴퓨터 공학과
학 번 201211704
이 름 김기홍

제 출 일 2015.11.17

목 차

Ⅰ. 서론

1. 개요

이번에 구현한 프로그램은 다른 프로세스간에 데이터를 서로 주고받는 기능을 한다. 조금 더 세부적으로 설명하자면, 간단한 사칙연산 식을 입력하면 다른 프로세스로 그 식을 전달하고, 그 식을 전달받은 프로세스는 계산을 한 후 결과값을 다시 돌려주고 식을 입력했었던 프로세스는 그 결과값을 출력하는 프로그램이다.

구현은 메모리 매핑 방식, 파이프 방식을 사용하였다.

Ⅱ. 본론

1. 프로그램 구현

1) mmap 사용

메모리 매핑 방식으로 IPC를 구현하기 위해 mmap 시스템 콜을 사용하였습니다. mmap를 사용하여 부모 자식 관계의 프로세스들 사이에 데이터 전송을 쉽게 구현할 수 있었습니다.

먼저 매핑을 할 파일을 생성해두고, 그 파일의 정보를 stat() 함수를 이용하여 stat형 구조체에 저장합니다.(이는 후에 mmap함수의 매개변수로 st_size가 필요하기 때문입니다.)

그 이후에는 그 파일을 open()을 이용해서 열고 그 상태에서 mmap()를 사용해서 caddr_t형 변수에 매핑을 시킵니다. 그 이후에는 fork()함수를 이용해주면 다른 프로세스간에 쉽게 매핑된 주소를 이용해 서로 데이터를 교환할 수 있습니다.

구현과정에서 한번 시행착오를 겪었는데, 미리 파일을 만들어두지않고 open에 O_CREAT 플래그를 주어 파일이 없으면 자동으로 생성하게 할 의도였지만, open을 한 후에 stat을 하였더니 거기서 문제가 생겼던 것 같습니다. 이 문제를 해결하기 위해서 fstat()함수를 이용할 수 있었겠지만, 저는 그냥 미리 파일을 만들어 놓은 상태에서 stat()함수를 이용해 정보를 빼 온 다음에 파일을 open하였습니다.

아래는 위에서 설명한 일련의 과정에 대한 부분적인 코드와 실행화면을 첨부하겠습니다.

```
    addr = mmap(0, statbuf.st_size, PROT_READ|PROT_WRITE, MAP_SHARED, fd,
(off_t)0);
    if(addr == MAP_FAILED)
    {
        perror("mmap");
        exit(1);
    }

    close(fd);
```

메모리를 매핑하고 파일 기술자를 닫는 모습입니다. 파일 기술자를 닫은 이후에도 addr 변수를 통해 메모리가 사상되어있기 때문에 read나 write 함수 없이도 addr을 수정함으로써 데이터를 주고받을 수 있습니다.

```
switch(pid=fork())
{
    case -1 :
        perror("fork");
        exit(1);
        break;

    case 0 :
```

fork()함수를 사용해 부모프로세스에 있던 addr 변수까지 자식 프로세스로 복사함으로써 서로 데이터 교환이 가능하게 되었습니다.

```
sprintf(addr, "%d", result);
```

자식 프로세스에서 계산 결과를 addr에 복사함으로써 부모 프로세스에서 결과값을 읽을 수 있도록 하였습니다.

```
Input the formula : 3*7
Child Process : Calculating...
Parent Process : Wait...
The Answer : 21
```

보기 좋게 각각의 과정에서 sleep()함수를 이용하여 시간차를 두었습니다.

2) 이름 없는 pipe 사용

부모 프로세스와 자식 프로세스간 통신을 할 때에는 fork() 함수를 이용하기 때문에 본래 만들어 놓은 파이프 자체도 복사가 됩니다. 이를 이용해서 쉽게 부모 프로세스와 자식 프로세스 간에 데이터 전송을 위한 파이프를 생성할 수 있었습니다.

먼저 크기가 2인 파일 기술자를 2개를 선언한다.(파이프는 데이터의 이동이 단방향이기 때문에 부모 프로세스에서 자식 프로세스로 이어지는 파이프, 자식 프로세스에서 부모프로세스로 이어지는 파이프를 따로 만들어주어야 했습니다.)

이제 pipe()함수를 이용해서 파이프를 2개 생성한 후에 fork()함수를 이용해서 부모 프로세스와 자식 프로세스간에 파이프를 공유합니다. 이제 파이프를 통해서 데이터를 서로 주고 받을 수 있게 되었습니다. 데이터는 write와 read를 통해서 읽고 쓸 수 있습니다.

```c
int fd_ptoc[2];
int fd_ctop[2];
```

먼저 부모에서 자식으로의 파이프를 생성하기 위한 파일기술자, 자식에서 부모로의 파이프를 생성하기 위한 파일기술자 총 2개를 선언하였습니다.

```c
if(pipe(fd_ptoc) == -1)
{
    perror("pipe");
    exit(1);
}

if(pipe(fd_ctop) == -1)
{
    perror("pipe");
    exit(1);
}
```

각각의 파일 기술자들을 이용해 파이프를 생성하는 모습입니다.

```c
switch(pid = fork())
{
    case -1:
        perror("fork");
        exit(1);
        break;

    case 0:
        close(fd_ptoc[1]);
        close(fd_ctop[0]);
```

fork()함수를 이용해 생성해둔 파이프를 자식 프로세스도 공유할 수 있게 합니다. 자식 프로세스에서 쓰지 않을 파이프의 입구나 출구는 닫아두었습니다.

```c
sprintf(buf, "%d", result);
write(fd_ctop[1], buf, len);
```

결과값을 buf에 복사하고 buf를 자식에서 부모로 가는 파이프의 입구에 쓰는 모습입니다.

```
Input the formula : 2*8
Child Process : Calculating...
Parent Process : Wait...
The Answer : 16
```

실행화면만 비교하면 위의 메모리매핑과 같은 화면임을 알 수 있습니다.

3) 이름 있는 pipe 사용

이번에는 부모 자식 관계의 프로세스 사이가 아닌 전혀 다른 프로세스 간의 데이터 전송을 구현하기 위해서 pipe 파일을 사용하였습니다.

먼저 Server 프로세스에서 mkfifo()함수를 이용하여 파이프 파일을 생성합니다. 그 후에 그 파일을 open하여 write()함수로 입력받은 식을 파이프에 실어서 보냅니다. 그럼 이제 Server 프로그램은 Client 프로그램에서 식을 받을 때까지 대기를 하게 됩니다.

이제 Client 프로그램을 살펴보자면, 이번에는 파이프 파일을 연 다음 read를 통해 읽어들입니다. 그 후에 계산을 한 후 결과값을 위와 같은 방식으로(이번에는 Client가 Server가 되고 Server가 Client가 됩니다.) 결과 값을 넘겨주고 Server 프로그램에서는 결과값을 출력합니다.

```c
if(mkfifo("./fifo", 0666) == -1)
{
    perror("mkfifo");
    exit(1);
}

if((pd = open("./fifo", O_WRONLY)) == -1)
{
    perror("open");
    exit(1);
}

n = write(pd, buf, strlen(buf)+1);
if(n==-1)
{
    perror("write");
    exit(1);
}
```

Server에서 mkfifo()함수를 이용해서 파이프 파일을 만들고 파이프 파일을 열어서 데이터를 쓰는 모습입니다.

```c
if((pd = open("./fifo", O_RDONLY)) == -1)
{
    perror("open");
    exit(1);
}

while((n = read(pd, buf, 30)) > 0)
    write(pd, buf, n);
```

이 부분은 Client에서 파이프 파일을 열어서 데이터를 읽어들여서 buf에 저장하는 모습입니다.
결과값을 Client에서 Server로 보내는 작업은 위의 작업을 반대로 하면 되기에 생략하였습니다.

```
hong@hong ~/Desktop/1116 $ ./server

Server ======
Input the formula : 6-3
Sending the formula to Client...
```

먼저 서버 프로그램에서 식을 입력하면 대기를 하는 모습입니다.

```
^Z
[1]+  Stopped                 ./server
hong@hong ~/Desktop/1116 $ bg 1
[1]+ ./server &
hong@hong ~/Desktop/1116 $ ./client

Client ======
Received the formula
Calculating...
Sending the answer to Server...

Server ======
Received the answer
The Answer : 3
[1]+  Done                    ./server
hong@hong ~/Desktop/1116 $
```

이후에 서버를 백그라운드로 전환하고 클라이언트 프로그램을 실행하면 식을 받아 계산하고 다시 서버로 결과값을 보내고, 이후에 서버에서는 그 결과값을 받아서 결과값을 출력하는 것을 알 수 있습니다.