

# 2016025469 컴퓨터공학과 서건식

운영 체제 HW#3

제출 일자 : 2020/04/11

## A. 과제 A

### - Named Pipe

#### 1. 프로그램 설명

##### 1) reader\_Aprocess.c

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

#define PIPENAME "./named_pipe_file"

int main()
{
    int ret;
    char msg[512];
    int fd;
    pid_t pid;
    int i = 0;
    ret = access(PIPENAME, F_OK);
    if(ret == 0)
    {
        unlink(PIPENAME);
    }

    //create a named pipe
    ret = mkfifo(PIPENAME, 0666);

    if(ret < 0)
    {
        printf("Creation of named pipe failed\n");
    }

    //open the named pipe
    fd = open(PIPENAME, O_RDWR);
    if(fd < 0)
    {
        printf("Opening of named pipe failed\n");
        return 0;
    }

    printf("Hello, this is A process. give me the data.\n");
}
```

```

while(1)
{
    ret = read(fd,msg,sizeof(msg));
    if(ret <0)
    {
        printf("Read failed\n");
        return 0;
    }
    printf("%s\n",msg);
    i++;
    if(i==10){
        printf("END\n");
        break;
    }
}
return 0;

```

Named pipe를 통해 데이터를 받은 프로세스입니다.

메커니즘은 2번에 기술하도록 하겠습니다.

추가된 부분은 While문이며, 파일디스크립터의 메시지를 받아와서 ret이 0보다 큰 경우에 메시지를 출력합니다. 최대 10번까지 읽습니다.

## 2) Writer\_Bprocess.c

```

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

#define MSG_SIZE 80
#define PIPENAME "./named_pipe_file"

int main()
{
    char msg[MSG_SIZE];
    int fd;
    int ret;
    int i=0;

    //open the named pipe
    fd = open(PIPENAME,O_WRONLY);
    if(fd<0)
    {
        printf("Open failed\n");
        return 0;
    }

    printf("Hello, this is B process. I'll give the data to A (limit: 10 data)\n");
    while(1)
    {
        scanf("%s",msg);
        ret = write(fd,msg,sizeof(msg));
        if(ret <0)
        {
            printf("Write failed\n");
            return 0;
        }
        i++;
        if(i == 10){
            printf("END\n");
            break;
        }
    }
}

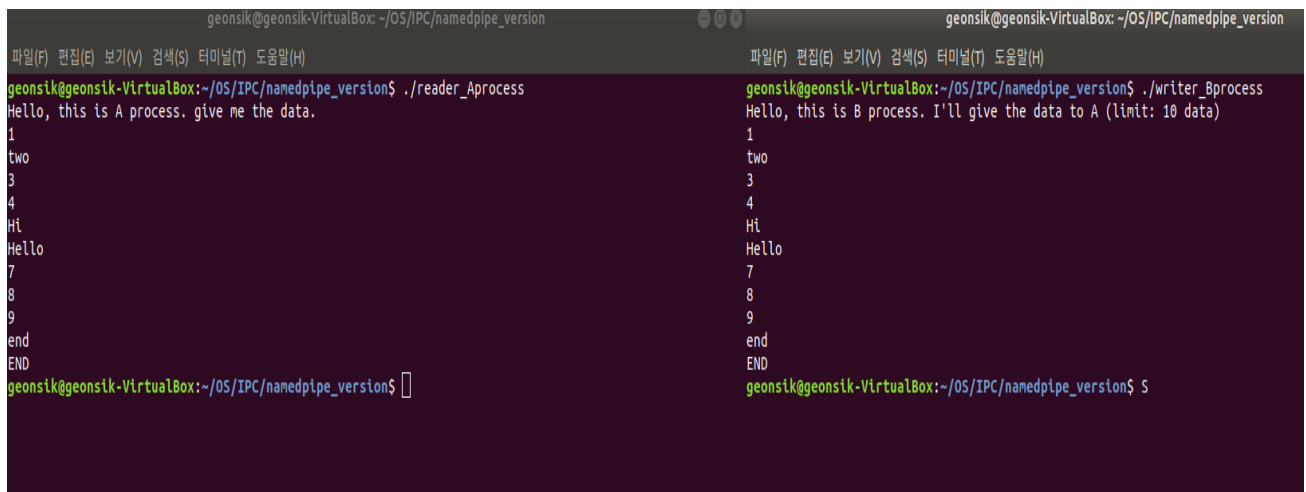
```

이 또한 named\_pipe\_file을 파일디스크럽터로 open후에 while문에서 10번까지 입력하는 프로세스입니다.

## 2. IPC 메커니즘 설명

Reader\_Aprocess 에서 mkfifo를 통해 파이프라인을 형성해야 하기 때문에 해당 실행파일을 먼저 실행하고, writer\_Bprocess를 실행한 후에 여기서 해당 파이프에 문자를 입력하면, A 프로세스에서 해당 값을 받습니다. 동기화가 되기 때문에 동기화 문제가 없습니다.

실행결과입니다.



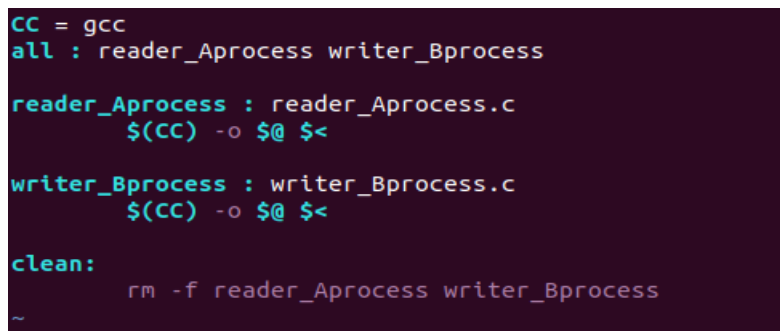
```
geonsik@geonsik-VirtualBox: ~/OS/IPC/namedpipe_version
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
geonsik@geonsik-VirtualBox:~/OS/IPC/namedpipe_version$ ./reader_Aprocess
Hello, this is A process. give me the data.
1
two
3
4
Hi
Hello
7
8
9
end
END
geonsik@geonsik-VirtualBox:~/OS/IPC/namedpipe_version$

geonsik@geonsik-VirtualBox: ~/OS/IPC/namedpipe_version
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
geonsik@geonsik-VirtualBox:~/OS/IPC/namedpipe_version$ ./writer_Bprocess
Hello, this is B process. I'll give the data to A (limit: 10 data)
1
two
3
4
Hi
Hello
7
8
9
end
END
geonsik@geonsik-VirtualBox:~/OS/IPC/namedpipe_version$ S
```

## 3. 컴파일 방법 설명

Makefile을 만들어 컴파일하였습니다.

Makefile은 밑에 첨부하겠습니다.



```
CC = gcc
all : reader_Aprocess writer_Bprocess

reader_Aprocess : reader_Aprocess.c
$(CC) -o $@ $<

writer_Bprocess : writer_Bprocess.c
$(CC) -o $@ $<

clean:
rm -f reader_Aprocess writer_Bprocess
```

## - Message Queue

### 4. 프로그램 설명

#### 1) reader\_Aprocess.c

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

typedef struct msgbuf
{
    long msgtype;
    char mtext[20];
} msgbuf;

int main(int argc, char **argv)
{
    key_t key_id;
    int i=0;
    msgbuf rsvbuf;
    int msgtype = 3;

    key_id = msgget((key_t)1234, IPC_CREAT|0666);

    if(key_id == -1)
    {
        perror("msgget error : ");
        return 0;
    }
    while(1){
        if(msgrcv(key_id, (void *)&rsvbuf , sizeof(msgbuf), msgtype , 0) == -1)
        {
            perror("msgrcv error : ");
        }
        else
        {
            printf("The %dth received message is: %s\n",i,rsvbuf.mtext);
        }
        i++;

        if(i==10)
        {
            printf("You received 10 messges. END\n");
            break;
        }
    }
}
```

구조체는 msgbuf로 선언하였습니다. 메커니즘은 2번에 기술하도록 하겠습니다.

반복문에서, 리시브버퍼 함수를 실행할 때 오류가 없다면 리시브버퍼의 내용을 출력합니다. 10번째 메시지를 받으면 종료합니다.

## 2) Writer\_Bprocess.c

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

typedef struct msgbuf
{
    long msgtype;
    char mtext[20];
} msgbuf;

int main(void)
{
    key_t key_id;
    int i=0;
    msgbuf sndbuf;

    key_id = msgget((key_t)1234, IPC_CREAT|0666);

    if(key_id == -1)
    {
        perror("msgget error : ");
        return 0;
    }

    sndbuf.msgtype = 3;
    while(1){

        scanf("%s",&sndbuf.mtext);

        if(msgsnd(key_id,(void *)&sndbuf, sizeof(msgbuf),IPC_NOWAIT) == -1)
        {
            perror("msgsnd error : ");
        }
        printf("Sending %dth message is succeed\n",i);
        i++;
        if(i == 10){
            printf("You sent 10 messages. END\n");
            break;
        }

    }

    printf("send");
    return 0;
}
```

writer이며, scanf로 sndbuf 내의 멤버변수 mtext에 입력 후 msgsnd 함수로 버퍼에 sndbuf를 담습니다.

최대 10번까지 보낼 수 있습니다.

실행결과 입니다.

```
geonsik@geonsik-VirtualBox:~/OS/IPC/messagequeue_version$ ./reader_Aprocess
The 0th received message is: 1
The 1th received message is: 2
The 2th received message is: two
The 3th received message is: 4
The 4th received message is: 5
The 5th received message is: 123
The 6th received message is: 555
The 7th received message is: 777
The 8th received message is: 10
The 9th received message is: 32
You received 10 messages. END
geonsik@geonsik-VirtualBox:~/OS/IPC/messagequeue_version$

geonsik@geonsik-VirtualBox:~/OS/IPC/messagequeue_version$ ./writer_Bprocess
1
Sending 0th message is succeed
2
Sending 1th message is succeed
two
Sending 2th message is succeed
4
Sending 3th message is succeed
5
Sending 4th message is succeed
123
Sending 5th message is succeed
555
Sending 6th message is succeed
777
Sending 7th message is succeed
10
Sending 8th message is succeed
32
Sending 9th message is succeed
You sent 10 messages. END
sendgeonsik@geonsik-VirtualBox:~/OS/IPC/messagequeue_version$
```

## 5. IPC 메커니즘 설명

Reader는 consumer, writer는 Producer 입니다. 커널에서 관리하는 메시지 큐에 Producer가 자료를 넣으면, 선입선출이기 때문에 consumer가 해당 값을 빼내 옵니다.

## 6. 컴파일 방법 설명

위와 동일합니다.

## - Shared Memory

### 7. 프로그램 설명

#### 1) reader\_Aprocess.c

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

#define READ_FLAG 0
#define WRITE_FLAG 1
#define PRINT_FLAG 2

int main(void)
{
    int shmid;
    char *shmaddr;
    char *string;
    int ret;
    int i=0;

    //get shared memory id
    shmid = shmget((key_t)1234, 1024, IPC_CREAT|0666);
    if(shmid == -1)
    {
        perror("shared memory access is failed\n");
        return 0;
    }

    //attach the shared memory to process memory space
    shmaddr = shmat(shmid, NULL, 0);
    if(shmaddr == (char*)-1)
    {
        perror("attach failed\n");
        return 0;
    }
    string = shmaddr+1;
    shmaddr[0] = READ_FLAG;

    while(1){
        if(shmaddr[0] == WRITE_FLAG)
        {
            printf("data read from shared memory: %s\n", string);
            i++;
            shmaddr[0] = READ_FLAG;
            if(i==10)
            {
                printf("Receive 10 messages. END\n");
                break;
            }
        }
    }
    ret = shmdt(shmaddr);
    if(ret == -1)
    {
        perror("detach failed\n");
        return 0;
    }

    ret = shmctl(shmid, IPC_RMID, 0);
    if(ret == -1)
    {
        perror("remove failed\n");
        return 0;
    }
    return 0;
}
```

Shared memory는 메시지큐나 파이프와 다르게 프로세스간 동기화가 이뤄지지 않습니다. 따라서 Shared memory에 Writer가 데이터를 입력했다는 Flag를 입력함으로써 동기화를 해줄 수 있습니다.

Flag는 READ,WRITE가 있습니다. PRINT는 사용하지 않았습니다.

string이라는 변수는 문자를 저장한 곳을 가리키는 포인터 변수이며, shmaddr[0]+1 을 가리킵니다. shmaddr[0]에는 플래그를 삽입합니다. 처음에는 READ\_FLAG를 넣습니다.

while문에서, flag가 WRITE\_FLAG로 바뀔 때 까지 if문에서 대기합니다. writer에서 쓰기를 마치고 플래그를 바꾸면, 그 때 if문 값이 참이 되면서 데이터를 읽습니다. 읽자마자 플래그를 READ상태로 바꾸기 때문에, 다음 플래그가 올 때 까지 대기합니다.

## 2) Writer\_Bprocess

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

#define MSG_SIZE 80
#define READ_FLAG 0
#define WRITE_FLAG 1
#define PRINT_FLAG 2

int main(void)
{
    int shmid;
    int i = 0;
    char *shmaddr;
    char *string;
    char *msg[MSG_SIZE];
    int ret;
    shmid = shmget((key_t)1234, 1024, IPC_CREAT | 0666);

    if(shmid < 0)
    {
        perror("shmget error");
        return 0;
    }

    //attach

    shmaddr = shmat(shmid, NULL, 0);
    if(shmaddr == (char*)-1)
    {
        perror("attach failed\n");
        return 0;
    }
}
```



```

string = shmaddr + 1;
shmaddr[0] = READ_FLAG;
while(1)
{
    scanf("%s",msg);
    if(shmaddr[0] == READ_FLAG)
    {
        strcpy(string,msg);
        shmaddr[0] = WRITE_FLAG;
        i++;
        if(i == 10)
        {
            printf("Send 10 messages. END\n");
            break;
        }
    }
}
ret = shmdt(shmaddr);
if(ret == -1)
{
    perror("detach failed\n");
    return 0;
}
return 0;
}

```

Reader와 거의 동일합니다만, 처음에 실행하는 것이 A process 이기 때문에 shmaddr[0]은 READ\_FLAG일것입니다. 그것이 아니더라도 while문 전에 첫 입력이기 때문에 flag를 교체해줬습니다. while문에서 READ FLAG일 경우 입력받은 값을 string에 저장하고, 플래그를 WRITE\_FLAG로 바꿔줍니다. 바꿔주는 동시에 A process가 읽게됩니다.

실행결과 입니다.

geonsik@geonsik-VirtualBox: ~/OS/IPC/sharedmemory_version	geonsik@geonsik-VirtualBox: ~/OS/IPC/sharedmemory_version
<pre> 파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H) geonsik@geonsik-VirtualBox:~/OS/IPC/sharedmemory_version\$ ./reader_Aprocess data read from shared memory: hi data read from shared memory: reader data read from shared memory: In data read from shared memory: B data read from shared memory: Process. data read from shared memory: Nice data read from shared memory: to data read from shared memory: meet data read from shared memory: you. data read from shared memory: !! Receive 10 messages. END geonsik@geonsik-VirtualBox:~/OS/IPC/sharedmemory_version\$ </pre>	<pre> 파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H) geonsik@geonsik-VirtualBox:~/OS/IPC/sharedmemory_version\$ ./writer_Bprocess hi reader In B Process. Nice to meet you. !! Send 10 messages. END geonsik@geonsik-VirtualBox:~/OS/IPC/sharedmemory_version\$ </pre>

## 8. IPC 메커니즘 설명

기본적으로 shared memory는 동기화를 지원하지 않는 이유가, FLAG가 없다면 reader는 그대로 해당 번지에서 값을 가져와 읽어오기 때문에 아무리 writer가 입력값을 넣는다고 하더라도 reader는 읽어간 시간에 존재하는 데이터만을 가지고 오게 됩니다.

이것이 shared memory 가 플래그를 통해 데이터 입력의 유무를 판별해야 하는 이유입니다.

## 9. 컴파일 방법 설명

위와 동일합니다.

## B. 과제 B

### 1. 멀티스레딩 내용

쓰레드 : Lightweight process

\*쓰레드 또한 실행의 단위이며, 스케줄링의 대상이다.

한 프로세스의 resources를 공유하며, 공유하는 것은 address space 중에서 code,data 영역을 공유한다. stack 공간과 processor context도 독자적으로 가진다.

이렇게 쓰레드를 생성하여 멀티스레딩을 통한 프로그래밍의 이점은, 응답성이 뛰어나다는 것이다. 만약 cpu와 process가 1개이고, 프로세스가 실행중이다가 I/O 디바이스의 처리로 wait상태가 되었다 하자. 쓰레드로 구현하게 되면 이 wait상태 에서도 CPU를 작동시키는게 가능하다. 또한 메모리 자원을 공유하며, 쓰레드간 데이터 전송이 아주 쉽고 저렴하다. 또한 cpu가 8개고 process가 1개일 때, 쓰레드를 8개 만들어서 cpu를 전부 사용할 수 있도록 할 수 있다. 그리고 프로그램은 본래 sequential하게 돌아가는데, 쓰레드를 사용해서 동시에 작업하는 일,사건,작업을 설계하고 구현하는 것이 쉬워진다. (Concurrent programming model)

이러한 멀티스레딩은 웹 서버에서도 많이 사용되는데, process 를 사용자 수 만큼 fork 하는 것이 아닌 쓰레드를 사용자수 만큼 만든다.

또한 쓰레드는 유저 쓰레드와 커널 쓰레드로 구분 되는데, 일반적으로 유저 쓰레드는 Many to one 방식으로 커널에서 봤을 때는 한 프로세스로만 보인다. 커널 쓰레드는 One to one / Many to many 모델이 있다.

쓰레드에 대한 이슈들도 있으며, 예를 들면 User level 에서 일어나는 스택 오버플로우 이다. 싱글 쓰레드 일 경우 스택이 다른 영역을 침범하면 exception 이 발생하지만, 멀티쓰레드일 경우에는 커널이 이를 한 프로세스 로만 보기 때문에 독립적으로 만들어진 스택 영역을 서로 침범할 수가 있다. 이를 대비할 필요가 있다.