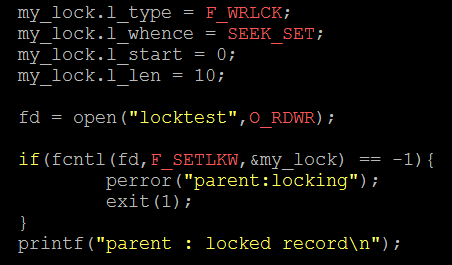
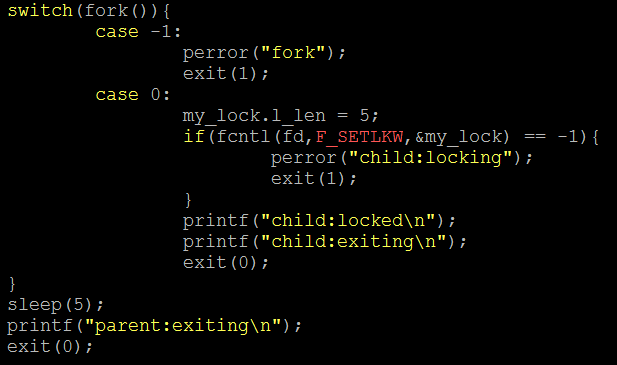
**LAB7 –Advanced IPC 20113315 이형준**

1. [강의노트](http://advsys.cs.kookmin.ac.kr/home/Lecture8-AdvancedIPC.pdf?attredirects=0&attachauth=ANoY7cobM3ZjSLmxPMgDAZzAOcsCtFczPUuYhbMrYaXGU7RBNJ5TLJSrAaqTWkXmGQVVj9dzVAf_Q5cZHl4GIcvqNN7ZKzjHBnKUp8xucrYdR4Ytk58MCRClLu9lsUijYTvrLSy4LOFHE36_y_6jA1hrt3CUwgZjQegMPhvjSho_UJZDKmWJKX51fdf9ezJInUZ4BfTpof_rr8jTB88X7xJwh763zdaV7NZLI67udhcHLCw3VHlwWzo%3D) p6, p10의 프로그램을 실행 시키시오.
2. P6 simple example.



“locktest”란 파일을 열고, my\_lock을 얻습니다. Parent가 합니다.

Lock은 0부터 10까지 얻습니다.



Parent는 fork를 합니다.

Child는 0부터 5까지 my\_lock을 얻으려고 합니다.

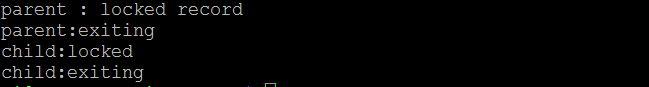
하지만 parent가 이미 lock을 얻은 상태입니다.

F\_SERLKW 옵션으로 인해서, child는 락을 얻으려고 기다리는 상태로 들어가게 됩니다.

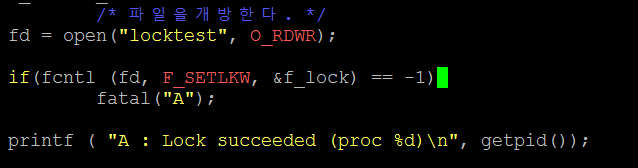
대략 5초 후 parent가 종료가 되고, lock을 반납하게됩니다.

그 때, clide는 lock을 얻고 정상종료 하면서 lock을 반납하게 됩니다.

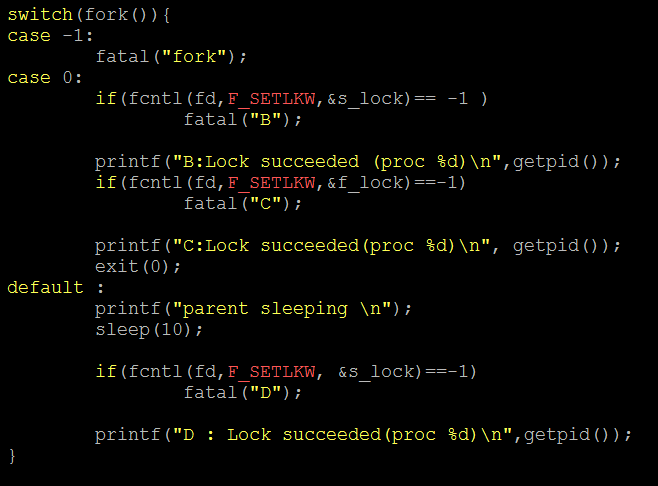
* **컴파일 결과**



1. P10, Dead lock example



Parent가 0부터 10까지 first lock을 얻습니다.

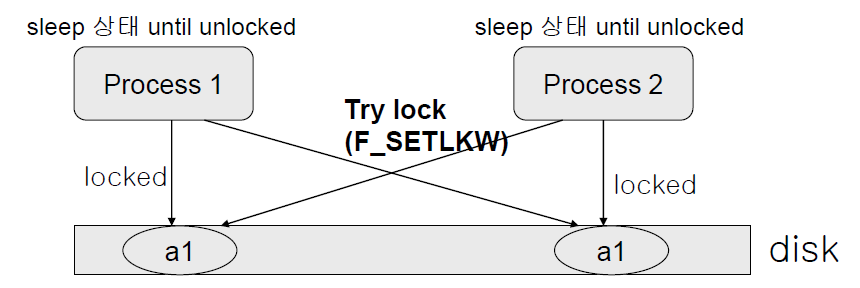


Parent 가 fork를 합니다. 그리고 sleep(10) 합니다.

Child는 second lock을 얻습니다.

그리고 child는 first lock 을 얻으려고 합니다. 이미 parent가 얻고 있는 상태 이므로, “cmd”옵션 F\_SETLKW에 의해서 기다리게 됩니다.

Parent는 sleep(10) 이후에 일어나서 second lock을 얻으려고 합니다. 이 때, 교착상태가 됩니다. 즉 Deadlock입니다.

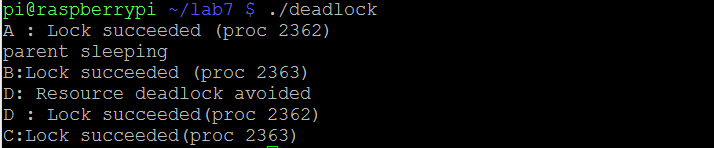


교착상태가 되면 커널에서 자동으로 avoided를 해 줍니다.

따라서 child가 first lock을 얻게 되고,

Parent도 second lock을 얻게 됩니다.

* **컴파일 결과**



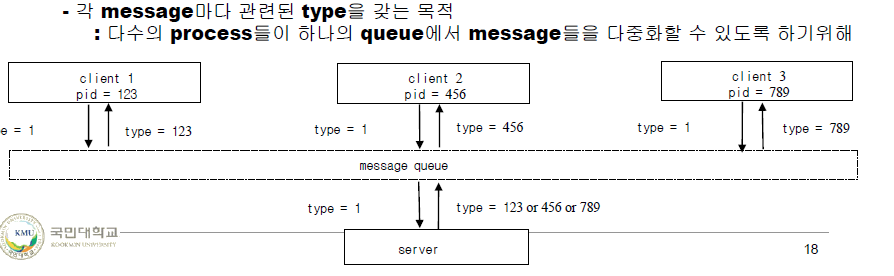
1. [**강의노트**](http://advsys.cs.kookmin.ac.kr/home/Lecture8-AdvancedIPC.pdf?attredirects=0&attachauth=ANoY7cobM3ZjSLmxPMgDAZzAOcsCtFczPUuYhbMrYaXGU7RBNJ5TLJSrAaqTWkXmGQVVj9dzVAf_Q5cZHl4GIcvqNN7ZKzjHBnKUp8xucrYdR4Ytk58MCRClLu9lsUijYTvrLSy4LOFHE36_y_6jA1hrt3CUwgZjQegMPhvjSho_UJZDKmWJKX51fdf9ezJInUZ4BfTpof_rr8jTB88X7xJwh763zdaV7NZLI67udhcHLCw3VHlwWzo%3D) p18 의 구조로 Client-Server 프로그램을 만드시오.

Message queues를사용한 client-server

FIFO 를 사용한 client-server 구현의 단점을 보완하고자 Message queues 를 이용합니다.

즉, 각각의 클라이언트에 매칭되는 여러 개의 FIFO가 아닌, 하나의 Message queue로 client-server를 구현하였습니다.

Message Queue



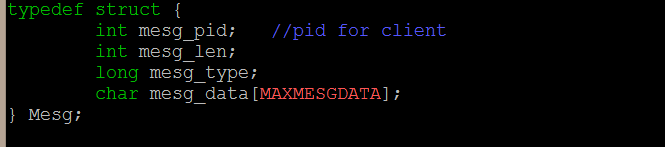
메시지 큐는 각각 type별로 데이터를 읽거나 쓸 수 있습니다.

Server : “ type=1” 로 통신을 하게 됩니다.

Client : “type = getpid()” 로 통신을 하게 됩니다.

각각 client마다 type을 다르게 할당하여, 다수의 process들이 다중화 할 수 있도록 합니다.

Mesg 구조체에 pid를 같이 넘기기 위해서 int mesg\_pid; 를 포함시킵니다.



다음은 코드가 대략적으로 어떻게 돌아가는지 작성한 것입니다.

* Client.c

1. 파일명을 stdin으로 입력받는다.
2. Type=1으로 파일명(mesg.mesg\_data) 를 보낸다.

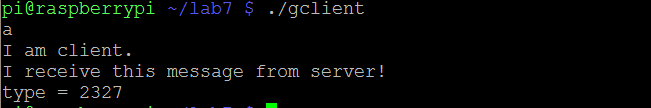
* Server.c

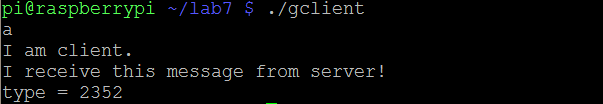
1. Type=1로 해서 client로부터 전송받을 수 있도록 세팅한다.
2. Recv를 통해서 mesg 구조체를 넘겨받는다.
3. 받은 구조체 안의 mesg\_pid 를 참조하여, Type을 client의 pid로 변경한다.
4. Open함수를 통해서 파일명(Mesg.mesg\_data) path를 가지고 fd를 얻는다.
5. Fd로 read한다. Read한 값은 line by line으로 mesg.mesg\_data배열에넣는다.
6. 한 line이 넣어지면 그 구조체를 client로 send한다

* Client

1. Type을 변경해서 그 type으로 recv한다.
2. line단위로 stdout으로 출력한다.

* **컴파일 결과**





* 하나의 Message queue로 client의 pid를 이용해서, 각각 client에게 메시지를 다르게 보낼 수 있습니다.
* 위의 예시에서는 pid:2327, 2325 두개의 client가 참여하고 있습니다.
* Client의 type을 1로 설정한다는 의미는, server에게 보낸다는 뜻입니다.