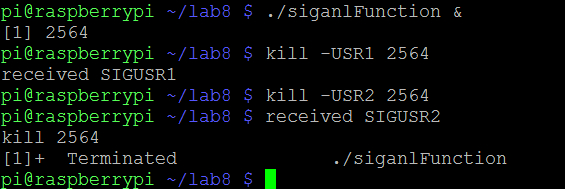
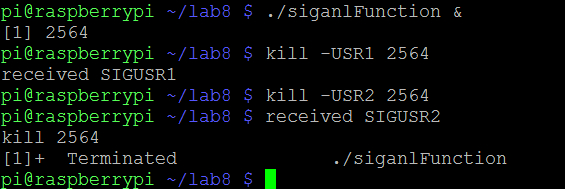
**LAB8 20113315 이형준.**

1. [강의노트](http://advsys.cs.kookmin.ac.kr/home/Lecture7-Interprocesscommunications.pdf?attredirects=0&attachauth=ANoY7cpx0CYf3-X8U-oZ2Rm1B6IZNQoOnpba9wbwBChuS9h5DCSNbmsN9MjKksSSjP0g18-JFNtoYOdqHDqzW2vPhYwhEiiNBvwk0cLKBmgruHWGHWBB_mgwEuriqLt60L9WcZL4tYaBG_or2nUdIcDxFG-p8tXdH1CAtBx5IMkBTZnfaG6Yb2B6aDcgqpbm4UOqmyObBaD62D3DMetHM4c5_KIzpEPHvBBwokURA5aBMPXKrIDkXZP777a46tGCyhfwQCowtrba) p12 프로그램을 실행시키고 관찰하시오.



1. signalFuction.c 를 백그라운드로 실행시킵니다.
2. Kill 함수를 이용해서 usr1 시그널을 2564에게 보냅니다.
3. 신호가 catch 되어, received SIGUSR1을 출력 하게되고, 프로세스는 죽지 않습니다.

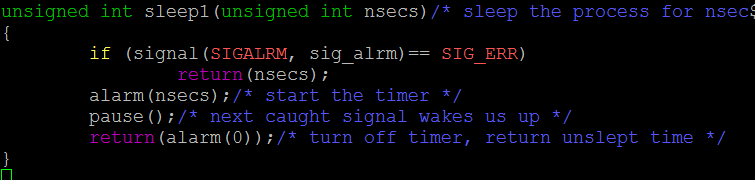


1. Kill –USR2 를 이용해서 signal을 보냅니다.
2. 신호가 catch 되어, received SIGUSR2을 출력 하게되고, 프로세스는 죽지 않습니다.
3. Kill 2564 명령어를 입력합니다
4. Catch 되지 않고 default로 되서 프로세스가 죽게 됩니다.

CF) 만약 kill –USR1 2564 명령어 이후에 한번더 kill –USR1 2564 를 입력한다면?

* unreliable이라서, catch는 1회용입니다. 그래서 default로 작용되서 프로세스가 죽게 됩니다. Signal을 catch해주는 sig\_usr함수를 재 정의 하더라도, 재 정의 하기전에 signal이 들어온다면, 프로세스가 죽을 수 있습니다. ( asynchronous )

1. [강의노트](http://advsys.cs.kookmin.ac.kr/home/Lecture5-Signals.pdf?attredirects=0&attachauth=ANoY7cqvzWv38BW5Dw9IoX6BqlzMF4KCBAbMy4w6A6TCCRklC29B5HUEuH_biC66guHGNY91_Cigy7jkUOoZFyGTNF2HiNSi3F5YAuneAMlxeiAf2C-kkRlT-hvLO2-G1KlYlvGb-ykVjiPsBofYutKI1odhMuB9MaNyAHy92Q91Z8F860IhnNHvAEnrIvsdzDZOYFCOF1tzRrwsMvz337RNM1sqmKwckDksMwdHHbP12PWrUgtm1nY%3D) p23,24,25 의 sleep1, sleep2를 실행시키시오.
2. Sleep1.c



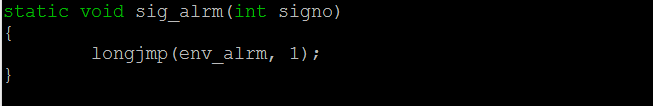
* 문제점 : alarm()과 pause() 는 atomic 하지 않아서 그 사이에 context switching이 일어나 다른 일을 하게 될 수 있는데, 다른 일이 오래 진행 되어 버리면 그 사이 alarm이 먼저 울리게 되고, pause는 영원히 block상태가 되어 버린다.

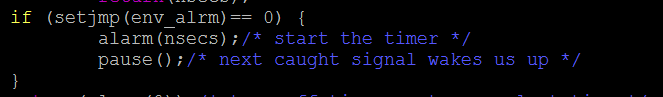
**즉, alarm이 울린 상태라도 pause를 한다.**

Alarm(0)은 혹시 pause가 알람에 의해서가아닌 다른 시그널에 의해서 pause가 깨어났을 때, 남아있는 초를 리턴해준다.

* 실행결과 : 대부분(99%) 이상 pause가 깨어나게 된다.

1. Sleep2.c





* Sleep1.c 의 atomic문제점을 해결했다.

**alarm이 울리면 pause를 안 하게 만든다.**

* Setjump함수 사용하면, 현재 상태가 “env\_alrm ”에 저장하고 0을 리턴합니다.

그리고 alarm()과 pause()를 실행하게 됩니다.

Sleep1.c에서 alarm() 함수와 pause()함수 사이에 또 다른 인터럽트가 들어와서 pause가 실행이 안될 수 있는데, sleep2.c에서는 alarm()이 오면 longjmp 함수가 실행되게 작성되어있습니다.

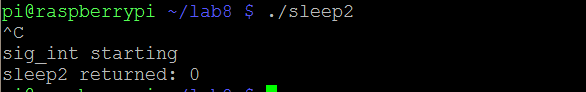
Longjmp(env\_alrm,1)

Longjmp : setjmp가 저장해 놓은 “env\_alrm” 상태로 되돌리게됩니다. Longjmp가 실행되면 setjmp함수 속으로 점프하게 되는데, 이때 setjmp가 1을 return 하게됩니다. 즉 **alarm이 울리면 pause를 안 하게 됩니다**

* Sleep2.c 의 문제점

Sleep2 함수의 alarm이 울리기 전에 SIGINT(^C)가 들어왔는데, 그 신호를 catch 하는 sig\_int함수가 너무 오래 실행되어, 알람이 울리게된된다면, longjmp가 된다. Longjmp는 setjmp에서 저장한 상태로 되돌리며, 스텍 위에 올려진 sig\_int가 날라가게 된다. 즉 Sig\_int함수에서 실행되고있는 출력값이 나오지 않는다.

* 실행결과



Sig\_int함수의 실행시간이 5초이상 걸려서 longjmp 때문에 스텍이 날라가게된다.

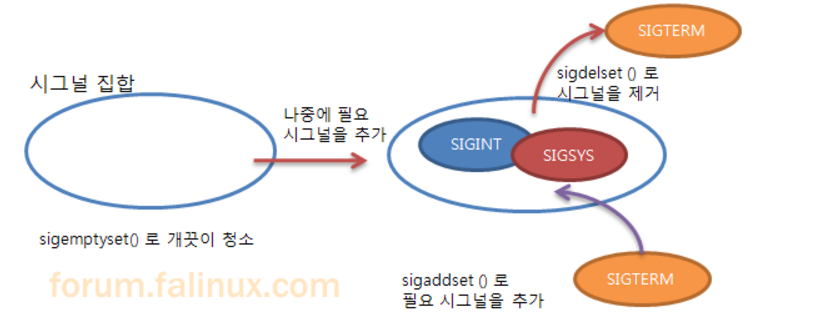
1. [강의노트(Signal 2)](http://advsys.cs.kookmin.ac.kr/home/Lecture6-Signals2.pdf?attredirects=0&attachauth=ANoY7cpexvB3Bj_XGhubE1sEjDi4o2UeVl9DRKb2L58ZVW-8Dg6bIS_44geZA3_e2bQY-D_hqgT8GOWdROaq5scYsBqroapl60yzTUJTGTH3VAYEB5Os3_5rJsolI7wJi7yUruWH-3Z8tiRKCkWItaTPrj2yKpdN0p63SgLPvQBQvehqP5K7I3U1na1gMBLuAqUSlTi42Yr8Cg3_c0ExaHuL9AnOMrqOJHv8WqNi-1mZOMVuXoJthn8%3D) p11,12 의 코드를 실행시키고 출력된 메세지가 어떤 상황에 나온 것인지 설명하시오.

시그널 여러 개를 하나로 묶어서 한꺼번에 처리 하게 할 수 있다. 빈 주머니에 내가 처리 할 시그널들을 하나씩 넣어서 handle 해 준다.

**Sigemptyset(sigset\_t \*set)** : 시그널 집합 내용을 모두 삭제

* Sigset\_t : (시그널 집합변수)

**Sigaddset(sigset\_t \* set, int signum)** : 시그널 집합 변수에 시그널을 추가합니다.



**Sigpromask(int how, const sigset\_t, sigset\_t \*oldset):**

일 처리 중간에 시그널이 발생했더라도, 일을 먼저 처리하고 나중에 발생한 시그널을 처리해 준다. 이때, 시그널은 대기상태가 된다. 시그널 대기 상태로 설정.

* how :

SIG\_BLOCK :기존 블록화된 시그널 집합에 두 번쨰 인수 set시그널 집합을 추가.

SIG\_UNBLOCK : 기존 블록화된 시그널 집합에서 두 번째 인수 set 시그널 집합에 있는 시그널을 제거.

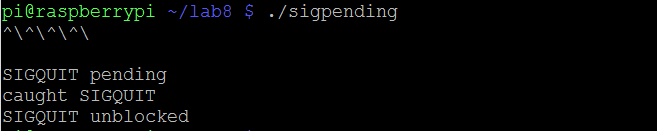
SIG\_UNMASK : 이전 블록된 시그널 집합을 모두 지우고 두 번째 인수인 set 시그널 집합으로 변경.

**Sigpending(sigset\_t \*set) :** 블록된 시그널 확인

* return : 성공 : 0 / 실패 : -1

**Sigismember(sigset\_t \*set, int signum) :** 시그널 집합(첫번째 파라미터)에 시그널(두번째 파라미터)이 있는지 확인

* Return value : 1 : 집합에 시그널이 있음 . 0: 없음 -1 : 실패
* 컴파일 결과



프로그램을 실행시키고 , 5초가 지나지 않은 상태에서 SIG\_QUIT (ctrl\)을 4번 신호를 줍니다. sigprocmask() 함수를 사용하였기에, 바로 반응을 하지 않고 시그널이 대기상태로 변환합니다. 그리고 5초 후에 반응하게 됩니다. 또한 newmask 집합을 추가합니다. sigismember(&pendmask, SIGQUIT)) 집합에 SIGQUIT이 있으므로 1이 반환됩니다. 따라서 SIGQUI pending 문구가 출력됩니다. sig\_quit함수가 caught를 하게 됨을 알 수 있게 하는 문구가 출력됩니다. 그리고 마지막으로 unblocked 되고 5초후 종료하게 됩니다.