오픈소스 소프트웨어 실습

Open-Source Software Lab

#6

실습 담당 조교 연락처

◆실습 조교 : 김민곤

◆연구실 : 학연산클러스터 601호

◆이메일 : phenix235@hanyang.ac.kr

메일양식: [오픈소스]학번_수업일(20190000)_이름

프로세스 상태 보기: ps

$ ps [-옵션]

현재 시스템 내에 존재하는 프로세스들의 실행 상태를 요약해서 출력한다.

```
[1111222333@node1 ~]$ ps
  PID TTY          TIME CMD
 4221 pts/0    00:00:00 bash
 4268 pts/0    00:00:00 ps
```

```
[1111222333@node1 ~]$ ps -u
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
1111222+   4221  0.3  0.0 116228  2836 pts/0    Ss   05:22   0:00 -bash
1111222+   4269  0.0  0.0 155328  1880 pts/0    R+   05:22   0:00 ps -u
```

프로세스 출력정보

항목	의미
UID	프로세스를 실행시킨 사용자 ID
PID	프로세스 번호
PPID	부모 프로세스 번호
C	프로세스의 우선순의
STIME	프로세스의 시작 시간
TTY	명령어가 시작된 터미널
TIME	프로세스에 사용된 CPU 시간
CMD	실행되고 있는 명령어(프로그램) 이름

특정 프로세스 리스트 : pgrep

$ pgrep [옵션] [패턴]

패턴에 해당하는 프로세스들만을 리스트 한다.

-l : pid와 함께 프로세스의 이름을 출력한다.

-f : 명령어의 경로도 출력한다.

-n : 패턴과 일치하는 프로세스들 중에서 가장 최근 프로세스만을 출력한다.

-x : 패턴과 정확하게 일치되는 프로세스만 출력한다.

...

특정 프로세스 리스트 : pgrep

```
[111222333@node1 ~]$ pgrep sshd
1512
4174
4220
[111222333@node1 ~]$ pgrep -l sshd
1512 sshd
4174 sshd
4220 sshd
[111222333@node1 ~]$
```

쉘 재우기

$ sleep 초

명시된 시간만큼 프로세스 실행을 중지시킨다.

```
[1111222333@node1 ~]$ (echo start; sleep 5; echo end; )
start
end
[1111222333@node1 ~]$
```

강제 종료

강제 종료 : ctrl – c

실행 중지 : ctrl - z

```
[lll1222333@node1 ~]$ (echo start; sleep 5; echo end; )
start
^C
[lll1222333@node1 ~]$ (echo start; sleep 5; echo end; )
start
^Z
[1]+  Stopped                 ( echo start; sleep 5; echo end )
[lll1222333@node1 ~]$
```

전면 작업의 후면 전환 : bg

$ bg %작업번호

작업번호에 해당하는 중지된 작업을 후면 작업으로 전환하여 실행한다.

```
[111122333@node1 ~]$ sleep 100; echo DONE
^Z
[2]+  Stopped                 sleep 100
DONE
[111122333@node1 ~]$ bg %2
[2]+ sleep 100 &
```

프로세스 끝내기 : kill

$ kill 프로세스 번호

$ kill %작업번호

프로세스 번호(혹은 작업 번호)에 해당하는 프로세스를 강제로 종료 시킨다.

```
[1111222333@node1 class2]$ (sleep 10; echo DONE)
^Z
[2]+  Stopped                 ( sleep 10; echo DONE )
[1111222333@node1 class2]$ kill %2
[2]-  종 료 됨                ( sleep 10; echo DONE )
```

프로세스의 사용자 ID

$ id [사용자명]

사용자의 실제 ID와 유효 사용자 ID, 그룹 ID등을 보여준다.

프로세스는 프로세스 ID 외에 프로세스의 사용자 ID와 그룹 ID를 갖는다. 프로세스를 실행시킨 사용자의 ID와 사용자의 그룹 ID

```
[1111222333@node1 class4]$ id
uid=1003(1111222333) gid=1004(1111222333) groups=1004(1111222333) context=unconfine
d_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

시그널(signal)

- Signal은 프로세스에게 어떤 사건의 발생을 알릴 때 사용
- 각 신호는 관련 default 동작이 있음 : exit, core, stop, ignore
- 신호이름은 파일 <signal.h>에 매크로로 정의되어 있음
- 목록 확인 방법
  - #) kill -l

```
[1111222333@node1 class4]$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

시그널(signal)

시그널 이름	의미	기본 처리
SIGABRT	abort()에서 발생되는 종료 시그널	종료(코어 덤프)
SIGALRM	자명종 시계 alarm() 울림 때 발생하는 알람 시그널	종료
SIGCHLD	프로세스의 종료 혹은 중지를 부모에게 알리는 시그널	무시
SIGCONT	중지된 프로세스를 계속시키는 시그널	무시
SIGFPE	0으로 나누기와 같은 심각한 산술 오류	종료(코어 덤프)
SIGHUP	연결 끊김	종료
SIGILL	잘못된 하드웨어 명령어 수행	종료(코어 덤프)
SIGIO	비동기화 I/O 이벤트 알림	종료
SIGINT	터미널에서 Ctrl-C 할 때 발생하는 인터럽트 시그널	종료
SIGKILL	잡을 수 없는 프로세스 종료시키는 시그널	종료
SIGPIPE	파이프에 쓰려는데 리더가 없을 때	종료

시그널(signal)

시그널 이름	의미	기본 처리
SIGPIPE	파이프에 쓰려는데 리더가 없을 때	종료
SIGPWR	전원고장	종료
SIGSEGV	유효하지 않은 메모리 참조	종료(코어 덤프)
SIGSTOP	프로세스 중지 시그널	중지
SIGSTP	터미널에서 Ctrl-Z 할 때 발생하는 중지 시그널	중지
SIGSYS	유효하지 않은 시스템 호출	종료(코어 덤프)
SIGTERM	잡을 수 있는 프로세스 종료 시그널	종료
SIGTTIN	후면 프로세스가 제어 터미널을 읽기	중지
SIGTTOU	후면 프로세스가 제어 터미널에 쓰기	중지
SIGUSR1	사용자 정의 시그널	종료
SIGUSR2	사용자 정의 시그널	종료

예제 프로그램 – sigaction (SIGINT 처리)

사용법

```
#include <signal.h>

int sigaction(int signo, const struct sigaction *act, struct sigaction *oact);
```

특정 시그널의 수신에 대해서 취할 액션을 설정하거나 변경하기 위해서 사용된다

signo
- 행동을 지정하고자 하는 개개 시그널을 식별
- SIGSTOP, SIGKILL을 제외한 모든 시그널 이름 지정가능

act : signo에 대해 지정하고 싶은행동
oact: 현재 설정 값을 채운다.

예제 프로그램 – sigaction (SIGINT 처리)

```
struct sigaction{
        void (*sa_handler)(int);      //취해질 행동
        sigset_t sa_mask;             //시그널을 처리하는 동안 추가의 시그널봉쇄
        int sa_flags;                 //시그널 형태에 영향을 미칠 플래그들
        void (*sa_sigaction) (int, siginfo_t*, void *);
                                      // 시그널 핸들러에 대한포인터
};
```

sa_handler
- signo번호를 가지는 시그널이 발생했을 때 실행될 함수를 설치한다.
- 함수외에도 SIG_DFL과 SIG_IGN을 지정할 수 있다.
- 전자는 시그널에 대한 기본행동을 후자는 시그널을 무시하기 위해서 사용

sa_mask
- sa_handler에 등록된 시그널 핸들러 함수가 실행되는 동안 블락되어야 하는 시그널의 마스크를 제공한다.

예제 프로그램 – sigaction (SIGINT 처리)

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

void catchint(int signum) {
        printf("\nCATCHINT: signum %d\n", signum);
        printf("CATCHINT: returning\n\n");
}
int main()
{
    static struct sigaction act;
    act.sa_handler = catchint;
    sigfillset(&(act.sa_mask));
    sigaction(SIGINT, &act, NULL);

    printf("sleep call #1\n");
    sleep(1);
```

예제 프로그램 – sigaction (SIGINT 처리)

```
    sigfillset(&(act.sa_mask));
    sigaction(SIGINT, &act, NULL);

    printf("sleep call #1\n");
    sleep(1);
    printf("sleep call #2\n");
    sleep(1);
    printf("sleep call #3\n");
    sleep(1);
    printf("sleep call #4\n");
    sleep(1);
    printf("sleep call #5\n");
    sleep(1);
    printf("Exiting\n");
    exit(0);
}
```

예제 프로그램 – sigaction (SIGINT 처리)

```
[111222333@node1 class4]$ ./sigaction
sleep call #1
sleep call #2
^C
CATCHINT: signum 2
CATCHINT: returning

sleep call #3
sleep call #4
sleep call #5
Exiting
```

예제 프로그램 – signal block

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

void sigint_handler(int signo)
{
    int i;
    for (i=3; 0<i; i--){
            printf("%d seconds left.\n", i);
            sleep(1);
    }
}
int main(void)
{
    struct sigaction act;
    act.sa_handler = sigint_handler;
    sigfillset(&act.sa_mask);
    sigaction(SIGINT, &act, NULL);
    while(1){
            printf("=== action ===\n");
            sleep(1);
    }
}
```

```
[1111222333@node1 class4]$ ./signalblock
=== action ===
=== action ===
=== action ===
^C3 seconds left.
2 seconds left.
1 seconds left.
=== action ===
=== action ===
=== action ===
=== action ===
=== action ===
^Z
[1]+  Stopped                 ./signalblock
```

시그널 - pause

사용법

```
#include <unistd.h>

int pause (void);
```

- 신호를 받을 때까지 호출 프로세스를 중지시킨다

시그널 - kill

사용법

```
#include <sys/types.h>
#include <signal.h>

int kill (pid_t pid, int sig);
```

- pid> 0 : pid를 가진 프로세스에게 시그널을 보낸다
- pid== 0 : 시그널은 보내는 프로세스와 같은 프로세스 그룹에 속하는 모든 프로세스에 보내짐
- pid== -1 & not superuser: 프로세스의 유효사용자 식별번호와 같은 모든 프로세스에 보내짐
- pid== -1 & superuser: 특수한 시스템 프로세스를 제외한 모든 프로세스에 보내짐
- pid< -1 : 프로세스의 그룹 식별번호가 pid의 절대값과 같은 모든프로세스에보내짐

시그널 - kill

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

int i = 0;
void p_handler(int signo)
{
    printf("Parent handler : call %d times. \n", i++);
}
void c_handler(int signo)
{
    printf("Child handler : call %d times. \n", i++);
}
int main(void)
{
    pid_t pid, ppid;
    struct sigaction act;
    pid = fork();
```

시그널 - kill

```
    pid = fork();
    if(pid == 0){
        act.sa_handler = c_handler;
        sigaction(SIGUSR1, &act, NULL);
        ppid = getppid();
        while(1){
            sleep(1);
            kill(ppid, SIGSUR1);
            pause();
        }
    }else if(pid > 0){
        act.sa_handler = p_handler;
        sigaction(SIGUSR1, &act, NULL);
        while(1){
            pause();
            sleep(1);
            kill(pid, SIGUSR1);
        }
    }else
        perror("Error\n")
}
```

```
[1111222333@node1 class4]$ ./kill
Parent handler : call 0 times.
Child handler : call 0 times.
Parent handler : call 1 times.
Child handler : call 1 times.
Parent handler : call 2 times.
Child handler : call 2 times.
Parent handler : call 3 times.
Child handler : call 3 times.
^C
```

시그널 – raise, alarm

사용법 - raise

```
#include <signal.h>
int raise (int sig);
```

현재프로세스에게 sig번호를 가지는 시그널을 전달한다

사용법 - alarm

```
#include <signal.h>
unsigned int alarm (unsigned int secs);
```

alarm은 secs초 후에 프로세스에 SIGALRM을 전달한다
alarm(0) 을 호출하면 alarm이 꺼진다
이전에 설정된 알람이 시그널을 전달할 때까지 남은시간을
초 단위 숫자로 반환하거나, 이전에 설정된 알람이 없을 경우
0을 되돌려준다

시그널 – raise, alarm

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

void myalarm()
{
        printf("ding dong dnag\n");
}
int main(void)
{
    int i = 0;
    printf("alarm setting\n");
    signal(SIGALRM, myalarm);
    alarm(1);
    while (i<5){
        printf("ok\n");
        pause();
        alarm(2);
        i++;
    }
}
```

```
[1111222333@node1 class4]$ ./alarm
alarm setting
ok
ding dong dnag
ok
ding dong dnag
ok
ding dong dnag
ok
ding dong dnag
ok
ding dong dnag
ok
ding dong dnag
```

실습 과제

- ✓ Alarm 예제에서 alarm함수를 kill, raise 등을 이용하여
  void newalarm(int secs)을 작성하시오!
  (기존 코드중 main 함수 부분은 alarm(n) -> newalarm(n) 부분을 제외하고 가급적 건드리지 않도록 함)

- ✓ Newalarm 작성 화면과 실행화면 캡쳐(총2개)해서 양식 맞춰 메일

실습 과제

```
void myalarm();                 // signal handler
void newalarm(int secs);        // secs초 후 SIGALRM발생

int main() { // alarm 예제에서
    alarm(n) → newalarm(n) 으로 교체
}
```

끝