Lab. 1 사용자 모드 vs. 시스템 모드

1) **시스템 콜: *strace***

: 시스템 콜 호출을 통해 사용자 모드와 시스템 모드의 전환 순서를 확인한다.

```
#include <stdio.h>

int main(void)
{
        puts("hello world");
        return 0;
}
```

그림 1 hello.c

다음과 같이 컴파일하고 *strace*를 통해 시스템 호출을 확인해 본다.

hello.log 파일을 분석해서 어떤 시스템 콜이 호출되었는지 확인한다. 'hello world'라는 문장이 출력되는 것이므로 가장 중요한 하나의 시스템 콜이 거의 마지막에 호출되었음을 확인할 수 있다. 다른 시스템 콜은 무시하고 넘어간다.

```
$ cc -o hello hello.c
$ ./hello        /* 실행 성공 확인 */
$ strace -o hello.log ./hello
$ cat hello.log
```

● 'cc' 컴파일 명령이 없다고 오류가 나면 'sudo apt install gcc'를 입력해서 gcc를 설치하고, 아래 'cc'를 모두 'gcc'로 변경한다.

2) **사용자 모드와 시스템 모드 비율 실험: *sar***

: 두 개 모드의 실행 비율을 측정한다.

```
$ sar -P ALL 1 1        /* 모든 프로세서, 1초 간, 1회 측정 */
```

측정 결과에서 CPU 코어 번호에 따라 %user, %system 등이 백분율로 표시된다.

```
#include <stdio.h>

int main(void)
{
        for (;;)
          ;
}
```

그림 2 loop.c

loop.c 무한루프 프로그램을 백그라운드 모드로 실행하고 sar을 다시 측정한다.

```
$ cc -o loop loop.c
$ ./loop &      /* 프로세스 번호 확인 */
$ sar -P ALL 1 1
$ kill <프로세스 번호>
```

loop.c 프로세스가 어느 코어에서 실행되었으며, 사용자 모드와 시스템 모드가 어떤 비율로 실행되고 있는지 분석한다. 무한루프 프로세스도 종료한다.

두 번째로 아래 ppidloop.c를 위와 동일하게 실험해보고 결과를 분석한다. loop.c와 ppidloop.c의 두 가지 실험이 다른 결과를 보여준 이유를 찾아보고 분석한다.

```
#include <sts/types.h>
#include <stdio.h>

int main(void)
{
        for (;;)
          getppid();      /* 부모 프로세스 id 구하기 */
}
```

그림 3 ppidloop.c

끝.