Operating Systems LAB 11

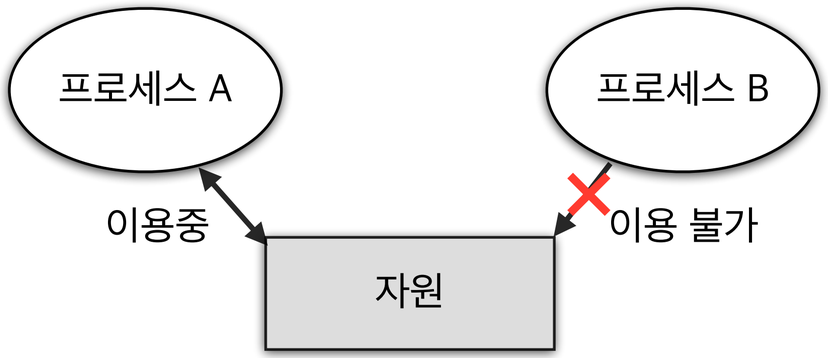
Wonpyo Kim
skykwp@gmail.com

Outline

- Substance
  - **Semaphore**

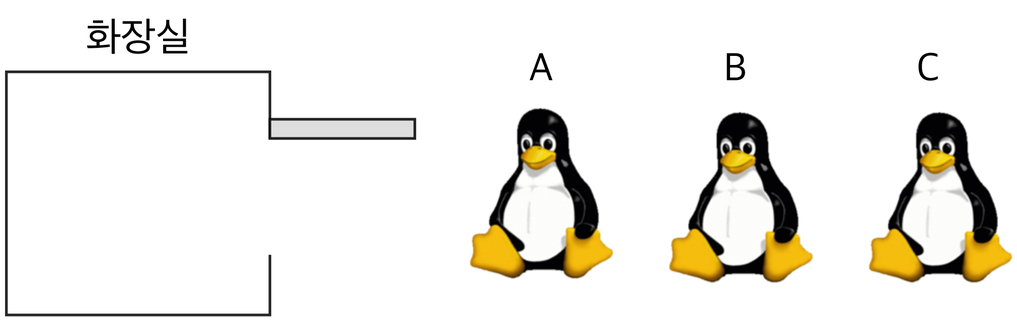
Semaphore

- 세마포어란 여러 프로세스들이 자원을 이용할 때, 한정된 수의 프로세스만 자원을 이용할 수 있게 하는 방법의 개념이다.



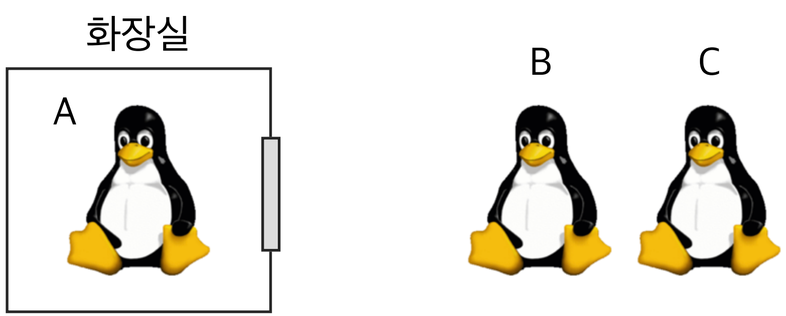
- 좀 더 자세히 나타내면 다음과 같은 예를 살펴본다.

  (1) 화장실의 칸이 단 한 개만 있다고 가정한다. 다행히도 문이 열려 있어서 펭귄 A가 들어간다.

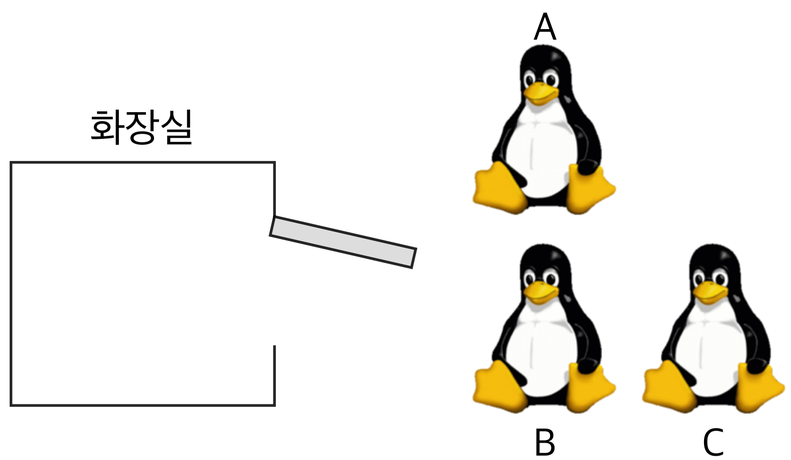


Semaphore

(2) 펭귄 A은 화장실에 들어가서 문을 잠근다. 펭귄 B와 펭귄 C는 화장실에 들어가고 싶어도 들어갈 수 없다.

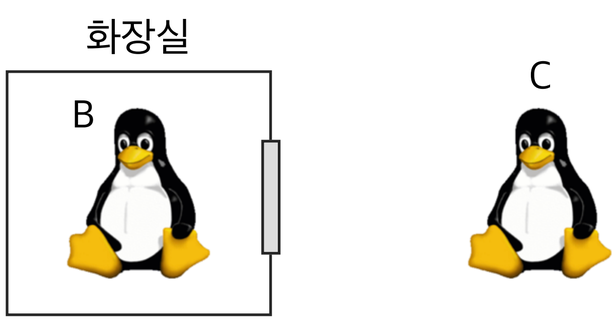


(3) 이후, 볼일을 마친 펭귄 A는 문을 열고 나온다.



Semaphore

(4) 화장실의 문이 열려 있으므로 펭귄 B가 들어가서 문을 잠근다. 펭귄 C은 문이 잠겨있기 때문에 들어갈 수 없다.



결국, 화장실이라는 자원 한 개를 하나의 펭귄. 즉, 한 개의 프로세스만 이용할 수 있다.

Semaphore

- 세마포어가 이러한 원리를 이용해 임의의 자원에 대해 한정된 프로세스만 이용할 수 있게 한다.

- 예를 들어, 네트워크로 연결된 5대의 PC와 프린터 한 대가 있을 때, 한 순간에 프린터를 이용할 수 있는 PC는 한 대로 엄격히는 한 개의 프로세스이다. 프린터가 한 대 이므로 프린터 관련 세마포어는 값이 1이 된다.

  1

  세마포어

- 프로세스 A가 프린터를 이용하려면 우선 세마포어 값이 0을 초과하는지를 확인한다. 0을 초과하면 프린터를 이용할 수 있고 초과하지 않으면 프린터를 이용할 수 없다. 세마포어 값이 1 이므로 프린터를 이용할 수 있다. 따라서, 우선 프로세스 A는 세마포어 값을 1로 감소시키고 프린터를 이용한다.

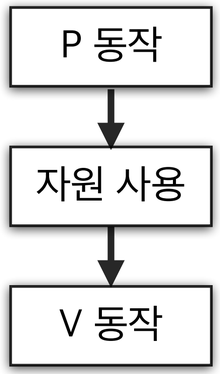
  0

  세마포어

Semaphore

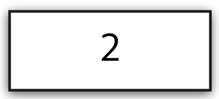
- 이 때, 다른 프로세스들이 프린터를 이용하려고 세마포어 값을 확인하지만 0 이므로 0을 초과할 때까지 기다린다. 프로세스 A가 프린터 사용을 마치면 세마포어 값을 다시 1 증가시킨다. 이후, 기다리던 다른 프로세스가 세마포어 값을 1로 감소시키고 프린터를 이용한다.

- 세마포어는 이와 같이 동작한다. 자원을 사용하기 위해 세마포어 값을 1 감소시키는 동작을 "P 동작" 이라 하고, 자원의 사용이 끝난 후 세마포어 값을 다시 1 증가시키는 동작을 "V 동작" 이라고 한다.



Semaphore

- 동일한 자원이 2개 있을 때는 세마포어 값이 2가 된다고 생각하면 된다.



세마포어

- 프로세스 A가 프린터를 사용하려 하는데, 세마포어 값이 0을 초과하므로 세마포어 값을 1 감소시키고 프린터 한 대를 이용한다.
  프로세스 B 역시 세마포어 값이 0을 초과하므로 세마포어 값을 1 감소시키고 프린터 한 대를 사용한다.

- 자원 값만 증가하였을 뿐 동작은 동일하다.

Semaphore

- 세마포어를 이용하기 위해서는 세마포어 집합을 생성하거나 기존에 생성된 세마포어 집합에 접근해야 한다. 이 때, semget( ) 함수를 사용한다.

```
semget 함수
기능
        세마포어 집합을 생성한다.
기본형
        int semget(key_t key, int nsems, int semflg);
        key: 시스템에서 식별하는 세마포어 집합 연결
        nsems: 세마포어 집합 내 세마포어 개수
        semflg: 동작 옵션
반환값
        성공: 세마포어 집합 식별자
        실패: -1
헤더파일
        <sys/types.h>
        <sys/ipc.h>
        <sys/sem.h>
```

Semaphore

- 첫 번째 인수인 key는 시스템 전체 영역에서 식별하는 숫자이며, IPC_PRIVATE 를 사용할 수도 있다. 예는 다음과 같다.

```
semid = semget((key_t)1234, ... );
semid = semget(IPC_PRIVATE, ... );
```

- 두 번째 인수인 nsems 는 세마포어의 개수이다. 예를 들어, 자원의 종류가 2개 (프린터, 스캐너)라면 세마포어가 2개 필요하므로 nsems 를 2 로 설정한다.

- 세 번째 인수인 semflg 에 의해 semget( ) 함수가 수행해야 할 동작이 정해지는데, 이와 관련된 값은 다음과 같다. 메시지 큐와 공유 메모리에서의 값과 같다.

semflg	의미
IPC_CREAT	key에 해당하는 세마포어 집합이 존재하지 않으면 생성하는데 접근 권한도 함께 부여해야 한다. 만약 세마포어 집합이 있으면 이 옵션은 무시한다.
IPC_EXCL	세마포어 집합이 있으면 실패로 -1을 반환한다. 이 값이 설정되지 않으면 기존 세마포어 집합에 접근하여 식별자를 반환한다.

- semget( ) 함수의 사용 예

```
semid = semget((key_t)1234, 1, IPC_CREAT|0666);
```

Semaphore

- 세마포어의 제어는 semctl( ) 함수가 수행하고, semid 세마포어 집합의 semnum 번 째 세마포어에 대해 cmd 명령을 수행한다.

semctl 함수
기능
세마포어를 제어한다.
기본형
int semctl(int semid, int semnum, int cmd, union semun arg);
semid: 세마포어 집합 식별자
semnum: 세마포어 집합 내에서의 세마포어 위치
cmd: 제어 종류
arg: cmd에 따라 사용 용도가 달라짐
반환값
성공: 0 이상의 수
실패: -1
헤더파일
<sys/types.h>
<sys/ipc.h>
<sys/sem.h>

Semaphore

- 첫 번째 인수인 semid 는 semget( ) 함수에 의해 반환된 세마포어 집합의 식별자이다.

```
semid = semget( ... );
semctl(semid, ... );
```

- 두 번째 인수인 semnum 은 제어 동작을 실행하기 원하는 세마포어의 위치로 첫 번째 세마포어는 0이 된다. 예를 들어, 두 종류의 자원을 관리하기 위해 세마포어 두 개를 갖는 세마포어 집합을 생성하고 두 번째 세마포어를 제어하기 원하면 다음과 같이 설정한다.

```
semid = semget((key_t)1234, 2, IPC_CREAT|0666);
semctl(semid, 1, ... );
```

- cmd 인수가 GETALL 또는 SETALL 이라면 이 값은 무시한다. 네 번째 인수인 arg 는 다음과 같은 구조의 union semun 데이터형이다. 특히, struct semid_ds 는 각 세마포어의 정보를 저장하는데 사용되는 데이터형이다.

```
/* arg for semctl system calls. */
union semun {
    int val;            /* value for SETVAL */
    struct semid_ds *buf;   /* buffer for IPC_STAT & IPC_SET */
    unsigned short *array;  /* array for GETALL & SETALL */
    struct seminfo *__buf;  /* buffer for IPC_INFO */
    void *__pad;
};
```

Semaphore

- 세 번째 인수인 cmd 는 어떤 동작을 할지 결정하는 인수로 다음과 같은 목록을 갖는다.

cmd	의미
GETVAL	세마포어의 현재 값을 얻는다.
GETPID	세마포어에 가장 최근에 접근했던 프로세스의 프로세스 ID를 얻는다.
GETNCNT	세마포어 값이 증가하기를 기다리는 프로세스의 수를 얻는다.
GETZCNT	세마포어 값이 0이 되기를 기다리는 프로세스의 수를 얻는다.
GETALL	세마포어 집합의 모든 세마포어 값을 얻는다.
SETVAL	세마포어의 값을 설정한다.
SETALL	세마포어 집합의 모든 세마포어 값을 설정한다.
IPC_STAT	세마포어의 정보를 얻는다.
IPC_SET	세마포어의 소유권과 접근 허가를 설정한다.
IPC_RMID	세마포어 집합을 삭제한다.

Semaphore

- 세마포어를 사용하기 위해 필수적으로 사용해야 하는 매크로는 SETVAL 로 세마포어 값을 설정한다. 이 때, semctl( ) 의 세 번째 인수인 arg 에서 val 멤버에 세마포어 값을 저장하고 호출한다.

- 세마포어 집합에서 세마포어 개수는 자원의 종류를 의미하고 세마포어 값은 각 자원의 수를 의미한다.

- 예를 들어, 프린터와 스캐너 2 종류의 자원이 있을 때는 아래와 같이 설정한다.

```
semid = semget((key_t)1234, 2, IPC_CREAT|0666);
```

- 이 때, 존재하는 프린터가 3개가 있을 때는 아래와 같이 프린터 관련 세마포어의 값을 3으로 설정한다.

```
union semun {
    int val;
    struct semid_ds *buf;
    unsigned short int *array;
} arg;
arg.val = 3;
semctl(semid, 0, SETVAL, arg);
```

Semaphore

- 이러한 세마포어 값은 각 세마포어를 관리하는 다음과 같은 struct semid_ds 데이터형으로 sem_nsems 에 저장된다.

```
/* Obsolete, used only for backwards compatibility and libc5 compiles */
struct semid_ds {
    struct ipc_perm sem_perm;       /* permissions .. see ipc.h */
    __kernel_time_t sem_otime;      /* last semop time */
    __kernel_time_t sem_ctime;      /* create/last semctl() time */
    struct sem  *sem_base;      /* ptr to first semaphore in array */
    struct sem_queue *sem_pending;      /* pending operations to be processed */
    struct sem_queue **sem_pending_last;    /* last pending operation */
    struct sem_undo *undo;          /* undo requests on this array */
    unsigned short  sem_nsems;      /* no. of semaphores in array */
};
```

- 또한, sem_base 가 가리키는(포인터) struct sem 데이터형의 semval 에도 저장된다. 그러나 semval 은 현재의 세마포어 값을 저장하므로 연산 중에 값이 변경된다.

```
struct sem {
    unsigned short semval;
    pid_t sempid;
};
```

Semaphore

- 자원을 사용하기 전에 세마포어 값을 1 감소시키고, 자원 사용이 끝나면 세마포어 값을 1 증가시키는데, 이러한 동작을 담당하는 함수가 semop( ) 함수로 다음과 같다.

```
semop 함수
기능
        세마포어 연산을 한다.
기본형
        int semop(int semid, struct sembuf *spos, unsigned nsops);
        semid: 세마포어 집합 식별자
        sops: 연산 내용
        nsops: 연산할 세마포어의 개수
반환값
        성공: 0
        실패: -1
헤더파일
        <sys/types.h>
        <sys/ipc.h>
        <sys/sem.h>
```

Semaphore

- 첫 번째 인수인 semid 는 semget( ) 함수에 의해 반환된 세마포어 집합의 식별자이다.

```
semid = semget( ... );
semctl(semid, 0, SETVAL, ... );
semop(semid, ... );
```

- 두 번째 인수인 sops가 포인터하는 struct sembuf 데이터 형은 세마포어에 대한 연산 내용을 저장한다.

```
/* semop system calls takes an array of these. */
struct sembuf {
    unsigned short  sem_num;    /* semaphore index in array */
    short       sem_op;     /* semaphore operation */
    short       sem_flg;    /* operation flags */
};
```

- 첫 번째 멤버인 sem_num 연산할 세마포어 번호가 설정되는데, 첫 번째 세마포어 번호는 0이 된다. 두 번째 멤버인 sem_op 에 의해 어떤 연산을 할지 결정한다. 세마포어 값을 감소시키려면 음수(일반적으로 -1)를, 세마포어 값을 증가시키려면 양수(일반적으로 1)를 설정한다.

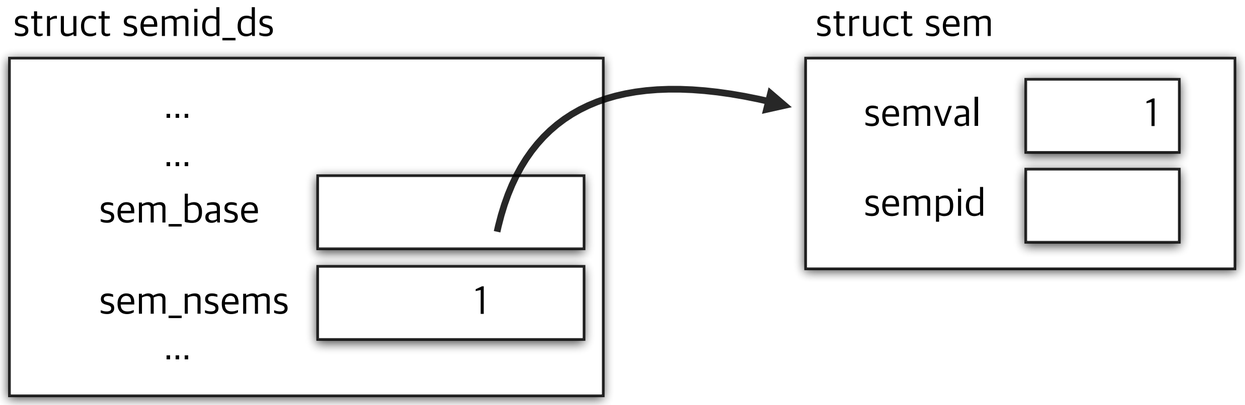
- sem_op 와 현재의 세마포어 값인 semval 을 더한 값을 semval 에 새롭게 설정한다.

semval = semval + sem_op

Semaphore

- 예는 다음과 같다.

  (1) 현재 세마포어 값(semval)이 1인 상태이고 자원을 이용하기 위한 프로세스 A와 B가 있다고 가정한다.

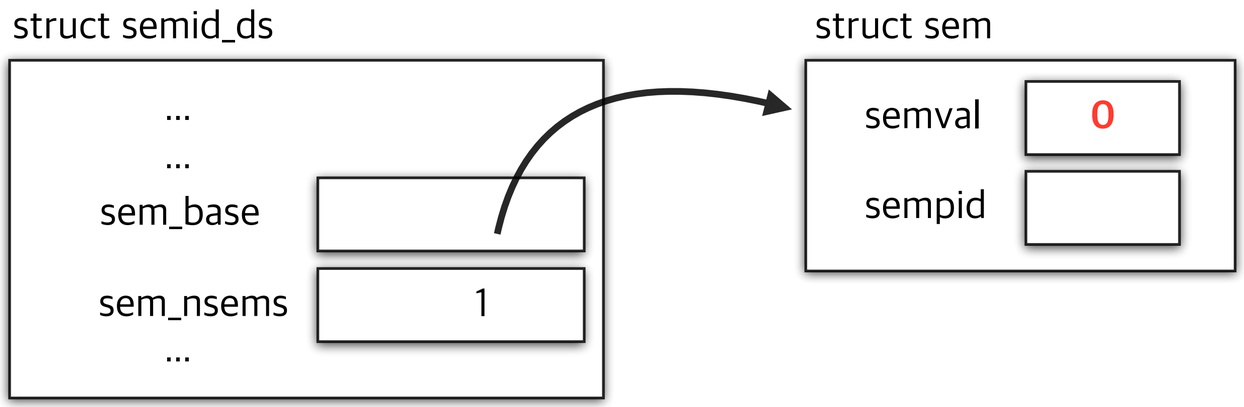


(2) 한 개의 자원을 이용하기 위해서는 세마포어 값을 1 감소시켜야 하므로 sem_op 를 -1로 하고 프로세스 A와 프로세스 B 순으로 semop( ) 함수를 호출한다.

```
struct sembuf buf;
buf.sem_num = 0;
buf.sem_op = -1;
buf.sem_flg = 0;
semop(semid, &buf, 1);
```

Semaphore

프로세스 A가 호출할 때 semval 값이 0을 초과하므로 호출에 성공하여 semval 과 buf.sem_op 를 더한 값인 0을 semval 에 저장하고 자원을 이용한다.

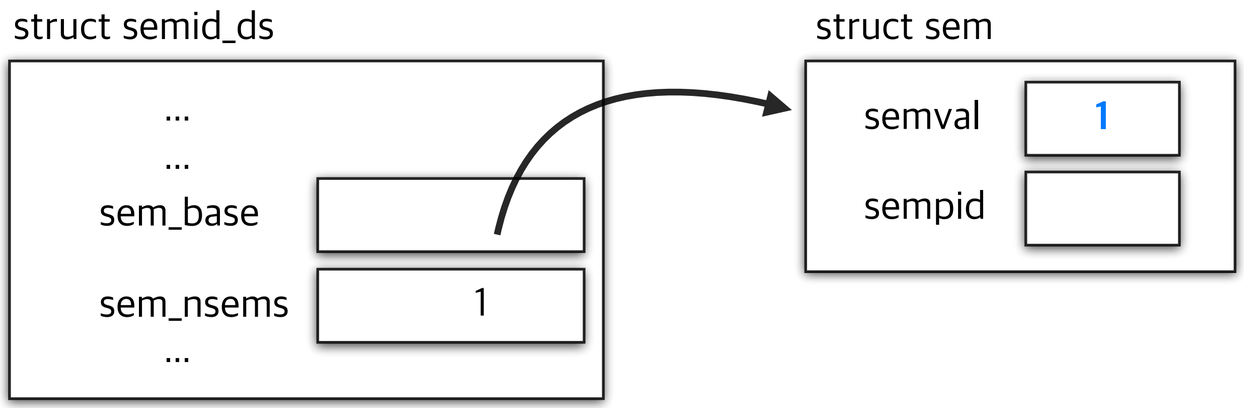


(3) 프로세스 B가 semop 를 호출하지만 semval 의 값이 0이므로 0을 초과할 때까지 기다린다.

(4) 프로세스 A가 자원 사용을 끝내고 semop 를 호출하여 semval 값을 1 증가시킨다.

```
struct sembuf buf;
buf.sem_num = 0;
buf.sem_op = 1;
buf.sem_flg = 0;
semop(semid, &buf, 1);
```

Semaphore



(5) semval 이 0을 초과하므로 기다리고 있던 프로세스 B의 semop( ) 호출이 성공하여 semval 을 1 감소시키고 자원을 이용한다.

Semaphore

- 세마포어 수가 2인 세마포어 집합에 세마포어 두 개에 대한 연산을 동시에 하기 위해서는 다음과 같이 설정한다.

```
struct sembuf buf[2] = {
    {0, -1, 0},
    {1, -1, 0}
};
semop(semid, buf, 2);
```

- 첫 번째의 {0, -1, 0}, 는 첫 번째 세마포어에 대해 -1 연산을 하라는 의미이며,
- 두 번째의 {1, -1, 0}, 는 두 번째 세마포어에 대해 -1 연산을 하라는 의미이다.

- struct sembuf 의 세 번째 멤버인 sem_flg 는 다음과 같은 값을 가질 수 있고, 0으로 설정할 수도 있다.

sem_flg	의미
IPC_NOWAIT	호출 즉시 실행하지 못할 경우 기다리지 않고 실패로 -1을 반환한다.
SEM_UNDO	프로세스가 종료되면 시스템에서 세마포어 설정을 원래 상태로 되돌린다. 설정하여 사용하는 것이 바람직하다.