세마포어[1]

세마포어

Critical Section 에 여러 프로세스가 들어가고자 한다면, 한 번에 하나의 프로세스가 들어갈 수 있도록 Mutual Exclusion 해주는 도구.

- 프로세스 사이의 동기를 맞추는 기능 제공
- 한 번에 한 프로세스만 작업을 수행하는 부분에 접근해 잠그거나, 다시 잠금을 해제하는 기능을 제공하는 정수형 변수
- 세마포어를 처음 제안한 에츠허르 데이크스트라가 사용한 용어에 따라 잠금함수는 p로 표시하고 해제함수는 v로 표시

그 밖에도 많은 것을 공헌함.

잠금을 나타내는 함수

해제를 나타내는 함수

세마포어 기본 동작 구조

- 중요 처리부분(critical section)에 들어가기 전에 p 함수를 실행하여 잠금 수행
- 처리를 마치면 v 함수를 실행하여 잠금 해제

```
p(sem);  // 잠금
중요한 처리 부분
v(sem);  // 잠금 해제
```

세마포어[2]

p 함수의 기본 동작 구조

```
p(sem) {
    while sem=0 do wait;  sem = 0 이면, 누군가 Critical Section 안에 들어가 있음을 의미. 그러므로, 기다림.
    sem 값을 1 감소;  sem = 1 이면, Critical Section 이 비어있음을 의미. 그러므로, sem = 0 으로 하고 작업을 수행.
}
```

- **sem의 초기값은 1**
- **sem이 0이면 다른 프로세스가 처리부분을 수행하고 있다는 의미이므로 1이 될 때까지 기다린다.**
- **sem이 0이 아니면 0으로 만들어 다른 프로세스가 들어오지 못하게 함**

v 함수의 기본 동작 구조

```
v(sem) {
    sem 값을 1 증가;  1로 다시 증가시켜서 Critical Section 이 비어있음을 나타냄.
    if (대기중인 프로세스가 있으면)
        대기중인 첫 번째 프로세스를 동작시킨다
}
```

세마포어[3]

세마포어 생성: semget(2)

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semget(key_t key, int nsems, int semflg);
```

여러 개의 세마포어를 동시에 생성 가능

- nsems : 생성할 세마포어 개수
- semflg : 세마포어 접근 속성 (IPC_CREAT, IPC_EXCL)

있으면 생성하지 않음.

있으면 에러
counting 세마포어가 있고, binary 세마포어가 있음.
counting 세마포어는 임의의 값을 가질 수 있고,
binary 세마포어는 1 또는 0 을 가질 수 있음.

semid_ds 구조체

```
struct semid_ds {
    struct ipc_perm sem_perm;
    struct sem *sem_base;
    ushort_t sem_nsems;
    time_t sem_otime;
    int32_t sem_pad1;
    time_t sem_ctime;
    int32_t sem_pad2;
    int sem_binary;
    long sem_pad3[3];
};
```

- sem_perm: IPC공통 구조체
- sem_base: 세마포어 집합에서 첫번째 세마포어의 주소
- sem_nsems: 세모포어 집합에서 세마포어 개수
- sem_otime: 세마포어 연산을 수행한 마지막시간
- sem_ctime: 세마포어 접근권한을 마지막으로 변경한 시간
- sem_binary: 세마포어 종류를 나타내는 플래그

세마포어[4]

sem 구조체

- 세마포어 정보를 저장하는 구조체

```
struct sem {
    ushort_t    semval;
    pid_t       sempid;
    ushort_t    semncnt;
    ushort_t    semzcnt;
    kcondvar_t  semncnt_cv;
    kcondvar_t  semzcnt_cv;
};
```

이 4가지는 꼭 알도록!

- **semval : 세마포어 값**
- **sempid : 세마포어 연산을 마지막으로 수행한 프로세스 PID**
- **semncnt: 세마포어 값이 현재 값보다 증가하기를 기다리는 프로세스 수** 증가하기를 기다리는 block 된 프로세스의 개수.
- **semzcnt: 세마포어 값이 0이 되기를 기다리는 프로세스 수** 0 를 기다리면서 block 된 프로세스의 개수.

세마포어[5]

세마포어 제어: semctl(2)

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semctl(int semid, int semnum, int cmd, ...);
```

semget 을 통해 얻은 semid

몇 번째 세마포어인지.

- semnum : 기능을 제어할 세마포어 번호
- cmd : 수행할 제어 명령
- ... : 제어 명령에 따라 필요시 사용할 세마포어 공용체 주소(선택사항)

```
union semun {
    int              val;
    struct semid_ds * buf;
    ushort_t        *array;
} arg;
```

세마포어의 value

세마포어에 대한 정보를 담는

여러 개의 세마포어의 초기값을 설정할 때 사용.

cmd에 지정할 수 있는 값

세마포어 삭제 IPC 일부 필드를 수정 세마포어에 대한 정보를 읽어올 때

- IPC_RMID, IPC_SET, IPC_STAT : 메시지 큐, 공유 메모리와 동일 기능
- GETVAL : 세마포어의 semval 값을 읽어온다.
- SETVAL : 세마포어의 semval 값을 arg.val로 설정한다. 세마포어의 value 로 설정해야 함.
- GETPID : 세마포어의 sempid 값을 읽어온다. p 함수나 v 함수를 마지막으로 수행한 프로세스의 ID
- GETNCNT, GETZCNT : 세마포어의 semncnt, semzcnt 값을 읽어온다.
- GETALL : 세마포어 집합에 있는 모든 세마포어의 semval 값을 arg.array에 저장
- SETALL : 세마포어 집합에 있는 모든 세마포어의 semval 값을 arg.array의 값으로 설정

세마포어[6]

세마포어 연산: semop(2)

p 함수와 v 함수를 정의할 때 사용.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semop(int semid, struct sembuf *sops, size_t nsops);
```

- **sops : sembuf 구조체 주소**
- **nsops : sops가 가리키는 구조체 크기**
- **struct sembuf의 sem_flg**
  - **IPC_NOWAIT, SEM_UNDO**

세마포어 연산을 의미

다른 프로세스가 이미 들어가 있다면,
block 하지 않고, 기다리지 않음. 즉시 리턴.

지금까지 수행한 연산은 UNDO 함.

```
struct sembuf {
    ushort_t    sem_num;
    short       sem_op;
    short       sem_flg;
};
```

몇 번째 세마포어인가

세마포어 연산

- **sembuf 구조체의 sem_op 항목에 지정**

p 나 v 함수를 암시.
음수 값인 경우, p 함수.
양수 값인 경우, v 함수.
0 인 경우,

```
if (sem_op < 0) {          /* 세마포어 잠금 */
    wait until semval >= | sem_op |;
    semval -= | sem_op |;
}
else if (sem_op > 0)       /* 세마포어 잠금 해제 */
    semval += sem_op;
else
    wait until semval is 0;
```

p 함수임.

semval >= | sem_ op | 이 될 때까지 기다림.

semval = 1, sem_op = -1 이라면,
1 >= | -1 | 이므로,
semval = 1 - | -1 | = 0

v 함수임.

semval = 0, sem_op = 1 이라면,
semval = 0 + 1 = 1

sem_op = 0 이면,

semval = 0 이 될 때까지 기다림.

세마포어[7]

1. sem_op가 음수 : 세마포어 잠금 기능 수행

semval 은 사용 가능 공간, sem_op 는 들어가면 차지하는 공간

Critical Section 을 잠금.

- semval 값이 sem_op의 절댓값과 같거나 크면 semval 값에서 sem_op의 절댓값을 뺀다.
- semval 값이 sem_op의 절대값보다 작고 sem_flg에 IPC_NOWAIT가 설정되어 있으면 semop 함수는 즉시 리턴

누군가 들어가있음.

- semval 값이 sem_op 값보다 작은데 sem_flg에 IPC_NOWAIT가 설정되어 있지 않으면 semop 함수는 semncnt 값을 증가시키고 다음 상황을 기다린다.

① semval 값이 sem_op의 절대값보다 같거나 커진다. 이 경우 semncnt 값은 감소하고 semval 값에서 sem_op의 절대값을 뺀다.

에러 발생 경우, ② 시스템에서 semid가 제거된다. 이 경우 errno가 EIDRM으로 설정되고 -1을 리턴한다.

③ semop 함수를 호출한 프로세스가 시그널을 받는다. 이 경우 semncnt 값은 감소하고 시그널 처리함수를 수행한다.

```
p 함수.
if (semval >= ABS(sem_op)) { semval >= | sem_op | 이면,
        set semval to semval – ABS(sem_op)    semval = semval - | sem_op |
}
else  아니면,
{
        if ((sem_flag & IPC_NOWAIT)) return -1 immediately   sem_flag 가 IPC_NOWAIT 으로 설정되어 있다면, 즉각 리턴.
        else { 설정되어 있지 않다면, semval >= | sem_op | 가 될 때까지 기다림.
                wait until semval reaches or exceeds ABS(sem_op)
                then subtract ABS(sem_op) as above   자신이 들어가고 나서는 semval = semval – | sem_op |
        }
}
```

세마포어[8]

1. sem_op가 양수면 이는 세마포어의 잠금을 해제하고 사용중이던 공유자원을 돌려준다. 이 경우 sem_op 값이 semval 값에 더해진다.

2. sem_op 값이 0일 경우
   - semval 값이 0이면 semop 함수는 즉시 리턴한다.
   - semval 값이 0이 아니고, sem_flg에 IPC_NOWAIT가 설정되어 있으면 semop 함수는 즉시 리턴한다.
   - semval 값이 0이 아니고, sem_flg에 IPC_NOWAIT가 설정되어 있지 않으면 semop 함수는 semzcnt 값을 증가시키고 semval 값이 0이 되길 기다린다.

[예제 10-7] (1) 세마포어 생성과 초기화 (test1.c)

```
...  우선 union 데이터 구조체를 생성.
09   union semun {
10       int val;
11        struct semid_ds *buf;
12       unsigned short *array;
13   };
14
15   int initsem(key_t semkey) {
16       union semun semunarg;
17       int status = 0, semid;
18                                  1개 생성.
19       semid = semget(semkey, 1, IPC_CREAT | IPC_EXCL | 0600);
20       if (semid == -1) {
21           if (errno == EEXIST)
22               semid = semget(semkey, 1, 0);
23       }
24       else {
25           semunarg.val = 1;
26           status = semctl(semid, 0, SETVAL, semunarg);
27       }          첫 번째 세마포어라는 의미로, 0
28
29       if (semid == -1 || status == -1) {
30           perror("initsem");
31           return (-1);
32       }
33
34       return semid;
35   }
```

semun 공용체 선언

세마포어 생성 및 초기화 함수

세마포어 생성

어떤 프로세스가 먼저 그 프로그램을 시작했을 때, 다른 프로세스가 먼저 다 수행할 때까지 기다려야 된다면, 블록이 되어야 하는 경우, semval = 0 로 초기화 할 수 도 있음.

세마포어 값을 1로 초기화

union 구조체

SETVAL 을 통해, 초기화.

[예제 10-7] (2) 세마포어 연산

```
36  p 함수
37  int semlock(int semid) {
38      struct sembuf buf;
39
40      buf.sem_num = 0;  0 부터 시작하므로
41      buf.sem_op = -1;
42      buf.sem_flg = SEM_UNDO;
43      if (semop(semid, &buf, 1) == -1) {
44          perror("semlock failed");
45          exit(1);
46      }
47      return 0;
48  }
49  v 함수
50  int semunlock(int semid) {
51      struct sembuf buf;
52
53      buf.sem_num = 0;
54      buf.sem_op = 1;
55      buf.sem_flg = SEM_UNDO;
56      if (semop(semid, &buf, 1) == -1) {
57          perror("semunlock failed");
58          exit(1);
59      }
60      return 0;
61  }
```

세마포어 잠금 함수

sem_op 값을 음수로 하여 잠금기능 수행

세마포어 잠금 해제 함수

sem_op 값을 양수로 하여 잠금해제기능 수행

[예제 10-7] (3) 세마포어 호출

```
63  void semhandle() {
64      int semid;
65      pid_t pid = getpid();
66
67      if ((semid = initsem(1)) < 0)
68          exit(1);
69
70      semlock(semid);
71      printf("Lock : Process %d\n", (int)pid);
72      printf("** Lock Mode : Critical Section\n");
73      sleep(1);
74      printf("Unlock : Process %d\n", (int)pid);
75      semunlock(semid);
76
77      exit(0);
78  }
79
80  int main(void) {
81      int a;
82      for (a = 0; a < 3; a++)
83          if (fork() == 0) semhandle();
84
85      return 0;
86  }
```

세마포어 생성 함수 호출

세마포어 잠금함수 호출

처리부분

세마포어 잠금 해제 함수 호출

자식 프로세스를 3개 만든다.

3개의 자식 프로세스가 semhandle() 함수를 수행.

[예제 10-7] 실행결과

□ 세마포어 기능을 사용하지 않을 경우

```
# ex10_7.out
Lock : Process 5262
** Lock Mode : Critical Section
Lock : Process 5263
** Lock Mode : Critical Section
Lock : Process 5264
** Lock Mode : Critical Section
Unlock : Process 5263
Unlock : Process 5262
Unlock : Process 5264
```

5262 프로세스가 처리부분을 실행하는 중에 다른 프로세스도 같이 수행된다.

□ 세마포어 기능을 사용할 경우

```
# ex10_7.out
Lock : Process 5195
** Lock Mode : Critical Section
Unlock : Process 5195
Lock : Process 5196
** Lock Mode : Critical Section
Unlock : Process 5196
Lock : Process 5197
** Lock Mode : Critical Section
Unlock : Process 5197
```

5262 프로세스가 처리부분을 실행하는 중에 다른 프로세스는 실행하지 않고 차례로 실행한다.