Part 02. 리눅스 시스템 프로그래밍

Chapter 14. IPC(3)

진행 순서

Chapter 14_01	세마포어 개요
Chapter 14_02	SysV 세마포어
Chapter 14_03	POSIX 세마포어

Chapter 14_01 세마포어 개요

개념
세마포어(Semaphore)는 다익스트라(Dijkstra)가 고안한, 두 개의 원자적 함수로 조작되는 정수 변수로서,
멀티프로그래밍 환경에서 공유 자원에 대한 접근을 제한하는 방법(상호배제)으로 사용된다. – Wikipedia

원리
세마포어 S는 정수값을 가지는 변수이며, 다음과 같이 P와 V라는 명령에 의해서만 접근할 수 있습니다.
(P와 V는 각각 try와 increment를 뜻하는 네델란드어의 머릿글자를 딴 것입니다.)
P는 임계 영역에 들어가기 전에 수행되고, V는 임계 영역에서 나올 때 수행됩니다.
이때 변수 값을 수정하는 연산은 모두 원자성을 만족해야 합니다.
즉, 한 프로세스에서 세마포어 값을 변경하는 동안 다른 프로세스가 동시에 이 값을 변경해서는 안됩니다.

종류
계수 세마포어(Counting Semaphore): 다수개의 자원에 대해 카운팅이 가능한 세마포어 (0 ~ n)
이진 세마포어(Binary Semaphore): 1개의 자원에 대해 카운팅, 즉 세마포어 값은 0또는 1을 가집니다.

설명
계수 세마포어의 경우 3개의 자원이 있다고 가정할 때(S = 3) 여러 프로세스 이 자원에 접근을 시도합니다.
자원에 접근할 때마다 세마포어를 하나씩 감소시키고(P), 세마포어 값이 0이 되면 자원에 접근을 시도하는
다른 프로세스는 대기하게 됩니다.
자원에 접근했던 프로세스가 접근을 해제하면 세마포어를 다시 하나씩 증가시키고(V),
대기하던 프로세스는 다시 자원에 접근이 가능하게 됩니다.
이진 세마포어는 계수 세마포어의 자원 값이 1개인 경우(S = 1)를 의미합니다. (뮤텍스와 유사)

Chapter 14_02 SysV 세마포어

SysV 세마포어의 주요 함수

semget - 세마포어의 IPC ID를 생성하거나 가져옴
semctl - 세마포어 조작 (초기화, 정보획득, 제거)
semop - 세마포어 증가/감소
semtimedop - 타임아웃 기능이 추가된 semop

흐름

1) semget()을 통해 ID 획득
2) 새로 생성된 경우 semctl()을 통해 세마포어 초기화
3) semop()를 이용해 세마포어 감소(P)
4) 처리
5) semop()를 이용해 세마포어 증가(V)
6) semctl()을 통해 세마포어 정리

Chapter 14_02 SysV 세마포어

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semget(key_t key, int nsems, int semflg);
```

semget() 시스템 호출은 인수 키와 연관된 System V 세마포어 세트 ID를 리턴합니다.
키에 IPC_PRIVATE 값이 있거나 키와 연관된 기존 세마포어 세트가 없고 semflg에 IPC_CREAT가 지정된 경우 새로운 nsems 개의 세마포어 세트가 작성됩니다. (shmget과 동일)

semflg에 IPC_CREAT 와 IPC_EXCL을 모두 지정하고 키와 연관된 세마포어 세트가 이미 존재하는 경우 결과는 실패하고 errno가 EEXIST로 설정됩니다. (open (2)의 O_CREAT | O_EXCL 조합의 효과와 유사합니다.)

semflg 인수의 최하위 9 비트는 세마포어 세트에 대한 권한 (소유자, 그룹, 기타에 대한 권한)을 정의합니다. 이러한 비트는 open (2)의 모드 인수와 동일한 형식 및 의미를 갖습니다 (실행 권한은 의미가 없음, 쓰기 권한은 세마포어 값을 변경할 수 있는 권한을 의미).

성공 시 양수 세마포어 세트 식별자 리턴, 실패 시 -1 리턴 후 errno 설정

ex) 키 값으로 1234를 사용하고 접근권한 666으로 세마포어를 새롭게 생성하는 경우

```
int sem_id = semget((key_t)1234, 1, 0666 | IPC_CREATE);
```

Chapter 14_02 SysV 세마포어

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semctl(int semid, int semnum, int cmd, ...);
```

semctl()은 semid로 식별 된 System V 세마포어 세트의 semnum번째 세마포어에서 cmd로 지정된 제어 조작을 수행합니다. (세트의 세마포어는 0부터 시작하여 번호가 매겨집니다.)
cmd의 종류에 따라 뒤에 semun 공용체 인자가 추가될 수 있습니다.

```
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 (Linux-specific) */
};
```

실패 시 -1 리턴 후 errno 설정
성공 시 cmd에 따라 리턴값 달라짐

Chapter 14_02 SysV 세마포어

cmd 값

IPC_STAT(*)	IPC 자원 정보 획득 (생성자, 생성시간, 접근권한 등)
IPC_SET(*)	IPC 자원 정보 설정
IPC_RMID	IPC 자원 제거
IPC_INFO(*)	IPC 자원 시스템 설정값 획득 (리눅스 전용)
SEM_INFO (*)	IPC_INFO와 유사, 시스템 전체 세마포어 수 획득 가능 (리눅스 전용)
SEM_STAT(*)	IPC_STAT과 유사, 시스템 전체 세마포어 배열 인덱스 획득 (리눅스 전용)
GETALL(*)	배열을 이용해 모든 세마포어를 한꺼번에 읽어들임
GETNCNT	semnum 위치의 세마포어의 semncnt 값을 리턴
GETPID	semnum 위치의 세마포어 sempid 값을 리턴
GETVAL	semnum 위치의 세마포어 값을 리턴
GETZCNT	semnum 위치의 세마포어 semzcnt 값을 리턴
SETALL(*)	배열을 이용해 모든 세마포어를 한꺼번에 초기화
SETVAL(*)	semnum 위치의 세마포어 값을 초기화

(*) cmd의 경우 4번째 인자로 semun 공용체 사용

Chapter 14_02 SysV 세마포어

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semop(int semid, struct sembuf *sops, size_t nsops);
int semtimedop(int semid, struct sembuf *sops, size_t nsops, const struct timespec *timeout);
```

semop ()은 semid로 표시된 세트에서 선택된 세마포어에 대한 작업을 수행합니다.
sops가 가리키는 배열의 각 nsops 요소는 단일 세마포어에서 수행 할 작업을 지정하는 구조입니다.

```
struct sembuf {
        unsigned short sem_num;     /* semaphore number */
        short sem_op;               /* semaphore operation */
        short sem_flg;              /* operation flags */
}
```

작업은 sops 배열 순서로 진행되며, 각 작업은 세마포어 세트의 sem_num 번째 세마포어에서 수행
sem_op 값이 양수이면 세마포어 값에 추가(V)하고, 음수이면 세마포어 값에서 뺀다(P).

sem_flg 플래그
IPC_NOWAIT 사용 가능한 자원이 없을 때 기다리지 않고 에러 리턴 (EAGAIN)
SEM_UNDO 프로세스가 종료 될 때 모두 취소
프로세스가 잠금을 한 채로 종료될 경우 문제(무한대기)가 발생할 수 있는데 SEM_UNDO가 이를 해결

Chapter 14_02 SysV 세마포어

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <pthread.h>

union semun {
        int val;
        struct semid_ds *buf;
        unsigned short  *array;
};

struct sembuf p = { 0, -1, SEM_UNDO };
struct sembuf v = { 0, +1, SEM_UNDO };
```

```
#define SEM_P(ID) ({ \
        if (semop(ID, &p, 1) < 0) { \
                perror("semop p"); \
                exit(1); \
        } \
})

#define SEM_V(ID) ({ \
        if (semop(ID, &v, 1) < 0) { \
                perror("semop v"); \
                exit(1); \
        } \
})

int sem_use;
int semid;
char shared_buf[] = "Hello, World!\n";
int shared_index = 0;
```

Chapter 14_02 SysV 세마포어

```
void *print_one(void *arg)
{
   int tidx = *(int *)arg;
   bool b = true;
   char c;

   while (b) {
      if (sem_use) SEM_P(semid);
      if (shared_index >= strlen(shared_buf)) {
         b = false;
      } else {
         if (tidx)
            c = toupper(shared_buf[shared_index++]);
         else
            c = tolower(shared_buf[shared_index++]);
         sleep(rand()%2);
         putchar(c);
         fflush(stdout);
      }
      if (sem_use) SEM_V(semid);
   }

   pthread_exit((void *)0);
}
```

```
int main(int argc, char *argv[])
{
   key_t key;
   union semun u;
   int pid;
   pthread_t thread_t;
   int i, ret;
   int tidx[2] = {0, 1};

   if (argc < 2) {
      printf("Usage: %s [0(OFF)/1(ON)]\n", argv[0]);
      exit(1);
   }

   sem_use = atoi(argv[1]);

   key = ftok("shmtest", 1234);

   semid = semget(key, 1, 0666 | IPC_CREAT);
   if(semid < 0) {
      perror("semget");
      exit(1);
   }
```

Chapter 14_02 SysV 세마포어

```
    u.val = 1;
    if(semctl(semid, 0, SETVAL, u) < 0) {
        perror("semctl");
        exit(1);
    }

    for (i = 0; i <= 1; i++) {
        ret = pthread_create(&thread_t, NULL, print_one, (void *)&tidx[i]);
        if (ret) {
            perror("pthread_create");
            exit(1);
        }
    }

    pthread_join(thread_t, NULL);
    return 0;
}
```

```
[root@localhost ch14]# gcc -g sem_example.c -o sem_example -lpthread

[root@localhost ch14]# ./sem_example
Usage: ./sem_example [0(OFF)/1(ON)]

[root@localhost ch14]# ./sem_example 0
eHlLo w,olR!d

[root@localhost ch14]# ./sem_example 1
hElLo, WoRlD!
```

Chapter 14_03 POSIX 세마포어

POSIX 세마포어를 사용하면 프로세스와 스레드가 작업을 동기화 할 수 있습니다.

POSIX 세마포어의 연산
sem_wait (3) : 세마포어 값을 1 감소 시킵니다. (P), 세마포어의 값이 0이면 0보다 커질 때까지 차단.
sem_post (3) : 세마포어 값을 1 증가 시킵니다. (V)

POSIX 세마포어는 두 가지 형태가 있습니다 : 명명된 세마포어와 익명 세마포어.

명명된 세마포어 (named semaphore)는 /somename 형식의 이름으로 식별됩니다.
즉, 초기 슬래시로 구성되는 최대 NAME_MAX-4 (즉, 251) 개의 null로 끝나는 문자열.
동일한 이름을 sem_open (3)에 전달하여 두 개의 프로세스가 동일한 이름의 세마포어에서 작동 할 수 있습니다.

sem_open (3) 함수는 새로운 명명 된 세마포어를 생성하거나 기존의 명명 된 세마포어를 엽니다.
세마포어를 연 후에는 sem_post (3) 및 sem_wait (3)을 사용하여 작동 할 수 있습니다.
프로세스가 세마포어 사용을 마치면 sem_close (3)를 사용하여 세마포어를 닫을 수 있습니다.
모든 프로세스가 세마포어 사용을 마치면 sem_unlink (3)를 사용하여 시스템에서 제거 할 수 있습니다.

Chapter 14_03 POSIX 세마포어

익명 세마포어 (nameless semaphore, 메모리 기반 세마포어)는 이름이 없습니다.
대신 세마포어는 여러 스레드 (스레드 공유 세마포어) 또는 프로세스 (프로세스 공유 세마포어)간에 공유되는 메모리 영역에 배치됩니다.
스레드 공유 세마포어는 프로세스의 스레드 (예 : 전역 변수)간에 공유되는 메모리 영역에 배치됩니다.
프로세스 공유 세마포어는 공유 메모리 영역 (예 : shmget (2)을 사용하여 작성된 System V 공유 메모리 세그먼트 또는 shm_open (3)을 사용하여 작성된 POSIX 공유 메모리 오브젝트)에 배치해야 합니다.
생성한 프로세스가 종료되면 세마포어가 없어지는 임시 세마포어.

사용하기 전에 익명 세마포어는 sem_init (3)을 사용하여 초기화해야 합니다.
그런 다음 sem_post (3) 및 sem_wait (3)을 사용하여 작동 할 수 있습니다.
세마포어가 더 이상 필요하지 않은 경우, 세마포어가 있는 메모리를 할당 해제하기 전에 sem_destroy (3)를 사용하여 세마포어를 제거해야 합니다.

주요 함수

sem_open	명명된 세마포어를 생성하거나 오픈
sem_close	명명된 세마포어 닫기
sem_unlink	명명된 세마포어 제거
sem_init	익명 세마포어 생성 및 초기화
sem_destroy	익명 세마포어 제거
sem_wait	P 연산
sem_post	V 연산

* 주의 사항: 컴파일 시 Link with -pthread.

Chapter 14_03 POSIX 세마포어

```
#include <fcntl.h>          /* For O_* constants */
#include <sys/stat.h>       /* For mode constants */
#include <semaphore.h>

sem_t *sem_open(const char *name, int oflag);
sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value);

int sem_close(sem_t *sem);

int sem_unlink(const char *name);

ex)
sem_t *sem;
sem = sem_open("/test_sem", O_CREAT|O_EXCL, 0600, 1);
if (sem == SEM_FAILED) {
      if (errno != EEXIST) {
         perror("sem_open");
         exit(1);
      }
      sem = sem_open("/test_sem", 0);
}
...
sem_close(sem);
sem_unlink("/test_sem");
```

Chapter 14_03 POSIX 세마포어

```
#include <semaphore.h>

int sem_init(sem_t *sem, int pshared, unsigned int value);

int sem_destroy(sem_t *sem);
```

익명 세마포어를 생성하는 sem_init()은 아래 인자를 받는다.
sem - 초기화 후 리턴받을 세마포어 객체
pshared - 여러 프로세스에서 공유할 것인지 결정하는 플래그 (0일 경우 현재 프로세스만, 0이 아닐 경우 공유)
value - 세마포어 초기값

익명 세마포어를 pshared를 통해 공유하려는 경우 sem 인수가 프로세스의 로컬이라면 공유는 안되므로
익명 세마포어를 공유하려면 sem 인수로 넘어오는 객체가 공유 메모리 공간에 위치해야 한다.

ex)

```
sem_t sem;

if (sem_init(&sem, 0, 1) == -1) {
   perror("sem_init");
   exit(1);
}
...
sem_destroy(sem);
```

Chapter 14_03 POSIX 세마포어

```
#include <semaphore.h>

int sem_wait(sem_t *sem);
int sem_trywait(sem_t *sem);
int sem_timedwait(sem_t *sem, const struct timespec *abs_timeout);

int sem_post(sem_t *sem);
```

sem_wait	P 연산
sem_trywait	sem_wait에 넌블락킹 기능 추가 (EAGAIN)
sem_timedwait	sem_wait에 타임아웃 기능 추가 (ETIMEOUT)

sem_trywait 이나 sem_timedwait 을 이용하면 sem_wait 이 무한대기에 빠질 가능성을 해소할 수 있다.

ex)

```
struct timespec ts;
ts.tv_sec = time(NULL) + 5;
ts.tv_nsec = 0;
if (sem_timedwait(sem, &ts) == -1) {
   if (errno == ETIMEOUT) {
      /* 5초 타임 아웃 */
   } else {
      /* 에러 처리 */
   }
}
```

Chapter 14_03 POSIX 세마포어

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <errno.h>

sem_t sem;

void* thread_t1(void* arg)
{
        int tidx = *(int *)arg;
        sem_wait(&sem);
        printf("[%d] Start...\n", tidx);
        /* critical section */
        sleep(3);
        printf("[%d] End!\n\n", tidx);
        sem_post(&sem);
}
```

```
void* thread_t2(void* arg)
{
        int tidx = *(int *)arg;
        int ret;

try:
        ret = sem_trywait(&sem);
        if (ret == -1) {
                if (errno == EAGAIN) {
                        printf("[%d] retry get semaphoe...\n", tidx);
                        sleep(1);
                        goto try;
                }
                perror("sem_trywait");
                exit(1);
        } else if (ret == 0) {
                printf("[%d] Start...\n", tidx);
                /* critical section */
                sleep(3);
                printf("[%d] End!\n\n", tidx);
                sem_post(&sem);
        }
}
```

Chapter 14_03 POSIX 세마포어

```
int main()
{
        int ret;
        pthread_t t1, t2, t3;
        int tidx[] = {0, 1, 2};

        if (sem_init(&sem, 0, 1) == -1) {
                perror("sem_init");
                exit(1);
        }

        ret = pthread_create(&t1, NULL, thread_t1, (void *)&tidx[0]);
        if (ret != 0) {
                fprintf(stderr, "pthread_create error: %m");
                exit(1);
        }

        ret = pthread_create(&t2, NULL, thread_t1, (void *)&tidx[1]);
        if (ret != 0) {
                fprintf(stderr, "pthread_create error: %m");
                exit(1);
        }
```

Chapter 14_03 POSIX 세마포어

```
        ret = pthread_create(&t3, NULL, thread_t2, (void *)&tidx[2]);
        if (ret != 0) {
                fprintf(stderr, "pthread_create error: %m");
                exit(1);
        }

        pthread_join(t1, NULL);
        pthread_join(t2, NULL);
        pthread_join(t3, NULL);

        sem_destroy(&sem);
        return 0;
}
```

```
[root@localhost ch14]# gcc -g sem_posix_example.c -o sem_posix_example -pthread
[root@localhost ch14]# ./sem_posix_example
[0] Start...
[2] retry get semaphoe...
[2] retry get semaphoe...
[2] retry get semaphoe...
[0] End!

[1] Start...
[2] retry get semaphoe...
[2] retry get semaphoe...
[2] retry get semaphoe...
[1] End!

[2] Start...
[2] End!
```