고급 프로세스간 통신 #2

세마포어

- 네델란드의 E.W.Dijikstra가 프로세스 동기화 문제의 해결 방안으로 제시
  - *p*( ) 또는 *wait*( )

    if (sem != 0)

        decrement sem by 1

    else

        wait until sem become non-zero, then decrement
  - *v*( ) 또는 *signal*( )

    increment sem by one

    if (queue of waiting processes not empty)

        restart first process in wait queue
  - 사용 방법

    p(sem);

    critical section

    v(sem);

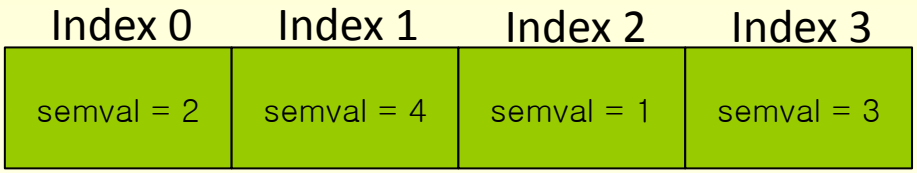
semget

- 사용법

  #include <sys/sem.h>

  int semget (key_t key, int nsems, int permflags);

  - nsems : 세마포어 집합에 필요한 세마포어 개수
    - 동시에 여러 개의 세마포어를 얻을 수도 있음
    - 세마포어의 인덱스는 0 ~ nsems – 1



  - 각 세마포어에 연관된 값
    - semval : 세마포어 값(항상 양의 정수로 지정)
    - sempid : 세마포어에 최근 접근한 프로세스 번호
    - semncnt : 세마포어 값이 현재 값보다 큰 값을 갖기를 기다리는 프로세스 수
    - semzcnt : 세마포어 값이 0이 되기를 기다리는 프로세스의 수

semctl

- 사용법

  #include <sys/sem.h>

  int semctl (int semid, int sem_num, int command, union semun ctl_arg);

  - semid : semget()에서 반환된 식별자
  - sem_num : 세마포어 집합에서 특정 세마포어 식별
  - ctl_arg : 다음과 같이 정의될 수 있는 하나의 union

```
Union semun {
        Int val;
        Struct semid_ds *buf;
        Unsigned short *array;
};
```

  - command
    - 표준 IPC 기능
      - IPC_STAT
      - IPC_SET
      - IPC_RMID
    - 단일 세마포어 연산
      - GETVAL
      - SETVAL
      - GETPID
      - GETNCNT
      - GETZCNT
    - 전체 세마포어 연산
      - GETALL : 모든 값을 ctl_arg.array에 저장
      - SETALL : ctl_arg.array에 저장된 값으로 모든 세마포어 설정

semop

- 사용법

  #include <sys/sem.h>

  int semop (int semid, struct sembuf *op_array, size_t num_ops);

  - semid : semget()에서 반환된 식별자
  - op_array : sembuf 구조의 배열
  - num_ops : 배열 내의 구조 수
  - sembuf 구조체의 주요 구성원

    unsigned short sem_num;

    short sem_op;

    short sem_flg;

    - sem_num : 세마포어 인덱스
    - sem_op : 수행할 기능
      - sem_op가 음수일 때
        - P( )의 일반화된 형태

          세마포어 값이 sem_op 절대값보다 크거나 같으면 절대값만큼 감소,

          그렇지 않으면

          IPC_NOWAIT인 상태이면 바로 -1 반환,

          그렇지 않으면 세마포어 값이 sem_op 절대값에 도달하거나 초과할 때까지 기다렸다가 위의 행동 수행
      - sem_op가 양수일 때
        - V( )의 일반화된 형태
        - sem_op 값을 세마포어 값에 더해준다
      - sem_op가 0일 때
        - 세마포어 값이 0이 될 때까지 기다린다
        - IPC_NOWAIT가 설정되어 있으면 즉시 -1을 돌려주고 반환
    - SEM_UNDO 플래그
      - 프로세스가 퇴장할 때 수행된 연산을 자동으로 취소
      - 내부적으로 semadj라는 변수을 유지하여 세마포어에 가했던 모든 연산을 취소하는효과를 갖도록 함

pv.h

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

#include <errno.h>

#define SEMPERM       0600
#define TRUE          1
#define FALSE         0

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

int initsem(key_t);
int p(int);
int v(int);
void handlesem(key_t);
```

initsem.c

```
#include "pv.h"

int initsem(key_t semkey)
{
        int status = 0;
        int semid;

        if ((semid = semget(semkey, 1, SEMPERM | IPC_CREAT | IPC_EXCL)) == -1) {
                if (errno == EEXIST)
                        semid = semget(semkey, 1, 0);
        //      fprintf(stderr, "semid = %d₩n", semid);
        }
        else {
                semun arg;

                arg.val = 1;
                status = semctl(semid, 0, SETVAL, arg);
        //      fprintf(stderr, "status = %d₩n", status);
        }

        if ((semid == -1) || (status == -1)) {
                perror ("initsem failed");
                return (-1);
        }

        return(semid);
}
```

p.c

```
#include "pv.h"

int p(int semid)
{
        struct sembuf    p_buf;

        p_buf.sem_num = 0;
        p_buf.sem_op = -1;
        p_buf.sem_flg = SEM_UNDO;

        if (semop(semid, &p_buf, 1) == -1) {
                perror("p(semid) failed");
                exit(1);
        }

        return(0);
}
```

V.C

```
#include "pv.h"

int v(int semid)
{
        struct sembuf v_buf;

        v_buf.sem_num = 0;
        v_buf.sem_op = 1;
        v_buf.sem_flg = SEM_UNDO;

        if (semop(semid, &v_buf, 1) == -1) {
                perror("v(semid) failed");
                exit(1);
        }

        return(0);
}
```

handlesem.c

```
#include "pv.h"

void handlesem(key_t skey)
{
        int semid;
        pid_t pid = getpid();

        if((semid = initsem(skey)) < 0) {
                perror("initsem(skey)");
                exit(1);
        }

        printf("₩nprocess %d before critical section₩n", pid);

        p(semid);

        printf("₩nprocess %d in critical section₩n", pid);

        sleep(10);

        printf("₩nprocess %d leaving critical section₩n", pid);

        v(semid);

        printf("₩nprocess %d exiting₩n", pid);

        exit(0);
}
```

testsem.c

```
#include "pv.h"

void main()
{
        key_t semkey = 0x200;

        int i;

        for (i = 0; i < 3; i++) {
                if (fork() == 0) {
                        handlesem(semkey);
                }
        }
}
```

실행 결과

```
cara% testsem

process 14888 before critical section

process 14888 in critical section

process 14889 before critical section

process 14890 before critical section
cara%
process 14888 leaving critical section

process 14888 exiting

process 14889 in critical section

process 14889 leaving critical section

process 14889 exiting

process 14890 in critical section

process 14890 leaving critical section

process 14890 exiting
ipcs
IPC status from <running system> as of 2011년 5월 25일 수요일 오후 05시 35분 32초
T         ID      KEY         MODE         OWNER    GROUP
Message Queues:
q          1    0x40       --rw-rw----     hana5       cs
Shared Memory:
Semaphores:
s          3    0x200      --ra-------      kgu      prof
cara%
```

공유 메모리

- 둘 이상의 프로세스가 물리적인 메모리 일부를 공유
- 3가지 IPC 기법 중 가장 효율적
- 사용 방법
  - shmget( )을 이용하여 생성

    #include <sys/shm.h>

    int shmget(key_t key, size_t size, int permflags);
  - shmat( )를 이용하여 자신을 그 메모리에 부착

    #include <sys/shm.h>

    int *shmat(int shmid, const void *daddr, int shmflags);
    - daddr : 호출에 의해 선택된 주소
      - NULL : 첫번째 가용 주소 (일반적)
      - NULL이 아닌 경우 해당 주소 또는 그 근처
    - shmflags
      - SHM_RDONLY
      - SHM_RND : daddr이 NULL이 아닌 경우 이를 처리하는 방법 지정 (이것이 설정되어 있으면 메모리 페이지 경계에 맞추고, 그렇지 않으면 그 주소를 이용)
    - 오류 발생시 반환 값 (void *) -1
  - shmdt( )를 이용하여 자신을 분리

    int shmdt(const void *shmaddr);
    - 공유 메모리 영역을 논리적 주소공간 영역에서 분리
    - 성공이면 0, 실패 시 -1 반환

share2.h

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>

#define SHMKEY  (key_t) 0x10
#define SEMKEY  (key_t) 0x30

#define SIZ     (5 * BUFSIZ)

typedef struct {
        int     d_nread;
        char    d_buf[SIZ];
} databuf;

typedef union {
        int     val;
        struct semid_ds *buf;
        ushort          *array;
} semun;

void getseg(databuf **);
int getsem(void);
void remobj(void);
void reader(int, databuf*);
void writer(int, databuf*);
```

share_ops2.c (1)

```
#include "share2.h"

#define IFLAGS   (IPC_CREAT | IPC_EXCL)

static int shmid;
static int semid;

void getseg(databuf **p)
{
        if ((shmid = shmget(SHMKEY, sizeof(databuf), 0600 | IFLAGS)) == -1) {
                perror("shmget");
                exit(1);
        }

        if ((*p = (databuf *) shmat(shmid, 0, 0)) == ((databuf *) -1)) {
                perror("shmat");
                exit(2);
        }
}
```

share_ops2.c (2)

```
int getsem(void)
{
      semun   x;

      x.val = 0;

      if ((semid = semget(SEMKEY, 2, 0600 | IFLAGS)) == -1) {
            perror("semget");
            exit(3);
      }

      if (semctl(semid, 0, SETVAL, x) == -1) {
            perror("semctl in getsem");
            exit(4);
      }

      if (semctl(semid, 1, SETVAL, x) == -1) {
            perror("semctl in getsem");
            exit(5);
      }

      return(semid);
}
```

```
void remobj (void)
{
      if (shmctl(shmid, IPC_RMID, NULL) == -1) {
            perror("shmctl in remobj");
            exit(6);
      }

      if (semctl(semid, 0, IPC_RMID, NULL) == -1) {
            perror("semctl in remobj");
            exit(7);
      }
}
```

rd_wr2.c

```
#include "share2.h"

struct sembuf p0 = {0, -1, 0};
struct sembuf v0 = {0, 1, 0};

struct sembuf p1 = {1, -1, 0};
struct sembuf v1 = {1, 1, 0};

void reader(int semid, databuf *buf)
{
      for(;;) {
            buf->d_nread = read(0, buf->d_buf, SIZ);

            semop(semid, &v0, 1);   /* signal(sem0) 읽기 완료 표시 */

            if (buf->d_nread <= 0) {
                  return;
            }

            semop(semid, &p1, 1);   /* wait(sem1) 쓰기 대기  */


      }
}
```

```
void writer(int semid, databuf *buf)
{
      for(;;) {
            semop(semid, &p0, 1);   /* wait(sem0) 읽기 대기 */

            if (buf->d_nread <= 0)
                  return;

            write(1, buf->d_buf, buf->d_nread);

            semop(semid, &v1, 1);   /* signal(sem1) 쓰기 완료 표시 */
      }
}
```

shm_test2.c

```
#include "share2.h"

void main()
{
	int semid;
	pid_t   pid;
	databuf *buf;

	semid = getsem();

	getseg(&buf);

	switch (pid=fork()) {
		case -1:
			perror("fork");
			exit(-1);
		case 0:
			writer(semid, buf);
			remobj();
			break;
		default:
			reader(semid, buf);
	}

	exit(0);
}
```

- 실행 방법
  - shm_test2 < testfile >result