10 시스템V의 프로세스간 통신

- 유닉스 시스템V에서 제공하는 IPC기법을 이해한다.
- 메시지 큐를 이용해 통신 프로그램을 작성할 수 있다.
- 공유 메모리를 이용해 통신 프로그램을 작성할 수 있다.
- 세마포어를 이용한 IPC기법을 배운다.

목차

- 시스템 V IPC 기초
- 시스템 V IPC 관련 명령
- 메시지 큐
- 공유 메모리
- 세마포어

시스템 V IPC 기초[1]

시스템 V IPC

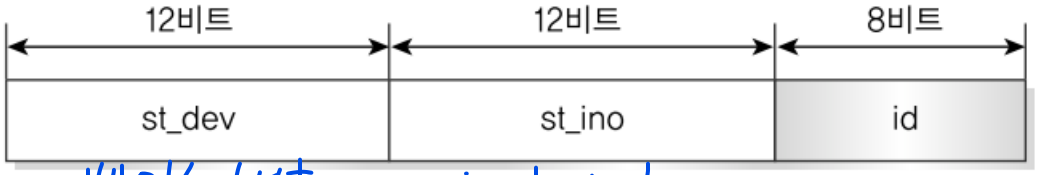
- 시스템 V 계열 유닉스에서 개발해 제공하는 프로세스 간 통신방법
- 메시지 큐, 공유 메모리, 세마포어

공통 요소

- 시스템 V IPC를 사용하기 위해서는 IPC 객체를 생성해야함.
- IPC 객체를 생성하기 위해 공통적으로 사용하는 기본 요소는 키와 식별자

키 생성

- 키로 IPC_PRIVATE 지정
- ftok 함수로 키 생성 file to key, path를 이용해서 암호화된 키 생성



[그림 10-1] 키의 구조

```
#include <sys/ipc.h>
key_t ftok(const char *path, int id);
```

경로, 1~255

- 인자로 파일시스템에 이미 존재하는 임의의 파일의 경로명과 1~255사이의 번호 지정
- 키의 구조에서 id(8비트)에 인자로 지정한 번호 저장. 번호에 0은 지정하지 않는다.

시스템 V IPC 기초[2]

IPC 공통 구조체

- IPC객체를 생성하면 IPC 공통 구조체가 정의된다.

```
struct ipc_perm {
    uid_t uid;
    gid_t gid;
    uid_t cuid;
    gid_t cgid;
    mode_t mode;
    uint_t seq;
    key_t key;
    int pad[4];
};
```

- uid, gid : 구조체 소유자ID 및 소유그룹ID
- cuid, cgid : 구조체를 생성한 사용자ID, 그룹ID
- mode : 구조체에 대한 접근 권한
- seq : 슬롯의 일련번호
- key : 키값
- pad : 향후 사용을 위해 예약된 영역

시스템 V IPC 관련 명령[1]

시스템 V IPC 정보 검색: ipcs 명령

- 시스템 V IPC의 정보를 검색하고 현재 상태 확인

```
ipcs [-aAbciJmopqstZ] [-D mtype]
```

- -m : 공유 메모리에 관한 정보만 검색
- -q : 메시지 큐에 관한 정보만 검색
- -s : 세마포어에 관한 정보만 검색
- -a : -b, -c, -o, -p, -t 옵션으로 검색하는 항목 모두 출력
- -A : 전체 항목을 모드 검색
- -b : 각 방법의 최댓값 검색
- -c : IPC 객체를 생성한 사용자의 로그인명과 그룹명 검색
- -D mtype : 메시지 큐에서 mtype으로 지정한 메시지만 검색
- -i : 공유 메모리 세그먼트에 연결된 ISM의 개수 출력
- -J : IPC 객체 생성자의 프로젝트명 출력
- -o : 현재 사용되고 있는 정보 출력
- -p : PID 정보 출력
- -t : 시간 정보 출력

시스템 V IPC 관련 명령[2]

IPCS 명령 사용 예

- 현재 동작중인 IPC 객체가 하나도 없는 경우

```
# ipcs
IPC status from <running system> as of 2009년 2월 18일 수요일 오전 09시 36분 41초
T         ID      KEY        MODE        OWNER    GROUP
Message Queues:
Shared Memory:
Semaphores:
```

- -A 옵션 지정시 : 모든 항목 출력

```
# ipcs -A
IPC status from <running system> as of 2009년 2월 18일 수요일 오전 10시 36분 41초
T         ID      KEY    MODE      OWNER    GROUP  CREATOR    CGROUP CBYTES
QNUM QBYTES LSPID LRPID   STIME     RTIME     CTIME           PROJECT
Message Queues:
T         ID      KEY    MODE      OWNER    GROUP  CREATOR    CGROUP NATTCH
SEGSZ  CPID  LPID   ATIME    DTIME    CTIME  ISMATTCH      PROJECT
Shared Memory:
T         ID      KEY    MODE      OWNER    GROUP  CREATOR    CGROUP NSEMS
OTIME    CTIME           PROJECT
Semaphores:
```

시스템 V IPC 관련 명령[3]

시스템 V IPC 정보 삭제: ipcrm

```
ipcrm [-m shmid] [-q msqid] [-s semid] [-M shmkey] [-Q msgkey] [-S emkey]
```

- -m shmid : 공유 메모리 삭제
- -q msqid : 메시지 큐 삭제
- -s semid : 세마포어 삭제
- -M shmkey : shmkey로 지정한 공유 메모리 삭제
- -Q msgkey : msgkey로 지정한 공유 메모리 삭제
- -S semkey : semkey로 지정한 공유 메모리 삭제

메시지 큐[1]

메시지 큐

- 파이프와 유사하나 파이프는 스트림 기반으로 동작하고 메시지 큐는 메시지 단위로 동작
- 각 메시지의 최대 크기는 제한되어 있음
- 각 메시지에는 메시지 유형이 있어 수신 프로세스는 어떤 유형의 메시지를 받을 것인지 선택 가능

메시지 큐 생성: msgget(2)

```
#include <sys/msg.h>
int msgget(key_t key, int msgflg);
```

키, 접근 권한

- key : IPC_PRIVATE 또는 ftok로 생성한 키값
- msgflg : 플래그와 접근 권한 지정
  - IPC_CREAT : 새로운 키면 식별자를 새로 생성
  - IPC_EXCL : 이미 존재하는 키면 오류 발생
- 메시지 큐 식별자를 리턴(msqid_ds 구조체)

메시지 큐[2]

msqid_ds 구조체

```
struct msqid_ds {
    struct ipc_perm
msg_perm;
    struct msg *msg_first;
    struct msg *msg_last;
    msglen_t msg_cbytes;
    msgqnum_t msg_qnum;
    msglen_t msg_qbytes;
    pid_t msg_lspid;
    pid_t msg_lrpid;
    time_t msg_stime;
    int32_t msg_pad1;
    time_t msg_rtime;
    int32_t msg_pad2;
    time_t msg_ctime;
    int32_t msg_pad3;
    short msg_cv;
    short msg_qnum_cv;
    long msg_pad4[3];
};
```

- msg_perm: IPC공통 구조체
- msg_first: 첫번째 메시지에 대한 포인터
- msg_last: 마지막 메시지에 대한 포인터
- msg_cbytes: 현재 메시지큐에 있는 총 바이트수
- msg_qnum: 메시지 큐에 있는 메시지 개수
- msg_qbytes: 메시지 큐의 최대 크기
- msg_lspid: 마지막으로 메시지를 보낸 프로세스ID
- msg_lrpid: 마지막으로 메시지를 읽은 프로세스ID
- msg_stime: 마지막으로 메시지를 보낸 시각
- msg_rtime: 마지막으로 메시지를 읽은 시각
- msg_ctime: 마지막으로 메시지 큐의 권한변경시각
- msg_pad1,2,3: 예비공간

메시지 큐[3]

메시지 전송: msgsnd(2)

```
#include <sys/msg.h>
int msgsnd(int msqid, const void *msgp,  size_t  msgsz,  int msgflg);
```

- msqid : 메시지 큐 식별자
- msgp : 메시지 버퍼 주소
- msgsz : 메시지 버퍼 크기
- msgflg : 블록모드(0)/비블록 모드(IPC_NOWAIT)

기다리지 마라-

- 메시지 버퍼 구조체

```
struct msqbuf {
    long mtype;
    char mtext[1];
};
```

- mtype : 메시지 유형으로 양수를 지정
- mtext : 메시지 내용 저장

[예제 10-1] 메시지 큐 생성 및 메시지 전송하기

ex10_1.c

<sys/msg.h> <stdlib.h> <stdio.h> <string.h>

```
06  struct mymsgbuf {
07      long mtype;
08      char mtext[80];
09  };
10
11  int main(void) {
12      key_t key;
13      int msgid;
14      struct mymsgbuf mesg;
15
16      key = ftok("keyfile", 1);
17      msgid = msgget(key, IPC_CREAT|0644);
18      if (msgid == -1) {
19          perror("msgget");
20          exit(1);
21      }
22
```

메시지 버퍼 정의

키 값 생성

메시지 큐 생성

[예제 10-1] 메시지 큐 생성 및 메시지 전송하기

ex10_1.c

```
23      mesg.mtype = 1;
24      strcpy(mesg.mtext, "Message Q Test\n");
25
26      if (msgsnd(msgid, (void *)&mesg, 80, IPC_NOWAIT) == -1) {
27          perror("msgsnd");
28          exit(1);
29      }
30
31      return 0;
32  }
```

보낼 메시지 만들기

메시지 전송

```
# ex10_1.out
# ipcs -q
IPC status from <running system> as of 2009년 2월 18일 수요일 오후 2시 01분 14초
T         ID      KEY        MODE        OWNER    GROUP CBYTES  QNUM
Message Queues:
q          1   0x100719c --rw-r--r--     root     other     80     1
```

← 이것만 치면 안나옴, 프로그램 자체가 화면에 보여주는 것 없음

0644

메시지 큐 수

-q 안붙이면 IPC 에 대한 모든 객체 출력,

-q 붙이면 메시지 큐만 나옴

------ Message Queues --------
key msqid owner perms used-bytes messages
0xffffffff 0 jinhwan 644 80 1

------ Shared Memory Segments --------
key shmid owner perms bytes nattch status
0x00000000 720896 jinhwan 600 524288 2 dest
0x00000000 851969 jinhwan 600 524288 2 dest
0x00000000 425986 jinhwan 600 16777216 2
0x00000000 524291 jinhwan 600 524288 2 dest
0x00000000 622596 jinhwan 600 524288 2 dest
0x00000000 1310725 jinhwan 600 524288 2 dest
0x00000000 1409030 jinhwan 600 524288 2 dest
0x00000000 1048583 jinhwan 600 524288 2 dest
0x00000000 1179656 jinhwan 600 524288 2 dest
0x00000000 1212425 jinhwan 600 16777216 2 dest
0x00000000 1507338 jinhwan 600 524288 2 dest

------ Semaphore Arrays --------
key semid owner perms nsems

메시지 큐[4]

메시지 수신: msgrcv(2)

```
#include <sys/msg.h>
ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long int msgtyp,
int msgflg);
```

- msqid : 메시지 큐 식별자
- msgp : 메시지 버퍼 주소
- msgsz : 메시지 버퍼 크기
- msgtyp : 읽어올 메시지 유형
- msgflg : 블록모드(0)/비블록모드(IPC_NOWAIT)

- msgtyp에 지정할 값
  - 0 : 메시지 큐의 다음 메시지를 읽어온다.
  - 양수 : 메시지 큐에서 msgtyp로 지정한 유형과 같은 메시지를 읽어온다.
  - 음수 : 메시지의 유형이 msgtyp로 지정한 값의 절대값과 같거나 작은 메시지를 읽어온다.

-3 이면 1,2,3 읽음

[예제 10-2] 메시지 수신하기

ex10_2.c

```
05  struct mymsgbuf {
06      long mtype;
07      char mtext[80];
08  };
09
10  int main(void) {
11      struct mymsgbuf inmsg;
12      key_t key;
13      int msgid, len;
14
15      key = ftok("keyfile", 1);
16      if ((msgid = msgget(key, 0)) < 0
17          perror("msgget");
18          exit(1);
19      }
20
21      len = msgrcv(msgid, &inmsg, 80, 0, 0);
22      printf("Received Msg = %s, Len=%d\n", inmsg
23
24      return 0;
25  }
```

메시지 버퍼 정의

기존에 있는 것을 사용

송신측과 같은 키값 생성

NO_WAIT 아니라 기다리기

메시지 수신

inmsg.mtext, len);

```
# ex10_2.out
Received Msg = Message Q Test, Len=80
# ipcs -q
IPC status from <running system> as of 2009년 2월 18일 수요일 오후 2시 03분 48초
T         ID      KEY        MODE        OWNER    GROUP CBYTES  QNUM
Message Queues:
q          1   0x100719c --rw-r--r--      root    other     0     0
```

메시지 큐[5]

메시지 제어: msgctl(2)

```
#include <sys/msg.h>
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

- msqid : 메시지 큐 식별자
- cmd : 수행할 제어기능
- buf : 제어 기능에 사용되는 메시지 큐 구조체 주소

- cmd에 지정할 값
  - IPC_RMID : 메시지 큐 제거
  - IPC_SET : 메시지 큐 정보 중 msg_perm.uid, msg_perm.gid, msg_perm.mode, msg_qbytes 값을 세번째 인자로 지정한 값으로 변경
  - IPC_STAT : 현재 메시지 큐의 정보를 buf에 저장

[예제 10-3] 메시지 큐 삭제하기

ex10_3.c

```
...
05  int main(void) {
06      key_t key;
07      int msgid;
08
09      key = ftok("keyfile", 1);
10      msgid = msgget(key, IPC_CREAT|0644);
11      if (msgid == -1) {
12          perror("msgget");
13          exit(1);
14      }
15
16      printf("Before IPC_RMID\n");
17      system("ipcs -q");
18      msgctl(msgid, IPC_RMID, (struct msqid_ds *)NULL);
19      printf("After IPC_RMID\n");
20      system("ipcs -q");
21
22      return 0;
23  }
```

키값 생성

메시지 큐 삭제

[예제 10-3] 실행결과

```
# ex10_3.out
Before IPC_RMID
IPC status from <running system> as of 2009년 2월 18일 수요일 오후 2시 21분 47초
T         ID      KEY        MODE        OWNER    GROUP
Message Queues:
q          1   0x100719c --rw-r--r--     root    other
After IPC_RMID
IPC status from <running system> as of 2009년 2월 18일 수요일 오후 2시 21분 47초
T         ID      KEY        MODE        OWNER    GROUP
Message Queues:
```

공유 메모리[1]

공유 메모리

- 같은 메모리 공간을 두 개 이상의 프로세스가 공유하는 것
- 같은 메모리 공간을 사용하므로 이를 통해 데이터를 주고 받을 수 있음

공유 메모리 생성: shmget(2)

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(key_t key, size_t size, int shmflg);
```

- key : IPC_PRIVATE 또는 ftok로 생성한 키값
- size : 공유할 메모리 크기
- shmflg : 공유 메모리의 속성을 지정하는 플래그
  - IPC_CREAT, IPC_EXCL

- 공유 메모리 식별자를 리턴(shmid_ds 구조체)

공유 메모리[2]

shmid_ds 구조체

```
struct shmid_ds {
    struct ipc_perm msg_perm;
    size_t shm_segsz;
    struct anon_map *shm_amp;
    pid_tt_t shm_lpid;;
    pid_t shm_cpid;
    shmatt_t shm_nattch;
    ulong_t shm_cnattch;
    time_t shm_atime;
    int32_t shm_pad1;
    time_t shm_dtime;
    int32_t shm_pad2;
    time_t shm_ctime;
    int32_t shm_pad3;
    int32_t shm_pad4[4];
};
```

- shm_perm: IPC공통 구조체
- shm_segsz: 공유 메모리 세그먼트 크기
- shm_lpid: 마지막으로 shmop동작을 한 프로세스ID
- shm_cpid : 공유 메모리를 생성한 프로세스ID
- shm_nattach: 공유 메모리를 연결하고 있는 프로세스 수
- shm_atime: 마지막으로 공유 메모리를 연결한 시각
- shm_dtime: 마지막으로 공유 메모리 연결을 해제한 시각
- shm_ctime: 마지막으로 공유 메모리 접근 권한을 변경한 시각

[예제 10-4] 공유 메모리 생성하기

ex10_4.c

```
...
07  int main(void) {
08      key_t key;
09      int shmid;
10
11      key = ftok("shmfile", 1);
12      shmid = shmget(key, 1024, IPC_CREAT|0644);
13      if (shmid == -1) {
14          perror("shmget");
15          exit(1);
16      }
17
18      return 0;
19  }
```

키 생성

공유 메모리 생성.

```
# ex10_4.out
# ipcs -m
IPC status from <running system> as of 2009년 2월 18일 수요일 오후 03시 06분 01초
T         ID      KEY        MODE        OWNER    GROUP NATTCH
Shared Memory:
m          0   0x100719b  --rw-r--r--     root     other      0
```

shared 메모리에 연결된 프로세스

공유 메모리[3]

공유 메모리 연결: shmat(2)

```
#include <sys/types.h>
#include <sys/shm.h>
void *shmat(int shmid, const void *shmaddr, int shmflg);
```

- shmid : 공유 메모리 식별자
- shmaddr : 공유 메모리를 연결할 주소
- shmflg : 공유 메모리에 대한 읽기/쓰기 권한
  - 0(읽기/쓰기 가능), SHM_RDONLY(읽기 전용)

공유 메모리 연결 해제: shmdt(2)

```
#include <sys/types.h>
#include <sys/shm.h>
int shmdt(char *shmaddr);
```

- shmaddr: 연결을 해제할 공유 메모리 주소

공유 메모리[4]

공유 메모리 제어: shmctl(2)

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

- cmd : 수행할 제어기능
  - IPC_RMID : 공유 메모리 제거
  - IPC_SET : 공유 메모리 정보 내용 중 shm_perm.uid, shm_perm.gid, shm_perm.mode 값을 세번째 인자로 지정한 값으로 변경
  - IPC_STAT : 현재 공유 메모리의 정보를 buf에 지정한 메모리에 저장
  - SHM_LOCK : 공유 메모리를 잠근다.
  - SHM_UNLOCK : 공유 메모리의 잠금을 해제한다.

[예제 10-5] 공유 메모리 사용하기

ex10_5.c

- 부모/자식 프로세스 간 공유 메모리 사용 예제

```
...
09  int main(void) {
10      int shmid, i;
11      char *shmaddr, *shmaddr2;
12
13      shmid = shmget(IPC_PRIVATE, 20, IPC_CREAT|0644);
14      if (shmid == -1) {
15          perror("shmget");
16          exit(1);
17      }
18
19      switch (fork()) {
20          case -1:
21              perror("fork");
22              exit(1);
23              break;
```

ftok 없이도됨

공유 메모리 생성

[예제 10-5] 공유 메모리 사용하기

ex10_5.c

```
24          case 0:
25              shmaddr = (char *)shmat(shmid, (char *)NULL, 0);
26              printf("Child Process =====\n");
27              for (i=0; i<10; i++)
28                  shmaddr[i] = 'a' + i;
29              shmdt((char *)shmaddr);
30              exit(0);
31              break;
32          default:
33              wait(0);
34              shmaddr2 = (char *)shmat(shmid, (char *)NULL, 0);
35              printf("Parent Process =====\n");
36              for (i=0; i<10; i++)
37                  printf("%c ", shmaddr2[i]);
38              printf("\n");
39              sleep(5);
40              shmdt((char *)shmaddr2);
41              shmctl(shmid, IPC_RMID, (struct shmid_ds *)NULL);
42              break;
43      }
44
45      return 0;
46  }
```

시작 주소 리턴

공유 메모리 연결

공유메모리에 데이터 기록 후 공유 메모리 연결 해제

공유 메모리 연결

공유 메모리 내용 출력

공유 메모리 연결 해제

공유 메모리 삭제

5초 지나기전에 ipcs -m 을 치는 것과, 그 후에 치는게 다름

```
# ex10_5.out
Child Process =====
Parent Process =====
a b c d e f g h i j
```

[예제 10-6] (1) listener (서버역할)

ex10_6s.c

- 독립적인 프로세스 간 공유 메모리 사용 예제 <string.h> <stdio.h>

```
...<sys/types.h><sys/mman.h><singal.h><ipc.h><shm.h><unistd.h>
10  void handler(int dummy) {
11      ;
12  }
13
14  int main(void) {
15      key_t key;
16      int shmid;
17      void *shmaddr;
18      char buf[1024];
19      sigset_t mask;
20
21      key = ftok("shmfile", 1);
22      shmid = shmget(key, 1024, IPC_CREAT|0666);
23
24      sigfillset(&mask);
25      sigdelset(&mask, SIGUSR1);
26      sigset(SIGUSR1, handler);
27
28      printf("Listener wait for Talker\n");
```

키 생성

공유 메모리 생성

모든 시그널을 다

시그널 처리 지정

이것만 0으로

SIGUSR1 오면 handler 실행

[예제 10-6] (1) listener (서버역할)

ex10_6s.c

```
29      sigsuspend(&mask);
30
31      printf("Listener Start =====\n");
32      shmaddr = shmat(shmid, NULL, 0);
33      strcpy(buf, shmaddr);
34      printf("Listener received : %s\n", buf);
35
36      strcpy(shmaddr, "Have a nice day\n");
37      sleep(3);
38      shmdt(shmaddr);
39
40      return 0;
41  }
```

시그널 올 때까지 대기

시그널이 오면 공유 메모리 연결

메모리 내용 읽고 출력

공유 메모리에 쓰기

연결해제

[예제 10-6] (2) talker(클라이언트 역할)

ex10_6c.c

<sys/types.h> <signal.h> <mman.h> <ipc.h> <shm.h> <unistd.h> <stdlib.h> <string.h> <stdio.h>

```
...
11  int main(int argc, char **argv) {
12      key_t key;
13      int shmid;
14      void *shmaddr;
15      char buf[1024];
16
17      key = ftok("shmfile", 1);
18      shmid = shmget(key, 1024, 0);
19
20      shmaddr = shmat(shmid, NULL, 0);
21      strcpy(shmaddr, "Hello, I'm talker\n");
22
23      kill(atoi(argv[1]), SIGUSR1);
24      sleep(2);
25      strcpy(buf, shmaddr);
26
27      printf("Listener said : %s\n", buf);
28      system("ipcs -m");
29      shmdt(shmaddr);
30      shmctl(shmid, IPC_RMID, NULL);
31
32      return 0;
33  }
```

서버와 같은 키 생성

공유 메모리 정보 가져오기

이미 존재하는 것 쓰기

공유 메모리 연결하고 데이터 기록

4096 b

시그널 발송

서버가 보낸 메시지 읽어 출력

공유 메모리 연결 해제 및 삭제

[예제 10-6] 실행결과

```
# listener &
[1]     4946
# Listener wait for Talker
Listener Start =====
Listener received : Hello, I'm talker
# talker 4946
Listener said : Have a nice day
IPC status from <running system> as of 2009년 2월 18일 수요일 오후 07시 53분 12초
T         ID      KEY        MODE        OWNER    GROUP NATTCH
Shared Memory:
m          4   0x100719b  --rw-rw-rw-      root    other      2
# ipcs
IPC status from <running system> as of 2009 년 2월 18일 수요일 오후 07시 53분 57초
T         ID      KEY        MODE        OWNER    GROUP
Message Queues:
Shared Memory:
Semaphores:
```

세마포어[1]

세마포어

- 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

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 값을 1 감소;
}
```

- 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;
    if (대기중인 프로세스가 있으면)
        대기중인 첫 번째 프로세스를 동작시킨
}
```

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);
```

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

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;
};
```

- semval : 세마포어 값
- sempid : 세마포어 연산을 마지막으로 수행한 프로세스 PID
- semncnt: 세마포어 값이 현재 값보다 증가하기를 기다리는 프로세스 수
- semzcnt: 세마포어 값이 0이 되기를 기다리는 프로세스 수

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, ...);
```

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

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

cmd 에 따라 선택 여부 결정

cmd에 지정할 수 있는 값

- IPC_RMID, IPC_SET, IPC_STAT : 메시지 큐, 공유 메모리와 동일 기능
- GETVAL : 세마포어의 semval 값을 읽어온다.
- SETVAL : 세마포어의 semval 값을 arg.val로 설정한다.
- GETPID : 세마포어의 sempid 값을 읽어온다.
- GETNCNT, GETZNCNT : 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)

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

세마포어 연산을 의미

세마포어 연산

- sembuf 구조체의 sem_op 항목에 지정

```
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;
```

세마포어[7]

1. sem_op가 음수 : 세마포어 잠금 기능 수행
   - semval 값이 sem_op의 절댓값과 같거나 크면 semval 값에서 sem_op의 절댓값을 뺀다.
   - semval 값이 sem_op 값보다 작고 sem_flg에 IPC_NOWAIT가 설정되어 있으면 semop 함수는 즉시 리턴
   - 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.
   ② 시스템에서 semid가 제거된다. 이 경우 errno가 EIDRM으로 설정되고 -1을 리턴한다.
   ③ semop 함수를 호출한 프로세스가 시그널을 받는다. 이 경우 semncnt 값은 감소하고 시그널 처리 함수를 수행한다.

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

3. 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

ex10_7.c

```
<errno.h><sys/types.h><sys/ipc.h><sys/sem.h><unistd.h><stdlib.h><stdio.h>
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
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      }
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

세마포어 값을 1로 초기화

[예제 10-7] (2) 세마포어 연산

ex10_7.c

```
36
37  int semlock(int semid) {
38      struct sembuf buf;
39
40      buf.sem_num = 0;
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
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

ex10_7.c

```
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.

[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.

Thank You !

IT CookBook, 유닉스 시스템 프로그래밍

한빛미디어 Hanbit Media, Inc.