Advanced Inter-process Communications – Part1

Advanced Inter-process Communication

- Record locking with fcntl
- Three IPC facilities
  - Message passing
    - Allow a process to send and receive messages, a message is an arbitrary sequence of bytes or characters
  - Semaphores
    - Provide a rather low-level means of process synchronization, not suitable to the transmission of large amounts of information
  - Shared memory
    - Allow two or more processes to share the data contained in specific memory segments

Record Locking with fcntl

- int fcntl(int *fd*, int *cmd*, struct flock **ldata*);
- Read lock
  - Simply prevents any process from applying the other type of fcntl lock, which is called a write lock.
  - Several processes may read lock the same segment simultaneously.
  - *fd* must have been opened using O_RDONLY or O_RDWR, so a file descriptor from creat won't work.
- Write lock
  - Stops any process from applying a read or write lock to the file.
  - *fd* must have been opened using O_WRONLY or O_RDWR.

File Lock Structure

```
struct flock {
    short l_type;      /* F_RDLCK, F_WRLCK, F_UNLCK */
    short l_whence;    /* SEEK_SET, SEEK_CUR, SEEK_END */
    long l_start;      /* byte offset, interpreted according to l_whence */
    long l_len;        /* number of bytes to lock.
    long l_pid;        /* ID of process that locked file */
};
```

File Lock Structure Fields

- l_type
  - F_RDLCK for read lock, F_WRLCK for write lock, and F_UNLCK for unlock operations.
- l_whence
  - SEEK_SET: (= 0) lock offset is from beginning of file.
  - SEEK_CUR: (= 1) current position of the file pointer.
  - SEEK_END: (= 2) end of file.
- l_start
  - Byte offset, interpreted according to l_whence.
- l_len
  - Number of bytes to lock
  - If 0, lock from l_start to end of file.

Lock Commands

- *cmd*
  - F_GETLK
    - Get lock description based on data passed via the *ldata* (describe the first lock that blocks the lock described in *ldata)*
  - F_SETLK
    - Lock or unlock the file as specified by *ldata*. Return *immediately* with -1 if unable to lock.
  - F_SETLKW
    - Lock or unlock data in file as specified by *ldata*. *Sleep* if unable to lock.
    - A process sleeping on a fcntl lock can be interrupted by a *signal*.

Features of Record Locking with fcntl()

- Locks are associated with a process and a file
- Locks are *never* inherited by the child across a fork( )
- A call to fcntl() does not alter the file's read-writer pointer
- All locks belonging to a process are removed automatically when the process dies

Example #1: Record Lock (1)

```
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>

main()
{
	int fd;
	struct flock my_lock;

	/* 쓰기 록의 인수를 지정 */
	my_lock.l_type = F_WRLCK;
	my_lock.l_whence = SEEK_SET;
	my_lock.l_start = 0;
	my_lock.l_len = 10;

	/* 화일을 개방한다. */
	fd = open ("locktest", O_RDWR);
```

Example #1: Record Lock (2)

```
/* 처음 10바이트를 록한다 */
if ( fcntl (fd, F_SETLKW, &my_lock) == -1)
{
        perror ("parent: locking");
        exit (1);
}
printf ("parent: locked record□n");
switch (fork()){
case -1:                        /* 오류 */
        perror ("fork");
        exit (1);
case 0:                         /* 자식 */
        my_lock.l_len = 5;
        if ( fcntl (fd, F_SETLKW, &my_lock) == -1)
        {
                perror ("child: locking");
                exit (1);
        }
        printf ("child: locked□n");
        printf ("child: exiting□n");
        exit(0);
}
```

Example #1: Record Lock (3)

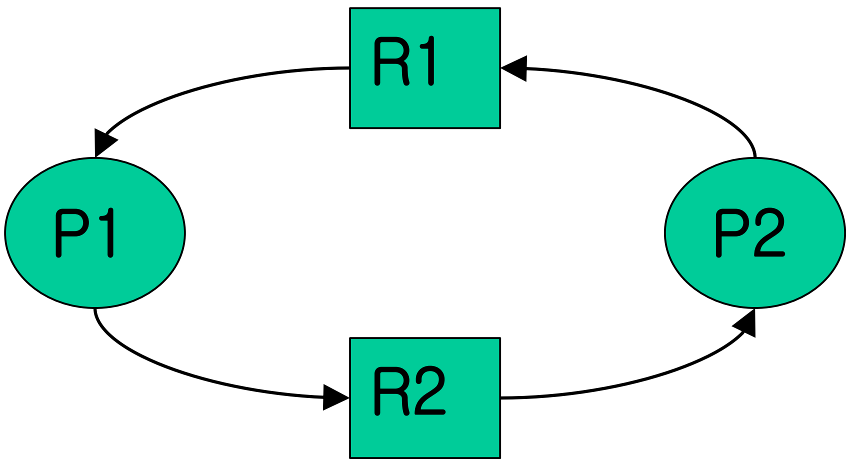
```
sleep(5);

/* 이제 퇴장(exit)한다. 따라서 록이 해제된다 */
printf ("parent: exiting□n");
exit (0);
}
```

```
dhlee@kde:~/Course/SP/example>flck-test
parent: locked record
parent: exiting
child: locked
child: exiting
```

Dead Lock 복습

- Dead Lock의 국문용어?
- Dead Lock이 일어날 4가지 조건
  - Mutual exclusion
  - Hold and wait
  - No preemption
  - Circular wait
- Dead Lock Handing
  - Prevention
  - Avoidance
  - Detection & Recovery



RAG

Example #2 : Dead Lock (1)

```
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>

main()
{
	int fd;
	struct flock first_lock;
	struct flock second_lock;

	first_lock.l_type = F_WRLCK;
	first_lock.l_whence = SEEK_SET;
	first_lock.l_start = 0;
	first_lock.l_len = 10;


	second_lock.l_type = F_WRLCK;
	second_lock.l_whence = SEEK_SET;
	second_lock.l_start = 10;
	second_lock.l_len = 5;
```

Example #2 : Dead Lock (2)

```
fd = open ("locktest", 0_RDWR);

if ( fcntl (fd, F_SETLKW, &first_lock) == -1)        /* A */
        fatal ("A");

printf ("A: lock succeeded (proc %d)□n", getpid());

switch (fork()){
case -1:
        /* 오류 */
        fatal ("error on fork");
case 0:
        /* 자식 */
        if ( fcntl (fd, F_SETLKW, &second_lock) == -1)  /* B */
                fatal ("B");
        printf ("B: lock succeeded (proc %d)□n", getpid());

        if ( fcntl (fd, F_SETLKW, &first_lock) == -1)   /* C */
                fatal ("C");
        printf ("C: lock succeeded (proc %d)□n", getpid());
        exit (0);
```

Example #2 : Dead Lock (3)

```
        default:
                /* 부모 */
                printf ("parent sleeping□n");
                sleep (10);
                if ( fcntl (fd, F_SETLKW, &second_lock) == -1)   /* D */
                        fatal ("D");
                printf ("D: lock succeeded (proc %d)□n", getpid());
        }
}
```

```
dhlee@kde:~/Course/SP/example>deadlock
A: lock succeeded (proc 21884)
B: lock succeeded (proc 21885)
parent sleeping
D: Resource deadlock avoided
D: lock succeeded (proc 21884)
C: lock succeeded (proc 21885)
```

IPC Common Features (1)

- Three commonly used IPC facilities
  - Message passing
  - Semaphores
  - Shared Memory
- An IPC facility must be created prior to use.
  - It can be created anytime before use.
- Permission and ownership for a facility are established at the time of creation.
  - But these attributes can be changed later.
- An IPC facility can exist with contents intact after all processes accessing it have exited.
  - Changes made to an IPC facility by a process persist after the process exits.
- An IPC facility must be explicitly removed.
  - Removal of an IPC facility can be done by either the owner or creator.

IPC Common Features (2)

- Needs an IPC *key* to gain access to an IPC facility.
- All processes wishing to use the same IPC facility must specify the same key.
  - This key must uniquely identify the IPC facility.
- An IPC facility is identified by a nonnegative integer.
  - In need of an IPC key to generate such a numeric ID.
- Three ways to generate an IPC key:
  - Let the system pick a key (IPC_PRIVATE)
  - Pick a key directly (use a predefined key and store it in a common header file)
  - Use key_t ftok(const char **path*, int *id*) to generate an IPC key.

Message Queues

- A message queue allows processes to send and receive discrete amounts of data, know as *messages*.
  - A message is a collection of bytes of varying lengths, ranging from zero to a system imposed maximum.
  - The message content can be anything -- ASCII or binary.
- Each message has a type associated with it.
  - A message type is a long integer.

Semaphores

- Semaphores are useful for process synchronization and resource management.
  - A semaphore is an unsigned integer shared among several competing processes.
  - Operating upon a semaphore involves adding or subtracting from the underlying semaphore value.
- The UNIX implementation of semaphores is modeled after Dijkstra's semaphores.

Shared Memory

- Shared memory allows one or more process to share the same segment of memory.
  - Shared memory may be attached to more than one process.
  - Once attached, it is part of a process's data space.
- Shared memory is the fastest IPC.

IPC System Call Categories

- *get*: Is used to create or access an IPC facility. It returns an IPC identifier.
  - semget( ), shmget( ), msgget( )
- *ctl*: Is used to determine status, set options and/or permissions, or remove an IPC facility.
  - semctl( ), shmctl( ), msgctl( )
- *op*: Is used to operate on an IPC identifier.
  - semop( )
  - shmat( ), shmdt( )
  - msgsnd( ), msgrcv( )

End-User Level Commands

- ipcs
  - Report inter-process communication facilities status.
- ipcrm
  - Remove a message queue, semaphore set, or shared memory ID.

Message Queues (1)

- Message queues are for exchange of data among processes.
- The data exchanged are in discrete portions known as *messages*.
  - A message is a sequence of bytes, from zero to the system limit, with a message type.
  - The structure of a message is up to the programmer.
- Once a message queue is created, processes can send/receive messages to/from the queue.
  - A process can selectively read messages on a queue:
    - First message on the queue.
    - First message of a specific type on the queue.
    - First message from a range of types (from 1 to upper bound of range) on the queue.

Message Queues (2)

- Messages queues are similar to unnamed pipes; however,
  - Each message has a type associated with it.
  - Data in a pipe is just a sequence of types with no header of format.
  - Unread data in a pipe vanishes when the last reader process close a file descriptor associated with the pipe's read end.
  - A message queue sticks around.
  - It will remain active until it is explicitly removed.

Message System Calls

- msgget( )
  - To create or to gain access to a message queue.
- msgctl( )
  - To determine the status of a message queue.
  - To change the permission or ownership of a message queue.
  - To change the maximum size.
  - To remove a message queue.
- msgsnd( )
  - To send a message.
- msgrcv( )
  - To receive a message.

Some Comments

- The System V IPC structures are system-wide.
  - They stick around until explicitly removed.
- Not compatible with standard UNIX file manipulation facilities.
  - Can't use open( ), close( ), read( ), write( ), etc.
  - Almost a dozen of new system calls were added to the kernel to support these IPC facilities.
- Basically, they are intra-system IPC facilities.

msgget

- #include <sys/types.h>

  #include <sys/ipc.h>

  #include <sys/msg.h>

  int msgget(key_t *key*, int *flag*);

  – msgget returns an identifier to a message queue whose name is *key*.
  – *key* can specify that the returned queue identifier should refer to a private queue (IPC_PRIVATE), in which case a new message queue is created.
  – *flag* specifies if the queue should be created (IPC_CREAT), and if creation of the queue should be exclusive (IPC_EXCL). In the latter case, msgget fails (-1 will be returned) if the queue already exists.
  – mgid = msgget((key_t)0100, 0644|IPC_CREAT|IPC_EXECL)

msgsnd and msgrcv (1)

- #include <sys/types.h>
  #include <sys/ipc.h>
  #include <sys/msg.h>
  int msgsnd(int *id*, struct msgbuf **msgp*, int *size*, int *flag*);
  int msgrcv(int *id*, msgbuf **msgp*, int *size*, int *type*, int *flag*);
  - msgsnd sends a message of *size* bytes in the buffer *msgp* to the message queue *id*. msgbuf is defined as
    - ```
      struct msgbuf {
          long mtype;
          char mtext[ ];
      };
      ```
      ```

msgsnd and msgrcv (2)

- If the IPC_NOWAIT bit is off in *flag*, msgsnd sleeps if the number of bytes on the message queue exceeds the maximum, or if the number of system-wide messages exceeds a maximum value. If IPC_NOWAIT is set, msgsnd returns immediately in these cases
- msgrcv receives messages from the queue identified by *id*.
  - If *type* is 0, the first message on the queue is received;
  - If positive, the first message of that type is received;
  - If negative, the first message of the lowest type less than or equal to type's absolute value is received.
  - Example
    - The queue contains three message with mtype value of 999, 5, 1. if type of msgrcv is 0, returns 999; if 5, returns 5; if -999, returns in order 1, 5, and 999;

msgsnd and msgrcv (3)

- *size* in msgrcv indicates the maximum size of message text the user wants to received. If MSG_NOERROR is set in *flag*, the kernel truncates the received message if its size is larger than *size*. Otherwise it returns an error.
- If IPC_NOWAIT is not set in *flag*, msgrcv sleeps until a message that satisfies *type* is sent. If IPC_NOWAIT is set, it returns immediately.
- msgrcv returns the number of bytes in the message text.

Example #3: Message (1)

```
/* q.h -- header for message facility example */

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <errno.h>

extern int errno;

#define QKEY (key_t)0150 /*identifying key for queue*/
#define QPERM 0660 /*permissions for queue*/
#define MAXOBN 50 /*maximum length obj. name*/
#define MAXPRIOR 10 /*maximum priority level*/

struct q_entry { /*struct we well use for message*/
    long mtype;
    char mtext[MAXOBN+1];
};
```

Example #3: Message (2)

```
/* function prototypes */
int enter(char *objname, int priority);
int init_queue(void);
int serve(void);
```

Example #3: Message (3)

```
/* enter -- place an object into queue */

#include <string.h>
#include "q.h"

static int s_qid = -1; /*message queue identifier*/

int enter(char *objname, int priority)
{
    int len;
    struct q_entry s_entry; /*structure to hold message*/

    /*validate name length, priority level*/
    if ((len = strlen(objname)) > MAXOBN) {
        fprintf(stderr, "name too logn");
        return (-1);
    }
```

Example #3: Message (4)

```
    if (priority > MAXPRIOR || priority < 0) {
        fprintf(stderr, "invalid priority level");
        return (-1);
    }

    /*initial message queue as necessary*/
    if (s_qid == -1 && (s_qid = init_queue()) == -1)
        return (-1);

    /* initialize s_entry */
    s_entry.mtype = (long) priority;
    strncpy(s_entry.mtext, objname, MAXOBN);

    /*send message, waiting if necessary*/
    if (msgsnd(s_qid, &s_entry, len, 0) == -1) {
        perror("msgsnd failed");
        return (-1);
    }
    else
        return (0);
}
```

Example #3: Message (5)

```
/* init_queue -- get queue identifier */

#include "q.h"

int init_queue(void)
{
    int queue_id;

    /*attemp to create message queue*/
    if ((queue_id = msgget(QKEY, IPC_CREAT|QPERM)) == -1)
        perror("msgget failed");
    return (queue_id);
}
```

Example #3: Message (6)

```
/*server -- serve object with highest priority on queue */

#include "q.h"

static int r_qid = -1;

Int proc_obj(struct q_entry *msg)
{
    printf("\npriority: %ld name: %s\n", msg->mtype, msg->mtext);
}

int serve(void)
{
    struct q_entry r_entry;
    int mlen;

    /*initialize queue as necessary*/
    if (r_qid == -1 && (r_qid = init_queue()) == -1)
        return (-1);
```

Example #3: Message (7)

```
    /*get and process next message, waiting if necessary*/
    for(;;) {
        if ((mlen = msgrcv(r_qid, &r_entry, MAXOBN, (long) -1*MAXPRIOR,
                                MSG_NOERROR)) == -1) {
            perror("msgrcv failed");
            return (-1);
        }
        else {
            /*make sure we've a string*/
            r_entry.mtext[mlen] = '\0';

            /*print object name*/
            proc_obj(&r_entry);
        }
    }
}
```

Example #3: Message (8)

```
/*etest -- enter object name*/

#include <stdio.h>
#include "q.h"

int main(int argc, char *argv[])
{
    int priority;

    if (argc != 3) {
        fprintf(stderr, "usage: %s objname priority\n", argv[0]);
        exit(1);
    }
    if ((priority = atoi(argv[2])) <= 0 || priority > MAXPRIOR) {
        fprintf(stderr, "invalid priority");
        exit(2);
    }
```

Example #3: Message (9)

```
    if (enter(argv[1], priority) < 0) {
        fprintf(stderr, "enter failure");
        exit(3);
    }
    exit(0);
}
```

Example #3: Message (10)

```
/*stest -- simple server for queue*/

#include <stdio.h>
#include "q.h"

int main(void)
{
    int pid;

    switch(pid = fork()) {
      case 0: /*child*/
          serve();
          break; /*actually, serve never exits */
      case -1:
          perror("fork to start sever failed");
          break;
      default:
          printf("server process pid is %d\n", pid);
    }
    exit(pid == -1 ? 1 : 0);
}
```

Example #3: Sample Run

```
dhlee@kde:~/Course/SP/example>etest objname1 3
dhlee@kde:~/Course/SP/example>etest objname2 4
dhlee@kde:~/Course/SP/example>etest objname3 1
dhlee@kde:~/Course/SP/example>etest objname4 9
dhlee@kde:~/Course/SP/example>stest

priority: 1 name: objname3

priority: 3 name: objname1

priority: 4 name: objname2

priority: 9 name: objname4
server process pid is 22496
```

msgctl (1)

- #include <sys/types.h>

  #include <sys/ipc.h>

  #include <sys/msg.h>

  int msgctl(int *id*, int *cmd*, struct msqid_ds **buf*);

  - msgctl allows process to set or query the status of the message queue *id*, or to remove the queue, according to the value of *cmd*.

msgctl (2)

– The structure msqid_ds is defined as follows:
  - struct msqid_ds {

```
struct ipc_perm msg_perm; /*permission struct */
ushort  msg_qnum; /*number of message on q*/
ushort  msg_qbytes; /*max number of bytes on q*/
ushort  msg_lspid; /*pid of last msgsnd operation*/
ushort  msg_lrpid; /*pid of last msgrcv operation*/
time_t  msg_stime; /*last msgsnd time*/
time_t  msg_rtime; /*last msgrcv time*/
time_t  msg_ctime; /*last change time*/
};
```

msgctl (3)

- The commands (*cmd*) and their meaning are as follows:
  - IPC_STAT
    - Read the message queue header associated with *id* into *buf*.
  - IPC_SET
    - Set the values of msg_perm.uid, msg_perm.gid, msg_perm.mode (9 low-order bits), and msg_qbytes from the corresponding values in *buf*.
  - IPC_RMID
    - Remove the message queue for *id*.

Example #4: Message (1)

```
/* showmsg -- show message queue details */

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <time.h>
#include <stdio.h>

void mqstat_print(int mqid, struct msqid_ds *mstat)
{
    printf("\nmsg_qid %d\n\n", mkey, mqid);
    printf("%d message(s) on queue\n\n", mstat->msg_qnum);
    printf("Last send by proc %d at %s\n", mstat->msg_lspid, ctime(&(mstat-
    >msg_stime)));
    printf("Last recv by proc %d at %s\n", mstat->msg_lrpid, ctime(&(mstat-
    >msg_stime)));
}
```

Example #4: Message (2)

```
int main(int argc, char *argv[])
{
    int msq_id;
    struct msqid_ds msq_status;

    if(argc != 2) {
        fprintf(stderr, "usage:showmsg keyval\n");
        exit(1);
    }

    /*get message queue identifier*/
    msq_id = atoi(argv[1]);

    /*get status information*/
    if (msgctl(msq_id, IPC_STAT, &msq_status) < 0) {
        perror("msgctl failed");
        exit(3);
    }
```

Example #4: Message (3)

```
    /*print out status infromation*/
    mqstat_print(msq_id, &msq_status);
    exit(0);
}
```

Example #4: Sample Run

```
dhlee@kde:~/Course/SP/example>ipcs

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 65537      root       777        393216     2          dest
0x00000000 98306      root       644        106496     3          dest
0x00000000 131075     root       644        106496     3          dest

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

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x00000068 163840     dhlee      660        0            0

dhlee@kde:~/Course/SP/example>showmsg 163840

msg_qid 163840

0 message(s) on queue

Last send by proc 22494 at Tue Nov 16 11:27:35 2004

Last recv by proc 22496 at Tue Nov 16 11:27:35 2004

dhlee@kde:~/Course/SP/example>ipcrm msg 163840
msgrcv failed: Identifier removed
resource(s) deleted
```