Part 02. 리눅스 시스템 프로그래밍

Chapter 15. IPC(4)

진행 순서

Chapter 15_01	메시지 큐 개요
Chapter 15_02	SysV 메시지 큐
Chapter 15_03	POSIX 메시지 큐

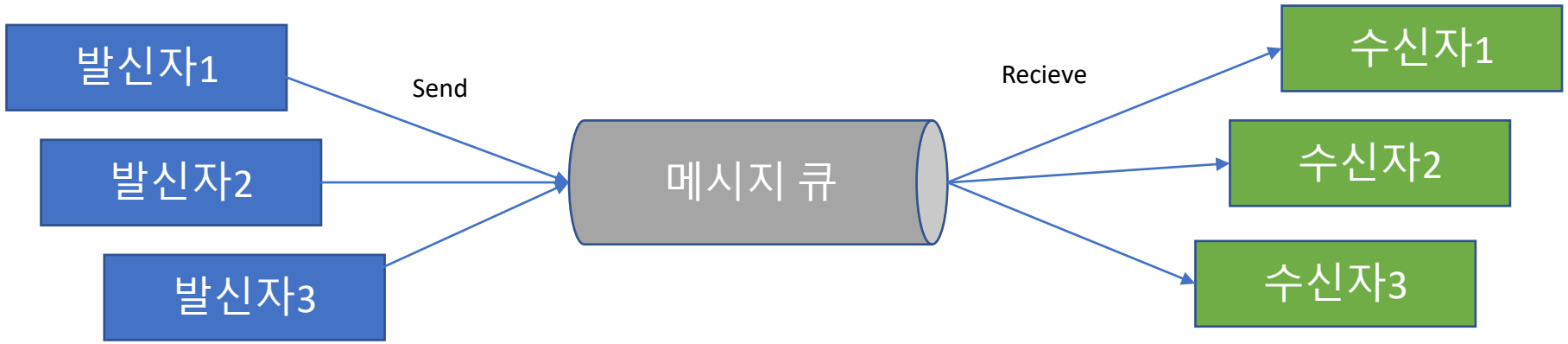
Chapter 15_01 메시지 큐 개요

메시지 큐란?

메시지 큐는 컴퓨터 과학에서 광범위하게 프로세스 간 통신 (IPC) 또는 동일한 프로세스 내의 스레드 간 통신에 사용되는 소프트웨어 엔지니어링 구성 요소입니다.

메시지 큐 패러다임은 게시자(publisher) / 구독자(subscriber) 패턴의 한 분류이며 일반적으로 더 큰 메시지 지향 미들웨어 시스템의 일부입니다. 대부분의 메시징 시스템은 API에서 게시자 / 구독자 및 메시지 큐 모델을 모두 지원합니다.

메시지 큐는 비동기 통신 프로토콜을 제공하므로 메시지의 발신자와 수신자가 동시에 메시지 큐와 상호 작용할 필요가 없습니다. 메시지 큐에 배치 된 메시지는 수신자가 메시지를 검색 할 때까지 저장됩니다. 메시지 큐에는 단일 메시지로 전송 될 수 있는 데이터 크기와 큐에서 처리되지 않은 메시지 수에 대한 암시 적 또는 명시 적 제한이 있습니다.



Chapter 15_02 SysV 메시지 큐

SysV 메시지 큐 주요 함수

msgget 메시지 큐의 IPC ID를 획득
msgsnd 메시지 큐에 데이터 송신
msgrcv 메시지 큐에서 데이터 수신
msgctl 메시지 큐 제어(제거 및 설정)

Chapter 15_02 SysV 메시지 큐

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

int msgget(key_t key, int msgflg);
```

공유메모리 shmget 나 세마포어 semget 함수와 유사
key는 고정된 정수형 키 값이나 IPC_PRIVATE을 인자로 받으며,
msgflg에는 옵션 플래그와 접근 권한을 지정할 수 있습니다. (ex. IPC_CREAT|644)

옵션 플래그
IPC_CREAT IPC 자원이 존재하지 않으면 생성
IPC_EXCL IPC 자원이 이미 존재하면 에러(EEXIST) 발생

성공하면 리턴 값은 메시지 큐 ID (음이 아닌 정수)이고, 그렇지 않으면 -1 리턴

ex) int msgid = msgget((key_t)1234, 0666 | IPC_CREAT);

Chapter 15_02 SysV 메시지 큐

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

int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

msgctl()은 ID가 msqid 인 System V 메시지 큐에서 cmd로 지정된 제어 조작을 수행합니다.

msqid_ds 자료구조는 <sys/msg.h> 에 선언되어 있으며

```
struct msqid_ds {
   struct ipc_perm msg_perm;     /* Ownership and permissions */
   time_t          msg_stime;    /* Time of last msgsnd(2) */
   time_t          msg_rtime;    /* Time of last msgrcv(2) */
   time_t          msg_ctime;    /* Time of last change */
   unsigned long   __msg_cbytes; /* Current number of bytes in queue (nonstandard) */
   msgqnum_t       msg_qnum;     /* Current number of messages in queue */
   msglen_t        msg_qbytes;   /* Maximum number of bytes allowed in queue */
   pid_t           msg_lspid;    /* PID of last msgsnd(2) */
   pid_t           msg_lrpid;    /* PID of last msgrcv(2) */
};
```

Chapter 15_02 SysV 메시지 큐

msgctl() cmd 종류

IPC_STAT - msqid와 관련된 커널 데이터 구조에서 buf가 가리키는 msqid_ds 자료구조로 정보 복사.
호출자는 메시지 큐에 대한 읽기 권한이 있어야 합니다.

IPC_SET - buf가 가리키는 msqid_ds 구조의 일부 멤버 값을 이 메시지 큐와 연관된 커널 데이터 구조에 쓰고
msg_ctime 멤버도 업데이트, 호출자는 소유자이거나 쓰기 권한이 있어야 합니다.

IPC_RMID - 메시지 큐를 즉시 제거, 대기중인 모든 reader 및 writer 프로세스 오류 리턴(errno EIDRM).
호출 프로세스는 적절한 권한을 가지고 있어야 합니다. msgctl ()의 세 번째 인수 buf는 무시

msgctl()은 성공하면 IPC_STAT, IPC_SET 및 IPC_RMID는 0을 리턴합니다.
오류가 발생하면 오류를 나타내는 errno와 함께 -1이 리턴됩니다.

ex)

```
int msgid = msgget((key_t)1234, 0666 | IPC_CREAT);
…
int result = msgctl(msgid, IPC_RMID, 0);
if (result == -1) {
   /* error */
}
```

Chapter 15_02 SysV 메시지 큐

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

int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);

ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp, int msgflg);
```

msgsnd() 및 msgrcv() 시스템 호출은 각각 System V 메시지 큐와 메시지를 주고받는 데 사용됩니다. 메시지를 보내려면 호출 프로세스에 메시지 큐에 대한 쓰기 권한이 있어야 하며 메시지를 받으려면 읽기 권한이 있어야 합니다.

msqid는 IPC ID 값, msgp는 송수신할 데이터 버퍼, msgsz는 msgp의 크기, msgtyp은 수신 받을 메시지 타입을 의미합니다.
msgp 인수는 다음과 같은 호출자 정의 구조체에 대한 포인터입니다.

```
struct msgbuf {
   long mtype;      /* message type, must be > 0 */
   char mtext[1];    /* message data */
};
```

mtype은 수신 프로세스에서 메시지 선택을 위해 사용할 수 있습니다.
mtext는 메시지 내용을 나타내는데, 실제로 1개의 char 배열이 아니라 저런 구조로 만든다는 것을 의미
(실제로는 메시지가 들어갈 충분한 크기의 배열로 선언)

Chapter 15_02 SysV 메시지 큐

msgsnd(), msgrcv()의 msgflg 플래그

IPC_NOWAIT
: msgsnd, msgrcv에서 요청 된 유형의 메시지가 큐에 없으면 즉시 에러 리턴 (errno ENOMSG)

MSG_NOERROR
: msgrcv를 통해 수신 받은 메시지가 msgsz 바이트 보다 긴 경우 텍스트를 자름

MSG_EXCEPT
: msgrcv를 통해 수신 시 인수 msgtyp(>0)과 다른 메시지 유형의 큐에서 수신

Chapter 15_02 SysV 메시지 큐

msgop(2) mag-page 예제 코드 분석

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

struct msgbuf {
   long mtype;
   char mtext[80];
};
```

```
static void
usage(char *prog_name, char *msg)
{
   if (msg != NULL)
      fputs(msg, stderr);

   fprintf(stderr, "Usage: %s [options]\n", prog_name);
   fprintf(stderr, "Options are:\n");
   fprintf(stderr, "-s       send message using msgsnd()\n");
   fprintf(stderr, "-r       read message using msgrcv()\n");
   fprintf(stderr, "-t       message type (default is 1)\n");
   fprintf(stderr, "-k       message queue key (default is 1234)\n");
   exit(EXIT_FAILURE);
}
```

Chapter 15_02 SysV 메시지 큐

```
static void
send_msg(int qid, int msgtype)
{
   struct msgbuf msg;
   time_t t;

   msg.mtype = msgtype;

   time(&t);
   snprintf(msg.mtext, sizeof(msg.mtext), "a message at %s",
                ctime(&t));

   if (msgsnd(qid, (void *) &msg, sizeof(msg.mtext),
                IPC_NOWAIT) == -1) {
      perror("msgsnd error");
      exit(EXIT_FAILURE);
   }
   printf("sent: %s\n", msg.mtext);
}
```

```
static void
get_msg(int qid, int msgtype)
{
   struct msgbuf msg;

   if (msgrcv(qid, (void *) &msg, sizeof(msg.mtext), msgtype,
                MSG_NOERROR | IPC_NOWAIT) == -1) {
      if (errno != ENOMSG) {
         perror("msgrcv");
         exit(EXIT_FAILURE);
      }
      printf("No message available for msgrcv()\n");
   } else
      printf("message received: %s\n", msg.mtext);
}
```

Chapter 15_02 SysV 메시지 큐

```
int main(int argc, char *argv[]) {
   int qid, opt;
   int mode = 0;              /* 1 = send, 2 = receive */
   int msgtype = 1;
   int msgkey = 1234;

   while ((opt = getopt(argc, argv, "srt:k:")) != -1) {
      switch (opt) {
      case 's':
         mode = 1;
         break;
      case 'r':
         mode = 2;
         break;
      case 't':
         msgtype = atoi(optarg);
         if (msgtype <= 0)
            usage(argv[0], "-t option must be greater than 0\n");
         break;
      case 'k':
         msgkey = atoi(optarg);
         break;
      default:
         usage(argv[0], "Unrecognized option\n");
      }
   }

   if (mode == 0)
      usage(argv[0], "must use either -s or -r option\n");

   qid = msgget(msgkey, IPC_CREAT | 0666);

   if (qid == -1) {
      perror("msgget");
      exit(EXIT_FAILURE);
   }

   if (mode == 2)
      get_msg(qid, msgtype);
   else
      send_msg(qid, msgtype);

   exit(EXIT_SUCCESS);
}
```

Chapter 15_02 SysV 메시지 큐

```
[parallels@localhost ch15]$ gcc -g sysv_mq_example.c -o sysv_mq_example

[parallels@localhost ch15]$ ./sysv_mq_example
must use either -s or -r option
Usage: ./sysv_mq_example [options]
Options are:
-s      send message using msgsnd()
-r      read message using msgrcv()
-t      message type (default is 1)
-k      message queue key (default is 1234)

[parallels@localhost ch15]$ ./sysv_mq_example -s -t 1 -k 1234
sent: a message at Fri Jun  5 22:46:37 2020

[parallels@localhost ch15]$ ./sysv_mq_example -r -t 1 -k 1234
message received: a message at Fri Jun  5 22:46:37 2020
```

Chapter 15_03 POSIX 메시지 큐

SysV 와 POSIX 방식의 차이 복습

POSIX 방식에서는 파일 기술자(file descriptor) 형식을 사용하는 통일된 형태를 사용.
POSIX는 가상화된 모든 자원을 기술자(descriptor)로 식별하는 방식을 사용하기 때문에 모든 입출력 형태가 마치 파일에 입출력하는 방식과 유사하다. (직관적인 인터페이스)

이에 비해 SysV 방식은 키(key)를 이용해서 ID 값을 얻어내고 접근하는 방식
유사하지만 IPC 각각의 사용법을 익여야 함.

POSIX 메시지 큐의 차이

POSIX 메시지 큐에는 이벤트 감지 기능이 추가되어 있음
SysV 방식에서 msgsnd, msgrcv 함수를 사용해 송수신할 때 메시지가 도착했는지 알 수 없기 때문에 기본적으로 블록킹 모드에서 대기하는 형태로 구현을 한다.
하지만 POSIX 방식에서는 메시지가 도착했을 때 이벤트를 감지할 수 있다. (mq_notify)
그리고 이벤트가 발생하면 비동기적으로 실행할 구문을 등록할 수 있다.

Chapter 15_03 POSIX 메시지 큐

POSIX 메시지 큐 주요 함수

mq_open	메시지 큐 객체를 획득
mq_close	메시지 큐를 닫음
mq_unlink	메시지 큐를 시스템에서 제거
mq_send	메시지 큐에 데이터 송신
mq_timedsend	타임아웃 송신
mq_receive	메시지 큐로부터 데이터 수신
mq_timedreceive	타임아웃 수신
mq_setattr	메시지 큐 속성 설정
mq_getattr	메시지 큐 속성 읽기
mq_notify	메시지 큐에 데이터 도착 시 통지 기능 (시그널, 콜백 스레드)

Chapter 15_03 POSIX 메시지 큐

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

mqd_t mq_open(const char *name, int oflag);
mqd_t mq_open(const char *name, int oflag, mode_t mode, struct mq_attr *attr);
```

Link with -lrt.

mq_open()은 새 POSIX 메시지 큐를 작성하거나 기존 큐를 엽니다. 메시지 큐는 name으로 식별됩니다.
oflag 인수는 호출 작업을 제어하는 플래그를 지정합니다.
(O_RDONLY / O_WRONLY / O_RDWR / O_CREAT / O_EXCL / O_NONBLOCK)

O_CREAT가 oflag에 지정된 경우 두 개의 추가 인수를 제공해야 합니다.
mode 인수는 open (2)와 같이 새 메시지 큐에 배치 할 권한을 지정합니다. (ex. 0600)

attr을 가리키는 struct mq_attr의 필드는 큐가 허용 할 최대 메시지 수와 최대 메시지 크기를 지정합니다.
이 구조체는 다음과 같이 정의됩니다. (NULL 일 경우 시스템 설정 값 사용)

```
struct mq_attr {
                long mq_flags;      /* Flags (ignored for mq_open()) */
                long mq_maxmsg;     /* Max. # of messages on queue */
                long mq_msgsize;    /* Max. message size (bytes) */
                long mq_curmsgs;    /* # of messages currently in queue (ignored for mq_open()) */
};
```

Chapter 15_03 POSIX 메시지 큐

```
#include <mqueue.h>

int mq_close(mqd_t mqdes);

int mq_unlink(const char *name);
```

Link with -lrt.

mq_close ()는 메시지 큐 디스크립터 mqdes를 닫습니다.
호출 프로세스가 mqdes를 통해 알림 요청 ((mq_notify (3) 참조)을이 메시지 큐에 첨부 한 경우 이 요청이 제거되고 다른 프로세스가 이제 알림 요청을 첨부 할 수 있습니다.

mq_unlink ()는 지정된 메시지 큐 이름을 제거합니다. 메시지 큐 이름이 즉시 제거됩니다.
큐가 열려있는 다른 프로세스가 큐를 참조하는 디스크립터를 닫으면 큐 자체가 삭제됩니다.

Chapter 15_03 POSIX 메시지 큐

```
#include <mqueue.h>

int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len, unsigned int msg_prio);

#include <time.h>
#include <mqueue.h>

int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len, unsigned int msg_prio,
                 const struct timespec *abs_timeout);

Link with -lrt.
```

mq_send()는 msg_ptr이 가리키는 메시지를 메시지 큐 디스크립터 mqdes가 참조하는 메시지 큐에 추가합니다. msg_len 인수는 msg_ptr이 가리키는 메시지 길이를 지정합니다. 이 길이는 큐의 mq_msgsize 속성보다 작거나 같아야 합니다. 길이가 0 인 메시지가 허용됩니다.

msg_prio 인수는이 메시지의 우선 순위를 지정하는 음이 아닌 정수입니다. 우선 순위가 낮은 순서로 메시지가 큐에 배치되고 우선 순위가 동일한 이전 메시지 뒤에 동일한 우선 순위의 새 메시지가 배치됩니다.

메시지 큐가 이미 가득 찬 경우 (즉, 큐의 메시지 수가 큐의 mq_maxmsg 속성과 동일), 기본적으로 mq_send()는 메시지를 큐에 넣을 수 있는 충분한 공간이 확보 되거나 시그널 핸들러에 의해 인터럽트가 호출될 때까지 차단됩니다. 메시지 큐 설명에 O_NONBLOCK 플래그가 사용 가능한 경우 대신 오류 EAGAIN과 함께 호출이 실패합니다.

mq_timedsend()의 경우 timeout 값 설정 외에 mq_send()와 동일

Chapter 15_03 POSIX 메시지 큐

```
#include <mqueue.h>

ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len, unsigned int *msg_prio);

#include <time.h>
#include <mqueue.h>

ssize_t mq_timedreceive(mqd_t mqdes, char *msg_ptr, size_t msg_len, unsigned int *msg_prio,
                        const struct timespec *abs_timeout);
```

Link with -lrt.

mq_receive()는 메시지 큐 디스크립터 mqdes가 참조하는 메시지 큐에서 우선 순위가 가장 높은 가장 오래된 메시지를 제거하고 msg_ptr이 가리키는 버퍼에 배치합니다. msg_len 인수는 msg_ptr이 가리키는 버퍼 크기를 지정합니다. 큐의 mq_msgsize 속성보다 크거나 같아야 합니다 (mq_getattr (3) 참조).

msg_prio가 NULL이 아닌 경우, 해당 버퍼가 수신 한 메시지와 연관된 우선 순위를 리턴하는 데 사용됩니다.

큐가 비어 있으면 기본적으로 mq_receive()는 메시지가 사용 가능해 지거나 시그널 핸들러에 의해 호출이 인터럽트 될 때까지 차단됩니다. 메시지 큐에 O_NONBLOCK 플래그가 사용된 경우 EAGAIN 오류와 함께 호출이 실패합니다.

mq_timedreceive()의 경우 timeout 값 설정 외에 mq_receive()와 동일

Chapter 15_03 POSIX 메시지 큐

```
#include <mqueue.h>

int mq_getattr(mqd_t mqdes, struct mq_attr *attr);

int mq_setattr(mqd_t mqdes, const struct mq_attr *newattr, struct mq_attr *oldattr);

Link with -lrt.
```

mq_getattr() 및 mq_setattr()은 각각 메시지 큐 디스크립터 mqdes가 참조하는 메시지 큐의 속성을 검색하고 수정합니다.

```
struct mq_attr {
                long mq_flags;      /* Flags: 0 or O_NONBLOCK */
                long mq_maxmsg;     /* Max. # of messages on queue */
                long mq_msgsize;    /* Max. message size (bytes) */
                long mq_curmsgs;    /* # of messages currently in queue */
};
```

Chapter 15_03 POSIX 메시지 큐

#include <mqueue.h>

int mq_notify(mqd_t mqdes, const struct sigevent *sevp);

Link with -lrt.

mq_notify()를 사용하면 새 메시지가 메시지 큐 디스크립터 mqdes가 참조하는 빈 메시지 큐에 도착할 때 호출 프로세스가 비동기 알림을 전달하기 위해 등록 또는 등록 취소 할 수 있습니다.

sevp 인수는 sigevent 구조에 대한 포인터입니다.

```
struct sigevent {
   int        sigev_notify; /* Notification method */
   int        sigev_signo;  /* Notification signal */
   union sigval sigev_value;  /* Data passed with notification */
   void       (*sigev_notify_function) (union sigval); /* Function used for thread notification (SIGEV_THREAD) */
   void       *sigev_notify_attributes; /* Attributes for notification thread (SIGEV_THREAD) */
   pid_t      sigev_notify_thread_id; /* ID of thread to signal (SIGEV_THREAD_ID) */
};
```

sevp가 널이 아닌 포인터 인 경우 mq_notify ()는 호출 프로세스를 등록하여 메시지 알림을 수신합니다. sevp가 가리키는 sigevent 구조의 sigev_notify 필드는 알림 수행 방법을 지정합니다. 이 필드는 다음 값 중 하나입니다.
SIGEV_NONE - 호출 프로세스가 알림 대상으로 등록되지만 메시지가 도착하면 알림이 전송되지 않습니다.
SIGEV_SIGNAL - sigev_signo에 지정된 신호를 보내 프로세스에 알립니다.
SIGEV_THREAD - 메시지 전달시 sigev_notify_function을 마치 새 스레드의 시작 함수 인 것처럼 호출

Chapter 15_03 POSIX 메시지 큐

posix_mq_receiver.c

```
#include <pthread.h>
#include <mqueue.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

#include <string.h>

#define handle_error(msg) \
   do { perror(msg); exit(EXIT_FAILURE); } while (0)
```

```
static void tfunc(union sigval sv)
{
   struct mq_attr attr;
   ssize_t nr;
   void *buf;
   mqd_t mqdes = *((mqd_t *) sv.sival_ptr);

   if (mq_getattr(mqdes, &attr) == -1)
      handle_error("mq_getattr");

   buf = malloc(attr.mq_msgsize);
   if (buf == NULL)
      handle_error("malloc");

   nr = mq_receive(mqdes, buf, attr.mq_msgsize, NULL);
   if (nr == -1)
      handle_error("mq_receive");

   printf("Read %zd bytes from MQ\n", nr);
   printf("message: %s\n", buf);
   free(buf);
   exit(EXIT_SUCCESS);
}
```

Chapter 15_03 POSIX 메시지 큐

posix_mq_receiver.c

```
int main(int argc, char *argv[])
{
    mqd_t mqdes;
    struct mq_attr attr;
    struct sigevent sev;

    if (argc != 2) {
        fprintf(stderr, "Usage: %s <mq-name>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    if (argv[1][0] != '/') {
        fprintf(stderr, "mq-name(%s) is must start with
slash(/)\n", argv[1]);
        exit(EXIT_FAILURE);
    }

    memset(&attr, 0, sizeof attr);
    attr.mq_maxmsg = 2;
    attr.mq_msgsize = 1024;
```

```
    mqdes = mq_open(argv[1], O_CREAT|O_RDONLY, 0644, &attr);
    if (mqdes == (mqd_t) -1)
        handle_error("mq_open");

    sev.sigev_notify = SIGEV_THREAD;
    sev.sigev_notify_function = tfunc;
    sev.sigev_notify_attributes = NULL;
    sev.sigev_value.sival_ptr = &mqdes;   /* for trunc argument */
    if (mq_notify(mqdes, &sev) == -1)
        handle_error("mq_notify");

    pause();
}
```

Chapter 15_03 POSIX 메시지 큐

posix_mq_sender.c

```
#include <pthread.h>
#include <mqueue.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#define handle_error(msg) \
   do { perror(msg); exit(EXIT_FAILURE); } while (0)

int
main(int argc, char *argv[])
{
   mqd_t mqdes;
   ssize_t nr;
   char *buf = "Hello, World!";

   if (argc != 2) {
      fprintf(stderr, "Usage: %s <mq-name>\n", argv[0]);
      exit(EXIT_FAILURE);
   }

   if (argv[1][0] != '/') {
      fprintf(stderr, "mq-name(%s) is must start with slash(/)\n", argv[1]);
      exit(EXIT_FAILURE);
   }

   mqdes = mq_open(argv[1], O_WRONLY);
   if (mqdes == (mqd_t) -1)
      handle_error("mq_open");

   nr = mq_send (mqdes, buf, strlen(buf)+1, 0);
   if (nr == -1)
      handle_error("mq_send");

   exit(EXIT_SUCCESS);
}
```

Chapter 15_03 POSIX 메시지 큐

실행결과

```
[root@localhost ch15]# gcc -g posix_mq_receiver.c -o posix_mq_receiver -lrt

[root@localhost ch15]# ./posix_mq_receiver
Usage: ./posix_mq_receiver <mq-name>

[root@localhost ch15]# ./posix_mq_receiver testmq
mq-name(testmq) is must start with slash(/)

[root@localhost ch15]# ./posix_mq_receiver /testmq

Read 14 bytes from MQ
message: Hello, World!
```

```
[root@localhost ch15]# gcc -g posix_mq_sender.c -o posix_mq_sender -lrt

[root@localhost ch15]# ./posix_mq_sender
Usage: ./posix_mq_sender <mq-name>

[root@localhost ch15]# ./posix_mq_sender testmq
mq-name(testmq) is must start with slash(/)

[root@localhost ch15]# ./posix_mq_sender /testmq
```