운영체제 과제 – 3장(2)

※ magics 시스템을 사용할 때에 로그인 직후에 다음과 같이 프롬프트를 변경한 후 사용하며 동작 결과를 보여줄 때에 프롬프트도 함께 보여주시오.

```
$ PS1='\u:\W \!$ ‘
```

1. **(IPC)** (1) 참고 프로그램의 프로그램1을 분석하고 실행시켜보시오.
   (2) 참고 프로그램의 프로그램2-1과 2-2를 분석하고 실행시켜보시오.
   (3) 그리고 이 프로그램과 각종 자료들(과목 홈페이지 게시 자료, 인터넷 검색자료 포함)을 참고하여 POSIX의 shared memory와 message queue를 사용한 interprocess communication 기능에 대한 API들을 소개하고 두 방법을 비교 설명하시오.

2. **(간단한 shell 작성)** UNIX/Linux 운영체제 환경에서 명령어를 입력받아서 실행시키는 기능을 갖는 프로그램 mysh을 작성하시오. (힌트: mysh에서 fork와 execvp를 사용하여 입력받은 명령어를 실행시킨다. 명령어의 인수가 가변개수이므로 execlp보다 execvp를 사용하는 것이 편리하다, exec계열 시스템 호출 사용법은 [process] 참고자료 및 인터넷 자료를 참조하시오.) 이 shell은 exit 명령어 또는 Ctrl-D를 입력받으면 종료한다. (동작 확인을 할 때에 다양한 Linux 명령어를 입력하여 실행해보시오.)

   $ **mysh**
   mysh> *command arg1 ...* mysh>은 프롬프트임, 명령어를 입력받음
   command 실행 입력 명령어를 실행함
   mysh> ls -l
   mysh> date
   ...
   mysh> exit 또는 Ctrl-D exit 또는 Ctrl-D가 입력되면 mysh을 종료함
   $

3. **(pipe)** Unix/Linux pipe와 관련하여 교과서 3장의 Figure 3-25 프로그램을 작성하여 실행시키고, 이 프로그램을 분석하여 Unix/Linux에서의 pipe 기능의 사용법에 대해서 알아보시오.

〈참고 프로그램〉

```
--- 프로그램 1. UNIX/Linux Shared memory를 사용하는 프로그램 ---
/* Simple program demonstrating shared memory in POSIX systems. */
#include <stdio.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main()
{
        int pid;                /* the identifier for the shared memory segment */
        int segment_id;         /* a pointer to the shared memory segment */
        char* shared_memory;    /* the size of the shared memory segment (byte) */
        const int segment_size = 4096;

        /** allocate  a shared memory segment */
        segment_id = shmget(IPC_PRIVATE, segment_size, S_IRUSR | S_IWUSR);
        printf("create shared memory : segment_id = %d\n", segment_id);

        /** attach the shared memory segment */
```

```
        shared_memory = (char *) shmat(segment_id, NULL, 0);

        pid = fork();
        if (pid < 0)
                return -1;
        else if (pid == 0) {  /* child */
                /** write a message to the shared memory segment   */
                sprintf(shared_memory, "Hello Parent");
                /** now detach the shared memory segment */
                if ( shmdt(shared_memory) == -1) {
                        fprintf(stderr, "Unable to detach\n");
                }
        } else {  /* parent */
                wait(NULL);
                /** now print out the string from shared memory */
                printf("*%s*\n", shared_memory);
                /** now detach the shared memory segment */
                if ( shmdt(shared_memory) == -1) {
                        fprintf(stderr, "Unable to detach\n");
                }

                /** now remove the shared memory segment */
                shmctl(segment_id, IPC_RMID, NULL);
        }

        return 0;
}
```

--- 프로그램 2-1. POSIX message queue를 사용하는 프로그램 (message send) ---

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

#define MSGSZ    128

/* Declare the message structure. */
typedef struct msgbuf
{
        long mtype;
        char mtext[MSGSZ];
} message_buf;

int main(void)
{
        int msqid;
        int msgflg = IPC_CREAT | 0666;
        key_t key;
        message_buf sbuf;
        size_t buf_length;

        /* Get the message queue id for the "name" 1234,
         * which was created by the server.  */
        key = 1234;
        fprintf (stderr, "\nmsgget: Calling msgget(%#lx,%#o)\n", key, msgflg);
        if ((msqid = msgget (key, msgflg)) < 0) {
                perror ("msgget");
                exit (1);
        } else
                fprintf (stderr, "msgget: msgget succeeded: msqid = %d\n", msqid);

        /* We'll send message type 1 */
        sbuf.mtype = 1;
        strcpy (sbuf.mtext, "Did you get this?");
        buf_length = strlen (sbuf.mtext) + 1;

        /* Send a message. */
```

```
        if (msgsnd (msqid, &sbuf, buf_length, IPC_NOWAIT) < 0) {
                printf ("%d, %d, %s, %d\n", msqid, sbuf.mtype, sbuf.mtext, buf_length);
                perror ("msgsnd");
                exit (1);
        } else
                printf ("Message: \"%s\" Sent\n", sbuf.mtext);

        exit (0);
}
```

--- 프로그램 2-2. POSIX message queue를 사용하는 프로그램 (message receive) ---

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

#define MSGSZ    128

/* Declare the message structure. */
typedef struct msgbuf
{
        long mtype;
        char mtext[MSGSZ];
} message_buf;

int main (void)
{
        int msqid;
        key_t key;
        message_buf rbuf;

        /* Get the message queue id for the "name" 1234,
         * which was created by the server. */
        key = 1234;

        if ((msqid = msgget (key, 0666)) < 0) {
                perror ("msgget");
                exit (1);
        }
        /* Receive an answer of message type 1. */
        if (msgrcv (msqid, &rbuf, MSGSZ, 1, 0) < 0) {
                perror ("msgrcv");
                exit (1);
        }
        /* Print the answer. */
        printf ("Received Message: %s\n", rbuf.mtext);
        exit (0);
}
```

--- Fig 3.25 UNIX 파이프사용 프로그램

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <string.h>

#define BUFFER_SIZE 25
#define READ_END       0
#define WRITE_END      1

int main(void)
{
       char write_msg[BUFFER_SIZE] = "Greetings";
       char read_msg[BUFFER_SIZE];
       pid_t pid;
       int fd[2];

       /* create the pipe */
       if (pipe(fd) == -1) {
```

```
		fprintf(stderr,"Pipe failed");
		return 1;
	}

	/* now fork a child process */
	pid = fork();

	if (pid < 0) {
		fprintf(stderr, "Fork failed");
		return 1;
	}

	if (pid > 0) {  /* parent process */
		/* close the unused end of the pipe */
		close(fd[READ_END]);

		/* write to the pipe */
		write(fd[WRITE_END], write_msg, strlen(write_msg)+1);

		/* close the write end of the pipe */
		close(fd[WRITE_END]);
	}
	else { /* child process */
		/* close the unused end of the pipe */
		close(fd[WRITE_END]);

		/* read from the pipe */
		read(fd[READ_END], read_msg, BUFFER_SIZE);
		printf("child read %s\n",read_msg);

		/* close the write end of the pipe */
		close(fd[READ_END]);
	}

	return 0;
}
```