

System Programming

Hyun-Wook Jin
System Software Laboratory
Department of Computer Science & Engineering
Konkuk University
jinh@konkuk.ac.kr

목차

- 리눅스 사용법 참고
- 프로세스 복제, fork()
- 프로세스 대기, wait()
- 프로세스 대체, exec()
- 실습 문제 및 챌린징
- IPC(Inter-process Communication)
- POSIX Message Queue
- 챌링징

리눅스 사용법 참고

- 매뉴얼 페이지 확인

- 명령어

- man <page name>

- man <section number> <page name>

- EX) man fork, man 2 fork

- 섹션

- 1. general commands

- 2. system calls

- 3. c library functions

- 4. special files and drivers

- 5. file formats and conventions

- 6. games and screensavers

- 7. miscellanea

- 8. system administration commands and daemons

리눅스 사용법 참고

- Foreground & Background 실행

```
int main()
{
    sleep(10);
    printf("Hello\n");
    return 0;
}
```

- Foreground 실행

- ./test

- \$ vim test.c (vim 대신 nano도 가능)

- Background 실행

- ./test &

- \$ nano test.c & (vim 대신 nano도 가능)

프로세스 복제, fork()

헤더 및 함수 원형	<pre>#include <sys/types.h> #include <unistd.h> pid_t fork(void);</pre>
함수 설명	현재 프로세스를 복제하여 동일한 프로세스를 생성
반환값	성공 시 : 부모 프로세스에게는 자식 프로세스의 pid, 자식 프로세스에게는 0이 반환됨 실패 시 : -1

- 일반적으로는 실행 파일을 통해 프로세스를 생성
- fork()를 사용하면 현재 프로세스를 복제하여 새로운 프로세스를 생성할 수 있음

프로세스 복제, fork()

- 예제

```
1 #include <stdio.h>
2 #include <unistd.h>
3
4 void main(void){
5     pid_t pid ;
6
7     printf("Before fork : %d \n", getpid());
8
9     pid = fork();
10
11     if( pid == 0){
12         printf("child process id : %d \n", getpid());
13         printf("pid :: %d \n", pid);
14     }
15     else if(pid > 0){
16         printf("parent process id : %d \n", getpid());
17         printf("pid :: %d \n", pid );
18     }
19     else{
20         printf("Fail to fork : %d \n", pid);
21     }
22 }
```

프로세스 대기, wait()

헤더 및 함수 원형	<pre>#include <sys/types.h> #include <sys/wait.h> pid_t wait(int *status);</pre>
함수 설명	자식 프로세스가 종료되어 종료 상태를 가져올 때까지 대기
인자값	int *status : 자식 프로세스의 종료 상태 정보가 저장될 변수
반환값	성공 시 : 종료된 자식 프로세스의 pid 실패 시 : -1

- 정상종료 시 status의 하위 8비트에는 0이 저장되며 상위 8비트에는 프로세스를 종료시킨 exit() 함수의 인자가 저장됨
- 비정상 종료 시 status의 하위 8비트에는 프로세스를 종료시킨 시그널의 번호가 저장되며 상위 8비트에는 0이 저장됨

프로세스 대기, wait()

- 예제

```
#include <stdio.h>
#include <sys/wait.h>

void main(void){
    pid_t pid, child;
    int data;
    int state;

    data = 10;

    pid = fork();

    if( pid < 0 ){
        printf("Fail to fork\n");
    }
    else{
        printf("Success to fork: %d\n", pid);
    }
}
```

```
    if( pid == 0 ){
        data += 10;
    }
    else{
        data -= 10;
        child = wait(&state);

        printf("Child pid: %d\n", child);
    }

    printf("\tdata: %d\n", data);

    return;
}
```


프로세스 대기, wait()

- 상태 정보 확인 매크로

매크로	매크로 반환값
WIFEXITED(status)	자식 프로세스가 정상적으로 종료되었다면 TRUE
WIFSIGNALED(status)	자식 프로세스가 시그널에 의해 종료되었다면 TRUE
WIFSTOPPED(status)	자식 프로세스가 중단되었다면 TRUE
WEXITSTATUS(status)	자식 프로세스가 정상 종료되었을 때 반환한 값

특정 프로세스 대기, waitpid()

헤더 및 함수 원형	<pre>#include <sys/types.h> #include <sys/wait.h> pid_t waitpid(pid_t pid, int *status, int options);</pre>
함수 설명	특정 자식 프로세스가 종료되어 종료 상태를 가져올 때까지 대기
인자값	pid_t pid : 특정 자식 프로세스의 pid int *status : 자식 프로세스의 종료 상태 정보가 저장될 변수 int options : 종료될 때까지 대기할지 등의 옵션
반환값	성공 시 : 종료된 자식 프로세스의 pid 실패 시 : -1

- wait()은 여러 자식 프로세스가 존재하더라도 하나의 자식 프로세스가 종료되면 호출됨
- 부모 프로세스가 대기를 하지 않고 해당 시점에서 자식 프로세스의 종료 여부만 확인하기를 원할 경우 options 인자에 WNOHANG을 넘기면 됨

특정 프로세스 대기, waitpid()

- 예제

```
#include <stdio.h>
#include <sys/wait.h>

int main(void){
    pid_t pid, child;
    int state;

    pid = fork();

    if( pid < 0 ){
        printf("Fail to fork\n");
    }
    else{
        printf("Success to fork: %d\n", pid);
    }

    if( pid == 0 ){
        printf("Child Process\n");
        sleep(1);
        return 2;
    }
    else{
        printf("Parent Process: wait for %d\n", pid);
        waitpid(pid, &state, 0);
        printf("Success to exit: %d\n", WEXITSTATUS(state));
    }

    return 0;
}
```

프로세스 대체, `exec1()`

헤더 및 함수 원형	<pre>#include <sys/types.h> #include <sys/wait.h> int exec1(const char *path, const char *arg, ...);</pre>
함수 설명	다른 프로그램을 실행하고 자신은 종료
인자값	<code>const char *path</code> : 실행할 파일의 상대 경로 또는 절대 경로 <code>const char *arg, ...</code> : 실행할 파일에게 전달될 인자들
반환값	실패 시 : -1

- `const char *arg, ...`의 마지막 항목은 NULL 포인터여야 함
- 프로그램 실행 시 첫번째 인자는 실행한 프로그램의 이름



프로세스 대체, `exec1()`

- 예제

- `whereis` 명령어를 통해 프로그램의 경로를 확인할 수 있음
 - `$ whereis ls`

```
#include <stdio.h>
#include <unistd.h>

void main(void){
    printf("Hello World!\n");

    execl("/bin/ls", "ls", "-l", NULL);

    printf("Goodbye World!\n");
}
```

프로세스 대체, exec() 계열

헤더 및 함수 원형	<pre>#include <sys/types.h> #include <sys/wait.h> int execl(const char *path, const char *arg, ...); int execlp(const char *file, const char *arg, ...); int execlx(const char *path, const char *arg, ..., char *const envp[]); int execv(const char *path, char *const argv[]); int execvp(const char *file, char *const argv[]);</pre>
인자값	<p>const char *path : 실행할 파일의 상대 경로 또는 절대 경로</p> <p>const char *file : 실행 파일의 이름</p> <p>const char *arg, ... : 실행할 파일에게 전달될 인자들</p> <p>char *const argv[] : 실행할 파일에게 전달될 인자들</p>
반환값	실패 시 : -1

- const char *file은 환경변수 PATH에 포함된 모든 경로의 프로그램을 실행하므로 실행할 파일의 이름만 전달해도 됨
- char *const argv[]의 마지막 항목은 NULL 문자열로 끝나야 함

실습 문제 1

- 아래 그림과 같이 동작이 되도록 코드 작성
 - 첫 번째 그림 처럼 작동이 된다면, 두 번째와 세 번째 그림 처럼 작동 되도록 수정

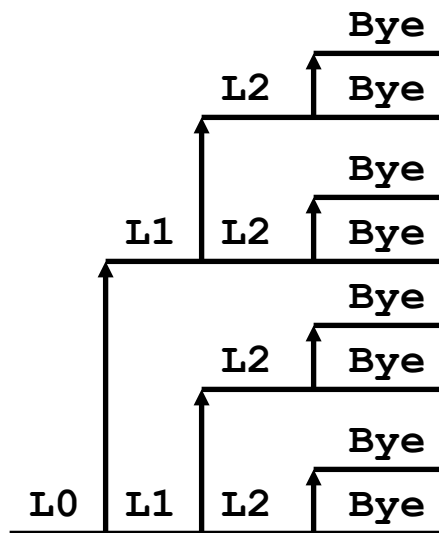


그림 1

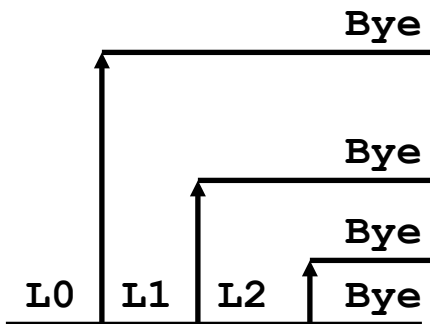


그림 2

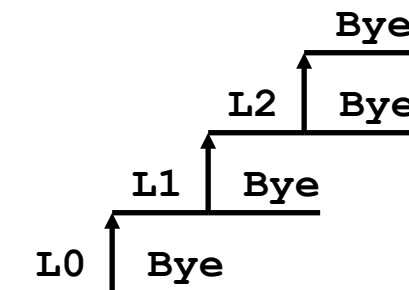


그림 3

실습 문제 2

- 부모와 자식 exec 실행
 - 부모 프로세스는 하나의 자식 프로세스를 생성한다
 - 부모 프로세스는 자신의 pid를 출력하고 자식 프로세스의 종료를 기다린 후 명령어 "ls -al" 실행
 - 자식 프로세스는 자신의 pid를 출력하고 명령어 "echo The test of system call"을 실행
 - `execl("/bin/echo", "echo", "The test of system call", NULL);`
 - 특정 위치에 있는 프로그램 실행
 - `execp("echo", "echo", "The test of system call", NULL);`
 - PATH에 등록된 경로의 프로그램 실행

Challenge #1

- MyShell 만들기

- 여러분이 사용하고 있는 터미널(셸)과 비슷한 역할을 수행하는 프로그램 제작
 - 사용자에게 명령을 받아서 프로그램을 실행
- 명령어를 입력하면, 명령어에 대한 프로그램을 실행하고 종료될 때 까지 대기
 - 예) \$ ls : 파일 명령 프로그램인 ls 수행
 - 예) \$ who : 현재 시스템에 접속 중인 사람 목록을 보여주는 who 명령어 실행
- 다음 장에 있는 베이스 코드에서 빈 박스 채워보기



Challenge #1

- MyShell 베이스 코드와 결과 화면


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

#define BUF_SIZE 255

int main(void) {
    char buf[BUF_SIZE] = {0, };

    for( ;; ) {
        pid_t child;
        int status;

        printf("MyShell:~$ ");
        scanf("%s", buf);

    }

    return 0;
}
```

```
jkpark@ubuntu:~/SP1$ ./shell
MyShell:~$ ls
code  code1.c  error  error.c  ex1  ex1.s  ex2  ex2.c  ex3  ex3.c  ex4.c  ex4.o
MyShell:~$ date
Mon Nov 27 05:51:51 PST 2017
MyShell:~$ ps
  PID TTY          TIME CMD
 2476 pts/0        00:00:00 bash
 19117 pts/0        00:00:00 shell
 19120 pts/0        00:00:00 ps
MyShell:~$ who
jkpark  :0                2017-11-06 22:42 (:0)
jkpark  pts/0          2017-11-06 22:44 (:0)
jkpark  pts/9            2017-11-08 05:59 (:0)
jkpark  pts/15            2017-11-08 06:03 (:0)
MyShell:~$ w
 05:52:28 up 16:33,  4 users,  load average: 0.08, 0.13, 0.10
USER    TTY      FROM          LOGIN@  IDLE   JCPU   PCPU WHAT
jkpark  :0       :0             06Nov17 ?xdm?  28:41  0.67s  init --user
jkpark  pts/0    :0             06Nov17 1.00s  0.79s  0.02s w
jkpark  pts/9    :0             08Nov17 18days 0.43s  0.36s  gdb ./error
jkpark  pts/15   :0             08Nov17 18days 0.03s  0.03s  bash
MyShell:~$
```

결과 화면

IPC

- IPC (Inter-Process Communication)
 - 내부 프로세스 혹은 스레드 간 통신을 위해서는 IPC 설비가 필요
 - Pipe
 - System-V IPC
 - POSIX IPC
 - Sockets
 - 서로 다른 호스트에 있는 프로세스 간의 IPC

POSIX Message Queue

- POSIX Portable Operating System Interface
 - 서로 다른 UNIX OS의 공통 API를 정리하여 이식성이 높은 유닉스 응용 프로그램을 개발하기 위한 목적으로 IEEE가 책정한 어플리케이션 인터페이스 규격

POSIX Message Queue

- mq_open

```
#include <fcntl.h>
#include <sys/stat.h>
#include <mq.h>
```

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

Arguments

const char * name : 다른 queue 와 구별되는 식별 인자
int oflag : 큐의 생성방식을 정의하기 위한 인자
mode_t mode : 권한 설정을 설정하기 위해 사용되는 인자
struct mq_attr : 메시지큐의 특성을 설정하기 위해 사용

Returns

메시지 큐를 가리키는 지정번호를 반환. 실패시 -1을 반환

- librt.a 정적 라이브러리 파일과 링킬해야 하며, 컴파일 시 -lrt 옵션을 추가하여 링크시킴 (/usr/lib)
- struct mq_attr

```
struct mq_attr {
{
    long mq_flags;          /* 0 or O_NONBLOCK */
    long mq_maxmsg;         /* 메시지 큐의 크기. 들어갈 수 있는 아이템의 수 */
    long mq_msgsize;        /* 메시지 크기 (바이트 단위) */
    long mq_curmsgs;        /* 현재 큐에 있는 메시지의 개수 */
}
```

POSIX Message Queue

- `mq_open`
 - `oflag`

oflag	의미
O_RDONLY	읽기 전용으로 열기
O_WRONLY	쓰기 전용으로 열기
O_RDWR	읽기, 쓰기 모두 가능
O_NONBLOCK	Nonblock 모드로 동작 (읽을 내용이 없을 때 기다리지 않고 바로 복귀)
O_CREAT	명시된 파일이 존재하지 않으면 파일을 생성 (생성시 접근권한 필요)

POSIX Message Queue

- mq_send

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

Arguments	
	mqd_t mqdes : 전달되는 메시지 queue const char * msg_ptr : 전달될 데이터의 위치를 지시 size_t msg_len : 메시지큐에 쓸 msg_ptr의 데이터 크기 unsigned msg_prio : 메시지의 우선순위
Returns	메시지 전송 성공시 0을 반환, 실패시 -1을 반환

- mq_receive

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

Arguments	
	mqd_t mqdes : 전달되는 메시지 queue const char * msg_ptr : 전달받을 데이터의 위치를 지시 size_t msg_len : 메시지큐에 받을 msg_ptr의 데이터 크기 unsigned msg_prio : 가져온 메시지의 우선순위를 복사
Returns	메시지 전송 성공시 읽은 메시지의 바이트 크기를 반환, 실패시 -1을 반환

POSIX Message Queue

- `mqd_t mq_close(mqd_t mqdes);`
 - 해당 메시지 큐의 `mqdes`를 닫음

<code>mqd_t mq_close(mqd_t mqdes);</code>	
<i>Arguments</i>	mqd_t mqdes : close 할 메시지 큐 디스크립터
<i>Returns</i>	성공시 0을 반환, 실패시 -1을 반환

- `mqd_t mq_unlink(const char * name);`
 - 지정된 메시지 큐 `name`을 제거함

<code>mqd_t mq_unlink(const char * name);</code>	
<i>Arguments</i>	const char * name : 제거할 메시지 queue name
<i>Returns</i>	성공시 0을 반환, 실패시 -1을 반환



POSIX Message Queue Example

- mq_sender.c

– gcc -o mq_sender mq_sender.c -lrt

```
#include <stdlib.h>
#include <fcntl.h>
#include <mqueue.h>
#include <stdio.h>

#define MSG_SIZE 4
#define NAME_POSIX "/my_mq"
#define MQ_PRIO_MAX 32
```

```
int main(int argc, char **argv)
{
```

```
    struct mq_attr attr;
    int value = 0;
    unsigned int prio;
    mqd_t mqdes;
```

```
    attr.mq_maxmsg = 10;
    attr.mq_msgsize = MSG_SIZE;
```

```
    mqdes = mq_open(NAME_POSIX, O_CREAT | O_WRONLY, 0666, &attr);
```

```
    if(mqdes == (mqd_t)-1)
```

```
    {
```

```
        perror("open error");
        exit(0);
    }
```

```
    for(prio = 0 ; prio <= MQ_PRIO_MAX ; prio += 8)
```

```
    {
```

```
        printf("Writing a message %d with priority %d.\n", value, prio);
```

```
        if(mq_send(mqdes, (char *)&value, MSG_SIZE, prio) == -1)
```

```
        {
```

```
            perror("mq_send()");
```

```
        }
```

```
        else
```

```
        {
```

```
            value += 3;
```

```
        }
```

```
    }
```

```
    mq_close(mqdes);
```

```
    return 0;
```

```
}
```



POSIX Message Queue Example

- `mq_receiver.c`
 - `gcc -o mq_receiver mq_receiver.c -lrt`

```
#include <stdlib.h>
#include <fcntl.h>
#include <mqueue.h>
#include <stdio.h>

#define MSG_SIZE 4
#define NAME_POSIX "/my_mq"

int main(int argc, char **argv)
{
    struct mq_attr attr;
    int value;
    unsigned int prio;
    mqd_t mqdes;

    attr.mq_maxmsg = 10;
    attr.mq_msgsize = MSG_SIZE;

    mqdes = mq_open(NAME_POSIX, O_RDWR | O_CREAT, 0666, &attr);
```

```
    if (mqdes == (mqd_t)-1)
    {
        perror("open error");
        exit(0);
    }

    while(mq_receive(mqdes, (char *)&value, MSG_SIZE, &prio) != -1)
    {
        printf("Received a message %d with priority %d.\n", value, prio);
        mq_getattr(mqdes, &attr);
        printf("%d messages are currently on the queue.\n", attr.mq_curmsgs);
        if(attr.mq_curmsgs == 0)
            break;
    }

    mq_close(mqdes);
    mq_unlink(NAME_POSIX);

    return 0;
```

```
}
```

POSIX Message Queue Example

- mq_fork.c
 - gcc -o mq_fork mq_fork.c -lrt

```
#include <stdlib.h>
#include <fcntl.h>
#include <mqueue.h>
#include <stdio.h>

#define NAME_POSIX "/my_mq"

int main(){
    struct mq_attr attr;
    int value = 0;
    unsigned int prio;
    int child = 0;
    mqd_t mqdes ;

    attr.mq_maxmsg = 10;
    attr.mq_msgsize = 8;

    mqdes = mq_open(NAME_POSIX, O_CREAT|O_RDWR, 0666, &attr) ;
    if(mqdes == (mqd_t)-1) perror("mqopen fail \n") ;

    child = fork();
    if(child == 0 ){
        printf("CHILD SEND \n");
        value = 1000;
        if(mq_send(mqdes, (char*)&value, 8, prio) == -1) perror("child send fail\n");
    }else{
        printf("PARENT RECV \n");
        if(mq_receive(mqdes, (char *)&value, 8, &prio) == -1) perror("parent recv fail\n");
        printf("received value : %d \n",value);
    }

    mq_close(mqdes);
    mq_unlink(NAME_POSIX);
}
```

Challenge #2

- PingPong 프로그램 만들기
 - 앞에서 만든 mq_fork.c 파일을 수정해서 핑퐁 프로그램으로 변경
 - 부모 프로세스에서 1초마다 "ping"을 자식 프로세스에 보내면, 자식 프로세스에서 "pong"을 부모 프로세스로 보내기
 - 1초마다 쉬는 것은 sleep() 함수 이용
 - 각 프로세스는 ping 또는 pong을 받았을 때 화면에 출력