시스템프로그래밍

총정리

내용

- 디버깅
- 파일 처리
- 디렉터리
- 프로세스
- 시그널
- IPC
- 쓰레드

GDB

- GDB (GNU DeBugger)
  - GNU 프로젝트의 디버거
  - C, C++, Objective-C, Modular 2, Ada 등의 언어 지원
  - 기능
    - 프로그램을 시작한다.
    - 프로그램이 특정 조건을 만족할 때 정지시킨다.
    - 프로그램이 정지했을 때 어떤 일이 벌어졌는지 조사한다.
    - 프로그램 일부를 수정하여 발견한 버그를 제거했을 때의 실행 결과를 미리 알아볼 수 있게 한다.
  - 요구사항
    - Gcc 컴파일 시에 "-g" 옵션을 사용하여 디버깅 정보를 오브젝트 코드에 삽입
    - 디버깅 수준에 따라 -g1, -g2, -g3 등과 같이 명시
      - 명시하지 않으면 기본적으로 –g2 옵션 사용
  - 공식 홈페이지 : http://www.gnu.org/software/gdb/

GDB 도움말

- Files 도움말
  - 작업 디렉토리 설정/검색, 실행 파일 지정 등의 파일과 관련된 명령어 모음
  - 잘 사용하는 명령어
    - cd: 디버거와 디버깅중인 프로그램에 대한 작업 디렉토리를 변경함.
    - directory: 소스 파일에 대한 탐색경로를 추가함.
    - file: 디버깅할 파일을 지정함.
    - list: 지정된 함수 또는 줄을 보여줌.
    - path: 목적 파일에 대한 탐색경로를 추가함.
    - pwd: 현재의 작업 디렉토리를 보여줌.
    - search: 정규 수식을 가장 최근에 보여진 줄 다음부터 탐색함.

GDB 도움말

- Running 도움말
  - 단계별 실행과 관련된 명령어 모음
  - 잘 사용하는 명령어
    - run: 디버깅할 프로그램을 시작함. run 명령어의 프로그램 인자는 디버깅할 프로그램의 프로그램 인자와 같은 역할을 함.
    - continue: 디버깅중인 프로그램의 진행을 계속함.
    - kill: 디버깅중인 프로그램의 실행을 정지함.
    - step: 다음 줄에 도달할 때까지 프로그램을 진행시킴. 인자 N은 이를 N번 반복함을 의미함. 함수 호출을 하게 되면 해당 함수로 제어가 넘어감.
    - next: step 명령어와 비슷하나, 함수 호출을 하나의 단위로 보아 함수 안을 디버깅할 수 없음. 올바르게 동작하는 함수의 경우 더 이상 디버깅이 필요없을 때 사용하면 편리함.

GDB 도움말

- Breakpoints 도움말
  - 프로그램을 특정 지점에서 정지하게 하기 위한 명령어 모음
  - 잘 사용하는 명령어
    - break: 소스 내의 특정 줄이나 특정 함수에 대한 정지점(breakpoint)을 설정할 때 사용함. break 명령어의 인자로 줄 번호가 지정되면 해당 줄 번호에 정지점이 설정되며, 인자로 함수 이름이 지정되면 함수의 시작 부분에 정지점이 설정됨.
    - watch: 변수 등에 대한 경계점(watchpoint)을 설정할 때 사용됨. 지정된 경계점의 값이 변경될 경우 프로그램 실행을 정지함.
    - clear: 특정 줄이나 특정 함수에 설정된 정지점을 해제할 때 사용함.
    - delete: 정지점 등을 해제할 때 사용함. clear와 달리 정지점의 번호를 인자로 사용함.

POSIX 표준 라이브러리 함수

- POSIX (Portable Operating System Interface) 표준
  - IEEE 에서 1988년에 UNIX 라이브러리를 위한 POSIX.1 발표
  - 1998년 The Open Group, IEEE POSIX, ISO/IEC 연합기술운영회의 회원들로 구성된 Austin 그룹 결성
  - 2001, 2002년 IEEE 표준 1003.1-2001 제정
  - 기타 표준 구성
    - POSIX.2 - 표준 쉘과 유틸리티 인터페이스
    - POSIX.4 - 쓰레드 관리
    - POSIX.5 - Ada 언어를 위한 바인딩
    - POSIX.6 - 보안 기능

표준 C 라이브러리 함수들(교재 p219참고)

- C 언어 표준 라이브러리

헤더 파일	함수 기능	표준 함수
stdio.h	표준입출력	printf(); scanf(); putchar(); getchar(); ...
	화일입출력	fopen(); fclose(); fprintf(); ...
stdlib.h	수치변환	atoi(); itoa(); ...
	난수발생	rand(); srand(); ...
	탐색 및 정렬	bsearch(); qsort();...
ctype.h	문자변환	tolower(); toupper(); ...
	문자판별	isalpha(); isdigit(); isupper(); ...
String.h	문자열처리	strcpy(); strlen(); strcmp(); ...
	동적메모리관리	memcpy(); memset(); memchr(); ...
Math.h	수학함수	sin(); cos(); sqrt(); ...
Search.h	탐색	lsearch(); hsearch(); tsearch(); ...
Time.h	시간관련함수	time(); difftime(); ctime(); ...

환경변수 처리

- 환경변수 처리
  - main() 함수의 확장형

```
main(int argc, char *argv[], char *envp[])
{
   프로그램 명령들
}
```

  - 많이 사용하는 환경변수

환경변수	설명
USER	로그인한 사용자 이름
LOGNAME	프로세스와 관련된 로그인 사용자 이름
HOME	사용자의 로그인 디렉토리
LANG	LC_ALL 등이 지정되지 않았을 때의 로케일 이름
LC_ALL	우선적으로 지정되는 로케일 이름. LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, LC_NUMERIC, LC_TIME 등을 포함한다.
PATH	실행 파일을 찾는 디렉토리들의 목록
PWD	현재 작업 디렉토리
SHELL	사용자의 로긴 쉘 파일 이름
TERM	출력 터미널 타입
TMPDIR	임시 디렉토리 경로
LD_LIBRARY_PATH	동적 로딩/링킹을 위한 라이브러리 경로

표준 입출력 함수

- ANSI C 표준 입출력 함수
  - 정수형 파일 기술자 대신 FILE 구조를 이용

기능	함수 원형
파일 열기/닫기	FILE *fopen(const char *path, const char *mode); int fclose(FILE *stream);

파일 읽기/쓰기	int fgetc(FILE *stream);
	int fputc(int c, FILE *stream);
	char *fgets(char *s, int size, FILE *stream);
	int fputs(const char *s, FILE *stream);
	int fscanf(FILE *stream, const char *format, …);
	int fprintf(FILE *stream, const char *format, …);

파일 위치 재배치	int fseek(FILE *stream, long offset, int whence);
	long ftell(FILE *stream);
	void rewind(FILE *stream);

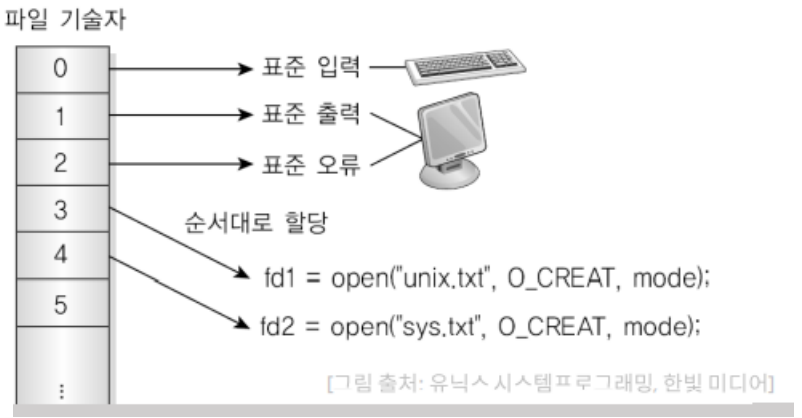
Low-level vs High-level file IO

- **Low-Level File IO (System call)**
  - System call을 이용해서 파일 입출력 수행
  - File descriptor 사용
  - Byte 단위로 디스크에 입출력
  - 특수 파일에 대한 입출력 가능

- **High-Level File IO (Buffered IO)**
  - C Standard library를 사용해서 파일 입출력 수행
  - File pointer 사용
  - 버퍼(block) 단위로 디스크에 입출력
    - 여러 형식의 입출력 지원

File descriptor

- **열려 있는 파일을 구분하는 정수(integer) 값**
  - 특수 파일 등 대부분의 파일을 지칭 가능
  - Process별로 kernel이 관리
- **파일을 열 때 순차적으로 할당 됨**
  - Process 당 최대 fd 수 = 1,024 (default, 변경 가능)
- **Default fds (수정 가능)**
  - 0 : stdin
  - 1 : stdout
  - 2 : stderr



Opening files – open(2)

- **flags** (Man page 및 <sys/fcntl.h> 참조)
  - 여러 플래그 조합 가능 (OR bit operation (|) 사용)

종류	기능
O_RDONLY	파일을 읽기 전용으로 연다.
O_WRONLY	파일을 쓰기 전용으로 연다.
O_RDWR	파일을 읽기와 쓰기가 가능하게 연다.
O_CREAT	파일이 없으면 파일을 생성한다
O_EXCL	O_CREAT 옵션과 함께 사용할 경우 기존에 없는 파일이면 파일을 생성하지만, 파일이 이미 있으면 파일을 생성하지 않고 오류 메시지를 출력한다.
O_APPEND	파일의 맨 끝에 내용을 추가한다.
O_TRUNC	파일을 생성할 때 이미 있는 파일이고 쓰기 옵션으로 열었으면 내용을 모두 지우고 파일의 길이를 0으로 변경한다.
O_SYNC/O_DSYNC	저장장치에 쓰기가 끝나야 쓰기 동작을 완료

Opening files – open(2)

- **mode**
  (Man page 및 <sys/stat.h> 참조)
  - 파일 권한 설정 값 사용 (예, 644)
    OR
  - 정의 된 플래그 사용
    - 조합하여 사용 가능
    - OR bit operation (|) 사용

플래그	모드	설명
S_IRWXU	0700	소유자 읽기/쓰기/실행 권한
S_IRUSR	0400	소유자 읽기 권한
S_IWUSR	0200	소유자 쓰기 권한
S_IXUSR	0100	소유자 실행 권한
S_IRWXG	0070	그룹 읽기/쓰기/실행 권한
S_IRGRP	0040	그룹 읽기 권한
S_IWGRP	0020	그룹 쓰기 권한
S_IXGRP	0010	그룹 실행 권한
S_IRWXO	0007	기타 사용자 읽기/쓰기/실행 권한
S_IROTH	0004	기타 사용자 읽기 권한
S_IWOTH	0002	기타 사용자 쓰기 권한
S_IXOTH	0001	기타 사용자 실행 권한

[그림 출처: 유닉스 시스템프로그래밍, 한빛 미디어]

저수준 파일 처리 함수

- 프로그램 예

```
/* 초보적인 프로그램 예 */

/* 이 헤더 파일들은 아래에서 논의한다 */
#include <fcntl.h>
#include <unistd.h>

main()
{
        int fd;
        ssize_t nread;
        char buf[1024];

        /* 파일 "data"를 읽기 위해 개방한다 */
        fd = open("data", O_RDONLY);

        /* 데이터를 읽어들인다 */
        nread = read(fd, buf, 1024);

        /* 파일을 폐쇄한다 */
        close(fd);
}
```

파일을 읽기 전용으로 개방

file descriptor

open 시스템 호출

- 기능
  - 기존의 파일을 읽거나 쓰기 전에 항상 파일 개방
- 사용법

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

int open(const char *pathname, int flags, [mode_t mode]);
```

- pathname : 개방될 파일의 경로
- Flags : 접근 방식 지정
  - **O_RDONLY** **읽기 전용으로 개방**
  - **O_WRONLY** **쓰기 전용으로 개방**
  - **O_RDWR** **읽기 및 쓰기 용으로 개방**
  - **아래 상수는 위의 상수와 OR 해서 사용**
  - **O_CREAT** **파일이 없으면 생성, 아래 mode 필요함**
  - **O_APPEND** **파일 쓰기 시 파일 끝에 추가**
  - **O_TRUNC** **파일이 이미 존재하고 쓰기 권한으로 열리면 크기를 0**
- Mode : 보안과 연관, 생략 가능

open 시스템 호출

- 사용 예

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

#define PERMS 0644    /* O_CREAT를 사용하는 open 을 위한 허가 */
char *workfile="junk";

main()
{
        int filedes;

        if ((filedes = open(workfile, O_RDWR | O_CREAT, PERMS)) == -1)
        {
                printf ("Couldn't open %s□n", workfile);
                exit (1);                       /* 오류이므로 퇴장한다 */
        }

        /* 프로그램의 나머지 부분이 뒤따른다 */

        exit (0);
}
```

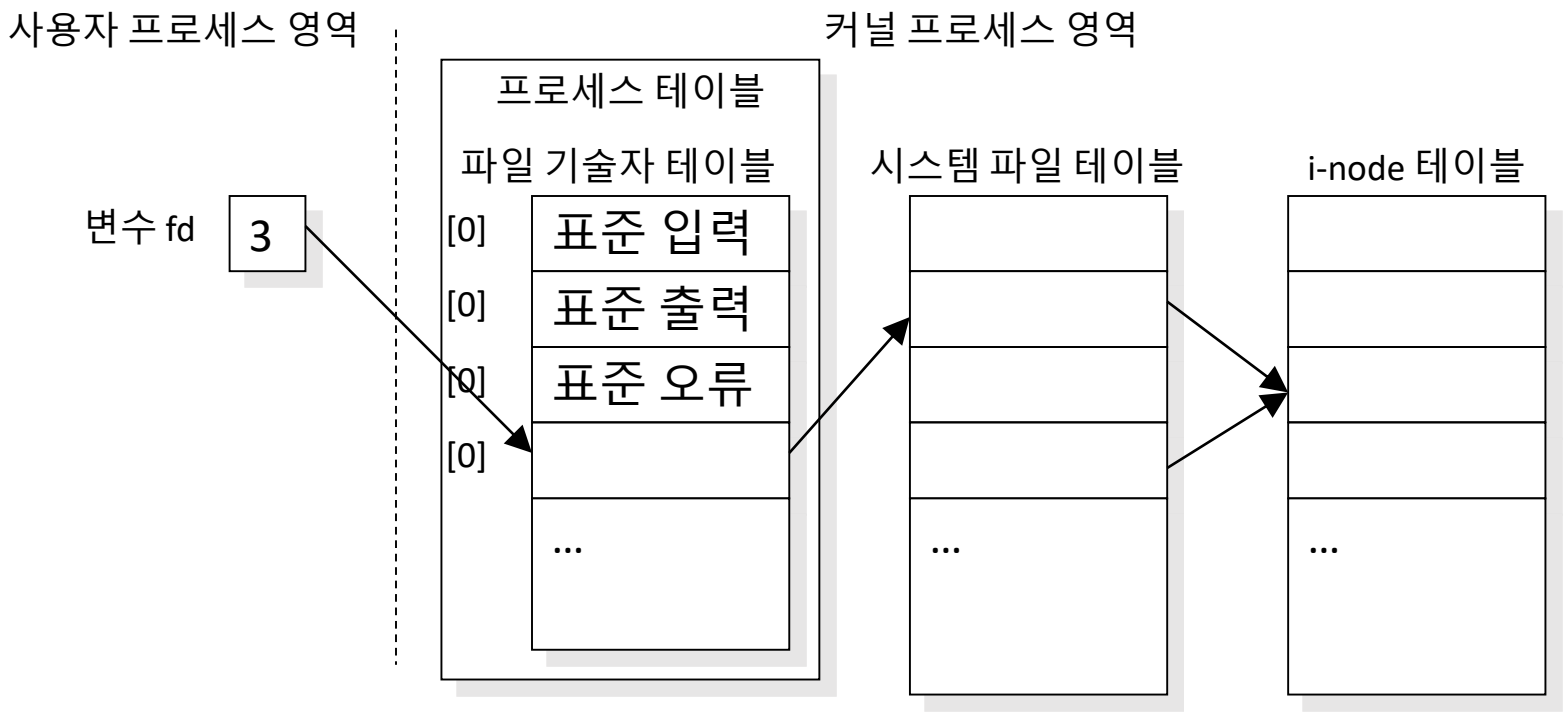
Opening files – open(2)

- **flag 조합의 예**
  - O_WRONLY | O_TRUNC
  - O_RDWR | O_APPEND

- **mode 조합의 예**
  - S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH

파일의 표현

- 시스템의 파일 관리
  - 파일 기술자 테이블, 시스템 파일 테이블, i-node 테이블을 통해 각 프로세스가 사용하는 파일 관리
    - 시스템 파일 테이블 – 시스템의 모든 파일에 대한 정보(상태, 현재 오프셋 등) 수록
    - i-node 테이블 – 실제 저장된 파일의 정보 수록



파일 상태

- LINUX stat 구조체
  - 파일 정보 저장
  - <bits/stat.h> 에 정의

```
struct stat {
        dev_t         st_dev;       /* device */
        ino_t         st_ino;       /* inode */
        mode_t        st_mode;      /* protection */
        nlink_t       st_nlink;     /* number of hard links */
        uid_t         st_uid;       /* user ID of owner */
        gid_t         st_gid;       /* group ID of owner */
        dev_t         st_rdev;      /* device type (if inode device) */
        off_t         st_size;      /* total size, in bytes */
        unsigned long  st_blksize;  /* blocksize for filesystem I/O */
        unsigned long  st_blocks;   /* number of blocks allocated */
        time_t        st_atime;     /* time of last access */
        time_t        st_mtime;     /* time of last modification */
        time_t        st_ctime;     /* time of last change */
};
```

- 파일 유형을 테스트하는 매크로들
  - <sys/stat.h> 에 정의

매크로	의 미
S_ISREG()	Regular file (일반적인 파일 형태)
S_ISDIR()	Directory file (다른 파일을 가지는 파일 형태)
S_ISCHR()	Character special file (문자 장치에 사용)
S_ISBLK()	Block special file (블록 장치에 사용)
S_ISFIFO()	FIFO pipe (프로세스간 IPC 에 사용)
S_ISLNK()	Symbolic link file (다른 파일에 대한 링크 파일 형태)
S_ISSOCK()	socket file (네트워크 통신에 사용)

기타 함수

- 파일 링크 관련 함수
  - 기능
    - 파일에 대한 링크와 관련된 동작 수행
  - 사용법

```
#include <unistd.h>

int link(const char *oldpath, const char *newpath);
int unlink(const char *pathname);
int symlink(const char *oldpath, const char *newpath);
```

    - link() : oldpath 파일에 대한 newpath 하드 링크를 생성
    - unlink() : 파일 시스템에서 pathname 파일 삭제. 파일이 심볼릭 링크이면 링크를 삭제
    - symlink() : oldpath 파일에 대한 newpath 심볼릭 링크를 생성

fcntl 함수

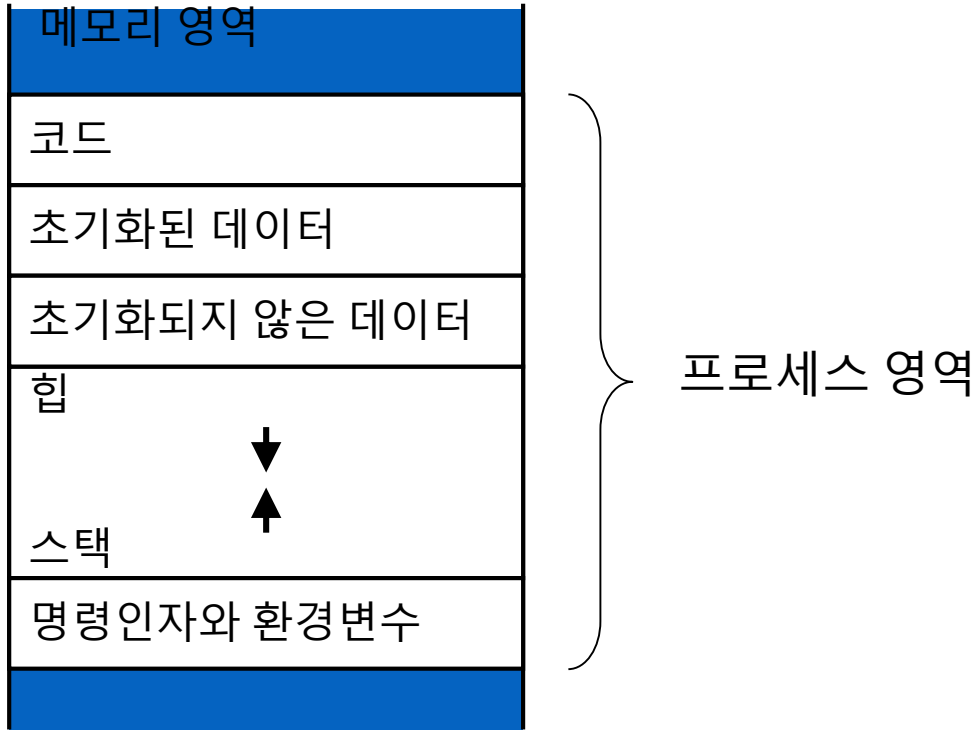
- 기능
  - 개방된 파일에 대한 제어
- 사용법

```
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>

/* 주의: 마지막 인수의 타입은 생략부호 "..." 에 의해 표시된 대로 가변적
*/
int fcntl(int filedes, int cmd, ...);
```

- filedes : 제어할 파일 기술자
- cmd : 제어에 대한 특정 기능
  - F_GETFL: open 에 의해 설정된 현재 파일의 상태 플래그 반환
  - F_SETFL: 파일에 연관된 상태 플래그를 재지정
- 세번째 인수부터는 인수 층 에 따라 좌우

프로세스 제어

- 프로그램 – 디스크 상에 저장되어 있는 실행 가능한 파일
- 프로세스 – 수행중인 프로그램
  - 쉘 은 하나의 명령을 수행하기 위해 어떤 프로그램을 시작할 때마다 새로운 프로세스를 생성



프로세스 개념

- 프로세스 – 수행중인 프로그램
  - 쉘 은 하나의 명령을 수행하기 위해 어떤 프로그램을 시작할 때마다 새로운 프로세스를 생성
    - Cat file1 file2 # 프로세스 생성
    - Ls | wc –l # 2 개의 프로세스 생성
  - UNIX 프로세스 환경은 디렉토리 트리처럼 계층적인 구조
    - 가장 최초의 프로세스는 init 이며 모든 시스템과 사용자 프로세스의 조상
  - 프로세스 시스템 호출들

이름	의 미
fork	호출 프로세스와 똑같은 새로운 프로세스를 생성
exec	한 프로세스의 기억공간을 새로운 프로그램으로 대체
wait	프로세스 동기화 제공. 연관된 다른 프로세스가 끝날 때까지 기다린다
exit	프로세스를 종료

프로세스 생성

- fork 시스템 호출
  - 기능
    - 기본 프로세스 생성 함수
    - 성공적으로 수행되면 호출 프로세스(부모 프로세스) 와 똑같은 새로운 프로세스(자식 프로세스) 생성
  - 사용법

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

pid_t fork(void);
```

  - 반환 값(return value)
    - 정상 실행
      - **실행(부모) 프로세스는 : 생성(자식) 프로세스의 프로세스 ID 반환**
      - **생성(자식) 프로세스는 : 0을 반환**
    - 이상 실행 : 음수 값을 반환
  - 프로세스 식별번호(IDentifier)

상속된 자료와 파일 기술자

- fork 수행 시 파일과 자료
  - 부모 프로세스로부터 자식 프로세스로 자료 복제
  - File
    - 부모 프로세스가 개방한 file 은 자식 프로세스에서도 개방
    - 파일 기술자만 복제
    - 파일 포인터는 공유 – 시스템에 의해 관리
      - **파일 포인터를 전진시키면 어느 프로세스에서든 전진됨**

좀비 프로세스

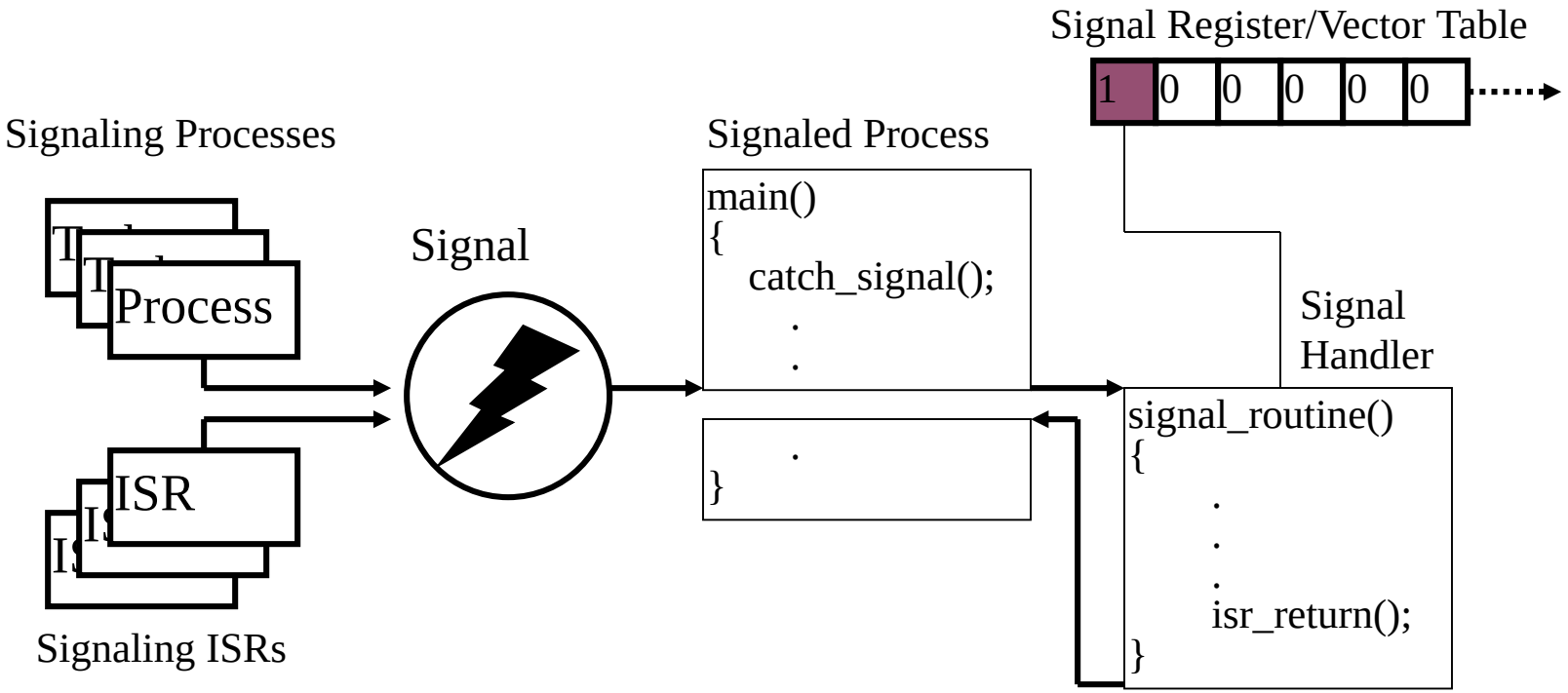
- 좀비(zombi) 와 너무 이른 퇴장
  - 부모 프로세스가 wait 를 수행하지 않고 있는 상태에서 자식이 퇴장할 때
    - 퇴장 프로세스는 좀비 프로세스가 됨
    - 부모 프로세스가 wait 를 수행할 때 삭제됨
  - 하나 이상의 자식 프로세스가 수행 중인 상태에서 부모가 퇴장할 때
    - 자식 프로세스는 init 프로세스(리눅스 최초 수행 프로세스)에게 맡겨짐

시그널

- 시그널
  - 다른 프로세스에게 이벤트 발생을 알리는 소프트웨어 인터럽트
  - 시그널 발생 예
    - 프로그램 실행 시 Ctrl-C(인터럽트 키) 를 눌러 강제 종료시킬 때
    - 백그라운드 작업을 종료시킬 때, kill 명령 사용

시그널

- 시그널 동작
  - 1. 기본적으로 설정한 동작 수행
  - 2. 시그널을 무시하고 프로그램 수행 계속
  - 3. 미리 정해진 일을 수행



시그널 처리

- sigaction 시스템 호출
  - 기능
    - 프로세스가 특정 시그널에 대해 다음 세 가지 행동 중 하나를 지정
      - 프로세스는 종료하고 코어 덤프
      - 시그널을 무시
      - 시그널 핸들러에 지정된 함수를 수행
  - 사용법

```
#include <signal.h>
int sigaction(int signo, const struct sigaction *act,
                         struct sigaction *oact);
```

    - signo: signal number
    - act : sigaction 구조체
    - oact: 이전 설정값

시그널 전송

- kill 시스템 호출
  - 기능
    - 다른 프로세스에게 특정 시그널 전송
  - 사용법

```
#include <sys/types.h>
#include <signal.h>

int kill(pid_t pid, int sig);
```

    - sig: 전송할 시그널
    - pid: 시그널 sig 를 받을 프로세스 ID
      - 양수: 해당 프로세스 식별번호를 가진 프로세스
      - 0: kill 을 호출한 프로세스와 그룹에 속하는 모든 프로세스에게 전송
      - -1: 1번 프로세스를 제외한 모든 프로세스에게 전송, 프로세스 유효사용자 ID가 수퍼유저가 아니면, 유효사용자 ID가 같은모든 프로세스에게 전송
      - 음수: 프로세스 그룹 식별번호가 pid 의 절대값과 같은 모든 프로세스에게 전송

시그널 전송

- raise 시스템 호출
  - 기능
    - 현재 프로세스에게 시그널 전송
  - 사용법

```
#include <signal.h>

int raise(int sig);
```

    - sig: 전송할 시그널

시그널 전송

- 시그널 전송 함수 사용 예
  - alarm 함수 사용 예제 프로그램

```
/* ex9_13.c */
/* alarm example */
#include <stdio.h>
#include <signal.h>
void alarm_handler(int);
int alarm_flag = 0;

main()
{
        struct sigaction act;

        act.sa_handler = alarm_handler;
        sigaction(SIGALRM, &act, NULL);

        alarm (5);       /* Turn alarm on. */
        pause(); /* pause */
        if (alarm_flag)
                printf("Passed a 5 secs.□n");
}
```

```
void alarm_handler(int sig)
{
        printf("Received a alarm signal.□n");
        alarm_flag = 1;
}
```

◈ 프로그램 실행 결과

```
[cprog2@seps5 signal]$ gcc ex9_13.c
[cprog2@seps5 signal]$ ./a.out
Received a alarm signal.
Passed a 5 secs.
[cprog2@seps5 signal]$
```

프로세스간 통신

- 프로세스간 통신 기법
  - 시그널, 파일 잠금, 파이프, 메시지 큐, 세마포어, 공유 메모리, 소켓 등

- 파일을 이용한 레코드 잠금
  - 레코드 잠금(record locking)
    - 프로세스가 특정 파일의 일부 레코드에 대하여 잠금 기능 설정
    - 다른 프로세스로 하여금 이 파일에 접근하지 못하도록 함
  - 종류
    - 읽기 잠금 – 다른 프로세스들이 해당 영역에 쓰기 잠금 불가
    - 쓰기 잠금 – 다른 프로세스들이 해당 영역에 읽기와 쓰기 잠금 모두 불가

표준 파이프 입출력 함수

- popen 과 pclose 함수
  - 기능
    - popen : pipe와 fork를 이용하여, 쉘 명령어의 입력 혹은 출력을 파이프와 연결한 다음 쉘 명령어를 실행
    - pclose : 사용하고 난 파이프와 명령 프로세스를 닫음
  - 사용법

```
#include <stdio.h>

FILE * popen(const char *command, const char *type);
int pclose(FILE *stream);
```

    - command :수행할 명령어를 나타내는 문자열의 포인터
    - type : pipe 의 읽기('r') 또는 쓰기('w') 방향
  - 반환값
    - popen 은 성공적인 호출에 대하여 파이프와 연관된 파일 기술자 반환
    - pclose 는 성공적인 호출에 대하여 0을 반환
    - 오류시에는 -1 이 반환되고, errno 가 적절히 설정

FIFO 를 이용한 프로그래밍

- 파이프의 단점
  - 프로그램 내에서는 부모와 자식 프로세스 간 데이터 전달 가능하지만, 프로세스가 종료되면 파이프도 삭제
  - 독립된 두 프로세스 사이에 파이프를 통해 데이터 전달 불가능
- ➔ FIFO 또는 명명 파이프(named pipe)
  - 특수한 형태의 파일
  - 입출력 처리 방식은 선입선출(first-in first-out) 방식
  - 파일 inode를 가지므로 영구적이고 임의의 프로세스가 접근 가능
- mkfifo 시스템 호출
  - 기능
    - FIFO 파이프 생성
  - 사용법

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char *pathname, mode_t mode);
```

    - pathname : 생성할 FIFO 파일의 경로 지정
    - mode : 이 파일의 허가권을 설정

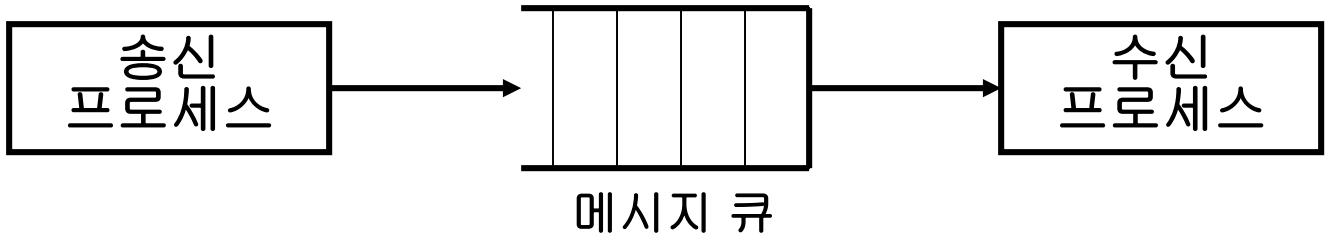
고급 프로세스간 통신

- 고급 IPC 개요
  - UNIX 와 LINUX 에서 지원하는 고급 IPC 자원 (System V IPC)
    1. 메시지 큐를 통하여 프로세스들 간에 메시지 전달
    2. 세마포어를 통하여 프로세스들 간의 동기화
    3. 프로세스 간 메모리 영역 공유

  - 자원 요청 시 동적으로 데이터 구조 생성
    - 명시적으로 삭제하지 않으면 시스템이 종료될 때까지 메모리에 존재
    - 독립적인 프로세스 간 통신에 사용 가능
  - IPC 자원의 구별
    - Key (32 bit)
      - **파일 경로와 비슷**
      - **프로그래머가 자유롭게 선택**
    - 식별자(32 bit)
      - **파일 기술자와 비슷**
      - **커널에 의해 자원에 부여되며, 시스템 내에서 고유한 값**

메시지 큐

- 메시지 큐
  - 프로세스 간에 문자나 바이트 열로 이루어진 메시지를 전달하기 위한 일종의 버퍼와 같은 큐
  - 메시지 큐와 파이프의 차이점
    - 파이프와 달리 메시지의 크기가 제한적
    - select 등을 사용할 수 없음
    - 우선순위 등에 따라 메시지 관리 가능



세마포어

- 세마포어
  - 프로세스 간 효율적인 동기화 방식
  - 열쇠와 비슷하여, 어떤 프로세스가 세마포어를 획득하면 공유 자원에 접근하거나 동작을 수행하도록 허용하고, 그렇지 않으면 세마포어가 해제되기를 기다리며 대기
  - 기능
    - 프로세스들 간의 공유 자원 접근 제공 (상호 배제 요구 조건 만족)
    - 파이프나 메시지 큐는 대규모 자료 전송에는 적합하지만 자원 낭비가 크므로 동기화 수단으로는 적합하지 않음

공유 메모리

- 공유 메모리
  - 둘 이상의 프로세스가 특정 메모리 영역을 공유하여 자료에 접근
  - 세 가지 고급 IPC 기법 중에서 가장 유용
  - 기능
    - 프로세스들 간의 자료 공유
- 공유 메모리 사용
  - IPC 공유 메모리 영역 (shared memory region) 에 대한 자료 구조
  - 프로세스 접근을 위해 프로세스 주소 공간에 공유 메모리 영역 추가
  - 사용이 끝나면 주소 공간에서 공유 메모리를 제거

쓰레드 개요

- 프로세스 – 중량 프로세스
- 쓰레드 – 경량 프로세스
  - 주소 공간 (프로그램, 데이터 영역) 공유
    - 프로세스 지시 사항, 대부분의 데이터, open 파일들, 시그널과 핸들러, 사용자와 그룹 ID 등
  - 스택, 쓰레드 문맥 (프로그램 카운터 등)만 따로 가짐
    - 쓰레드 ID, 프로그램 카운터 등의 문맥 정보, 스택, errno, 시그널 마스크, 우선순위
  - 프로세스에 비해 쓰레드 간 통신이 간편하고, 생성 시간도 짧은 장점을 가짐

쓰레드 분리와 결합

- pthread_detach 함수
  - 기능
    - 쓰레드를 분리하여 분리 상태로 둔다.
  - 사용법

```
#include <pthread.h>

int pthread_detach (pthread_t thread);
```

    - Thread: 분리할 쓰레드
  - 반환값
    - 성공적인 호출: 0, 그렇지 않을 경우: 0 이 아닌 값
  - 분리 상태(detached state)
    - 프로세스와 분리되어 쓰레드 종료 시 그동안 사용한 메모리를 즉시 해제하도록 보장
  - 또 다른 쓰레드 분리 방법
    - 쓰레드 생성 시 detachstate 속성을 지정

쓰레드 분리와 결합

- pthread_join 함수
  - 기능
    - 쓰레드의 종료를 기다린다.
  - 사용법

```
#include <pthread.h>


int pthread_join (pthread_t thread, void
**thread_return);
```

    - thread: 종료하기를 기다리는 쓰레드
    - Thread_return: 쓰레드 thread 의 반환값을 저장하는 장소
      - 쓰레드가 반환한 값 또는 쓰레드 취소의 경우 PTHREAD_CANCELED 값
  - 반환값
    - 성공적인 호출: 0, 그렇지 않을 경우: 0 이 아닌 값
  - 결합 가능한 상태(joinable state)
    - 쓰레드 종료 시 쓰레드 ID 와 스택과 같은 사용 메모리를 해제하지 않음. 다른 쓰레드가 pthread_join() 함수를 호출하거나 전체 프로세스가 종료되면 해제.
- 메모리 누출 방지를 위해 생성된 쓰레드는 pthread_detach 또는 pthread_join 을 한 번 호출

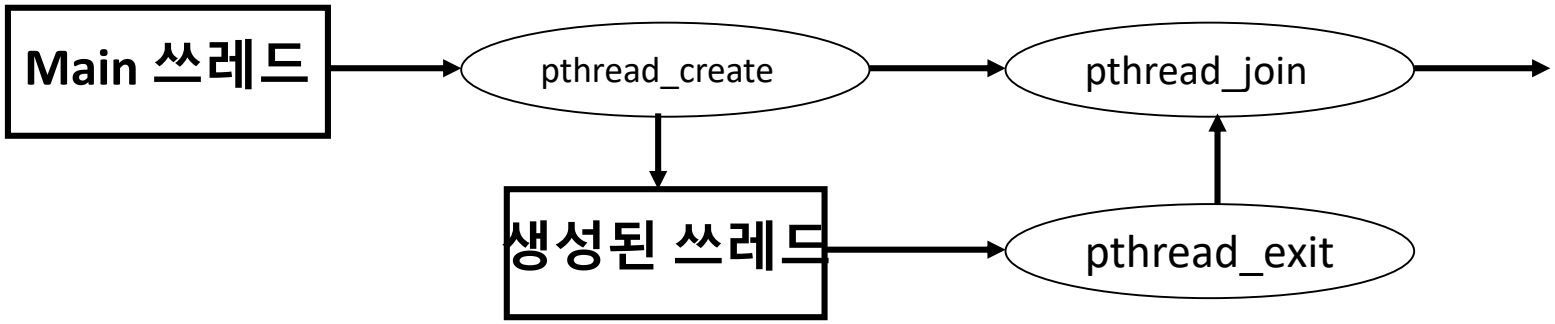
쓰레드 분리와 결합

- 쓰레드 결합 예
  - 쓰레드 결합 예제 프로그램

```
/* jointhread.c */
/* pthread join example */
#include <stdio.h>
#include <pthread.h>

void *join_thread (void *arg)
{
        pthread_exit(arg); /* return arg; */
}

int main(int argc, char *argv[])
{
        pthread_t tid;
        int arg, status;
        void *result;

        if (argc < 2) {
                fprintf (stderr, "Usage: jointhread
<number>□n");
                exit (1);
        }
        arg = atoi (argv[1]);

        /* 쓰레드 생성 */
        status = pthread_create (&tid, NULL,
join_thread, (void *)arg);
        if (status != 0) {
                fprintf (stderr, "Create thread: %d",
status);
                exit (1);
        }
        status = pthread_join (tid, &result);
        if (status != 0) {
                fprintf (stderr, "Join thread: %d", status);
                exit (1);
        }
        return (int) result;
}
```

쓰레드 분리와 결합

- 쓰레드 결합 예
  - 쓰레드 결합 예제 프로그램 동작 과정



  - 쓰레드 결합 예제 프로그램 실행 결과

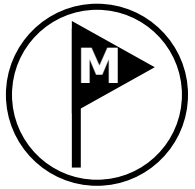
```
[cprog2@seps5 ch11]$ gcc -o jointhread jointhread.c -lpthread
[cprog2@seps5 ch11]$ ./jointhread
Usage: jointhread number
[cprog2@seps5 ch11]$ ./jointhread 0
[cprog2@seps5 ch11]$ echo $?
0
[cprog2@seps5 ch11]$ ./jointhread 2
[cprog2@seps5 ch11]$ echo $?
2
[cprog2@seps5 ch11]$
```

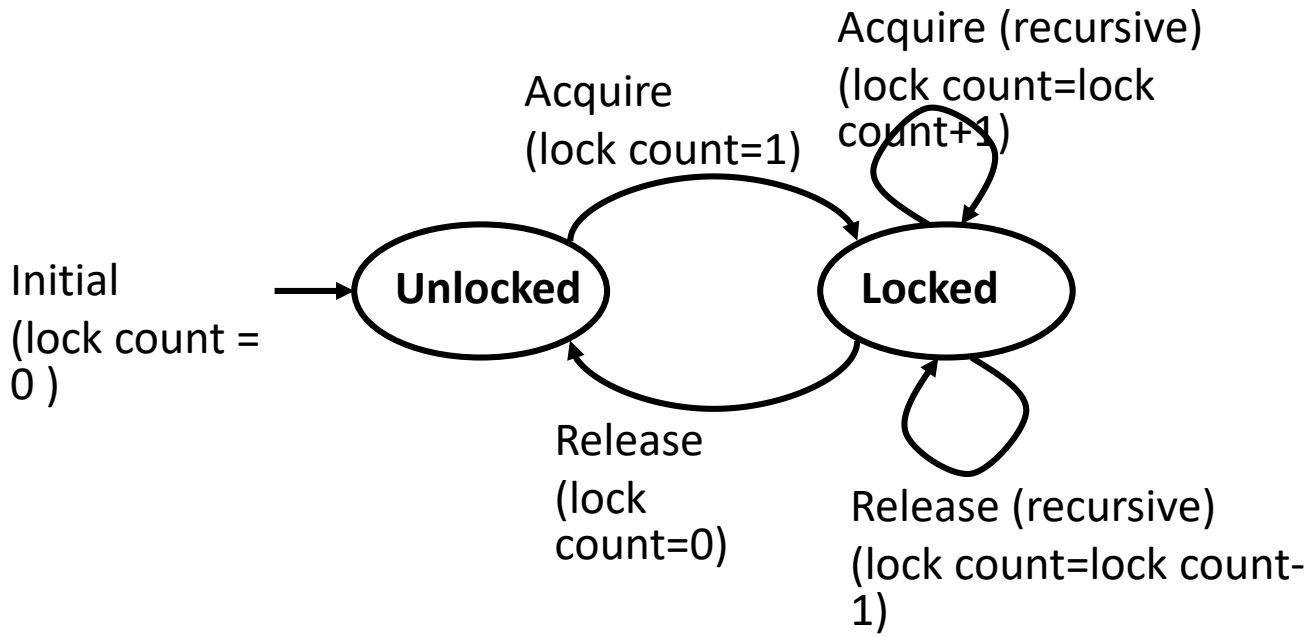
쓰레드 동기화

- 쓰레드 동기화
  - 프로세스와 마찬가지로 쓰레드들도 서로 동기화하거나 통신하여 더욱 효율적인 프로그램 수행 가능
- 쓰레드 동기화 방식
  - 뮤텍스 (mutex)
  - 조건변수 (condition variable)
  - 읽기-쓰기 잠금(read-write lock)
  - 시그널
  - 세마포어
  - 배리어 (barrier)
  - ….

뮤텍스

- 뮤텍스
  - Mutual Exclusion (Mutex) 의 약자
  - 공유하는 자원을 보호하기 위한 짧은 잠금(lock) 장치
  - 소유권, 재귀 잠금, 쓰레드 삭제 안전성, 우선순위 역전 문제 회피 기능 포함
  - 2 가지 상태를 가짐 : Unlocked(0) or locked(1)
- 세마포어와의 차이점
  - 공유 자원에 대해 상호배제적으로 접근
    - 계수 세마포어는 여러 개의 자원을 공유
  - 소유권을 가져 뮤텍스를 획득한 쓰레드가 뮤텍스를 해제
    - 이진 세마포어는 임의의 프로세스/쓰레드가 세마포어 해제 가능





실습파일

Open & Close a file

[code link]

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

int main(void) {
    int fd;
    mode_t mode;

    mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH; // 644

    fd = open("hello.txt", O_CREAT, mode);
    if (fd == -1) {
        perror("Creat"); exit(1);
    }
    close(fd);

    return 0;
}
```

$ ls hello.txt
ls: cannot access 'hello.txt': No such file or directory
$ vi fileOpenClose.c

$ gcc -o fileOpenClose.out fileOpenClose.c
$ ./ fileOpenClose.out
$ ls -l hello.txt
-rw-r--r-- 1 bluekds bluekds 0 Aug 29 17:24 hello.txt

open 시스템 호출

- 사용 예

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

#define PERMS 0644    /* O_CREAT를 사용하는 open 을 위한 허가 */
char *workfile="junk";

main()
{
        int filedes;

        if ((filedes = open(workfile, O_RDWR | O_CREAT, PERMS)) == -1)
        {
                printf ("Couldn't open %s□n", workfile);
                exit (1);                       /* 오류이므로 퇴장한다 */
        }

        /* 프로그램의 나머지 부분이 뒤따른다 */

        exit (0);
}
```

```
/*
 * 파일 이름: file_cat.c
 */

#include <stdio.h>
main(int argc, char *argv[])
{
    FILE *src;  /* source file      */
    char ch;
    src = fopen(argv[1], "r");
    while ( !feof(src) ) {
        ch = (char) fgetc(src);
        if ( ch != EOF )
            printf("%c", ch);
    }
    fclose(src);
}
```

Allocating file descriptor

[code link]

```c
int openFile(void) {
     int fd = open("hello.txt", O_RDWR);
     if (fd == -1) {
          perror("File Open");
          exit(1);
     }
     return fd;
}

int main(void) {
     int fd = 0;

     fd = openFile(); printf("fd = %d\n", fd);
     close(fd);

     close(0);

     fd = openFile(); printf("fd = %d\n", fd);
     close(fd);

    return 0;
}
```

간단한 쉘 만들기

- 간단한 쉘 프로그램

```
/* ex9_8.c */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

main()
{
        char buf[256];
        char *argv[50];
        int narg;
        pid_t pid;

        while (1) {
                printf("shell> ");
                gets(buf);
                narg = getargs(buf, argv);

                pid = fork();
                if (pid == 0)
                        execvp(argv[0], argv);
                else if (pid > 0)
                        wait((int *) 0);
                else
                        perror("fork failed");
        }
}

int getargs(char *cmd, char **argv)
{
        int narg = 0;

        while (*cmd) {
                if (*cmd == ' ' || *cmd == '□t')
                        *cmd++ = '□0';
                else {
                        argv[narg++] = cmd++;
                           while (*cmd != '□0' && *cmd != ' '
      && *cmd != '□t')
                                cmd++;
                }
        }
        argv[narg] = NULL;
        return narg;
}
```

```c
/* file_mkdir.c */
#include<stdio.h>
main(int argc, char *argv[])
{
    mkdir(argv[1], 0777);
}
```

```
[dongyang@localhost ch09]$ ./threadsam
thread2 : 1
thread1 : 2
thread2 : 3
thread1 : 4
thread2 : 5
thread1 : 6
thread2 : 7
thread1 : 8
^C
```

```c
/* file_ln.c */
#include<stdio.h>
main(int argc, char *argv[])
{
    link(argv[1], argv[2]);
}
```

```c
/* f_rm.c */
#include<stdio.h>
main(int argc, char *argv[])
{
    unlink(argv[1]);
}
```