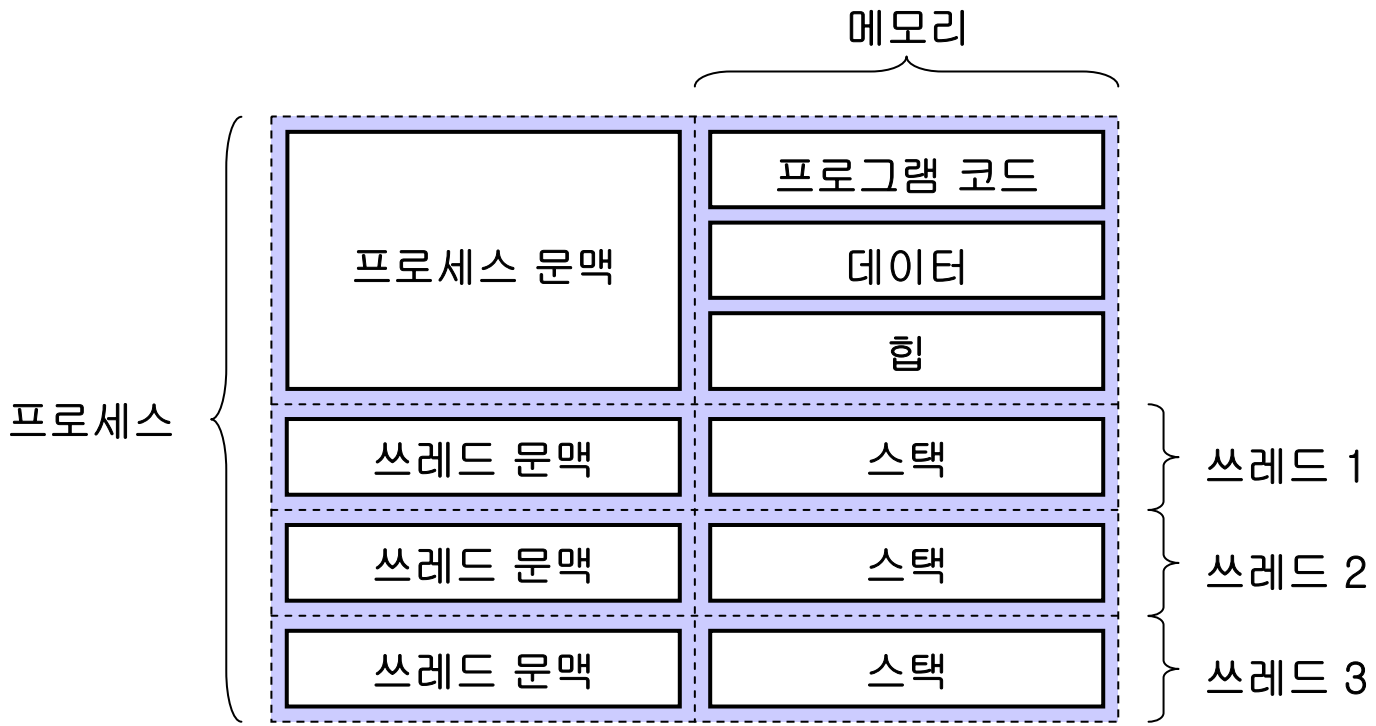
11장. 쓰레드 프로그래밍

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 개요

▣ 쓰레드와 프로세스의 관계



POSIX 쓰레드 개요

- **POSIX (Portable Operating System Interface)**
  - **UNIX** 기반 운영체제를 위한 포터블 인터페이스에 관한 국제 표준
- **POSIX Thread**
  - **POSIX** 표준 중 쓰레드에 관한 **API** 부분을 말함
    - **IEEE POSIX 1003.1c (1995)**
  - 구성 요소
    - 쓰레드 관리
    - 뮤텍스
    - 조건변수
  - 줄여서 **pthread** 라고도 함 - 모든 쓰레드 함수 이름이 **pthread_** 로 시작

접두사	의 미
pthread_	쓰레드와 기타 함수
pthread_attr_	쓰레드 속성 개체
pthread_mutex_	뮤텍스
pthread_mutexattr_	뮤텍스 속성 개체
pthread_cond_	조건변수
pthread_condattr_	조건변수 속성 개체
pthread_key_	쓰레드 특정-자료 키

쓰레드 ID 와 참조

- **POSIX** 쓰레드는 **pthread_t** 자료형의 쓰레드 **ID** 를 통해 참조됨
- 쓰레드 참조 함수들
  - 기능
    - **pthread_self() :** 자신의 쓰레드 **ID** 획득
    - **pthread_equal() :** 두 개의 쓰레드 **ID** 를 비교
  - 사용법

```
#include <pthread.h>

int pthread_self(void);
int pthread_equal(pthread_t t1, pthread_t t2);
```

    - **Pthread_self** 함수는 성공적인 호출에 대하여, 자신의 쓰레드 **ID**
      Pthread_equal 함수는 두 인자가 같으면 **0** 이 아닌 값, 다르면 **0**을 반환

쓰레드 생성과 종료

pthread_create 함수

기능

- 쓰레드 생성

사용법

```
#include <pthread.h>

int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start)(void *), void *arg);
```

- **thread:** 생성될 쓰레드의 **ID** 가 저장될 위치의 주소
- **attr:** 쓰레드의 속성을 나타내는 **pthread_attr_t** 형의 속성 개체
- **start:** 쓰레드의 시작 루틴
- **arg:** 쓰레드 시작 루틴이 넘겨받는 **void *** 형의 인자
- 반환값
  - 성공적인 호출에 대하여 0 을 반환하고, 실패하면 오류 코드 반환

쓰레드 생성과 종료

pthread_exit 함수

기능

- 호출하는 쓰레드를 종료

사용법

```
#include <pthread.h>

void pthread_exit(void *retval);
```

- **retval:** 쓰레드가 종료할 때 반환되는 값
- 반환값
  - 성공적인 호출에 대하여 0 을 반환하고, 실패하면 오류 코드 반환

주의점

- 쓰레드 시작 함수에서 **return** 문도 **pthread_exit()** 를 수행한 것과 동일
- **main()** 함수를 수행하는 초기 쓰레드만 종료하기 위해 **pthread_exit(NULL)** 호출

쓰레드 생성과 종료

▣ 쓰레드 생성과 종료 사용 예

◈ 쓰레드 생성과 종료 예제 프로그램

```
/* hellothread.c */
/* pthread create example */
#include <pthread.h>

void *hello_thread (void *arg)
{
        printf ("Thread: Hello, World!₩n");
        return arg;
}

main()
{
        pthread_t tid;
        int status;

        /* 쓰레드 생성 */
        status = pthread_create (&tid, NULL, hello_thread, NULL);
        if (status != 0)
                perror ("Create thread");
        pthread_exit (NULL);
}
```

쓰레드 생성과 종료

▣ 쓰레드 생성과 종료 사용 예

◈ 쓰레드 생성과 종료 예제 프로그램 실행 결과

```
[cprog2@seps5 ch11]$ gcc -o hellothread hellothread.c -lpthread
[cprog2@seps5 ch11]$ hellothread
Thread: Hello, World!
[cprog2@seps5 ch11]$
```

쓰레드 인자 전달

▣ 쓰레드 인자 전달 사용 예

◈ 쓰레드 인자 전달 예제 프로그램

```
/* hellothreads.c */
/* pthread argument passing example */
#include <stdio.h>
#include <pthread.h>

#define NUM_THREADS 3

void *hello_thread (void *arg)
{
        printf("Thread %d: Hello, World!₩n", arg);
        return arg;
}

main()
{
        pthread_t tid[NUM_THREADS];
        int i, status;

        /* 쓰레드 생성 */
        for (i = 0; i < NUM_THREADS; i++) {
                status = pthread_create (&tid[i], NULL,
                                hello_thread, (void *) i);
                if (status != 0) {
                        fprintf (stderr, "Create thread
                                %d: %d", i, status);
                        exit (1);
                }
        }
        pthread_exit (NULL);
}
```

쓰레드 인자 전달

▣ 쓰레드 인자 전달 사용 예

◈ 쓰레드 인자 전달 예제 프로그램 실행 결과

```
[cprog2@seps5 ch11]$ gcc -o hellothreads hellothreads.c -lpthread
[cprog2@seps5 ch11]$ hellohreads
Thread 0: Hello, World!
Thread 1: Hello, World!
Thread 2: Hello, World!
[cprog2@seps5 ch11]$
```

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

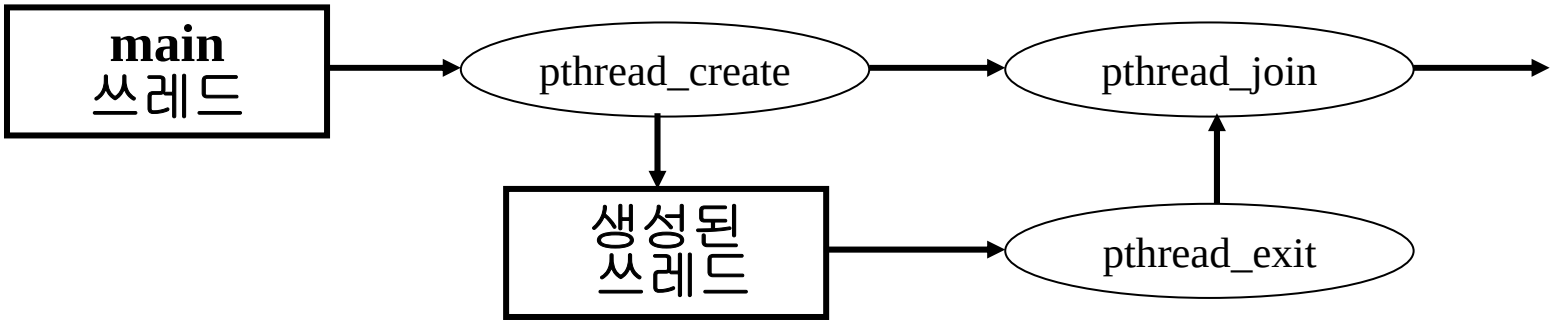
◈ 쓰레드 결합 예제 프로그램

```
/* 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 취소

▣ 쓰레드 취소

◈ 한 쓰레드는 다른 쓰레드를 종료하도록 취소 요청 가능

◈ 최소 요청을 받은 쓰레드는 설정 상태에 따라 동작

➢ 요청 무시, 즉시 취소, 지정된 취소 지점(**cancellation point**)에 도달할 때까지 지연

취소 유형	취소 상태	동작
모두	비활성화 **(disabled)**	취소 활성화될 때까지 보류된다.
지연 **(deferred)**	활성화 **(enabled)**	다음 취소 지점에 도달할 때까지 보류된다.
비동기 **(asynchronous)**	활성화 **(enabled)**	모든 취소 요청이 즉시 처리된다.

◈ 취소 지점

➢ 쓰레드 취소를 허용하는 프로그램의 특정 부분

➢ 리눅스에서 **pthread_join(), pthread_cond_wait(), pthread_cond_timedwait(), pthread_testcancel(), sem_wait(), sigwait()** 와 같은 **POSIX** 쓰레드 함수들과 **read()** 와 같은 일부 시스템 함수들은 취소 지점 포함

쓰레드 취소

- 쓰레드 취소 관련 함수들
  - 기능
    - **pthread_cancel** – 쓰레드를 취소
    - **pthread_setcancelstate** – 쓰레드 취소 상태 변경
    - **pthread_setcanceltype** – 쓰레드 취소 유형 변경
    - **pthread_testcancel** – 쓰레드 취소 요청 점검
  - 사용법

```
#include <pthread.h>

int pthread_cancel(pthread_t thread);
int pthread_setcancelstate(int state, int *oldstate);
int pthread_setcanceltype(int type, int *oldtype);
void pthread_testcancel(void);
```

    - **thread** – 취소되기를 바라는 쓰레드, 취소 요청이 허용되면 쓰레드는 모든 정리 및 종료 작업이 수행되고 **PTHREAD_CANCELED** 값 반환
    - **state** – 새로운 최소 상태. **PTHREAD_CANCEL_ENABLE, PTHREAD_CANCEL_DISABLE**
    - **Type** – 새로 지정되는 취소 유형. **PTHREAD_CANCEL_DEFERRED, PTHREAD_CANCEL_ASYNCHRONOUS.**
  - 반환값
    - 성공적 수행: **0**, 그렇지 않을 경우: **0** 이 아닌 값

쓰레드 취소

▣ 쓰레드 취소 예

◈ 쓰레드 취소 예제 프로그램

```
/* cancelthread.c */
/* pthread cancel example */
#include <stdio.h>
#include <pthread.h>

void *cancel_thread(void *arg)
{
    int i, state;

    for (i=0;;i++) {
        /* disables cancelability */
        pthread_setcancelstate(
                PTHREAD_CANCEL_DISABLE, &state);
        printf("Thread: cancel state disabled₩n");
        sleep (1);
        /* restores cancelability */
        pthread_setcancelstate(state, &state);
        printf("Thread: cancel state restored₩n");
        if (i % 5 == 0)
                pthread_testcancel();
    }
    return arg;
}

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

        if (argc < 2) {
                fprintf (stderr, "Usage: cancelthread
time(sec)₩n");
                exit(1);
        }
         /* 쓰레드 생성 */
        status = pthread_create (&tid, NULL,
cancel_thread, NULL);
```

쓰레드 취소

▣ 쓰레드 취소 예

◈ 쓰레드 취소 예제 프로그램 계속

```
        if (status != 0) {
                fprintf (stderr, "Create thread: %d", status);
                exit (1);
        }
        arg = atoi (argv[1]);
        sleep (arg);

        status = pthread_cancel (tid);
        if (status != 0) {
                fprintf (stderr, "Cancel 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 취소

쓰레드 취소 예

쓰레드 취소 예제 프로그램 실행 결과

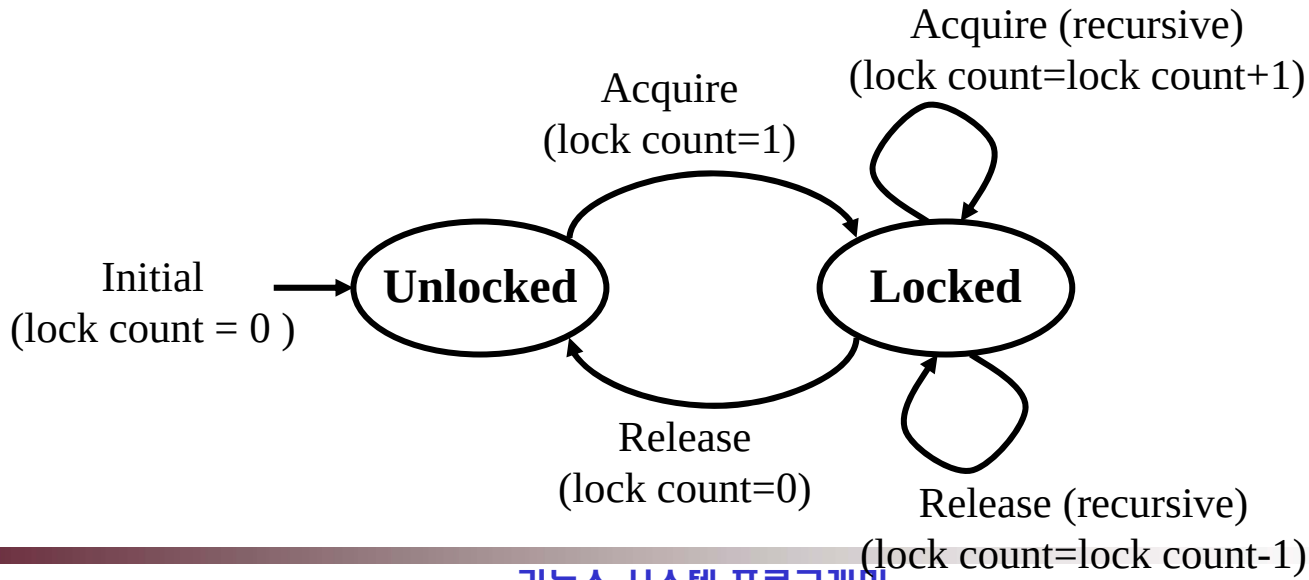
```
[cprog2@seps5 ch11]$ gcc -o cancelthread cancelthread.c -lpthread
[cprog2@seps5 ch11]$ ./cancelthread
Usage: cancelthread time(sec)
[cprog2@seps5 ch11]$ ./cancelthread 1
Thread: cancel state disabled
Thread: cancel state restored
[cprog2@seps5 ch11]$ ./cancelthread 2
Thread: cancel state disabled
Thread: cancel state restored
Thread: cancel state disabled
Thread: cancel state restored
Thread: cancel state disabled
Thread: cancel state restored
Thread: cancel state disabled
Thread: cancel state restored
Thread: cancel state disabled
Thread: cancel state restored
Thread: cancel state disabled
Thread: cancel state restored
[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



뮤텍스

- 속성
  - 뮤텍스 소유권
    - 쓰레드가 뮤텍스를 처음 잠글 때 뮤텍스를 획득
    - 뮤텍스를 획득할 때, 다른 태스크가 뮤텍스를 잠그거나 풀 수 없다.
  - 재귀 잠금
    - 잠금 상태에서 여러 번 뮤텍스를 획득하도록 허용.
    - 계수는 뮤텍스를 소유한 쓰레드가 뮤텍스를 잠그거나 푼 횟수를 추적하는 데 사용.
  - 쓰레드 삭제 안전성
    - 쓰레드가 뮤텍스를 잠그거나 풀 때 쓰레드 삭제가 잠겨진다.
  - 우선순위 역전 회피
    - 더 높은 우선순위 쓰레드가 더 낮은 우선 순위 쓰레드가 사용하는 자원을 위해 대기할 때 우선순위 역전 발생.
    - **Priority Inversion Protocol**
    - **Ceiling Priority Protocol**

뮤텍스

▣ 뮤텍스 생성 및 파괴 함수

◈ 기능

- **PTHREAD_MUTEX_INITIALIZER** – 뮤텍스를 초기화
- **pthread_mutex_init** – 뮤텍스를 동적으로 생성하고 초기화
- **pthread_mutex_destroy** – 동적으로 초기화한 뮤텍스를 파괴

◈ 사용법

```
#include <pthread.h>

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
int pthread_mutex_init(pthread_mutex_t *mutex,
                       const pthread_mutexattr_t *mutexattr);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

- **mutex** – 초기화하거나 파괴하는 뮤텍스 개체의 주소
- **mutexattr** – 초기화하는 뮤텍스의 속성, **NULL** 이면 기본 속성값 설정

◈ 반환값

- **pthread_mutex_init** 은 항상 **0** 을 반환
- **pthread_mutex_destroy** 는 성공적 수행**: 0,** 그렇지 않을 경우**: 0** 이 아닌 값

뮤텍스

▣ 뮤텍스 잠금 및 해제 함수

◈ 기능

- **pthread_mutex_lock** – 뮤텍스를 잠금
- **pthread_mutex_trylock** – 뮤텍스를 잠그지만, 즉시 반환
- **pthread_mutex_unlock** – 뮤텍스를 해제

◈ 사용법

```
#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

- **mutex** – 잠그거나 해제하려는 뮤텍스 개체의 주소

◈ 반환값

- 성공적 수행: **0**, 그렇지 않을 경우: **0** 이 아닌 값

◈ 주의점

- **Pthread_mutex_lock** 는 뮤텍스가 다른 쓰레드에 의해 잠겨져 있는 상태이면 해제될 때까지 호출 쓰레드를 대기시키지만, **pthread_mutex_trylock** 는 즉시 반환.

뮤텍스

▣ 뮤텍스 사용 예

◈ 뮤텍스 예제 프로그램

```
/* mutexthread.c */
/* mutex example */
#include <stdio.h>
#include <pthread.h>

#define NUM_THREADS 3

pthread_mutex_t mutex;
int sum;

void *mutex_thread(void *arg)
{
        pthread_mutex_lock (&mutex);
        sum += (int) arg;
        pthread_mutex_unlock (&mutex);

        return arg;
}
```

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

        if (argc < 4) {
                fprintf (stderr, "Usage: mutexthread
number1 number2 number3₩n");
                exit(1);
        }

        for (i = 0; i < NUM_THREADS; i++)
                arg[i] = atoi (argv[i+1]);

        pthread_mutex_init (&mutex, NULL);

        /* 쓰레드 생성 */
        for (i = 0; i < NUM_THREADS; i++) {
                status = pthread_create (&tid[i], NULL,
mutex_thread, (void *)arg[i]);
```

뮤텍스

▣ 뮤텍스 사용 예

◈ 뮤텍스 예제 프로그램 계속

```
            if (status != 0) {
            fprintf (stderr, "Create thread %d: %d", i, status);
                    exit (1);
              }
        }
        for (i = 0; i < NUM_THREADS; i++) {
              status = pthread_join (tid[i], &result);
              if (status != 0) {
                    fprintf (stderr, "Join thread %d: %d", i, status);
                    exit (1);
              }
        }
        status = pthread_mutex_destroy (&mutex);
        if (status != 0)
              perror ("Destroy mutex");

        printf ("sum is %d₩n", sum);
        pthread_exit (result);
}
```

뮤텍스

▣ 뮤텍스 사용 예

◈ 뮤텍스 예제 프로그램 실행 결과

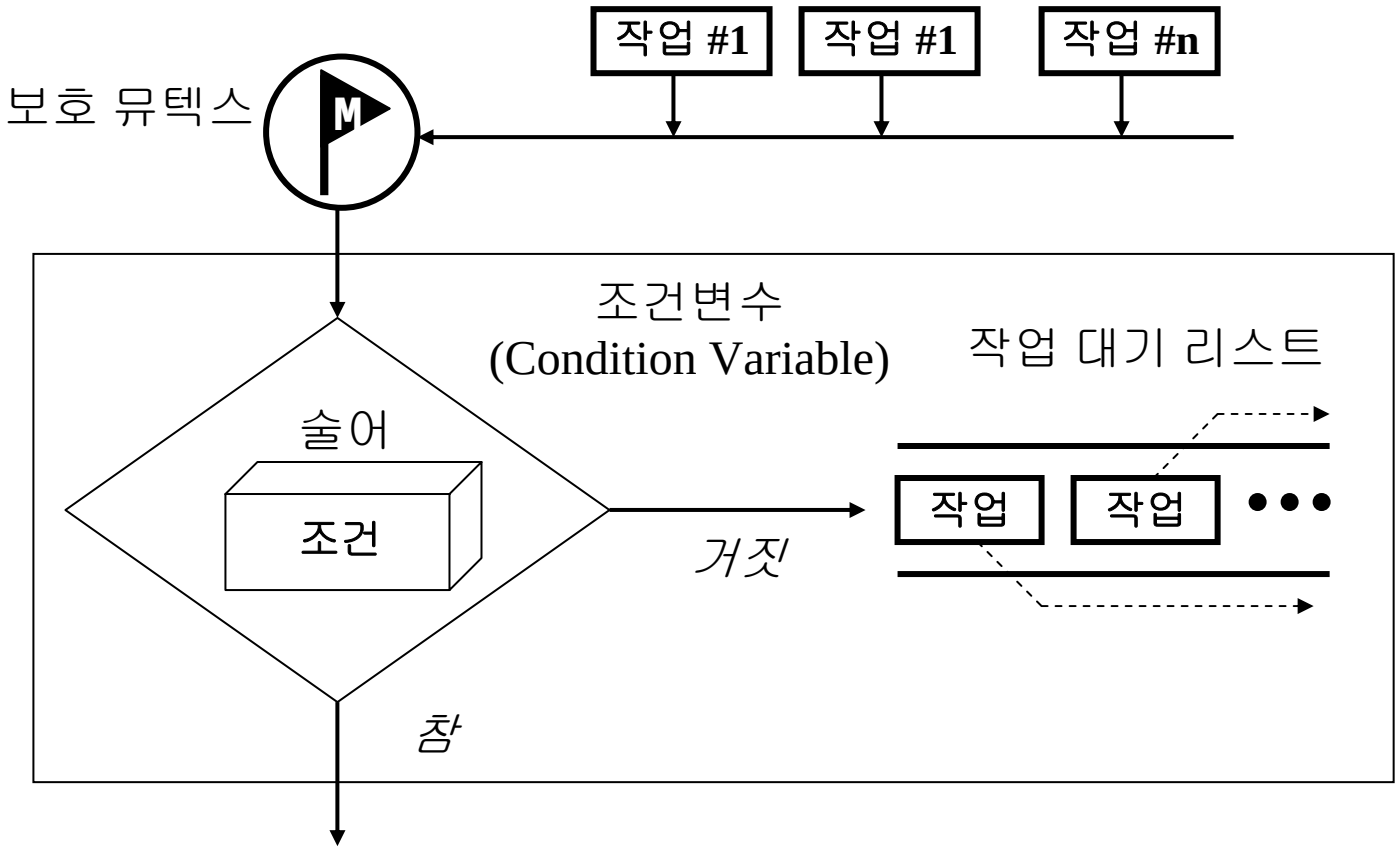
```
[cprog2@seps5 ch11]$ gcc -o mutexthread mutexthread.c -lpthread
[cprog2@seps5 ch11]$ ./mutexthread
Usage: mutexthread number1 number2 number3
[cprog2@seps5 ch11]$ ./mutexthread 1 1 1
sum is 3
[cprog2@seps5 ch11]$ ./mutexthread 1 2 3
sum is 6
[cprog2@seps5 ch11]$
```

조건변수

- 조건변수
  - 공유 자원에 접근할 때, 특정한 조건이 만족할 때에만 접근 가능하고 그렇지 않을 때는 대기하도록 하는 동기화 메커니즘
- 구성 요소
  - 술어 **(Predicate)**
    - 공유 자원의 조건과 연관된 논리 표현식 – 참**(true)** 또는 거짓**(false)** 으로 평가
    - 참이면 조건이 만족되어 쓰레드는 실행 계속
    - 거짓이면 다른 쓰레드가 조건에 대한 신호를 보낼 때까지 대기
  - 보호 뮤텍스
    - 조건 변수에 대한 배제적 접근 보장
    - 쓰레드는 술어를 평가하기 전에 먼저 뮤텍스를 획득하여야 함
  - 쓰레드 대기 목록
    - 쓰레드 재실행 순서는 우선순위 또는 **FIFO** 기반
- 조건변수의 구현 핵심
  - 조건 변수를 사용할 때, 쓰레드 대기 시 **unlock-and-wait** 와 쓰레드 재실행 시 **resume-and-lock** 이 **“원소적”**으로 수행

조건변수

- 조건변수 개념도



조건변수

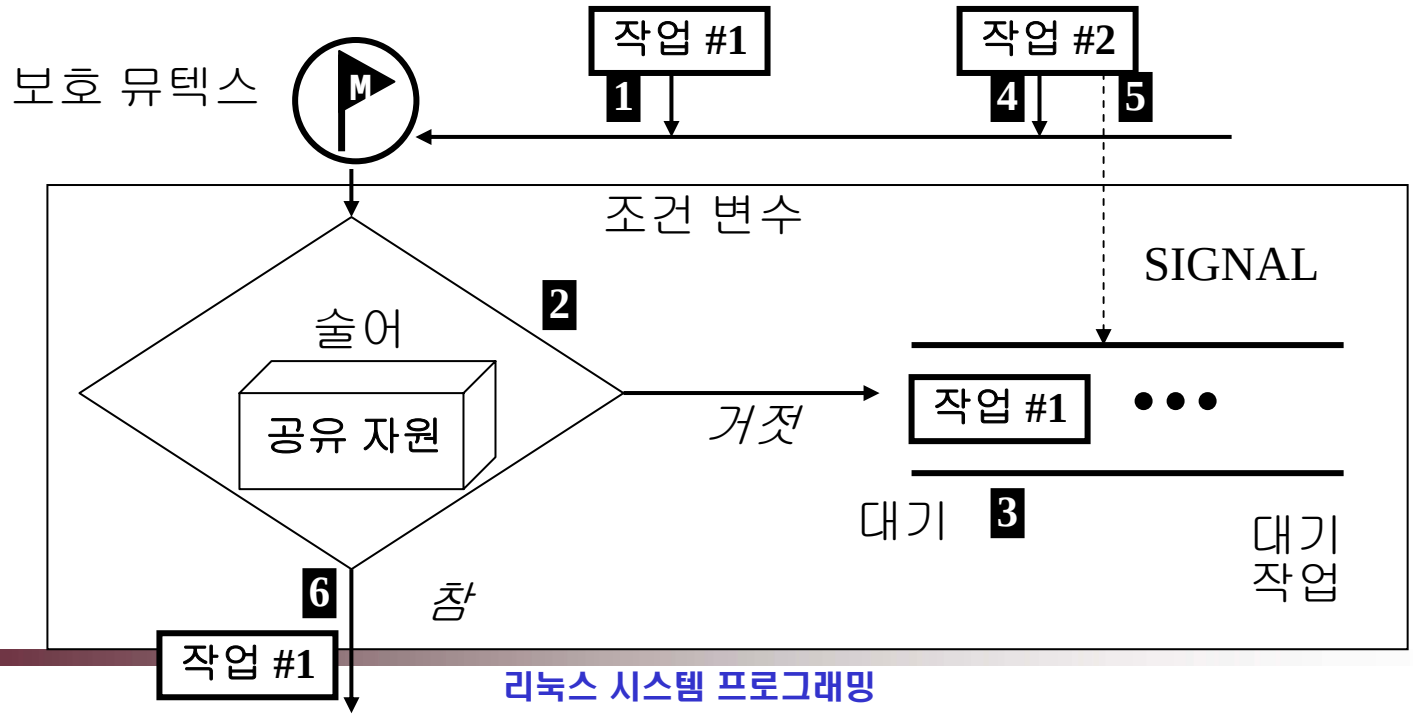
- 조건변수 주요 연산
  - 생성 (Create)
    - 조건변수 생성 및 초기화
  - 대기 (Wait)
    - 조건변수 대기
    - 조건변수 대기를 위해 쓰레드는 먼저 보호 뮤텍스를 획득하여야 함
    - 뮤텍스 해제와 쓰레드 대기를 원소적으로 수행
  - 신호 (Signal)
    - 조건변수 신호
    - 조건변수 신호를 위해 쓰레드는 먼저 보호 뮤텍스를 획득하여야 함
    - 조건변수에서 대기하는 쓰레드들 중 하나를 깨움
    - 쓰레드 재실행과 뮤텍스 재획득을 원소적으로 수행
  - 브로드캐스트(Broadcast)
    - 조건변수 대기하는 모든 쓰레드들 신호
    - 모든 쓰레드들을 깨우지만, 한 쓰레드만 뮤텍스를 획득하고 다른 쓰레드들은 뮤텍스의 쓰레드 대기 목록에 들어감

조건변수

Wait 및 Signal 동작 예

task1
- Lock mutex
  - Examine shared resource
  - While (shared resource is Busy)
    - Wait (condition variable)
  - Mark shared resource as Busy
- Unlock mutex

task2
- Lock mutex
  - Mark shared resource as Free
  - SIGNAL (condition variable)
- Unlock mutex



조건변수

▣ 조건변수 생성 및 파괴 함수

◈ 기능

- **PTHREAD_COND_INITIALIZER** – 조건변수를 초기화
- **pthread_cond_init** – 조건변수를 동적으로 생성하고 초기화
- **pthread_cond_destroy** – 동적으로 초기화한 조건변수를 파괴

◈ 사용법

```
#include <pthread.h>

pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
int pthread_cond_init(pthread_cond_t *cond,
                      const pthread_condattr_t *condattr);
int pthread_cond_destroy(pthread_cond_t *cond);
```

- **cond** – 초기화하거나 파괴하는 조건변수 개체의 주소
- **condattr** – 초기화하는 조건변수의 속성, **NULL** 이면 기본 속성값 설정

◈ 반환값

- **pthread_cond_init** 은 항상 **0** 을 반환
- **pthread_cond_destroy** 는 성공적 수행**: 0,** 그렇지 않을 경우**: 0** 이 아닌 값

조건변수

▣ 조건변수 대기 함수

◈ 기능

- **pthread_cond_wait** – 조건변수 상에서 쓰레드를 대기
- **pthread_cond_timedwait** – 조건변수 상에서 쓰레드를 특정 시간 동안 대기

◈ 사용법

```
#include <pthread.h>

int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
int pthread_cond_timedwait(pthread_cond_t *cond, pthread_mutex_t *mutex,
                           const struct timespec *abstime);
```

- **cond** – 대기하는 쓰레드들의 큐를 가진 조건변수 개체의 주소
- **mutex** – 원소적으로 잠금을 해제할 뮤텍스 개체의 주소
- **abstime** – 대기할 절대 시간. **time()** 또는 **gettimeofday()** 함수에 의해 획득 가능

◈ 반환값

- **pthread_cond_wait** 는 항상 **0**을 반환
- **pthread_cond_timedwait** 는 성공적 수행**: 0,** 그렇지 않을 경우**: 0** 이 아닌 값

조건변수

▣ 조건변수 신호 함수

◈ 기능

- **pthread_cond_signal** – 조건변수 상에서 신호하여 대기하는 쓰레드를 재실행
- **pthread_cond_broadcast** – 조건변수 상에서 신호하여 대기하는 모든 쓰레드를 재실행

◈ 사용법

```
#include <pthread.h>

int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_broadcast(pthread_cond_t *cond);
```

- **cond** – 대기하는 쓰레드들의 큐를 가진 조건변수 개체의 주소

조건변수

▣ 조건변수 사용 예

◈ 조건변수 예제 프로그램

```
/* boundedbuffer.c */
/* bounded buffer example */
#include <stdlib.h>
#include <pthread.h>

#define BUFFER_SIZE 20
#define NUMITEMS 30

typedef struct {
        int item[BUFFER_SIZE];
        int totalitems;
        int in, out;
        pthread_mutex_t mutex;
        pthread_cond_t full;
        pthread_cond_t empty;
} buffer_t;

buffer_t bb = { {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, 0, 0, 0,
        PTHREAD_MUTEX_INITIALIZER,
        PTHREAD_COND_INITIALIZER,
        PTHREAD_COND_INITIALIZER};
```

```
int produce_item ()
{
        int item = (int) (100.0*rand()/(RAND_MAX+1.0));
        sleep((unsigned long) (5.0*rand()/(RAND_MAX+1.0)));
        printf("produce_item: item=%d₩n", item);
        return item;
}

insert_item (int item)
{
        int status;

        status = pthread_mutex_lock (&bb.mutex);
        if (status != 0)
                return status;

        while (bb.totalitems >= BUFFER_SIZE && status ==
NULL)
                status = pthread_cond_wait (&bb.empty,
&bb.mutex);
        if (status != 0) {
                pthread_mutex_unlock(&bb.mutex);
                return status;
        }
```

조건변수

▣ 조건변수 사용 예

◈ 조건변수 예제 프로그램 계속

```
        bb.item[bb.in] = item;
        bb.in = (bb.in + 1) % BUFFER_SIZE;
        bb.totalitems++;

        if (status = pthread_cond_signal(&bb.full)) {
                pthread_mutex_unlock (&bb.mutex);
                return status;
        }
        return pthread_mutex_unlock (&bb.mutex);
}

consume_item (int item)
{
        sleep((unsigned long)
(5.0*rand()/(RAND_MAX+1.0)));
        printf("₩t₩tconsume_item: item=%d₩n", item);
}

remove_item (int *temp)
{
        int status;
```

```
        status = pthread_mutex_lock (&bb.mutex);
        if (status != 0)
                return status;

        while (bb.totalitems <= 0 && status == NULL)
                status = pthread_cond_wait (&bb.full, &bb.mutex);
        if (status != 0) {
                pthread_mutex_unlock(&bb.mutex);
                return status;
        }

        *temp = bb.item[bb.out];
        bb.out = (bb.out + 1) % BUFFER_SIZE;
        bb.totalitems--;

        if (status = pthread_cond_signal(&bb.empty)) {
                pthread_mutex_unlock (&bb.mutex);
                return status;
        }
        return pthread_mutex_unlock (&bb.mutex);
}
```

조건변수

▣ 조건변수 사용 예

◈ 조건변수 예제 프로그램 계속

```
void * producer(void *arg)
{
        int item;

        while (1) {
                item = produce_item ();
                insert_item(item);
        }
}

void * consumer(void *arg)
{
        int item;

        while (1) {
                remove_item (&item);
                consume_item (item);
        }
}
```

```
main ()
{
        int status;
        void *result;
        pthread_t producer_tid, consumer_tid;

        /* 쓰레드 생성 */
        status = pthread_create (&producer_tid, NULL,
producer, NULL);
        if (status != 0)
                perror ("Create producer thread");
        status = pthread_create (&consumer_tid, NULL,
consumer, NULL);
        if (status != 0)
                perror ("Create consumer thread");

        status = pthread_join (producer_tid, NULL);
        if (status != 0)
                perror ("Join producer thread");
        status = pthread_join (consumer_tid, NULL);
        if (status != 0)
                perror ("Join consumer thread");
}
```

조건변수

▣ 조건변수 사용 예

◈ 조건변수 예제 프로그램 실행 결과

```
[cprog2@seps5 ch11]$ gcc -o boundedbuffer boundedbuffer.c -lpthread
[cprog2@seps5 ch11]$ ./boundedbuffer
produce_item: item=84
produce_item: item=78
                consume_item: item=84
produce_item: item=19
produce_item: item=27
                consume_item: item=78
                consume_item: item=19
produce_item: item=47
                consume_item: item=27
produce_item: item=95
produce_item: item=71
                consume_item: item=47
[cprog2@seps5 ch11]$
```

읽기–쓰기 잠금

- 읽기-쓰기 잠금(**read-write lock**)
  - 잠그는 동안 여러 쓰레드가 동시에 읽을 수는 있지만, 하나의 쓰레드만 쓰기가 가능한 동기화 메커니즘
  - 데이터베이스와 같은 공유 자료 개체에 접근하기 위해 사용
- 뮤텍스와의 차이점
  - 읽기와 쓰기를 구분
    - 공유 읽기 접근 **(shared read access)**
      - 읽기 잠금을 수행하는 동안 읽기 쓰레드는 접근을 허용하고 쓰기 쓰레드는 허용하지 않음
    - 배타적 쓰기 접근 **(eclusive write access)**
      - 쓰기 잠금을 수행하는 동안 다른 쓰레드의 접근을 허용하지 않음

읽기-쓰기 잠금

읽기-쓰기 잠금 생성 및 파괴 함수

기능

- **PTHREAD_RWLOCK_INITIALIZER** – 읽기-쓰기 잠금을 초기화
- **pthread_rwlock_init** – 읽기-쓰기 잠금을 동적으로 생성하고 초기화
- **pthread_rwlock_destroy** – 동적으로 초기화한 읽기-쓰기 잠금을 파괴

사용법

```
#include <pthread.h>

pthread_rwlock_t rwlock = PTHREAD_RWLOCK_INITIALIZER;
int pthread_rwlock_init(pthread_mutex_t *rwlock,
                                const pthread_rwlockattr_t *rwlockattr);
int pthread_rwlock_destroy(pthread_rwlock_t *rwlock);
```

- **rwlock** – 초기화하거나 파괴하는 읽기-쓰기 잠금 개체의 주소
- **rwlockattr** – 초기화하는 읽기-쓰기 잠금의 속성, **NULL** 이면 기본 속성값 설정

반환값

- **pthread_rwlock_init** 은 항상 **0** 을 반환
- **pthread_rwlock_destroy** 는 성공적 수행**: 0,** 그렇지 않을 경우**: 0** 이 아닌 값

읽기-쓰기 잠금

- 읽기-쓰기 잠금 및 해제 함수
  - 기능
    - **pthread_rwlock_rdlock** – 읽기-쓰기 잠금을 읽기를 위해 잠금
    - **pthread_rwlock_tryrdlock** – 읽기-쓰기 잠금을 읽기를 위해 잠그지만, 즉시 반환
    - **pthread_rwlock_wrlock** – 읽기-쓰기 잠금을 쓰기를 위해 잠금
    - **pthread_rwlock_trywrlock** – 읽기-쓰기 잠금을 쓰기를 위해 잠그지만, 즉시 반환
    - **pthread_rwlock_unlock** – 읽기-쓰기 잠금을 해제
  - 사용법

```
#include <pthread.h>

int pthread_rwlock_rdlock(pthread_rwlock_t *rwlock);
int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);
int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock);
int pthread_rwlock_trywrlock(pthread_rwlock_t *rwlock);
int pthread_rwlock_unlock(pthread_rwlock_t *rwlock);
```

    - **rwlock**– 잠그거나 해제하려는 읽기-쓰기 잠금 개체의 주소
  - 반환값
    - 성공적 수행: **0**, 그렇지 않을 경우: **0** 이 아닌 값

쓰레드와 시그널

- 쓰레드와 시그널의 관계
  - 시그널 동작과 시그널 핸들러는 프로세스 단위로 존재
  - 한 프로세스 내의 모든 쓰레드는 시그널 핸들러를 공유
  - 각 쓰레드는 자신의 시그널 마스크 가짐
- **POSIX** 쓰레드의 시그널 관련 함수
  - 기능
    - **pthread_kill** – 특정 쓰레드에게 시그널 보냄
    - **pthread_sigmask** – 호출 쓰레드의 시그널 마스크 변경
    - **sigwait** – 특정한 시그널 집합을 받을 때까지 대기
  - 사용법

```
#include <pthread.h>

int pthread_kill(pthread_t thread, int signo);
int pthread_sigmask(int how, const sigset_t *newmask, sigset_t *oldmask);
int sigwait(const sigset_t *set, int *sig);
```

    - **pthread_kill - thread** 가 가리키는 쓰레드에게 **signo** 가 지정하는 시그널 보냄
    - **pthread_sigmask - how: SIG_SETMASK(newmask** 시그널 설정**), SIG_BLOCK (newmask** 시그널 추가**), SIG_UNBLOCK (newmask** 시그널 제거**)**
    - **sigwait – set** 의 시그널 집합을 받을 때까지 호출 쓰레드를 대기

쓰레드와 시그널

▣ 쓰레드와 시그널 사용 예

◈ 쓰레드와 시그널 예제 프로그램

```
/* signalthread.c */
/* thread and signal example */
#include <stdio.h>
#include <pthread.h>
#include <signal.h>

#define NUM_THREADS 3

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
sigset_t sigset;
int completed;

void *signal_thread(void *arg)
{
        int signal;
        int count = 0;

        while (1) {
                sigwait (&sigset, &signal);
                if (signal == SIGINT) {
                        printf ("Signal thread: SIGINT %d₩n",
++count);
                        if (count >= 3) {
                                pthread_mutex_lock (&mutex);
                                completed = 1;
                                pthread_mutex_unlock (&mutex);
                                break;
                        }
                }
        }
        return arg;
}

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

        if (argc < 2) {
            fprintf (stderr, "Usage: signalthread time(sec)₩n");
            exit(1);
        }
        arg = atoi (argv[1]);

        sigemptyset(&sigset);
        sigaddset(&sigset, SIGINT);
```

쓰레드와 시그널

▣ 쓰레드와 시그널 사용 예

◈ 쓰레드와 시그널 예제 프로그램

```
        status = pthread_sigmask (SIG_BLOCK, &sigset, NULL);
        if (status != 0) {
                fprintf (stderr, "Set signal mask");
                exit (1);
        }
        /* 쓰레드 생성 */
        status = pthread_create (&tid, NULL, signal_thread, NULL);
        if (status != 0) {
                fprintf (stderr, "Create thread: %d", status);
                exit (1);
        }
        while (1) {
                sleep (arg);
                pthread_mutex_lock (&mutex);
                printf ("Main thread: mutex locked₩n");
                if (completed) break;
                pthread_mutex_unlock (&mutex);
        }
        pthread_mutex_unlock (&mutex);

        pthread_exit (result);
}
```

쓰레드와 시그널

▣ 쓰레드와 시그널 사용 예

◈ 쓰레드와 시그널 예제 프로그램 실행 결과

```
[cprog2@seps5 ch11]$ gcc -o signalthread signalthread.c -lpthread
[cprog2@seps5 ch11]$ ./signalthread
Usage: signalthread time(sec)
[cprog2@seps5 ch11]$ ./signalthread 1
Main thread: mutex locked
Main thread: mutex locked
Signal thread: SIGINT 1
Main thread: mutex locked
Signal thread: SIGINT 2
Main thread: mutex locked
Signal thread: SIGINT 3
Main thread: mutex locked
[cprog2@seps5 ch11]$
```

쓰레드 속성

- 쓰레드 속성
  - 쓰레드의 특성에 대한 자료
  - **POSIX** 쓰레드 속성 개체는 **pthread_attr_t** 자료형으로 표현
  - 종류
    - **detachstate** – 쓰레드의 분리/결합 상태
    - **stack** – 스택 속성
    - **schedpolicy** – 쓰레드의 스케줄링 정책
    - **schedparam** – 쓰레드의 스케줄링 매개변수
- 쓰레드 속성 연산
  - 초기화 및 파괴 – **pthread_attr_init(), pthread_attr_destroy()**
  - 속성 설정 – **pthread_attr_set<attrname>()**
  - 속성 획득 – **pthread_attr_get<attrname>()**

쓰레드 속성

▣ 쓰레드 속성 객체 함수

◈ 기능

- **pthread_attr_init()** - 속성 객체를 생성하고 초기화
- **pthread_attr_destroy()** - 속성 객체를 파괴
- **pthread_attr_setdetachstate()** - 속성 객체의 상태를 설정
- **pthread_attr_getdetachstate()** - 속성 객체의 상태를 획득

◈ 사용법

```
#include <pthread.h>

int pthread_attr_init(pthread_attr_t *attr);
int pthread_attr_destroy(pthread_attr_t *attr);
int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
int pthread_attr_getdetachstate(const pthread_attr_t *attr, int *detachstate);
```

- **attr :** 쓰레드 속성 개체의 주소
- **detachstate :** 쓰레드 속성 개체의 결합/분리 상태
  - **PTHREAD_CREATE_JOINABLE :** 쓰레드가 결합 가능 상태로 생성
  - **PTHREAD_CREATE_DETACHED :** 쓰레드가 분리 상태로 생성

◈ 반환값

- 성공적인 호출 **: 0,** 그렇지 않으면 **0** 이 아닌 값

쓰레드 속성

- 리눅스 쓰레드 스케줄링 모델
  - **3가지 POSIX** 쓰레드 스케줄링 정책 지원
    - **SCHED_FIFO (first-in first-out)** – 스스로 중지하거나, 더 높은 우선순위를 가지는 실시간 쓰레드가 준비 상태일 때까지 실행
    - **SCHED_RR (round robin)** – 특정한 시간 간격 동안 실행
    - **SCHED_OTHER** – 일반적인 비실시간**(non-realtime)** 쓰레드
  - 우선순위 쓰레드 스케줄링
    - 쓰레드의 스케줄링 매개변수 – 우선순위**(priority)**
  - **2**가지 쓰레드 상속 방식
    - **PTHREAD_INHERIT_SCHED** – 부모 쓰레드의 스케줄링 정책과 매개변수를 상속
    - **PTHREAD_EXPLICIT_SCHED** – 자신의 스케줄링 정책과 매개변수 지정
  - 경합 범위 **(contention scope)** 는 **PTHREAD_SCOPE_SYSTEM** 만 지원
    - **PTHREAD_SCOPE_SYSTEM** – 쓰레드가 자원 획득을 위해 시스템의 모든 쓰레드와 경합
    - **PTHREAD_SCOPE_PROCESS** – 쓰레드가 프로세스 내의 쓰레드들과 경합

쓰레드 속성

- 쓰레드 스케줄링 함수
  - 기능
    - **pthread_attr_setschedpolicy()** 는 쓰레드 스케줄링 정책 설정
    - **pthread_attr_getschedpolicy()** 는 쓰레드 스케줄링 정책 획득
    - **pthread_attr_setschedparam()** 는 쓰레드 스케줄링 매개변수 설정
    - **pthread_attr_getschedparam()** 는 쓰레드 스케줄링 매개변수 획득
    - **pthread_attr_setinheritsched()** 는 쓰레드 스케줄링 속성 상속 설정
    - **pthread_attr_getinheritsched()** 는 쓰레드 스케줄링 속성 상속 획득
    - **pthread_attr_setscope()** 는 쓰레드 경합 범위 설정
    - **pthread_attr_getscope()** 는 쓰레드 경합 범위 획득
  - 사용법

```
#include <pthread.h>

int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
int pthread_attr_getschedpolicy(const pthread_attr_t *attr,  int  *policy);
int pthread_attr_setschedparam(pthread_attr_t *attr, const struct sched_param *param);
int pthread_attr_getschedparam(const pthread_attr_t *attr, struct sched_param *param);
int pthread_attr_setinheritsched(pthread_attr_t *attr, int inherit);
int pthread_attr_getinheritsched(const pthread_attr_t *attr, int *inherit);
int pthread_attr_setscope(pthread_attr_t *attr, int scope);
int pthread_attr_getscope(const pthread_attr_t *attr, int *scope);
```

  - 반환값
    - 성공적인 호출 : **0,** 그렇지 않으면 **0** 이 아닌 값

쓰레드 속성

▣ 쓰레드 속성 사용 예

◈ 쓰레드 속성 예제 프로그램

```
/* attrthread.c */
/* thread attributes example */
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

void * attr_thread1 (void *arg)
{
        printf ("Thread 1₩n");
        sleep (1);
        return (void *) 1;
}

void * attr_thread2 (void *arg)
{
        printf ("Thread 2₩n");
        sleep (1);
        return (void *) 2;
}

int main (void)
{
        pthread_attr_t attr;
        pthread_t tid1, tid2;
        int status;
        void *result1;
        void *result2;

        pthread_attr_init (&attr);
        status = pthread_attr_setdetachstate (&attr,
                                PTHREAD_CREATE_JOINABLE);
        if (status != 0) {
                fprintf (stderr, "Setdetachstate₩n");
                exit (1);
        }
        pthread_create (&tid1, &attr, attr_thread1, NULL);
        status = pthread_attr_setdetachstate (&attr,
                                PTHREAD_CREATE_DETACHED);
        if (status != 0) {
                fprintf (stderr, "Setdetachstate₩n");
                exit (1);
        }
        pthread_create (&tid2, &attr, attr_thread2, NULL);
        pthread_attr_destroy (&attr);

        pthread_join (tid1, &result1);
        pthread_join (tid2, &result2);

        printf ("res1 = %p₩n", result1);
        printf ("res2 = %p₩n", result2);

        return result1 != (void *) 1 || result2 != (void *) 2;
}
```

쓰레드 속성

쓰레드 속성 사용 예

쓰레드 속성 예제 프로그램 실행 결과

```
[cprog2@seps5 ch11]$ gcc -o attrthread attrthread.c -lpthread
[cprog2@seps5 ch11]$ ./attrthread
Thread 1
Thread 2
res1 = 0x1
res2 = 0x4002dab0
[cprog2@seps5 ch11]$
```

쓰레드 특정 자료

- 쓰레드 특정 자료 **(Thread-specific data)**
  - 각 쓰레드에서 사용되는 고유한 자료
    - 각 쓰레드가 한 변수에 대한 개별 복사본을 소유하도록 함
  - 고려사항
    - 프로세스 내의 모든 쓰레드는 정적 저장 공간 등을 공유하므로, 고유한 값을 저장하기는 쉽지 않다.
    - 정적 자료를 사용하여 각 쓰레드의 유일한 값(쓰레드 **ID** 등)을 통해 검색할 수 있는 테이블 생성 → 얼마나 많은 쓰레드가 함수를 호출할 것인지 예측할 수 없음. 일반적이지 않음.
    - 힙 공간을 할당**(malloc)**하는 경우, 임의의 쓰레드에서 적절한 고유 자료들을 연결 리스트로 관리 → 리스트 검색으로 인한 속도 저하와 쓰레드 종료에 의한 할당 공간 관리가 어려움
    - 함수 호출자가 필수적인 지속 상태**(persistent state)** 를 할당한 후, 할당 위치를 호출자에게 알려줌 – 예) **strtok_r()**
- 쓰레드 특정 자료의 구현
  - 프로그램은 키**(key)** 를 생성하여, 각 쓰레드는 키에 대한 고유한 값을 읽고 쓸 수 있음

쓰레드 특정 자료

▣ 쓰레드 특정 자료 함수

◈ 기능

- **pthread_key_create** - 쓰레드-특정 자료 키를 생성
- **pthread_key_delete** - 쓰레드-특정 자료 키를 제거
- **pthread_setspecific** - 쓰레드-특정 자료 키와 관련된 값을 변경
- **pthread_getspecific** - 쓰레드-특정 자료 키와 관련된 값을 획득

◈ 사용법

```
#include <pthread.h>

int pthread_key_create(pthread_key_t *key, void (*destr_function) (void *));
int pthread_key_delete(pthread_key_t key);
int pthread_setspecific(pthread_key_t key, const void *pointer);
void * pthread_getspecific(pthread_key_t key);
```

- **key :** 쓰레드 특정 자료를 위해 생성, 파괴, 설정, 획득할 키 값
- **destr_function :** 동적으로 생성한 쓰레드 특정 자료의 경우 메모리 해제할 함수
- **pointer :** 변경할 키와 관련된 값

◈ 반환값

- 성공적인 호출 **: 0,** 그렇지 않으면 **0** 이 아닌 값
- **Pthread_getspecific** 은 성공 시 키와 관련된 값을 반환

쓰레드 특정 자료

▣ 쓰레드 특정 자료 사용 예

◈ 쓰레드 특정 자료 예제 프로그램

```
/* tsdthread.c */
/* thread-specific data example */
#include <stdio.h>
#include <pthread.h>

#define NUM_THREADS 3

pthread_key_t tsd_key;
typedef struct tsd_data {
        int thread;
        int key_data;
} tsd_data;

void *tsd_thread(void *arg)
{
        tsd_data *data = (tsd_data *)arg;
        tsd_data *result;
        int status;

        status = pthread_setspecific (tsd_key, data);
        if (status != 0) {
                fprintf (stderr, "Set tsd key: %d", status);
                exit (1);
        }
        result = (tsd_data *) pthread_getspecific (tsd_key);
        printf("Thread %d: key_data is %d₩n", result->thread,
                                                result->key_data);

        return arg;
}

int main(int argc, char *argv[])
{
        pthread_t tid[NUM_THREADS];
        int i;
        tsd_data data[NUM_THREADS];
        void *result;
        int status;

        if (argc < 4) {
                fprintf (stderr, "Usage: tsdthread number1
number1 number3₩n");
                exit(1);
        }
        for (i = 0; i < NUM_THREADS; i++) {
                data[i].thread = i;
                data[i].key_data = atoi (argv[i+1]);
        }
```

쓰레드 특정 자료

▣ 쓰레드 특정 자료 사용 예

◈ 쓰레드 특정 자료 예제 프로그램

```
status = pthread_key_create (&tsd_key, NULL);
if (status != 0) {
        fprintf (stderr, "Create key: %d", status);
        exit (1);
}

/* 쓰레드 생성 */
for (i = 0; i < NUM_THREADS; i++) {
    status = pthread_create (&tid[i], NULL, tsd_thread,
                             (void *)&data[i]);
    if (status != 0) {
        fprintf (stderr, "Create thread %d: %d", i, status);
        exit (1);
    }
}

for (i = 0; i < NUM_THREADS; i++) {
    status = pthread_join (tid[i], &result);
    if (status != 0) {
            fprintf (stderr, "Join thread %d: %d", i, status);
            exit (1);
    }
}
```

```
        status = pthread_key_delete (tsd_key);
        if (status != 0)
                perror ("Destroy key");

        pthread_exit (result);
}
```

쓰레드 특정 자료

▣ 쓰레드 특정 자료 사용 예

◈ 쓰레드 특정 자료 예제 프로그램 실행 결과

```
[cprog2@seps5 ch11]$ gcc -o tsdthread tsdthread.c -lpthread
[cprog2@seps5 ch11]$ ./tsdthread
Usage: tsdthread number1 number1 number3
[cprog2@seps5 ch11]$ ./tsdthread 1 2 3
Thread 0: key_data is 1
Thread 1: key_data is 2
Thread 2: key_data is 3
[cprog2@seps5 ch11]$
```

쓰레드-안전성

쓰레드-안전성 (thread safety)

- 여러 쓰레드가 특정한 함수를 동시에 여러 번 호출하여도 정사적으로 수행될 때, 그 함수를 쓰레드 안전적이라고 함.
  - 재진입성 (reentrancy) 등을 가져야 함
- 다음 함수들을 제외한 모든 POSIX 표준 함수는 쓰레드-안전적

쓰레드-안전성을 보장하지 않는 POSIX 함수들

asctime( )	dlerror()	getgrgid( )	getpwuid( )	lgammal( )	setgrent( )
basename( )	drand48( )	getgrnam( )	getservbyname( )	localeconv( )	setkey( )
catgets( )	ecvt( )	gethostbyaddr( )	getservbyport( )	localtime( )	setpwent( )
crypt( )	encrypt( )	gethostbyname( )	getservent( )	lrand48( )	setutxent( )
ctime( )	endgrent( )	gethostent( )	getutxent( )	mrand48( )	strerror( )
dbm_clearerr( )	endpwent( )	getlogin( )	getutxid( )	nftw( )	strtok( )
dbm_close( )	endutxent( )	getnetbyaddr( )	getutxline( )	nl_langinfo( )	ttyname( )
dbm_delete( )	fcvt( )	getnetbyname( )	gmtime( )	ptsname( )	unsetenv( )
dbm_error( )	ftw( )	getnetent( )	hcreate( )	putc_unlocked( )	wcstombs( )
dbm_fetch( )	gcvt( )	getopt( )	hdestroy( )	putchar_unlocked( )	wctomb( )
dbm_firstkey( )	getc_unlocked( )	getprotobyname( )	hsearch( )	putenv( )	
dbm_nextkey( )	getchar_unlocked( )	getprotobynumber( )	inet_ntoa( )	pututxline( )	
dbm_open( )	getdate( )	getprotoent( )	l64a( )	rand( )	
dbm_store( )	getenv( )	getpwent( )	lgamma( )	readdir( )	
dirname( )	getgrent( )	getpwnam( )	lgammaf( )	setenv( )	

쓰레드-안전성

- 쓰레드-안전성을 보장하지 않는 **POSIX** 함수들
  - 쓰레드-안전성을 보장하는 버전의 함수 이름 끝에 **_r** 을 붙임
    - 예) **strerror_r(), strtok_r()**
- 쓰레드-안전성 보장 방법
  - 재진입성을 가지지 않는 함수의 경우 재진입성을 가지도록 함.
    - 내부적으로 정적 변수나 버퍼를 사용하는 경우 동적 메모리 할당
    - 함수를 호출하는 외부에서 이런 부분을 다룰 수 있도록 인터페이스 수정
  - 쓰레드들이 자료를 공유하는 경우 상호 배제 및 동기화
    - 뮤텍스 등을 사용하여 동기화 수행

쓰레드-안전성

▣ 쓰레드-안전적 함수 사용 예

◈ strerror_r, perror_r 함수 예제

```
/* strerror_r.c */
#include <errno.h>
#include <pthread.h>
#include <signal.h>
#include <stdio.h>
#include <string.h>

static pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

int strerror_r (int errnum, char *strerrbuf, size_t buflen) {
    char *buf;
    int error1, error2, error3;
    sigset_t maskblock, maskold;

    if ((sigfillset(&maskblock) == -1) ||
        (sigprocmask(SIG_SETMASK, &maskblock, &maskold) ==
-1))
        return errno;
    if (error1 = pthread_mutex_lock(&lock)) {
        (void)sigprocmask(SIG_SETMASK,  &maskold, NULL);
        return error1;
    }
    buf = strerror(errnum);
    if (strlen(buf) >= buflen)
        error1 = ERANGE;
    else
        (void *)strcpy(strerrbuf, buf);
    error2 = pthread_mutex_unlock(&lock);
    error3 = sigprocmask(SIG_SETMASK, &maskold, NULL);
    return error1 ? error1 : (error2 ? error2 : error3);
}

int perror_r (const char *s) {
    int error1, error2;
    sigset_t maskblock, maskold;

    if ((sigfillset(&maskblock) == -1) ||
        (sigprocmask(SIG_SETMASK, &maskblock, &maskold)
== -1))
        return errno;
    if (error1 = pthread_mutex_lock(&lock)) {
        (void)sigprocmask(SIG_SETMASK,  &maskold, NULL);
        return error1;
    }
    perror(s);
    error1 = pthread_mutex_unlock(&lock);
    error2 = sigprocmask(SIG_SETMASK, &maskold, NULL);
    return error1 ? error1 : error2;
}
```