

강원혁신플랫폼

리눅스프로그래밍

멀티스레드





Linux 시스템에서 상위 프로세스와 동일한 메모리 공간 및 리소스를 공유하는 경량 프로세스로 단일 프로세스 내에서 동시 실행을 달성하는 데 사용되므로 여러 작업을 동시에 수행 가능한 메커니즘을 무엇이라고 하나요?

스레드/멀티스레드





Linux 시스템에서 스레드 동기화에 사용되어 코드의 중요한 섹션을 보호하고 한 번에 하나의 스레드만 공유 리소스에 액세스할 수 있도록 함으로써, 경합 상태를 방지하고 스레드 간에 상호 배제를 제공하는 메커니즘을 무엇이라고 하나요?

무텍스(상호 배제 잠금)





학습 내용

- 1 멀티스레드의 주요 함수
- 2 스레드 간 동기화 문제
- 3 스레드 동기화의 주요 함수

학습 목표

- 📖 멀티스레드의 주요 함수를 파악할 수 있다.
- 📖 스레드 간 동기화 문제에 대해 설명할 수 있다.
- 📖 스레드 동기화의 주요 함수를 파악할 수 있다.

강원혁신플랫폼

리눅스프로그래밍



멀티스레드의 주요 함수





멀티 프로세스와 멀티 스레드

스레드의 특징

- ◆ 상위 프로세스와 동일한 메모리 공간 및 리소스를 공유하는 경량 프로세스
- ◆ 단일 프로세스 내에서 동시 실행을 달성하는 데 사용되므로 여러 작업을 동시에 수행 가능



멀티 프로세스와 멀티 스레드

프로세스1

스택
힙
데이터
텍스트

프로세스2

스택
힙
데이터
텍스트

멀티 프로세스

스레드1

스택

스레드2

스택

스레드3

스택

힙

데이터

텍스트

멀티 스레드



pthread_create(): 새 스레드를 생성하며
스레드 식별자, 속성, 시작 루틴 및 인수를 매개변수로 사용

```
#include <pthread.h>
```

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

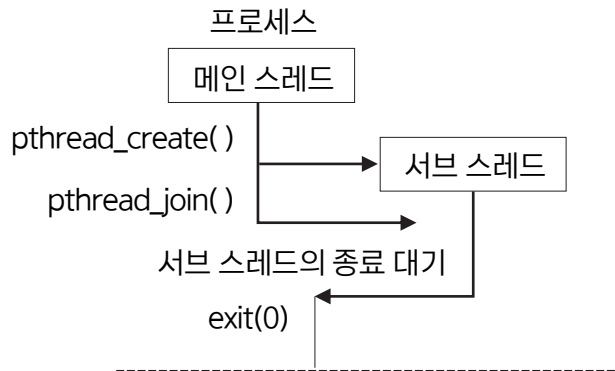
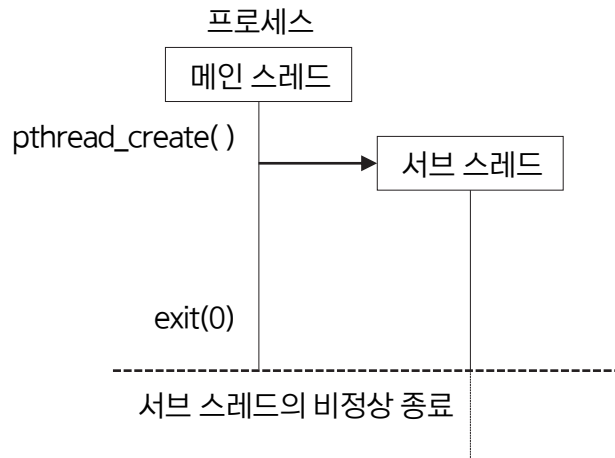



pthread_join(): 특정 스레드가 종료될 때까지 기다림

```
#include <pthread.h>
```

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

pthread_join() 함수의 관계





```
#include <stdio.h>          /* printf() 함수를 위한 헤더 파일 */
#include <unistd.h>
#include <fcntl.h>          /* O_CREAT, O_EXEC 매크로를 위한 헤더 파일 */
#include <pthread.h>
#include <semaphore.h> /* sem_open(), sem_destroy(), sem_wait() 등 함수 헤더 */

sem_t *sem;                /* 세마포어를 위한 전역 변수 */
static int cnt = 0;        /* 세마포어에서 사용할 임계 구역 변수 */
```



```
void p()                /* 세마포어의 P 연산, 임계구역 변수 감소 */
{
    sem_wait(sem); /* wait(), Semaphore 값을 1 감소, 음수라면 대기 */
}

void v()                /* 세마포어의 V 연산, 임계구역 변수 증가 */
{
    sem_post(sem); /* post(), Semaphore 값 1 증가, 대기중인 thread가 있으면 깨움 */
}
```



```
void *ptheadV(void *arg) { /* V 연산(증가)을 수행하기 위한 함수를 작성한다. */
    int i;

    for(i = 0; i < 10; i++) {
        if(cnt >= 7) usleep(100); /* 7 이상이면 100밀리초 동안 대기한다. */
        cnt++;
        printf("increase : %d\n", cnt) ;
        fflush(NULL);
        v(); /* post(), Semaphore 값 1 증가, 대기중인 thread가 있으면 깨움 */
    }
    return NULL;
}
```



```
void *ptheadP(void *arg) {    /* P 연산(감소)을 수행하기 위한 함수를 작성한다. */
    int i;

    for(i = 0; i < 10; i++) {
        p(); /* wait(), Semaphore 값을 1 감소, 0이되면 대기 */

        cnt--;
        printf("decrease : %d\n", cnt);
        fflush(NULL);
        usleep(100);    /* 100밀리초 간 기다린다. */
    }
    return NULL;
}
```



```
int main(int argc, char **argv)
{
    pthread_t ptV, ptP;      /* 스레드를 위한 자료형 */
    const char* name = "posix_sem";
    unsigned int value = 7;  /* 세마포어의 값 */

    /* 세마포어 열기 */
    sem = sem_open(name, O_CREAT, S_IRUSR | S_IWUSR, value);
    pthread_create(&ptV, NULL, pthreadV, NULL); /* 스레드 생성 */
    pthread_create(&ptP, NULL, pthreadP, NULL);
```



```
pthread_join(ptV, NULL); /* 스레드가 종료될 때까지 대기 */
```

```
pthread_join(ptP, NULL);
```

```
/* 다 쓴 세마포어 닫고 정리 */
```

```
sem_close(sem);
```

```
printf("sem_destroy() : %d\n", sem_destroy(sem));
```

```
/* 세마포어 삭제 */
```

```
sem_unlink(name);
```

```
return 0;
```

```
}
```




실행결과

```
$ gcc -o thread thread.c
```

```
$ ./thread
```

```
increase : 1
```

```
decrease : 0
```

```
increase : 1
```

```
increase : 2
```

```
increase : 3
```

```
increase : 4
```

```
increase : 5
```

```
increase : 6
```

```
increase : 7
```

```
decrease : 6
```

```
increase : 7
```

```
decrease : 6
```

```
increase : 7
```

```
decrease : 6
```

```
decrease : 5
```

```
decrease : 4
```

```
decrease : 3
```

```
decrease : 2
```

```
decrease : 1
```

```
decrease : 0
```

```
sem_destroy() : 0
```

```
$
```

강원혁신플랫폼

리눅스프로그래밍



스레드 간 동기화 문제





스레드 간의 동기화 문제

Linux에서 스레드의 역할

01 • 공유 리소스에 대한 액세스 조정

02 • 스레드 간의 활동을 동기화하는 데 사용할 수 있는
여러 동기화 메커니즘 제공

**경쟁 조건이나 기타 동시성 문제를 일으키지 않고
여러 스레드가 공유 데이터에 안전하게
액세스할 수 있음**



스레드 간의 동기화 문제

Linux에서 뮤텍스의 역할

01 • 스레드 동기화에 사용되어 코드의 중요한 섹션 보호

02 • 한 번에 하나의 스레드만 공유 리소스에
액세스할 수 있도록 함

03 • 경합 상태를 방지하고 스레드 간에 상호 배제를
제공함



\$./thread_no_mutex

Inc : 1 < Before

Dec : 1 < Before

Inc : 2 > After

Dec : 1 < After

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

int g_var = 1;
void *inc_function(void*); void
*dec_function(void*);

int main(int argc, char** argv)
{
    pthread_t ptInc, ptDec;
    pthread_create(&ptInc, NULL, inc_function, NULL);
    pthread_create(&ptDec, NULL, dec_function, NULL);
}
```

pthread_create() 함수 : 스레드 생성

증가를 위한 스레드

감소를 위한 스레드

```
void *inc_function(void*arg)
{
    /*~ 중간 생략 ~*/
    printf("Inc : %d < Before\n", g_var);
    g_var++;
    printf("Inc : %d > After\n", g_var);
    /*~ 중간 생략 ~*/
    return NULL;
}
```

```
void *dec_function(void*arg)
{
    /*~ 중간 생략 ~*/
    printf("Dec : %d < Before\n", g_var); g_var-
-;
    printf("Dec : %d > After\n", g_var);
    /*~ 중간 생략 ~*/
    return NULL;
}
```



스레드 동기화, Mutex





pthread_mutex_init(): 뮤텝스를 초기화하는 데 사용


```
#include <pthread.h>
```

```
int pthread_mutex_init(pthread_mutex_t *mutex, const  
pthread_mutexattr_t *attr);
```

④ pthread_mutex_destroy(): 뮤텝스 제거와 관련 리소스 해제에 사용

```
#include <pthread.h>
```

```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```


 pthread_mutex_lock(): 뮉텍스 잠금을 획득하는 데 사용되며, 다른 스레드가 잠금을 이미 보유하고 있으면 잠금이 사용 가능해질 때까지 호출 스레드가 차단

```
#include <pthread.h>
```

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

⊕ pthread_mutex_trylock(): 뮤텍스 잠금을 획득하려고 시도하는 데 사용되며 잠금을 획득하고 0을 반환하거나 잠금이 다른 스레드에서 이미 보유하고 있는 경우 0이 아닌 값을 반환

```
#include <pthread.h>
```

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



pthread_mutex_unlock(): 뮤텍스 잠금을 해제하여 다른 스레드가 이를 획득할 수 있도록 하는 데 사용

```
#include <pthread.h>
```

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



```
#include <stdio.h>
#include <pthread.h>
int g_var = 1;
pthread_mutex_t mid;

void *inc_function(void *);
void *dec_function(void *);

int main(int argc, char **argv) {
    pthread_t ptInc, ptDec;

    pthread_mutex_init(&mid, NULL); /* 뮤텍스초기화 */
```



```
pthread_create(&ptInc, NULL, inc_function, NULL);  
pthread_create(&ptDec, NULL, dec_function, NULL);  
pthread_join(ptInc, NULL);  
pthread_join(ptDec, NULL);  
  
pthread_mutex_destroy(&mid); /* 뮤텝스 삭제 */  
  
return 0;  
}
```



```
void *inc_function(void *arg)
{
    pthread_mutex_lock(&mid);      /* 임계 구역 설정 */
    printf("Inc : %d < Before\n", g_var);
    g_var++;
    printf("Inc : %d > After\n", g_var);
    pthread_mutex_unlock(&mid);    /* 임계 구역 해제 */

    return NULL;
}
```



```
void *dec_function(void *arg)
{
    pthread_mutex_lock(&mid);      /* 임계 구역 설정 */

    printf("Dec : %d < Before\n", g_var);
    g_var--;
    printf("Dec : %d > After\n", g_var);
    pthread_mutex_unlock(&mid);    /* 임계 구역 해제 */

    return NULL;
}
```



실행결과

```
$ ./thread_mutex
```

```
Inc : 1 < Before
```

```
Inc : 2 > After
```

```
Dec : 2 < Before
```

```
Dec : 1 > After
```

```
$ ./thread_no_mutex
```

```
Inc : 1 < Before
```

```
Dec : 1 < Before
```

```
Inc : 2 > After
```

```
Dec : 1 < After
```




01 • 멀티스레드의 주요 함수

02 • 스레드 간 동기화 문제

03 • 스레드 동기화의 주요 함수