5강 :
Pthread 프로그래밍

제공: LLNL POSIX 스레드 프로그래밍 튜토리얼을 기반으로 한 슬라이드

스레드란 무엇입니까?

OS에 의해 동시에 실행되는 독립적인 명령 스트림

멀티스레드 프로그램

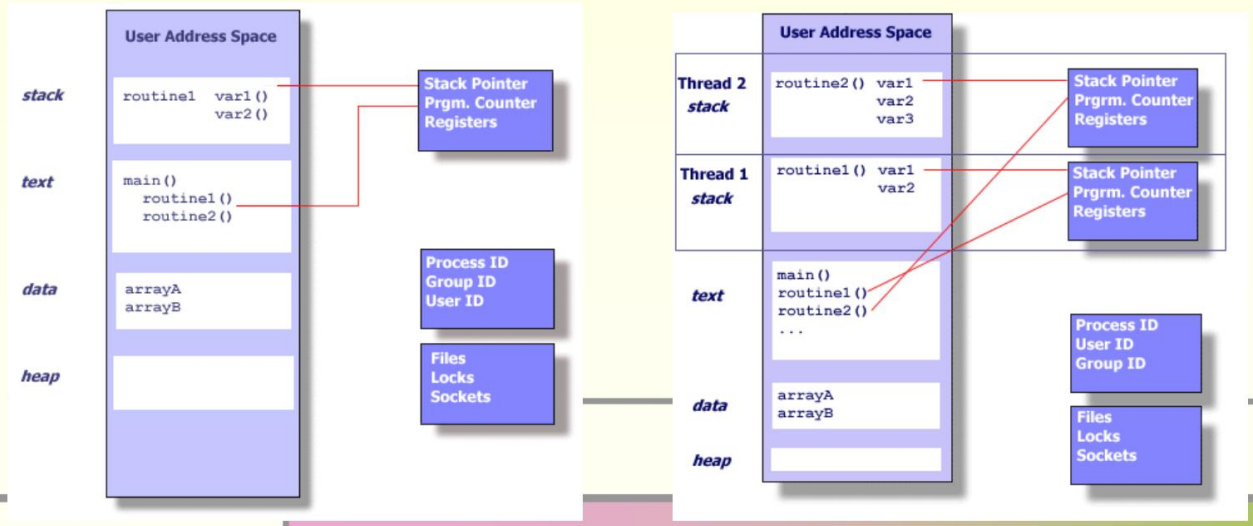
기본 프로그램에는 OS별로 동시에 독립적으로 실행되도록 예약된 여러 프로시저가 포함되어 있습니다.

프로세스와 스레드

스레드는 프로세스의 리소스를 공유합니다.

- 한 스레드의 변경 사항이 다른 스레드에 영향을 미칩니다.
- 동일한 값을 갖는 두 포인터가 동일한 데이터를 가리킴
- 동일한 메모리 위치에 읽고 쓰는 것은
- 가능하므로 프로그래머의 명시적인 동기화가 필요합니다.

프로세스는 리소스를 공유하지 않습니다.



스레드 속성

프로세스 내에 존재하며 프로세스 리소스를 사용합니다.

상위 프로세스가 존재하고 OS가 이를 지원하는 한 자체적으로 독립적인 제어 흐름 을 갖습니다.

독립적으로 예약하는 데 필요한 필수 리소스만 복제합니다.

작업을 수행하는 다른 스레드와 프로세스 리소스를 공유할 수 있습니다.
똑같이 독립적으로 (그리고 의존적으로)

상위 프로세스가 죽으면 죽습니다 - 또는 이와 유사한 것

간접비의 대부분이 이미 처리되었기 때문에 "경량"입니다.
프로세스 생성을 통해 달성됩니다.

프로세스 내의 모든 스레드는 동일한 주소 공간을 공유합니다. 따라서
프로세스 간 통신 보다 스레드 간 통신이 더 효율적입니다.

pthread

p스레드

POSIX 스레드

UNIX용 표준화된 C 언어 스레드 이식성을 위해 공유 메모리 다중 프로세서에서 작업 pthread를 사용하는 이유는 무엇입니까?

성능 향상 프로세스보다 적은 시스템 리소스 필요

fork()와 pthread_create() 비교 : 10~50회

포크() 대 pthread_create()

성능(50,000개 포크 또는 pthread_create)

Platform	fork()			pthread_create()		
	real	user	sys	real	user	sys
Intel 2.6 GHz Xeon E5-2670 (16 cores/node)	8.1	0.1	2.9	0.9	0.2	0.3
Intel 2.8 GHz Xeon 5660 (12 cores/node)	4.4	0.4	4.3	0.7	0.2	0.5
AMD 2.3 GHz Opteron (16 cores/node)	12.5	1.0	12.5	1.2	0.2	1.3
AMD 2.4 GHz Opteron (8 cores/node)	17.6	2.2	15.7	1.4	0.3	1.3
IBM 4.0 GHz POWER6 (8 cpus/node)	9.5	0.6	8.8	1.6	0.1	0.4
IBM 1.9 GHz POWER5 p5-575 (8 cpus/node)	64.2	30.7	27.6	1.7	0.6	1.1
IBM 1.5 GHz POWER4 (8 cpus/node)	104.5	48.6	47.2	2.1	1.0	1.5
INTEL 2.4 GHz Xeon (2 cpus/node)	54.9	1.5	20.8	1.6	0.7	0.9
INTEL 1.4 GHz Itanium2 (4 cpus/node)	54.5	1.1	22.2	2.0	1.2	0.6

P스레드 API

세 그룹 스레드

관리

스레드 생성 및 소멸

뮤텍스(상호 배제) 동기화

조건변수

공유하는 스레드 간의 통신

뮤텍스

스레드 관리

pthread_create(스레드, 속성, start_routine, arg)

pthread_exit(상태)

pthread_cancel(스레드)

스레드 생성

```
int pthread_create(
        pthread_t *스레드 제한, const
        pthread_attr_t *restrict attr, void *(*start_routine)
        (void *), void *restrict arg);
```

새로운 스레드를 생성하고 실행 가능하게 만듭니다.

생성 프로세스(또는 스레드)는 스레드 ID를 저장할 위치를 제공해야 합니다. 세 번째 매개변수는 스레드가 실행할 함수의 이름입니다.

마지막 매개변수는 인수에 대한 포인터입니다.

스레드 생성

새로운 스레드가 생성되면 동시에 실행됩니다.

만드는 과정과 함께. 스레드를 생성할 때

어떤 함수인지 표시합니다.

스레드가 실행되어야 합니다.

pthread_t 구조를 통해 반환된 스레드 핸들 기본 속성을 사용하려면 NULL 지정 함수에 단일 인수 전송 함수에 인수가 없으면 NULL 지정

오류 코드를 확인하세요!

스레드 종료

pthread를 종료하는 방법에는 여러 가지가 있습니다. 스레드는 시작 루틴(초기 스레드의 메인 루틴)에서 돌아옵니다.
실).

스레드는 pthread_exit 서브루틴 을 호출합니다 . pthread_cancel 루틴을 통해 다른 스레드에 의해 스레드가 취소됩니다.

exec() 또는 메소드 호출로 인해 전체 프로세스가 종료됩니다.
출구()

pthread_exit 자체를 명시적으로 호출하지 않고 main()이 먼저 완료되면

스레드 종료

```
void pthread_exit(void *value_ptr);
```

일반적으로 pthread_exit() 루틴은 스레드를 종료하기 위해 호출됩니다.

main()이 자신이 생성한 스레드보다 먼저 완료되고 pthread_exit()와 함께 종료되면 다른 스레드는 계속 실행됩니다.

그렇지 않으면 main()이 완료될 때 자동으로 종료됩니다 .

프로그래머는 선택적으로 종료 상태를 지정할 수 있습니다.

이는 호출 스레드에 참여할 수 있는 모든 스레드에 대한 void 포인터로 저장됩니다 . 정리: pthread_exit() 루틴은 파일을 닫지 않습니다. 어느 스레드 내부에서 열린 파일은 스레드가 종료된 후에도 열린 상태로 유지됩니다.

스레드 취소

한 스레드는 다음을 사용하여 다른 스레드가 종료되도록 요청할 수 있습니다.

pthread_cancel

int pthread_cancel(pthread_t 스레드);

pthread_cancel은 요청 후 반환됩니다.

예제 1: 스레드 생성

```
#include <pthread.h> #include
<stdio.h> #define NUM_THREADS
5

void *PrintHello(void *threadid) { int tid; tid = (int)threadid;
      printf("Hello
      World! 나야, 스레드 #%d!\n", tid);
      pthread_exit(NULL);

 }

 int main (int argc, char *argv[]) { pthread_t 스레드[NUM_THREADS];
      int rc, t; for(t=0; t<NUM_THREADS; t++){


           printf("메인: 스레드 %d 생성 중\n", t); rc = pthread_create(&threads[t], NULL,
           PrintHello, (void *)t); if (rc) { printf("오류 코드는 %d입니다\n", rc); 종료(-1);



           }

      } pthread_exit(NULL);
 }
```

예 2: 스레드에 매개변수 전달

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



void *PrintHello(void *ptr) {

        char *파일명; int j; 파일명 =
        (char *)
        ptr; while (1) { printf("Hello World! 나야, 스레
        드 %s!\n", filename);
                수면(1);



        } pthread_exit(NULL);
}

int main (int argc, char *argv[]) {

        pthread_t 스레드[100]; int err_code, i=0;
        char *파일명;



        printf("스레드를 생성하려면 언제든지 스레드 이름을 입력하세요\n");

        while (1) { 파일 이름 =
                (char *) malloc (80*sizeof(char)); scanf("%s", 파일명); printf("메인: %d 스레드 생성\n", i);
                err_code = pthread_create(&thread[i],
                NULL, PrintHello, (void *)filename); if (err_code){ printf("오류 코드는 %d입니다\n",
                err_code); 종료(-1); } 그렇지 않으면 나는++;









        } pthread_exit(NULL);
}
```

예 3: 파일

```
void *PrintHello(void *ptr) {

        파일 *파일; char *파일
        명; 문자 텍스트라인[100]; int j;



        파일명 = (char *) ptr; printf("Hello World! %s을
        (를) 여는 중입니다!\n", 파일 이름); file = fopen(파일명, "r"); if (파일 != NULL) {


            while(fscanf(file, "%s\n", textline) != EOF) printf("%s\n", textline);

        } fclose(파일);
}
```

합류

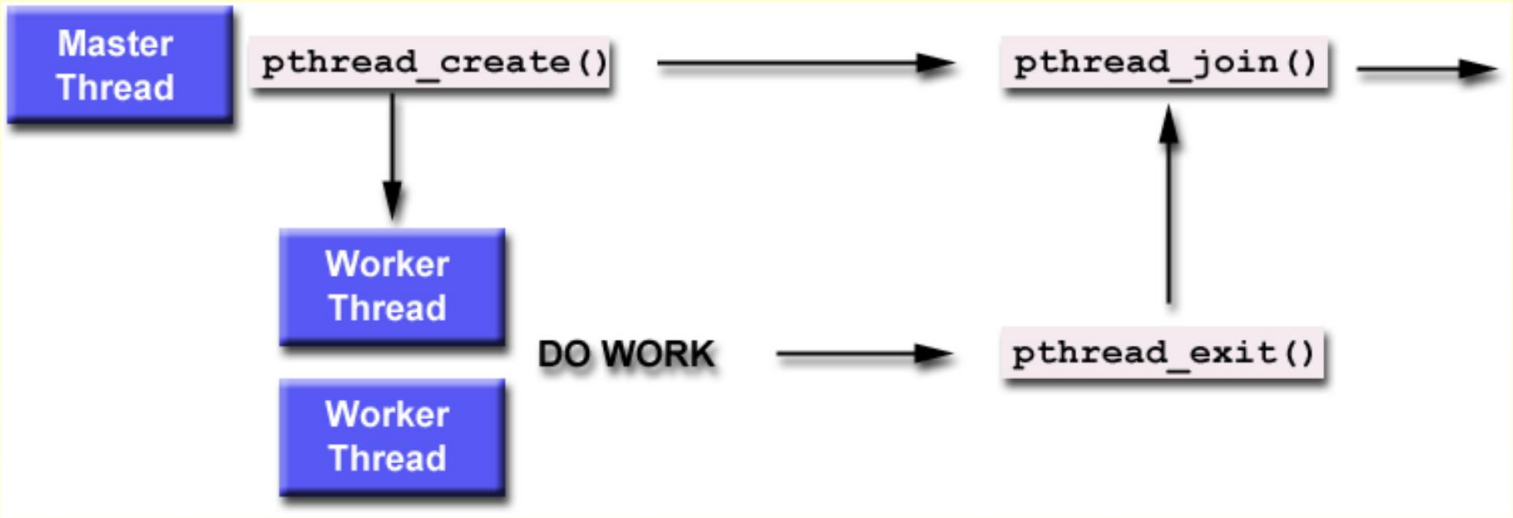
```
int pthread_join(pthread_t thread, void **value_ptr);
```

pthread_join() 서브루틴은 지정된 스레드가 종료될 때까지 호출 스레드를 차단합니다.

프로그래머는 대상 스레드의 pthread_exit() 호출에 지정된 경우 대상 스레드의 종료 반환 상태를 얻을 수 있습니다.

조인 스레드는 하나의 pthread_join() 호출과 일치할 수 있습니다. 동일한 스레드 에서 여러 조인을 시도하는 것은 논리적 오류 입니다 .

합류



예시 4: 가입

```
#include <pthread.h> #include
<stdio.h> #include <stdlib.h>
#include <math.h> #define
NUM_THREADS 4


void *BusyWork(void *t) {

        나는 int; 오
        랫동안; 이중 결과
        =0.0; tid = (긴)t; printf("%ld 스레드
        시작 중...\n",tid); (i=0;
        i<1000000; i++) {


                결과 = 결과 + sin(i) * tan(i);

        } printf("스레드 %ld가 완료되었습니다. 결과 = %e\n",tid, result); pthread_exit((void*) t);

}
```

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

        pthread_t 스레드[NUM_THREADS]; int rc; 긴 t; 무효 *상태;




        for(t=0; t<NUM_THREADS; t++) { printf("Main: 스레드
                %ld\n 생성 중", t); rc = pthread_create(&thread[t], &attr, BusyWork, (void *)t);
                if (rc) { printf("ERROR; pthread_create()의 반환 코드


                                        %d입니다\n", rc); 종료(-1); }


                }

        for(t=0; t<NUM_THREADS; t++) { rc = pthread_join(thread[t],
                &status); 만약 (rc) {

                        printf("오류; pthread_join()의 반환 코드는 %d입니다.\n", rc); 종료(-1);


                } printf("메인: 상태가 있는 스레드 %ld와의 조인이 완료되었습니다.
                                %ld\n",t,(long)상태);

        } printf("메인: 프로그램이 완료되었습니다. 종료합니다.\n"); pthread_exit(NULL);

}
```

뮤텍스

상호 배제

스레드 동기화를 구현하고 여러 쓰기가 발생할 때 공유 데이터를 보호합니다.

뮤텍스 변수는 공유 데이터 리소스에 대한 접근을 보호하는 "잠금" 역할을 합니다.

단 하나의 스레드만이 언제든지 뮤텍스 변수를 잠그거나 소유할 수 있습니다.
주어진 시간

경합 상태 방지를 위해 사용됩니다 .

여러 스레드가 뮤텍스를 놓고 경쟁할 때 패자는 해당 호출을 차단합니다.

"lock" 호출 대신 "trylock"을 사용하여 차단 해제 호출을 사용할 수 있습니다.

뮤텍스 루틴

pthread_mutex_init (뮤텍스, 속성)

pthread_mutex_destroy(뮤텍스)

뮤텍스 변수는 다음과 같이 선언되어야 합니다.

pthread_mutex_t를 입력하고 사용하기 전에 초기화해야 합니다.

뮤텍스 잠금/잠금 해제

pthread_mutex_lock(뮤텍스)

지정된 항목에 대한 잠금을 획득합니다. 뮤텍스 변하기 쉬운

pthread_mutex_trylock(뮤텍스)

뮤텍스를 잠그려고 시도합니다. 그러나 뮤텍스가 이미 잠겨 있으면 루틴은 "busy" 오류 코드와 함께 즉시 반환됩니다.

pthread_mutex_unlock(뮤텍스)

소유 스레드가 호출하면 뮤텍스를 잠금 해제합니다.

Mutex 사용에 대한 사용자의 책임

공유 데이터를 보호할 때는 프로그래머의 책임입니다.
뮤텍스를 사용해야 하는 모든 스레드가 이를 수행하는지 확인하는 책임입니다.

예를 들어 3개의 스레드가 동일한 내용을 업데이트하는 경우
데이터가 있지만 한두 개만 뮤텍스를 사용하면 데이터가 여전히 손상될 수 있습니다.

스레드 1 스레드 2 스레드 3
잠그다 잠그다
A = 2 A = A+1 A = A*B
잠금 해제 잠금 해제

예제 5: 뮤텍스

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

typedef 구조체
  {
      더블                    * ㅏ;
      더블 더블               *비;
                              합집합;
      int veclen;
  } 도트데이터;

#NUMTHRDS 4 정의
#VECLEN 100000 정의
      DOTDATA 도트str; pthread_t
      callThd[NUMTHRDS];
      pthread_mutex_t mutexsum;

무효 *dotprod(무효 *arg)
{
      int i, 시작, 끝, len ;
      긴 오프셋;
      이중 mysum, *x, *y;
      오프셋 = (긴)arg;

      len = dotstr.veclen;
      시작 = 오프셋*len;
      끝 = 시작 + 길이;
      x = dotstr.a; y = dotstr.b;
      마이섬 = 0;
      for (i=start; i<end ; i++) mysum += (x[i] * y[i]);
      pthread_mutex_lock(&mutexsum);
      dotstr.sum += mysum;
      printf("스레드 %ld가 %d에서 %d까지 수행했습니다: mysum=%f global sum=%f\n",offset,start,end,mysum,dotstr.sum);
      pthread_mutex_unlock(&mutexsum);
      pthread_exit((void*) 0);
}
```

예제 5: 뮤텍스

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

    오랫동안 나
    는; 더블 *a, *b; 무효 *상태;


    a = (double*) malloc (NUMTHRDS*VECLEN*sizeof(double)); b = (double*) malloc
    (NUMTHRDS*VECLEN*sizeof(double));

    for (i=0; i<VECLEN*NUMTHRDS; i++) { a[i]=1; b[i]=a[i];


    }

    dotstr.veclen = VECLEN; dotstr.a = a;
    dotstr.b = b;
    dotstr.sum=0;


    pthread_mutex_init(&mutexsum, NULL);

    for(i=0;i<NUMTHRDS;i++) pthread_create(&callThd[i], &attr, dotprod, (void *)i);

    for(i=0;i<NUMTHRDS;i++) pthread_join(callThd[i], &status); printf("합계 = %f \n", dotstr.sum);


    무료 (a); 무료 (b);

    pthread_mutex_destroy(&mutexsum); pthread_exit(NULL);

}
```

조건 변수

스레드를 동기화하는 또 다른 방법

뮤텍스

데이터에 대한 스레드 접근을 제어하여 동기화

조건변수

데이터의 실제 가치를 기반으로 동기화합니다.

조건 변수가 없으면 프로그래머는 조건이 충족되는지 확인하기 위해 스레드가 지속적으로 폴링(아마도 임계 섹션에서)하도록 해야 합니다.

항상 뮤텍스 잠금과 함께 사용됩니다.

조건 변수 루틴

pthread_cond_init(조건, 속성)

pthread_cond_destroy(조건)

조건 변수는 다음과 같이 선언되어야 합니다.

pthread_cond_t를 입력 하고 사용하기 전에 초기화해야 합니다.

속성 조건변수 속성을 설정하는데 사용됩니다.(NULL: 기본값)

pthread_cond_destroy()는 조건을 해제하는 데 사용해야 합니다.
더 이상 필요하지 않은 변수입니다.

조건 변수 루틴

pthread_cond_wait(조건,뮤텍스)

지정된 시간까지 호출 스레드를 차단합니다. 상태 신호됩니다.

이 루틴은 다음과 같은 동안 호출되어야 합니다. 뮤텍스 잠겨있습니다

기다리는 동안 자동으로 뮤텍스 잠금을 해제합니다.

신호를 수신하고 스레드가 깨어난 후 뮤텍스 될거야
사용을 위해 자동으로 잠김

pthread_cond_signal(조건)

조건 변수를 기다리고 있는 다른 스레드에 신호를 보내거나 깨웁니다.

pthread_cond_signal()을 호출하기 전에 호출하는 것은 논리적 오류입니다.
pthread_cond_wait()를 호출합니다.

pthread_cond_broadcast(조건)

둘 이상의 스레드가 차단 대기 상태에 있는 경우 pthread_cond_signal() 대신 을 사용해야 합니다.

예 6: 제어 변수

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


#define NUM_THREADS 3 #define TCOUNT
10 #define COUNT_LIMIT 12


정수 개수 = 0;
정수 thread_ids[3] = {0,1,2}; pthread_mutex_t
count_mutex; pthread_cond_t count_threshold_cv;


무효 *inc_count(void *t) {

    나는 int; 긴
    my_id = (long)t;

    for (i=0; i<TCOUNT; i++)
        { pthread_mutex_lock(&count_mutex); 카운트++;


        if (개수 == COUNT_LIMIT) {
            pthread_cond_signal(&count_threshold_cv); printf("inc_count(): 스레드 %ld,
            count = %d 임계값에 도달했습니다.\n", my_id, count);

        } printf("inc_count(): 스레드 %ld, 개수 = %d, 뮤텍스 잠금 해제\n", my_id, count); pthread_mutex_unlock(&count_mutex);


        수면(1);

    } pthread_exit(NULL);
}
```

```
void *watch_count(void *t) {

    긴 my_id = (long)t;

    printf("watch_count() 시작 중: 스레드 %ld\n", my_id); pthread_mutex_lock(&count_mutex); 동안(개수
    <COUNT_LIMIT) {

        pthread_cond_wait(&count_threshold_cv, &count_mutex); printf("watch_count(): 스레드 %ld 조건
        신호가 수신되었습니다.\n", my_id); 개수 += 125; printf("watch_count(): 스레드 %ld 개수는 이제 %d.\n", my_id, count);



    } pthread_mutex_unlock(&count_mutex); pthread_exit(NULL);

}

int main (int argc, char *argv[]) {

    int i, rc; 긴 t1=1,
    t2=2, t3=3; pthread_t 스레드[3];
    pthread_mutex_init(&count_mutex,
    NULL); pthread_cond_init(&count_threshold_cv, NULL);
    pthread_create(&threads[0], NULL, watch_count, (void *)t1);
    pthread_create(&threads[1], NULL, inc_count, (void *)t2); pthread_create(&threads[2], NULL,
    inc_count, (void *)t3);


    for (i=0; i<NUM_THREADS; i++) pthread_join(threads[i], NULL); printf("Main(): %d개의 스레드에서 대기했습니다. 완료되
    었습니다.\n", NUM_THREADS); pthread_mutex_destroy(&count_mutex); pthread_cond_destroy(&count_threshold_cv);
    pthread_exit(NULL);


}
```