챕터 4: 스레드 & 동시성

챕터 4: 스레드

- 개요멀티코어 프로그래밍멀
- 티스레딩 모델스레드 라이
- 브러리암시적 스레딩스레딩
- 문제운영 체제 예제
-
-
-

목표

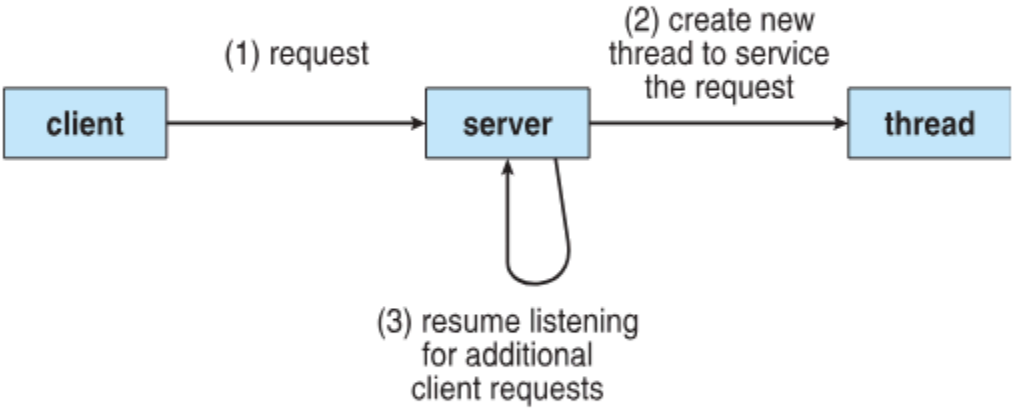
- 멀티스레드 컴퓨터 시스템의 기초를 형성하는 CPU 사용률의 기본 단위인 스레드의 개념을 소개합니다.Pthreads, Windows 및 Java 스레드 라이브러리의 API에 대해 논의합니다.암시적 스레딩을 제공하는 몇 가지 전략을 탐색합니다.멀티스레드 프로그래밍과 관련된 문제를 조사합니다.Windows 및 Linux의 스레드에 대한 운영 체제 지원을 다룹니다.

동기

- 대부분의 최신 응용 프로그램은 다중 스레드응용 프로그램 내에서 실행되
- 는 스레드응용 프로그램의 여러 작업을 별도의 스레드로 구현할 수 있습
- 니다.

  디스플레이 업데이트데이터 가져오기맞춤법 검사네트워크 요청에 응답프로세스 생성은 무겁고 스레드 생성은 가볍습니다.코드를 단순화하고 효율성을 높일 수 있습니다.커널은 일반적으로 다중 스레드입니다.

멀티스레드 서버 아키텍처



혜택

- 응답성 – 프로세스의 일부가 차단된 경우 지속적인 실행을 허용할 수 있음, 특히 사용자 인터페이스에 중요리소스 공유 – 스레드는 프로세스의 리소스를 공
- 유하며 공유보다 쉬움메모리 또는 메시지 전달경제 – 프로세스 생성보다 저렴, 스레드 스위칭보다 오버헤드가 낮음컨텍스트 스위칭확장성 – 프로세스는 다중
- 프로세서 아키텍처를 활용할 수 있습니다.

멀티코어 프로그래밍

- 프로그래머에게 부담을 주는 멀티코어 또는 멀티프로세서 시스템의 문제점은 다음과 같습니다.

  활동 나누기BalanceData 분할데이터 종속성테스트 및 디버깅

  Parallelism은 시스템이 둘 이상의 작업을 동시에 수행할 수 있음을 의미

  합니다동시성은 둘 이상의 작업을 지원하여 진행 중입니다

-
-
  - 단일 프로세서/코어, 동시성을 제공하는 스케줄러

멀티코어 프로그래밍(계속)

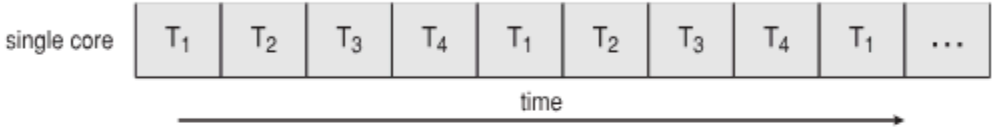
- 병렬 처리의 종류

  데이터 병렬 처리 – 동일한 데이터의 하위 집합을 여러 코어에 배포하고 각 코어에서 동일한 작업Task parallelism – 코어에 스레드를 분산하고 각 스레드는 고유한 작업을 수행합니다.스레드의 #이 증가함에 따라 스레딩에 대한 아키텍처 지원도 증가합니다.

-
  - CPU에는 코어와 하드웨어 스레드가 있습니다.8개의 코어와 코어당 8개의 하드
  - 웨어 스레드가 있는 Oracle SPARC T4를 고려하십시오.

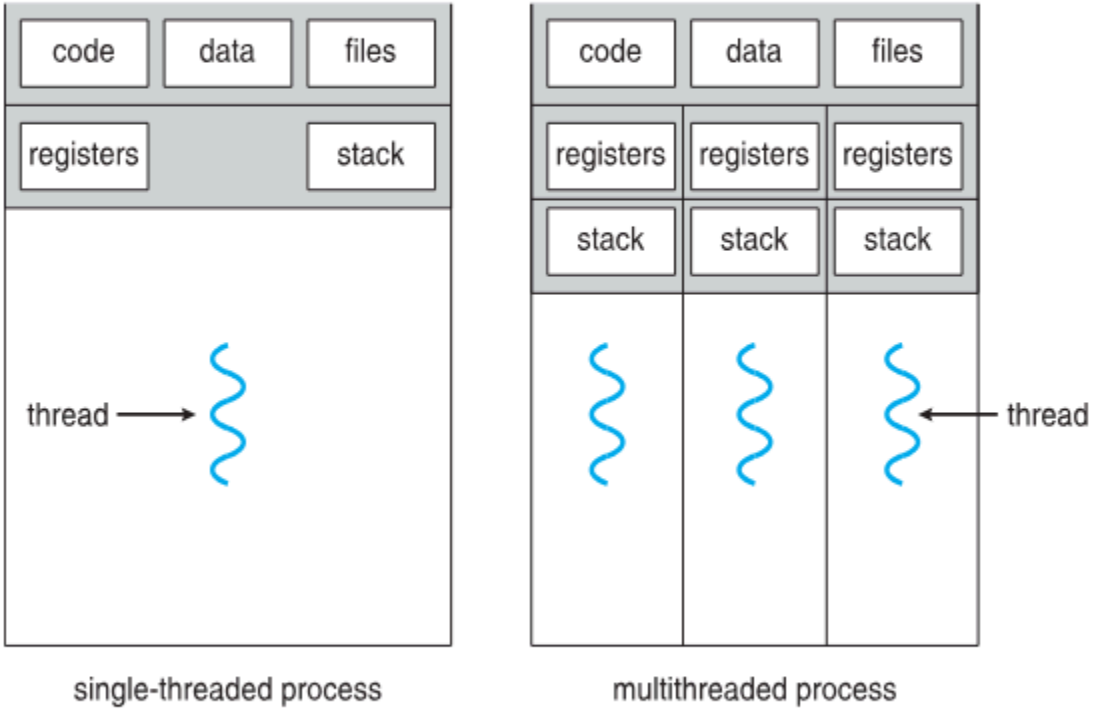
동시성 vs. 병렬 처리

- 단일 코어 시스템에서 동시 실행:



- 다중 코어 시스템에서의 병렬 처리:

단일 및 다중 스레드 프로세스



암달의 법칙

- 직렬 및 병렬 구성 요소가 모두 있는 응용 프로그램에 추가 코어를 추가하여 얻을 수 있는 성능 향상을 식별합니다.S는 직렬 부분N 처리 코어입니다.
-
-

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

- 즉, 응용 프로그램이 75% 병렬/25% 직렬인 경우 1코어에서 2코어로 이동하면 속도가 1.6배 빨라집니다.N이 무한대에 가까워지면 속도 향상이 1/S에 접근합니다
-

응용 프로그램의 직렬 부분은 추가 코어를 추가하여 성능 향상에 불균형한 영향을 미칩니다.

- 그러나 법은 현대의 멀티코어 시스템을 고려합니까?

사용자 스레드 및 커널 스레드

- **사용자 스레드 - 사용자 수준 스레드 라이브러리에서 수행하는 관리세 가지 기본 스레**
- **드 라이브러리:**

  POSIX PthreadsWindows 스레드Java 스레드커널 스레드 - KernelExamples에서 지원 – 다음을 포함한 거의 모든 범용 운영 체제:

  - 윈도우솔라
  - 리스리눅스
  - 트루64 유닉
  - 스맥 OS X

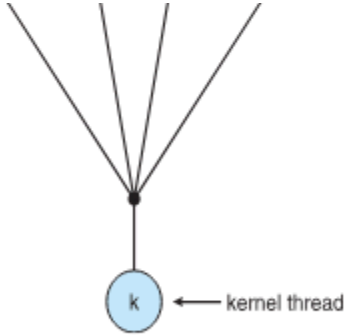
멀티스레딩 모델

- 다대일
- 일대일
- 다대다

다대일

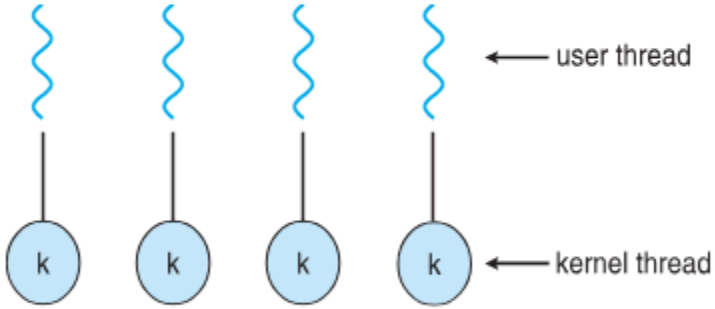
- 단일 커널 스레드에 매핑된 많은 사용자 수준 스레드하나의 스레드 차단으로 인
- 해 모든 스레드가 차단됨한 번에 하나의 커널에만 있을 수 있기 때문에 다중 스레
- 드는 muticore 시스템에서 병렬로 실행되지 않을 수 있습니다.현재 이 모델을 사용하는 시스템은 거의 없습니다.예:
-
-
  - **Solaris Green ThreadsGNU 휴대용 스레드**
  -

d



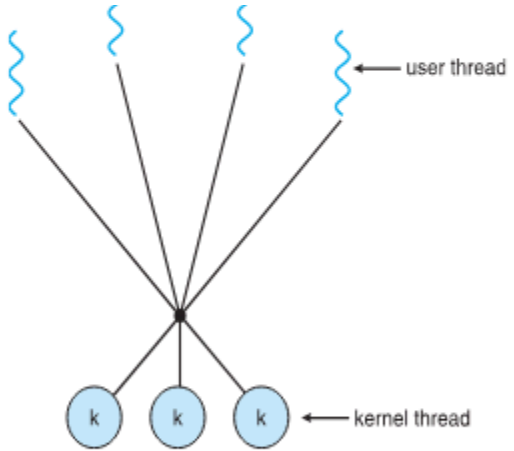
일대일

- 각 사용자 수준 스레드는 커널 스레드에 매핑됩니다사용자 수준 스
- 레드를 생성하면 커널 스레드가 생성됩니다.다대일보다 더 동시성
- 입니다.오버헤드로 인해 때때로 제한되는 프로세스당 스레드 수입
- 니다.예제
-
  - WindowsLinux
  - Solaris 9 이상
  -



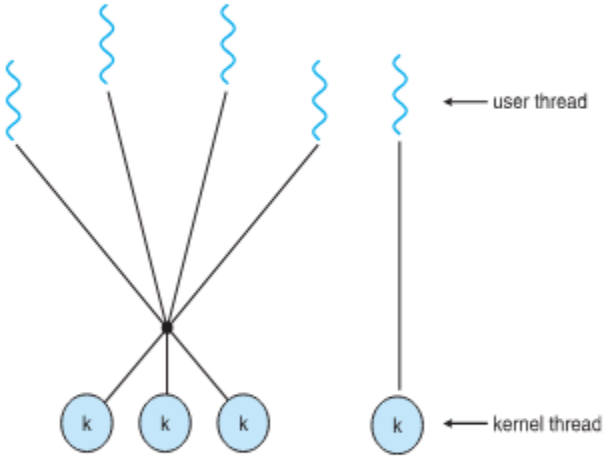
다대다 모델

- 많은 사용자 수준 스레드를 여러 커널 스레드에 매핑할 수 있습니다.운영
- 체제가 충분한 수의 커널 스레드를 생성할 수 있습니다.Solaris 버전 9 이
- 전 버전WindowsWindows ThreadFiber 패키지 사용
-



2단계 모델

- M:M과 유사하지만, 사용자 스레드를 커널 threadExamples 에 바인딩할 수 있다는
- 점이 다릅니다
  - IRIXHP-UXTru64
  - UNIXSolaris 8 이
  - 하



스레드 라이브러리

- **스레드 라이브러리는 프로그래머에게 스레드 생성 및 관리를 위한 API를 제공합니다.**
- **두 가지 주요 구현 방법**
  - 완전히 사용자 공간에 있는 라이브러리OS에서 지원하는 커널 수준 라이브러리
  -

Pthreads

- 사용자 수준 또는 커널 수준으로 제공 될 수 있습니다스레드 생성 및 동기화를위
- 한 POSIX 표준 (IEEE 1003.1c) API구현이 아닌 사양API는 스레드 라이브러리
- 의 동작을 지정하며 구현은 라이브러리 개발까지 가능합니다UNIX 운영 체제
- (Solaris, Linux, Mac OS X)에서 공통
-

Pthreads 예제

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

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* threads call this function */

int main(int argc, char *argv[])
{
   pthread_t tid; /* the thread identifier */
   pthread_attr_t attr; /* set of thread attributes */

   if (argc != 2) {
      fprintf(stderr,"usage: a.out <integer value>\n");
      return -1;
   }
   if (atoi(argv[1]) < 0) {
      fprintf(stderr,"%d must be >= 0\n",atoi(argv[1]));
      return -1;
   }
```

Pthreads 예제(계속)

```
   /* get the default attributes */
   pthread_attr_init(&attr);
   /* create the thread */
   pthread_create(&tid,&attr,runner,argv[1]);
   /* wait for the thread to exit */
   pthread_join(tid,NULL);

   printf("sum = %d\n",sum);
}

/* The thread will begin control in this function */
void *runner(void *param)
{
   int i, upper = atoi(param);
   sum = 0;

   for (i = 1; i <= upper; i++)
      sum += i;

   pthread_exit(0);
}
```

10개의 스레드를 결합하기 위한 Pthreads 코드

```
#define NUM_THREADS 10

/* an array of threads to be joined upon */
pthread_t workers[NUM_THREADS];

for (int i = 0; i < NUM_THREADS; i++)
   pthread_join(workers[i], NULL);
```

Windows 다중 스레드 C 프로그램

```
#include <windows.h>
#include <stdio.h>
DWORD Sum; /* data is shared by the thread(s) */

/* the thread runs in this separate function */
DWORD WINAPI Summation(LPVOID Param)
{
   DWORD Upper = *(DWORD*)Param;
   for (DWORD i = 0; i <= Upper; i++)
      Sum += i;
   return 0;
}

int main(int argc, char *argv[])
{
   DWORD ThreadId;
   HANDLE ThreadHandle;
   int Param;

   if (argc != 2) {
      fprintf(stderr,"An integer parameter is required\n");
      return -1;
   }
   Param = atoi(argv[1]);
   if (Param < 0) {
      fprintf(stderr,"An integer >= 0 is required\n");
      return -1;
   }
```

Windows 다중 스레드 C 프로그램(계속)

```
/* create the thread */
ThreadHandle = CreateThread(
   NULL, /* default security attributes */
   0, /* default stack size */
   Summation, /* thread function */
   &Param, /* parameter to thread function */
   0, /* default creation flags */
   &ThreadId); /* returns the thread identifier */

if (ThreadHandle != NULL) {
    /* now wait for the thread to finish */
   WaitForSingleObject(ThreadHandle,INFINITE);

   /* close the thread handle */
   CloseHandle(ThreadHandle);

   printf("sum = %d\n",Sum);
   }
}
```

자바 스레드

- Java 스레드는 기본 OSJava 스레드에서 제공하는 스레드 모델을 사용하
- 여 JVMTypically 구현되어 다음과 같이 생성될 수 있습니다.

```
public interface Runnable
{
    public abstract void run();
}
```

  - Thread 클래스 확장Runnable 인터페이스 구현

Java 멀티스레드 프로그램

```
class Sum
{
  private int sum;

  public int getSum() {
    return sum;
  }

  public void setSum(int sum) {
    this.sum = sum;
  }
}

class Summation implements Runnable
{
  private int upper;
  private Sum sumValue;

  public Summation(int upper, Sum sumValue) {
    this.upper = upper;
    this.sumValue = sumValue;
  }

  public void run() {
    int sum = 0;
    for (int i = 0; i <= upper; i++)
       sum += i;
    sumValue.setSum(sum);
  }
}
```

Java 멀티스레드 프로그램(계속)

```
public class Driver
{
  public static void main(String[] args) {
    if (args.length > 0) {
      if (Integer.parseInt(args[0]) < 0)
        System.err.println(args[0] + " must be >= 0.");
      else {
        Sum sumObject = new Sum();
        int upper = Integer.parseInt(args[0]);
        Thread thrd = new Thread(new Summation(upper, sumObject));
        thrd.start();
        try {
           thrd.join();
           System.out.println
                   ("The sum of "+upper+" is "+sumObject.getSum());
      } catch (InterruptedException ie) { }
      }
    }
    else
      System.err.println("Usage: Summation <integer value>"); }
}
```

암시적 스레딩

- 스레드의 수가 증가함에 따라 인기가 높아짐에 따라 명시 적 스레드로 인해 프로그램 정확성이 더 어려워짐프로그래머가 아닌 컴파일러와 런타임 라이브러리에
- 의해 수행되는 스레드의 생성 및 관리세 가지 방법을 탐구했습니다.

-

  스레드 풀OpenMPGrand
  Central Dispatch기타 메
  소드포함

- 마이크로소프트 스레딩 건물 블록 (미정), java.util.concurrent 패키지

스레드 풀

- 작업을 기다리는 풀에 여러 스레드를 만듭니다.장점:
  - 일반적으로 새 스레드를 만드는 것보다 기존 스레드로 요청을 서비스하는 것이 약간 더 빠르다응용 프로그램의 스레드 수를 풀의 크기에 바인딩 할 수 있습니다.작업 생성 메커니즘에서 수행 할 작업을 분리하면 작업 실행을위한 다른 전략을 사용할 수 있습니다.
  - ☒ 즉, 작업이 주기적으로 실행되도록 예약 할 수 있음Windows API는 스레드 풀을 지원합니다.

```
DWORD WINAPI PoolFunction(AVOID Param) {
    /*
    * this function runs as a separate thread.
    */
}
```

오픈MP

- 컴파일러 지시문 집합 및 APIfor C, C++, FORTRAN공유 메모리 환경에서 병렬 프로그래밍에 대한 지원 제공
- 병렬로 실행할 수 있는 코드 블록인 병렬 영역을 식별합니다.

```
#pragma OMP 병렬
```

코어가 있는 만큼 스레드를 만듭니다.

```
#pragma OMP 병렬
    for(i=0; 나<엔; i++) {
c[i] = a[i] + b[i];}
```

for 루프를 병렬로 실행

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

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

   #pragma omp parallel
   {
      printf("I am a parallel region.");
   }

   /* sequential code */

   return 0;
}
```

그랜드 센트럴 디스패치

- Mac OS X 및 iOS 운영 체제용 Apple 기술 C, C++ 언어,
- API 및 런타임 라이브러리 확장병렬 섹션 식별 가능스레딩의
- 대부분의 세부 사항 관리블록은 "^{ }" - ^{ printf("I am a
- block"); }디스패치 대기열에 배치된 블록
-
-
  - 큐에서 제거될 때 스레드 풀의 사용 가능한 스레드에 할당됩니다.

그랜드 센트럴 디스패치

- 두 가지 유형의 디스패치 대기열:
  - serial – FIFO 순서로 제거된 블록, queue는 mainqueue라고 하는 프로세스별 입니다.

    프로그래머는 programconcurrent 내에서 추가 직렬 큐를 생성할 수 있습니다
  - – FIFO 순서에 따라 제거되지만 한 번에 여러 개가 제거될 수 있습니다.

    ☒ 우선 순위가 낮음, 기본, 높음인 3개의 시스템 전체 대기열

```
dispatch_queue_t queue = dispatch_get_global_queue
  (DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

dispatch_async(queue, ^{ printf("I am a block."); });
```

스레딩 문제

- fork() 및 exec() 시스템 호출의 의미신호 처리
- 대상 스레드의 Synchronous 및 asynchronousThread 취소
- 비동기 또는
- deferredThread-local
- storageScheduler 활성화

fork()와 exec()의 의미론

- fork ()는 호출 스레드 만 복제합니까 아니면 모든 스레드도 복제합니까?
- 일부 UNIX에는 두 가지 버전의 forkexec()가 있으며 일반적으로 정상적으로 작동합니다 – 모든 스레드를 포함하여 실행 중인 프로세스를 교체합니다

신호 처리

- **신호는 UNIX 시스템에서 특정 이벤트가 발생했음을 프로세스에 알리는 데 사용됩니다.**
- 신호 처리기는 신호를 처리하는 데 사용됩니다
  1. 신호가 특정 이벤트에 의해 생성됩니다.
  2. 신호가 프로세스로 전달됩니다.
  3. 신호는 두 가지 신호 처리기 중 하나에 의해 처리됩니다.
     1. 기본값
     2. 사용자 정의
- 모든 신호에는 커널이 신호를 처리할 때 실행되는 기본 핸들러가 있습니다
  - **사용자 정의 신호 처리기는 기본값을 재정의할 수 있습니다.**
  - 단일 스레드의 경우 프로세스에 전달되는 신호

신호 처리(계속)

- 멀티스레드를 위해 신호를 어디에 전달해야 합니까?
  - 신호가 적용되는 스레드에 신호를 전달합니다.
  - 프로세스의 모든 스레드에 신호 전달
  - 프로세스의 특정 스레드에 신호 전달
  - 프로세스에 대한 모든 신호를 수신하도록 특정 스레드를 할당합니다.

스레드 취소

- 완료되기 전에 스레드 종료스레드를 취소
- 하는 것은 대상 스레드입니다.두 가지 일
- 반적인 방법은 다음과 같습니다.

  비동기 취소는 대상 스레드를 즉시 종료합니다지연된 취소를 사용하면 대상 스레드가 취소해야 하는지 주기적으로 확인할 수 있습니다.Pthread 코드를 사용하여 스레드를 만들고 취소합니다.

```
pthread_t tid;

/* create the thread */
pthread_create(&tid, 0, worker, NULL);

. . .

/* cancel the thread */
pthread_cancel(tid);
```

스레드 취소(계속)

- 스레드 취소를 호출하면 취소가 요청되지만 실제 취소는 스레드 상태에 따라 달라집니다

Mode	State	Type
Off	Disabled	–
Deferred	Enabled	Deferred
Asynchronous	Enabled	Asynchronous

- 스레드에서 취소를 사용할 수 없는 경우 취소는 thread가 활성화될 때까지 보류 상태로 유지됩니다.Default 형식이 지연됩니다
-
  - 취소는 스레드가 취소 지점에 도달할 때만 발생합니다

  □ 즉 pthread_testcancel()□ 그런 다음 정리 핸들러가 호출됩니다.Linux 시스템에서는 스레드 취소가 신호를
- 통해 처리됩니다.

스레드-로컬 스토리지

- **TLS(Thread-local Storage)를 사용하면 각 스레드가 고유한 데이터 복사본**
- **을 가질 수 있습니다.스레드 생성 프로세스를 제어할 수 없는 경우(즉, 스레드 풀을 사용할 때)로컬 변수와 다름**
-

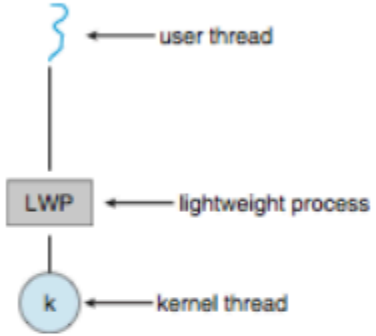
  단일 함수 호출 중에만 표시되는 로컬 변수함수 호출에서 볼 수 있는 TLS정적 데이터와 유사
-
  - TLS는 각 스레드에 고유합니다.

스케줄러 활성화

- M : M 및 2 레벨 모델 모두 적절한 수를 유지하기 위해 통신이 필요합니다.응용 프로그램에 할당 된 커널 스레드일반적으로 사용자와 커널 스레드 사이의 중
- 간 데이터 구조를 사용합니다 - 경량 프로세스 (LWP)

프로세스가 사용자 스레드를 실행하도록 예약 할 수있는 가상 프로세서 인 것 같습니다.커널 스레드에 연결된 각 LWP를 생성하려면 몇 개의 LWP를 만들 수 있습니까? 스케줄러 활성화는 업콜을 제공합니다 - 커널에서

- 스레드 라이브러리의 업콜핸들러로의 통신 메커니즘이 통신을 통해 응용 프로그램은 올바른 수의 커널 스레드를 유지할 수 있습니다
-



운영 체제 예제

- Windows 스레드Linux 스레드
-

Windows 스레드

- Windows는 Win 98, Win NT, Win2000, Win XP 및 Win 7에 대한 기본 API인 Windows API를 구현합니다.일대일 매핑, 커널 수준을 구현합니다.각 스레드에
- 는 다음이 포함됩니다.
-

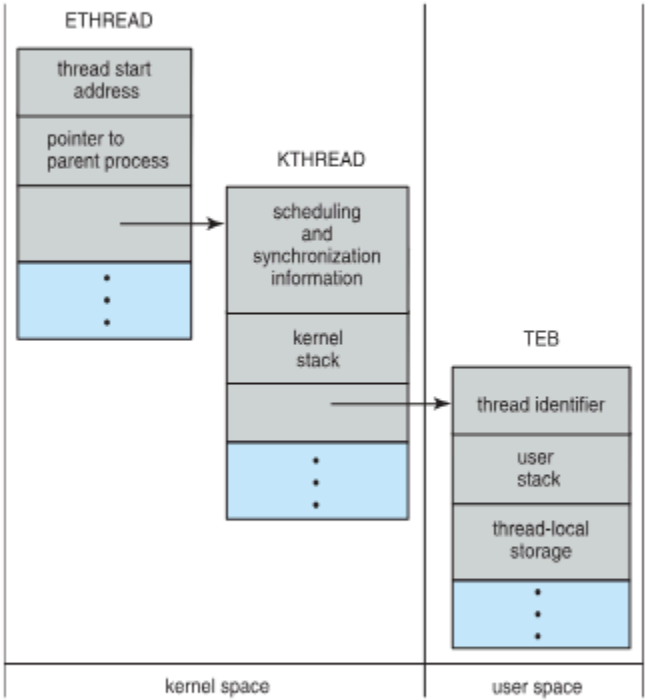
  스레드 ID프로세서의 상태를 나타내는 레지스터 집합스레드가 사용자 모드 또는 커널 모드에서 실행될 때를 위해 사용자 스택과 커널 스택을 분리한다.런타임 라이브러리 및 동적 링크 라이브러리(DLL)에서 사용하는 개인 데이터 저장 영역레지스터 세트, 스택 및 개인 저장 영역을 스레드의 컨텍스트라고 합니다.

-

Windows 스레드(계속)

- 스레드의 기본 데이터 구조는 다음과 같습니다.
  - ETHREAD (실행 스레드 블록) – 커널 spaceKTHREAD (커널 스레드 블록) – 스케줄링 및 동기화 정보, 커널 모드 스택, TEB 포인터, 커널 spaceTEB
  - (스레드 환경 블록) – 스레드 ID, 사용자 모드 스택, 스레드 로컬 스토리지, 사용자 공간

Windows 스레드 데이터 구조



리눅스 스레드

- Linux는 스레드를 통해 작업이라고 합니다.스레드 생성은 clone()을 통해 수행됩
- 니다.system callclone()을 사용하면 자식 작업이 부모 작업(프로세스)의 주소
- 공간을 공유할 수 있습니다.
  - 플래그는 동작을 제어합니다.

flag	meaning
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.

- struct task_struct 데이터 구조(공유 또는 고유)를 처리하기 위한 지점입니다.

챕터 4의 끝