操作系统原理

第五章：线程管理

洪明坚

重庆大学软件学院

February 19, 2016

目录

Outline

Process revisited

Process revisited

- The process is

Process revisited

- The process is
  - a unit of resource allocation;

Process revisited

- The process is
  - a unit of resource allocation;
  - a unit of dispatching (scheduling).

Process revisited

- The process is
  - a unit of resource allocation;
  - a unit of dispatching (scheduling).
- Traditionally, a process has only one thread of control.

Process revisited

- The process is
  - a unit of resource allocation;
  - a unit of dispatching (scheduling).
- Traditionally, a process has only one thread of control.
  - If we separate the above two concepts and allow multiple threads of control within one process, we get the threads.

Process revisited

- The process is
  - a unit of resource allocation;
  - a unit of dispatching (scheduling).
- Traditionally, a process has only one thread of control.
  - If we separate the above two concepts and allow multiple threads of control within one process, we get the threads.
  - That is, processes are used to group resources together;

Process revisited

- The process is
  - a unit of resource allocation;
  - a unit of dispatching (scheduling).
- Traditionally, a process has only one thread of control.
  - If we separate the above two concepts and allow multiple threads of control within one process, we get the threads.
  - That is, processes are used to group resources together; threads are the entities dispatched (scheduled) for execution on the CPU.

Thread (1/2)

Thread (1/2)

- A thread is a basic unit of CPU utilization in modern operating system.

Thread (1/2)

- A thread is a basic unit of CPU utilization in modern operating system.
  - Also known as **lightweight process (LWP)**.

Thread (1/2)

- A thread is a basic unit of CPU utilization in modern operating system.
  - Also known as **lightweight process (LWP)**.
- **Multi-threading**

Thread (1/2)

- A thread is a basic unit of CPU utilization in modern operating system.
  - Also known as **lightweight process (LWP)**.
- **Multi-threading**
  - Allowing multiple threads in the same process.

Thread (1/2)

- A thread is a basic unit of CPU utilization in modern operating system.
  - Also known as **lightweight process (LWP)**.
- **Multi-threading**
  - Allowing multiple threads in the same process.
  - They share resources belonging to the same process, such as its code section, data section, open files, etc.

Thread (1/2)

- A thread is a basic unit of CPU utilization in modern operating system.
  - Also known as **lightweight process (LWP)**.
- **Multi-threading**
  - Allowing multiple threads in the same process.
  - They share resources belonging to the same process, such as its code section, data section, open files, etc.
  - But, each thread within one process has **a private thread context (including the CPU register set and other state information) and a private stack**.

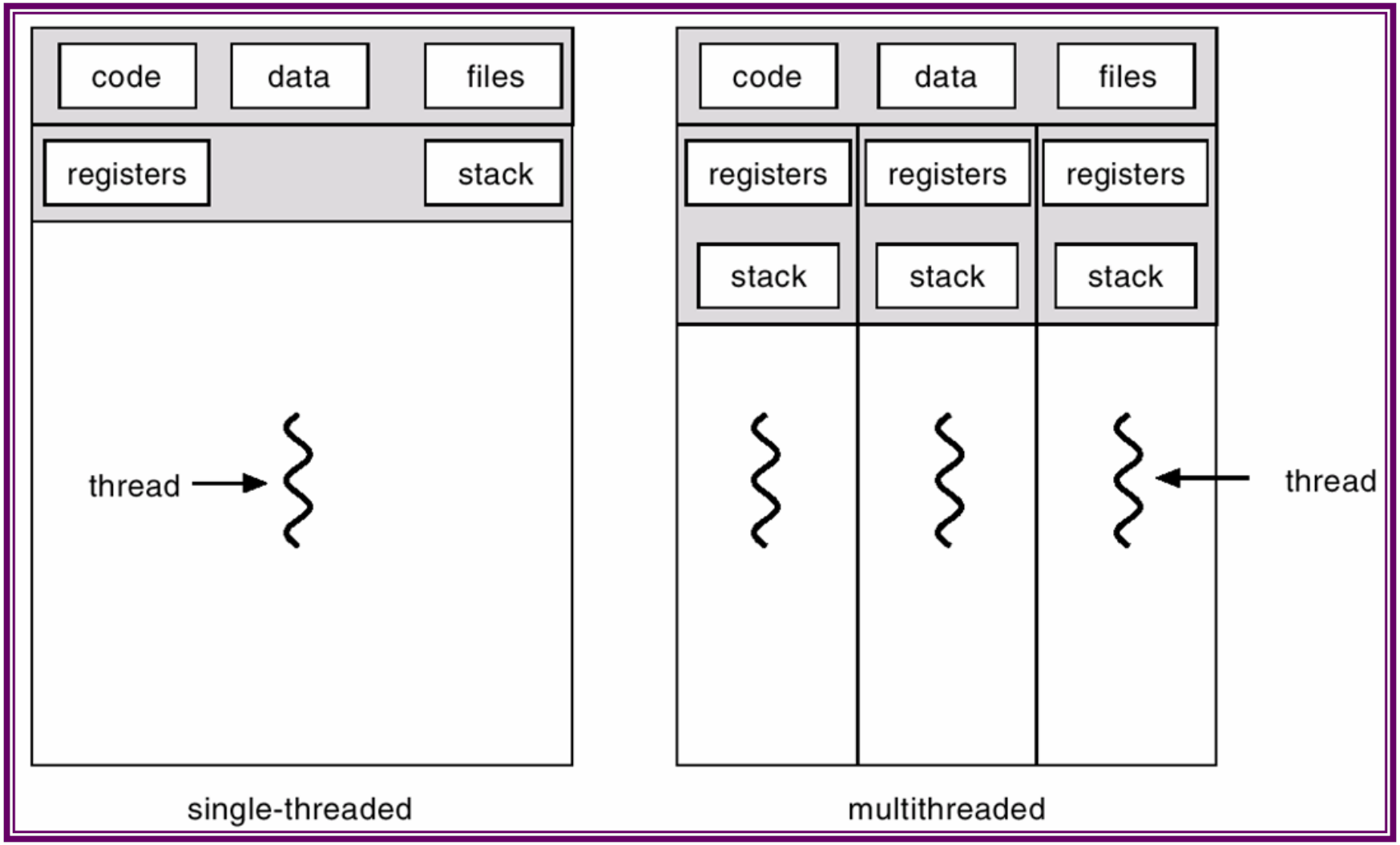
Thread (2/2)

Thread (2/2)

Per process items	**Per thread items**
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

Single- and multi-threaded process

Single- and multi-threaded process



Benefits of thread

Benefits of thread

- Responsiveness

Benefits of thread

- Responsiveness
  - Allows other threads to continue responding to the user even if one or several threads is blocked or is performing a lengthy operation.

Benefits of thread

- Responsiveness
  - Allows other threads to continue responding to the user even if one or several threads is blocked or is performing a lengthy operation.
- Resource sharing

Benefits of thread

- Responsiveness
  - Allows other threads to continue responding to the user even if one or several threads is blocked or is performing a lengthy operation.
- Resource sharing
  - Since threads within the same process share memory and files, they can communicate with each other without invoking the kernel.

Benefits of thread

- Responsiveness
  - Allows other threads to continue responding to the user even if one or several threads is blocked or is performing a lengthy operation.
- Resource sharing
  - Since threads within the same process share memory and files, they can communicate with each other without invoking the kernel.
- Economy

Benefits of thread

- Responsiveness
  - Allows other threads to continue responding to the user even if one or several threads is blocked or is performing a lengthy operation.
- Resource sharing
  - Since threads within the same process share memory and files, they can communicate with each other without invoking the kernel.
- Economy
  - Takes MUCH less time and resource to create a new thread than a process.

Benefits of thread

- Responsiveness
  - Allows other threads to continue responding to the user even if one or several threads is blocked or is performing a lengthy operation.
- Resource sharing
  - Since threads within the same process share memory and files, they can communicate with each other without invoking the kernel.
- Economy
  - Takes MUCH less time and resource to create a new thread than a process.
  - Takes MUCH less time to context switch threads within the same process.

Benefits of thread

- Responsiveness
  - Allows other threads to continue responding to the user even if one or several threads is blocked or is performing a lengthy operation.
- Resource sharing
  - Since threads within the same process share memory and files, they can communicate with each other without invoking the kernel.
- Economy
  - Takes MUCH less time and resource to create a new thread than a process.
  - Takes MUCH less time to context switch threads within the same process.
- Utilization of multiprocessor architectures

Benefits of thread

- Responsiveness
  - Allows other threads to continue responding to the user even if one or several threads is blocked or is performing a lengthy operation.
- Resource sharing
  - Since threads within the same process share memory and files, they can communicate with each other without invoking the kernel.
- Economy
  - Takes MUCH less time and resource to create a new thread than a process.
  - Takes MUCH less time to context switch threads within the same process.
- Utilization of multiprocessor architectures
  - Parallelism is possible by assigning each CPU a thread.

Questions

- Any questions?

Outline

Thread implementation

Thread implementation

- Multi-threading can be implemented

Thread implementation

- Multi-threading can be implemented
  - at user space for **user threads**;

Thread implementation

- Multi-threading can be implemented
  - at user space for **user threads**;
  - at the kernel for **kernel threads**.

Thread implementation

- Multi-threading can be implemented
  - at user space for **user threads**;
  - at the kernel for **kernel threads**.
  - with a hybrid scheme by combining user- and kernel- threads.

User threads (1/2)

User threads (1/2)

- It's implemented outside of the kernel as a **thread library** at the user space.

User threads (1/2)

- It's implemented outside of the kernel as a **thread library** at the user space.
  - It's the library which provides support for thread creation, scheduling and management.

User threads (1/2)

- It's implemented outside of the kernel as a **thread library** at the user space.
  - It's the library which provides support for thread creation, scheduling and management.
  - As far as the kernel is concerned, it's managing ordinary, single-threaded process.

User threads (1/2)

- It's implemented outside of the kernel as a **thread library** at the user space.
  - It's the library which provides support for thread creation, scheduling and management.
  - As far as the kernel is concerned, it's managing ordinary, single-threaded process.
- Examples

User threads (1/2)

- It's implemented outside of the kernel as a **thread library** at the user space.
  - It's the library which provides support for thread creation, scheduling and management.
  - As far as the kernel is concerned, it's managing ordinary, single-threaded process.
- Examples
  - Mach **C-threads**

User threads (1/2)

- It's implemented outside of the kernel as a **thread library** at the user space.
  - It's the library which provides support for thread creation, scheduling and management.
  - As far as the kernel is concerned, it's managing ordinary, single-threaded process.
- Examples
  - Mach **C-threads**
  - Solaris 2 **UI-threads**

User threads (2/2)

User threads (2/2)

- Advantages of the user threads

User threads (2/2)

- Advantages of the user threads
  - Thread management and context switch need not to trap to the kernel.

User threads (2/2)

- Advantages of the user threads
  - Thread management and context switch need not to trap to the kernel.
    - This will save a lot of CPU cycles.

User threads (2/2)

- Advantages of the user threads
  - Thread management and context switch need not to trap to the kernel.
    - This will save a lot of CPU cycles.
  - Allow each process to have its own customized scheduling algorithm.

User threads (2/2)

- Advantages of the user threads
  - Thread management and context switch need not to trap to the kernel.
    - This will save a lot of CPU cycles.
  - Allow each process to have its own customized scheduling algorithm.
- Disadvantages of the user threads

User threads (2/2)

- Advantages of the user threads
  - Thread management and context switch need not to trap to the kernel.
    - This will save a lot of CPU cycles.
  - Allow each process to have its own customized scheduling algorithm.
- Disadvantages of the user threads
  - Any user-level thread performing a blocking system call will cause the entire process to block.

User threads (2/2)

- Advantages of the user threads
  - Thread management and context switch need not to trap to the kernel.
    - This will save a lot of CPU cycles.
  - Allow each process to have its own customized scheduling algorithm.
- Disadvantages of the user threads
  - Any user-level thread performing a blocking system call will cause the entire process to block.
    - Even if other threads are ready to run within the process.

User threads (2/2)

- Advantages of the user threads
  - Thread management and context switch need not to trap to the kernel.
    - This will save a lot of CPU cycles.
  - Allow each process to have its own customized scheduling algorithm.
- Disadvantages of the user threads
  - Any user-level thread performing a blocking system call will cause the entire process to block.
    - Even if other threads are ready to run within the process.
  - On a system with multiprocessors, the user-level threads cannot be dispatched for execution in parallel.

Kernel threads

Kernel threads

- It's supported directly by the operating system.

Kernel threads

- It's supported directly by the operating system.
  - The kernel performs thread creation, scheduling and management in kernel space.

Kernel threads

- It's supported directly by the operating system.
  - The kernel performs thread creation, scheduling and management in kernel space.
- Examples

Kernel threads

- It's supported directly by the operating system.
  - The kernel performs thread creation, scheduling and management in kernel space.
- Examples
  - Windows NT/XP

Kernel threads

- It's supported directly by the operating system.
  - The kernel performs thread creation, scheduling and management in kernel space.
- Examples
  - Windows NT/XP
  - Solaris

Kernel threads

- It's supported directly by the operating system.
  - The kernel performs thread creation, scheduling and management in kernel space.
- Examples
  - Windows NT/XP
  - Solaris
- Advantages and disadvantages

Kernel threads

- It's supported directly by the operating system.
  - The kernel performs thread creation, scheduling and management in kernel space.
- Examples
  - Windows NT/XP
  - Solaris
- Advantages and disadvantages
  - Inverse of the user threads.

Questions

- Any questions?

Outline

Multi-threaded application programming interface

[1]POSIX or "Portable Operating System Interface for uniX" is the collective name of a family of related standards specified by the IEEE to define the application programming interface (API) for software compatible with variants of the Unix operating system

Multi-threaded application programming interface

- **Pthreads**

[1] POSIX or "Portable Operating System Interface for uniX" is the collective name of a family of related standards specified by the IEEE to define the application programming interface (API) for software compatible with variants of the Unix operating system

Multi-threaded application programming interface

- **Pthreads**
  - Pthreads refers to the POSIX[1] standard (IEEE 1003.1c) defining an API for thread creation and synchronization.

[1] POSIX or "Portable Operating System Interface for uniX" is the collective name of a family of related standards specified by the IEEE to define the application programming interface (API) for software compatible with variants of the Unix operating system

Multi-threaded application programming interface

- **Pthreads**
  - Pthreads refers to the POSIX[1] standard (IEEE 1003.1c) defining an API for thread creation and synchronization.
    - It's a *specification* for thread behavior, NOT an *implementation*.

[1] POSIX or "Portable Operating System Interface for uniX" is the collective name of a family of related standards specified by the IEEE to define the application programming interface (API) for software compatible with variants of the Unix operating system

Multi-threaded application programming interface

- **Pthreads**
  - Pthreads refers to the POSIX[1] standard (IEEE 1003.1c) defining an API for thread creation and synchronization.
    - It's a *specification* for thread behavior, NOT an *implementation*.
- Win32

[1] POSIX or "Portable Operating System Interface for uniX" is the collective name of a family of related standards specified by the IEEE to define the application programming interface (API) for software compatible with variants of the Unix operating system

Multi-threaded application programming interface

- **Pthreads**
  - Pthreads refers to the POSIX[1] standard (IEEE 1003.1c) defining an API for thread creation and synchronization.
    - It's a *specification* for thread behavior, NOT an *implementation*.
- Win32
  - CreateThread, ExitThread and TerminateThread, etc.

[1]POSIX or "Portable Operating System Interface for uniX" is the collective name of a family of related standards specified by the IEEE to define the application programming interface (API) for software compatible with variants of the Unix operating system

Multi-threaded application programming interface

- **Pthreads**
  - Pthreads refers to the POSIX[1] standard (IEEE 1003.1c) defining an API for thread creation and synchronization.
    - It's a *specification* for thread behavior, NOT an *implementation*.
- Win32
  - CreateThread, ExitThread and TerminateThread, etc.
  - POSIX Threads for Win32 (http://sources.redhat.com/pthreads-win32)

[1] POSIX or "Portable Operating System Interface for uniX" is the collective name of a family of related standards specified by the IEEE to define the application programming interface (API) for software compatible with variants of the Unix operating system

Example: Pthreads

Example: Pthreads

```c
#include <pthread.h>

int sum; /*shared variable*/

/*the thread function*/
void *runner(void *arg);

int main()
{
   pthread_t tid; /*thread ID*/
   pthread_attr_t attr; /*attributes*/

   /*get default attributes*/
   pthread_attr_init(&attr);

   /*create the thread*/
   pthread_create(&tid,&attr,runner,82);

   /*wait for the thread to exit*/
   pthread_join(tid,NULL);

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

```c
void *runner(void *arg)
{
   int i, upper=(int)arg;
   sum=0;
   if(upper>0)
      for(i=1;i<=upper;i++)
         sum+=i;

   pthread_exit(0);
   //or return (void *)0;
}
```

Example: Win32

Example: Win32

```
#include <windows.h>

int sum; /*shared variable*/

/*the thread function*/
DWORD WINAPI runner(LPVOID arg);

int main()
{
  DWORD tid; /*the thread ID*/
  HANDLE hThr; /*the thread handle*/

  /*create the thread*/
  hThr=CreateThread(0,0,runner,82,0,&tid
      );

  /*wait for the thread to exit*/
  WaitForSingleObject(hThr,INFINITE);

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

```
DWORD WINAPI runner(LPVOID
    arg)
{
    int i, upper = (int)arg
        ;
    sum = 0;
    if(upper > 0)
        for(i = 1; i <=
            upper; i++)
            sum += i;

  ExitThread(0);
  // or return 0L;
}
```

Questions

- Any questions?