CS2106 L5: Thread

Mu Changrui

May 10, 2021

1 Motivation of thread

1. Process creation under the fork() model(Duplicate memory space and Duplicate most of the process context etc) (Process is expensive)
2. Context switch: Requires saving/restoration of process Information
3. hard for independent processes to communicate with each other(Independent memory space: Inter-process communication needed)
4. In multi-core era, to let a process be executed in multiple core, the way is to make multiple control flow(threads)

The multiple threads can work on different code(task parallelism), or the same code work on different data(data parallelism)

2 Basic idea

1. raditional process: a single thread of control(Only one instruction of the whole program is executing at any time)
2. Multi-thread: multiple threads of control to the same process(i.e. Multiple parts of the programs is executing at the same time conceptually)

3 The resources that can be shared by multiple threads

1. PC: no, three control flow, so three program counter to follow each control flow
2. General purpose register: every thread needs independent access to the register for common function(otherwise they will interfere each other)

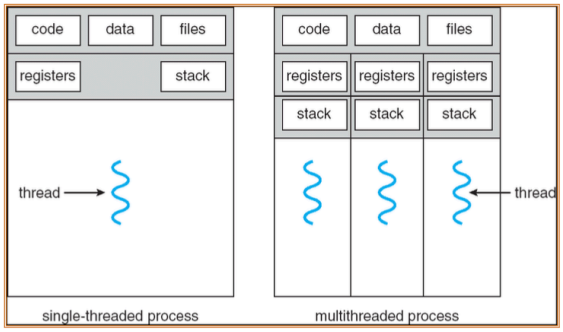
3. Text(code) segment: Yes(They will share the same code but execute different parts)
4. Data segment: Yes, data and heap are shared data
5. stack: cannot be shared, because the stack is used for control flow of functions
6. files: shared memory
7. process ID: usually different thread ID(depends on implementation); many system did not make a clear distinction between the process ID and thread ID

In general
The threads share:

1. Memory Context: Text, Data, Heap
2. OS Context: Process id, other resources like files, etc

Unique information needed for each thread:

1. Identification (usually thread id)
2. Registers (General purpose and special)
3. "Stack" (more about this later)(In linux: each stack will be given a reasonable stack size)



,

4 Advantage of threads

1. Economy:Multiple threads in the same process requires much less resources to manage compared to multiple processes
2. Resource sharing: Threads share most of the resources of a process; No need for additional mechanism for passing information around
3. Responsiveness: Multithreaded programs can appear much more responsive
4. Scalability:M ultithreaded program can take advantage of multiple CPUs

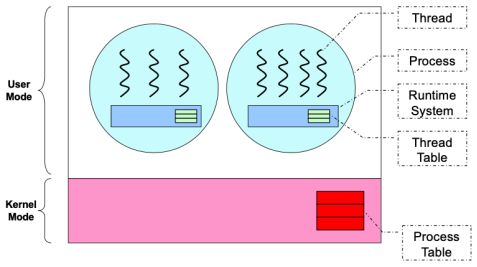
5 Problem of thread

1. Synchronization around shared memory much worse, there may be problems: all memory except the stack is shared between threads
2. System Call Concurrency: Parallel execution of multiple thread makes parallel system call possible (Have to guarantee correctness and determine the correct behavior)(e.g. what if one thread call fork(), will it create a new multi-thread process?) (The reaction of the thread called system call is system dependent)
3. Process Behavior: Impact on process operations
4. if the tasks involves executing potentially buggy code, multi-process is better

6 implementation of thread

6.1 User Thread:

Thread is implemented as a user library(A runtime system (in the process) will handle thread related operation)
Kernel is not aware of the threads in the process

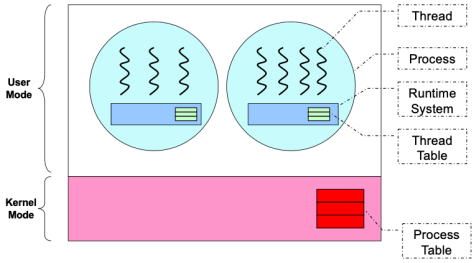


The user thread: process get one thread from the OS, then the process uses some user library and needs to run some code to switch between threads, the OS does know there is any switch within the process; In this case, because the process just call local library, it can apply any scheduling algorithm

6.2 Kernel Thread

1. Thread is implemented in the OS(Thread operation is handled as system calls)
2. Thread-level scheduling is possible: Kernel schedule by threads, instead of by process
3. Kernel may make use of threads for its own execution

OS is slower, but more supportive



,

The OS knows about each threads in the process, the OS can switch these thread and separate them on different core

6.2.1 Kernel thread vs multi-thread kernel

Kernel thread: thread implemented by OS
multi-thread kernel: the kernel makes use of threads for its own execution

7 Kernel thread vs User kernel

7.1 Pro and Con of user thread

7.1.1 Advantage

1. Can have multi-threaded program on ANY OS(multi-core or single core)
2. Thread operations are just library calls
3. Generally more configurable and flexible(Customized thread scheduling policy)

7.1.2 Disadvantage

1. OS is not aware of threads, scheduling is performed at process level, which means one thread blocked then the whole process is blocked
2. cannot exploit multiple CPUs(the OS only gives one process anyway)

7.2 Pro and Con of kernel thread

7.2.1 Advantages

Kernel can schedule on thread levels: More than 1 thread in the same process can run simultaneously on multiple CPUs

7.2.2 Disadvantages

1. Thread operations is now a system call, Slower and more resource intensive
2. Generally less flexible: If implemented with many features: expensive, overkill for simple program

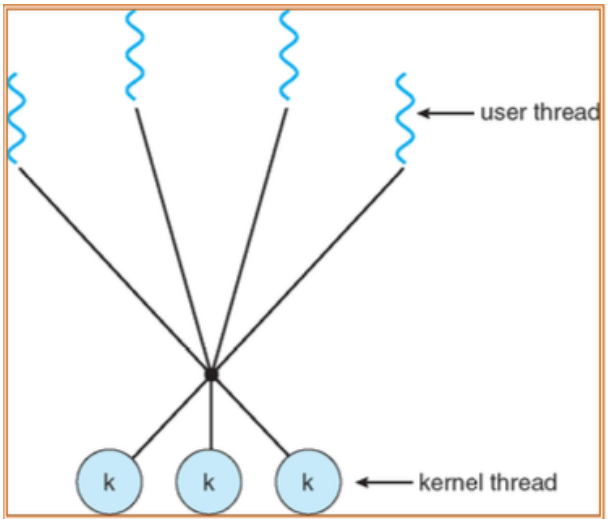
On one core machine, no matter user thread or kernel thread, multithread is slower than sequential implementation. because of the more context-switch

8 Hybrid Thread Model

Have both Kernel and User threads

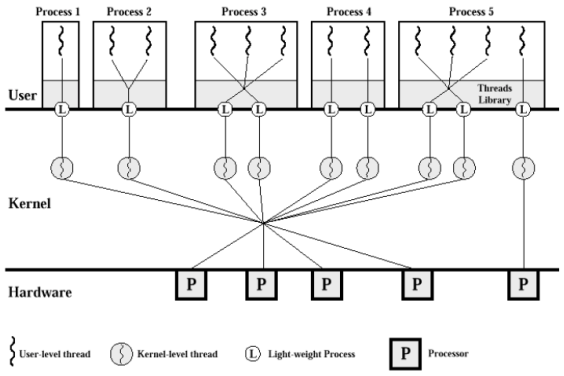
1. OS schedule on kernel threads only
2. User thread can bind to a kernel thread

Offer great flexibility: can limit the concurrency of any process / user



,

There are three kernel threads, and the four user threads share the three kernel threads

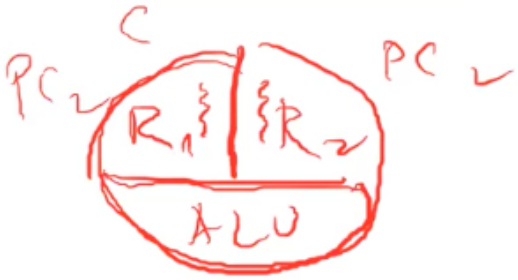


,

Solaris's hybrid thread model

9 hardware support on modern processors

Multiple sets of registers (GPRs, and special registers) are supplied to allow threads to run natively and in parallel on the same core
This is known as simultaneous multi-threading (SMT)
(e.g.Hyperthreading on Intel processor)



,

The modern hardware support for multi-thread
In the core, there is one ALU, there r two/or more set of (pc, registers) for execution of the core

10 POSIX Threads