Threads SMP Microkernels

Arnav Gupta

April 11, 2024

Contents

1 Processes and Threads

Main characteristics of a process are:

- resource ownership
  - space to hold process info
  - control over resources
- scheduling/execution
  - process follows trace through program
  - has state and priority

1.1 Multithreading

Multithreading: ability of the OS to support multiple, concurrent paths of execution within a single process

In a multithreaded environment, a process has:

- virtual address space for process image (resource allocation)
- protected access to resources (protection)

Threads each have:

- state (running, ready, etc)
- saved thread context when not running
- execution stack
- static storage for local variables
- access to the memory and resources of process (shared with other threads)

When one thread alters memory, other threads see the results if and when they access that item. One thread's access permissions cascade to other threads in the same process.

Key benefits of threads:

1. faster to create thread in a process than a new process
2. faster to terminate a thread than a process
3. less time to switch between threads within a process than between processes
4. communication between threads more efficient

Threads are useful for:

1. splitting foreground/background work
2. asynchronous processing
3. speed of execution
4. modular program structure

Scheduling and dispatching is done on threads, not processes.

1.2 Thread Functionality

Key states for a thread are running, ready, and blocked (no suspend).

Four basic thread operations associated with a change in thread state:

1. **spawn**: caused by process spawn or another thread, then placed on ready queue (register context and stack space)
2. **block**: when waiting for an event
3. **unblock**: when event occurs, moved to ready queue
4. **finish**: thread completes and is deallocated

With multiple threads, blocked threads can wait simultaneously, for threads within the same process and in different processes.

Must synchronize the activity of various threads so that they do not interfere with each other.

2 Types of Threads

2.1 User-Level and Kernel-Level Threads

For pure user-level:

- all thread management done by the application and the kernel is not aware of them
- done using threads libraries that manage thread control and state

Advantages to user-level threads over kernel-level threads are:

1. no need for kernel-mode privilege to switch threads, saves overhead of mode switching
2. scheduling algorithm can be tailored to the application
3. user-level threads can run on any OS without knowledge of kernel

Disadvantages of user-level threads to kernel-level threads are:

1. any system call blocks all threads in the process
2. multithreaded application cannot take advantage of multiprocessing, only application-level multiprogramming (no synchronization)

Workarounds include:

- using multiple processes, but this increases overhead
- jacketing: convert a blocking system call into a non-blocking system call

For pure kernel-level:

- all thread managemenet done by the kernel
- application just uses an API to the kernel thread facility

Advantages of kernel-level threads over user-level threads are:

1. overcoming disadvantages of user-level threads
2. kernel routines can be multithreaded

Disadvantages of kernel-level threads to user-level threads are:

1. requires more mode switching (order of magnitude difference)

Benefits of user-level threads and kernel-level threads depend on the nature of the applications involved.

Combined system:

- thread creation done in user space
- most scheduling and synchronization of threads done within an application
- user-level threads mapped to kernel-level threads
- combines advantages of pure user-level and pure kernel-level, while minimizing disadvantages

2.2 Other Arrangements

Many-to-many relationship between threads and processes

- multiple threads in a process
- thread can be performed in multiple address spaces

One-to-many relationship between threads and processes

- thread is the unit of activity, can move across address spaces
- useful for distributed computing

3 Multicore and Multithreading

3.1 Performance of Software on Multicore

Performance benefits of multicore organization depend on the ability to exploit parallel resources.

Amdalhl's Law

$$\text{Speedup} = \frac{\text{time to execute program on single processor}}{\text{time to execute program on } N \text{ parallel processors}} = \frac{1}{(1-f)+\frac{f}{N}}$$

Small amount of serial code can reduce speedup, while parallelization introduces overhead for communication and distribution of work.

Applications that benefit from the ability to scale throughput with number of cores:

- Multithreaded native applications: have small number of highly threaded processes
- Multiprocess applications: have many single-threaded processes
- Java applications: JVM is multithreaded
- Multi-instance applications: allows for isolation and security