Operating System: Threading

Alex Chi

Update: March 12, 2020

Contents

1 Thread

- basic unit of CPU utilization
- contains a thread ID, program counter, registers, stack
- share address space with other threads in the same address, including:
  - code
  - data
  - file handlers
- refer to figures in slides

2 Motivation

- multiple threads within application
- an application can do a number of different things
- a personal note: we can also use event-driven I/O to achieve concurrency

3 Benefits

- provides concurrency inside process
- share resource (address space)
- much faster than creating process
- thread in one process can be scheduled on multiple cores

4 Drawbacks

- data race
  - unpredictable results
  - I/O racing
- job has to be divided
  - complexity in programming
  - debugging is hard

    - personal note: This also occurs in making an algorithm run on cluster. For example, you have to make your model into a map-reduce fashion before running it on cluster.
- cache coherence

5 Comparison with Process

- subset of process
- share process state, memory, etc.
- share address space
- more ways to communicate
- faster context switch

6 Supports for Threads

- kernel-level thread
    - for example, each xv6 process has a kernel thread for doing syscalls, handle interrupts, etc.
- user-level thread
    - POSIX pthread
    - Win32 thread
    - Java thread

7 Thread Model

- refer to figure in slide
- kernel thread - light weight process - user thread
- multiple user threads inside one process
- user threads can be put into light weight processes
- thread library schedules user threads

8 Multi-threading Models

- n-to-1 (Solaris Green Threads, GNU Portable Threads, no concurrency)
- 1-to-1 (Windows NT, Linux, Solaris 9+, concurrency, more overhead)

- n-to-n (Windows NT with ThreadFiber)
    - maybe this is done by moving LWP into kernel space?
- two-level (n-to-n and allow 1-to-1 binding)

9 pthreads

- either in user-level or kernel-level
- POSIX standard API
- developers implement the API with specifications
- refer to code snippet in slides

```
void* runner(void* param) {
  // do something
  printf("%d\n", *(int*)param);
  pthread_exit(0);
}

int main() {
  pthread_t tid;
  int param = 1;
  pthread_create(&tid, NULL, runner, &param);
  pthread_join(tid, NULL);
  return 0;
}
```

10 Thread issues

1. Semantics of `fork()` and `exec()`
    - `fork()`
        - duplicate only the calling thread
        - duplicate all threads
        - OS provide two kinds of fork `fork` and `clone`
        - you can specify what to duplicate
    - `exec()`
        - replace the entire process image
2. Thread Cancellation

- asynchronous cancellation: cancel immediately
- deferred cancellation: target periodically check if should be canceled

3. Signal Handling
    - deliver to the thread where the signal applies
    - deliver to every thread
    - deliver to certain threads
    - assign a thread to receive all signals
4. Thread Pools
    - create a number of threads in a pool
    - reduce overhead for creating and destroying kernel thread
5. Scheduler

11 Examples

11.1 Linux

- `fork()` and `clone()`

11.2 Windows

- 1-to-1 mapping
- register set, stacks, private storage area are known as context
- data structures
    - ETHREAD
    - KTHREAD
    - TEB

12 Q&A

1. How can we define sequence of thread running?
   This is done by programmer. But there's no easy abstraction for this. You may implement this based on synchronization mechanisms, for example, use mutex.
2. Distinguish between Process and Thread
   In terms of concurrency, they both provide means of running multiple tasks at the same time. But threads reside in process.

3. How is n-to-n mapping implemented?

   For example, M kernel threads, N user-level threads