Operating Systems

Karim Lounis

Queen’s University, Canada

Winter’19, Jan-Apr 2019
Threads or Lightweight processes
Threads

Definition

**Thread**: It is a lightweight process.

Lightweight: Does not need to allocate additional memory space.
- The process that creates threads shares its memory space, heap, code, resources (e.g., open files), user/group ID, ...
- Threads only require dedicated registers and memory stack.
- Thread context-switching is very fast.

Similar to processes:
- Threads are identified by a TID (Thread IDentifier).
- Each thread is associated to a data structure called TCB (Thread Control Block) containing: thread identifier, thread state, priority, thread registers content including SP & PC, and a pointer to the original process PCB.

  *e.g.*, in Linux: `$top -H -p 55130`. 
Threads vs Processes

Assuming you have a process that creates another process (child process) by invoking the system call `fork()`:
Threads vs Processes

By executing `fork()` the OS duplicates the PCB, allocates a new memory space, and updates the PCB data fields:

```c
int a, b, c;
void main() {
    int x, y, pid; P;
    x = "x",
    p = (int*) malloc(sizeof(int));
    *p = 10;
    P = fork();
}
```
Threads vs Processes

Assuming you have a process as the main thread that creates another thread (child thread) e.g., Thread T = new Thread(); T.start();
Threads vs Processes

By executing the statements Thread T = new Thread(); T.start();
Threads

- Older process model used to be composed of only one thread (single flow of execution). We talk about **single-threaded processes**.

- Modern processes are **multithreaded processes** i.e., the process can perform more than one task at a time (multiple flows of execution).

- Asynchronous and Synchronous Threading.

包袱 New challenges for system designers and application programmers.
Threads

Most modern OS kernels are now multithreaded e.g., Linux use a kernel thread for managing the amount of free memory in the system.

Advantages of Threads

- **Responsiveness**: If a thread is blocked, the other one can continue running.
- **Resource sharing**: No need for interprocess communication. Threads of a same process share code and data.
- **Economy**: We don’t need to allocate a separate memory space.
- **Scalability**: utilization of multiprocessors, whereas a single-thread uses only one core.

Executing a multithreaded application on a single-core CPU system is meaningless (fake concurrency ≠ (real concurrency = parallelism)).

Modern CPUs can run more than 1 thread* per CPU core (hyperthreaded processors) e.g., Intel i5-2410M, has 4 cores, runs 2 threads per core.
# Threads vs Processes

You can see how many threads your processor is able to execute at a time:

```
root@Human-Device:~# lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    2
Core(s) per socket:    2
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 42
Model name:            Intel(R) Core(TM) i5-2410M CPU @ 2.30GHz
Stepping:              7
CPU MHz:               798.231
CPU max MHz:           2900.0000
CPU min MHz:           800.0000
BogoMIPS:              4599.81
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              3072K
NUMA node0 CPU(s):     0-3
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
                        pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp lm constant_tsc
                        arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq
                        qdtse64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic popcnt
                        tsc_deadline_timer aes xsave avx lahf_lm epb pti sbd ibrs ibpb stibp tpr_shadow vnmi
                        flexpriority ept vpid xsaveopt dtherm ida arat pln pts flush_l1d
```
Threads Scheduling

Traditionally We used to work with:
- Non-multithreaded operating systems.
- Process is the basic unit of CPU utilization.
- Scheduler deals with processes.
- Full context switching.

Nowadays We work with:
- Thread-aware operating systems.
- Thread is the basic unit of CPU utilization.
- Scheduler deals with threads.
- Full and partial context switching.
Threads level

Threads are often implemented in two levels:

**User-level threads** The OS is not aware about the presence of multiple threads in one process (Many-to-one threading model):
- A threading library is used to perform threads context switching.
- Process memory will be used for threads stack allocation.
- System calls from thread may block the main process (all threads).
- The OS schedules the process as a single-threaded process entity.
- Internal thread scheduling (lighter, flexible, and portable).

**Kernel-level threads** The OS is aware of the concept of threads and offers system calls to create and manages threads (one-to-one threading model):
- System calls are used to create physical threads.
- Each user thread is mapped to one kernel thread.
- Limited by the number of created threads.
- Use multiple CPU core at a time.
DMA (Direct Memory Access)

DMA or Direct Memory Access
DMA (Direct Memory Access)

Each I/O device has a dedicated device controller, which is a hardware component that interfaces b/w the device and the computer.

Each I/O device is connected to a computer via a plug whereas the device controller is connected via a socket.

Sometimes, one device controller handles multiple I/O devices.
DMA (Direct Memory Access)

A device Controller receives data from a connected device and stores it temporarily in some special purpose registers (i.e. local buffer).

  e.g., Network traffic coming from LAN.

Also each device has a dedicated driver, which is a software code that interfaces b/w the device controller and the OS.

The driver contains interrupt handlers that service the interrupts that are generated by that device controller.

  Interrupt handlers registration
DMA (Direct Memory Access)

**Problem to solve:** Large amount of data need to be transferred from main memory (i.e., RAM) to a I/O devices or the reverse.

**Old solution.** The use of polling:
- CPU periodically sends polls to I/O devices asking for any event.
- I/O device controllers respond with a positive acknowledgment.
- I/O starts sending data to CPU which then sends them to memory.

**Better solution.** The use of interrupts:
- I/O devices request the CPU to perform an I/O operation with RAM.
- The CPU will have to wait for the device to read/write data.
- There is a speed mismatch and lot of CPU cycles are lost.
- The CPU is not meant to wait but to constantly execute instructions.

**DMA principle:** Allow I/O devices to directly access the memory without [involving] the CPU (in particular I/O devices that do large transfers).
DMA (Direct Memory Access)

Many device controllers use DMA to transfer data from or into memory: HDD controllers, graphic cards, network cards, and sound cards.

This is implemented by a special purpose processor called DMA controller (a.k.a., DMAC).

Motorola MC6844P (left) and Intel 8237A-4 (right)
Because there is only one system bus (address bus, control bus, and data bus), technically, DMA controller can operate in one of the following modes:

1. **Burst mode**: The CPU grants DMA controller access over the system bus which then occupies the system bus for transferring a complete block of data ($\lambda$-Bytes) before releasing the buses back to the CPU.

2. **Cycle stealing mode**: The CPU grants DMA controller access over the system bus which then occupies the system bus for transferring one byte of data before the system bus is taken over by the CPU. In the meantime, the DMAC keeps asking for the buses.
DMA (Direct Memory Access)

At the same time, there are two types of DMA controllers:

1. **Third-party DMA:** A.k.a., standard DMA, in which the DMA engine resides on the main system board.

2. **First-party DMA:** A.k.a., bus mastering, in which the DMA engine resides on the device board.
Example of First-party DMA transfer from device (e.g., Floppy disk) into memory:

- Device driver is instructed to transfer data from device to memory.
  
  transfer one Byte from disk at $@_d$ to RAM at $@_m$

- Device driver instructs the device controller to transfer $\lambda$-Bytes to RAM.

- The device controller places a request for transfer to the DMA controller (a.k.a., DRQ).

- The DMA controller requests the CPU for the system bus (a.k.a., HRQ).

- The CPU delegates the buses to the DMA controller (a.k.a., HLDA).
Example of First-party DMA transfer from device into memory:

- The DMA controller seize the system bus (CPU is momentarily prevented from accessing RAM).

  At this time, the CPU can or cannot execute instructions.

- The DMA controller sets up the memory controller and informs the device controller to start the data transfer.

- Upon reception, the device controller starts sending each byte of data to the memory.

- For each transferred byte, the DMA controller byte counter is decreased and its internal address register increased.

- Once done (counter=0), the DMA interrupts the CPU to signal a DMA transfer completion so that the CPU resumes.
End