Operating Systems

Tutorial 1

Michael Tänzer

os-tut@nhng.de
http://os-tut.nhng.de

30th November 2010

Outline

1. Review
2. Scheduling in SMP Systems
3. Interprocess Communication
   - Design Parameters
   - Problems with IPC
   - Emulating asynchronous or synchronous IPC
   - L4 IPC

True or False

- When using round-robin starvation can never happen
- Shortest job first provides the minimal turnaround time among all non-preemptive scheduling algorithms
- As multilevel feedback queue is a variation of priority scheduling $\Rightarrow$ it is well-suited for real time systems
- If Jobs are CPU-bound and independent the speedup on SMP systems is maximal

True or False

- When using round-robin starvation can never happen
- Shortest job first provides the minimal turnaround time among all non-preemptive scheduling algorithms
- As multilevel feedback queue is a variation of priority scheduling $\Rightarrow$ it is well-suited for real time systems
- If Jobs are CPU-bound and independent the speedup on SMP systems is maximal

True or False

- When using round-robin starvation can never happen
- Shortest job first provides the minimal turnaround time among all non-preemptive scheduling algorithms
- As multilevel feedback queue is a variation of priority scheduling $\Rightarrow$ it is well-suited for real time systems
- If Jobs are CPU-bound and independent the speedup on SMP systems is maximal

True or False

- When using round-robin starvation can never happen
- Shortest job first provides the minimal turnaround time among all non-preemptive scheduling algorithms
- As multilevel feedback queue is a variation of priority scheduling $\Rightarrow$ it is well-suited for real time systems
- If Jobs are CPU-bound and independent the speedup on SMP systems is maximal

True or False

- When using round-robin starvation can never happen
- Shortest job first provides the minimal turnaround time among all non-preemptive scheduling algorithms
- As multilevel feedback queue is a variation of priority scheduling $\Rightarrow$ it is well-suited for real time systems
- If Jobs are CPU-bound and independent the speedup on SMP systems is maximal

What considerations apply to SMP but not single-processor scheduling?

What considerations apply to SMP but not single-processor scheduling?

- Shared data between threads (reuse entries in per processor caches)
- Communication between threads (communicating threads should be run in parallel so that they don't have to wait until the partner gets the message)

$\Rightarrow$ need to adapt policy according to the behaviour of the concrete application

Central vs. per processor ready queue

Central vs. per processor ready queue

Central queue

- Access has to be synchronized (poor scalability)
- Changes to the ready queue cause caches of it to be invalidated on the other processors

Per processor queue

- Extra mechanisms for load balancing needed
- Better cache utilisation as processes are run on the same CPU (cache affinity)

How can concurrent activities interact?

How can concurrent activities interact?

- Shared memory
  - Implicit: Same address space
  - Explicit: Shared memory area
- OS communication facilities (messages, pipes, sockets)
- High-level abstractions (files, database entries)
- Devices (DMA, interrupts)
- Covert channels

Symetric vs. Asymmetric Communication

Unidirectional vs. Bidirectional Communication

Unidirectional vs. Bidirectional Communication

Unidirectional Communication

+ Sender may send out message and continue execution without having to wait for a reply
– Delivery of the message may fail without the sender noticing (even if the message system signals correct submission to the receivers queue)

Bidirectional Communication

+ Allows the receiver to pass back status information (message handled, message invalid, etc.)

Connection-Oriented vs. Connectionless

Direct vs. Indirect Addressing

Send by Copy vs. Send by Reference

Fixed-Sized vs. Variable-Sized Messages

Why does it make sense to define a timeout for IPC operations?

Where could the message buffer for asynchronous messages be located?

Why provide an atomic `sendReceive` syscall?

How to perform asynchronous IPC if the OS only provides synchronous IPC?

How to provide synchronous IPC if the OS only offers asynchronous IPC?

Is it possible to use the fast path on SMP systems?

L4 IPC does a thread switch bypassing the scheduler

Could policies be violated and might this be justified?

Fast path has following code in the `send` routine

Are there race conditions on SMP systems and if so how to avoid them?

```
if (TCB(receiver)->state == WAITING) {

    TCB(receiver)->state = RUNNING;
    SWITCH_TO(receiver);

} else {
    slowpath();
}
```

Fast path has following code in the `send` routine

Are there race conditions on SMP systems and if so how to avoid them?

```
if (TCB(receiver)->state == WAITING &&
        TCB(receiver)->cpu == current->cpu){

    TCB(receiver)->state = RUNNING;
    SWITCH_TO(receiver);

} else {
    slowpath();
}
```

Questions & Comments

Any questions or comments?

It's not a bug it's a feature