Process Scheduling in Linux

- Scheduling Mechanism: how to switch.
- Scheduling Policy: when to switch and what process to choose. Some scheduling objectives:
  - fast process response time
  - avoidance of process starvation 放置进程别饿死（要使用的资源被占用）
  - good throughput for background jobs 更好的吞吐量
  - support for soft real time processes
- Linux uses dynamically assigned process priorities for non real-time processes. Processes running for a long time have their priorities decreased while processes that are waiting have their priorities increased dynamically.

Classification of Processes

计算密集型　　IO密集型

- Compute-bound versus I/O bound. Linux implicitly favors I/O bound processes over compute bound processes (why?).
- Another classification:
  互动式进程：shell
  - Interactive processes. Examples: shells, text editors, GUI applications.
  - Batch processes. Examples: compilers, database search engine, web server, number-crunching. 分批式进程
  - Real-time processes. audio/video applications, data-collection from physical sensors, robot controllers.

实时进程

Process States in Linux

There are five process states defined in `/usr/include/linux/sched.h`.

```
#define TASK_RUNNING        0
#define TASK_INTERRUPTIBLE  1
#define TASK_UNINTERRUPTIBLE    2
#define TASK_ZOMBIE     4
#define TASK_STOPPED        8
```

Scheduling Parameters

- Processes are preemptible in user mode but not in kernel mode.
- How long is a quantum? Examine `INIT_TASK` macro in include/linux/sched.h header file. All processes inherit the default quantum value via fork from the `init` task.

```
#define DEF_COUNTER (10*HZ/100) /* 100 ms time slice */
#define MAX_COUNTER (20*HZ/100)
#define DEF_NICE    (0)
```

- Processes have two types of priorities:
  - Static priority. Between 1 and 99. Used by real-time processes.
  - Dynamic priority. For non real-time processes. Sum of the base time quantum and of the number of ticks of CPU time left to the process before its quantum expiers in the current epoch.

Data Structures Used by Scheduler

From the task_struct in include/linux/sched.h.

```
struct task_struct {
    ...
    volatile long need_resched;
    ...
    long counter;
    long nice; // replaces the priority variable in earlier kernel
    unsigned long policy; //SCHED_OTHER, SCHED_FIFO, SCHED_RR
    ...
    int has_cpu, processor;
    unsigned long cpus_allowed;
    ...
    unsigned long rt_priority;
    ...
}
```

Passing quantums after fork()

From the `do_fork(...)` function in `kernel/fork.c`.

```
// from do_fork in kernel/fork.c (different from the textbook)
    p->counter = (current->counter + 1) >> 1;
    current->counter >>= 1;
    if (!current->counter)
        current->need_resched = 1;
```

再调度

- The number of ticks left to the parent is split in two halves, one for the parent, one for the child. This prevents processes from getting an unlimited amount of CPU time.

Invoking the scheduler

The scheduler proper is the function `schedule()` in `kernel/sched.c`.

- Direct Invocation. Examples: A process must be blocked because a resource is not available, a device driver can invoke `schedule()` directly if it will be executing a long iterative task.
- Lazy Invocation. By setting the `need_resched` flag field of `current` to 1. Examples:
  - When `current` has use up its quantum of CPU time (done by `update_process_times` function in `kernel/timer.c`)
  - When a process is woken up and its priority is higher than that of the `current` process.
  - When a `setscheduler()` or `sched_yield()` system call is issued.

Goodness of a Process

This is the function that decides how desirable a process is.. You can weigh different processes against each other depending on what CPU they've run on lately etc to try to handle cache and TLB miss penalties.

Return values:

- -1000: never select this
- 0: out of time, recalculate counters (but it might still be selected)
- +ve: "goodness" value (the larger, the better)
- +1000: realtime process, select this.

Actions performed by schedule()

- treat current process
- select process
- switch process

1.处理当前进程
2.选择下面将被调度的进程
3.交换进程

Linux SMP scheduler

- Each cpu runs the schdule function on its own and communicate via shared data structures.
每个CPU运行自己的调度函数，并通过共享数据类型进行交互

```
/*
 * We align per-CPU scheduling data on cacheline boundaries,
 * to prevent cacheline ping-pong.
 */
static union {
    struct schedule_data {
        struct task_struct * curr;
        cycles_t last_schedule;
    } schedule_data;
    char __pad [SMP_CACHE_BYTES];
} aligned_data [NR_CPUS] __cacheline_aligned = { {{&init_task,0}}};
```
多处理器

- Processor affinity is implemented through extra goodness for staying on the same cpu.

Performance of Linux Scheduler

- *The scheduling algorithm does not scale well.*
- *The predefined quantum is too large for high system loads.*
- *I/O bound process priority boosting is not optimal.*
- *Support for real-time applications is weak.* 让我想起了物理层

Several attempts have been made to provide alternate schedulers. An interesting tool called Lockmeter is available to study performance of SMP scheduling.

System Calls Related to Scheduling

Name
`nice()`
`getpriority()`
`setpriority()`
`sched_getscheduler()`
`sched_setscheduler()`
`sched_getparam()`
`sched_setparam()`
`sched_yield()`
`sched_get_priority_min()`
`sched_get_priority_max()`
`sched_rr_get_interval()`