Operating Systems
Project #1

2002/11/20

Outline

- GLOBAL
- schedule( )
  - Invocation
  - Process Selection
  - goodness( )
- Project #1 Requirements
GLOBAL

- What’s *Global*?
  - *Global* can find the locations of a specified object quickly.
  - *Global* can locate not only object definitions but also object references.

GLOBAL

- How To Start?
  - Installation
    - `% ./configure`
    - `% make`
    - `# make install`
  - gtags, htags
**gtags**

- Create Tag Databases
  - GTAGS – the database of function definitions
  - GRTAGS – the database of function references
  - GSYMS – the database of other symbols
  - GPATH – the database of path names

% cd /usr/src/linux-2.4.19
% gtags

---

**htags**

- Create Hypertext Files (under HTML/) for a Web-Based Interface for Global
- Create Tag Databases and Hypertext Files in a Compact Format

% gtags -c
% htags -c
Invocation of schedule( )

- Direct Invocation
  - Invoked when any resources that the current process needs is not available.

- Lazy Invocation
  - Set the `need_resched` field of the `struct current` to 1.
  - `need_resched`: A flag checked by `ret_from_intr()` to decide whether to invoke function `schedule()`.
Lazy Invocation – Case1 (1/2)

- When struct current has used up its quantum of CPU time.
  - done by function update_process_times( )

- The occurrence of a timer interrupt
  - Trigger timer_bh( )
  - Invoke update_times( )
  - Invoke update_process_times( )

Lazy Invocation – Case1 (2/2)

- `update_process_times(user_ticks)
  struct task_struct *p = current;
  ...
  if (--p->counter <= 0) {
    p->counter = 0;
    p->need_resched = 1;
  }
  ...

*counter: The number of ticks of CPU time left to the process before its quantum expires.

**On multiprocessor systems, it was necessary to define current to an array entry – one element for each available CPU.
Lazy Invocation – Case2 (1/3)

- When a process is woken up and its priority is higher than that of the current process, the following action is taken
  - Performed by function \texttt{reschedule\_idle( )}
  - Make a process runnable
    - Invoke \texttt{wake\_up\_process( )}
    - Invoke \texttt{reschedule\_idle( )}

Lazy Invocation – Case2 (2/3)

- \texttt{reschedule\_idle(* p)}
  
  struct task\_struct *tsk;

  tsk = cpu\_curr(this\_cpu);
  if (preemption\_goodness(tsk, p, this\_cpu) > 0)
    tsk-\rightarrow\text{need\_resched} = 1;

- \texttt{preemption\_goodness(* prev, * p, cpu)}

  return goodness(p, cpu, prev-\rightarrow\text{active\_mm}) –
  goodness(prev, cpu, prev-\rightarrow\text{active\_mm});
Lazy Invocation – Case2 (3/3)

- A small bonus is given to $p$ if it shares the address space with $\text{prev}$ (i.e., if their process descriptors’ $\text{active_mm}$ fields point to the same memory descriptor). The rationale for this bonus is that if $p$ runs right after $\text{prev}$, it will use the same page tables, hence the same memory; some of the valuable data may still be in the hardware cache.

Lazy Invocation – Case3

- It happens when a process invokes system call $\text{sched_setscheduler( )}$ or $\text{sched_yield( )}$.
  - $\text{sched_setscheduler( )}$ sets the scheduling policy and the priority of a process.
  - $\text{sched_yield( )}$ relinquishes the processor voluntarily without blocking.
Which Process to Select (1/3)

- The objective is to select the most valuable process to run!
  - Save the process descriptor pointer of the highest priority process in `next`.

- `schedule()` repeatedly invokes function `goodness()` on all the runnable processes iteratively to determine the best candidate.

```
Which Process to Select (2/3)

- `schedule()`:
  ```c
  struct list_head *tmp;
  ...
  repeat_schedule:
      /* Default process to select.. */
      next = idle_task(this_cpu);
      c = -1000; // initial value
      list_for_each(tmp, &runqueue_head) {
          p = list_entry(tmp, struct task_struct, run_list);
          if (can_schedule(p, this_cpu)) {
              int weight = goodness(p, this_cpu, prev->active_mm);
              if (weight > c)
                  c = weight, next = p;
          }
      }
  ...
  ```
```
Which Process to Select (3/3)

```c
#ifdef CONFIG_SMP
#define idle_task(cpu) (init_tasks[cpu_number_map(cpu)])
#define can_schedule(p,cpu) \
   (((p)->cpus_runnable & (p)->cpus_allowed & (1 << cpu))
#else
#define idle_task(cpu) (&init_task)
#define can_schedule(p,cpu) (1)
#endif

*cpus_runnable is ~0 if the process is not running on any CPU.
*It is (1 << cpu) if it's running on a CPU. This mask is updated
under the runqueue lock. To determine whether a process
might run on a CPU, this mask is AND-ed with cpus_allowed.
```

How Good Is a Runnable Process?

- `goodness( )`
- Return Values:
  - -1000: never select this process.
  - 0: out of time quantum ➔ recalculate counters.
  - +ve: “goodness” value (the larger, the better).
  - +1000: real time process ➔ Select this process.
goodness(* p, this_cpu, * this_mm)

/* Non-RT process - normal case first. */
if (p->policy == SCHED_OTHER) {
    weight = p->counter;
    if (!weight)
        goto out;
#endif
if (p->processor == this_cpu)
    weight += PROC_CHANGE_PENALTY;
#endif
if (p->mm == this_mm || !p->mm)
    weight += 1;
weight += 20 - p->nice;
    goto out;
}
weight = 1000 + p->rt_priority;
out:
return weight;

Remark:
- The *nice* value ranges from –20 to +19, where a –20 is a “high-priority” task, and a +10 is a “low-priority” task.
Revised Multilevel Feedback Queue Scheduling

- `per_cpu_utime`: stores the number of ticks during which the process has been running in the user mode.
- `per_cpu_stime`: stores the number of ticks during which the process has been running in kernel mode.

Platform/Environment:
- RedHat 7.3 CLE v1.1 on VMware 3.2.0
- Linux-2.4.19
- Two Persons/Team
- Deadline: 3 weeks later (12 / 11 24:00)
- Submit files to TA:
  - d90002@csie.ntu.edu.tw
Operating Systems Project #1 (3/3)

- Your files SHOULD include:
  - Modified files
  - 5-Page Report (as detailed as possible!!)
    - You should explain:
      - the codes you modified
      - the thresholds you choose
  - Problems Encountered and Your Solutions
  - Observations or Anything Else
- Bonus (20%): How to verify your WORKS!!