Implementation of the Least Slack Time scheduling algorithm for Linux

Panagiotis Papadopoulos
panpap@csd.uoc.gr

Process Scheduling

- Switching from one process to another in a very short time frame

- Scheduler:
  - When to switch processes
  - Which process to choose next
  - Major part of the operating system kernel

Linux Scheduler

- The scheduler makes it possible to execute multiple programs at the same time, thus sharing the CPU with users of varying needs.
  - minimizing response time
  - maximizing overall CPU utilization

- Preemptive:
  - Higher priority processes evict lower-priority running processes

- Quantum duration
  - Variable
  - Keep it as long as possible, while keeping good response time

Linux Scheduling Algorithm

- Dividing CPU time into epochs
  - In each epoch, every process has a specified quantum
    a. Varies per process
    b. Its duration is computed when the epoch begins

  - Quantum value is the maximum CPU time portion for this process in one epoch
    - When this quantum passes, the process is replaced

- Process priorities
  - Defines process's quantum

How it works

- At the beginning of each epoch:
  - Each process is assigned a quantum (Based on its priority, previous epoch, etc)

- During each epoch:
  - Each epoch runs until its quantum ends, then replaced
    - If a process blocks (e.g., for I/O) before the end of its quantum, it can be scheduled for execution again in the same epoch

Linux Scheduler (in practice)

- Implemented in linux-source-2.6.38.1/kernel/sched.c
- Main scheduler’s routine is schedule()

- Data structures:
  - policy (SCHED_FIFO, SCHED_RR, SCHED_RR)
  - priority (base time quantum of the process)
  - counter (number of CPU ticks left)

Runqeueue list

- A list with all runnable processes
- Processes that are not blocked for I/O
- Candidates to be selected by schedule() for execution

- struct rq: (Defined in sched.h)

The schedule() function

- Implements the Linux scheduler
- Finds a process in the runqueue list for execution
- Invoked when a process is blocked
- Invoked when a process quantum ends
  - Done by update_process_times()
- Invoked when a process with higher priority than the current process wakes up
- Invoked when sched_yield() is called

Actions performed by schedule()

- First it runs kernel functions that have been queued (drivers, etc)
  - run_task_queue(&tq_scheduler);
- Current process becomes prev
  - prev=current
- Next will point to the process that will be executed when schedule() returns

Round-robin policy

If prev has exhausted its quantum, it is assigned a new quantum and moved to the bottom of the runqueue list

```
if (!prev->counter && prev->policy == SCHED_RR)
{
prev->counter = prev->priority; move_last_runqueue(prev);
}
```

State of prev

- Wake up a process:

```
if (prev->state == TASK_INTERRUPTIBLE && signal_pending(prev))
        prev->state = TASK_RUNNING;
```

- Remove from runqueue if not TASK_RUNNING:

```
if (prev->state != TASK_RUNNING)
        del_from_runqueue(prev);
```

Select next process for execution

Scan the runqueue list starting from init_task.next_run and select as next the process with higher priority:

```
p = init_task.next_run;
while (p != &init_task) {
	weight = goodness(prev, p);
	if (weight > c) {
		c = weight;
		next = p;
	}
	p = p->next_run;
}
```

Goodness function

- Find the best candidate process
  - c=-1000 must never be selected
  - c=0 exhausted quantum
  - 0<c<1000 not exhausted quantum
  - c>=1000 real time process

```
if (p -> policy != SCHED_OTHER)
        return 1000 + p -> rt_priority;
if (p -> counter == 0)
        return 0;
if (p -> mm == prev -> mm)
        return p -> counter + p -> priority + 1;
return p -> counter + p -> priority;
```

Empty runqueue or no context switch

- If the runqeue list is empty
  - No runnable process exists
  - Next points to the init_task

- If all processes in the runqueue list has lower priority than the current process prev
  - No context switch
  - prev will continue its execution

New epoch

When all processes in the runqueue list have exhausted their quantum

- All of them have zero counter field
- Then a new epoch begins

```
if (!c) {
    for_each_task(p)
    P -> counter = (p -> counter >> 1) + p -> priority;
}
```

Context Switch

```
if (prev != next) {
	kstat.context_swtch++;
	switch_to(prev,next);
}
return;
```

schedule() in 2.6.38.1 (1/2)

```
asmlinkage void __sched schedule(void)
{
        struct task_struct *prev, *next;
        unsigned long *switch_count;
        struct rq *rq;
        int cpu;
        ........
```

schedule() in 2.6.38.1 (2/2)

```
raw_spin_lock_irq(&rq->lock);
pre_schedule(rq, prev);
if (unlikely(!rq->nr_running))
        idle_balance(cpu, rq);
put_prev_task(rq, prev);
next = pick_next_task(rq);
clear_tsk_need_resched(prev);
rq->skip_clock_update = 0;
if (likely(prev != next)) {
        sched_info_switch(prev, next);
        rq->nr_switches++;
        rq->curr = next;
        ++*switch_count;
        context_switch(rq, prev, next);
}
raw_spin_unlock_irq(&rq->lock);
post_schedule(rq);
```

Pick up the next task

```
static inline struct task_struct * pick_next_task(struct rq *rq) {
	const struct sched_class *class;
	struct task_struct *p;
	if (likely(rq->nr_running == rq->cfs.nr_running)) {
		p = fair_sched_class.pick_next_task(rq);
		if (likely(p)) return p;
	}
	for_each_class(class) {
		p = class->pick_next_task(rq);
		if (p) return p;
	}
}
```

Assignment 4

Continue from assignment 3

Copy your code from assignment 3 and start with the new fields in task struct and the two new system calls

- You will use the *set_lst_parameters()* for your tests

Use qemu and same process to compile Linux kernel and boot with the new kernel image

Real-Time Systems

Definition:

– Systems whose correctness depends on their temporal aspects as well as their functional aspects

Performance measure:

– Timeliness on timing constraints (deadlines)

– Speed/average case performance are less significant.

Key property:

– Predictability on timing constraints

Real-time systems (examples)

- Real-time monitoring systems
- Signal processing systems (e.g., radar)
- On-line transaction systems
- Multimedia (e.g., live video multicasting)
- Embedded control systems:
  - Automotives
  - Robots
  - Aircrafts
  - Medical devices ...

In this assignement

Implementation of a Real-time scheduling algorithm named Least Slack Time algorithm which assigns priority based on the slack time of a process. Slack time is the amount of time left after a job if the job was started now.

So:

- Filter out processes that have exceeded their deadlines

From the rest, execute the process with the least slack time Slack time:

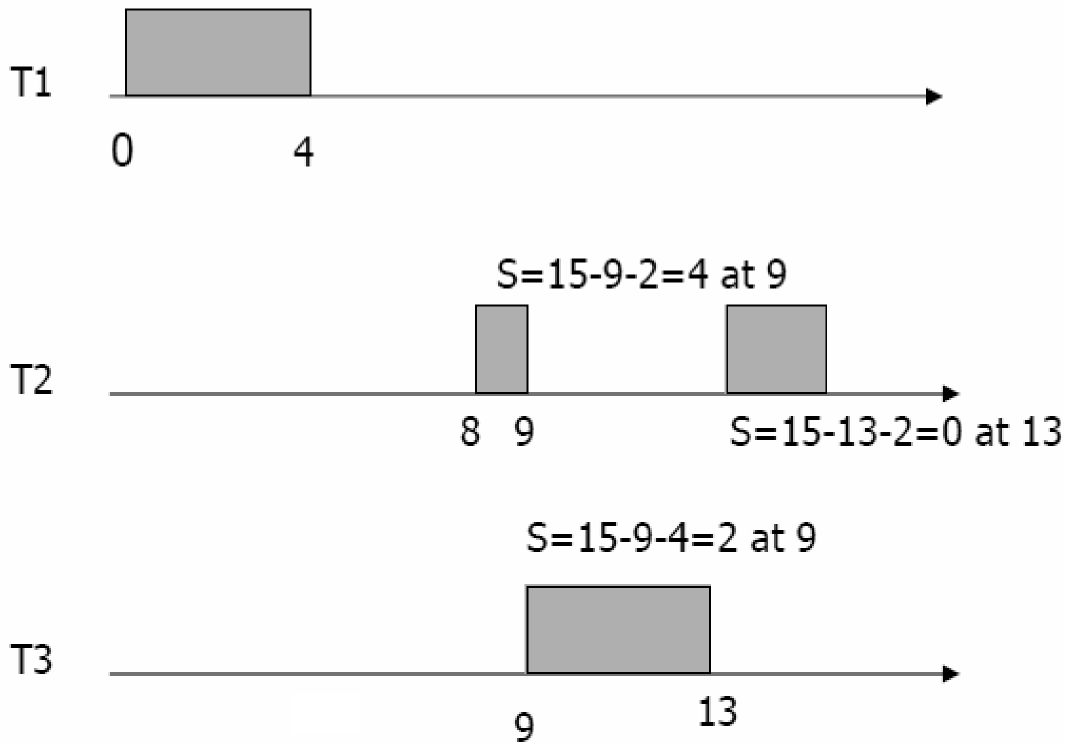
slack = deadline – [remaining_computation_time – (utime+stime)]

warning! Deadline's type is time_t when remaining_computation_time's is int...

LST Example

	T1	T2	T3
A	0	8	9
C	4	2	4
D	20	15	15

(D=absolute deadline)



A: Arrival time, C: Estimated Calculation Time, D: Deadline

Pre-process and filtering in runqeue list

- Before schedule() selects the next process
- (You may clone the runqueue list rq localy for convenience to rq’)
- Scan all processes in the runqueue list and find if there is any process that has a deadline (deadline!=-1).
  - If so, calculate its slack time. If this process has exceeded the given deadline.
    - If so, remove this process from the runqueue list so it'll never be executed
    - If not, iterate the runqueue list rq. For each process p, check if p has less slack value.
      - If so execute process p first.

Demonstrating the modified scheduler

A demo program that:

1. will create 10 child processes.
2. for each child process "i" the parent process will set its remaining computation time to "i" and its deadline to "100".
3. each daughter process will sleep for i seconds and then it will print "i".

What to submit

1. bzImage
2. Only modified or created by you source files of the Linux kernel 2.6.38.1 (both C and header files)
3. Demo program and header files used in the guest OS for testing the modified scheduler
4. README with implementation details

Gather these files in a single directory and send them using the submit program as usual (by using turnin tool)