Operating Systems Lab

Part 1: Threads

Youjip Won

4.4BSD like scheduler

Outline of Advanced Scheduler

- Main Goal
  - Implement 4.4 BSD scheduler MLFQ like scheduler without the queues.
    - Give priority to the processes with interactive nature.
    - Priority based scheduler
  - Use “equation”.
- Files to modify
  - threads/thread.*
  - devices/timer.c

nice

- Nice value
  - Represents the ‘niceness’ of a thread.
  - If a thread is nicer, it is willing to give up some of its CPU time.
- Value (from -20 to 20)
  - Nice (0) : not influence on priority. (initial value)
  - Nice (positive) : decrease priority.
  - Nice (negative) : increase priority.
- Function
  - `thread_get_nice()`
  - `thread_set_nice(int new_nice)`

Priority

- Priority
  - From 0 (PRI_MIN) to 63 (PRI_MAX)
  - The larger the number, the higher the priority.
  - It initialized when thread is created (default: 31)

Priority

```
priority = PRI_MAX - (recent_cpu / 4) - (nice * 2)
```

- Philosophy
  - If the thread is nicer, lower the priority.
  - If the thread have been using lots of CPU recently, lower the priority.
  - For all threads, priority is recalculated once in every fourth clock tick.
  - The result is truncated to its nearest integer.

CPU usage

- Update the cpu usage

`recent_cpu`

  - Increase the `recent_cpu` of the currently running process by 1 in every tim er interrupt.
  - Decay `recent_cpu` by `decay` factor in every second.

```
recent_cpu = decay * recent_cpu
```

  - Adjust `recent_cpu` by `nice` in every second.

```
recent_cpu = recent_cpu + nice
```

  - Putting them together,

```
recent_cpu = decay * recent_cpu + nice
```

Decay factor

- In SVR3

$$decay = \frac{1}{2}$$

- In BSD4.4
  - In heavy load, decay is nearly 1.
  - In light load, decay is 0.

```
decay = (2*load_average)/ (2*load_average + 1)
```

load _average

- load_average

```
load_avg = (59/60)*load_avg + (1/60)*ready_threads
```

  - At booting, load_avg is initially set to 0.
  - ready_threads: the number of threads in ready_list and threads in executin g at the time of update. (except idle thread)

In summary,

In every fourth tick, recompute the priority of all threads

```
priority = PRI_MAX - (recent_cpu / 4) - (nice * 2)
```

In every clock tick, increase the running thread's `recent_cpu` by one.

In every second, update every thread's `recent_cpu`

```
recent_cpu = decay * recent_cpu + nice,
```

where

```
decay = (2*load_average)/ (2*load_average + 1)
```

and

```
load_avg = (59/60)*load_avg + (1/60)*ready_threads
```

```
Priority = PRI_MAX - (recent_cpu/4) - (nice*2)
recent_cpu = (2*load_avg)/(2*load_avg+1)*recent_cpu + nice
load_avg = (59/60)*load_avg + (1/60)*ready_threads
```

nice = 0,
load_avg = 0

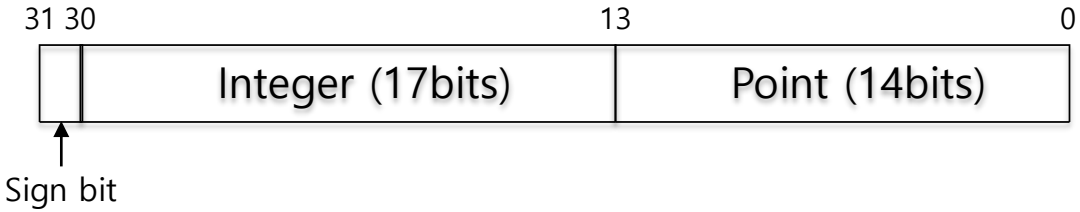
Sec	Tick	P1		P2		P3	
		Priority	recent_cpu	Priority	recent_cpu	Priority	recent_cpu
0	0	63	0	63	0	63	0
	1	63	1	63	0	63	0
	2	63	2	63	0	63	0
	3	63	3	63	0	63	0
	4	62	4	63	0	63	0
	5	62	4	63	1	63	0
	6	62	4	63	2	63	0
	7	62	4	63	3	63	0
1	8	63	0	62	4	63	0
	9	63	0	62	4	63	1
	10	63	0	62	4	63	2
	11	63	0	62	4	63	3
	12	63	0	63	0	62	4
	13	63	1	63	0	62	4
	14	63	2	63	0	62	4
	15	63	3	63	0	62	4

Priority
=PRI_MAX(63)–(4/4)–(0*2)
=62

Running Priority recalculate Recent CPU recalculate

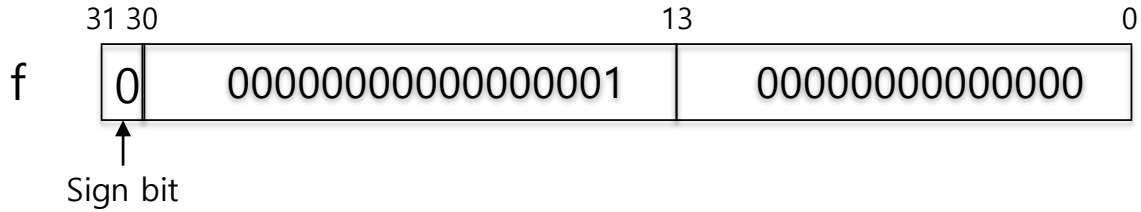
Implement fixed point arithmetic

- Inside kernel, you can do only integer arithmetic.
  - Kernel does not save floating point register when switching the context.
- We need to implement fixed point arithmetic using integer arithmetic.
  - `priority`, `nice`, `ready_threads` value is integer, and `recent_cpu`, `load_avg` value is real number.
  - Implement the fixed-pint arithmetic using 17.14 fixed-point number representation.
    - Decimal point is 14 right-most bits.
    - Integer is 17 next bits to the left.
    - Last of left 1bit is sign bit.



Operations to implement

- n : integer x, y : fixed-point numbers f : 1 in 17.14 format



Convert n to fixed point:	n * f
Convert x to integer (rounding toward zero):	x / f
Convert x to integer (rounding to nearest):	(x + f / 2) / f if x >= 0, (x - f / 2) / f if x <= 0.
Add x and y:	x + y
Subtract y from x:	x - y
Add x and n:	x + n * f
Subtract n from x:	x - n * f
Multiply x by y:	((int64_t) x) * y / f
Multiply x by n:	x * n
Divide x by y:	((int64_t) x) * f / y
Divide x by n:	x / n

Examples

- Convert n to fixed point: n * f: shift n by 14 bits to the left.
- Convert x to integer: shift x by 14 to the right.

Implementation

- Each thread maintains
  - `nice` and `recent_cpu`
  - Add `nice, recent_cpu` to `thread` structure.
- Functions to be added
  - The function that calculate priority using `recent_cpu` and `nice`.
  - The function that calculate `recent_cpu`.
  - The function that calculate `load_avg`.
  - The function that increase `recent_cpu` by 1.
  - The function that recalculate priority and `recent_cpu` of all threads.
- Multiple ready queues vs. single ready queue
  - You can use single in implementing BSD scheduler.
  - **If you use multiple ready queues, you will get 10 extra mark for 100 mark.**

Functions to modify

- `static void init_thread(struct thread *t, const char *name, int priority)`
  - Initialize `nice, recent_cpu`
- `void thread_set_priority(int new_priority)`
  - Disable the priority setting when using the advanced scheduler.
- `static void timer_interrupt(struct intr frame *args UNUSED )`
  - Recalculate `load_avg, recent_cpu` of all threads, priority every 1 sec.
  - Recalculate priority of all threads every 4th tick.

Functions to modify

- `void lock_acquire(struct lock *lock)`
  - Forbid the `priority donation` when using the advanced scheduler.

- `void lock_release(struct lock *lock)`
  - Forbid the `priority donation` when using the advanced scheduler.

Functions to modify

- `void thread_set_nice(int nice UNUSED)`
  - Set `nice` value of the current `thread`.
- `int thread_get_nice(void)`
  - Return `nice` value of the current `thread`.
- `int thread_get_load_avg(void)`
  - Return `load_avg` multiplied by 100
  - `timer_ticks() % TIMER_FREQ == 0`
- `int thread_get_recent_cpu(void)`
  - Rerturn `recent_cpu` multiplied by 100

Result

```
$ make check
```

```
pass tests/threads/mlfqs-block
pass tests/threads/alarm-single
pass tests/threads/alarm-multiple
pass tests/threads/alarm-simultaneous
pass tests/threads/alarm-priority
pass tests/threads/alarm-zero
pass tests/threads/alarm-negative
pass tests/threads/priority-change
pass tests/threads/priority-donate-one
pass tests/threads/priority-donate-multiple
pass tests/threads/priority-donate-multiple2
pass tests/threads/priority-donate-nest
pass tests/threads/priority-donate-sema
pass tests/threads/priority-donate-lower
pass tests/threads/priority-fifo
pass tests/threads/priority-preempt
pass tests/threads/priority-sema
pass tests/threads/priority-condvar
pass tests/threads/priority-donate-chain
pass tests/threads/mlfqs-load-1
pass tests/threads/mlfqs-load-60
pass tests/threads/mlfqs-load-avg
pass tests/threads/mlfqs-recent-1
pass tests/threads/mlfqs-fair-2
pass tests/threads/mlfqs-fair-20
pass tests/threads/mlfqs-nice-2
pass tests/threads/mlfqs-nice-10
pass tests/threads/mlfqs-block
All 27 tests passed.
```