ISSN:0975-9646

G.Keerthi et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 6 (5) , 2015, 4660-4662

New Approach of Scheduling Algorithms in Linux Operating System with Goodness Function

G.Keerthi
Assistant Professor
Dept of IT
St. Martin's Engineering College
Hyderabad.

Dr.R.China Appala Naidu
Professor
Dept of CSE
St. Martin's Engineering College
Hyderabad.

Abstract: Now a day's most of the people are using scheduling algorithms in most of the resources where in this paper we are highlighting the concept of process scheduling used in Linux operating system , with the use of goodness function . This function is used to reduce the CPU time scheduling and effects and also find the task with relative desirability. Comparative to the other scheduling algorithms goodness function works well in all circumstances like giving IO-bound processes with a good response and efficiently. It took less time with good response, the same we explained and shown in this paper.

Key terms: Scheduling, Process scheduling algorithms, priorities, goodness

INTRODUCTION:

The first ever Linux process scheduler that came with kernel version 0.01 in 1991 used minimal design. Scheduling is the process in which the assigned work has to be completed with in a specified time with the given resources. Resources can be Hardware, Software and Networking such as processors, network links or expansion cards Scheduling is nothing but it carries out the scheduling activities.

The main goal of scheduling is to reach the target with the quality of services. It also implements to keep all computer resources busy which allow multiple users to share system resources effectively. Scheduling is the fundamental concept of computation and it is intrinsic part of execution model of computer system. The final execution model should be capable of performing multi tasking by using single central processing unit(CPU).

The main advantage of scheduling algorithm is to reduce starvation and to help in ensuring the fairness among the utilizing resources.

PROCESS SCHEDULING :

The main definition of process scheduling is nothing but it is which handles the removal of running process from the CPU and the selection is based on particular strategy of the process.

Process scheduling is an important part of a Multiprogramming operating system where more than one process is loaded into the memory at a time and the loaded process shares the CPU by using time multiplexing.

Process scheduling algorithms are basically two in linux:

1.A time-sharing algorithm – It makes uses of scheduling between multiple processes

2.A real-time algorithm – Finishing the task by using priorities with absolute fairness

- For **time-sharing processes**, Linux uses a credit based algorithm. In this credit based rule it has following
  Credits= credits/2 + priority
  Time sharing process basically takes two factors into consideration i.e one processes and another priority.
- Linux implements process scheduling like **FIFO** and **round-robin real-time scheduling** that make use of priority and scheduling class.
  FIFO: Priorities are basically classified as highest, smallest and equal priority. Highest priority are the one which take additional time then the smallest and equal priority will be completed in estimated time without considering which type of priority it is. FIFO will continue the process until they either exit or block the process.
  Priorities:
  1. Static priority
     It allow maximum size of the time slice for a process before it has being forced by other processes to compete for the CPU.
  2. Dynamic priority
     As long as the process has the CPU, the amount of time remaining in this time slice declines with time of process. The process is marked for rescheduling when its dynamic priority falls to zero.
  3. Real-time priority
     Priorities with real time values will be executed. Higher real-time values always beat lower values

Scheduling :

- Need_resched field of prev is set to zero.
- Schedule( ) assign a new quantum to prev and places it at the bottom of run queue list.
- If the process state is TASK_INTERUPTIBLE then the function wakes up the process.
- Schedule( ) invokes goodness() function so as to identify best candidate that is runnable process.

The Linux scheduler is a priority based scheduler where scheduling task is based upon the static and dynamic priorities. When all these priorities are combined together they form a task's *goodness() function* . Every time where the Linux scheduler runs their each task on the run queue is examined and their *goodness* value is computed. The task which is having the highest *goodness*() function is chosen to run next.

goodness( ) function

- It identify the best process among all other processes in the run queue list.
- Goodness( ) function receives as input parameters ***prev*** and ***p*** , where prev means descriptor of previous running process, p means the descriptor pointer of the process which is evaluated.
- The integer value ***c*** returned by ***goodness( )function*** measures the "goodness" of ***p*** which has the following meanings:
  - ***c = -1000***,where ***p*** must never be selected; this value is returns the run queue list if it contains only ***init_task***
  - ***c =0***,then ***p*** has exhausted its quantum, if not ***p*** is the first process in the run queue list and all the runnable process have also exhausted their quantum, it will not be selected for execution.
  - ***0 < c < 1000***,then ***p*** is conventional process that has not exhausted its quantum which means the higher value of ***c*** is denoted to a higher level of goodness function.
  - **c >= 1000**,where ***p*** is a real-time process means it is a higher value of ***c*** denotes a higher level of goodness.

Linux scheduler behavior has two paths involved in it they are

1. schedule – schedule means running or current task is SCHED_OTHER task that expires the time slice.
2. reschedule_idle – reschedule idle means wakeuping the task with the best CPU by invoking a schedule on it.

Both paths shares the goodness ( ) function which consider the core of the SMP scheduler time. It calculates the goodness function on the following bases, they are

- the task which is currently running
- the task that wants to run
- e current CPU

```
rep_sch:
 next=idle_tsk( this_cpu);
c=-1000;
list_entry (tmp, &runqueue_head)
{
       p=list_entry(tmp, struck tsk_struct, run_list);
      If(can_sch(p,this_cpu))
    {
          int wt = goodness(p,this_cpu,prev->active_mm);
           if(wt>c)
          c=wt, next=p;
    }
}
```

Based on goodness function only plain scheduling works. It is SMP – aware. Goodness potential increments last CPU task and changes it to current CPU task.

The main objective of reschedule_idel is wakenup the previous task and to call the schedule on to the CPU.. We use goodness in reschedule_idle because of predicting the effect of the future schedule that will send to that CPU. By predicting the effect of the future schedule, we can select the best CPU to reschedule at wakeup time. This, of course, saves us the trouble of executing on a CPU. If the CPU to reschedule is not the current then reschedule event via inter-CPU message passing.

In Linux scheduler , goodness function is the core part and it is also SMP aware, while reschedule_idle is the core of the clever SMP heuristics.

Features of reschedule_idle:

1. It makes use of goodness function.
2. Goodness function decides which process is desirable.
3. It finds about interdependent processes.
4. It also finds about CPU time and TLB miss penalities.

```
static  int goodness(struct task_struct * p, int this_cpu,
struct mm_struct *this_mm)
{
  int wt;
  /*
   * Realtime process, select the first one on the
   * runqueue (taking priorities within processes
   * into account).
   */
  if (p->policy != SCHED_OTHER)
              {
                wt = 1000 + p->rt_priority;
    goto out;
  }
  /*
   * It gives the process a first-appropriate goodness value
  .
   *
   * It should not do any other calculations if the time slice
is
   * over..
   */
  wt = p->counter;
  if (!wt)
    goto out;

#ifdef __SMP__
  /*Same process will be having largish advantage...  */
  /* (this is equivalent to penalizing other processors) */
  if (p->processor == this_cpu)
    wt += PROC_CHANGE_PENALTY;
#endif
  if (p->mm == this_mm)
    wt += 1;
  wt += p->priority;
out:
  return wt;
}
```

Factors that makes goodness() function as important concept in linux are:

1. It is easy to find the task with related desirability.
2. Its calculation depends upon processor affinity. The significant advantage is given to the last processor which is running on the schedule because of existing possibility which have some memory lines in that processors cache.
3. Its calculation also depends upon its memory map address. If a task shares the same address space, then the task's goodness value is increased by one because of the reduced context switch fixed cost which is involved.
4. The effect of calculation is based on task counter value and its priority.
5. The task priority plays a major role in it.

CONCLUSION:

Process scheduling has always been the corner stone in operating systems' development. So the current Linux scheduling has grown large with all these Scheduling algorithms which we have been tried to explore in this paper with the effectiveness of goodness function, with this goodness function we proved it is better than the previous scheduling functions. As this function took less time to calculate with accurate result.

REFERENCES:

1. Ankita Garg, "Real-Time Linux Kernel Scheduler", Linux Journal, PP 0-4, Aug 2009.
2. https://en.wikipedia.org/wiki/Scheduling
3. http://www.linuxjournal.com/article/3910?page=0,0
4. Sivarama P. Dandamudi and Samir Ayachi. "Performance of hierarchical processor scheduling in shared-memory multiprocessor systems". IEEE Transactions on Computers, Vol 48 issue 11 PP1202–1213, 1999.
5. S. Haldar and D. K. Subramanian. "Fairness in processor scheduling in time sharing systems". Operating Systems Review, Vol 25. Issue 1 PP 4–18, 1991.
6. Daniel P. Bovet & Marco Cesati, "Chapter 10, Processing Scheduling, Understanding the Linux Kernel," 2000.
7. Daniel P. Bovet and Marco Cesati, Understanding the Linux Kernel, Third Edition. O'Reilly Media, 2005.
8. Thang Minh Le , A STUDY ON LINUX KERNEL SCHEDULER Version 2.6.32
9. Avinesh Kumar Multiprocessing with the Completely Fair Scheduler . Introducing the CFS for Linux.
10. Silberschatz, A., P.B. Galvin, and G. Gagne,"CPU Scheduling, Operating System Concepts, Sixth Ed.," John Wiley & Son, 2003.