IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 3 Issue 7, July 2016
ISSN (Online) 2348 – 7968 | Impact Factor (2015) - 4.332
www.ijiset.com

Scheduling Feedback Algorithms for Real-Time System

Amit Thakur and R.K. Dubey

Department of Electronics and communication
Swami Vivekanand University, Sagar (M.P.) India -470228

ABSTRACT

Actually scheduling is a sequence or series of jobs released over time when ever the processing time period of a job is only known at its completion is a classical problem in CPU scheduling in time sharing and real time operating systems [1]. This paper introduces a new scheduling concept i.e. Feedback Queue Algorithm (FQA). The goals for real time scheduling are completing tasks within specific time constraints and preventing from simultaneous access to shared resources and devices.

Keywords: FQA, scheduling, CPU, processing, real-time, access.

Introduction

A real time system is defined as: any system in which the time at which produced output is significant. This is usually because the input corresponds to some movement is the physical world and the output has to relate to that same movement. The lag from input time to output time must be sufficiently small for acceptable time. The correct behavior of a real system depends as much on the timing of computations as it does on the result produced by those computation, and calculation. Result delivered too late may be useless or even harmful [1,2].

The main objective of this research paper is to study the available task schedulers in practice. After comparing some scheduling policies on idea of new feedback queue scheduler is proposed.

Features of real time systems

RTS is a multi tasking and ability to quickly respond to external interrupts if any. It has small kernel and fast context switching and basic mechanism for process communication and synchronization. It has priority based scheduling but these are limited number of priority levels. RTS support real time clock as internal time interface.

Important characteristics of RTOS's

A real time operating system is an operating system intended to serve real time application process data as it comes in, typically without buffering delays. Processing time requirements including any as delay are measured in tenth of second or shorter.

A key characteristic of a RTOS is the level of its consistency concerning the amount of time it takes to accept and complete an application's

task. These are two distinct types of systems in this field: a hard real time systems and soft real time systems. A hard real time operating system has less jitter than a soft real time operating system.

The chief design goal is not high throughput but rather a guarantee of a soft or hard performance category. A RTOS has an advanced algorithm for scheduling. Scheduler flexibility enables a wider, computer system orchestration of process priorities but a real time operating system is more frequently dedicated to a narrow set of applications key factories in a real time 05 are minimal interrupt latency and minimal threads switching latency.

Design Philosophies

The most common designers are Event driven designs, in which switches tasks only when an event of higher priority needs servicing, called preemptive priority or priority scheduling. Time sharing designs, in which switch tasks on a regular clocked interrupt, and on events called round robin.

Time sharing designs switch tasks more often than strictly needed but give smoother multitasking.

The basic building block of RTOS is the task. Actually task is a segment of codes which is treated by the system software as a program unit which can be started, stopped, suspended, resumed, and interrupts. There are 3 major states of a task running, Ready and Blocked. In case of running task only one task can be in the running condition. It is executing on the CPU. In case of Ready task, task is ready to be executed or run only RTOS is holding it. Any number of tasks can be in ready state. Blocked task cannot run. It is waiting for something to happen. Actually most of the tasks are blocked or ready state, most of the time because generally only one task can run at a time per CPU.

The number of items in the ready queue can vary greatly, depending in the number of tasks the system needs the perform and the type of scheduler that the system uses. On simpler non-preemptive but still multitasking systems, a task has to give up its time on the CPU to other tasks which can cause the ready queue to have a greater number of overall tasks in the ready to be executed state of well.

Uniprocessor Scheduling Algorithms

The set of uniprocessor real-time scheduling algorithms is divided into two major subsets, namely off-line scheduling algorithms and on-line scheduling algorithms. [1,5,9].

In systems using off-line scheduling, there is generally, if not always, a required ordering of the execution of processes. This can be accommodated by using precedence relations that are enforced during off-line scheduling. Preventing simultaneous access to shared

resources and devices is another function that a priority based preemptive off-line algorithm must enforce. Another goal that may be desired for off-line schedules is reducing the cost of context switches caused by preemption.

The real advantage of off-line scheduling is that in a predictable environment it can guarantee system performance.

On-line algorithms generate scheduling information while the system is running [7,9]. The on-line schedulers do not assume any knowledge of process characteristics which have not arrived yet. These algorithms require a large amount of run-time processing time. At run-time, a small on-line scheduler can choose the proper one. The major advantage of on-line scheduling is that there is no requirement to know tasks characteristics in advance and they tend to be flexible and easily adaptable to environment changes.

Static-priority based algorithms

Static-priority based algorithms are relatively simple to implement but lack flexibility. They are arguably the most common in practice and have a fairly complete theory. They work well with fixed periodic tasks but do not handle periodic tasks particularly well, although there are some methods to adapt the algorithms so that they can also effectively handle periodic tasks. Static priority-based scheduling algorithms have two disadvantages, which have received a significant amount of study. Their low processor utilization and poor handling of periodic and soft-deadline tasks have prompted researchers to search for ways to combat these deficiencies.

The two following non-preemptive algorithms attempt to provide high processor utilization while preserving task deadline guarantees and systems schedule ability.

Parametric dispatching algorithm [12,15]. This algorithm uses a calendar of functions, which maintains for each task two functions, Min, and Max, describing the upper and lower bounds on allowable start times for the task. During on off-line component, the timing constraints between tasks are analyzed to generate the calendar of functions.

Predictive algorithm [2,17] this algorithm depends upon known a priori task execution and arrival times. When it is time to schedule a task for execution, the scheduler not only looks at the first task in the ready queue, but also looks at the deadlines for tasks that are predicted to arrive prior to the first task's completion.

These algorithms have drawbacks when applied to real-world systems. These algorithms require significant a prior knowledge of the systems tasks, both execution times and ordering. Therefore, they are quite rigid and inflexible.

Dynamic-priority based algorithms require a large amount of on-line resources. However, this allows them to be extremely flexible. Many dynamic-priority based algorithms also contain on off-line component. This reduces the amount of on-line resources required while still retaining the flexibility of a dynamic algorithm. There are two subsets of dynamic algorithms: planning based and best effort.

Planning Based Algorithms guarantee that if a task is accepted for execution, the task and all previous tasks accepted by the algorithm will meet their time constraints [6,9]. The planning based algorithms attempt to improve the response and performance of a system to periodic and soft real-time tasks while continuing to guarantee meeting the deadlines of the hard real-time tasks.

The general model for these types of algorithms is a system where all periodic tasks have hard deadlines equal to the end of their period, their period is constant, and their worst's case execution times are constant. All a periodic tasks are assumed to have no deadlines and their arrival or ready times are unknown.

Best Effort Algorithms seek to provide the best benefit to the application tasks in overload conditions. The Best Effort scheduling algorithms seek to provide the best benefit to the application tasks. The best benefit that can be accrued by an application task is based on application-specified benefit functions such as the energy consumption function. More precisely, the objective of the algorithms is to maximize the accrued which is defined as the ratio of total accrued benefit to the sum of all task benefits.

There exist many best effort real-time scheduling algorithms. Two of the most prominent of them are the Dependent Activity Scheduling Algorithm (DASA) the Locke's. Best Effort Scheduling Algorithm (LBESA) . DASA and LBESA are equivalent to the Earliest Deadline First (EDF) algorithm during under loaded conditions where EDF is optimal and guarantees.

Multiprocessor Scheduling Algorithms

The scheduling of real-time systems has been much studied, particularly upon Uniprocessor platforms, that is, upon machines in which there is exactly one shared processor available, and all the jobs in the system are required to execute on this single shared processor. In multiprocessor platforms there are several processors available upon which these jobs may execute.

Feedback queue scheduler

Scheduler FQS Queue
Pid, Value
Pri _ first, pri_temp
Get _ Process _ d ( )
Get _ arrival _ time ( )

```
Get _burst  _ time ( )
Get _ Feedback _time ( )
Get _turn _ around _ time ( )
Get _ waiting _ time ( )
Process ( )
Scheduler ( )
Feedback queue scheduler –
# include scheduler _ parametric _ queue . cpp'
struct priority
{
int pid;
int value;
struct priority * next ;
};
class parametric_triple_queue_public scheduler
{
priority * pri_fist, * pri_temp;
priority * pri-second, * pri_temp-second;
priority * pri_third, * pri-temp_third;
priority * pri – feedback * pri _feedback;
int high – quantum, medium quantum,
Low _ quantum.
public;
int set _ value (int, int..)
void set quantum (int),
int compute ( );
void destory ( );
};
```

Result and Discussions-

In a feedback queue scheduling is the best algorithm for embedded systems. Despite this many well known RTOS's i.e. Win CE, embedded NT, Linux and pharLap also utilize priority time slicing. Under this algorithm, when a higher priority task becomes ready to run, it must wait until the end of the current time slice to be dispatched. Hence response time is governed by the granularly of the time slice. However granularity is set too fine, the processor spends too much time. In this case feed book algorithm and queue scheduling without Interrupting the current task to find out it a higher priority task is waiting and therefore higher priority task becomes ready to run without time delay.

References

1. MV Panduranga Rao,KC Seth(August2009)," A Research in Real Time Scheduling Policy for Embedded System Domain",CLEI Electronic Journal, Volume 12, August 2009.
2. Leung, Joseph; Zhao, Hairong (November 2005). Real-Time Scheduling Analysis (Technical report). DOT/FAA/AR-05/27.
3. Liu, Zhiming; Joseph, Mathai (17 February 2001). "Verification, Refinement and Scheduling of Real-time Programs" Theoretical Computer Science **253** (1): 119–152. Retrieved4 December 2015.
4. Gauthier L. ,etal "A automatic generation and targeting of application specific operating system and embedder system software," IEEE Trans. on Computer aided design of Integrated

circuits and systems,pp 1293-1301, 2005.

5. Sorin, Manolache,; Petru, Eles; Zebo, Peng (November 2004). " Schedulability Analysis of Applications with Stochastic Task Execution Times". ACM Transactions on Embedded Computing Systems **3** (4): 706–735. Retrieved 4 December 2015.
6. Audsley, N.; Burns, A. (1990). Real-Time System Scheduling (PDF) (Technical report). University of York, UK.
7. Derrick, Harris. "Big Data in real time is not fantasy". Gigaom. Retrieved 4 December 2015.
8. Castanet, R.; Laurençot, P. "Testing real-Time Systems". 15th World Conference on Nondestructive Testing. AIPnD. Retrieved 4 December 2015.
9. Tanenbaum, Andrew (2008). Modern Operating Systems. Upper Saddle River, NJ: Pearson/Prentice Hall. p. 160. ISBN 978-0-13-600663-3.
10. "Context switching time". Segger Microcontroller Systems. Retrieved 2009-12-20. Phraner, Ralph A. (Fall 1984). "The Future of Unix on the IBM PC". BYTE. pp. 59–64.
11. "2014 Embedded Market Study" (PDF). EE Live!. UBM. p. 44.
12. Ghose.,S etal," Fault Tolerant Rate Monotonic Scheduling", Journal of Real-Time Systems,pp.149-181,1998.
13. Manimaran.G.,etal,"A Fault tolerant dynamic scheduling algorithm for multiprocessor real time systems and its analysis" ,IEEE Trans on parallel and distributed system, volume 9,issue 11,pp 1137-1152, 1998.
14. Yamada S and Kusakabe S," Effect of context aware schedule on TLB," IEEE International Symposium on parallel Distributed processing" pp 1-8, 2008.
15. Lu.C.,Stankovic A, etal," Feedback control real time scheduling: frame work modeling and algorithm, special issue of real time systems Journal on control theoretic approaches to real time computing, vol.23,pp 85-126,2002.
16. chin-Lin Hu, 'on demand real time information dissemination: A general approach with fairness, productivity and urgency"21 International conference on advanced information networking and applications, pp 362-369,2005.