ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 2, Issue 4, pp: (87-91), Month: October - December 2014, Available at: www.researchpublish.com

# IMPLEMENTATION OF 'POLLING SERVER' IN RT SCHEDULING

Prof.G.K.Gaikwad, Prof.B.S.Chaudhary

Department of Computer Engineering, Lecturer in Sandip Polytechnic, Nashik, India

***Abstract:*** **Scheduling is a central activity of a computer system, usually performed by the operating system. Fixed-priority scheduler is the Polling-server in rate monotonic scheduling algorithm, which assigns higher priorities to tasks with shorter relative deadlines. It is intuitive to see that if every task's period is the same as its deadline, then the RM and DM scheduling algorithms are equivalent. Hard real-time systems have been defined as those containing processes that have deadlines that cannot be missed. Such deadlines have been termed hard: they must be met under all circumstances otherwise catastrophic system failure may result. To meet hard deadlines implies constraints upon the way in which system resources are allocated at runtime. This includes both physical and logical resources. Conventionally, resource allocation is performed by scheduling algorithms whose purpose is to interleave the executions of processes in the system to achieve a pre-determined goal. For hard real-time systems the obvious goal is that no deadline is missed. Polling Server monotonic priority ordering is similar in concept to rate monotonic priority ordering. Priorities assigned to processes are inversely proportional to the length of the deadline. Thus, the process with the shortest deadline is assigned the highest priority, the longest deadline process the lowest priority. This priority ordering defaults to a rate monotonic ordering when period=deadline.**

***Keywords:*** **scheduling, priority, monotonic, catastrophic.**

## I. INTRODUCTION

Real-time systems are computing systems in which the meeting of timing constraints is essential to correctness. If the system delivers the correct answer, but after a certain deadline, it could be regarded as having failed. In the context of real-time applications, the actions are tasks(also called processes) and the organization of their execution by the processors of thecomputing architecture (sequencing, interleaving, overlapping, parallel computing) iscalled real-time schedulingof tasks. The schedule must meet the timing constraints of the application; the procedure that rules the task execution ordering is called the scheduling policy.

The main objectives are to study the most significant real-time scheduling policies which are in use today in the industry for coping with hard real-time constraints. The bases of real-time scheduling and its major evolutions are described using unified terminology and notations. Basic on-line scheduling algorithms are designed with a simple rule that assigns priorities according to temporal parameters of tasks. If the considered parameter is fixed, i.e. request rate or deadline, the algorithm is static because the priority is fixed. The priorities are assigned to tasks before execution and do not change over time. The basic algorithms with fixed-priority assignment are rate monotonic and inverse deadline or deadline monotonic.Polling server in rate Monotonic algorithm overcomes the drawback of RM i.e. if an aperiodic task occur in the execution, it schedules It according to the free slots available.

## II. LITERATURE SERVEY

### *A. Real Time Systems*

Real-time systems are computing systems in which the meeting of timing constraints is essential to correctness. The correctness depends not only on the logical result but also the time it was delivered. If the system delivers the correct

answer, but after a certain deadline, it could be regarded as having failed.Real time systems fall into two different parts depending upon the behaviour of system under the time constraints. Types of real time systems are:

1. Hard real time systems.

2. Soft real time systems.

Hard real time is a system where "something very bad" happens if the deadline is not met. An overrun in response time leads to potential loss of life and/or big financial damage. Soft real time system is a system where the performance is degraded below what is generally considered acceptable if the deadline is missed. In this deadline overruns are tolerable, but not desired. There are no catastrophic consequences of missing one or more deadlines. These are often connected to Quality-of-Service(QoS).

***B. Real-time applications issues***

In real-time applications, the timing requirements are the main constraints and their mastering is the predominant factor for assessing the quality of service. Timing con- straints span many application areas, such as industrial plant automation, embedded systems, vehicle control, nuclear plant monitoring, scientific experiment guidance, robotics, multimedia audio and video stream conditioning, surgical operation moni- toring, and stock exchange orders follow-up. Applications trigger periodic or random events and require that the associated com- puter system reacts before a given delay or a fixed time.

The timing latitude to react is limited since transient data must be caught, actions have a constraint on both start and finish times, and responses or commands must be sent on time. The time scale may vary largely, its magnitude being a microsecond in a radar, a second in a human–machine interface, a minute in an assembly line, or an hour in a chemical reaction. The source of timing constraints leads to classifying them as hard or soft.

A real-time system has hard timing constraints when a timing fault (missing a deadline, delivering a message too late, sampling data irregularly, too large a scatter in data supposed to be collected simultaneously) may cause some human, economic or ecological disaster. A real-time system has soft timing constraints when timing faults can be dealt with to a certain extent. A real-time computer system is a computer system whose behaviour is fixed by the dynamics of the application. Therefore, a real-time application consists of two connected parts: the controlling real-time computer system and the controlled process.

Time mastery is a serious challenge for real-time computer systems, and it is often misunderstood. The correctness of system reactions depends not only on the logical results of the computations, but also on the time at which the results are produced. Correct data which are available too late are useless; this is a timing fault (Burns and Wellings, 1997; Lelann, 1990; Stankovic, 1988). A controlling real-time computer system may be built as: a cyclic generator, which periodically samples the state of the controlled process, computes the measured data and sends orders to the actuators (this is also called synchronous control).

***C. Real-time operating systems***

In real-time systems, resources other than the proces- sor are often statically allocated to tasks at their creation. In particular, time should not be wasted in dynamic memory allocation. Real-time files and databases are not stored on disks but reside in main memory; this avoids the non-deterministic disk track seeking and data access. Input–output management is important since the connections with the controlled process are various. Therefore, the main allocation parameter is processor time and this gives importance to the kernel and leads to it being named

The real-time operating system. Nevertheless, conventional operating sys- tem services are needed by real-time applications that have additional requirements such as, for example, management of large data sets, storing and implementing pro- grams on the computer also used for process control or management of local network interconnection. Thus, some of these conventional operating systems have been reengi- neered in order to provide a reentrant and interruptible kernel and to lighten the task structure and communication. This has led to real-time Unix implementations. The market seems to be showing a trend towards real-time systems proposing a Posix standard interface (Portable Operating System Interface for Computer Environments; international standardization for Unix-like systems).

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 2, Issue 4, pp: (87-91), Month: October - December 2014, Available at: www.researchpublish.com

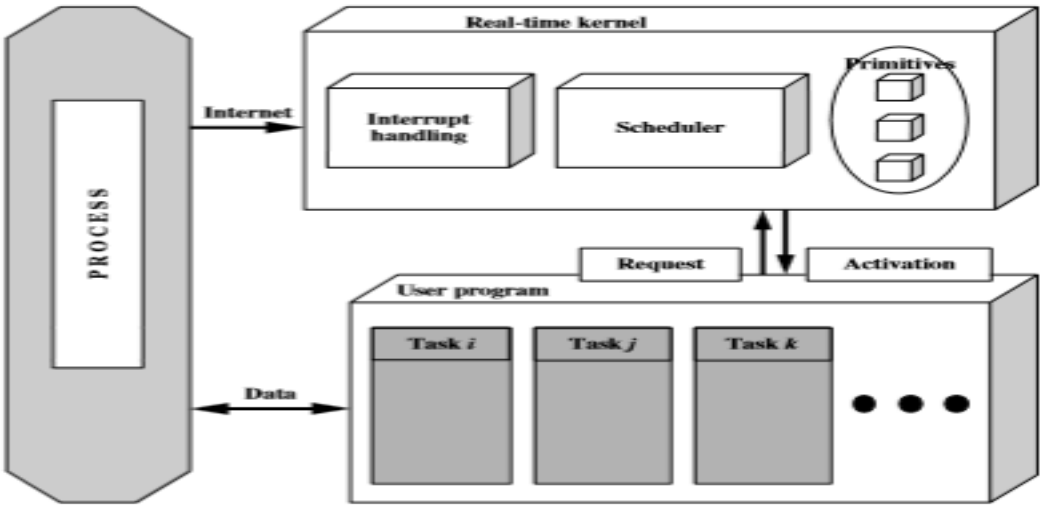


**Fig. 1. Schema of RTOS**

***Scheduling Algorithms***

Wide range of scheduling algorithms used:

- Rate monotonic (RM)
- Earliest deadline first (EDF)
- Least Laxity First (LLF)
- Deadline monotonic (DM)

***Rate Monotonic (RM)***

- The priority of a task is a monotonically decreasing function of its period.
- At any time, a highest priority task among all those that are ready for execution is allocated.
- If all assumptions are met, schedulability is guaranteed.
- For a single processor with n tasks, the following equation holds for the accumulated utilization μ:

***Earliest deadline first (EDF)***

- Each time a new ready task arrives.
- It is inserted into a queue of ready tasks, sorted by their deadlines.
- If a newly arrived task is inserted at the head of the queue, the currently executing task is pre-empted.

***Least Laxity First (LLF)***

- Priorities = decreasing function of the laxity (the less laxity, the higher the priority)
- Dynamically changing priority.
- Pre-emptive.
- Requires calling the scheduler periodically, and to recomputed the laxity. Overhead for many calls of the scheduler and many context switches. Detects missed deadlines early.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 2, Issue 4, pp: (87-91), Month: October - December 2014, Available at: www.researchpublish.com

***Deadline Monotonic (DM)***

- Deadlines less than period
- Priorities assigned according to Inverse of relative deadlines.

***Task Servers***

A server is a periodic task whose purpose is to serve aperiodic requests. A server is characterized by a period and a computation time called server capacity. The server is scheduled with the algorithm used for the periodic tasks and, once it is active, it serves the aperiodic requests within the limit of its capacity. The ordering of aperiodic requests does not depend on the scheduling algorithm used for periodic tasks. Several types of servers have been defined. The simplest server, called polling server, serves pending aperiodic requests at regular intervals equal to its period. Other types of servers (deferrable server, priority exchange server, sporadic server) improve this basic polling service technique and provide better aperiodic responsiveness. This section only presents the polling server, deferrable server and sporadic server techniques.

Polling server The polling server becomes active at regular intervals equal to its period and serves pending aperiodic requests within the limit of its capacity. If no aperiodic requests are pending, the polling server suspends itself until the beginning of its next period and the time originally reserved for aperiodic requests is used by periodic tasks.

Fig.2 shows an example of aperiodic service obtained using a polling server. The periodic task set is composed of three tasks, τ1(r0 = 0,C= 3,T = 20), τ2(r0 = 0,C= 2,T = 10) and τs(r0 = 0,C= 2,T = 5). τs is the task server: it has the highest priority because it is the task with the smallest period.

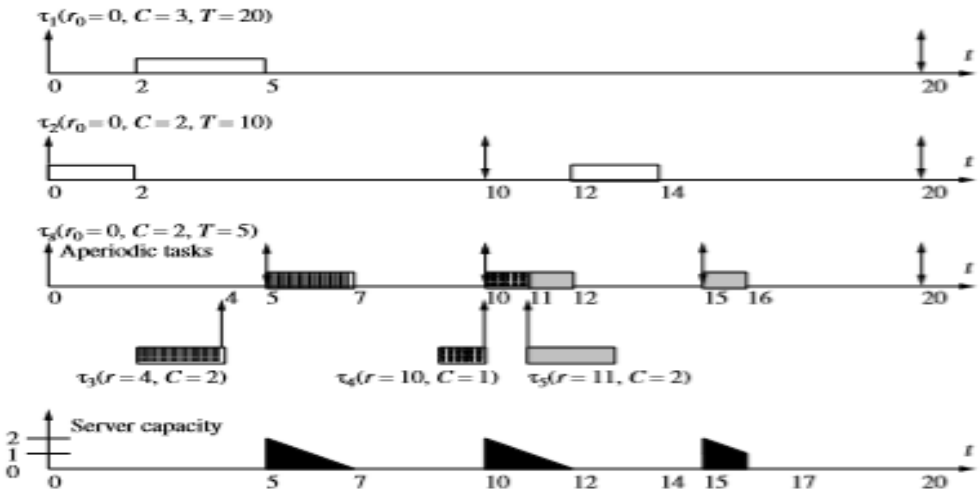


**Fig.2. Example of Polling Server**

The three periodic tasks are scheduled with the RM algorithm . The processor utilization factor is: 3/20+2/10+ 2/5= 0.75 < 3(21/3 −1) = 0.779. At time t = 0, the processor is assigned to the polling server. However, since no aperiodic requests are pending, the server suspends itself and its capacity is lost for aperiodic tasks and used by periodic ones. Thus, the processor is assigned to task τ2, then to task τ1. At time t = 4, task τ3 enters the system and waits until the beginning of the next period of the server (t = 5) to execute.

The entire capacity of the server is used to serve the aperiodic task. At time t = 10, the polling server begins a new period and immediately serves task τ4, which just enters the system. Since only half of the server capacity has been used, the server serves task τ5, which arrives at time t = 11. Task τ5 uses the remaining server capacity and then it must wait until the next period of the server to execute to completion. Only half of the server capacity is consumed and the remaining half is lost because no other aperiodic tasks are pending. The main drawback of the polling server technique is the following: when the polling server becomes active, it suspends itself until the beginning of its next period if no aperiodic requests are pending and the time reserved for aperiodic requests is discarded. So, if aperiodic tasks enter the system just after the polling server suspends itself, they must wait until the beginning of the next period of the server to execute.

## III. CONCLUSION

Here we have discussed all the issues related to the Polling Server Algorithm in Real Time System.We have got sufficient knowledge from the literature survey about other terminologies used in Real Time System.We have simulated Polling Server Scheduling Algorithm which uses Aperiodic independent task and we have verified it with the various case studies that we have seen. After Simulating so many case studies we came to conclude that yes it is true that the processor utilization condition in Polling Server Scheduling Algorithm is necessary but it is not sufficient one. We also observe that Polling Server Scheduling Algorithm only follows one parameter for scheduling that is it is deadline of process.The task with shortest deadline is given highest priority.It assumes that deadline is equal to period of invocation.

## REFERENCES

[1] http://www.artist-embedded.org/docs/Events/2005/Barcelona/ARTS4.pdf

[2] http://temporeal.lesc.ufc.br/downloads/Trabalho

[3] http://www.idt.mdh.se/kurser/ct3340/ht10/FinalPapers/2Lindh_Otnes_Wennestrom.p

[4] http://www.seas.upenn.edu/~lee/10cis541/lecs/lec-RT-sched-part2-v2-1x2.pdf

[5] http://www.cse.wustl.edu/~lu/cse467s/slides/rtos1.pdf

[6] https://csperkins.org/teaching/rtes/lecture07.pdf

[7] http://www.cs.fsu.edu/~baker/realtime/restricted/notes/servers.html