International Journal of Electronics Signals and Systems

Volume 3 | Issue 1 Article 15

July 2013

SELF OPTIMIZING KERNEL WITH HYBRID SCHEDULING ALGORITHM

AMOL VENGURLEKAR
Department of Electronics Engineering, D J. Sanghavi College of Engineering, Mumbai, India,
amol_1991@gmail.com

ANISH SHAH
Department of Electronics Engineering, D J. Sanghavi College of Engineering, Mumbai, India,
shahanish001@yahoo.com

AVICHAL KARIA
Department of Electronics Engineering, D J. Sanghavi College of Engineering, Mumbai, India,
avichalkaria@gmail.com

SELF OPTIMIZING KERNEL WITH HYBRID SCHEDULING ALGORITHM

AMOL VENGURLEKAR[1], ANISH SHAH[2] & AVICHAL KARIA[3]

[1,2&3]Department of Electronics Engineering, D J. Sanghavi College of Engineering, Mumbai, India
E-mail: amol_1991@gmail.com, shahanish001@yahoo.com, avichalkaria@gmail.com

Abstract— In this paper an operating system with a unique Hybrid scheduling algorithm (HSA) is proposed. The HSA incorporates the scheduling policies of both Round robin and priority based scheduling techniques along with a smart kernel which will self-optimize the system by providing a variable time slice to the tasks. The objective of developing this operating system is to improve the system efficiency and with an optimized usage of the resources available for the tasks, radically in terms of context switches and average waiting time.

Keywords*: Operating system, Hybrid scheduling algorithm, Self-optimize , variable time slice, context switcing.*

I. INTRODUCTION

An operating system is basically an interface between applications and physical resources. Operating Systems are basically classified under two major categories, Real Time Operating Systems and General Purpose Operating Systems.

A Real Time Operating System is the one which is deterministic in terms of its application and is very time sensitive whereas, General Purpose Operating System is the one which is non-deterministic, time insensitive and the one which can use virtual memory concept. Here an RTOS should be pre-emptible to enable multi-tasking. A system of priority inheritance has to exist along with a proper estimate of the interrupt latency so that we can have an optimized scheduling algorithm with which we can carry out two major functions of multi-tasking and resource guarding which will in turn affect the task handling capacity and improve the efficiency of the operating system making it more reliable.

II. FUNCTIONS OF KERNEL

A. Context Switching:

Context Switching is the process of switching over from one process to another in as operating system environment. The context consists of all the things that define the state of the process and the processor. This varies depending upon the architecture of the processor and the operating system constraints. Usually the context consists of the Program Counter, the Stack Pointer, all CPU registers. The entire context is saved on the process stack during the context switch.

Steps of Context Switching:

- Scheduler Interrupt arrives.
- All context of the current process is saved onto current process stack.
- The kernel is executed and scheduler is called.
- The scheduler determines the next process to be executed.
- The stack pointer is changed to the top of stack of the new process
- The context of the new process is popped from the process stack.
- Return from Scheduler Interrupt

B. Resources Guarding:

Resource guarding is a critical function of any operating system kernel. The kernel has to allocate resources to processes as and when the processes request them. If a particular resource is not available then the kernel cannot allow the process access to that resource. The process will then take appropriate action or the kernel will block the process from execution. For example if two processes request for a USART access then only one (whichever requests first) will get access to the USART. The kernel cannot allow both the processes to access because they will corrupt each other's communication. This is accomplished by programming practices like semaphores and mutexes. Situations like the 'Dining Philosophers Problem must also be handled by the kernel by effective manipulation of the semaphores.

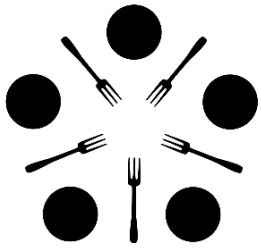


Figure (a) Dinning Philoshophers Problem

III. SCHEDULING ALGORITHMS

Whenever we have more than one task to handle, its necessary to decide the task that has to be executed

and proper resource allocation so that we can avoid the condition of deadlock. Some of the traditional scheduling algorithms are:

A. Shortest Job First:

In SJF priority is associated with each process, and the CPU is allocated to the process with the highest priority. Equal-priority processes are scheduled in FCFS order. An SJF algorithm is simply a priority algorithm where the priority (p) is the inverse of the (predicted) next CPU burst. The larger the CPU burst, the lower the priority, and vice versa. Here the execution of the task is also decided by the waiting time of the task.

Here we can take the following as an example of the SJF scheduling algorithm.

	Burst		Waiting	Turnaround
Process	Time	Priority	Time	Time
P_1	10	3	6	16
P_2	1	1	0	1
P_3	2	4	16	18
P_4	1	5	18	19
P_5	5	2	1	6
Average	-	-	8.2	12

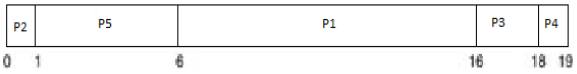


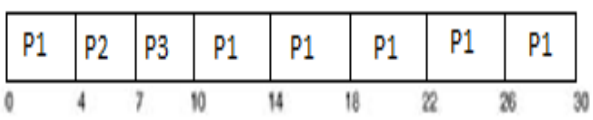
Figure (b) SJF

Thus according to the burst time and the waiting time in the above example task P2 is executed first and P4 at the end.

B. Round Robin scheduling algorithm:

The round-robin (RR) scheduling algorithm is designed especially for time-sharing systems. It is similar to FCFS scheduling, but pre-emption is added to switch between processes. A small unit of time, called a time quantum or time slice, is defined. A time quantum is generally from 10 to 100 milliseconds. The ready queue is treated as a circular queue.

	Burst	Waiting	Turnaround
Process	Time	Time	Time
P_1	24	6	30
P_2	3	4	7
P_3	3	7	10
Average	-	5.66	15.66



Figure(c) Round Robin

The above is an example which illustrates the round rin scheduling algorithm in which the time slice for a task is of four units. Here a task after being serviced for the provided number of time slices will have the lowest priority. Thus the above tasks will be executed in the order shown in figure (c).

IV. HYBRID SCHEDULING ALGORITHM

Many of the traditional algorithms concentrate on either varying the time slice provided or varying the priority of the task in order to implement task handling efficiently. Ours algorithm brings in a new concept of combining static prioritized scheduling with sub priorities to be executed in a round robin pattern. Our novel Hybrid Algorithm expands the domain to schedule to a high priority task in a proficient way.

Time Slice =100

		C0	C1	C2	C3	C4	C5	C6	C7
P0 (highest)	0	0	0	0	0	0	0	0	0
P1	0	0	0	0	0	0	0	0	0
P2	1	0	1	0	0	0	0	0	0
P3	0	0	0	0	0	0	0	0	0
P4	1	0	0	1	0	0	1	0	0
P5	1	0	1	0	0	0	0	0	0
P6	0	0	0	0	0	0	0	0	0
P7 (lowest)	1	0	0	0	0	0	0	0	0

Array Check

0	1	2	3	4	5	6	7

Eight available tasks in each priority level

Figure (d) HSA

The above figure illustrates the Hybrid Scheduling Algorithm, in which we have eight static priorities and each static priority has eight sub-priorities. To recognize the task which has the highest priority we have a concept of array check.

A. Array Check

Here we have an 8x8 matrix to represent occurrence of 64 different tasks. Further, to represent every row we have an 8x1 array. The array check is used to represent occurrence of a task in any row. According to the figure (d) the priority of task in row P0 is the highest and in P7 is the lowest.

B. Sub-Priority

In a row we have 8 different tasks with same priority. These tasks are scheduled in a round robin pattern based on the sub priority with C0 as the highest and C7 to be the lowest.

	Burst	Waiting	Turnaround
Process	Time	Time	Time
P2 ,C1	215	10	225
P4 ,C2	160	10	170
P4 ,C5	424	10	434
P5 ,C1	300	10	310

Figure (e) Tasks

The above figure (e) shows the sorting of the tasks as shown in figure (d) with respect to their priorities. The bits in the array check shown in figure (d) are used to identify the row in which tasks with the highest priority are available. It then checks for columns for the highest sub priority task which will be then executed.

Tasks ready in the same row are circularly queued for execution. Thus according to the algorithm P2,C1 will be executed first. The tasks P4,C2 and P4,C5 will be executed in round robin pattern after execution of P2,C1 and the task P5,C1 will be executed in the end.

V. SELF OPTIMIZATION IN HYBRID ALGORITHM

A general idea of improving on scheduling policy in a system seems inadequate. Hence we came up with a concept of self-optimization. The unique and effective idea of self-optimization during the task handling will be removal of the time taken by the kernel to shift between two tasks. Here, self-optimization will require a large number of samples of time slices required to get the task done or the amount of system ticks after which the task terminates itself. Based on these values factors like range, threshold and mean are calculated. These values are used to decide the self-optimizing factor.

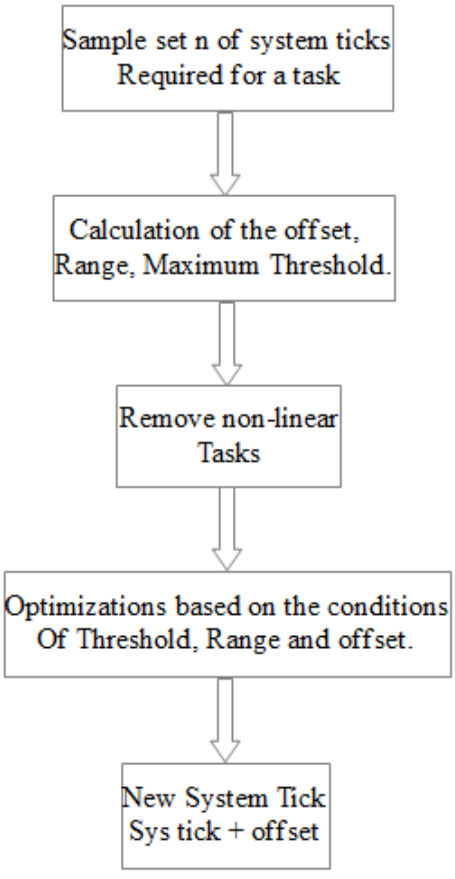


Figure (f): Flow Chart for Self Optimization.

The following figure shows execution of the tasks as per the suggested Hybrid Scheduling Algorithm with self-optimization, taking into account the priorities in both the rows and columns for problem in figure (e).

-	P2,C1	P2,C1	P2,C1	P4,C2	P4,C5	P4,C2	P4,C5	P4,C5	P4,C5	P4,C5	P5,C1	P5,C1	P5,C1
10	110	210	225	325	425	485	585	685	785	809	909	1009	1109

Figure (g)

Increasing the sample space and considering non-liner factors we can further improve the self-optimization factor. We are still working on process of incorporating various other factors like event t_max, event t_min and the above mentioned non-linear factor to further improve the process of self-optimization and implement it on a larger scale. Let us consider an example as shown in the figure (h) below for task1 to task4

Task1	100	100	10	100	100	15	100	100	14	100	100	13
Task2	100	100	100	20	100	100	100	25	100	100	100	23
Task3	100	10	100	100	10	100	100	100	12	100	100	100
Task4	100	20	100	40	100	50	100	35	100	10	100	50

Figure (h)

Taking modulous for all the system ticks we get

Task1	0	0	10	0	0	15	0	0	14	0	0	13
Task2	0	0	0	20	0	0	0	25	0	0	0	23
Task3	0	10	0	0	10	0	0	0	13	0	0	0
Task4	0	20	0	40	0	50	0	35	0	10	0	50

Figure(i)

The various deciding factors are:

Task	Range	Mean	Standard Deviation
Task1	5	13	3.74
Task2	5	22.67	3.55
Task3	3	11	2.23
Task4	40	31.17	37.77

Table (1)

Considering linear system with factors like mean and standard deviation:

Offset = Mean + Range

For the above tasks the offset values are

Task	Offset	Self-optimization
Task1	18	Yes
Task2	27.67	Yes
Task3	14	Yes
Task4	-	Non-linear system

Table (2)

VI. PARAMETERS FOR SELF OPTIMIZATION

- Range: After taking the modulus of the system ticks (Burst- time) with the time slice provided to the task , the maximum difference between non-zero values is known as range.
- Mean: The average of non-zero entities after taking the modulous is known as mean.
- Threshold (T_max): The maximum value of the non-zero enity after taking the modulous for which optimization is performed .
- Event T_max and Event T_min: The number of occurances of T_max and T_min respectively.
- Offset: Offset is self-optimizing factor which is integrated with the time slice to achieve optimization. The value of offset is the main factor to increase the effeciency of the system.

VII. COMPARISION BETWEEN ROUN-DROBIN, SJF & HSA WITH SELF OPTIMIZATION

A. Optimised Hybrid (HSA)Vs round robin

Consider the example of four tasks taken earlier in figure (e) where burst time is defined. The tasks will be serviced in a different way for different algorithms, it is as follows:

Round Robin

P2 C1	L	P4 C2	L	P4 C5	L	P5 C1	L	P2 C1	L	P4 C2	L	P4 C5	L	P5 C1	L	P2 C1	L	P4 C5	L	P5 C1	L	P4 C5	L	P4 C5
10 - 110	10	120 - 220	10	230 - 330	10	340 - 440	10	450 - 550	10	560 - 620	10	630 - 730	10	740 - 840	10	850 - 865	10	875 - 975	10	985 - 1085	10	1095 - 1195	10	1205 - 1229

Hybrid Algorithm with Optimization

P2 C1	L	P2 C1	L	P4 C2	L	P4 C5	L	P4 C2	L	P4 C5	L	P4 C5	L	P4 C5	L	P5 C1	L	P5 C5	L	P5 C1
10 - 110	10	120 - 235	10	245 - 345	10	355 - 455	10	455 - 515	10	525 - 625	10	635 - 735	10	745 - 869	10	879 - 979	10	989 - 1089	10	1099 - 1199

Figure (j)

As we can see that the number of system ticks required for executing all the four tasks in round robin type scheduling is more than the system ticks required in HSA with optimization. Here in the above example we have saved 30 system ticks, 0.024% of the total system ticks is saved when we used HSA in place of round robin for scheduling the tasks. For 'n' times execution of the tasks the saved ticks would be '30*n'.

B. Optimised Hybrid (HSA) Vs Shortest Job First

Shown below in the figure (k) is the task scheduling scheme using Optimized Hybrid Scheduling Algorithm and Shortest Job First scheduling algorithms for the problem in figure (e).

Shortest Job First

P2,C1	L	P4,C2	L	P4,C5	L	P5,C1
10-170	10	180-395	10	405-705	100	805-1129

Hybrid Algorithm with Optimization

P2 C1	L	P2 C1	L	P4 C2	L	P4 C5	L	P4 C2	L	P4 C5	L	P4 C5	L	P4 C5	L	P5 C1	L	P5 C5	L	P5 C1
10 - 110	10	120 - 235	10	245 - 345	10	355 - 455	10	455 - 515	10	525 - 625	10	635 - 735	10	745 - 869	10	879 - 979	10	989 - 1089	10	1099 - 1199

Figure (k)

Here the SJF alogirthm executes the tasks in less number system ticks but it does not switching to the other task untill it executes the task with comparivtively low burst time first. This gives you low flexibility as the programmer. Here the priority is always decided by the burst time which might not work well for various applications. All the above drawbacks are covered well in the optimized HSA algorithm.

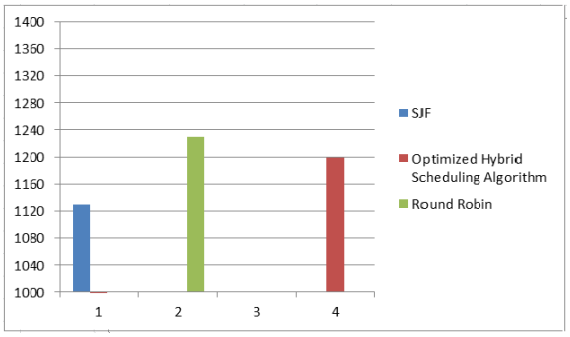


Figure (l): Comparison

The above graph shows the comparison for the various scheduling algorithms i.e. The SJF, Optimized HSA and Round Robin.

VIII. EXPERIMENTAL AND THEROITICAL RESEARCH

We are currently developing a Real Time Operating System based on our hybrid scheduling algorithm. We aim to be able to introduce a new task at runtime without any change in current execution. We have worked on an auto-optimization algorithm that will increase the overall efficiency of the operating system. We are trying to incorporate non-linear factors and other variables like Event_Tmax and Event_Tmin to further improve the efficiency of the system. For our RTOS we have chosen an LPC1768 microcontroller by NXP. The LPC1768 uses the ARM cortex-m3 architecture. This architecture provides us with an on chip dedicated 24 bit system tick timer which we use as our kernel timer. This microcontroller has an NVIC (Nested Vectored Interrupt Controller). It allows high priority interrupts to preempt low priority interrupts. It also supports Tail-Chaining and reduces the latency period.

IX. CONCLUSION

From the above experimental analysis it is clear that the proposed Hybrid Scheduling algorithm has advantages in terms of the user having more freedom in terms of priorities and it is more efficient when combined with the concept of self-optimization of the system.

REFERENCES

[1] White Paper - An Introduction to the ARM Cortex-M3 Processor by Shyam Sadasivan October 2006.

[2] David B. Stewart and Pradeep K. Khosla: Real-TimeScheduling of Dynamically Reconfigurable Systems,Proceedings of the IEEE International Conference on Systems pp 139-142, August, 1991.

[3] D. Probert, J.L. Bruno, and M. Karzaorman. SPACE: A new approach to operating system abstraction. In International Workshop on Object Orientation in Operating Systems, pages 133–137, October 1991.

[4] Yaashuwanth and R. Ramesh, : *A New SchedulingAlgorithm for Real Time System*, International Journal of Computer and Electrical Engineering (IJCEE), Vol. 2, No. 6,pp 1104-1106, December, 2010

[5] Exokernel: An Operating System Architecture for Application-Level Resource Management Dawson R. Engler, M. Frans Kaashoek, and James O'Toole Jr.M.I.T. Laboratory for Computer Science.

[6] http://www.quasarsoft.com/downloads/qKernelFeatureGuide.pdf