OFFICIAL TRANSCRIPT
WEBINAR: How to Prioritize RTOS Tasks (and Why it Matters)

Slide 1: How to Prioritize RTOS Tasks (And Why It Matters)

Sherry: Welcome and thank you for attending Barr Group's Webinar – How to Prioritize RTOS Tasks and Why it Matters. My name is Sherry and I will be your moderator for the next hour. Today's webinar presenter is Salomon Singer, Principal Engineer for Barr Group. For any of you live tweeting, you can see our Barr Group Twitter feed and our #barrwebinar on your screen now. Today's presentation will be approximately 40 minutes long, after which there will be a moderated question and answer session. To ask a question during the event, please type your question into the Q&A/chat area which is located near the bottom left of the webinar window. Please wait to send your technical questions until we get to the question and answer session when our presenter will be addressing them. As we are prepare to start the webinar, please make sure all of your other programs are closed so they do not affect your audio and video feeds. Before we get started with Salomon's presentation, I'm pleased to introduce Michael Barr our Chief Technical Officer who will provide a brief company overview.

Slide 2: About Barr Group

Michael Barr: Hello, everyone, and thank you for joining us. My name is Michael Barr. I'm a Co-Founder and the Chief Technical Officer of Barr Group. At Barr Group, our mission is to help as many people as possible build safer, more reliable, and more secure embedded systems. We are an independent consulting firm specializing in embedded systems and software. And we regularly consult with numerous companies in many industries on process and re-architecture for systems and software. As well, we take on consulting work of the design and development sort where we participate in the software, sometimes also electrical and mechanical design systems. As well, we train engineers in best practices through webinars like this, through public training events, and also at private companies. And finally, we sometimes, our experts sometimes testify before judges and juries about issues relating to embedded systems and embedded software.

Slide 3: Barr Group Training Courses

Before we begin today's webinar, I would just like to give you three URLs relating to our training courses. The first is for our training calendar for Upcoming Public Courses. This is where one person or a couple of people from a company can join others from the industry at a hotel meeting room or our headquarters training room to learn about different topics and if you go

to that first URL, you'll find a list of upcoming courses, dates, and locations. The second URL, I'd like to bring to your attention is for our Course Catalog. This is a list of all the courses that we offer including courses that are offered publicly, but also courses that are only offered at on-site locations at private companies. And of course, any of the courses in the catalog can be brought to your company or can be even sometimes be customized for your needs. And finally, we have an archive of all of our past webinars at this third URL, and that's where you can find after today's webinar has been transcribed and recorded, you can find this one there as well. And we have a number of other interesting and valuable webinars there. So with that, I turn the microphone over to today's speaker.

Slide 4: Instructor: Salomon Singer

Salomon Singer: Good afternoon ladies and gentlemen. My name is Solomon Singer, and it's my pleasure to be here today to talk to you about how to assign priorities to tasks in ISRs. If you want to prove that all the tasks in your system are going to meet all the deadlines you must first learn how to properly assign priorities to those tasks in ISRs.

Slide 5: Preemption

Before we start talking about how to assign priorities to tasks, we must start talking about preemption because only in the context of preemption are relative priorities relevant at all. So the definition of preemption is that the running task might be interrupted at any time. Why? A good example is because an interrupt just happened and so the CPU is going to treat the interrupt as a higher priority task, but also it is entirely possible that there is now a task that is ready to run that has higher priority than the current running task and DOS, he must get the CPU now. There are two main variations to preemption: one is time-slice and the other one is our priority-based; and one that we are going to be concentrating on is priority-based.

Slide 6: Priority-Based Preemption

When we are working with a priority-based preemption system, each and every one of the tasks that are comprised to system will be assigned a relative priority. Typically, this is a unique static integer, typically zero or one based, and the scheduling algorithm is extremely simple. The task with the highest priority gets the CPU as long as it is ready to run and it doesn't need anything else to do its work other than the CPU.

Slide 7: Priority-Based Preemption

So let's take a look at a simple example of a priority-based preemption system. In this particular example we have three tasks: Task A in purple has the highest priority; Task B in green has medium priority; and Task C in blue has the lowest priority. At T equals to zero, only Task C or the lowest priority task is ready to run, so he gets the CPU and while it's running and it hasn't finished all the work that it has to do Task B, which is medium priority becomes ready to run. And so at the next opportunity, the OS will perform a context switch and will replace Task C

with Task B. Now Task B runs for a while, and again he hasn't finished all the work he has to do when Task A which has higher priority becomes ready to run. And so at the next opportunity, the OS will replace Task B with Task A. Now A has the CPU, runs until it finishes its work, then the OS will swap it out because A doesn't need a CPU anymore, and so Task B which has medium priority will run. And in the example, he runs until it's finished all its work at which point the OS will bring tasks C, so the Task C can finish doing all the work that it wanted to do.

Slide 8: Commerical RTOSES

Commercial real-time operating systems are preemptive priority-based schedulers. And so when we design software for any of these systems we're going to separate the code into tasks, and each one of these tasks is going to be assigned a fixed priority. The highest priority task that is ready to run will always be running and that is the guarantee that the OS makes to us. If there is a running task that has lower priority than any others tasks in the ready list, there will be a context switch in that higher priority task will replace the lower priority task that is currently running. According to surveys, about 50% of firmware designers are currently working on projects that involve the commercial RTOS; however, the number of projects is much smaller than 50%. There is a significant size skew, the larger the project is the more likely it is that it will be using an RTOS.

Slide 9: What is the "Real-Time" in RTOS?

RTOS stands for Real-Time Operating System, but what exactly is the real-time in RTOS? So we can say that we have a real-time system, if some calculations or decisions have deadlines that we must meet. And so a late answer is the wrong answer. Real-time does not necessarily mean really fast. In fact, I'm not sure what really fast would mean. And so when deadlines are involved, it's real-time like the vast majority of embedded systems.

Let's take a look at an example where we have an assembly line that must place a lid on a bottle as it passes under the arm that places the lids. And so let's say we have a photo detector, and as soon as we see the bottle detected by this photo detector, we have a one millisecond plus-minus three microseconds to place the lid on this bottle. If we get there past the deadline, much past the deadline, we might miss the bottle completely and not place the lid at all. If we miss the deadline maybe by a little bit then the cap might go crooked on the bottle and by the same token, if we're early we might miss the bottle completely or the cap might go crooked slightly on the opposite side. Either way, it is not a good system if we are constantly missing bottles completely or placing lids that go crooked on the bottle.

Slide 10: Real-Time

So let's define real-time as having some type of timeliness requirements. Typically in the form of deadlines that cannot be missed, but now we need to be – we need to elaborate a little bit more and we need to talk about the difference between hard real-time and soft real-time.

Slide 11: Hard or Soft?

When we're talking about hard real-time, we're talking about deadlines that must be met each and every time. There is no skipping, there is no excuses. The result of missing one of these deadlines is typically some kind of irreversible loss. And so when we talk about hard real-time, I want you to think about industrial control, I want you to think about medical devices, right? It is not okay to ever miss a deadline for a pacemaker. It is not okay to ever miss a deadline for a respirator, et cetera, et cetera.

Soft real-time on the other hand, it's okay to miss a deadline every now and then, it is not desirable, but it's not the end of the world. And so here I want you to think for example, a set top box for a TV satellite receiver. It's getting 120 frames a second, and once every five minutes it might drop a frame. And so not a big to do, at the worse you might get a little bit of pixelation, but there is no irreversible damage, and more than likely nobody will be able to detect a dropped frame at a rate of 120 a second. And so not a big deal if we missed the deadline; not desirable, but not the end of the world.

Slide 12: Static Priority Assignment

So when it comes to static priority assignment, how exactly should developers set those static priorities? In many cases, developers will assign the highest priorities to those tasks they consider the most important. And if that's the case, more than likely those tasks with the highest priority will run whenever they need to and they should meet all or most of their deadlines. However, that is not the optimal algorithm for making sure that multiple tasks in the system meet their deadlines.

Slide 13: Transient Overload

So before I go any further I need to define transient overload, and it's going to be defined as a brief period of time for which the processor is overcommitted. What do I mean by that? So let's assume that I have 20 milliseconds worth of work to do, but I only have 10 milliseconds to do that work. Obviously not all the work is going to get done. Why? Because I only have 10 milliseconds to do the work and I have 20 milliseconds worth of work to do; so somebody is going to miss their deadline. This could easily happen if the stars align themselves just right in a bunch of interrupts of different priorities fire at about the same time.

Slide 14: Rate Monotonic Algorithm

Rate Monotonic Algorithm is a mathematical technique to select the relative priorities of tasks, and it is used in conjunction with an RTOS or some kind of fixed priority preemptive scheduler. It has a couple of extremely nice properties. The first one is that the critical set of task is guaranteed to meet all its deadlines and the second excellent property is the fact that performance is going to degrade very gracefully. Even during transient overloads, in other words, if we have a transient overload and we don't have enough time to perform all the tasks

that we need to perform in that period of time, then the lowest priority task will be the first one to yield the CPU. If that wasn enough for everybody else to meet their deadlines, that's fine, but if that wasn't enough then the next task with a lowest priority will yield the CPU, and so on and so forth until the rest of the task in the critical set can meet their deadlines.

Slide 15: RMA Example

Let's take a look at a very simple RMA example. We have two tasks: Task 1 and Task 2. Task 1 has a period of 50 milliseconds and Task 2 has a period of 100 milliseconds. And just for simplicity sake, for now we're going to assume that the deadlines are equals to that period so the deadline for Task 1 is at 50 milliseconds and the deadline for Task 2 is at 100 milliseconds.

Slide 16: Worst-Case Execution Time

Each time a task runs, it's going to consume CPU cycles, and each time a task runs it might take a different path through the code. Thus, there is a best-case scenario when it chews up the least amount of cycles and there is worst-case path when it chews up the most amount of cycles, and of course, there's the average case which is just the average number of CPU cycles that are meant to open any particular path. If the worst-case is longer than the period then the task is going to miss the deadline sometimes, and that sometimes is when it chooses any of those tasks which the time is longer than the period. There is absolutely no scheduling **algorithm that can fix this issue.**

And so we're going to say that each task has a computable worst-case cost and that is when it takes that longest path, and for the example that we're working on, we're going to say that the cost of Task 1 is 25 milliseconds and the cost of Task 2 is 40 milliseconds.

Slide 17: Priority Assignment

So we have two tasks: Task 1 with a period of 50 milliseconds and a cost of 25 milliseconds; Task 2 with a period of 100 milliseconds and a cost of 40 milliseconds. So how exactly should we select the relative priorities of these two tasks? In this particular case, we have only two prioritization options: Option A, Task 1 has a priority higher than Task 2; and Option B, Task 1 has a lower priority than Task 2.

Slide 18: Option A: P1 > P2

Let's see what happens under Option A where the priority of Task 1 is greater than the priority of Task 2. So at T equal zero, both tasks are ready to run, but Task 1 has a higher priority, so he gets to run first. He runs for 25 milliseconds, and at that point he has met the deadline that he had at 50. Now the OS swaps Task 1 with Task 2, and Task 2 gets to run for 25 milliseconds at which point Task 1 is ready to run again, and because he has higher priority, he gets the CPU, so there's another context switch. Task 1 runs for another 25 milliseconds at which point it satisfies its deadline that was at 100. And Task 2 gets the CPU again, he runs for 15 milliseconds

at which point he has satisfied his deadline that was at 100, and now there are 10 more milliseconds of idle-time, nobody – neither Task 1 nor Task 2 want the CPU, and T equals 100, the sequence repeats itself completely. And as we've seen, both Tasks 1 and 2 meet their deadlines every time.

Slide 19: Option B: P2 > P1

Now let's take a look at what happens under Option B where the priority of Task 2 is higher than the priority Task 1. So at T equals zero, both tasks are ready to run, but Task 2 has higher priority, and so he gets to run first. So Task 2 runs for 40 milliseconds at which point he has satisfied the deadline that he had for 100, and there is a context switch and Task 1 gets to run. After running 10 milliseconds, he has just blown his deadline. He had a deadline at 50 milliseconds, but he needed to do 25 milliseconds worth of work, and so it didn't complete. He continues to run until he does the 25 milliseconds worth of the work. He now completes even though he is 15 milliseconds too late and he gets to run again to do his next 25 milliseconds worth of work, and so he finishes at T equals 90 at which point he meets the deadline that he had at 100. And now between T equals 90 and T equals 100, neither T1 nor T2 want to run, so it's idle. At T equals 100, the sequence repeats itself all over again with T1 making only one of its two deadlines and T2 making its one deadline.

Slide 20: Rate Monotonic Algorithm

Rate Monotonic Algorithm is a priority assignment algorithm to be used with a real-time operating system. It is the optimal fixed-priority scheduling algorithm for those of you who are curious and want to see the math and approve of it, I refer you to journal of the ACM, an article by Liu and Leyland, 1973.

Slide 21: Rate Monotonic Algorithm

So how exactly does this rate monotonic algorithm work? We are going to assign task priorities according to their relative period. And so the highest frequency task or the task with the shortest period is going to get the highest priority, then after taking them out of the equation then the next highest frequency task is going to get the highest priority, and then after that the next highest frequency task will get the highest priority and so on and so forth. At the end the lowest frequency task or the task with the longest period is the one who is going to have the lowest priority.

The rationale behind this priority assignment algorithm is that least frequent task may span high frequency deadlines, and that is exactly what happened in the example that we just saw when Task 1 missed its deadline under Option B.

Slide 22: Exercise: Prioritize This

Let's take a look at this simple example. In this example, we have six tasks. Task R has a period of 10 milliseconds, Task T has a period of 30 milliseconds. Task G has a period of 1 millisecond, Task A has a period of 3 milliseconds, Task C has a period of 1 millisecond, and Task L has a period of 40 milliseconds. So how exactly should we prioritize these tasks?

Slide 23: Solution: Prioritize This

And here is the solution to the previous exercise. And so tasks C and G have the highest priority because they have the shortest period, after that comes Task A, then Task R, then Task T, and then at the end Task L. Notice that like in many real-time operating systems, the lower the number, the higher the priority.

Slide 24: Aperiodics Defined

Until now we've been talking only about periodic tasks, but that is not the only type of task that you might have in your system. It is entirely possible that you have an aperiodic task which is a task that recurs at random times like a spurious interrupt or you can have sporadic tasks that recur at random intervals, but with a minimum inter-arrival time. And so in this particular case, I want you to think UART, and so characters cannot come into UART any faster than the baud rate. And so you can think “occurs sporadically”. We don't know when any of these interrupts will happen, but we know that they cannot happen any faster than their minimum inter-arrival time which is a special case of aperiodic which is much easier to analyze.

Slide 25: Aperiodics (Including ISRS)

So we have two options to deal with aperiodics, and aperiodics will in this case include ISRs. Option 1 is to change the software, to poll the hardware rather than use interrupts. And so we have this periodic task that wakes up every so often, it polls the hardware and it does what it needs to do, and so this will work even for total aperiodics and we can guarantee a worst-case response time equal to the polling period. If we need faster response we could always change the period of the task that is to win the polling. Option 2 is to worst-case the analysis, which is a much better option. And so we are going to account for each of its worst-case period. In other words, we're going to assign a period equal to its worst-case inter-arrival time. So going back to the example with the UART, we're going to assume that we're getting characters into UART at a given baud rate 24/7 nonstop. And so we are going to then assign a priority based on that period.

Slide 26: Sporadic Server Abstraction

So we are not going to waste CPU cycles polling the hard work for these aperiodics. Instead, we're going to use this sporadic server abstraction and we are going to account for the period as if this aperiodics are ready to go and actually do go and run every time the minimum inter-arrival time has been met.

Slide 27: Interrupts and Priority

Let's take a look at a rather simple example. In this particular case, we have three tasks and one interrupt. And we need to remember that the CPU is going to treat the ISR as the highest priority task in the system regardless of its frequency or regardless of the RMA priority that we might have assigned to it. And so the ISR effectively is going to add a blocking time to the worst-case execution time of all tasks and it should be below an RMA priority.

Slide 28: Problem: ISR Priority vs. Task Priority

This table shows the periods and the RMA priorities for all three tasks and the ISR. As you can see, there is a pretty big conflict between RMA and reality right now. According to RMA, the ISR should be priority two because it has a period of 10 milliseconds. However, reality is that the ISR will have priority of one because the CPU is going to treat him as the highest priority task. And so how can we modify anything in this particular table to make sure that RMA and reality match again. Is there anything in this table that we can artificially tweak such that ISR now gets a priority of one?

And so the only – the answer to that question is – that the only thing that can be artificially tweaked is the period. We are going to worst-case the analysis even more. And if we reduce the period of the ISR to 2.9, now we can assign it a RMA priority of one and reality and RMA are all agreeing.

Slide 29: Problem: ISR Priority vs. Task Priority

This is the same table as in the previous slide, but I have now tweaked the period of the ISR to be 2.9 milliseconds, and now the real priorities jive perfectly with the RMA priorities, and there is no longer a conflict between RMA and reality.

Slide 30: Key Takeaways

So just to wrap up, please don't use “most important” to assign priorities. RMA is simple enough that it won't require too much of your time. Both tasks and interrupts must be assigned priorities using RMA, and we need to always remember that even the lowest priority ISR is treated as a higher priority than the highest priority task.