



universidade de aveiro

Dispositivos Conectados

Operação de um sistema embutido

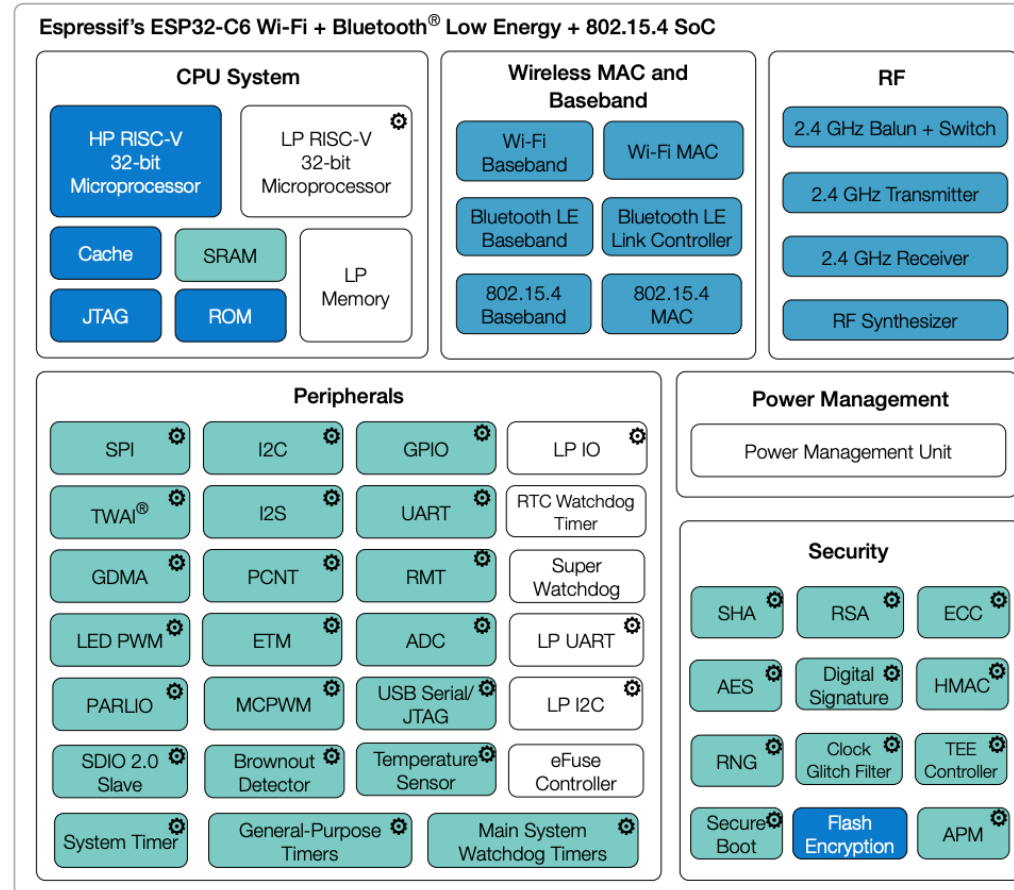
Paulo C. Bartolomeu

O que é um Sistema Embebido?

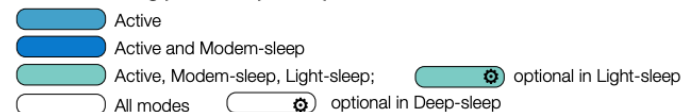
- **Combinação de *hardware* e *software*** que operam em conjunto para **realizar uma tarefa específica e pré-definida**.
- Ao contrário de um computador de uso geral (como um PC), que pode realizar uma vasta gama de tarefas, **um sistema embebido é dedicado e geralmente possui restrições de custo, energia e tamanho**.
- **Questão:** Qual é a diferença entre um *smartphone* (um computador de uso geral) e um forno micro-ondas (um sistema embebido).
 - O forno micro-ondas está permanentemente "embutido" no seu propósito.



Arquitetura do ESP32-C6



Modules having power in specific power modes:





Arquitetura RISC-V

- Arquitetura de conjunto de instruções (ISA) aberta, o que a torna popular em sistemas embebidos.
- Integra um núcleo HP (*High-Performance*) e um núcleo LP (*Low-Power*)
- O núcleo HP pode ser dedicado à aplicação principal e o outro a tarefas de baixo nível ou de comunicação.

Multitarefa de Tempo Real

- O modelo de programação do ESP32-C6, ao usar o ESP-IDF, é um modelo de Sistema Operativo de Tempo Real (RTOS focado em Tarefas (*Tasks*) concorrentes, e não no tradicional *loop* infinito de microcontroladores simples.
 - **Multitarefa Preemptiva:** Em vez de ter um único ciclo infinito onde todas as tarefas se realizam sequencialmente, o sistema é dividido em múltiplas tarefas independentes (*tasks*). O escalonador (*scheduler*) do FreeRTOS gere estas tarefas, decidindo qual deve correr em cada momento, com base na sua prioridade e no algoritmo de escalonamento adoptado.
 - **Concorrência:** É essencial para permitir que as funcionalidades críticas, como as *stacks* de Wi-Fi 6, Bluetooth LE, Thread e Zigbee, corram como tarefas de alta prioridade geridas pelo sistema, enquanto o código da aplicação executa-se noutras tarefas.
 - **A Função `app_main()`:** A função de entrada da aplicação (`app_main`) é a primeira tarefa a ser executada pelo FreeRTOS.
 - Usada para inicializar o *hardware* e o *software* e subsequentemente criar as tarefas permanentes da aplicação antes de retornar ou suspender-se.

Comunicação entre Tarefas

- **Notificações de Tarefa (Task Notifications):** Forma altamente eficiente e de baixo *overhead* para uma tarefa comunicar um evento ou transmitir dados a outra tarefa. É a primitiva de sincronização preferida do FreeRTOS em muitos casos.
- **Semáforos e Mutexes:** Utilizados para sincronização e exclusão mútua.
 -  **Mutex (Exclusive Mutual):** Usado para proteger um recurso partilhado (como um periférico ou uma variável global) de ser acedido por múltiplas tarefas ao mesmo tempo, prevenindo *race conditions*.
 -  **Semáforo (Contador ou Binário):** é usado para sinalização de eventos ou para limitar o acesso a um conjunto de recursos.
- **Filas (Queues):** Utilizadas para comunicação de dados entre tarefas.
 - Uma tarefa pode colocar dados na fila (produtor) e outra pode retirá-los (consumidor). É o principal mecanismo para desacoplar a lógica da aplicação.

Eventos e Drivers

O ESP-IDF complementa o FreeRTOS com um modelo de programação baseado em eventos para a gestão de *hardware* e comunicação:

- **Event Loop:** Os *drivers* de comunicação (como Wi-Fi e Bluetooth) funcionam com um Sistema de *Event Loop*.
 - Quando ocorre um evento de rede (ex: a conexão Wi-Fi foi estabelecida ou um pacote Bluetooth foi recebido), o sistema gera um Evento que é capturado pela aplicação.
- **Callbacks e ISRs:** O modelo faz uso intensivo de rotinas de serviço a interrupção (ISRs) de baixo nível para reagir rapidamente a eventos de *hardware* (como a mudança de estado de um pino GPIO).
 - Em geral, as ISRs fazem o mínimo de trabalho possível (apenas sinalizam uma tarefa) para depois passarem o processamento complexo para uma *Task* através de uma Fila ou Notificação.

Conceitos FreeRTOS

- **Multitarefa:** Execução de várias tarefas aparentemente em simultâneo (concorrência) através do *scheduler* do RTOS.
 - Permite que o ciclo principal da aplicação (a sua lógica) corra independentemente de tarefas de baixo nível como a gestão de rede (Wi-Fi/Bluetooth).
- **Prioridades:** Atribuição de prioridades a cada tarefa, garantindo que as funções críticas (*real-time*) são executadas primeiro.
 - Crucial para assegurar que a comunicação de rede e a resposta a eventos (ex: interrupções de sensores) ocorrem sem *delays* significativos.
- **Sincronização:** Uso de Notificações, Semáforos, Mutexes e Filas (*Queues*) para gerir o acesso a recursos partilhados (ex: uma variável global, uma porta UART, um periférico) e para comunicação entre tarefas.
 - Essencial para evitar condições de corrida (race conditions) e garantir a estabilidade do sistema.
- **Tempo Real:** Garantia de que as tarefas críticas têm um tempo de resposta previsível.
 - Permite o desenvolvimento de aplicações onde o *timing* é essencial, como controlo industrial ou processamento de áudio/vídeo.

Ambientes e linguagens de desenvolvimento p/ ESP32-C6

- **C/C++ (ESP-IDF):** Oferece o controlo máximo sobre a memória, *drivers* e o *kernel* FreeRTOS. É o ambiente recomendado para projetos robustos de IoT onde o desempenho e a gestão de recursos são críticos.
- **Python/MicroPython:** Embora o *core* do sistema ainda dependa do FreeRTOS, a utilização de linguagens de alto nível (como o MicroPython) abstrai o programador da complexidade do FreeRTOS, focando-o apenas na lógica da aplicação, à custa de alguma perda de desempenho e controlo de baixo nível.
- **Rust:** Existe também um suporte crescente para o desenvolvimento de aplicações em Rust, aproveitando as suas garantias de segurança de memória para sistemas embebidos.

FreeRTOS como base do ESP-IDF

- **Sistema Operativo:** O ESP-IDF não é apenas um conjunto de drivers e bibliotecas: é uma *framework* de desenvolvimento que é, ela própria, construída sobre o kernel FreeRTOS.
- **Ambiente de Execução:** Todas as aplicações e a maioria dos componentes *core* (como o Wi-Fi e Bluetooth) no ESP-IDF correm como Tasks do FreeRTOS.
- **Inicialização Automática:** Ao contrário de um ambiente *bare-metal* onde o programador tem de chamar `vTaskStartScheduler()`, no ESP-IDF, o FreeRTOS é iniciado automaticamente pela *framework* antes de chamar a função principal da sua aplicação (`app_main()`). A sua função `app_main` é executada como uma tarefa do FreeRTOS.

Porquê usar um sistema operativo de tempo-real?

- **Abstração das informações de tempo:** O RTOS é responsável pelo tempo de execução e fornece uma API relacionada ao tempo para a aplicação. Isso permite que a estrutura do código da aplicação seja mais direta e que o tamanho geral do código seja menor.
- **Manutenção/Extensibilidade:** abstração dos detalhes de temporização resulta em menos interdependências entre os módulos e permite que o software evolua de forma controlada e previsível. Além disso, o *kernel* é responsável pela temporização, portanto, o desempenho da aplicação é menos suscetível a alterações no *hardware* subjacente.
- **Modularidade:** As tarefas são módulos independentes, cada um dos quais deve ter um objetivo bem definido.
- **Desenvolvimento em equipa:** As tarefas também devem ter interfaces bem definidas, permitindo um desenvolvimento em equipa mais fácil.

Porquê usar um sistema operativo de tempo-real?

- **Testes mais fáceis:** Tarefas que são módulos independentes bem definidos com interfaces limpas são mais fáceis de testar isoladamente.
- **Reutilização de código:** O código projetado com maior modularidade e menos interdependências é mais fácil de reutilizar.
- **Maior eficiência:** O código da aplicação que usa um RTOS pode ser totalmente orientado a eventos. Não é necessário desperdiçar tempo de processamento a fazer *polling* a eventos que não ocorreram.
 - Opondo-se a esta eficiência, há a necessidade de processar a interrupção do RTOS e alternar a execução entre tarefas. No entanto, as aplicações que não utilizam um RTOS normalmente incluem alguma forma de interrupção de qualquer forma.
- **Tempo *Idle*:** A tarefa *idle* criada automaticamente é executada quando não há tarefas da aplicação que exijam processamento. A tarefa *idle* pode medir a capacidade de processamento disponível, realizar verificações em segundo plano ou colocar o processador em modo de baixo consumo de energia.

Porquê usar um sistema operativo de tempo-real?

- **Gestão de energia:** Os ganhos de eficiência resultantes da utilização de um RTOS permitem que o processador passe mais tempo em modo de baixo consumo de energia. O consumo de energia pode ser reduzido significativamente colocando o processador num estado de baixo consumo cada vez que a tarefa *idle* é executada.
- **Tratamento flexível de interrupções:** Os manipuladores (*handlers*) de interrupções podem ser mantidos muito curtos, adiando o processamento para uma tarefa criada pelo programador da aplicação ou para a tarefa *daemon* RTOS criada automaticamente (também conhecida como *timer task*).
- **Requisitos de processamento mistos:** Pode-se alcançar uma mistura de processamento periódico, contínuo e orientado a eventos dentro de uma aplicação. Além disso, os requisitos de tempo real rígidos e flexíveis podem ser atendidos selecionando as prioridades apropriadas de tarefas e interrupções.

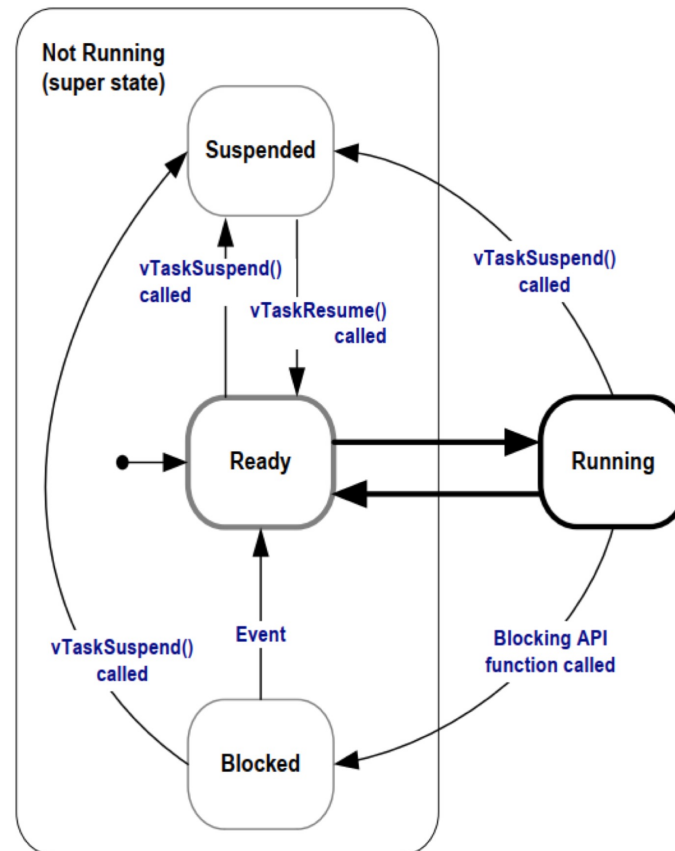


Porquê usar um
sistema operativo
de tempo-real?

Kahoot! *Time*



Máquina de estados de uma tarefa



Máquina de estados de uma tarefa

- Uma tarefa que está realmente em execução (usando tempo de processamento) está no estado **Running** (Em execução). Num processador de núcleo único, só pode haver uma tarefa no estado **Running** em qualquer momento.
- As tarefas que não estão realmente em execução, mas que não estão no estado **Blocked** (Bloqueado) ou **Suspended** (Suspendo), estão no estado **Ready** (Pronto).
- As tarefas no estado **Ready** estão disponíveis para serem selecionadas pelo escalonador como a tarefa a entrar no estado **Running**. O escalonador sempre escolherá a tarefa no estado Pronto com a prioridade mais alta para entrar no estado Em execução.
- As tarefas podem esperar no estado **Bloqueado** por um evento e são automaticamente movidas de volta para o estado **Pronto** quando o evento ocorre.
- Eventos temporais ocorrem num momento específico, por exemplo, quando um tempo de bloqueio expira, e são normalmente usados para implementar um comportamento periódico ou de tempo limite.
- Os eventos de sincronização ocorrem quando uma tarefa ou rotina de serviço de interrupção envia informações usando uma notificação de tarefa, fila, grupo de eventos, buffer de mensagens, buffer de fluxo ou um dos muitos tipos de semáforo. Geralmente são usados para sinalizar atividades assíncronas, como dados chegando a um periférico.



Máquina de estados de uma tarefa

- Deve haver sempre pelo menos uma tarefa que possa entrar no estado **Running**. Para garantir que isso aconteça, o escalonador cria automaticamente uma tarefa *Idle* quando `vTaskStartScheduler()` é chamado.
- A tarefa *idle* (ociosa) faz muito pouco além de ficar num ciclo, portanto está sempre pronta para ser executada.
- A tarefa ociosa tem a prioridade mais baixa possível (prioridade zero), para garantir que nunca impeça uma tarefa de aplicação com prioridade mais alta de entrar no estado **Running**.
 - No entanto, nada impede que os *developers* de aplicações criem tarefas com a prioridade da tarefa ociosa e, portanto, que a compartilhem, se desejarem.

Algoritmos de escalonamento de tarefas

- O algoritmo de escalonamento é a rotina de *software* que decide qual tarefa do estado **Pronto** transitará para o estado **Em Execução**.
- Em todas as configurações possíveis de núcleo único, o escalonador do FreeRTOS seleciona tarefas que partilham uma prioridade vez à vez. Essa política de "assumir a ordem" é frequentemente chamada de "Escalonamento Round Robin".
- Um algoritmo de escalonamento Round Robin não garante que o tempo seja partilhado igualmente entre tarefas de igual prioridade, apenas que tarefas no estado **Pronto** com a mesma prioridade entrem no estado **Em Execução** por vez à vez.

Configuração do algoritmos de escalonamento

- Preemptive: Algoritmos de escalonamento preemptivo "preemptarão" imediatamente a tarefa em estado de **Execução** se uma tarefa com prioridade maior que a tarefa em estado de **Execução** entrar no estado **Pronto**. Ser preemptivo significa ser movido involuntariamente do estado de **Execução** para o estado **Pronto** (sem ceder ou bloquear explicitamente) para permitir que uma tarefa diferente entre no estado de **Execução**. A preempção de tarefas pode ocorrer a qualquer momento, não apenas na interrupção de *tick* do RTOS.

Scheduling Algorithm	Prioritized	configUSE_PREEMPTION	configUSE_TIME_SLICING
Preemptive With Time Slicing	Yes	1	1
Preemptive Without Time Slicing	Yes	1	0
Co-Operative	No	0	Any



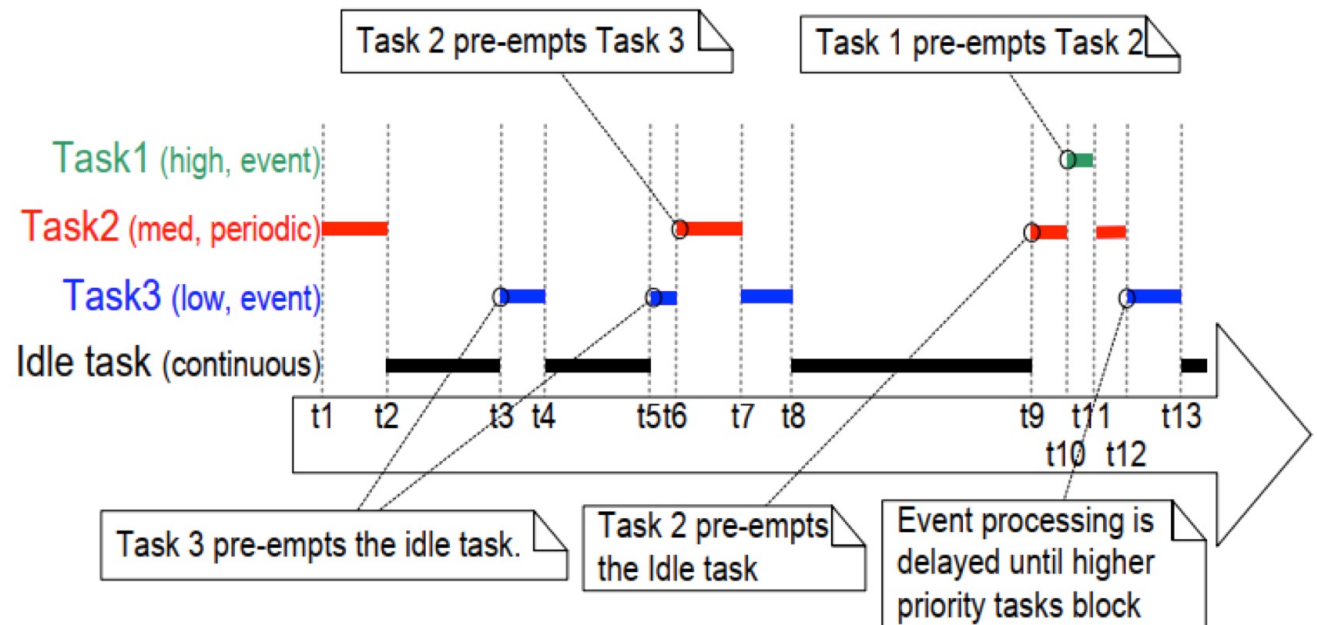
Configuração do algoritmos de escalonamento

- Time Slicing: O fatiamento (*slicing*) de tempo é usado para partilhar o tempo de processamento entre tarefas da mesma prioridade, mesmo quando as tarefas não cedem ou entram explicitamente no estado **Bloqueado**. Algoritmos de escalonamento descritos como usando Fatiamento de Tempo selecionam uma nova tarefa para entrar no estado **Em Execução** ao final de cada fatia de tempo se houver outras tarefas no estado **Pronto** com a mesma prioridade que a tarefa **Em Execução**. Uma *time slice* é igual ao tempo entre duas interrupções de *tick* do RTOS.

Scheduling Algorithm	Prioritized	configUSE_PREEMPTION	configUSE_TIME_SLICING
Preemptive With Time Slicing	Yes	1	1
Preemptive Without Time Slicing	Yes	1	0
Co-Operative	No	0	Any

Algoritmo de agendamento preemptivo de prioridade fixa com time slicing

- Padrão de execução destacando a priorização e a preempção de tarefas numa aplicação hipotética na qual cada tarefa recebeu uma prioridade única.



Algoritmo de agendamento preemptivo de prioridade fixa com time slicing

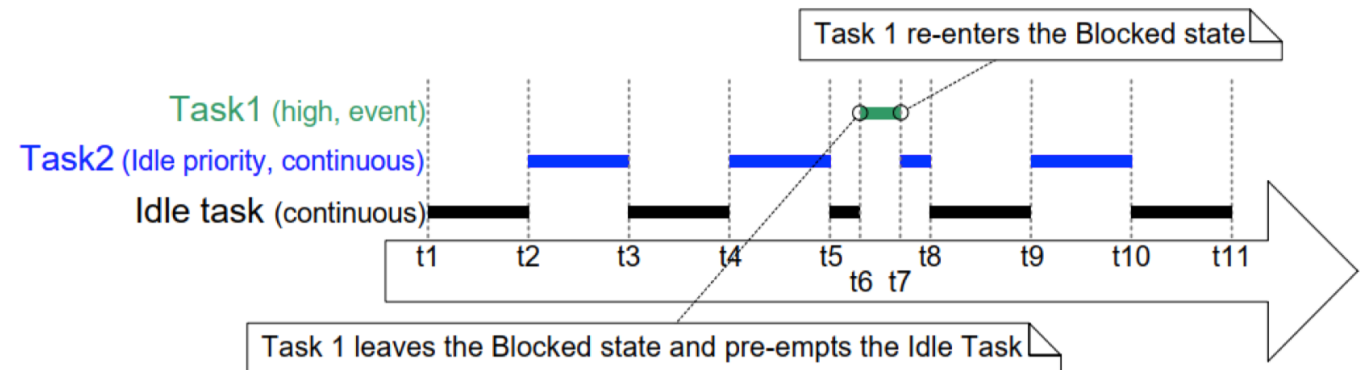
- **Idle task:** A tarefa *idle* é executada na prioridade mais baixa. É interrompida sempre que uma tarefa de prioridade mais alta entra no estado **Pronto**, por exemplo, nos momentos t3, t5 e t9.
- **Task 3:** A Tarefa 3 é uma tarefa orientada a eventos que é executada com uma prioridade relativamente baixa, mas acima da prioridade **Ociosa**.
 - Passa a maior parte do tempo no estado **Bloqueado**, aguardando o evento de interesse, transitando do estado **Bloqueado** para o estado **Pronto** sempre que o evento ocorre.
 - Todos os mecanismos de comunicação entre tarefas do FreeRTOS (notificações de tarefas, filas, semáforos, grupos de eventos, etc.) podem ser usados para sinalizar eventos e desbloquear tarefas dessa maneira.
 - Os eventos ocorrem nos instantes t3 e t5, e também em algum ponto entre t9 e t12. Os eventos que ocorrem nos instantes t3 e t5 são processados imediatamente porque, nesses instantes, a Tarefa 3 é a tarefa de maior prioridade que pode ser executada.
 - O evento que ocorre em algum ponto entre t9 e t12 não é processado até t12 porque, até então, as tarefas de maior prioridade, Tarefa 1 e Tarefa 2, ainda estão em execução. É somente no instante t12 que as Tarefas 1 e 2 estão no estado **Bloqueado**, tornando a Tarefa 3 a tarefa de maior prioridade no estado **Pronto**.

Algoritmo de agendamento preemptivo de prioridade fixa com time slicing

- **Task 2:** A Tarefa 2 é uma tarefa periódica executada com uma prioridade acima da prioridade da Tarefa 3, mas abaixo da prioridade da Tarefa 1. O intervalo de tempo da tarefa significa que a Tarefa 2 deseja executar nos tempos t_1 , t_6 e t_9 .
 - No instante t_6 , a Tarefa 3 está no estado **Em Execução**, mas a Tarefa 2 tem a prioridade relativa mais alta, então interrompe a Tarefa 3 e inicia a execução imediatamente.
 - A Tarefa 2 conclui seu processamento e retorna ao estado **Bloqueado** no tempo t_7 , momento em que a Tarefa 3 pode retornar ao estado **Em Execução** para concluir seu processamento. A própria Tarefa 3 bloqueia no tempo t_8 .
- **Task 1:** A Tarefa 1 também é uma tarefa orientada a eventos. Ela é executada com a maior prioridade de todas, podendo, portanto, sobrepor-se a qualquer outra tarefa no sistema.
 - O único evento da Tarefa 1 mostrado ocorre no instante t_{10} , momento em que a Tarefa 1 sobrepõe-se à Tarefa 2. A Tarefa 2 só pode concluir seu processamento após a Tarefa 1 retornar ao estado **Bloqueado** no instante t_{11} .

Algoritmo de agendamento preemptivo de prioridade fixa com time slicing

- Padrão de execução destacando a priorização de tarefas e o fatiamento de tempo numa aplicação hipotética no qual duas tarefas são executadas com a mesma prioridade.



Algoritmo de agendamento preemptivo de prioridade fixa com time slicing

- **Idle task** e **Task 2**: A tarefa *idle* e a tarefa 2 são tarefas de processamento contínuo e ambas têm prioridade 0 (a menor prioridade possível).
 - O escalonador aloca tempo de processamento para as tarefas de prioridade 0 somente quando não há tarefas de prioridade mais alta que possam ser executadas e compartilha o tempo alocado para as tarefas de prioridade 0 por meio de fatiamento de tempo.
 - Uma nova fatia de tempo inicia a cada interrupção de *tick*, o que na Figura 4.19 ocorre nos tempos t1, t2, t3, t4, t5, t8, t9, t10 e t11.
 - A tarefa *idle* e a tarefa 2 entram no estado de execução alternadamente, o que pode resultar em ambas as tarefas permanecerem no estado de execução por parte da mesma fatia de tempo, como ocorre entre os tempos t5 e t8.

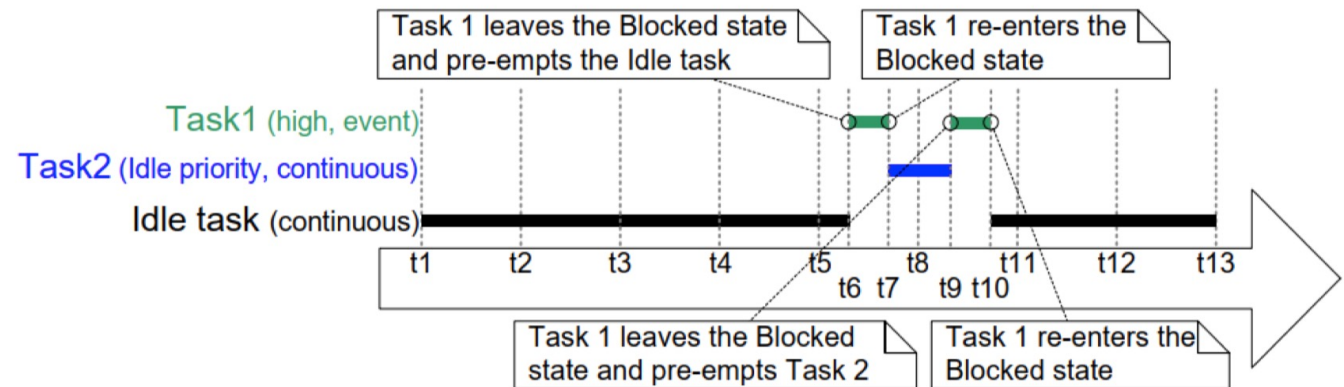
Algoritmo de agendamento preemptivo de prioridade fixa com time slicing

- **Task 1:** A prioridade da Tarefa 1 é maior que a prioridade **Ociosa**.
 - A Tarefa 1 é uma tarefa orientada a eventos que passa a maior parte do tempo no estado **Bloqueado**, aguardando seu evento de interesse, transitando do estado **Bloqueado** para o estado **Pronto** sempre que o evento ocorre.
 - O evento de interesse ocorre no instante t_6 . Em t_6 , a Tarefa 1 se torna a tarefa de maior prioridade que pode ser executada e, portanto, a Tarefa 1 antecipa a tarefa **Ociosa** no meio de um intervalo de tempo.
 - O processamento do evento é concluído no instante t_7 , momento em que a Tarefa 1 retorna ao estado **Bloqueado**.

A Figura mostra a tarefa *idle* a partilhar o tempo de processamento com uma tarefa criada pelo *developer* da aplicação. Alocar tanto tempo de processamento para a tarefa *idle* pode não ser desejável se as tarefas prioritárias ociosas criadas pelo autor do aplicativo tiverem trabalho a fazer, mas a tarefa *idle* não.

Algoritmo de agendamento preemptivo de prioridade fixa sem time slicing

- Padrão de execução que demonstra como tarefas de igual prioridade podem receber quantidades muito diferentes de tempo de processamento quando o fatiamento de tempo não é usado.



Algoritmo de agendamento preemptivo de prioridade fixa sem time slicing

- O Agendamento Preemptivo priorizado sem fatiamento de tempo mantém os mesmos algoritmos de seleção e preempção de tarefas descritos antes, mas não utiliza fatiamento de tempo para compartilhar o tempo de processamento entre tarefas de igual prioridade.
- Como referido, se o fatiamento de tempo for utilizado e houver mais de uma tarefa em estado pronto com a prioridade mais alta capaz de ser executada, o escalonador seleciona uma nova tarefa para entrar no estado Em Execução durante cada interrupção de tique do RTOS (uma interrupção de tique marca o fim de uma fatia de tempo).
- **Se o fatiamento de tempo não for utilizado, o escalonador somente seleciona uma nova tarefa para entrar no estado Em Execução quando:**
 - Uma **tarefa de prioridade mais alta entra no estado Pronto**.
 - A tarefa em estado **Em Execução** entra no estado **Bloqueado** ou **Suspenso**.
- Há menos trocas de contexto de tarefas quando o fatiamento de tempo não é usado do que quando o fatiamento de tempo é usado.
 - Portanto, desativar o fatiamento de tempo resulta em uma redução na sobrecarga de processamento do escalonador.
 - No entanto, desativar o fatiamento de tempo também pode fazer com que tarefas de mesma prioridade recebam quantidades muito diferentes de tempo de processamento, um cenário demonstrado.

universidade de aveiro



theoria poiesis praxis