highintegritysystems

WITTENSTEIN

OPENRTOS® データシート

OPENRTOS® は、幅広い組込みプロセッサをサポートする使いやすいリアルタイム OS です。最小限のリソースで高いパフォーマンスを提供します。

機能概要

OPENRTOS® のリアルタイム OS は、以下の機能と利点を備えています。

- プリエンプティブ、協調型、ラウンドロビン、ハイブリッドのスケジューリング方式に対応
- システム RAM の制約を除き、任意の数のタスクを生成可能
- 各タスクに優先度を割り当て可能（任意の数の優先度を使用可能）
- 同一優先度のタスクを複数生成可能
- タスクの間や割り込みサービスルーチンとタスク間のデータ送受信にキューを使用
- キューセットにより、複数のキューを監視してブロック可能
- バイナリセマフォおよびカウントセマフォはキュープリミティブを使用して実装され、コードサイズを最小限に抑制
- 優先度継承機能付きミューテックス
- 再帰ミューテックス
- タスク通知機能
- ソフトウェアタイマ
- イベントフラグ
- FPU 対応
- 実行時間統計
- カーネル対応プラグインとプロファイリングツール
- 低消費電力対応のティックレスモード
- 統合されたミドルウェアと BSP を提供
- 初年度の無償サポート＆メンテナンス付属

コンパクトなフットプリント

典型的な ROM 要件	4-32K bytes
典型的な RAM 要件	1 k bytes
典型的なスタック要件	**タスクあたり** 400 bytes

主な特徴

小型・高応答・使いやすい RTOS
幅広いユーザー層に支持される実績ある製品
フルソースコードを提供
主要な組込みプロセッサをサポート

システム概要

OPENRTOS® をアプリケーションに組み込むことで、アプリケーションは複数の独立したタスクの集合として構成できるようになります。システム全体の機能は、これら複数のタスクがそれぞれ担う機能の合計として実現されます。

各タスクは、それぞれ独立したコンテキストで実行され、システム内の他のタスクやスケジューラに偶発的に依存することはありません。

タスク状態

任意の時点で、実際に実行されるタスクは常に 1 つだけです。
スケジューラは、各タスクの相対的な優先度および状態に基づいて、実行するタスクを選択する役割を担います。

タスクは、「表 1. タスク状態」に示されたいずれかの状態を取ります。
状態の遷移は「図 1. 有効なタスク状態遷移図」に示されています。

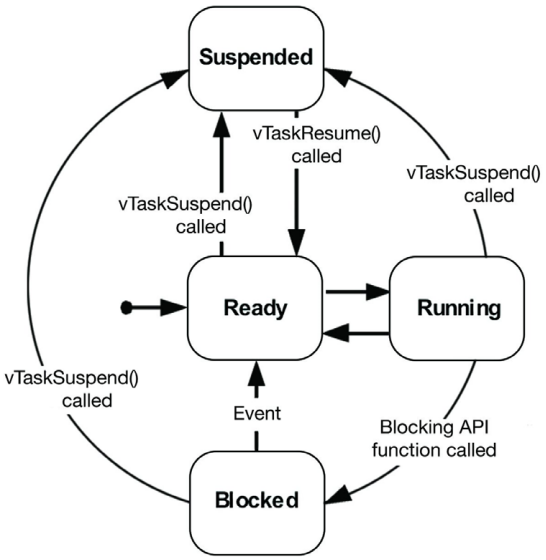

図 1. 有効なタスク状態遷移図

DIOIZ
Enhance Your Embedded System

株式会社ディオイズジャパン
https://www.dioiz-japan.com
dioiz@dioiz.com

タスク状態	説　明
実行状態 (Running)	スケジューラによって実行対象として選ばれ、現在プロセッサを使用しているタスクです。
ブロック状態 (Blocked)	タスクは、時間的なイベントや外部イベントを待っている場合にブロック状態になります。たとえば、タスクが vTaskDelay API を呼び出すと、指定された遅延時間が経過するまでブロックされます。また、キューやセマフォのイベントを待つためにブロックされることもあります。 ブロック状態のタスクには常に「タイムアウト」期間が設定されており、その期間が経過しますと、タスクは自動的にブロック解除されます。
サスペンド状態 (Suspended)	vTaskSuspend API 関数が呼び出されたときに「サスペンド状態」へ遷移します。その後、vTaskResume API 関数が呼び出されるまで、「サスペンド状態」のままとなります。
実行可能状態 (Ready)	実行可能な状態であるにもかかわらず、現在実行対象として選ばれていない場合に「実行可能状態」になります。

表 1. タスクの状態

タスクの優先度

各タスクには､作成時に優先度が割り当てられます。タスクの優先度は、実行中に変更することも可能です。

数値が小さいほど優先度は低く、最も低い優先度は 0 です。数値が大きいほど優先度は高く、タスクに割り当て可能な最大優先度はユーザーが設定できます。

スケジューラ

スケジューラの主な役割は以下のとおりです。

- 「実行状態」に入るタスクを選択し、それに応じたコンテキストスイッチを実行すること
- 時間の経過を計測すること
- タイムアウトの満了やセマフォ・キューの操作などの外部イベントに応じて、タスクを「ブロック状態」から「実行可能状態」へ遷移させること

時間の測定

時間の計測には、周期的な（ティック）タイマー割り込みが使用されます。

2 回連続するタイマー割り込みの間隔を 1 ティック周期と定義し、時間はこの「ティック」単位で測定および指定されます。ティックの周波数は、コンパイル時にユーザーが設定可能です。

プリエンプティブスケジューリングポリシー

プリエンプティブスケジューリングを使用する場合、OPENRTOS® は「実行可能状態」にあるタスクの中から、最も高い優先度のタスクを「実行状態」に選出します。言い換えれば、実行対象として選ばれるのは、実行可能な中で最も高い優先度を持つタスクです。「ブロック状態」や「サスペンド状態」にあるタスクは、実行対象にはなりません。

協調型スケジューリングポリシー

OPENRTOS® を協調型スケジューラとして使用する場合、コンテキストスイッチは、実行中のタスクが「ブロック状態」に入るか、明示的に TaskYIELD マクロを呼び出すときにのみ発生します。タスクは決してプリエンプトされず、同じ優先度のタスク同士が自動的に処理時間を共有することもありません。

このような協調型スケジューリングは構造が単純である一方で、システムの応答性が低下する可能性があるという欠点もあります。

ラウンドロビンスケジューリングポリシー

プリエンプティブスケジューリングモードでは、異なるタスクに同じ優先度を割り当てることができます。この場合、同じ優先度を持つタスクは順番に「実行状態」へ選ばれます。オプションとして、各タスクは最大で 1 ティック期間だけ実行され、その後スケジューラが次の同じ優先度のタスクを「実行状態」に選出します。

スケジューラは同じ優先度のタスクを順番に実行させますが、すべてのタスクが処理時間を均等に得られることは保証されません。

ハイブリッドスケジューリングポリシー

ハイブリッド方式では、割り込みサービスルーチンから明示的にコンテキストスイッチを発生させることができます。これにより、同期イベントによってはプリエンプション（タスクの強制切り替え）が発生しますが、時間的なイベントによる切り替えは行われません。その結果、タイムスライスなしのプリエンプティブなシステムが構成されます。この方式は、効率性の向上という点で好まれる場合があります。

譲渡

タスクの譲渡とは、自発的に「実行状態」を離れて「実行可能状態」へ戻る動作を指します。タスクが譲渡を行うと、スケジューラはどのタスクを「実行状態」にすべきかを再評価します。譲渡したタスクよりも高い、または同じ優先度のタスクが「実行可能状態」に存在しない場合は、再びその譲渡したタスクが「実行状態」に選ばれます。譲渡は、TaskYIELD マクロを明示的に呼び出すか、アプリケーション内で他のタスクの状態や優先度を変更する API を呼び出すことで行うことができます。

株式会社ディオイズジャパン
https://www.dioiz-japan.com
dioiz@dioiz.com

スケジューラの状態

スケジューラは、「表 2. スケジューラの状態」に記載された、いずれかの状態を取ります。状態間の有効な遷移は、「図 2. スケジューラの有効な状態遷移図」に示されています。

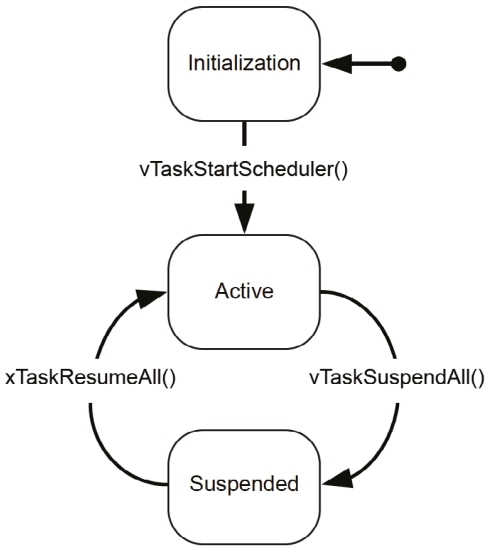

図 2. スケジューラの有効な状態遷移図

スケジューラは、vTaskStartScheduler API の呼び出しによって開始されます。この関数が呼ばれると、アイドルタスクが生成されます。
アイドルタスクは「ブロック状態」や「サスペンド状態」に遷移することはありません。これは、常に実行可能なタスクが少なくとも 1 つ存在する状態を確保するために生成されます。

スケジューラは、vTaskSuspendAll API の呼び出しにより「サスペンド状態」に入り、xTaskResumeAll API の呼び出しにより「アクティブ状態」に戻ります。

クリティカルコードセクション

クリティカルセクションとは、データの整合性を保証するために、他のタスクや割り込みによる割り込みなしに実行されなければならないコード領域を指します。クリティカルセクションを実装する従来の方法は、該当領域への進入時に割り込みを無効にし、終了時に再度有効にするという手法です。この目的のために、taskENTER_CRITICAL と taskEXIT_CRITICAL マクロが用意されています。

ただし、これらのマクロによる実装には欠点があり、クリティカルセクションの実行中は一部の割り込みに応答できなくなるという問題があります。この課題に対する代替手段として、スケジューラの一時停止機構があります。

スケジューラの状態	説明
初期状態	スケジューラが開始される前の初期状態を「初期化状態」と呼びます。 この状態では、スケジューラはアプリケーションの実行を制御していません。ただし、「初期化状態」でもタスクやキューの作成は可能です。
アクティブ状態	スケジューラがアプリケーションの実行を制御し、「実行状態」にあるタスクを選択して実行します。
サスペンド状態	スケジューラが「サスペンド状態」に入った時点で「実行状態」にあったタスクは、スケジューラが「アクティブ状態」に戻るまで引き続き実行状態を維持します。

表 2. スケジューラの状態

vTaskSuspendAll API を呼び出すことでスケジューラを一時停止状態にすると、その間はタスクの切り替えが一切発生せず、現在実行中のタスクが vTaskResumeAll API を呼ぶまで「実行状態」を保持します。

この方法では、スケジューラが停止していても割り込みは有効なままなので、他のタスクからのアクセスは防げますが、割り込みからのアクセスは防げません。そのため、スケジューラが「サスペンド状態」にある間でも、割り込みサービスルーチンからキューを利用してアクセスするのが安全です。

なお、スケジューラが停止している間に高優先度のタスクが「実行可能状態」になった場合でも、vTaskResumeAll API が呼ばれるまで切り替えは保留されます。そのため、スケジューラの停止状態を長時間維持するのは望ましくなく、高優先度タスクの応答性が低下する可能性があります。

メモリ管理

OPENRTOS® は、タスク、キュー、ミューテックス、ソフトウェアタイマー、セマフォが生成されるたびに RAM を確保します。組込みシステムは、用途によって RAM の容量やタイミング要件が大きく異なるため、すべてのアプリケーションに適した単一のメモリ管理方式というものは存在しません。この課題に対応するために、OPENRTOS® では用途に応じて選択可能な 5 種類のメモリ管理方式が用意されており、詳細は、「表 3. メモリ管理方式」に記載されています。

株式会社ディオイズジャパン
https://www.dioiz-japan.com
dioiz@dioiz.com

メモリ管理方式	説明
スキーム 1	メモリ要求に応じて単一の配列を小さなブロックに分割していきます。 一度確保されたメモリは解放できない仕様ですが、それにもかかわらず多くの組込みアプリケーションに適しています。その理由は、多くの組込み用途では、システム起動時に必要なタスク、キュー、セマフォなどをすべて生成し、プログラムの終了までそれらを使い続け、削除することがないためです。
スキーム 2	ベストフィットアルゴリズムを使用します。スキーム 1 とは異なり、以前に確保したメモリブロックを解放することが可能です。ただし、隣接する空きブロックを 1 つの大きなブロックに統合することはありません。動的にタスクを生成する必要がある、小規模なリアルタイムシステムに適しています。
スキーム 3	標準 C ライブラリの malloc および free 関数に対するシンプルなラッパーを実装したものです。 ラッパーは、malloc および free 関数をスレッドセーフにするだけの役割を果たします。
スキーム 4	先着適合アルゴリズムを使用しており、スキーム 2 とは異なり、隣接する空きメモリブロックを 1 つの大きなブロックに統合することで、ヒープの断片化を防ぎます。
スキーム 5	スキーム 4 と同じ先着適合アルゴリズムとメモリの併合アルゴリズムを使用しており、隣接していない複数のメモリ領域（非連続メモリ領域）にまたがって動作することが可能です。

表 3. メモリ管理方式

低消費電力モード

OPENRTOS® を実行しているマイコンの消費電力を削減する一般的な方法として、アイドルタスクフックを使用してマイコンを低消費電力状態へ移行させる手法があります。しかし、このシンプルな方法で実現できる省電力効果には限界があります。定期的にティック割り込みを処理するために低電力状態を抜けて再び入る必要があるからです。

さらに、ティック割り込みの頻度が高すぎると、毎回の出入りで消費されるエネルギーと時間が、むしろ無駄となり、省電力効果を打ち消してしまう可能性があります。

このような状況を改善するため、OPENRTOS® では「ティックレスアイドルモード」が用意されています。このモードでは、実行可能なアプリケーションタスクが存在しないアイドル期間中に、定期的なティック割り込みを停止し、動作再開時にカーネルのティックカウントを補正することで、省電力化を図ります。

ティック割り込みを停止することで、マイコンは深い低消費電力状態を維持したまま、割り込みが発生するか、カーネルがタスクを「実行可能状態」に移行させる必要がある時点まで、スリープを継続できます。

タスク間通信

OPENRTOS® は、タスク間でデータを安全に転送できるキュー機能を提供しています。このキューの実装は柔軟性が高く、単純なデータ転送だけでなく、タスク間の同期やセマフォ的な動作など、さまざまな目的に利用することができます。

キューの特性

以下は、キュー実装の要点をまとめたものです。

- 各アイテムのサイズと、最大アイテム数は、キュー作成時に設定されます。
- アイテムは Queue Send API を使用してキューに送信されます。
- アイテムは Queue Receive API を使用してキューから受信されます。
- キューはスレッドセーフなバッファであり、通常は新しいデータがキューの末尾に追加されますが、高優先度のデータは先頭に送信することも可能です。

キューイベント

キューに送受信されるデータは「キューイベント」と呼ばれます。
タスクが Queue Send API を呼び出す際、すでにキューが満杯である場合に、空きができるまでどのくらいの間ブロック状態で待機するかを指定することができます。
この場合、タスクは「キューイベント待ち」でブロック状態に入り、他のタスクや割り込みがキューからアイテムを取り除くと自動的に「ブロック状態」を抜けます。

同様に、タスクが Queue Receive API を呼び出す際、すでにキューが空である場合には、データが到着するまでどのくらいの間ブロック状態で待機するかを指定できます。この場合もタスクは「キューイベント待ち」でブロックされ、他のタスクや割り込みがキューにデータを書き込むと、自動的にブロック状態から復帰します。

株式会社ディオイズジャパン
https://www.dioiz-japan.com
dioiz@dioiz.com

もし複数のタスクが同じイベントを待ってブロック状態にある場合、

- 優先度が最も高いタスクが最初にアンブロックされます。
- 同じ優先度のタスクが複数ある場合は、最も長くブロック状態にあったタスクがアンブロックされます。

バイナリセマフォ

セマフォは通常、タスク間、またはタスクと割り込み間の同期に使用されます。

たとえば、あるタスクが周辺機器（ペリフェラル）を制御するケースを考えてみましょう。周辺機器をポーリングする方式では CPU 資源を浪費し、他のタスクの実行も妨げてしまいます。そのため、タスクはほとんどの時間をブロック状態で待機し、実際に処理すべきことがあるときだけ動作するのが望ましいと言えます。

これを実現するのが バイナリセマフォ です。タスクはセマフォを「取得」しようとしてブロック状態に入り、周辺機器の割り込みサービスルーチンはサービスが必要なタイミングでそのセマフォを「解放」します。

OPENRTOS® では、バイナリセマフォを完全にサポートする API 関数が用意されています。コードサイズを最小限に抑えるために、セマフォの実装にはキュープリミティブ が使用されています。

バイナリセマフォは、最大で 1 つのアイテムしか保持できないキューと考えることができます。効率のため、アイテムサイズは 0 に設定でき、データのコピーは行われません。重要なのは、キューが空か満かという状態であり、データの値そのものではありません。

- リソースが利用可能なとき：キュー（セマフォ）は「満」に
- タスクがセマフォを取得するとき：キューから受信し、「空」に
- セマフォを解放するとき：キューに送信し、「満」に戻る

タスクがキューから受信しようとして、それがすでに空である場合、そのリソースは利用できないことを意味し、タスクは再びリソースが利用可能になるまでブロック状態で待つかどうかを選択できます。

カウントセマフォ

カウントセマフォは、バイナリセマフォと同様の方法で実装されています。OPENRTOS® では、カウントセマフォの API の内部で キュープリミティブ を使用することで、効率的な設計を実現しています。

カウントセマフォには、キューの深さ（最大カウント値）に上限があります。キュー内にアイテムが存在する限り、リソースは利用可能です。バイナリセマフォと同様に、カウントがゼロに達した場合はリソースが利用できない状態となり、タスクは再びリソースが利用可能になるのを待つために、ブロック状態へ入るかどうかを選択することができます。

ミューテックス

ミューテックスは、優先度継承機構を備えたバイナリセマフォの一種です。バイナリセマフォはタスク間やタスクと割り込み間の同期を実現するのに適していますが、ミューテックスはシンプルな排他制御を実現するのに適しています。

排他制御に使用される場合、ミューテックスはリソースを保護するためのトークンのように動作します。タスクがリソースへアクセスしたい場合、まずそのトークンを「取得」する必要があります。使用が完了したら、他のタスクがそのリソースにアクセスできるように、必ず「解放」しなければなりません。

ミューテックスでは、ブロック時間を指定することもできます。ブロック時間とは、ミューテックスがすぐに利用できない場合に、タスクがミューテックスを「取得」するためにブロック状態に入る最大ティック数のことです。

ただし、バイナリセマフォと異なり、ミューテックスは優先度継承を行います。これは、低優先度のタスクがミューテックスを保持している間に高優先度のタスクがそのミューテックスを取得しようとしてブロックされた場合、一時的にミューテックスを保持しているタスクの優先度が、ブロックされている高優先度のタスクと同じレベルまで引き上げられるという仕組みです。この仕組みにより、高優先度タスクがブロック状態で待機する時間を最小限に抑え、すでに発生した優先度逆転の影響を軽減することができます。

なお、優先度継承は優先度逆転を完全に防ぐものではありません。
あくまで影響を軽減する仕組みであり、リアルタイム性が求められるアプリケーションでは、そもそも優先度逆転が発生しないように設計することが重要です。

再帰ミューテックス

再帰的に使用されるミューテックスは、所有者が繰り返し「取得」することが可能です。ミューテックスは、所有者が xSemaphoreTakeRecursive API を呼び出した回数と同じ回数だけ xSemaphoreGiveRecursive API を呼び出すまで、他のタスクには解放されません。たとえば、あるタスクが同じミューテックスを 5 回「取得」した場合、そのタスクが 5 回「解放」するまでは、他のタスクがそのミューテックスを取得することはできません。

この種類のセマフォは優先度継承機構を使用しており、セマフォを取得したタスクは、使用が終了したら必ずセマフォを返却する必要があります。

なお、ミューテックスタイプのセマフォは割り込みサービスルーチンでは使用できません。

株式会社ディオイズジャパン
https://www.dioiz-japan.com
dioiz@dioiz.com

タスクと割り込み間の通信

キューやセマフォは、割り込みサービスルーチンからも使用できます。

イベントフラグ

イベントフラグは、タスクにイベントの発生を通知するために使用されます。タスクは、単一のイベントまたは同じイベントグループ内の複数のイベントの組み合わせによって起床されることがあります。イベントグループは、一連のイベントフラグによって構成されます。

イベントグループに対しては、タスクがイベントフラグを 1 つ以上設定またはクリアしたり、イベントグループ内で 1 つ以上のイベントフラグが設定されるのを待つためにペンド（ブロック状態に入り、CPU 時間を消費しないようにする）する API が提供されています。

イベントグループは、タスク間の同期にも使用でき、これは一般的に「タスクのランデブー」と呼ばれることがあります。タスクの同期ポイントとは、アプリケーションコード内で、他のすべての同期対象タスクが同じ同期ポイントに到達するまで、タスクがブロック状態（CPU 時間を消費しない状態）で待機する場所を指します。

タスク通知

タスク通知機能は、キュー、セマフォ、イベントグループの軽量な代替手段を提供します。タスク通知は、提供される情報を 1 つのタスクのみが消費する場合に適しており、パフォーマンスおよび RAM 使用量の面で大きな利点があります。

各タスクは 32 ビットの通知値を持ちます。タスク通知とは、特定のタスクに直接送信されるイベントであり、受信タスクをアンブロックさせることができ、オプションで通知値を更新することも可能です。通知値は以下の方法で更新できます。

- 既存の値を上書きせずに通知値を設定する
- 通知値を上書きする
- 通知値の 1 つ以上のビットを設定する
- 通知値をインクリメントする

このような柔軟性により、別途キュー、バイナリセマフォ、カウントセマフォ、イベントグループを作成する必要があった用途にも、タスク通知を活用できます。タスクを直接通知でアンブロックする場合、バイナリセマフォを使用する場合に比べて最大で 45% 高速で、かつ少ない RAM で済みます。

通知は xTaskNotify および xTaskNotifyGive API（およびそれらの割り込み安全版）を使って送信され、受信タスクが xTaskNotifyWait または ulTaskNotifyTake API を呼び出すまで保留状態のままとなります。もし受信タスクが通知待ちで既にブロック状態にあった場合、通知を受信した時点でブロック状態から復帰し、通知はクリアされます。

ソフトウェアタイマ

ソフトウェアタイマーは、将来の特定の時点でコールバック関数を実行する機能を提供します。

タイマーには「ワンショットタイマー」と「オートリロードタイマー」の 2 種類があります。ワンショットタイマーは一度だけコールバック関数を実行し、手動で再起動しない限り自動的には再実行されません。これに対して、オートリロードタイマーは一度起動されると、コールバック関数を実行するたびに自動的に再起動され、周期的な実行が行われます。

OPENRTOS® のソフトウェアタイマーは非常に効率的に実装されています。コールバック関数は割り込みコンテキストでは実行されず、タイマーが実際に満了しない限り CPU 時間を消費しません。また、タイマー機能はシステムティック割り込みに余分な負荷をかけず、割り込みを無効にした状態でリンクリスト構造を走査することもありません。

タイマー機能は必須機能ではなく、OPENRTOS® カーネルのコアには含まれていません。代わりに、タイマーサービスまたはデーモンタスクによって提供されます。このタスクは、タイマー処理以外の目的でも使用することができます。

スタックオーバーフロー保護

スタックオーバーフローは、アプリケーションの不安定性を引き起こす非常に一般的な原因です。OPENRTOS® では、そのような問題の検出および修正を支援するために、2 つのオプション機能が用意されています。

スタックが最大値に達するのは、カーネルがタスクを実行状態から切り替えた直後である可能性が高く、このときスタックにはタスクのコンテキスト情報が格納されます。このタイミングで、RTOS カーネルはプロセッサのスタックポインタが有効なスタック領域内に収まっているかを確認できます。スタックポインタの値が有効範囲外であった場合、スタックオーバーフローフック関数が呼び出されます。

もう一つの方法は、タスクスタック全体を既知の値で初期化しておくことです。タスクが実行状態から切り替えられる際に、RTOS カーネルは有効なスタック領域の最後の 16 バイトが初期値のままであるかを確認することで、タスクや割り込みによる書き換えが発生していないかを検出します。これらの 16 バイトのいずれかに初期値が保持されていない場合は、スタックオーバーフローフック関数が呼び出されます。

株式会社ディオイズジャパン
https://www.dioiz-japan.com
dioiz@dioiz.com

スレッドローカルストレージ

スレッドローカルストレージは、アプリケーション開発者が各タスクのコントロールブロック内に値を格納できるようにする機能で、その値はタスク固有となり、各タスクがそれぞれ独自の値を持つことができます。

スレッドローカルストレージは、通常、シングルスレッドのプログラムであればグローバル変数として扱われるような値を格納するために使用されます。たとえば、多くのライブラリには errno と呼ばれるグローバル変数が含まれています。ライブラリ関数がエラー状態を返した場合、呼び出し元は errno の値を調べてエラーの内容を判断できます。シングルスレッドのアプリケーションでは errno をグローバル変数として定義すれば十分ですが、マルチスレッドのアプリケーションでは、各スレッド（タスク）がそれぞれ固有の errno 値を持つ必要があります。そうでなければ、あるタスクが別のタスク用の errno 値を読み取ってしまう可能性があるためです。

実行時間統計

OPENRTOS® では、各タスクが使用した処理時間に関する情報をオプションで収集することができます。

各タスクには以下の 2 つの値が示されます。

- Abs Time（絶対時間）：タスクが実際に実行されていた総時間（実行状態にあった総時間）を表します。どのような時間基準を用いるかは、アプリケーションに応じてユーザーが選択します。
- % Time（パーセンテージ時間）：絶対時間と同様の情報を、全体の処理時間に対する割合（%）として示したものです。

フック関数

フック関数を使用することで、カーネル自体を変更することなく、特定の関数やイベントに対してアプリケーションコードを関連付けることができます。

OPENRTOS® では、ユーザー定義のオプション機能として、「表 4. フック関数」に示す 4 つのフック関数をサポートしています。

フック関数	説明
スタック オーバーフロー フック関数	スタックオーバーフローが検出されると、このフック関数が呼び出されます。 ホストアプリケーションはシステムを安全な状態へ移行させるための、アプリケーション固有のエラーハンドリングを実装できます。
メモリ確保失敗 フック関数	pvPortMalloc が NULL を返した場合に呼び出されます。 ヒープメモリ不足に起因する問題、特にアプリケーション内で pvPortMalloc の呼び出しが失敗した場合の原因を特定しやすくなります。
アイドル フック関数	アイドルタスクのコンテキスト内でアプリケーション固有の処理を実行できるようにします。 低優先度のバックグラウンド処理の実行や、プロセッサを低電力スリープモードへ移行させるといった用途で使用されるのが一般的です。
ティック フック関数	ティックハンドラが実行されるたびに呼び出されます。 アプリケーション固有の処理を周期的に実行するために使用されます。
デーモンタスク スタートアップ フック関数	デーモンタスクが最初に実行を開始するタイミングで呼び出されます。 スケジューラの開始後にリアルタイム OS の機能を使用する初期化処理を行う場合に有用です。

表 4. フック関数

株式会社ディオイズジャパン
https://www.dioiz-japan.com
dioiz@dioiz.com

IDE 対応

OPENRTOS® は、IAR、Keil、Rowley、CodeWarrior、GCC、Tasking、Atollic、Code Composer Studio をはじめとする主要な統合開発環境でご利用いただけます。

OPENRTOS® の構成

OPENRTOS® は、開発者が選択したプロセッサおよびツールチェーンに応じてライセンスされます。

OPENRTOS® の API およびコアコードは、すべてのバリアントで共通です。ポートレイヤのみが選択されたプロセッサおよびコンパイラに合わせて適応されています。

対応デバイス一覧

Manufacturer	Device	Family
Altera	Nios II	Nios II
Atmel	AVR32	AVR32
	SAM3	ARM Cortex-M3
	SAM4	ARM Cortex-M4
	SAMV7	ARM Cortex-M7
	SAM7	ARM7
	SAM9	ARM9
Cortus	APS3	APS3
Cypress	PSoc5	ARM Cortex-M3
Fujitsu	FM3	ARM Cortex-M3
Infineon	Tricore	Tricore
	AURIX	Tricore
	XMC4000	ARM Cortex-M4
Microchip	PIC32	PIC32
	PIC24	PIC24
	dsPIC	dsPIC
Microsemi	SmartFusion	ARM Cortex-M3
	SmartFusion2	ARM Cortex-M3
NXP	LPC4000 LPC4300	ARM Cortex-M4
	LPC1300 LPC1700 LPC1800	ARM Cortex-M3
	LPC1100, 1200	ARM Cortex-M0
	Kinetis K series	ARM Cortex-M4
	Kinetis L series	ARM Cortex-M0
	Coldfire V2	Coldfire V2
	Coldfire V1	Coldfire V1
	HCS12	HCS12
	i.MX6x	ARM Cortex-A9
Renesas	RH850	RH850
	RX600	RX600
	RX200	RX200
	SuperH	SuperH
	RL78	RL78
	H8/S	H8/S
	RZ	ARM Cortex-A9
Silicon Labs	Gecko Range	ARM Cortex-M0+ ARM Cortex-M3 ARM Cortex-M4
ST Microelectronics	STM32F7	ARM Cortex-M7
	STM32F4 STM32F3 STM32L4	ARM Cortex-M4
	STM32F2 STM32F1 STM32L1 STM32W	ARM Cortex-M3
	STM32F0	ARM Cortex-M0
	STR7	ARM7
	STR9	ARM9
Synopsys	ARC	ARC600
Texas Instruments	Stellaris LM3	ARM Cortex-M3
	Tiva	ARM Cortex-M4
	TMS320 Delfino	C28x
	Hercules RM4x	ARM Cortex-R4
	Hercules TMS470	ARM Cortex-M3
	Hercules TMS570	ARM Cortex-R5
	Concerto	ARM Cortex-M3
	MSP430X	MSP430X
	MSP432	ARM Cortex-M4F
Xilinx	Microblaze	Microblaze
	PowerPC	PowerPC
	Zynq-7000	ARM Cortex-A9
	UltraScale MPSoC	ARM Cortex-R5 ARM Cortex-A53

株式会社ディオイズジャパン
https://www.dioiz-japan.com
dioiz@dioiz.com