2004－OS－96　(14)
2004／6／18

センサノード用アプリケーション開発環境の設計と実装

須之内 雄司[1] 中西 健一[2] 高汐 一紀[2] 徳田 英幸[1 2]
[1] 慶應義塾大学 環境情報学部 [2] 慶應義塾大学大学院 政策メディア研究科

ユビキタスコンピューティング環境ではセンサやアクチュエータを持ち，無線でネットワークにつながるセンサノードが我々の生活環境に偏在するようになる．センサノード用アプリケーションの多くは，ポーリングや割り込みやタスク・イベントなどのモデルに従って開発されているが，これらのモデルでアプリケーションを開発するには，開発者にハードウェアと密接した知識が要求される．本論文では，小型のセンサノードで動作する軽量なブロッキング I/O 及びプリエンプティブマルチスレッドの設計と実装を行う．本システムの利用により，ハードウェアに関する知識を有していない開発者でもセンサノード上のアプリケーションを開発できるようになる．

Design and Implementation of Application Development Environment for Sensor Nodes

Yuji SUNOUCHI[1] Ken NAKANISHI[2] Kazunori TAKASHIO[2] Hideyuki TOKUDA[1 2]

[1] Faculty of Environmental Information, Keio University
[2] Graduate School of Media and Governance, Keio University

In ubiquitous computing environments, sensor nodes will pervasively exist in our lives. Many applications for sensor nodes are based on models such as polling, interrupts, and task and event model. Application developers who utilize these methods are required to be familiar with hardware architecture. In this paper, we propose a light-weight blocking I/O and preemptive multithread library for sensor nodes' application. Using this library, application developers, who aren't familiar with hardware architecture, can develop applications.

1　はじめに

ユビキタスコンピューティング環境ではセンサやアクチュエータを持ち，無線でネットワークにつながるノードが我々の生活環境に偏在するようになる．ここ数年で Mica Mote[1] や Smart-Its[2] や U^3[3] などの小型センサノードが研究用に開発されてきた．アプリケーション開発者はセンサノードにアプリケーションを書き込むことで，センサノードをネットワークで連携させ，利用者の好みや周囲の状況に応じたサービスを提供できる．

センサノードの開発により，ユビキタスコンピューティング環境を実現するためのハードウェアプラットフォームは揃ってきたが，ソフトウェアプラットフォームはまだ発展途上である．センサノードは，メモリサイズやプログラムサイズや電力などのリソースが限られているため，開発者はハードウェアに密接したプログラミングモデルでアプリケーションを開発している．そのため，開発者がセンサノード用のアプリケーションを開発するには，割り込みやスケジューリングや電力消費といったハードウェアに関する深い知識が必要となる．開発者がより効率的にアプリケーション開発を行うには，ハードウェアに関する深い知識無しで開発できるプログラミングモデルが必要である．

本稿は開発者がセンサノード用アプリケーションを容易に開発できる軽量なライブラリを提案する．本ライブラリはブロッキング I/O とプリエンプティブマルチスレッドの利用により，ハードウェアに関する深い知識を持たない開発者でも容易にアプリケーション開発できる環境を実現する．

以降，2 章で既存プログラミングモデルの考察を行い，3 章で問題に対する解決のアプローチを説明する．4 章では本ライブラリの設計を説明し，5 章で実装と評価について述べる．6 章で関連研究を挙げ，本ライブラリとの性質面での比較を行う．7 章で本研究をまとめ，今後の研究課題について述べる．

2　既存のプログラミングモデルの考察

本稿では，センサノードを，MICA2 Mote[4] のような 4MHz の処理能力の CPU と 4KB のメモリと 128KB のプログラム領域を持ち，電池又は電源で駆動するハードウェアであると想定する．

このようなセンサノードで動作する，入出力を行うアプリケーションには，ポーリングや割り込みやイベントとタスクなどのモデルがある．以下でそれぞれのモデルの特徴と問題点を述べる．

2.1　ポーリングによる入出力

ポーリングは最も単純なモデルで，アプリケーションはデバイスが入出力可能かをループで監視し，入出力可能な時点で入出力を行う．この手法は Smart-Its のソフトウェアライブラリなどで用いられている．ポーリングによるデバイスの入出力シーケンスを図 1 に示す．このモデルでは，デバイスが入出力できない間もスリープせずに監視し続けるため，電力消費が大きくなってしまう．よって，ポーリングは電池で駆動するセンサノードでの利用には向いていない．

2.2　割り込みによる入出力

デバイスからの割り込みを利用するアプリケーションは，割り込みハンドラで処理を行う．割り込みによるデバイスからの入出力シーケンスを図 2 に示す．アプリケーションは割り込みが発生するまで

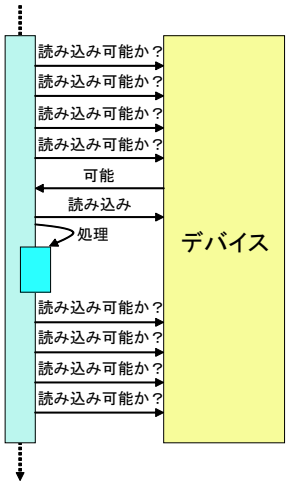

図 1: ポーリングによる入出力

CPU をスリープさせられるので，電力消費を必要最小限に抑えられる．このモデルでは，割り込みハンドラが長時間割り込みを禁止すると，他の割り込みを受けつけられない．割り込みハンドラ内での割り込みを許可すると，再帰的な割り込みの発生により，スタックオーバーフローする危険性がある．よって，開発者は割り込みハンドラの処理にかかる時間を考慮して開発する必要がある．

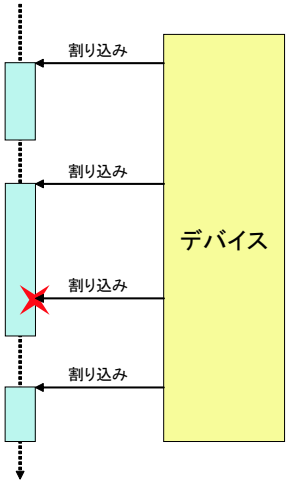

図 2: 割り込みによる入出力

2.3 タスクとイベントによる入出力

タスクとイベントのモデルでは，アプリケーションのメインスレッドが，キューに入っているタスクを FIFO で処理する．タスクはイベントや他のタスクにより追加される．このモデルは TinyOS[5] や U^3 のソフトウェアライブラリ [6] などで採用されている．イベントとタスクの構成を図 3 に示す．イベントとはハードウェア割り込みなどにより発生する処理で，メインスレッドに割り込める．イベントは割り込みを長時間禁止しないよう，即座に制御を返す必要がある．そのため，イベントでは時間のかかる処理は行わず，代わりに処理用のタスクをキューに追加する．キューに追加されたタスクは，メインスレッドにより順次実行されていく．タスクがキューに入っていない場合，メインスレッドは CPU をスリープさせ，電力消費を抑える．イベントとタスクによる入出力シーケンスを図 4 に表す．このモデルでは，キューに加えられたタスクは FIFO で処理されていくため，時間のかかるタスクがすでにキューにあると，その後の時間のかからないタスクが待たされる．また，タスクの処理よりタスクの追加が多いと，キューが一杯になってしまう可能性がある．これらの問題を解決するには，処理に時間がかかるタスクを分割し，個々のタスクが適度なタイミングで完了する必要がある．その概要を図 5 に表す．

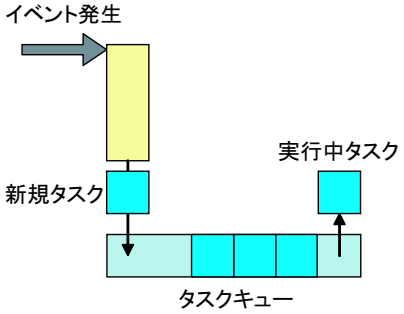

図 3: イベントとタスク

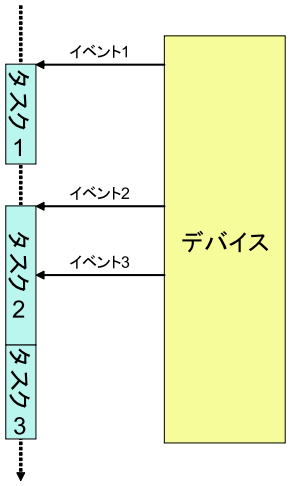

図 4: イベントとタスクによる入出力

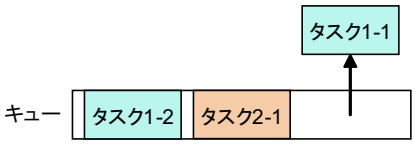

図 5: 分割されたタスク

2.4 既存のプログラミングモデルの問題点

前節までに挙げたプログラミングモデルでアプリケーションを開発するには，ハードウェアに関する知識が開発者に要求される．開発の際に注意が必要となる点を以下にまとめる．

電力の利用効率

電池で駆動するセンサノードにとって，電力は重要なリソースである．処理するデータがない時は，アプリケーションは CPU に無駄な処理をさせてはならない．

割り込み

センサや通信の入出力に対する応答が長時間遅れてはならない．割り込みを長時間禁止しないように，割り込みハンドラで時間がかかる処理をしてはならない．

スケジューリング

時間のかからない処理はすぐ終わらなくてはならない．そのために，時間のかかる処理は分割し，時間のかからない処理の妨げになってはならない．

3 目的とアプローチ

本研究の目的は，ハードウェアに関する知識のない開発者が，容易にセンサノード用アプリケーションを開発できる環境の構築である．割り込みやスケジューリングや電力消費は開発者にとって複雑であり，隠蔽しなくてはならない．

本稿では，これらの知識無しでアプリケーションが開発できる C 言語用ライブラリを提案する．本ライブラリはブロッキング I/O とプリエンプティブマルチスレッドを利用する．ライブラリが電力の利用効率や割り込みやスケジューリングに対する考慮を担うことで，開発者はアプリケーションの機能の開発に専念できる．

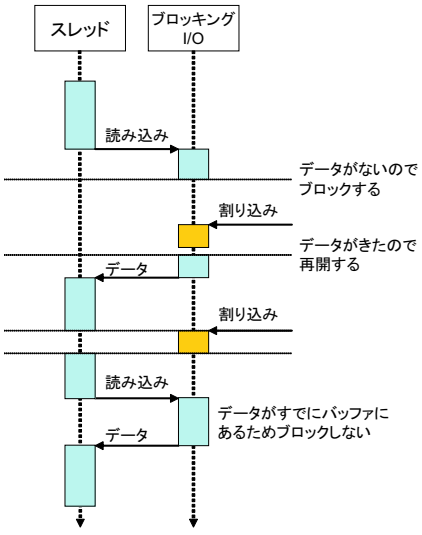

図 6: ブロッキング I/O による待ちうけ

ブロッキング I/O

本ライブラリはブロッキング I/O により，電力の節約と短い割り込み禁止期間を実現する．ブロッキング I/O では，入出力ができない時は，ポーリングせずに割り込みを待つ．その間は CPU をスリープさせられるので，電力の利用効率が向上する．また，割り込みハンドラでは入出力以外の処理を行わないため，割り込みが長時間禁止されることはない．ブロッキング I/O を利用した入出力のシーケンスを図 6 に示す．

プリエンプティブマルチスレッド

本ライブラリはプリエンプティブマルチスレッドにより，プログラムの並行性を実現する．前章で挙げたモデルでは，時間がかかる処理を行うと，割り込み禁止の期間が長くなってしまう問題や，他の時間がかからない処理が待たされる問題があった．本ライブラリでは，スレッドを決められた時間ごとにプリエンプトし，別のスレッドに実行権限を与える．図 7 にそのシーケンスを示す．時間のかかる処理は一定期間ごとにプリエンプトされるため，時間のかからない処理が時間のかかる処理によって妨げられない．

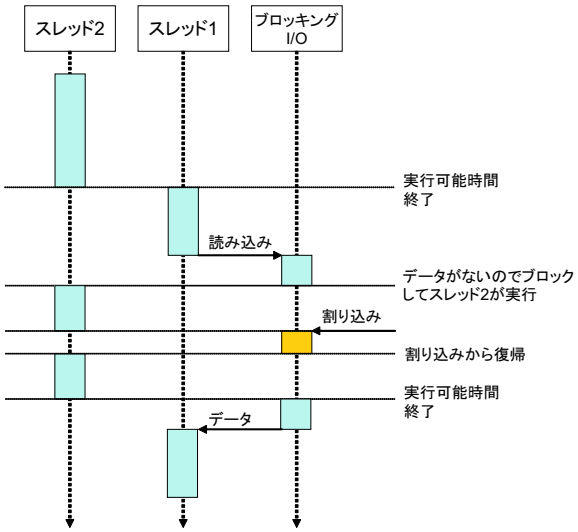

図 7: スレッドによる並行処理

4 設計

本章では本ライブラリの設計について述べる．本ライブラリは図 8 に示すような 5 つのモジュールから構成される．アプリケーションはスレッドモジュール，ミューテクスモジュール，ブロッキング I/O モジュールを通じて本ライブラリを利用する．

4.1 スレッド

スレッドはプログラムの実行単位であり，それぞれが独自のレジスタの値とスタックを持つ．本ライブラリでは，処理を行うスレッドを切り替えることで並行性を実現している．実行するスレッドは，ミューテクスやブロッキング I/O によって切り替えられる他，一定時間ごとにスケジューラによって切り替えられる．スレッドモジュールは，スレッド固有の情報を保存するためのスレッド情報構造体を持ち，スレッド生成関数とスレッド停止関数とスレッド再開関数を提供する．

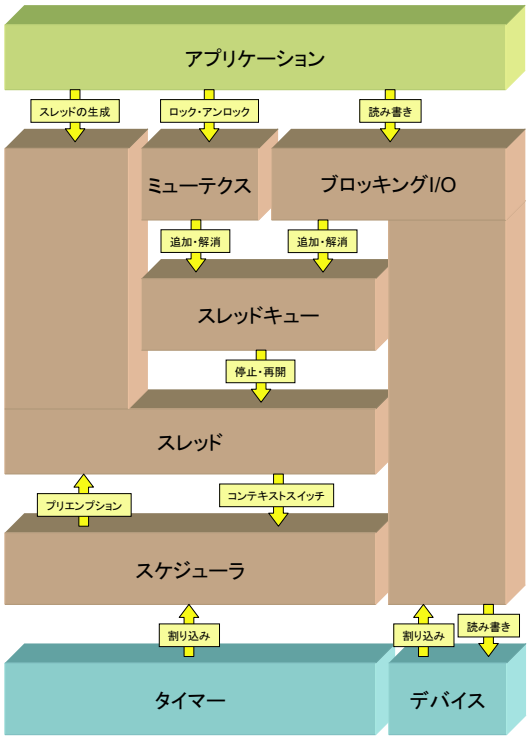

図 8: モジュールの構成

スレッド生成関数

スレッド生成関数はアプリケーションからスレッドモジュールにアクセスできる唯一のインターフェースである．スレッド生成関数のプロトタイプ宣言を図 9 に示す．スレッド生成関数の引数にはエントリポイントとなる関数のアドレス，その関数に対する引数，そしてスレッドに割り当てるスタックのアドレスを渡す．新しく作成するスレッドのスタック領域をアプリケーションが指定することで，最低限のメモリ領域を割り当てることができる．本ライブラリはメモリサイズの節約のため，スレッドの数をコンパイル時に決定し，実行時はそれ以上のスレッドは作成できない．スレッド生成関数はスレッド情報構造体を初期化し，実行中スレッドのリストに加える．実行中スレッドのリストはスレッド情報構造体の循環リストで管理される．

```
/* スレッド生成関数 */
char createThread(
  /* エントリポイント */
  void (*entrypoint)(char),
  /* エントリポイントへの引数 */
  char entrypointarg,
  /* スレッドのスタック領域 */
  char* stack);
```

図 9: スレッド生成関数

スレッド停止関数・スレッド再開関数

スレッドには実行可能状態と停止状態の二つの状態がある．他のモジュールはスレッド停止関数とスレッド再開関数を通じてスレッドの状態を変更する．スレッド停止関数は現在実行中のスレッドを実行可能状態から停止状態に変更し，スケジューラモジュールのスレッド切り替え関数を呼び出す．スレッド再開関数は指定されたスレッドを停止状態から実行可能状態に変更する．実行可能状態になったスレッドは，後にスレッド切り替えにより実行される．

4.2 スケジューラ

スケジューラモジュールは実行するスレッドの管理を行う．スケジューラはスレッド切り替え関数を提供する．スレッド切り替え関数はスレッドモジュールのスレッド停止関数とタイマ割り込みから呼び出される．スレッド切り替え関数はスレッド固有情報の保存，実行スレッドの選択，スレッド固有情報のロードの 3 段階で構成される．スレッド切り替えにはレジスタを直接扱う必要があるため，スレッド切り替え関数は一部アセンブリで記述しなくてはならない．

スレッド固有情報の保存

最初に，スレッド切り替え関数は実行していたスレッド固有の情報を保存する．スレッド固有の情報は，汎用レジスタ，スタックレジスタ，ステータスレジスタ，プログラムカウンタである．スレッド切り替え関数は，汎用レジスタとスタックレジスタの値をスレッド情報構造体に保存し，ステータスレジスタの値をスタックに push する．プログラムカウンタはスレッド切り替え関数の呼び出し時にスタック上に push される．

実行スレッドの選択

次に，スレッド切り替え関数は実行するスレッドを選択する．センサノードの限られたメモリサイズとプログラムサイズで動作させるため，スケジューリングアルゴリズムはシンプルなラウンドロビンを採用した．スレッド切り替え関数はスレッド情報構造体の循環リストを辿り，実行可能状態のスレッドを探す．全てのスレッドが停止状態の場合，CPU をスリープさせ，割り込みによってどれかのスレッドが実行可能になるまで待つ．

スレッド固有情報のロード

最後に，スレッド切り替え関数は次に実行するスレッド固有の情報を CPU にロードする．スケジューラはスレッド情報構造体から，汎用レジスタとスタックポインタの値を CPU にロードし，ステータスレジスタをスタックから pop する．プログラムカウンタの値はスタック上にあり，スレッド切り替え関数から抜ける際に自動的にロードされる．

4.3 スレッドキュー

スレッドキューモジュールはスレッドの停止と再開を FIFO で行う．ミューテクスやブロッキング I/O はスレッドキューを利用して，リソースへのアクセスを FIFO で行う．スレッドキューには，ブ

ロックしたスレッドを保持するスレッドリストがあり，キュー初期化関数とキュー追加関数とキュー解消関数により操作される．構成を図 10 に示す．

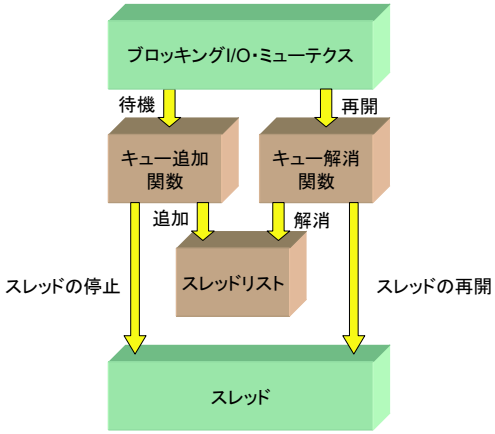

図 10: スレッドキューモジュールの構成

キュー初期化関数
キュー初期化関数はスレッドリストを初期化する．キュー追加関数とキュー解消関数を呼ぶ前にこの関数を呼ばなくてはならない．

キュー追加関数
キュー追加関数は実行中のスレッドをキューで待たせる．実行中のスレッドをスレッドリストの末尾に追加し，スレッド停止関数を呼び出し，スレッドをブロックする．

キュー解消関数
キュー解消関数はキューで待っているスレッドを再開させる．スレッドリストの先頭にあるスレッドを取り除き，スレッド再開関数を呼び出し，スレッドを再開させる．

4.4 ミューテクス

ミューテクスはスレッドの排他処理を行う．開発者はアプリケーション内のクリティカルセクションをミューテクスで守り，スレッドセーフなアプリケーションを開発できる．アプリケーションは割り込みを禁止することでアトミックな処理を行えるが，クリティカルセクション内の処理が長いと割り込み禁止の期間も長くなってしまう．ミューテクスは短い割り込み禁止期間でスレッド間の排他処理を実現する．ミューテクスはロック状態を保持するロックフラグとスレッドキューを持ち，アプリケーションはミューテクス初期化関数とミューテクスロック関数とミューテクスアンロック関数を通じてミューテクスを操作する．ミューテクスモジュールの構成を図 11 に示す．

ミューテクス初期化関数
ミューテクス初期化関数はミューテクスを初期化する．スレッドキューを初期化し，ロックフラグをロックされた状態かアンロックされた状態に初期化する．アプリケーションはミューテクスロック関数とミューテクスアンロック関数を呼ぶ前にこの関数でミューテクスを初期化しなくてはならない．

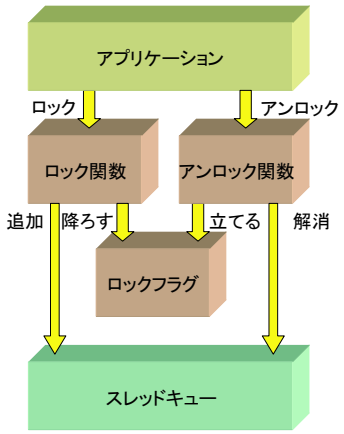

図 11: ミューテクスモジュールの構成

ミューテクスロック関数
ミューテクスロック関数は割り込みを禁止し，ロックフラグをチェックする．ロックされていない場合はロックフラグを立て，割り込みを許可し，関数から抜ける．すでにロックされている場合はスレッドキューのキュー追加関数を呼び出し，スレッドをブロックさせる．ブロックしたスレッドはミューテクスアンロック関数により再開され，ロックフラグを立て，関数を抜ける．

ミューテクスアンロック関数
ミューテクスアンロック関数はロックフラグを降ろす．スレッドキューにスレッドが入っている場合はキュー解消関数でブロックしているスレッドを再開させる．

4.5 ブロッキング I/O

ブロッキング I/O モジュールはデバイスに対する入出力の機構を提供する．本稿では，USART のブロッキング I/O を例に説明をする．USART のブロッキング I/O モジュールには入力用モジュールと出力用モジュールがある．

4.5.1 入力用モジュール

入力用のブロッキング I/O モジュールは，リングバッファとスレッドキューを持ち，入力初期化関数とデータ取得関数とデータ保存関数を通じて操作される．構成を図 12 に示す．

入力初期化関数
入力初期化関数はリングバッファとスレッドキューを初期化し，USART の受信完了割り込みを許可する．アプリケーションはデータ取得関数を呼び出す前にこの関数を呼び出さなくてはならない．

データ取得関数
アプリケーションはデータ取得関数を通じてシリアルのデータを取得する．データ取得関数はデータ保存関数によってリングバッファに保存されたデータを読み取る．リングバッファにデータがない場合，スレッドキューのキュー追加関数を呼び出し，スレッドをブロックさせる．ブロックしたスレッドはデータ保存関数によって再開される．

データ保存関数

データ保存関数は USART のデータ受信完了割り込みで呼び出され，シリアルからのデータを読み取り，リングバッファに格納する．スレッドキューにスレッドが入っている場合，キュー解消関数でブロックしているスレッドを再開させる．

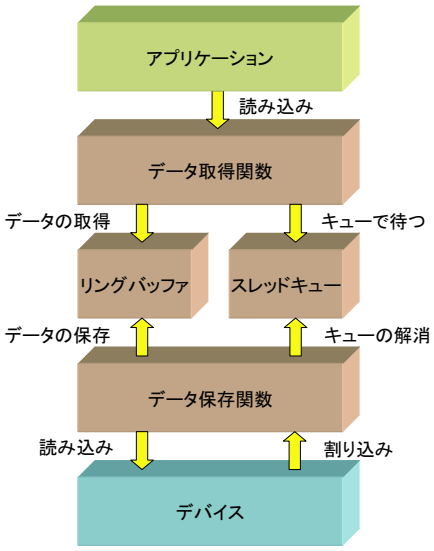

図 12: 入力用ブロッキング I/O の構成

4.5.2 出力用モジュール

出力用のブロッキング I/O モジュールはスレッドキューを持ち，出力初期化関数とデータ出力関数と出力可能通知関数を通じて操作される．構成を図 13 に示す．

出力初期化関数

出力初期化関数はスレッドキューを初期化し，USART の送信完了割り込みを許可する．アプリケーションはデータ出力関数を呼び出す前にこの関数を呼び出さなくてはならない．

データ出力関数

アプリケーションはデータ出力関数を通じてデータを書き込む．データ出力関数は USART が出力可能な場合，データをデバイスに出力する．出力可能でない場合は，スレッドキューのキュー追加関数を呼び出し，スレッドをブロックさせる．ブロックされたスレッドは出力可能通知関数により再開され，USART にデータを出力する．

出力可能通知関数

出力可能通知関数は USART からの送信完了割り込みで呼び出され，データ出力関数によってブロックされているスレッドを再開させる．スレッドキューにスレッドが入っている場合は，キュー解消関数でブロックしているスレッドを再開させる．

5 実装と評価

本稿では，本ライブラリを MICA2 Mote で使用されている Atmel 社の ATmega128L[7] 用に実装し，WinAvr(avr-gcc)[8] の 3.3.2 でコンパイルした．定

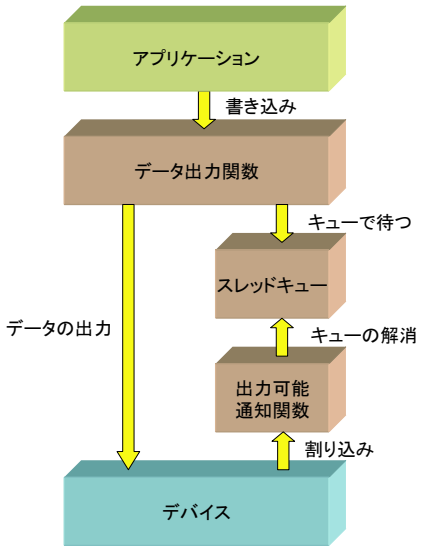

図 13: 出力用ブロッキング I/O の構成

量的評価として本ライブラリの基本性能を調べ，定性的評価としてその他のプログラミングモデルとの比較を行った．

5.1 基本性能

最適化なしでコンパイルした本ライブラリのコードサイズは約 2,700 バイトである．スレッド 2 つと USART の I/O とミューテクス 1 つを使うプログラムは 107 バイトのメモリを使用する (アプリケーションが使用する領域とスレッドのスタック領域を含まない)．各モジュールが必要なメモリサイズを表 1 に示す．スレッドの切り替えは全スレッドが実行可能であれば 260 クロックで完了する．

ATmega128L は 128KB のプログラム領域と 4KB のメモリ領域を持つため，ほとんどの領域をアプリケーションが利用できる．

モジュール	サイズ (バイト)
スレッド・スケジューラ	スレッド数 * 37 + 7
ブロッキング I/O	スレッド数 * 4 + 11
ミューテクス	スレッド数 * 2 + 3

表 1: モジュールの必要メモリサイズ

5.2 その他のプログラミングモデルとの比較

本節では，本ライブラリとその他のプログラミングモデルをスケジューリングと電力消費の面で比較する．まとめを表 2 に示す．

ポーリング

ポーリングを用いたアプリケーションでは，メインスレッドがデバイスをループで監視するため，スケジューリングを考慮する必要はない．しかし，入出力できなくても常に CPU が動作するため電力消費の面では非効率的である．本ライブラリはデータが入出力可能になるまで他のスレッドに実行権限を渡すか CPU をスリープさせるため，電力の利用効率は高い．

手法＼特徴	スケジューリング	電力消費
ポーリング	○	×
割り込み	×	○
イベントとタスク	△	○
本ライブラリ	○	○

表 2: 各手法の特徴の比較

割り込み

割り込みを用いたアプリケーションでは，割り込みが発生するまで CPU をスリープさせられるため，電力の利用効率は高い．しかし，割り込みハンドラで長時間割り込みを禁止してはならないため，スケジューリングを考慮しなくてはならない．本ライブラリでは割り込みハンドラの処理はデータの格納やブロックしているスレッドの再開だけなので，長時間割り込みを禁止することはない．

イベントとタスク

イベントとタスクを用いたアプリケーションでは，イベントでデータを読み取り，データの処理はタスクに行わせるため，割り込みハンドラの処理にかかる時間は短い．また，タスクがキューにない時は CPU がスリープするため電力消費の面でも優れている．しかし，時間のかかるタスクによって，タスクの実行の遅延やキューが一杯になる可能性がある．本ライブラリではスレッドはプリエンプトされるため，処理にどれだけ時間がかかるかを考慮せずにアプリケーションを開発できる．

6 関連研究

本節では関連研究を述べ，定性的な評価を行う．

6.1 TinyOS

TinyOS は UC Berkley で開発されたワイヤレスセンサネットワーク向けの OS である．開発者は nesC[9] という C 言語に似た独自の言語でコンポーネントを記述し，コンポーネントを組み合わせることで TinyOS 用ライブラリやアプリケーションを構築できる．TinyOS はイベントとタスクを用いたアーキテクチャで構成されている．

TinyOS では開発者が nesC を憶え，スケジューリングを考慮する必要がある．本ライブラリは GCC でリンク可能なライブラリを提供し，プリエンプティブマルチスレッドによりこれらの問題を解決している．

6.2 Smart-Its

Smart-Its は Lancaster 大学，ETH Zurich，Karlsruhe 大学，Interactive Institute，VTT Electronics で共同開発されたセンサノードである．Smart-its のソフトウェアライブラリはセンサや無線通信を操作するための関数を提供する．

Smart-Its のソフトウェアライブラリはデバイスの入出力をポーリングで行うため，電力を多く消費してしまう．本ライブラリは実行可能なスレッドがない時に CPU をスリープさせることで電力を節約している．

7 まとめと今後の課題

既存のセンサノード用のプログラミングモデルは CPU のアーキテクチャに密接しているため，開発者にはスケジューリングや割り込みや電力消費などの専門知識が要求される．この問題を解決するために本研究ではブロッキング I/O とスレッドを採用した．開発者はブロッキング I/O の利用により，電力の節約と短い割り込み禁止期間を実現し，プリエンプティブマルチスレッドの利用により，プログラムの並行性を実現した．

一方，現在のブロッキング I/O の実装はシリアルのみ対応するが，より多くのセンサ等のデバイスへの対応が望まれる．また，マルチスレッド化によるオーバーヘッドを最小限に抑えるために，スレッド切り替えにかかる時間をより短くするための工夫やメモリを節約する工夫が必要である．

参考文献

[1] : Mica Mote. http://www.xbow.com/Products/productsdetails.aspx?sid=3.

[2] Michael Beigl, Tobias Zimmer, A. K. C. D. and Robinson, P.: Smart-Its - Communication and Sensing Technology for Ubicomp Environments.

[3] 猿渡 俊介 川原 圭博 南 正輝 森川 博之 青山 友紀 篠田 庄司永原 崇範: ユビキタス環境に向けたセンサネットワークアプリケーション構築支援のための開発用モジュール U^3（U-cube）の設計と実装, 電子情報通信学会技術研究報告, IN2002-243, NS2002-270 (2003).

[4] : MICA2 Mote. http://www.xbow.com/Products/productsdetails.aspx?sid=72.

[5] Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D. E. and Pister, K. S. J.: System Architecture Directions for Networked Sensors, *Architectural Support for Programming Languages and Operating Systems*, pp. 93–104 (2000).

[6] 川原 圭博 南 正輝 森川 博之 青山 友紀鹿島 拓也: センサネットワーク開発用モジュール U^3 におけるソフトウェアデザイン及びプロトタイプアプリケーションの実装, 情報処理学会マルチメディア, 分散, 協調とモバイル（DICOMO2003）シンポジウム, pp. 305–308 (2003).

[7] : ATmega128L. http://www.atmel.com/dyn/products/product_card.asp?part_id=2019.

[8] : Winavr. http://winavr.sourceforge.net/.

[9] Gay, D., Levis, P. and von Behren, R.: The nesC Language: A Holistic Approach to Networked Embedded Systems.