5D−1

マルチスレッド プログラム構成支援ツール MTP の設計と実装*

紺田 和宣, 矢向 高弘, 安西 祐一郎†
慶應義塾大学‡

1 はじめに

物理空間を移動する自律移動ロボットの制御においては、外界の変化がロボットの内部状態に直接反映し、それに対し敏感に反応するような sensitive system が有効である。このような system には、Direct-interrupt 機能を持った自律ロボット制御用マルチスレッド OS PULSER[1] を基にした PULSER-II[2] がある。しかし、PULSER 上でのマルチスレッドプログラミングは、記述性が悪く、複雑になりやすい。本研究では、様々なスレッド間の関係について形式化し、これらの関係からマルチスレッド プログラムのアウトラインを生成するツール MTP(*M*ulti-thread program constructing support *T*ool for *P*ULSER) を作成する。最後に MTP を用いてロボットの制御プログラムを実装し、プログラムの記述性について検討する。

2 自律移動ロボットの制御プログラム

自律移動ロボットは常に環境を監視し、その時々の状態に対応した処理を行なうことが必要である。このような機能は、PULSER の Direct-interrupt(*DI*) よって実現できる。

PULSER の Direct-interrupt 機構の概要

PULSER の *DI* 機構はプロセススケジューラと密に関係しながら開発された機構であり、割り込み処理と同様な即時性を持つ。また、ユーザは自由に *DI* を発生させることができる。*DI* に対する処理をスレッドとして定義することにより、スレッドは *DI* によって始動する。

プログラミングにおける問題点

PULSER ではスレッド制御に *DI* を用いるので、*DI* とスレッドとの対応関係が繁雑になりやすい。また、スレッドは *DI* によって動的に始動するので、スレッド間の関係が複雑になる。

3 MTP の設計

設計方針

DI とスレッドの対応関係の管理を一括して行いユーザが *DI* を意識せずにスレッドの制御を行なうことを可能にする。また、スレッド間の関係を形式化しそれらを静的に実現する。以上の2点により、プログラムの記述性を高める。

関係の形式化

MTP ではスレッド間の関係として以下の7種類を扱う。

始動 スレッドが他のスレッドを始動させる関係

終了 スレッドが他のスレッドを終了させる関係

中断 スレッドが他のスレッドの処理を一時的に中断する関係

再開 スレッドが中断されている他のスレッドの処理を再開させる関係

同期 スレッドが他のスレッドと同期をとる関係

休止 スレッドが実行を開始する際に他のスレッドの処理を中断し、実行が終了した後は中断していたスレッドの実行を再開する関係

再開始 スレッドが実行を開始する際に他のスレッドの処理を終了させ、実行が終了した後は終了させたスレッドを再始動させる関係

設計

スレッドは C 言語で記述することが前提とされている。

MTP はスレッド名とスレッド間の関係を記述した relation_definition ファイル (RD ファイル) を入力として、一つのソースファイル `mtp_main.c` と各スレッド (スレッド名 *foo*) に対して`foo.h` を出力する。`mtp_main.c` にはプログラムの実行環境のための処理が記述されている。また、`foo.h` にはスレッド間関係に必要な操作がマクロで定義されている。

各スレッドは関数の形で記述する。また、スレッド間の関係は`foo.h` 中に定義されたマクロを用いて記述する。

スレッド名及びスレッド間関係の記述は以下のように行なう。

```
スレッド名 TH_NAME foo;
始動 START foo1 -> foo2 , ... ;
終了 STOP  foo1 -> foo2 , ... ;
中断 SUSP foo1 -> foo2 , ... ;
再開 RESM foo1 -> foo2 , ... ;
同期 SYNC  foo1 -> foo2 , ... ;
再始動RESTART foo1 -> foo2 , ... ;
休眠SLEEP  foo1 -> foo2 , ... ;
```

また、定義されるマクロは、以下のとおりである。

```
始動 mtp_start_foo
終了 mtp_stop_foo
中断 mtp_susp_foo
再開mtp_resm_foo
再始動 MTP_START MTP_STOP
休眠 再始動と同様
同期 mtp_wait_foo mtp_sig_foo
```

4 MTP の実装

スレッド間の関係を実現マクロ定義する。

開始 `mtp_start_foo` のマクロで対応した割り込み信号を発生させるシステムコールに展開される。

*MTP:Multi-thread program constructing support Tool for PULSER

†Kazunobu KONDA,Takahiro YAKOH,Yuichiro ANZAI
‡Keio University

終了 mtp_stop_foo のマクロでスレッドを終了させるシステムコールに展開される。

中断 mtp_susp_foo のマクロでスレッドの処理を中断させるシステムコールに展開される。

再開 mtp_resm_foo のマクロで中断されていた処理を再開するシステムコールに展開される。

同期 同期をとるスレッド同士は1対1に限定し、1つ同期の組合わせに対して1つの *DI* を割り当てる。待ちのスレッド foo1 にはmtp_wait_foo2 が与えられ、それは自分の *DI* を同期用の *DI* に変更して待ち状態に入る操作に展開される。また、対になる呼び出し側のスレッドfoo2 にはmtp_sig_foo1 が与えられ、これは同期用の *DI* 信号を発生するシステムコールに展開される。

再開始 MTP_START 内にスレッドを終了させるシステムコールを定義し MTP_END 内に始動させるシステムコールを定義する。これらはスレッドを記述する際に処理の最初と最後に記述する。

休眠 MTP_START 内にスレッドの処理を中断させるシステムコールを定義し MTP_END 内に再開させるシステムコールを定義する。これらはスレッドを記述する際に処理の最初と最後に記述する。

また、スレッド間の関係を記述する際に INITSTART foo1,foo2,...; と記述し、プログラムの始動時に実行を開始するスレッドを指定する。これらのスレッドはmtp_main.c 内で始動される。

5 プログラム例及び記述性の検討

ここではMTP を用いてロボットの制御プログラムの例を示す。図.1 はスレッド間の関係である。plan_and_move は目的地までの道順を計画し、その計画をもとに移動する。sense_forward は前方を確認し障害物の有無を調べる。obstacle_avoidance は障害物回避行動をおこなう。sense_for_oa は障害物回避行動中に環境からの情報を採り入れる。

plan_and_move とsense_forward は並行に動作し後者は前方に障害物がある場合には obstacle_avoidance を始動する。また、obstacle_avoidance と plan_and_move は排他的に実行しなければならない。したがって、obstacle_avoidance は plan_and_move に対してを再始動の関係にある。また、obstacle_avoidance は sense_for_oa から環境の情報を受け取りながら障害物回避を行うので、双方で同期をとる。

以上のような、関係を図1のように静的に一括して記述することによりスレッド間の関係を容易に把握できる。したがってプログラミングにおけるスレッド間関係の実現に伴う複雑さを軽減できた。

図 2 はobstacle_avoidance の記述の例である。MTP_START 及びMTP_END は、plan_and_end を再始動させる命令に展開される。またmtp_wait_sense_for_oa の命令では、sense_for_oa からの同期用の *DI* を待ち、mtp_sig_sense_for_oa は同期用の *DI* を発生する。

以上のようにして、*DI* を意識することなくスレッド間での同期を実現することができる。同様にして他の関係も *DI* を意識せずに実現できるので、スレッドと *DI* との対応関係に伴う繁雑さがなくなった。

6 おわりに

ロボットの制御プログラムに焦点をあて、マルチスレッドプログラムを作成する上でスレッド間の関係を調べ、プログラム中のスレッドの関係を与えることによりそれらをPULSER 上に実現するツールを作成した。

参考文献

[1] 菅原 智義, 矢向 高弘, 安西 祐一郎: 自律移動ロボット用 OS PULSER における Direct-interrupt 機構, 情報処理学会第 44 回（平成 4 年前期）全国大会予稿集,1992.

[2] 矢向 高弘, 菅原 智義, 安西 祐一郎: 物理世界の状態変化に敏感なシステム PULSER-II の提案と実装, 情報処理学会第 44 回（平成 4 年前期）全国大会予稿集,1992.

```
TH_NAME plan_and_move;
TH_NAME sense_forward;
TH_NAME obstacle_avoidance;
TH_NAME sense_for_oa;
%%
START sense_forward -> obstacle_avoidance;
RESTRT obstacle_avoidance -> plan_and_move;
START obstacle_avoidance -> sense_for_oa;
STOP obstacle_avoidance -> sense_for_oa;
SYNC obstacle_avoidance -> sense_for_oa;
SYNC sense_oa -> obstacle_avoidance;
%%
INITSTRT plan_and_move,sense_forward;
```

図 1. スレッドの定義と関係

```
#include "obstacle_avoidance.h"
obstacle_avoidance()
{
/* auto variable definition */
MTP_START
mtp_start_sense_for_oa;
/* obstacle avoiding */
mtp_wait_sense_for_oa;
...
mtp_sig_sense_for_oa;
...
mtp_stop_sense_for_oa;
MTP_STOP
}
```

図 2. スレッドobstacle_avoidance() の記述例