時間制約構文の実装*

1V－3

佐藤 孝治†
NTT 基礎研究所

1 はじめに

ハードウェアや圧縮技術などの進歩により、UNIX ワークステーション上で音声や映像などの連続データを処理することが可能となってきている。連続データを処理する際には、それを構成する個々のデータが持つ時間制約を満たすようにしなければならない。しかし UNIX 環境では、時間に関する処理は通常タイマーやシグナルなどの機構を組み合わせなければならないので、プログラムが非常に複雑になってしまう。そのため時間制約を直接記述する構文が必要となる [1]。本論文では、C 言語の文レベルで時間制約を明示的に記述するための構文とその実装について述べる。時間制約構文を用いることにより、時間制約を持つ処理を簡潔に記述することが可能となる。

2 時間制約構文

連続データの処理では、処理の開始時間、終了時間、周期的実行などの時間制約の指定や、時間制約が守れなかったときの例外処理の指定が必要となる。ここでは C 言語の文レベルでこれらを記述するための構文について述べる。

2.1 BEFORE

BEFORE は処理の終了時間に関する制約を規定する。

```
BEFORE (struct timespec *end, int flag)
        {statement}*
[EXCEPT
        {statement}*]
END_BEFORE
```

*end*は処理が終了すべき時間であり、絶対時間 (時刻) または相対時間で表される。*flag*が ABSTIME のとき、*end*は絶対時間 (世界標準時 (Universal Coordinated Time)1970 年 1 月 1 日 0 時 0 分からの経過時間) である。*flag*が RELTIME のとき、*end*は現在時刻からの相対時間である。

*end*までに本体の処理が終了したときは、処理の終了とともに **BEFORE** から抜け出る。*end*までに本体の処理が終了しなかったときはタイムアウトが発生する。その時点で本体の処理は中断され、**EXCEPT** 以下の例外処理部に制御が移る。例外処理部の処理が終了すると、**BEFORE** から抜け出る。例外処理部が省略されているときは、タイムアウトが発生するとともに **BEFORE** から抜け出る。

明示的に **BEFORE** からに抜け出るには **BREAK** を用いる。**BREAK** はそれを含む静的に最も内側の **BEFORE** をただちに終了させる。またタイムアウトが発生するまで待つには **CONTINUE** を用いる。**CONTINUE** はタイムアウトが発生するまで待ち、例外処理部を実行せずに **BEFORE** から抜け出る。

2.2 AFTER

AFTER は処理の開始時間に関する制約を規定する。

```
AFTER (struct timespec *start, int flag)
        {statement}*
END_AFTER
```

*start*は処理を開始すべき時間であり、絶対時間または相対時間で表される。*flag*が ABSTIME のとき、*start*は絶対時間である。*flag*が RELTIME のとき、*start*は現在時刻からの相対時間である。

現在時刻が *start* 以前ならば、*start*になるまで待ち、その後本体の処理を開始する。現在時刻がすでに *start* 以降ならば、ただちに本体の処理を開始する。

2.3 CYCLE

CYCLE は周期的実行を規定する。

```
CYCLE (struct timespec *start, struct timespec *end,
        struct timespec *period, struct timespec *deadline,
        int flag)
        {statement}*
[EXCEPT
        {statement}*]
END_CYCLE
```

start、*end*、*period*、*deadline*はそれぞれ処理の開始時間、終了時間、周期、デッドラインである。*start*と *end*は絶対時間または相対時間で表される。*flag*が ABSTIME のとき、*start*と *end*は絶対時間である。*flag*が RELTIME のとき、*start*と *end*は現在時刻からの相対時間である。

*start*から *end*までの間、周期 *period*、デッドライン *deadline* で繰り返し処理を行なう。*deadline*までに本体の処理が終了したときは、次の周期の開始時間まで待ち、本体の先頭から処理を再開する。*deadline*までに本体の処理が終了しなかったときはタイムアウトが発生する。その時点で本体の処理は中断され、**EXCEPT** 以下の例外処理部に制御が移る。例外処理部の処理が終了すると、次の周期の開

*Implementation of Constructs for Expressing Timing Constraints
†Koji Sato (koji@square.ntt.jp), NTT Basic Research Laboratories

始時間まで待ち、処理を再開する。例外処理部の処理中に次の周期の開始時間になると、その時点でタイムアウトが発生する。この時点で本体の先頭に制御が移り、次の周期を開始する。本体や例外処理部をタイムアウトから保護するには後述する **PROTECT** を用いる。例外処理部が省略されているときは、次の周期の開始時間まで待ち、処理を再開する。

明示的に **CYCLE** から抜け出すには **BREAK** を用いる。また次の周期の開始まで待つには **CONTINUE** を用いる。

2.4 PROTECT

PROTECT はタイムアウトからの保護を規定する。

```
PROTECT
        {statement}*
END_PROTECT
```

PROTECT 本体の実行中にタイムアウトが発生しても、そのタイムアウトは本体の終了まで延期させられる。**PROTECT** は **BEFORE** や **CYCLE** の本体や例外処理部で用いることができる。

2.5 その他の関数

関数 getresttime(**int** *pos*, **struct** timespec **rest*) は、各構文で指定した時間までの残り時間を *rest* に返す。*pos* には BEFORE_END、CYCLE_END、CYCLE_PERIOD、CYCLE_DEADLINE のいずれかを指定する。これらはそれぞれ **BEFORE** における終了時間、**CYCLE** における終了時間、周期の終わり、デッドラインまでの時間を表す。

3 時間制約構文を用いた例題

以下の例題では、開始から 10 秒間、周期 0.1 秒、デッドライン 0.08 秒で関数 get_frame と proc_frame を呼び出し、フレームを処理する。デッドラインが守れないときは例外処理部の関数 skip_frame が呼び出され、処理中のフレームを飛ばす。例外処理部はタイムアウトから保護されている。

```
#include "timing.h" /* 時間制約構文ヘッダファイル */

struct timespec start, end, period, deadline;
start.tv_sec    =  0; start.tv_nsec    =         0;
end.tv_sec      = 10; end.tv_nsec      =         0;
period.tv_sec   =  0; period.tv_nsec   = 100000000;
deadline.tv_sec =  0; deadline.tv_nsec =  80000000;
CYCLE (&start, &end, &period, &deadline, RELTIME)
        get_frame(...);  /* フレームを取得 */
        proc_frame(...); /* フレームを処理 */
EXCEPT
        PROTECT
                skip_frame(...); /* フレームを放棄 */
        END_PROTECT
END_CYCLE
```

4 実装方法

時間制約構文を Sun SPARCstation 10/SunOS 5.3[5] 上に実装した。これらは C 言語のマクロ、および実行時ライブラリで構成されている。マクロはタイマーの管理、シグナルハンドラやシグナルマスクの設定、大域的ジャンプによる制御を行なうコードに展開される。

開始時間、終了時間、周期、デッドラインのそれぞれに対してタイマーを生成し、指定された時間にシグナルが送られるようにセットする。**BEFORE**、**AFTER** は 1 つ、**CYCLE** は 3 つのタイマーを使用している。シグナルを受け取ると、対応するシグナルハンドラが起動され、大域的ジャンプにより例外処理部などの適切な場所に制御を移す。

時間制約構文は入れ子にすることができる。時間制約の情報はスタックを用いて管理している。新しい時間制約構文に入るたびに、その時間制約の情報はスタックにプッシュされる。時間に関する各種の関数は通常スタックの最も上にある時間制約に対して適用される。終了時に時間制約の情報はスタックからポップされる。

また本構文は Solaris スレッドライブラリ [4] と共用することが可能である。本構文をスレッドライブラリや実時間スケジューリング機能と組み合わせることにより、周期的スレッド [6] などの時間制約を持つスレッドが実現ができる。

5 まとめ

本論文では、時間制約を明示的に記述するための構文とその実装について述べた。これにより、時間制約を持つ処理を簡潔に記述することが可能となる。

参考文献

[1] Yutaka Ishikawa, Hideyuki Tokuda, and Clifford W. Mercer. "Object-Oriented Real-Time Language Design: Constructs for Timing Constraints". In *ECOOP/OOPSLA '90 Proceddings*, pp. 289–298, October 1990.

[2] Eugene Kligerman and Alexamder D. Stoyenko. "Real-Time Euclid: A Language for Reliable Real-Time Systems". *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 9, pp. 941–949, September 1986.

[3] Vivek. M. Nirkhe, Satish K. Tripathi, and Ashok K. Agrawala. "Language Support for the Maruti Real-Time System". In *Proceedings of the 11th IEEE Real-Time Systems Symposium*, pp. 257–266, December 1990.

[4] Sun Microsystems, Inc. *SunOS 5.3 Guide to Multi-Thread Programming*, May 1993.

[5] Sun Microsystems, Inc. *SunOS 5.3 Reference Manual*, May 1993.

[6] Hideyuki Tokuda, Tatsuo Nakajima, and Prithvi Rao. "Real-Time Mach: Towards a Predictable Real-Time System". In *Proceedings of USENIX Mach Workshop*, pp. 1–10, October 1990.

[7] Victor Wolfe, Susan Davidson, and Insup Lee. "*RTC*: Language Support for Real-time Concurrency". In *Proceedings of the 12th IEEE Real-Time Systems Symposium*, pp. 43–52, December 1991.