CMU Common Lisp上の疑似マルチタスク機構の設計と実装

4E−8

古坂孝史[†1]
情報処理振興事業協会　技術センター

1 はじめに

本稿は、ウィンドウシステムにおけるマウスイベントを処理するための疑似マルチタスク機構について述べる。分散ウィンドウツールキット *YyonX* [1] では、マウスイベントの処理機構としてリアルタイムにマウスイベントの処理をする仕組みを提供している。その仕組みは、キーボードが押下された／マウスが移動した／ある領域に入った／出た／マウスボタンが押下されたなどのイベントを非同期的に受け付け、利用者によって登録された関数を呼び出す。イベントの受け付けは、他の関数が実行中でも行なわれる。この仕組みを *YyonX* では、いくつかCommon Lisp処理系が持つマルチタスク機構で実装している。

CMU Common Lisp[2] は、CMUで開発されたCommon Lisp処理系である。しかし、この処理系は、イベント処理のためのマルチタスク機構がない。そこで、この処理系の持つ割り込み処理機構を利用してイベント処理特有の疑似マルチタスク機構を設計し、イベント処理のための機構を実装した。実装の結果、CMU Common Lisp上で *YyonX* の稼働が確認できた。

2 ウィンドウシステムのイベント処理の特徴

多くのウィンドウシステムでは、イベントを処理するためにFIFOキューを用いて、発生したイベントをキューに蓄えたりキューからイベントを取り出す仕組みを提供している。通常、キューにイベントを蓄える処理とイベントを取り出す処理は、同一プロセス上に同期して存在している。その結果、キューからイベントを取り出して割り当てられた処理を呼び出している間は、イベントの取り出し処理ができない。従って、イベントに対応した処理の追い越しが存在しない。

多くのLisp上のウィンドウツールキットでは、ウィンドウツールキット自身の操作を行なう環境とウィンドウマネージャ機能が混在している。そのため、ある状況下では、ウィンドウツールキットの操作である描画処理等を行なっている間でもウィンドウマネージャ的な操作が行なわれる。従って、イベントに対応した処理の追い越しが必要となる。

また、別の状況下では、イベント処理を同期して行ないたい場合がある。例えば、マウスの移動のイベントにより処理を行なう場合である。具体的には、マウスの移動に伴って絵が移動する場合である。この場合、移動に伴って絵の移動が行なわれ、描画処理が終了するまで次のイベントに対応した処理を実行して欲しくない。そこで、イベント処理は、

(1) イベントは、非同期的に発生するので、イベントを取り込む処理は、いつでもイベントを取り込む。

(2) イベントに対応した処理は、イベントを受けとったら即座に起動する。

(3) 利用者の指示によりイベントに対応した処理を実行中は、イベントに対応した処理を実行しないモードを設ける。

とする必要がある。

3 イベント処理の設計

3.1 *YyonX* でのイベント処理の基本設計

イベントを操作する処理は、イベントを監視する処理とイベントを取り出す処理、取り出したイベントで対応する処理を呼び出す処理の3つの処理からなる。本稿では、それぞれをイベント監視処理、イベント取り出し処理、イベントディスパッチ処理と呼ぶ。これら3つの処理は、それぞれが同期して処理を行なうが、他の処理との同期の必然性はない。また、これらの一連の処理はウィンドウツールキットが稼働中は停止せず、常に繰り返して処理を行なう。これらの処理をまとめた流れを図1に示す。

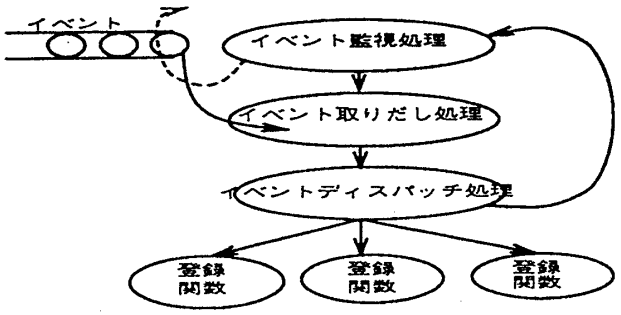

図1: イベントを操作する処理の流れ

イベント監視処理は、マウスやキーボードデバイスまたは、ウィンドウシステムで起こるイベントの発生を監視する。イベントの発生が確認できたら、イベント取り出し処理を起動する。

イベント取り出し処理は、イベントを取り出しイベントディスパッチ処理を起動する。

イベントディスパッチ処理は、取り出したイベントに対応する利用者が設定した関数の起動を行なう。予め利用者は、設定する関数に同期モードと非同期モードの指示を与えておく。イベントディスパッチ処理は、モードにより関数の呼びだし方式を変更する。同期モードであれば、単純な関数呼びだしを行なう。非同期モードであれば、イベントに対応する利用者が設定した関数の実行だけを行なわないような関数呼びだしを行なう。

以上ような一連の処理をまとめて関数化し、処理系の持つマルチタスク機構を利用すれば、イベントに対

A Design and Implementation of the pseudo multitask mechanism on CMU Common Lisp,
Takashi KOSAKA, Software Technology Center,
Information-technology Promotion Agency (IPA)

†1古坂 孝史 (株)CSK から出向中

応した処理の追い越しが可能となる。

3.2 マルチタスク機構をもたない処理系での設計

マルチタスク機構がない処理系では、イベント処理をまとめた関数を呼び出すだけでは、イベントに対応した処理の追い越しができない。そこで、追い越し処理を行なうためには、何らかのタイミングで定期的にイベント処理を起動できれば良い。

定期的にイベント処理を起動するためには、

(1) イベントに対応した処理の関数のボディを分割して、基本となる関数を逐次実行する度に起動する。

(2) オペレーティングシステムに依存したタイマー割り込みの機構を利用して一定の時間で起動する。

の二つの方法がある。

前者は全ての処理系で汎用的に利用可能である反面イベント処理に対応した関数に対して、他の環境でコンパイルできない等の制限が加わる。従って、イベント処理に制限が加わることになりウィンドウツールキットの内部機構に向かない。

後者は、他言語呼びだし等の処理系独自の機構を利用するが、イベント処理に対応した関数に関して制限がない。従って、イベント処理に制限がないため、ウィンドウツールキットの内部機構に向いている。

4 イベント処理の実装

割り込み処理による疑似マルチタスクを実装するために、UNIX のタイマー割り込みを利用した。タイマー割り込みは、プログラマが意図する時間で設定した割り込み関数を起動する仕組みを提供している。(図 2参照)

```
alamsetup()
{
   struct itimerval value;
   value.it_interval.tv_sec = 0;
   value.it_interval.tv_usec = 0;
   value.it_value.tv_sec = 0;
   value.it_value.tv_usec = 500000;
   setitimer (ITIMER_REAL,&value,NULL);
}
```

図 2: タイマー割り込みのプログラム

図 2中の it_interval は、サイクリックに割り込みを発生する時間を与え、また、it_value は割り込みが発生する時間を与える。図 2の例では、0.5 秒に一回割り込みが発生することになる。

図 2の関数を Lisp 側で起動すれば、設定した割り込み関数が起動される。この割り込み関数の設定を CMU Common Lisp では、enable-interrupt という関数が利用可能である。その使用法は、

(enable-interrupt 14 #'yy-tuuchi)

のようにシグナル番号と関数オブジェクトを引数に記述する。

YyonX で利用しているプログラムは、図 2のプログラムを使って、0.5 秒に一回 yy-tuuchi の起動がかかる。yy-tuuchi は、イベント監視処理を行ない、基本イベントがあれば、イベント取りだし処理、イベントディスパッチ処理を行なう。これらの処理を実装した例を図 3に示す。

```
(in-package :YY :use
   '("LISP" "SYSTEM" "ALIEN" "C-CALL"))
;;; 他言語関数の設定
(def-alien-routine "alamsetup" int)
;;; イベント処理
(defun yy-tuuchi (signal code scp)
  (let ((event nil))
    (if (イベント監視処理)
      (progn
        (setf event (イベント取りだし処理))
        (alamsetup)
        (イベントディスパッチ処理 event))
      (alamsetup))
  ))
;;; 割り込み関数のセットアップ
(enable-interrupt 14 #'yy-tuuchi)
```

図 3: 割り込みを利用したイベント処理関数

図 3では、イベントの有無に拘らず alamsetup を呼びだしている。この設定により yy-tuuchi は、必ず 0.5 秒後に割り込みにより起動がかかる。

上述のような割り込み処理を起動する関数で、イベントに関連した一連の処理を疑似マルチタスク化が実現できる。イベントに関連した一連の処理は、Lisp の持つ TOP-LEVEL-LOOP とは独立に実行している。更に、イベントに関連した利用者定義の関数が実行されている時でも新たなイベントが来ればそのイベントに対応した処理の実行が可能となる。

利用者定義の関数が同期モードの場合、yy-tuuchi の割り込み処理の必要がない。その場合は、利用者定義の関数を呼び出す時に CMU Common Lisp で用意しているマクロ without-interrupts を利用する。このマクロで囲まれた関数が実行している間は、割り込み処理が起動されない。

5 おわりに

CMU Common Lisp のもつ割り込み処理機構を用いた疑似マルチタスクでイベント処理の設計と実装を報告した。割り込み処理を利用する上で、同期モードで利用者の定義した関数を利用する場合、without-interrupts を利用するが、実際の実装では、このマクロが入れ子処理にならないように工夫する必要がある。

今後の課題として、マルチタスク機構がない他の処理系でも適応可能な統一的なインタフェースとして定義することがある。

参考文献

[1] 井田昌之、他 : *YyonX*: 概要設計, 情報処理学会第 40 回全国大会, March 1990.

[2] Robert A.MacLachlan: CMU Common Lisp User's Manual, Carnegie Mellon Univ. May 15 1992.