対象システムと独立したプロセスによるヘルプ機能の実現 *

2Q－2

竹中佐和　青柳龍也　有山正孝†‡
電気通信大学

1　はじめに

現在ヘルプ機能の多くは、対象システムに組み込まれているためヘルプ内容が固定化されている。しかし学校の演習などでこれを用い、学生の意見を聞いた後、より使いやすいように教師がヘルプ機能を修正したい場合、組み込み型ではヘルプ部分がシステム本体のプログラムと深く依存するため修正することは困難である。

そこで本稿ではマルチタスク環境を利用し、ヘルプ機能を対象システムの外に出して、対象プロセスとヘルププロセスの間で情報をやりとりをしながらその場に応じたヘルプ情報を提供するという、対象システムと独立したヘルプシステムの構築を目指す。こうするとヘルプシステムだけに注目してヘルプ内容を追加変更することができるため、修正しやすく、その結果内容が固定化しない柔軟性のあるヘルプシステムになる。

2　システムの概要

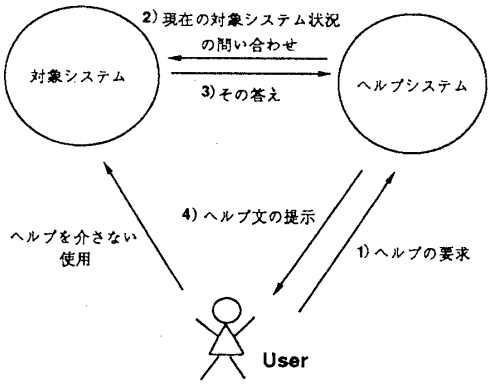

図 1: 全体構成

図1に全体構成を示す。ヘルプシステムは親プロセスである対象システムから起動する非同期プロセスであり、これら2つのプロセスが並列に動くことで全システムが実現する。

ユーザはヘルプが必要でない場合には対象システムを普段と変わりなく使用し、ヘルプ情報が欲しい時のみヘルプウインドウをクリックする。ユーザからヘルプ要求が起こると、対象プロセスとヘルププロセスが互いに情報をやりとりして、現状況に応じたヘルプ情報を提示する。

3　ヘルプの動作例

並列処理が可能なUNIXワークステーション環境で動作している。図2の上部に示すヘルプウインドウが常に表示されており、その中央のヘルプボタンをクリックすると、下部のようにヘルプ画面にヘルプ情報が表示される。

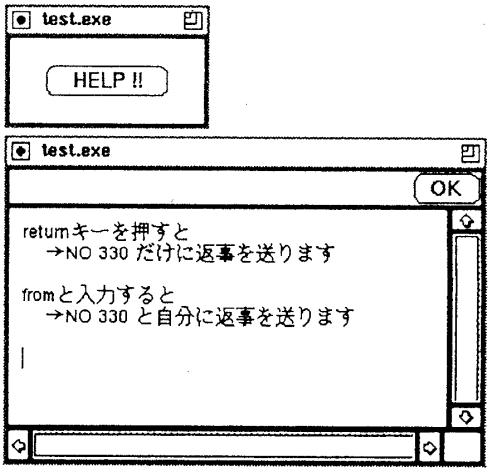

図 2: ヘルプウインドウとヘルプ画面の1例

4　通信プロトコル

効率の良いシステムにするためには、プロセス間の通信のプロトコルは重要な部分である。

本稿での通信の流れを説明する。

1. ヘルププロセスがユーザによるヘルプウインドウのクリック動作を受けとり、通信を開始する。まず対象システムにHELPという文字を送る。

2. 対象プロセスはこれを受けとり、ある定まった初期情報を返す。

3. ヘルププロセスはその情報を判定して、ヘルプに必要な残りの情報を問い合わせる。

*Help system with independent process of target system
†Sawa TAKENAKA, Tatsuya AOYAGI, Masataka ARIYAMA
‡The University of Electro-Communications

情報	内容
major-mode	カレントバッファのメジャーモード名
minibuffer-window-selected	カーソルがミニバッファウインドウにあるかないか (t/nil)
minibuffer-prompt	ミニバッファに出るプロンプトの文字列
minibuffer-prompt-default	プロンプト表示のデフォルト部分の文字列
user-last-command	ユーザがキーボードから入力した最新のコマンド名
user-answer	ユーザがプロンプトに対して入力した文字列

表 1: Emacs から渡される最小限のヘルプ情報の 1 例

4. 対象プロセスは、それにより対応する情報を返す。
5. 3.4. の動作をヘルププロセスが目標とするヘルプ情報を得るまで繰り返す。
6. ヘルププロセスが必要なヘルプ情報をすべて得たと判断すると、それに対応するヘルプ文をヘルプウインドウに表示する。

このようにプロトコルの基本は、ヘルププロセスが情報を問い合わせ、対象プロセスがそれに答える、というものである。

5 システムの実装方法

5.1 ヘルプシステムの実装方法

ここではヘルプをおこなうために必要な情報を問い合わせ、その答えを受けとり、判断し、ヘルプ文をヘルプウインドウに提示している。これは C++ により構築されいる。

5.2 対象システムの実装方法

対象システムはヘルプシステムからの問い合わせを受けとると、それに対応する情報を返す。

よって情報を外のプロセスに渡す機能が必要となる。本稿ではもともとあるライブラリ関数を利用して、ヘルプに必要な情報を取り出す部分と、ヘルププロセスとやりとりをする部分を追加することができる Emacs エディタを対象システムに選ぶ。

対象システムのプロトタイプとして、Emacs の MH (Message Handling System) パッケージを選び、これをヘルプをするために最小限必要な MH 情報を表 1 に示す。

5.3 通信のアルゴリズム

ここで、図 2 を例にとってヘルプ文ができるまでの通信のアルゴリズムを具体的に示す。

注) `Help>` はヘルププロセスがおこなう動作を、`Emacs>` は Emacs 対象プロセスがおこなう動作を示す。

`Help>` HELP 文字列を送る。
`Emacs>` `major-mode`, `user-last-command`, `minibuffer-window-selected` として、`Fundamental-mode`, `mh-reply`, `t` を返す。
`Help>` それぞれの情報から判断し、次に `minibuffer-prompt` を問い合わせる。
`Emacs>` その答として現在ミニバッファに表示されている文字列、`Reply to whom:` を返す。
`Help>` `mail-number` を問い合わせる。
`Emacs>` 現在カーソルが指しているメールの番号 330 を返す。
`Help>` この状況に対応するヘルプ文が書かれた外部のテキストファイルを読み込み、メール番号を入れたヘルプ文をヘルプウインドウに表示する。

6 おわりに

本稿では、ヘルプ機能の変更がヘルプ部分だけに注目しておこなえるように、ヘルプシステムの新しい形を提案した。この形を利用すると、対象システムに手を加えることなく、ヘルプシステムを変更するだけでいろいろな応用例が考えられる。たとえば、ヘルプシステムにユーザモデルを持たせることで、初心者向けのヘルプや上級者向けのヘルプなど、ユーザに応じてヘルプを使い分けることができる。また教え方を定めたコースウェアを追加すると、システム主導型の CAI システムとなる。

さらに対象システム側に、常にユーザの情報を流す部分を追加すると、ヘルプシステムがそれによりユーザの動作を判断し、その動作よりもよい方法を教えるというチュートリアルシステムにもなりうる。

このように対象システムとヘルプシステムの 2 つを独立させることで、ヘルプ機能をさまざまな形に発展しやすくなり、柔軟性のあるヘルプシステムとなる。

参考文献

[1] 「コンピュータ支援の教育システム – CAI」中山和彦 木村捨雄 東原義訓 ：東京書籍刊

[2] 「GNU Emacs Lisp Manual」 Bill Lewis 監督 Walking Lint 訳 ：ビレッジセンター出版局