Emacs 解剖学

Lecture 4

Multics Emacs

井田昌之

Multics にも Emacs があった

1970年代に大きな注目を浴びていた大型汎用機のシステムに，Multics というシステムがあります．Multics はベル研究所，GE，および MIT の Project MAC の共同研究の形で1965年頃に立案が始まりました．大規模なコンピュータシステムの利用形態を実現する研究として有名で，67年くらいからその姿を見せ始め，1969年10月に一般の利用者に公開されるようになります．

Multics システムはその後も開発が続けられますが，69年にベル研が Multics の肥大化などから完成を待たずに抜け，その後 GE のコンピュータ部門がハネウェルに買収され，80年代中期に HIS DPS-8 上のものを最後に姿を消していきます．この Multics は，当時としては画期的な概念を多数生み，それは今も UNIX その他に多くが引き継がれています．手を引いたベル研は，この Multics 流の OS の経験（と反省）を生かして，UNIX の開発へと向かうのです．このため UNIX には小型版の Multics とも言えるほど，いろいろなアイディアが取り入れられています．たとえば，文献1)などから当時の概念をかいま見ることができます．

Multics は，GE 635，645 そして，HIS 6180，DPS-8 といった機種の上で動作しました．これらはすべて36ビット1語のワードマシンです．この Multics で初めて実用的に使われ，その後にも継がれていった技術概念には次のようなものがあります．

1) 階層的ファイルシステム
2) 仮想メモリ
3) ダイナミックリンキング

これらの詳細や，ハードウェアとの関連などは，たとえば，文献2)などから参照していくことができます．また，最近はやりの WWW に，かなり大きな規模で Multics に関する情報をまとめている Web ページがあります．

http://www.best.com/~thvv/features.html

というもので，ここからいろいろな情報を参照することができます．

Multics は MIT の Project MAC の産物でもあり，その近辺での研究にたくさん使われていたので，Multics は MIT 的なシステムだったということもできるでしょう．そうすると，当然，Lisp や Emacs が意味をもってきます．

この Multics での Emacs は使っていた人たちの間では高い評価を受けていたツールでしたが，Multics の終えんとともに歴史の中に埋もれていきました．また，Richard Stallman が作った Emacs 処理系とは別の処理系が同時期に作られていたことを知る人はそう多くないようになってきました．

そこで，今回は，この Multics のために生まれ，Multics とともにその役割を終えた Emacs について，紹介します．

MacLisp＋画面エディタ＝Multics Emacs

Multics 上の Lisp

Multics 上の最初の Lisp は，David Reed によって PL/I を使って開発された Version 1 Lisp というものです．これはほかの Lisp との互換性がほとんどなかったので，あまり使われなかったようです．

この後，Project Mac のニーズとして，Macsyma が大きくなってその頃使っていた PDP-10 のアドレス空間ではうまくいかなくなり，Multics に移植しようという要求が出てきます．その結果，その記述言語である MacLisp の Multics への移植が進められました．これは 1970 年から 73 年頃にかけてのことです．Reed に加えて，David Moon が MIT の学部学生として加わり，彼は実行速度に大きな影響のある部分，特に eval を ALM（アセンブラ）で書き換えました．

1974 年に Bernerd Greenberg が加わり，彼がハネウェルを離れる 1980 年までは MacLisp の保守と改良は彼の手によって行なわれます．1980 年以降は Richard Soley，Barry Margolin らがその機能を引き継ぎます．Multics MacLisp 自身の紹介もいろいろな点でしておきたいのですが，この辺にとどめておきましょう．

Multics MacLisp は PL/I で書かれています．MacLisp だけでなく，Multics システムは PL/I でほとんどが書かれているとされますが，その初期には EPL（Early PL/I）と呼ばれる言語が使われていました．Multics ハードウェアおよび OS の開発と PL/I 言語処理系の開発が並行して行なわれていたからです．EPL は TMG（Transmoglifier）と呼ばれる小さな言語（コンパイラコンパイラ）で書かれ，その EPL を用いて Multics が書かれ，そしてそれが後に PL/I に置き換えられました．EPL は PL/I の完成が遅れたので，5 年ほど使われ，そうなると，それ自身の最適化フェイズの開発や，機能拡張が持ち込まれました．EPL の担当者達の苦労話は Corbato によって文献 3) に記されています．一口に言って 80 年以前では，アセンブリ言語でない高級言語で OS などのシステムを開発すること自身が大きなテーマでした．ソフトウェアシステムをポータブルにする，あるいは移植性を考えて書くというような概念はあまり存在していなかったのです．それには，ソフトウェアシステムの記述はアセンブリ言語が普通だったということも関係があります．たとえば，文献 4) をまとめたのもそのころでした．その中にも書きましたが，当時は PL/I 系のシステム記述言語が中心的に検討されていました．ちょうど Pascal や C が出る少し前です．

最近では，なんでも C 言語で書くというのが普及しています．その理由の一つには，「C で書いておけばいろいろなシステムで動かせるから」ということがあります．当時はそんな概念はありませんでした．

なお，Corbato は同じ論文[3]の中で，「次に作るなら，機能の豊富さのためにルーチンライブラリを用意し，言語仕様としてはもっと軽装にしたい」という文章を付けています．

C 言語が普及する以前にあるこうしたコメントは，みごとに C の時代を予見させるものだといえます．この C 言語に期待された特徴は Lisp にも共通するものですが，C はスタティック，Lisp はダイナミックであって，その後の歩みを分かれさせるもとになっていきます．

Multics 上のエディタと Emacs

Emacs の登場の前に Multics には四つのエディタがあったことが文献 5) に記されています．

それによると，CTSS の EDL エディタから発した edm エディタ，QED から出た qedx エディタの二つがともにハネウェルの製品として標準的なエディタとして存在していました．それらは，次のような特徴をもっていました．

- 行エディタ
- コマンドモードと入力モードをもつ
- 半二重の（低速）プリンタ端末を想定

第 3 のエディタは ted と呼ばれる qedx で，J. Falksen が作ったものです．これは，製品版の qedx より強力な機能をもっていました．

第 4 のエディタは TECO で，PDP-10 から 1971 年に流れてきました．これは，ディスプレイサポートはないもので，AI 研で拡張された機能はほとんど入っていませんでした．一般用のツールというより，システムツールとして使われました．製品としてハネウェルによりサポートされていました．

ITS Emacs を Multics に持ち込むか?

ITS Emacs というのは，Richard Stallman が作った Emacs で，TECO 上のマクロとして作られた，PDP-10 の ITS という OS 上のものです．その開発の時期はおおむね，1975 年から 77 年でした．これは，GNU Emacs の元版だということも言えます．

TECO のもともとのソースは，PDP-10 のアセンブラで書かれていて，それに依存していました．ITS OS の場合，Tenex/Twenex の場合，TOPS-10 (20) の場合というような切分けは書き込まれていたようです．しかし，それでもほかの機種に移植するのは困難でした．したがって，TECO が移植できなければ ITS Emacs は移植できません．

Multics Emacs は 78 年に開発が始まりますが，TECO 上ではなく，MacLisp を記述言語として開発されます．のちの GNU Emacs のように機能拡張は Lisp 言語によってなされるようになります．Bernie Greenberg は，「記述言語として Lisp は最高だ．そして機能拡張言語としても Lisp が一番いい．Lisp をベースにする Emacs は私の Emacs が最初のものだ．私はできるだけ Richard の ITS Emacs の考え方を尊重して機能も同じになるようにしたが，その処理系はすべて私が設計し，開発したものだ．私が考え出したもので Richard が取り入れていったものもたくさんあるよ．」と言っています．これはおよそ正しい主張だと思われます．

「Richard Stallman の作った Emacs は，TECO のマクロをずるずると改良していっただけなんだ．それにひきかえ Multics Emacs は，最初から Emacs エディタとして作った．こういう完全なシステムなんだ．」

「当時，Multics では PL/I が最も広く使われていて，バグも少なかった．性能も良かった．でも，Emacs で必要な機能拡張性は PL/I を使ったのでは得られない．それで，Lisp を使った．」と，大変に自信と愛着をもって話してくれます．

MacLisp で開発

Multics Emacs は MacLisp で書かれています．Multics Emacs は 1978 年 3 月に誕生しました．Multics の誕生は 60 年代ですから最初から，これらがあったわけではありません．むしろ歴史から言えば，だいぶ後の時代に登場したということもできます．一番重要な働きをしたのは，Bernie Greenberg という人です．協力者としては Bill York，Gary Palter，Richard Lamson などがいました．そして，彼が離れた 1980 年 12 月以降は R. Soly，Barry Margolin らによって保守されていきます．Multics Emacs は，Multics の終りまでハネウェルの製品として使われていったのです．

Multics の特徴的な概念

Multics の特徴的な概念として，次の三つを冒頭で紹介しました．

1) 階層的ファイルシステム
2) 仮想メモリ
3) ダイナミックリンキング

これら三つの基本的な特徴概念は，必然的に Multics 上の Emacs の特徴にも影響しています．

1) **階層的ファイルシステム**：現在のほとんどのコンピュータシステムにある「ディレクトリ」，「パス名」，あるいは「ワーキングディレクトリ」な

どの概念です．それ以前のシステムにはこうした概念はありませんでした．エディタが本来対象とするファイルシステムの操作概念はこれ以後，徐々に共通化できるようになったと言えます．

2) **仮想メモリ**：セグメンテーションとページングを併用した機構があり，Multics では大変大きなメモリ空間を扱うことができました．18 ビットでセグメント番号を示し，さらに 18 ビットでセグメント内のアドレスを示します．間接指定などというフラグもアドレス指示（ポインタ）に用意されていました．

このポインタは，（セグメント番号を 12 ビットだけにして 36 ビット 1 語で表現する方法もありますが）2 語，言い換えれば 72 ビット長で表現されていました．つまり，メモリオブジェクトへのポインタはそもそも 72 ビット長であったわけです！

これが Lisp のオブジェクトになります．これは大変な大きさです．これを生かさない方法はありません．アドレスの指定には 18＋18 で 36 ビットあればよく，また，未使用のビットがたくさんあり，これら未使用の部分を使って，MacLisp ではその Lisp オブジェクトの型の識別情報を入れていました．Multics Emacs は MacLisp で書かれていましたから，Emacs のデータオブジェクトの識別もこれらの未使用ビットを生かして行なわれたのです．

3) **ダイナミックリンキング**：Multics では，プログラムのリンクは完全にダイナミックにできる機構が考えられ，また，それを支えるハードウェア機構と連動させていました．

「完全にダイナミック」と言ったのは，究極の遅延も可能になっていたからです．つまり，あるプログラム，これをプログラム A とします，で，別のプログラム，プログラム B，を呼び出すようになっていたときに，プログラム A の起動時にプログラム B がリンクされるのではなく，プログラム B が必要となるときまでにあればよいのです．このことは，呼出しを記述しておくが，それが現実に呼び出されるなら，そのときになって初めてそのプログラムのコーディング，そのソースプログラムの作成，を始めてもいいじゃないか，という極限のサボりにつながっていきます．

そして，これは Emacs を開発環境のベースにおいて，ソースプログラムの開発とそのテストをインクリメンタルに進めていく開発スタイルとも結びついていきます．

親ガメこけると皆こける

「Multics は大変すばらしいシステムだった．けれども商業的には失敗だった．We turned out to be backing the wrong horse.」と Bernie Greenberg は言っています．彼はかなりくやしい思いをしたようにも感じられます．

「ハネウェルは Multics のようなシステムで動かすメインフレームコンピュータが死ぬとは思っていなかった．Multics Emacs は Multics のために作った．Multics Emacs のライフサイクルはみごとに Multics と同じ運命をたどった．特定のシステムのためにソフトウェアを作るというのはまずいね．なぜかってベースになるシステムが死ぬとその上のものは皆枯れてしまうから．その点，UNIX は長持ちしているね．」これは，Emacs のことを離れてソフトウェアシステムの開発の宿命とでも言いましょうか，そうしたことに対する示唆でもあると思っています．

Bernie Greenberg という人

Bernie の人となりについて紹介しましょう．一口に言って，あまり名を知られていない天才型の人だと言えます．そもそも Multics Emacs について書こうと思ったのは，彼のことがもっと紹介されるべきだと考えたからです．

彼はオルガニストとしても一流の腕をもっています．正直言って驚くべき腕前です．彼の家には本格的な電子オルガンが置いてあって，気が向くといろいろな曲を弾いてくれました．**写真**は，その前で撮ったものです．バッハ全集などもそらんじています．趣味に曲のアレンジなどもしています．演奏に熱中していても電話が鳴っているのに気がつくように，廃線になったところから信号機をもらってきて電話につけ，直径 20 cm もあるライトをピカピカさせて電話をとります．

写真　自宅のオルガンを演奏する Bernie Greenberg 氏

彼の愛すべきキャラクタについては，紹介したいエピソードがたくさんあります．一口で言うのは困難ですが，彼一流の博識と哲学と世界観に基づいた，熱心な宗教家でもあります．また，頭の回転が速く，時としてそれが早合点になって，損をしている部分もあるようにも見受けます．

TECO で書くのは美しくない

Bernie ががぜんがんばって MacLisp ですべてを書き上げた背景には，TECO はプログラミング言語として美しくないという意識がありました．TECO はまるで暗号の羅列のようだと表現しています．

それが Lisp を選択する動機になっていて，そして Richard が後に，機能拡張言語として Emacs Lisp を作り，それを使うきっかけにもなっています．

競い合いながら，相手の良くない点を改善するメカニズムを設計し，それをまた相手が見て，いいと思ったら取り込んでいく．そういういい関係があったように思います（もっとも，そう仲が良かったとも思えませんが）．

Multics に画面エディタを！

さて，本題に戻って，Multics Emacs に関して見るべき点を追いかけてみます．

当時の表現で言うと，「リアルタイムのビデオ指向エディタ」．これが，Bernie がほしかったものです．そして，多数の人がそういうものをほしがっていました．「リアルタイムの」というのは今では当り前のことなので，その表現にピンと来る人が少ないかもしれませんが，編集指示をすると，そのままそれが実行されるということです．

Bernie Greenberg は，Richard Stallman が作った ITS 上の TECO が画面モードで動作するのを見て，それを Multics でもできるようにしようと考えたのです．

Multics Emacs での標準のキーバインド

Multics Emacs での標準キーバインドはどんなものだったのでしょうか？　何を Emacs と呼ぶかと言うと，やはり，キーバインドがどうなっているか，どれだけ共通性があるかが鍵になってくるでしょう．見た目と操作感覚があまりに違えば，Emacs とは言い難いことになります．

Multics Emacs は，Stallman の Emacs とは独立して作成されましたが，基本的な概念は受け継いでいます．また，後の GNU Emacs などに取り入れられた機能などが組み込まれました．

Multics Emacs は普通の ASCII 端末で動作するように考えられました．したがって，Meta-なんとかはなくて，ESC を押して次に別の字を押すということをするようになっています．図に Multics Emacs の標準キーバインドのうち，主だったものを選び，現在の GNU Emacs でのバインドとの対比で示します．なお，Multics Emacs の仕様は文献 6)からとっていますが，内情をお話すると，その文献がどうしても手に入らないので，Bernie Greenberg に依頼して，電子メールで送ってもらいました．

まずは安心できることですが，ほとんどのキーの意味は同じであることがわかります．

いくつかの特徴

まず割込みですが，ITS では，C-z が interrrupt になっていました．システムそのものにそうした字に対応する機能があったのです．ところが，Multics ではブレーク信号がそれに対応します．つまり，ブレー

key	GNU Emacs binding	Multics Emacs binding
C-@	set-mark-command	set-or-pop-the-mark
C-a	beginning-of-line	go-to-beginning-of-line
C-b	backward-char	backward-char
C-d	delete-char	
C-e	end-of-line	go-to-end-of-line
C-f	forward-char	forward-char
C-g	keyboard-quit	command-quit
C-h	help-command	rubout-char
TAB	indent-for-tab-command	self-insert
LFD	newline-and-indent	noop
C-k	kill-line	kill-lines
C-l	recenter	redisplay-command
RET	newline	new-line
C-n	next-line	next-line-command
C-p	previous-line	prev-line-command
C-q	quoted-insert	quote-char
C-r	isearch-backward	reverse-string-search
C-s	isearch-forward	string-search
C-t	transpose-chars	twiddle-chars
C-u	universal-argument	multiplier
C-v	scroll-up	next-screen
C-w	kill-region	wipe-region
C-x	Control-X-prefix	prefix-char
C-y	yank	yank
C-z	iconify-or-deiconify-frame	(a different prefix)
C-_	undo	help-on-tap
C-x C-@	pop-global-mark	
C-z C-@		set-or-pop-the-mark
C-x C-b	list-buffers	list-buffers
C-x C-c	save-buffers-kill-emacs	quit-the-editor
C-x C-d	list-directory	
C-x C-e	compile	comout-command
C-x C-f	find-file	find-file
C-x TAB	indent-rigidly	
C-z C-f		get-filename (as text)
C-x C-k	mule-prefix	
C-x C-l	downcase-region	lower-case-region
C-x RET	set-mark-command	eval-multics-command-line
C-x C-n	set-goal-column	
C-x C-o	delete-blank-lines	delete-blank-lines
C-x C-p	mark-page	
C-x C-q	vc-toggle-read-only	
C-x C-r	find-file-read-only	read-file
C-x C-s	save-buffer	save-same-file
C-x C-t	transpose-lines	toggle-redisplay
C-x C-u	upcase-region	upper-case-region
C-x C-v	view-file	
C-z C-v		scroll-current-window
C-x C-w	write-file	write-file
C-x C-x	exchange-point-and-mark	exchange-point-and-mark
C-x C-z	iconify-or-deiconify-frame	
C-z C-z		signalquit
C-x ESC	Prefix Command	
C-x SPC	henkan-region	
C-x $	set-selective-display	
C-x '	expand-abbrev	
C-x (	start-kbd-macro	begin-macro-collection
C-x)	end-kbd-macro	end-macro-collection
C-x *		show-last-or-current-macro
C-x +	balance-windows	
C-x -	shrink-window-if-larger-than-buffer	
C-x .	set-fill-prefix	
C-x /	point-to-register-compatibility-binding	
C-x 0	delete-window	remove-window
C-x 1	delete-other-windows	expand-window-to-whole-screen
C-x 2	split-window-vertically	create-new-window-and-go-there
C-x 3	split-window-horizontally	create-new-window-and-stay-there
C-x 4	ctl-x-4-prefix	select-another-window
C-x 5	ctl-x-5-prefix	
C-x 6	2C-command	
C-x ;	set-comment-column	set-comment-column
C-z ;		kill-comment
C-x <	scroll-left	
C-x =	what-cursor-position	line-counter
C-x >	scroll-right	
C-x [	backward-page	
C-x]	forward-page	
C-x ^	enlarge-window	
C-x `	next-error	
C-x a	Prefix Command	
C-x b	switch-to-buffer	select-buffer
C-x d	dired	edit-dir
C-x e	call-last-kbd-macro	execute-last-editor-macro
C-x f	set-fill-column	set-fill-column
C-x g	insert-register-compatibility-binding	get-variable
C-z g		go-to-named-mark
C-x h	mark-whole-buffer	mark-whole-buffer
C-x i	insert-file	insert-file
C-x j	jump-to-register-compatibility-binding	
C-x k	kill-buffer	kill-buffer
C-x l	count-lines-page	
C-x m	mail	send-mail
C-x n	next-error	
C-x o	other-window	select-another-window
C-x q	kbd-macro-query	macro-query
C-x r	Prefix Command	rmail
C-x s	save-some-buffers	global-print-command
C-x u	advertised-undo	
C-x v	Prefix Command	view-lines
C-x w	dictserve-get-thesaurus	multi-word-search
C-x x	copy-to-register-compatibility-binding	put-variable
C-x z	dictserve-get-translation	
C-x {	shrink-window-horizontally	
C-x }	enlarge-window-horizontally	
C-x DEL	backward-kill-sentence	
C-x C-SPC	pop-global-mark	
ESC C-@	mark-sexp	
ESC C-a	beginning-of-defun	
ESC C-b	backward-sexp	balance-parens-backward
ESC C-c	exit-recursive-edit	
ESC C-d	down-list	
ESC C-e	end-of-defun	
ESC C-f	forward-sexp	balance-parens-forward
ESC TAB	complete-tag	indent-to-fill-prefix
ESC LFD	indent-new-comment-line	
ESC C-k	kill-sexp	
ESC C-o	split-line	split-line
ESC C-u	backward-up-list	
ESC C-v	scroll-other-window	page-other-window
ESC C-w	append-next-kill	merge-last-kills-with-next
ESC ESC	Prefix Command	eval-lisp-line
ESC C-\	indent-region	
ESC SPC	just-one-space	
ESC !	shell-command	
ESC # .. ESC $	ispell-word	
ESC #		rubout-word (# is multics rubout)
ESC DEL		rubout-word
ESC _		underline-word
C-z _		remove-underlining-from-word
ESC [		beginning-of-paragraph
ESC]		end-of-paragraph
ESC %	query-replace	query-replace
ESC '	abbrev-prefix-mark	
ESC (	insert-parentheses	
ESC)	move-past-close-and-reindent	
ESC ,	tags-loop-continue	
ESC -	negative-argument	
ESC .	find-tag	
ESC /	dabbrev-expand	regexp-search-command
ESC 0 .. ESC 9	digit-argument	
ESC :	eval-expression	
ESC ;	indent-for-comment	indent-for-comment
ESC <	beginning-of-buffer	go-to-beginning-of-buffer
ESC =	count-lines-region	
ESC >	end-of-buffer	go-to-end-of-buffer
ESC ?	apropos	describe-key
ESC @	mark-word	
ESC \	delete-horizontal-space	delete-white-sides
ESC ^	delete-indentation	
ESC `	tmm-menubar	
ESC a	backward-sentence	backward-sentence
ESC b	backward-word	backward-word
ESC c	capitalize-word	capitalize-initial-word
ESC d	kill-word	delete-word
ESC e	forward-sentence	forward-sentence
ESC f	forward-word	forward-word
ESC g	goto-line	go-to-line-number
ESC h	mark-paragraph	mark-paragraph
ESC i	tab-to-tab-stop	indent-relative
ESC j	indent-new-comment-line	
ESC k	kill-sentence	kill-to-end-of-sentence
ESC l	downcase-word	lower-case-word
ESC n		next-comment-line
ESC p		prev-comment-line
ESC q	fill-paragraph	runoff-fill-paragraph
ESC r	replace-string	move-to-screen-edge
ESC s		center-line
ESC t	transpose-words	
ESC u	upcase-word	upper-case-word
ESC v	scroll-down	prev-screen
ESC w	kill-ring-save	copy-region
ESC x	execute-extended-command	extended-command
ESC y	yank-pop	wipe-this-and-yank-previous
ESC z	zap-to-char	
ESC {	backward-paragraph	
ESC |	shell-command-on-region	
ESC }	forward-paragraph	
ESC ~	not-modified	
ESC DEL	backward-kill-word	rubout-word

図　Multics Emacs と GNU Emacs のキーバインドの対照表

クキーとかを押して回線断相当を引き起こさないといけません．文字では割込みになりませんでした．逆にこのことから，C-z が空きます．それで，それに別のキーコンビネーションの prefix としての役割を与えています．

C-h は，rubout でした．つまり，今のキーボードで言えば backspace キーの機能です．delete キー相当ではありません．これは，少なくとも GNU Emacs の標準より私の使用感覚には合います．オンライン help 機能は用意されていて，呼出しは control-underscore でした（しかし，詳細な資料がなく，その機能についてはわかりません）．

キーボードマクロ

Bernie とやりとりをしているうちに彼も自分でやったことの中で，その後に影響を与えた部分などが見えてきます．電子メールのやりとりをしている最中に，キーボードマクロについて，「そういえば，これは私が発明した機能なんだ．Richard Stallman がちょうどそれを見ていて，さっさと彼の Emacs にも入れてしまったんだ．」などと言い出して，それで，その裏付けを取り出したことがありました．どうも彼の言っているとおりのようです．

キーボードマクロというのは，現在あるコマンドから新しいコマンドを定義する方法です．一連のいくつかのコマンドを一つのコマンドとして実行させます．

いくつかのコマンドの実行をさせてみて，それを覚えさせ，次には，それらをいっぺんに一つのコマンドとして起動させるような仕組みは GNU Emacs 以前に作られました．Bernie はこの仕組みに macro-collection という名前をつけていて，表にもあるように，GNU Emacs の start-kbd-macro や end-kbd-macro になったのと同じキーバインドを与えています．

save-excursion

save-excursion は，たとえば，GNU Emacs Lisp Manual を見ると，

```
(save-excursion forms) =
  (let ((old-buf (current-buffer))
        (old-pnt (point-marker))
        (old-mark (copy-marker
                   (mark-marker))))
    (unwind-protect
        (progn forms)
      (set-buffer old-buf)
      (goto-char old-pnt)
      (set-marker (mark-marker)
                  old-mark)))
```

と定義されている機能で，カレントバッファ，ポインタ，マーカを覚えておいて，forms を実行し，その実行が終わると，かならず元のバッファ，ポインタ，マーカに戻す仕組みです．

これは大変便利な機構です．あるバッファで処理をしているときに別のバッファを開いて，そこで仕事をさせるようなことがあるとします．そうすると，それが終わったときに，元の場所に戻りたいと思うのが普通ですが，そのための処理をちゃんとしておかないと，特にポインタの位置，つまりカーソルの位置の値が変わってしまっていて，元のバッファに戻ったときに変なところを指してしまう，ということが生じやすいのです．それを防いでくれます．例で言えば，テキストを作っているときに辞書を引きたくなったとします．その機能は別のバッファに和英辞典を開いて単語を引くようになっていたとします．save-excursion で実現されているような機能を利用しないと，単語を引き終わった後で，もとのバッファに戻ると，ポインタの値は，和英辞典のバッファで使われたポインタの値に変わってしまっていて，元々指していたところではなく，カーソルが変な場所を指してしまうことがあり得るわけです．

この save-excursion マクロは，Multics Emacs で Bernie Greenberg が考案し，それが便利なものとして GNU Emacs などに受け継がれていっています．

M-x は evaluate だった

Emacs の特徴の一つとして，機能をその名称で呼んで起動できることがあります．M-x のあとに機能名を入れるのです．これは，今でも時々使います．だんだん年をとってきて，とっさにキーバインドを忘れることがあるからです．普通のエディタではこれでアウトで何もその先の処理ができませんが，Emacs では，キーバインドは単なる便宜です．名前をはっきりと覚えているかあるいは名前の一部などを知っていれ

ば探していって実行できます．

たとえば，Emacs では^d (control d) は1字抹消ですが，今，このキーを忘れたとします．それが delete-char という機能だと覚えていれば M-x delete-char と入力すればその機能が実行できます．さらに，いずれ述べる completion や help 機能を使えば一部分を覚えているだけでも到達できます．

Multics Emacs でもこの機能が当然できます．TECO をベースとした ITS Emacs では違いましたが，Multics Emacs 以降の Emacs では，呼び出す機能は Lisp で定義した関数/コマンドでした．このことは，前回の UNIX Emacs の話でも触れました．

そうなると，呼び出す仕組みが気になります．現在使われている GNU Emacs では，interactive という宣言機能があって，それを使っていろいろとできるのですが（これらについてはいずれ述べます），Multics Emacs あるいはそれ以前の ITS Emacs ではどうなっているのでしょうか？

「Multics Emacs では，M-x は evalquote と同じになっている」

これが，答えになります（ITS Emacs については第1回で触れました）．起動時に引数がいるような機能の場合，それらをどうやって渡すのでしょうか？そこに「Multics Emacs では，M-x は evalquote と同じになっている」ということの意味が現われてきます．つまり，起動時に実際に渡すパラメータを（もしあれば）機能名の横に並べて書きます．それは（eval 関数に対する引数のようではなく）すでに実体そのものを指しているので，Lisp 的な概念で evalquote のようだと言っているのです．

たとえば，iterate-region という機能があったとしましょう．回数を起動時に与えるようになっているとすると，

M-x iterate-region 30

などのようにその横に書くのです．そうすると，

(defun iterate-region (*n*) ...)

のような具合で定義されているはずなので，*n* に 30 が束縛されて実行されます．複数の引数がある場合はさらにその横に並べて書きます．前回の UNIX Emacs では，arg あるいは argv という機能で引数をもってきました．

のちに Gary Palter がコマンドプロセッサとして，define-command というマクロを作り，それによって，コマンドとして実行する機能の定義をするようになります．これは，のちに Richard Stallman が，「同じことをするのに，コマンドして起動される場合と，関数として起動される場合と別々に書かなければならないのはおかしい」という指摘をして GNU Emacs に interactive という宣言を導入するきっかけになります．

端末の制御を1文字単位で

Multics での端末は，半二重でした．ディスプレイ端末も登場するようになってきましたが，それらは，1行分あるいは1画面分全部！！ をまとめて送信する方式をとっていました．エディタのような応用にとって「とんでもない」こういう設計は，現在でもメインフレームにまだ残っています．極端な言い方をすると，1字の訂正でも1画面分送ることが生じるわけです．TSS でさまざまな処理を並行して進めているコンピュータの，メモリや処理時間がもったいないので，端末側にインテリジェンスを与えて，入力の編集などはコンピュータ側ではやらないで，ある程度端末側でやってしまい，OK となったら一度にすべてを送るようにする考え方です．そうすれば中央の負荷が減るということに基づいているようですが，これでは，ちょうど，実習の授業なのに先生が「じゃあ，勝手にやっといて」といって部屋からいなくなり，終わる頃に帰って来るような仕組みだといえます．つまり，テキストの編集というようなプロセスはどっか別のところでもできるような仕事だという考え方が背景にあったように思うのです．それだと，ソースプログラムの編集と実行とを少しずつ並行して切り替えながらやるようなことも難しいし，そもそも編集中のテキストがどこまで編集されているかは端末からコンピュータへ一括送信された後でないと把握されません．

半二重の端末はグラスタイプライタとしてか，あるいは local-edit モードを介して使われます．local-edit モードの場合，入力の終りに送信キーを押すのです．このローカル編集機能は編集という立場で見ると大変貧弱なものでした．

当然誰かが文字単位の会話を Multics でもさせようとします．Eugene Ciccurelli という人が Multics 上

で最初に多少気の効いた文字単位の編集の出来る行エディタを作った人だといわれています．

「端末側で行単位に（あるいは1画面分）編集しておいて，送信キーによりまとめて送信する」という方式と，「端末には知能はなく，1字1字すべてがその都度送られ処理される」という方式は，究極的に前者は半二重通信，後者は全二重通信とフィットします．後者では，入力のエコーもコンピュータ側で制御されます．つまり，入力した字の表示はキー入力がコンピュータに送られ，それが送り返されて表示されます．それで，パスワード入力のようなものに対して表示を抑制する仕組みなどもやりやすくなります．

Emacs のニーズだけでなく，全二重の，文字単位の処理の，リアルタイム画面編集機能への期待というか，そちらの優位性を認める声が高まってきます．

しかし，Multics（およびそれを支えているハードウェア）は，文字単位の全二重通信に向いた設計にはなっていませんでした．Multics だけでなく，当時のメインフレームの通信システムはほとんどそうでした．唯一 DEC は，そのミニコンピュータからの経験で，文字単位の処理をコンピュータ側でするようになっていたのです．

加えて，Multics の通信システムを複雑にさせたもう一つの要因は，FNP というフロントエンドプロセッサの存在でした．FNP がコンピュータと端末の間にあって，実際の端末とのやりとりを制御するのです．そして，コンピュータと FNP の間の通信は，少量の転送には不向きで，大量の一括転送に向いていました．FNP によるコンピュータへの割込み要求は高価なものだったからです．Multics でのプロセスの wake up も高価だということもありました．それらのことがあるので，1字ごとの転送をやる価値が認められても，現実どれだけ有効なものなのか疑問がありました．誰かがテストして，1文字単位の転送をしてみて，その有効性を認めてもらってその上で，効率を改善しないといけないということになります．

私事になりますが，ちょうどこの頃大学院生でしたが，ACOS-6 用のある言語処理系の開発を請け負って作っていました．ACOS-6 は Multics の概念に非常に近いものをもっていました，あるいはもとうとしていました．また，ハードウェアもたいへん似ていました．シールをはがすと，その下から別のアメリカの

会社の名前が出てくるマシンでテストしたこともありました．1年間ほとんど休みなしにマシンルームにこもっていて，ここでの議論と疑問を自分のテーマとしても扱ったことがあるのです．それで，よーくわかるのです．

1978年3月3日，ハネウェルのCISL（Cambridge Information Systems Lab.）でBernieらはITS Emacsのデモを企画します．文字単位の処理の画面エディタの優位性をMultics開発者たちに理解してもらうつもりだったと述べています．しかし，よくある話ですが，デモはアクシデントでそれほどうまくいきませんでした．多くの人には必ずしもその強力さを感じさせることができませんでした．

その中で，Larry Johnsonという通信の技術者が興味をもちます．そして，その日のうちに彼はFNPのプログラムを直し，1字1字をそのままコンピュータに渡すようにします．

これは，効率という点ではたいへん悪いプログラムでしたが，ともかく1文字単位の処理ができるようになります．Bernie Greenbergらはすぐにプログラムを書き始め，その日のうちに原形ができます．それで，Multics Emacsの誕生日は78年3月3日だと断定できるわけです．

Echo Negotiation

本来半二重向きにできていたFNP経由の応答を1文字単位に変更するというのは，システムの側からすればEmacsは大変非効率なアプリケーションということになります．半二重の場合には，FNPでエコーをして，効率を稼いでいたのが，Emacsの実行ではその機能がオフになって，すべてをコンピュータ側で処理するからです．それで，Bernie GreenbergはEcho Negotiationということを次に考えます．これは，ある程度の範囲の機能ではFNPでローカルエコーさせ，CPUの負荷を下げるというもののようです．詳細は，文献7)に記述されていますが，筆者は確認していません．

これが，およそのMultics Emacsのストーリーです．

参考文献

1) Ritchie, D.M., The evolution of the UNIX time-sharing system, In *Language Design and Programming Methodology*, Jeffery M. Tobias (Ed.), Springer-Verlag, New York, 1980.
（邦訳：**bit**，1983年7月号.）
2) E.I. Organick : *The Multics System*, MIT, 1972.
（邦訳：『Multics システム（上，下）』，共立出版，1974.）
3) Corbato, F.J. : PL/I as a Tool for System Programming, *Datamation*, pp. 68, 73-76, May 1969.
4) 井田昌之，中田育男：基本ソフトウェアの記述ツール，情報処理，Vol. 20, No. 6, pp. 519-526, 1979.
5) Bernie Greenberg,
http://www.basistech.com/bsg/mepap.htm
Multics Emacs: The History, Design and Implementation.
6) Emacs Text Editor User's Guide, Order # 27, HIS, Dec. 1979.
7) Bernie Greenberg, The Echo Negotiation, Multics Technical Bullet 418, Honeywell, July, 1979.

（いだ まさゆき　青山学院大学 国際政治経済学部）
[ida@sipeb.aoyama.ac.jp]