流れて行かないＵｎｉｘ環境の評価

電気通信大学　角田　博保、東京工業大学　久野　靖
Hiroyasu Kakuda、　Yasushi Kuno

1. はじめに

Unix の X ウインドウ上に「流れて行かない Unix 環境」を構築し、従来のウインドウを使わない Unix 環境と、いわゆる普通のウインドウを使った Unix 環境との客観的比較評価をおこなった。

Unix オペレーティングシステムはプログラミング環境として優れたものであるが、そのユーザーインタフェースは依然としてタイプライタ時代のイメージを引き継いだ「たれ流し」の端末にすぎない。この傾向はいわゆるウインドウシステムにおいても同様で、基本的に「一つの窓が、一つの端末に相当する」世界のままである。ウインドウの利点をいかすためには、マッキントッシュのようにすべてのツールをウインドウ環境にあわせて再構成することも考えられるが、すでに Unix 上には膨大なソフトウェアの蓄積があり、それらをすべて作り直すことは実際的ではない。そこで、ウインドウを利用するためのできるだけ少数のツールを開発し、それを既存のプログラム群と併用するという方針で、ウインドウ向きのプログラミング環境を構築することにした。従来の Unix コマンドでは「流れて行ってしまう」出力を画面単位で堰止めるために「more」コマンドを使っているが、ここで開発したツール群は「more」コマンドを必要としない、流れて行かない Unix 環境を提供するものである（「no more unix」から NMU と略す）[1]。

このようにして開発した NMU であるが、その利点、欠点について客観的な評価をし、それをもとにしてさらに改良をおこなうことが大切に思える。そこで、まず、X ウインドウの X サーバに手を加えて時間データを計測できるようにし、端末環境、普通のウインドウ環境、および NMU 環境において 3 種の実験をおこなって時間データを収集した。あわせて、ビデオカメラを使って実験状況を記録した。この時間データとビデオ記録にもとづいて評価をおこなうことにした。

実験としては、Unix を使ってプログラム開発をするといった作業から典型的なものを選んだ。採取データは細部にわたって詳細に検討し、いくつかの補助実験を追加して総合的な評価をえた。

2. NMU のまとめ

ある特定の作業に役に立つ特別なツールを多量に作るのではなく、一般的に使えるツールを少量作るというのが、NMU の設計思想である。現在のところ、xcl、xcom、xe、xls、xman、xt、xvr、xw、xx の 9 個である。（詳しくは文献 [1] を参照されたい。）

NMU ツールは基本的に端末窓でそのコマンド名を打ち込むと、それに対応した窓が生成される。窓は画面上の適当に開いたところにとられる。NMU ツールの窓はマッキントッシュの窓に似せて raise ボックス、lower ボックス、iconify ボックス、resize ボックス、タイトルバーおよびスクロールバーを持っている。タイトルバーをドラッグすれば窓の移動ができる。

実験で扱った xe、xls、xman、xx について少し詳しく述べることにする。

(1) xe

画面エディタである。窓にはテキストポインタが表示されている。マウスで場所を指定してクリックすればテキストポインタはそこに移動する。ドラッグによって適当な範囲のテキストを選択することができる。選択されたテキスト部分はプルダウンメニューでの指令によってカット、コピー、ペーストができる。サーチボックスを開いて内容探索もできる。xe は more のかわりに使うことになる。

(2) xls

指定されたディレクトリのエントリ名が窓に表示される。このエントリ名をマウスで指定してダブルクリックすると、エントリがディレクトリであればエントリ名表示窓が、ファイルであればファイル編集窓（つまり xe）が開かれる。

(3) xman

オンラインマニュアル専用窓を開く。最初は、xls /usr/man で間に合わせていたが、頻繁に使うものであるから、やはり専用のツールとした。スクロールバーで探してもよいし、サーチボックスを開いて名前を打ち込んで探すこともできる。

(4) xx

「xx　コマンド列」によってそのコマンドを実行し、同時に新たな窓が作られ、結果の出力が窓に表示される。その表示がファイル名と行番号を含んでいる場合（cc のエラー出力など）はその行をダブルクリッ

クすると対応するファイルの編集窓(つまりxe)が開かれ、対応する位置にテキストポインタが行く。コンパイル後のエラー修正に便利である。メニュー選択で再実行ができる。

3. 計測機構

Xウインドウのxサーバ内を修正して、Xサーバがクライアントに渡すイベントをファイルに記録できるようにした。クライアントが要求したイベントだけが記録されることになる。この実験で記録したイベントの一覧を表1に示す。

1	keypressed	キー押し下げ
4	buttonpressed	マウスボタン押し下げ
8	buttonreleased	マウスボタン離し
10	enterwindow	ウインドウに入る
20	leavewindow	ウインドウから出る
40	mousemoved	ウインドウ内でマウスカーソルの移動
80	exposewindow	ウインドウ内の変更
100	exposeregion	ウインドウ内のリージョンの変更
200	exposecopy	リージョンの表示変更

表1: イベントの一覧

4. 評価方法

Cardらの開発した計算機システム利用者の作業時間を予測するためのモデルである打鍵レベルモデル[2]でとられている技法にしたがって、利用者の作業を細かな基本的な操作の列としてあらわすことにする。

ここでは以下の4分類からなる操作を考えた。

(1) 手の動作

打鍵(K)、手をマウスからキーボードへ(HK)、手をキーボードからマウスへ(HM)、マウスボタンクリック(MB)、マウスカーソルの移動(MC)。MB(expand)のように書いて窓の拡大操作をあらわす。また、MC(expand)と書いて拡大操作のための位置へのマウスカーソルの移動をあらわす。MC(T)としてある目標(target)にむかうマウスカーソル移動を、MC(WB)として窓の境目までの移動、MC(W)として窓の中までの移動(位置を気にしないカーソル移動)をあらわす。

(2) 知覚と思考をあわせた作業(MP)

MP(s)で画面をみながらの探索(search)作業を、MP(read)で画面の判読作業をあらわす。

(3) 思考作業(M)

いくつかの思考作業を区別するときは、M1、M2のようにあらわす。

(4) 計算機の反応(R)

CPUビジー(R(b))、画面表示(R(s))など。

以下の実験でおこなう作業をこれらの操作の列に分解して、個々の操作がどの位の時間かかるかを割り出すことにする。また、操作のいくつかの列で一つの作業単位をあらわすものをマクロ操作と呼ぶ。実験対象となるツールごとにこれらのマクロ操作にかかる時間がどう違うかをみていくことにする。

5. 実験1

5.1 実験と結果

最初の実験はプログラム開発の前段階でよくおこなわれる情報の探索を扱うことにした。例としてXウインドウのソースファイル群から特定の機能を持つ関数の定義箇所を探すといった作業をとりあげる。

Xウインドウのソースプログラムが含まれたディレクトリを現在ディレクトリとして、探索をはじめ、イベントがどこからとれてくるのかをみつけるまでの被験者の動作を記録した。被験者はXウインドウの内部構成について、ある程度の基礎知識はあって、ディレクトリの一覧やプログラムの中をみるだけで何を探索すればよいかわかることにした。

具体的には以下の通りである。

(1) 現在ディレクトリ内のエントリ名の表示。調べるエントリがXlibとわかる。

(2) Xlibを現在ディレクトリとして、その中のエントリ名の表示。たくさんあるので、端末窓のlsでは流れて行ってしまう。ls -Clmoreと打ち直す。表示を目で追ってそれらしいファイル名を探す。ない場合は次の表示へ進める。XNextEvent.cだとわかる。

(3) XNextEvent.c内を探す。目で追って関数XReadだとわかる。

(4) XReadに近いファイル名を探す。みつからない。

(5) egrepによって関数XReadを定義している箇所を探す。Xlibinternal.cだとわかる。

(6) Xlibinternal.c内をさがして、関数XReadの定義箇所をみつける。

[実験対象]

端末窓を使った従来のUnix環境での操作とNMUツールを使った操作。

[被験者] HK、YK。

[実験方法]

実験対象と被験者の4通りの組み合わせで以下の順で実験する。

端末窓-YK、NMU-HK、NMU-YK、端末窓-HK

それぞれの組み合わせで3回連続して試行を繰り返す。実験に先立って各被験者とも両方の方式について1回づつ練習した。

[結果]

端末窓とNMUの場合をそれぞれ操作単位に分解してマクロ操作に区切ってまとめたものを表2に示す。*印は打鍵誤りがスムースに動作しなかった場合を示す。被験者それぞれの各マクロ操作単位で一番速いものを線で囲んで示す。線で囲んだ値を合計したものを選択合計として示してある。たとえば、HKのNMUの場合は一番速かったのはHK2 (2回目)で、69.46秒であるが、選択合計は63.6秒となる。つまり、もっともうまく作業ができた場合を推定したいわけである。この選択合計の値をみれば、HKのNMU (63.6)、YKの端末窓(63.88)、HKの端末窓(64.3)、YKのNMU (66.12)という順序になる。

しかし、画面をみてどのファイルかを探す作業では何度も同じ条件で繰り返すことはできない。つまりどのファイルかがあらかじめわかってしまうので、はじめて探すようにはできず、つい流し見てしまうのである。表2のファイル名探索の対応する時間をみると、はじめて探したという状況が再現されているとは思えない短さである。そこで、この探索だけを別立てで再度実験することにした。

5.1追実験

被験者HKのみがおこなった。上の実験でのXlib内を探索する箇所だけをとりだして、4つの方法でそれぞれ3回測定した。端末窓の場合は、ls lmoreとls -Clmore、NMUの場合は行スクロールとジャンプスクロールである。表示の各行を順に必ずみて、探すように心掛けた。

端末窓の場合はスペースキー打鍵で次の表示へと移る。lsだと22行、ls -Cだと5段で表示されていたので110行表示されていた。行スクロールは実験1で使った方法だが、スクロールバーをマウスで指定し、マウスボタンを押し下げている間中表示がスクロールする。しかし、目で追いかけるには速すぎた。ジャンプスクロールバーではクリックすると次の表示へ移る。23行表示されていた。

探索すべき文字列はXNextEventであり、ちょうど190行目に位置していた。結果を表3に示す。3回の平均値と標準偏差を示した(単位はミリ秒)。線で囲んだ部分は、画面の表示をみて、さがして、みつけて、次の行動をとろうとするところまでをあらわしている。その部分の小計をとって示してある。

操作MP(s)が画面をみながら探索する作業に対応している。各方法の一番最後のMP(s)は次の動作とのかからみがあってそれまでのものとは異なるので、それを除いたMP(s)の合計を計算したい。しかし、えられたデータはMP(s),M1というように、「さがしおわってから次の動作(マウスのクリックかスペースバーの打鍵)をおこすまでの」思考操作M1が不可分で測定されている。この値はたとえばマウスでポイントしてからマウスボタンを押すまでの時間とほぼ同じで考えられる。別の実験によってその時間は320 msであったので、このデータを仮にM1の値として使うことにする。するとMP(s)の合計と、探索の間にみた画面の行数、および、それで割った値は以下のようになる。

方法	ls	ls-C	JS	LS	
時間	15352	19134	13468	9580	
行数	176	220	161	120	(144)
1行	87.2	87.0	83.7	79.8	(66.5)

行スクロールの場合はスクロールされた行数は144行であったが、実際に画面が静止して被験者がみたであろう行数は120行である。つまり、差額の24行はみすてられてしまったと思える。そこで、120行の方で割り算をした。こうしてみてみると、人が目で探索する時間は方式によらず同一だと仮定してもそうひどいことではないように考えられる。大体1行当たり87 msで読み進んでおり、NMUでは少しとばし見をしたのであろう。lslmoreの87.2msに合わせて補正をしてみると、全体の時間は、順に25.8秒、27.0秒、23.0秒、25.1秒となった。

ジャンプスクロールが速いのは画面の表示の変更が瞬時におこるからである。それは、xlsツールが表示すべき全データを内部バッファに持っていることによる。その分xls起動時のR(b)がlsの場合に比べて大きくなっている。ls-Cが遅かったのは5段で表示するの

表2：実験1の結果

実験1： 端末窓

	HK1	HK2	HK3	YK1	YK2	YK3	平均値	標準偏差
―― ディレクトリ名探索								
K(ls~M)	400	240	280	160	220	100	233	94
R(b)	2120	2440	2120	2040	2080	2140	2157	131
R(s)	760	760	740	760	740	700	743	21
MP(s),M1	1580	1380	1300	2200	1940	1260	1610	349
小計	4860	4820	4440	5160	4980	4200	4743	326
―――― ファイル名探索								
K(cd Xlib~M)	1900	1240	1360	1960	1460	1840	1627	283
M1	500	980	500	1100	580	440	683	258
K(ls~M)	140	160	160	100	100	160	137	27
R(b)	2300	2200	2460	2260	2300	2440	2327	94
R(s)	3460	3680	3720	3840	3800	3560	3677	132
M	800	1100	920	760	3940*	1700	1056	343
K(ls -C|more~M)	7200	5680	2440	2740	2260	2560	3813	1914
R(b)	2460	2360	2640	2640	2380	2580	2510	116
R(s)	700	660	700	680	720	680	690	19
―― 文字列探索								
MP(s),M総計	3200	2800	2060	3820	3880	2620	3063	649
K( ),R(b)総計	240	260	240	240	240	260	247	9
R(s)総計	1340	1340	1360	1360	1340	1340	1347	9
MP(s),M	2400	1780	2020	1540	3160	940	1973	693
――								
K(q),M	2160	900	760	960	720	900	1067	496
小計	28800	25140	21340	24000	26880	22020	24697	2605
―――― ファイル内探索								
K(more XNextEvent.c~M)	5140*	4600	3660	3040	3340	3400	3608	534
R(b)	420	380	380	420	380	360	390	22
R(s)	500	520	500	500	520	520	510	10
―― 文字列探索								
MP(s),M総計	3340	3660	3440	1840	2560	4480	3220	835
K( ),R(b)総計	840*	140	160	140	160	160	152	10
R(s)総計	980	1060	1040	1080	1060	1020	1040	33
MP(s),M	2420	3080	1700	3720	2000	2240	2527	681
小計	13640	13440	10880	10740	10020	12180	11817	1376
―――― ファイル名探索（継続）								
K(ls *XRead*~M)	3920	3580	2620	4380	3160	2860	3420	608
R(b)	460	500	460	520	460	440	473	27
M	2560	1440	1360	3140	1780	3880	2360	926
小計	6940	5520	4440	8040	5400	7180	6253	1230
―――― egrep								
K(egrep -n XRead *.c~M)	8580*	6280	5060	6940	7660	4020	5992	1305
R(b)	1200	1180	1660	1180	1220	1180	1270	175
R(s)	7100	7200	6800	7120	6900	6840	6993	153
MP(s),M	2740	6160*	2140	5280	3240	3640	3408	1062
小計	19620	20820	15660	20520	19020	15680	18553	2121
―――― ファイル内探索								
K(more Xlibinternal.c~M)								
	3400	3220	5580	4960	3340	2860	3893	1004
R(b)	440	440	440	440	460	420	440	12
R(s)	540	540	520	540	540	540	537	7
M	1140	2280	920	660	340	740	1013	617
K(/XRead~M)	1420	1540	1760	1480	2160	1360	1620	272
R(b)	100	100	100	100	120	100	103	7
R(s)	500	500	520	520	520	540	517	14
小計	7540	8620	9840	8700	7480	6560	8123	1059
合計	81400	78360	66600	77160	73780	67820	74.19	5.42
選択合計	64300			63880				

実験1： NMU

	HK1	HK2	HK3	YK1	YK2	YK3
―― ディレクトリ名探索						
K(xls $cwd~M)	3320*	2700*	1660	1660	1720	2000
R(b) | HM	2060	1700	2000	1620	1780	1700
R(s)	740	620	640	620	620	620
MP(s),MC(T),M1	2660	2380	2400	6240*	3700	3420
小計	8780	7400	6700	10140	7820	7740
―――― ファイル名探索						
MB(d.c.open='Xlib')	660	320	240	340	280	280
R(b)	7520	7520	7240	7900	7480	7580
R(s)	160	180	200	140	260	220
―― 文字列探索						
MP(s),MC(SU)	2860	2100	1720	4120	2440	2000
MB(SU)総計	5640	6140	6120	4410	4900	4800
MP(s)総計	3520	1480	1120	3820	3040	4640
MP(s),MC(T),M1	2220	1540	2140	2640	2000	2340
小計	22580	19280	18780	23660	20400	21860
―――― ファイル内探索						
MB(d.c.open='XNextEvent')						
	300	300	1260*	260	360	320
R(b)	160	140	280	160	120	120
R(s)	320	320	300	300	300	320
―― 文字列探索						
MP(s),MC(SU)	2380	2580	2320	2080	1860	2240
MB(SU)総計	1160	1440	1480	1320	1040	1100
MP(s)総計	600	880	680	1120	660	1700
MP(s),MC(WB)	1800	2940	1720	1940	2300	2000
MC(expose),M1	1640	1520	1320	4660*	940	620
小計	8360	10120	9360	11840	7580	8420
―――― ファイル名探索（継続）						
MB(expose)	20	20	20	20	20	20
R(s)	60	80	60	80	60	60
―― 文字列探索						
MP(s),MC(SU)	2020	1660	2120	3820*	2520	2640
MB(SU)総計	1500	1580	1480	1320	1340	1520
MP(s)総計	540	100	680	2960	640	1680
MP(s),MC(WB)	3140	2180	3540	3420	3280	2160
MC(W),HK,M	1240	1620	1840	940	620	1800
小計	8520	7240	9740	12560	8480	9880
―――― egrep						
K(xx egrep -n XRead Xlib/*.c~M)						
	7920	7020	8960	7940	8420	5360
R(b)|HM	2640	2480	2320	2280	2300	2300
R(s)	660	680	520	640	460	600
M,MC(expand)	1920	2760	2720	1520	2400	1560
MB(expand)	1020	1020	1460	1540	1440	1600
R(b)	60	260	120	160	20	40
R(s)	1080	1140	1520	1540	1160	1640
MP(s),MC(T),M1	7400	5980	6780	9740	8440	7580
小計	22700	21340	24400	25360	24640	20680
―――― ファイル内探索						
MB(d.c.open=Xlibinternal.c)						
	260	220	220	320	240	320
R(b)	200	3040*	120	140	200	180
R(s)	820	820	840	960	800	860
小計	1280	4080	1180	1420	1240	1360
合計	72220	69460	70160	84980	70160	69940
選択合計	63600			66120		

でその分大目に表示をしてしまい、被験者が見なければならない行数も多くなる(190行に対して271行)からである。

以上まとめると、実際は実験1の端末窓での探索には27秒、NMUでの探索には25.1秒かかるものと考えられる。

5.3考察

表2に戻って順に検討することにしよう。

(1)ディレクトリ名探索

端末窓の方が速い。NMUではMC(T)によって目標までマウスカーソルを移動する操作だけ遅くなっているわけである。ただし、この操作は次のファイル名の指定の前準備となっている。

(2)ファイル名探索

文字列探索の部分は端末窓HK、端末窓YK、NMU HK、NMU YKでそれぞれ5680、5160、11100、12380(ms)である。追実験によれば、端末窓HKとNMU HKではNMUの方が1.9秒早くなっているので、この実験1でのデータは正しく探索作業をしたとはいえないことが明白である。YKの場合は追実験をしなかったが、実験1でのHKとYKの比較から同じデータをYKの場合にも使ってもそう誤りとはならない。こうして、追実験のデータで補正すると、ファイル名探索の時間は、順に31880、32120、42660、43860(ms)となる。

端末窓の方が圧倒的に遅いが、その理由の一つは一度lsを実行したら流れていったのでもう一度やりなおしたことにある。この部分はHKとYKでそれぞれ6840、6600(ms)である。もし、十分注意深くてlsが流れていくことを前以て分かっていたとしたなら、この部分の値を引いて、HKとYKでそれぞれ35820、37260(ms)ということになる。これでもまだNMUが優位(4～5秒)に立っているのはxlsでディレクトリ名をダブルクリックするとcdとlsを連続して実行したことになるので、端末窓でその分打鍵する時間(4秒)がかからないことにある。そのかわり、ディレクトリ名探索の最後でMC(T)によりマウスカーソルの移動という手間がかかっている。しかし、探しているものをカーソルで指すのであるから、人間にとって直接的な操作と考えられる。大きくみつもってもMC(T)は1秒ちょっとであるから、打鍵にして5文字くらい。十分にひきあうと思える。

また、端末窓の場合は、moreの終了用にqを打ち込む必要がある。その点NMUでは次の動作へと移ればよい。

(3)ファイル内探索

この場合の文字列探索の時間は両者ともほぼ同じである。(2)はファイル名という短い文字列の表示であったから、行スクロールは速すぎたが、この場合表示されるプログラムの　1行は長いので行スクロールの速度が遅くなり、十分目で追いかけることができた。この場合もNMUが端末窓より少し速いのはmore XNextEvent.cと打鍵するかわりに、ダブルクリックをすればよいからである。とくにXNextEventのような名前を画面の別の場所の表示をみながら打ち込むのは大変である。そこに表示があるのだから、それをみたいといえるNMUは自然である。

(4)ファイル名探索(継続)

XRead　と同じかそれらしいファイル名があるかどうかを探す作業であるが、このように名前がわかっている場合はその名前を教えて探す方が直接的である。端末窓の場合はls　*XRead*と打ち込むだけでよい。NMUでも同様なことはできるが、その場合は手をマウスからキーボードへ持っていき・・・といったことになり、めんどうなので(2)の継続で探してしまう。この場合はxlsの表示がソートされているので、XRがみえるまで、スクロールで高速にとばしみることができるが、それでも、端末窓の場合よりは長い時間がかかった。

(5) egrep

egrepとはファイルの中を調べて、与えられた文字列があればその場所を表示するUnixのコマンドである。　結局、XReadというファイルがないので、XReadを定義している場所を探すためにegrepを使うことになった。NMUでも同様にegrepをxxツールで使うが、手をキーボードに持っていくために、620 ms (YKの場合)くらい余分にかかっている。egrep打鍵の直後のR(b)をNMUと端末窓で比べると、820～1120ほどNMUが遅い。これは、NMUが内部バッファにegrepの出力をためる時間だけ、余分にかかっているのである。端末窓の場合はR(s)操作がゆっくりおこるので実はMP(s)操作が同時に平行しておこっているはずである。したがって、端末窓でのR(s)の次のMP(s)はNMUの場合と比べてかなり小さくなっている。

NMUの場合は、xxコマンドで作られる窓はいまのところ大きさ固定で、egrepの出力には小さすぎるのでexpand(窓拡大)が必要になった。この手間が3～4秒になっている。

表３：文字列の視覚による探索

端末窓
(1) ls | more の場合 (22行)

K(ls|more^M)	1807
R(b)	2120
R(s)	591
MP(s),M1	2239| ８回
K,R(b)	57|
R(s)	591|
MP(s),M2	2800
小計	25800

(2) ls -C |more の場合 (22行*5段)

K(ls -C|more^M)	2700
R(b)	2193
R(s)	922
MP(s),M1	9887| ２回
K,R(b)	130|
R(s)	922|
MP(s),M2	5133
小計	27000

NMU
(1) Jump-scroll の場合 (23行)

MC(d.c.open)	453
R(b)	4500
R(s)	133
MC(to JS),MP(s),M1	4140
MB(JS),MP(s),M1	2244| ７回
MB(JS),MP(s),M1,MC(to T)	2500
小計	22300

(2) Smooth-scroll の場合

MC(d.c.open)	313
R(b)	4560
R(s)	120
MC(to SU),MP(s),M1	2980
MB(SU)	703| ５回
MP(s),M1	2236|
MB(SU)	703
MP(s),M1,MC(to T)	2820
小計	21200

```
(1) 挿入          pcnt pcnt, LMARGIN+1, heading, DMARGIN+1, datestr, PMARGIN+1, pcnt);
                  pcnt, pcnt, LMARGIN+1, heading, DMARGIN+1, datestr, PMARGIN+1, pcnt);
(2) 削除                ints    ishift, oshift;
                int     ishift, oshift;
(3) 挿入                (c) {
                        switch (c) {
(4) 置き換え    char    *satrt[10];
                char    *start[10];
(5) 置き換え                    for(i=0; fl=LMARGIN+1; i< curfield; i++, fl+=fieldlen){
                                for(i=0, fl=LMARGIN+1; i< curfield; i++, fl+=fieldlen){
(6) 行削除              XXXXX;
(7) 削除                iff(curfield == endfield) curfield--; else *(p++) = 'Y0';
                        if(curfield == endfield) curfield--; else *(p++) = 'Y0';
(8) 行挿入                              }
(9) 削除                                        }}
                                                }
(10) 挿入               fieldlen, fieldno, heading)
                        fieldlen, fieldno, heading);
```

図１：実験２での修正データ

表４：　実験２の測定結果

NMU	HK	YK	端末窓	HK	YK	Emacs
コンパイル・エディタ起動						
cc	15800	12540	cc	6040	7440	ed&cc 21640
			retry	9320	9900	
ed1	740	720	ed	2920	3360	
ed2	2500	1820				
ed3	3440	3680				
修正						
	60720	51940		48660	61720	50040 9780*
再コンパイル・修正						
check	5740	3800	check0	1960	3500	check1 1380
			check1	1220	2760	check2 1040
						check3 1040
svrd	3840	3180	svrd	3600	4540	svrd 6500
run	7440	6920	run	5600	5960	run 7920
ed	4040	2760	ed0	2320	3120	ed 4900
			ed1	4360	8000	
svrd	5440	4920	svrd	2700	3420	svrd 4900
合計	109700	92280		88700	113720	109140

表５：編集作業の時間計測

NMU	HK	YK	端末窓（vi）	HK	YK	Emacs	YK
(1-MB.DC)	----	----	(1-line)	760	1307	(1-line)	1740
(1-point)	2699	1887	(1-mental)	1240	1713	(1-mental)	1640
(1-keyin)	1833	1580	(1-command)	1770	2960	(1-command)	2160
小計	4532	3467	小計	3770	5980	小計	5540
(2-MB.DC)	2446	2293	(2-line)	1773	1840	(2-line)	1740
(2-point)	2306	1773	(2-mental)	1320	1273	(2-mental)	1180
(2-keyin)	673	973	(2-command)	660	820	(2-command)	1320
小計	5425	5039	小計	3753	3933	小計	4240
(3-MB.DC)	3020	3747	(3-line)	3060	5707	(3-line)	1020
(3-point)	3980	2153	(3-mental)	1146	1220	(3-mental)	1300
(3-keyin)	2013	2433	(3-command)	2013	2730	(3-command)	2680
小計	9013	8333	小計	6220	9657	小計	5000
(4-MB.DC)	2813	3493	(4-line)	3233	4086	(4-line)	1840
(4-point)	3946	3033	(4-mental)	1620	2127	(4-mental)	4280
(4-keyin)	2220	1913	(4-command)	5663	7480	(4-command)	6240
小計	8979	8439	小計	10516	13693	小計	12360
(5-MB.DC)	2819	2490	(5-line)	2940	4787	(5-line)	2060
(5-point)	4140	3060	(5-mental)	1720	1360	(5-mental)	1420
(5-keyin)	1586	1440	(5-command)	1546	1713	(5-command)	3500
小計	8545	6990	小計	6206	7860	小計	6980
(6-MB.DC)	2466	2420	(6-line)	2033	4026	(6-line)	1100
(6-select)	3826	2794	(6-mental)	1040	1440	(6-mental)	1160
(6-keyin)	880	713	(6-command)	40	100	(6-command)	300
小計	7172	5927	小計	3113	5566	小計	2560
(7-MB.DC)	2246	2487	(7-line)	2080	4420	(7-line)	1280
(7-point)	3200	2753	(7-mental)	1580	1400	(7-mental)	1260
(7-keyin)	820	1013	(7-command)	773	1020	(7-command)	1400
小計	6266	6253	小計	4433	6840	小計	3940
(8-MB.DC)	2706	2140	(8-line)	2013	4907	(8-line)	1060
(8-point)	3813	2427	(8-mental)	900	1833	(8-mental)	1300
(8-keyin)	5773	2867	(8-command)	6813	5793	(8-command)	6200
小計	12292	7434	小計	9726	12533	小計	8560
(9-MB.DC)	3413	4934	(9-line)	2926	7447	(9-line)	1660
(9-point)	3152	1973	(9-mental)	1240	2073	(9-mental)	1520
(9-keyin)	733	847	(9-command)	1800	1953	(9-command)	5020
小計	7298	7754	小計	5966	11473	小計	8200
合計	69522	59636	合計	53706	77535	合計	57380

表６：実験２の各操作列ごとの平均と標準偏差

	NMU		X（端末窓vi）		Emacs
	HK	YK	HK	YK	YK
(1)	2741 (344)	3000 (911)	2313 (752)	4280 (1752)	1500 (365)
(2)	3404 (626)	2382 (482)	1311 (261)	1604 (323)	1673 (933)
(3)	1836 (1500)	1531 (704)	2342 (2185)	2729 (2288)	3202 (2059)

表７：マウス操作の時間データ

操作列	最短	最長	平均	標準偏差	回数
HM,MC	940	5280	2528	944	64
HM,MC(expand)	1860	3760	2690	570	6
MC(T)	1220	5140	2593	928	60
MC(expand)	1480	2160	1843	267	6
MC(move)	600	2320	1317	567	6
MC(redo)	1240	3420	1782	577	12
MC(save)	300	1240	604	272	11
HK	620	2540	977	352	51
MB(SU)	440	660	528	83	5
MB(expand)	920	2460	1467	377	12
MB(move)	1080	1860	1417	307	6
MB(open)	160	600	321	85	58
MB(point)	140	320	200	62	54
MB(redo)	640	1420	930	218	12
MB(save)	680	1240	864	175	11
MB(select)	940	1980	1300	389	6

(6)ファイル内探索

NMUの場合はegrepの出力表示からXReadの定義行をダブルクリックすれば、独りでにそのファイルが開いて、その場所にポインタが行くので、かかる時間は圧倒的に短い。

NMUでは目標を探し、その内容をまた探すといった作業に非常にうまく適合できる。端末窓と比較して劣った作業は、expandだけであった。(2)での補正をすると、端末窓HK、YK、NMUのHK、YKはそれぞれ、85620、85720、76700、78840となる。端末窓での流れていったlsの実行を除いても、78780、79120となり、NMUの方が速く作業できた。

この実験は一つの典型例を示したものである。同様なディレクトリ内を探して行く処理においてもNMUが端末窓よりずいぶん優位となると推測できる。

6. 実験2

実験2では複数の窓を使った作業を調べる。プログラミング作業でよくおこるプログラムの構文誤りの修正作業である。ある完成したプログラム(177行)にランダムに虫を入れた。文字の挿入、削除それぞれ3箇所、文字の置き換え2箇所、行の挿入削除それぞれ1箇所の計10箇所である。図1に順に修正すべきデータを示す。

作業はこのプログラム(v1.c)をコンパイル(cc)し、エラーメッセージをみながら修正し、虫取りをするというものである。1回の再コンパイルではまだ虫が残り、再度修正して完成という筋になっている。(つまり、too many errorsで最初のコンパイル時にすべてのエラーがみつからないようにしてある。)

[実験対象]

(1) NMUを使う。(2) X Windowを使って、あらかじめ窓を2つ用意しておき、コンパイルと編集とで窓を使い分ける。エディタにはviを使う。(3)窓は1つでEmacsを使う。

[被験者] HK、YK。ただし、EmacsはYKのみ。

[実験方法]

X-YK、NMU-HK、NMU-YK、X-HK、Emacs-YKの順で実験した。それぞれの組み合わせで3回連続して試行を繰り返す。実験に先立って各被験者とも1回づつ練習した。

[結果]

実験1と同様に操作単位での分析をおこなった。たとえば、修正作業では、次のエラーメッセージをみて修正する箇所をみつけカーソルを移動して修正打鍵をするといった操作の列をマクロ操作として取り出し、各人の3回の試行の中からマクロ操作単位でもっとも速いものの合計としてまとめた。表4により大きな作業単位でまとめて示す。コンパイル・エディタ起動作業のうちの、NMUの場合のccという作業は、xx cc v1.c の打鍵と窓の拡大である。つづく、ed1 でエラーメッセージ表示をダブルクリックし(対応するファイルを表示する窓が開いて、対応する行にカーソルが移動)、ed2で窓を移動し、ed3で窓を拡大している。端末窓のretryはccを実行したらエラーメッセージが多過ぎて、流れていってしまったのでcc | moreとやりなおした作業をあらわしている。再コンパイル・修正作業では、checkとはエラーメッセージの残りをみる作業、svrdとはファイルを保存して(save)、コンパイルをやりなおす(redo)作業である。一回の修正作業では虫が全部とれなかったのでもう一度編集をしている。

修正作業は9つの修正単位(マクロ操作)からなっているが、その各々はより細かな操作列3つから構成される。NMUの場合はエラーメッセージの行をマウスでダブルクリックするMB・DC、エディタ画面にマウスカーソルを移動し、マウスをクリックするpoint、およびキーボードからの入力をするkeyinからなり、XとEmacsの場合はエラーメッセージで示された行へ行くline、打ち込むコマンドを考えるmental、およびキーボードからコマンドを打ち込むcommandからなっている。それぞれの操作列ごとの計測結果を表5に示す。この場合は3回の試行の平均値を示した。また、各操作列ごとの実行時間の平均値と標準偏差を表6に示す。表7にはマウス操作の時間データをまとめてのせる。

[考察]

この実験では被験者と方式の組み合わせによる時間の違いが明白であった。NMUではYKがHKよりほとんどすべての項目で速い。端末窓では逆にHKがYKより速い。また、YKはXよりEmacsの方が速い。この違いはおもに修正作業からきている。エディタへの慣れが関係していると思われる。編集作業について、具体的にどこがどう違っているかをみていくことにしよう(表5)。

まず、エラーメッセージをみて、対応するプログラ

ムの行へカーソルを移動する作業を考える。NMUではHM、MC、MB(open)であり、手をマウスに持っていき、マウスを移動して対応するメッセージをダブルクリックする。端末窓(vi)ではエラーメッセージを目でみて対応する行番号を打鍵する。Emacsではいつでも^X^と打鍵するだけでよい。ただし、エラーメッセージはあるエラーの波及で余分に出るのが常であるから、つぎに修正すべき行はどれかを判断しなければならない。付け加えたエラーの2、3、8番目のそれぞれあとには読みとばすべきエラーメッセージがいくつか表示されていた。したがって、対応する(3-MB.DC)、(4-MB.DC)、(9-MB.DC)は平均値(表6)より大幅に遅くなっている。端末窓の場合も同様である。Emacsの場合は表6でもわかるように標準偏差が非常に小さく単純に同じコマンドを打鍵していることがわかる。ただし、Emacsの場合はエラーメッセージをよみとばして行を指定することができないので、不要なエラーメッセージもコマンドを実行してよみとばさざるをえない。その分のロスタイムは9780 ms (表4)にも及ぶ。

つぎは、対応するプログラムの行の中の正確な文字位置へカーソルを移動する作業を考える。NMUの場合はMC(T)、MB(point)であり、ダブルクリックした位置から対応するプログラムの位置へマウスカーソルを移動し、クリックすることで画面カーソルをその位置へ持っていく。これはマウスカーソルを持っていかなければならない位置の精度によってかかる時間が異なる。エラーの3、4、5が平均を大幅に越えている(HK、YKとも)が、マウスによる位置付けがしにくい状況に対応している。

端末窓ではmental操作として、行番号ないしは^X^を打鍵してから実際のコマンドを打鍵するまでの時間が計測されており、viでもEmacsでも似たような値(YKの場合、1604 msと1673 ms)になっている。これは、Cardらの打鍵レベルモデルでのM操作の時間1350 msに近い値である。

NMUではkeyin操作は正確にはHK,M,Kとかける。つまり、手をキーボードに持ってきて、少し考えて、打鍵すると分解できる。ほとんどの打鍵が^Hなどの1文字であるからこの計測値の大部分はHK操作とM操作である。ただし、viやEmacsの場合のmental操作と違ってずっと小さくなっていると思える。

大雑把にみると、修正する行を指定する時間はマウスを使った場合とそうでない場合とで(MB.DCとlineの比較)あまり違いがない。行内の正確な位置をさす時間はNMUではpointであるが、端末窓ではcommand操作列の中に埋め込まれていてはっきりわからない。したがって、行内の位置を指し、修正するまでまとめてみると、NMU-YKで3959ms、X-HKで3654msとなった。若干NMUの方が遅いがほとんどかわりはない。NMUではマウスの持ち替えといった作業をともなうのでかなり遅くなるかと思っていたが、編集作業ではmental操作の比率が大きいので思考している間に手が動くといった状況になれば(慣れるとそうなる)端末窓でエディタを使ったと同等の速度で編集ができるものと考えられる。

表7をもとにマウスの各操作にかかる時間を計算してみる。HM,MC(expand)とMC(expand)の差額より、HMは847 msとなる。この値はCardらの持ち替え時間0.4秒より少し大きいが、CardらのモデルではHMのまえにMを付け加えることになるのでより大きい値(0.4+1.35)となる。ここでのHMは若干の思考操作(0.847-0.4程度)を含んでいることになる。MCの時間をみると、MC(T)は文字間への移動であるのでそれ以外のボックスへの移動よりおそい。MC(redo)が遅いのは、MB(save)のあとにMC(redo)を実行するのだが、このMB(save)の結果が画面に表示されないからである。つまり、saveが成功したのかなとちょっと考えてしまうのである。save完了の表示をするように改良しなければならない。

HKもHMとほぼ同じ値となっている。MBのうち、MB(expand)、MB(move)、MB(select)が遅いが、それぞれボタンを押してから離すまでの動作によって指令を与えているからである。

7. 実験3

最後にもう少し規模の大きな実験をおこなう。文献[1]で模擬実験として扱ったもので、今回は実際に時間データをとることにした。作業は、ファイル名を引数として与えるとその名前をファイルの変更日付とともに書き出すプログラムfdatesを作ることである。被験者はUnixやCは知っているが、システムコールやライブラリについてはマニュアルをみないとわからないものとする。

実験はまずエディタで概略のプログラムを打ち込むところからはじまり、マニュアルのキーワードインデックスから変更日付をとる機能を説明したマニュアルページを探し出し、そのマニュアルを読み、対応する関数をエディタで打ち込んで、といった具合にすす

表8：実験3の結果

```
NMU
 7780 K(xman : xls /usr/i
 nclude^M)
3660 R(b)
3100 R(s)
4660 M
5160 K(cp /dev/null fdate
 s.c^M)
1700 M
小計      25060
—— 1 プログラム戦略作成
3200 K(xe fdates.c^M)
2380 R(b)
 400 R(s)
1420 HM,MC(W)
3900 HK M
54560 K()= text
12280? HM,M
1480 MC(menu)
小計      79620
—— 2 キーワード探索
1680 MB(menu.s)
 300 R(b)
 120 R(s)
1720 MC(W),HK M
 280 K(date)
1240 HM,MC(select)
 160 MB(select)
23000 M#1
小計      28500
—— 3 キーワード探索
 680 HK M
2200 K(^H^H^Hfile)
2420 HM,MC(all)
 120 MB(all)
 580 MC(select)
 160 MB(select)
小計       6160
——
6200 MP(s) M#2
 220 MB(JU)
 760 MP(s) M#2
 180 MB(JU)
 560 MP(s) M#2
 160 MB(JU)
12700 MP(s) M#2
小計      20780
—— 4 マニュアル
 380 MB(open=man.fstat)
1160 R(b)
 580 R(s)
4020 MP(read) M#3
 160 MB(JU)
6780 MP(read) M#3
 500 MB(SD)
 440 MP(s) M#3
 60 MB(SD)
 560 MP(s) M#3
 40 MB(SD)
 120 MP(s) M#3
 120 MB(SD)
2480 MP(s) MC(select) M#3
小計      17400
—— 5 編集開始
2980 MB(select)
1660 MC(copy)
2100 MB(copy)
——
4440 MC(point)
 180 MB(point)
 920 MC(paste)
1620 MB(paste)
小計      12900
——
10220? M MC(select)
小計      10220
——
1920 MB(select)
 900 MC(copy)
 600 MB(copy)
5540 MC(point)
 160 MB(point)
2280 MC(paste)
1400 MB(paste)
小計      12800
——
13060? M MC(point)
 140 MB(point)
 800 HK M
344007 K(^M ^Hstae^Ht(arg
 v[]^Hi], buf^H^H^H&buf);
 )
4660 HM,MC(point)
 180 MB(point)
 800 HK M  K(^H)
小計      54040
—— 6 st_mtimeの型さがし
11500 HM MP(read-fstat)
 #4
 440 MB(JU)
4000 MP(read) M#4
 320 MB(JU)
 560 MP(read) M#4
 140 MB(JU)
3140 MP(read) M#4
 340 MB(SD)
 420 MP(read) M#4
 440 MB(SD)
21240 MP(read) MC(close)
 M#4
 480 MB(close=fstat)
4480 MC(expose)
小計      47500
—— 7 types.hをみる
 460 MB(expose=/include)
 40 R(b)
 180 R(s)
5060 M(read)
 580 MB(SU)
 300 MP(read)
 560 MB(SU)
 180 MP(read)
 440 MB(SU)
 200 MP(read)
 160 MB(SU)
4320 MP(read) MC(T) M
小計      12480
——
 380 MB(open=/sys)
 440 R(b)
 180 R(s)
1540 MP(read) MC(SU)
 820 MB(SU)
 420 MP(read)
 140 MB(SU)
 300 MP(read)
 100 MB(SU)
3440 MP(read) MC(T)
小計       7760
——
 280 MB(open=type.h)
 220 R(b)
 340 R(s)
4580 M#5
 900 MB(SU)
10640 M#5
 720 MB(close=type.h)
1280 MC(expose)
 320 MB(expose-sW)
 680 HK M
1600 K(^H^H^Htime)
1560 HM,MC(all)
 240 MB(all)
 420 MC(search)
 180 MB(search)
M17060 M#6
小計      21060
—— 9 マニュアル
 480 MB(open=ctime)
 220 R(b)
 260 R(s)
5520 MP(read) M#7
 560 MB(JU)
9340 MP(read) M#7
1040 MB(SD)
11360 M#7
小計      28780
—— 10 編集継続
 840 MB(select)
1840 MC(copy)
1160 MB(copy)
10360? MP(s)
 140 MB(point)
11960? MC(paste)
1740 MB(paste)
小計      28040
——
2580 MC(point)
 100 MB(point)
1120 HK M
1180 K(^H^H^H^H^H)
1420 HM,MC(point)
 280 MB(point)
 800 HK M K(;)
9760? HM,MC(T) M
小計      17240
——
 780 MB(open=fstat)
3720 MC(point) M
 160 MB(point)
1520 HK M
33500? K(,buf^H^H^Hctime(
 &buf.st_mtime))
5940 HM,MC(point)
 180 MB(point)
 920 HK M
 860 K( %s)
2500 HM,MC(point)
 140 MB(point)
 680 HK M
 520 K(^H^H)
1660 HM,MC(point)
 100 MB(point)
1240 HK M
2560 K(Y),HM,MC(point)
 120 MB(point)
 320 M
9000? K( )=left hand,MC(W
 ) M
1460 HK M
小計      67880
—— 11 コンパイル
6680 K(xx cc -o fdates fd
 ates.c^M)
5320 R(b)
1540 R(s)
3020 HM,MC(move)
1500 MB(move)
2160 MC(T)
小計      20220
—— 12 修正
 360 MB(open=v1.c)
 280 R(b)
 480 R(s)
2020 MC(point)
 200 MB(point)
 800 HK M
2560 K(^M int i;)
2080 HM,MC(save)
 660 MB(save)
1820 MC(redo)
小計      11260
—— 13 再コンパイル
1020 MB(redo)
7920 R(b)
小計       8940
——
合計     559600
```

```
X Window
—— 1 プログラム戦略作成
1960 K(vi fdates.c^M)
2560 M R(b)
13020 K(imain(argv^H^Hgc,
 argv)^M)
2940 K(int argc;^M)
4620 K(char *argv[];^M)
1380 K({^M)
14480 K( for (i = 1; i <a
 rgc; i++) {^M)
11760 K( printf("%s\n", a
 rgv[i]);^M)
2940 K( }^M)
2320 K()^M^[)
小計      58080
——
3980 HM,MC(W)
1640 HK M
2240 K(xterm &^M)
3760 HM,MC(place)
1340 MB(place)
3320 R(b)
 40 MC(W)
1900 HK M
小計      18220
—— 2 キーワード探索
4280 K(man -k date^M)
2540 R(b)
3080 MP(s) R(s)
8480 M#1
小計      18380
—— 3 キーワード探索
3420 K(man -k file^M)
 580 R(b)
6240 MP(s) R(s)
 740 M
小計      10980
——
2760 K(!! | more^M)
 600 R(b)
1840 R(s)
1900 MP(s) M#2
 120 K( )
1640 R(s)
1460 MP(s) M#2
 200 K( )
 900 R(s)
1260 MP(s) M#2
 200 K( )
1540 R(s)
 700 MP(s) M#2
 120 K( )
 820 R(s)
 700 MP(s) M#2
 80 K( )
 660 R(s)
 860 MP(s) M#2
 160 K( )
 580 R(s)
11170 MP(s) M#2
 1940 K(q) M
小計      32210
—— 4 マニュアル
1220 MB(place)
1800 MC(W)
7920 HK M
小計      22620
——
8840 K(more /usr/include/
 sys/types.h^M )
 560 R(s)
16360 M#5
5740 K(q) M
小計      31500
—— 8 キーワード
3900 K(man -k time | more
 ^M)
4400 R(b)
 600 R(s)
15300 M#6
2140 K(q) M
小計      26340
—— 9 マニュアル
2460 K(man ctime^M)
540 R(s)
2800 HM,MC(W) MP(s) M#3
1780 HK M
小計      34140
—— 5 編集開始
2660 K(kkkkkkkk)
6440 K(O#include <sys/typ
 es.h>^M)
27660? K(#include <sys/st
 s^Hat.h>^M)
2040 K(j^H^[)
1380 K(jjjj)
15060? K(o struct stat bu
 f;^[)
16440? K(jo stat(argv[i],
  &buf);^[)
4360 HM,MC(W) M
1880 HK
小計      77920
—— 6 st_mtimeの型さがし
 480 K( )
 800 R(s)
1100 MP(s) M#4
 100 K( )
 600 R(s)
11720 HM,MC(W) MP(s) M#4
1180 HK M
小計      15980
——
2960 K(man -^Hstat^M)
 40 R(b)
2600 R(s)
1520 MP(s) M#4
 100 K( )
 640 R(s)
 580 MP(s) M#
 100 K( )
 620 R(s)
小計       9160
—— 7 types.hをみる
2960 HM,MC(W) M
1440 HK M
2240 K(xterm &^M)
 720 R(b)
4320 HM,MC(place) R(s)
1220 MB(place)
1800 MC(W)
7920 HK M
小計      22620
——
8840 K(more /usr/include/
 sys/types.h^M )
 560 R(s)
16360 M#5
5740 K(q) M
小計      31500
—— 8 キーワード
3900 K(man -k time | more
 ^M)
4400 R(b)
 600 R(s)
15300 M#6
2140 K(q) M
小計      26340
—— 9 マニュアル
2460 K(man ctime^M)
540 R(s)
2800 HM,MC(W) MP(s) M#3
1780 HK M
小計      34140
—— 5 編集開始
2660 K(kkkkkkkk)
6440 K(O#include <sys/typ
 es.h>^M)
27660? K(#include <sys/st
 s^Hat.h>^M)
2040 K(j^H^[)
1380 K(jjjj)
15060? K(o struct stat bu
 f;^[)
16440? K(jo stat(argv[i],
  &buf);^[)
4360 HM,MC(W) M
1880 HK
小計      77920
—— 6 st_mtimeの型さがし
 480 K( )
 800 R(s)
1100 MP(s) M#4
 100 K( )
 600 R(s)
11720 HM,MC(W) MP(s) M#4
1180 HK M
小計      15980
——
2960 K(man -^Hstat^M)
 40 R(b)
2600 R(s)
1520 MP(s) M#4
 100 K( )
 640 R(s)
 580 MP(s) M#
 100 K( )
 620 R(s)
小計       9160
—— 7 types.hをみる
2960 HM,MC(W) M
1440 HK M
2240 K(xterm &^M)
 720 R(b)
4320 HM,MC(place) R(s)
1220 MB(place)
1800 MC(W)
7920 HK M
小計      22620
——
8840 K(more /usr/include/
 sys/types.h^M )
 560 R(s)
16360 M#5
5740 K(q) M
小計      31500
—— 8 キーワード
3900 K(man -k time | more
 ^M)
4400 R(b)
 600 R(s)
15300 M#6
2140 K(q) M
小計      26340
—— 9 マニュアル
2460 K(man ctime^M)
540 R(s)
2800 HM,MC(W) MP(s) M#3
1780 HK M
小計      34140
—— 5 編集開始
2660 K(kkkkkkkk)
6440 K(O#include <sys/typ
 es.h>^M)
27660? K(#include <sys/st
 s^Hat.h>^M)
2040 K(j^H^[)
1380 K(jjjj)
15060? K(o struct stat bu
 f;^[)
16440? K(jo stat(argv[i],
  &buf);^[)
4360 HM,MC(W) M
1880 HK
小計      77920
—— 10 編集継続
4160 HK M
3920 K(kkkkkkk)
9300 K(ichar *ctime();^[)
4560 K(jjjjjjjjjlllllllll
 lll)
18860? K(i,ctime(&buf.)
小計      46460
——
5140 HM,MC(W)
2980 M
 260 K()<=left-hand!
 280 MB(raise)
2340 MC(W)
1260 HK M
小計      12260
——
6200 K(st_mtime)^[)
2460 K(hhhhhhhhhhhhhhhhhh
 hhhhhhhhhhhhhh)
9220 K(i %s:^H^[)
1820 K(:w^M)
2380 HM,MC(W)
1500 HK M
4520 K(q) M
小計      28100
—— 11 コンパイル
5640 K(cc -o fdates fa^Hd
 ates.c^M)
2020 R(b)
1900 M R(s)
小計       9560
——
5140 K(cc -o fdates fdate
 s.c)
1340 K(^H^H^H^H^H^H^H^H^H
 ^H^H^H^H^H^H^H^H^H^H^H
 ^H^H^H^H^H)
2140 M
小計       8620
——
9240 K(pu xrec/j5^M)
 160 R(b)
1240 M
小計      10640
——
5200 K(cc -o fdates.^H fd
 ates.c^M)
2100 R(b)
2400 HM,MC(W) R(s)
小計       9700
—— 12 修正
1280 HK M
9620 K(:9^Mkko int i;^[:w
 ^M)
1420 HM,MC(W)
小計      12320
—— 13 再コンパイル
 780 HK M
1000 K(!!^M)
7160 M R(b)
小計       8940
——
合計     517310
```

む。インクルードファイルの内容をみることも必要である。X Windowをふつうに使った場合は、窓を3つ開けなければならない。

実験は1セッション通して約10分かかった。結局NMUとX Windowの2つの方法でそれぞれ1回のデータしか計測できていない。NMU-HK、X-HKの順で実験した。被験者はHKのみである。操作単位での分析結果を表8に示す。(X Windowの場合の結果をみればUnixユーザには何をしたかがわかるであろう。)

[考察]

実行時間をみてみると、NMUで559.6秒、X Windowで517.3秒であった。ただし、NMUを最初に実験したので、X WIndowでの実験には慣れの効果が出ていると思われる。表9の中でM#1。。。M#7までの#記号のついたM操作はそれぞれの番号で両方の方式に対して画面でのマニュアルを読む作業に対応づけてある。たとえば、M#1は両方の方式でのマニュアルdateを読んだ時間である。慣れの効果をなくすためには、対応した操作にかかった時間を同じとしなければならない。NMUでのM#操作の実際の合計時間は156.98秒、Xでは105.75秒であった。この差額(51.23)をXの全実験時間に付け加えると、568.53秒となりNMUのより長くなる。また、画面の読み取り以外で極端に多くの時間がかかっている操作(表8での時間データのあとに?をつけたもの)を抜き出してみると、NMUでは144.54秒、Xでは78.02秒となる。この時間も公平に割り振ると、NMUのほうがだいぶXよりも速いということがいえそうである。しかし、これだけの実験ではそう結論づけるには早急すぎるであろう。

つぎに、キーボードマウス間の移動による負荷がどの位かをこのデータをもとに検討してみることにしよう。

X Windowをふつうに使った場合は、vi用、man用、およびインクルードファイル検索用の3つの窓が同時に必要となった。もちろんこれらの窓を開くための制御用窓がもうひとつあるわけである。窓を新たに開くためには、HM,MC,HK,K(xterm &),HM,MC,MB(place),MC,HKのようにHK,HMペアが2組必要である。この場合は窓を新たに2つ開いたので合計4組必要であった。実際にXの場合で実行されたHK,HMペアは12組であった。注目している窓の切り換えには1組必要であるから、8回窓を切り換えたことになる。各々の切り換えの実測データにはM操作が含まれてとられているので、正確には計算できない。最小のもので2200 msであった。つまり、窓を切り換える手間は2秒以上かかったのである。

NMUの場合はXの場合と逆にふつうはマウスを持っていて、テキストを打鍵するときだけキーボードに手がいくという動作となる。NMUの場合は再コンパイル終了時にマウスを持っているので、HMが15回、HKが14回であった。Xの場合と同程度であった。実験2では現われていなかったマウスボタン操作としては、MB(copy)とMB(paste)がある。MB(copy)が平均953 ms、MB(paste)が平均1586 msであった。これは両方ともプルダウンメニューからの選択であり、1回のクリックでよいopenやsearchなどに比べて遅くなっている。

8. おわりに

実験1と実験2でファイルを探す、プログラムを編集するという典型的な作業を例にとったが、ファイル探しはかなり優位である。エディタ作業はエディタの性能に強く依存するのでNMUがどれだけよいのかははっきりしないが、現状のNMUのエディタであるxeの機能はかなり小さいのに、端末窓のviやEmacsと対当の時間で作業ができたことをみると、十分役に立つといえる。

この実験では経過時間からのみ考察したが、使いやすさの要因は時間だけではない。探したいものがそこにみえているのにその文字列を打ち込むといった端末窓のやりかたに比べて、NMUではそれをクリックすればよいといった直接的な扱いは今回は計測できなかったが十分ユーザにとって質のよい使いやすさを提供していると考えられる。

[参考文献]

1. 久野、角田:流れて行かないUnix環境をめざして、情報処理学会夏のシンポジウム「究極のプログラミング環境」報告集、1988.
2. S.K.Card, T.P.Moran & A.Newell: The Keystroke-Level Model forUser Performance Time with Interactive Systems, Comm.ACM,Vol.23,No.7(July 1980), pp.396-410.

本 PDF ファイルは 1988 年発行の「第 29 回プログラミング・シンポジウム報告集」をスキャンし、項目ごとに整理して、情報処理学会電子図書館「情報学広場」に掲載するものです。

この出版物は情報処理学会への著作権譲渡がなされていませんが、情報処理学会公式 Web サイトに、下記「過去のプログラミング・シンポジウム報告集の利用許諾について」を掲載し、権利者の捜索をおこないました。そのうえで同意をいただいたもの、お申し出のなかったものを掲載しています。

https://www.ipsj.or.jp/topics/Past_reports.html

過去のプログラミング・シンポジウム報告集の利用許諾について

情報処理学会発行の出版物著作権は平成 12 年から情報処理学会著作権規程に従い、学会に帰属することになっています。

プログラミング・シンポジウムの報告集は、情報処理学会と設立の事情が異なるため、この改訂がシンポジウム内部で徹底しておらず、情報処理学会の他の出版物が情報学広場 (=情報処理学会電子図書館) で公開されているにも拘らず、古い報告集には公開されていないものが少からずありました。

プログラミング・シンポジウムは昭和 59 年に情報処理学会の一部門になりましたが、それ以前の報告集も含め、この度学会の他の出版物と同様の扱いにしたいと考えます。過去のすべての報告集の論文について、著作権者（論文を執筆された故人の相続人）を探し出して利用許諾に関する同意を頂くことは困難ですので、一定期間の権利者捜索の努力をしたうえで、著作権者が見つからない場合も論文を情報学広場に掲載させていただきたいと思います。その後、著作権者が発見され、情報学広場への掲載の継続に同意が得られなかった場合には、当該論文については、掲載を停止致します。

この措置にご意見のある方は、プログラミング・シンポジウムの辻尚史運営委員長 (tsuji@math.s.chiba-u.ac.jp) までお申し出ください。

加えて、著作権者について情報をお持ちの方は事務局まで情報をお寄せくださいますようお願い申し上げます。

期間： 2020 年 12 月 18 日 ～ 2021 年 3 月 19 日
掲載日： 2020 年 12 月 18 日

プログラミング・シンポジウム委員会

情報処理学会著作権規程
https://www.ipsj.or.jp/copyright/ronbun/copyright.html