プログラミング環境 第9回

久野 靖 *

1991.11.26

12 ウィンドウシステム

12.1 ウィンドウシステムの由来と歴史

計算機環境全般において、人間が直接計算機とやりとりする手段ないし媒体はシステムの使われ方に大きな影響を持つ。歴史的に見ると:

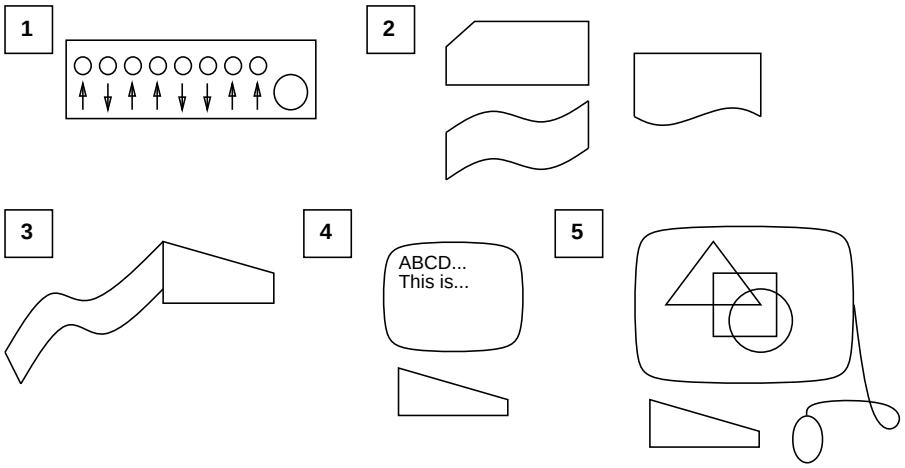

図 1: 計算機と人間の対話手段の変遷

1. コンソールパネル:押しボタン、スイッチ、ランプなどのあつまり。主記憶とレジスタの内容を読み書きする→ビット列そのものを扱う。

2. オフライン媒体。パンチカードとか、紙テープ。字の形で入出力できる、という点では大きな進歩だが、何しろ「予め別の場所で用意する」ので融通が効かない。

3. テレタイプ端末。その場で対話的に計算機を使えるのでかなりな進歩だが、入力はともかく出力が遅い。行エディタによる編集。

4. CRT 端末。これで画面エディタ、画面モード入出力などぐっと使いやすくなるのだが、しょせん表示できるものは文字のみ。図形端末もなくはないが、高価だし一般的でない。画面もそんなに広くないので、同時に複数の仕事を表示させておくのは現実的でない。

*筑波大学経営システム科学専攻

5. ウィンドウ。字だけでなく、絵も表示に混ぜられる。字のスタイルなども変化可能。図形が自由に使える。複数の仕事を並行して表示できる。ポインティングデバイス (マウスなど) で「指さす」ことができる。

のような経過をたどってきているわけである。こうして見ると、計算機が贅沢品でなくなるとともに、計算機が利用者のために色々な作業をやって「使いやすく」してくれる、という流れがわかる。その意味ではウィンドウシステムは現在の所一番「使いやすくできる可能性を持つ」計算機との対面方法である、ということになる。

このようなウィンドウ文明を最初に開発したのは Xerox 社で、Alto というシステムが現在のような高性能ワークステーションの「はしり」とされている。その後 Xerox 社ではこれを文書作成用システム (今風にいえば DTP マシン) として、Star という商品名で売り出したがあまり成功しなかった。日本では富士ゼロックスがその日本語版 J-Star を扱っている。一方、Alto に影響を受けた Steve Jobs が Apple 社で開発したのが Lisa とその後継機種 Macintosh で、その隆盛はご存じの通り。

一方、Sun を始めとするワークステーションメーカでも次々にこのようなシステムを開発し自社のシステムに搭載するようになった。ただし、これらのウィンドウシステムはそれぞれまったく別個のものであった。しかし、MIT が DEC 等と協力して開発した X-Window というシステムを無料公開し、各社のシステムがこれを搭載することが相次いだ結果 (Sony NEWS もそうだ)、X が少なくとも Unix を搭載するワークステーション文明の中では「事実上の標準」の地位を獲得するに至ったわけである。というわけで、ここで皆様が使われているのも、また以下で説明するのも主として X-Window(version11, Release4) ということになる。

12.2 ウィンドウシステムの概念

ではここで、ウィンドウシステムとはどんなものか改めて見てみよう。図 2 に、典型的なウィンドウシステムの「道具だて」を示す。まず、スクリーンというのは要するに物理的な画面全体に

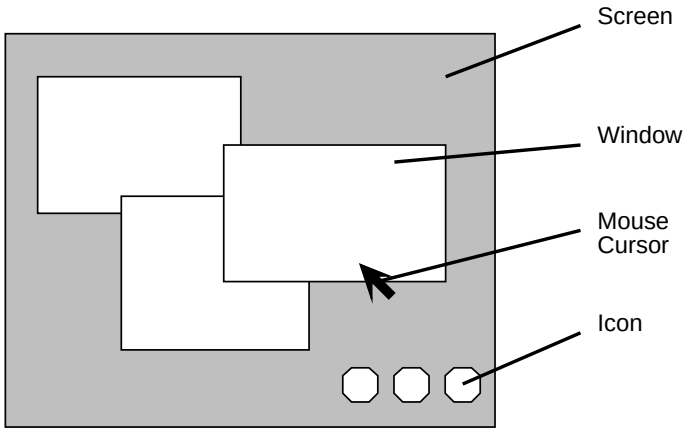

図 2: ウィンドウシステムの道具だて

相当する。その中に四角い[1]窓が複数[2]存在し、それぞれが別々の作業に対応する。[3]そして、マウスの動きに呼応して画面上を動くマウスカーソルがあり、これを使って「どの窓」という切替を

[1]最近は四角くないのも可能なのが増えているが。
[2]互いに重なりあっていいものと、重なりを許さないのの 2 通りの流儀があるが。
[3]場合によっては別の窓の作業のための補助的な窓であることもあるが。

やるのが多い。[4][5]

さらに、画面上に窓のほかに小さな絵柄をもった「アイコン」なるものを置けるのが普通である。アイコンが何を表すかはシステムの流儀によってまた大幅に違う。たとえばXを含むUnixのウィンドウシステムでは、アイコンは「窓の閉じたもの」であり、窓がたくさんできた時にごちゃごちゃにならないためにあるという感じが強い。一方、Alto/Star流ではアイコンはディレクトリやファイル(あちらの用語では文書とフォルダ)とか、プリンタ装置などの「もの」に対応している。Macもだいたいそうであるが、加えてプログラムもファイルに入っているからアイコンが「動かす前のプログラム」である、という意識が強い。

さて、今度は「窓」の中の「作業」であるが、Unixをベースにしたシステムの場合、これまで前節4.のCRT端末を使って仕事をしてきた、という経緯があるので、「CRT端末の真似をする」窓を使ってその上でこれまでと同様に仕事をする、という場合が圧倒的に多い。これだと従来のプログラムがすべてそのまま使えるので便利ではあるが、端末が一杯ある、というだけではあまり面白くないのも確かである。これに対して、StarやMacでは最初からウィンドウを前提としたシステム作りを行なったので各ソフトとも絵やマウスなどウィンドウの機能を活用しているものが多く、より「使っていて楽しい」といえる。とりあえず皆様に使って頂くのが「楽しくない」方のUnix-WSなのは残念だが、もっとも最近はウィンドウを活用したプログラムもいくらかは増えつつある(といっても、Unixでのソフトの蓄積は膨大なので...)。

12.3 ウィンドウシステムの構造

ウィンドウシステムの外観は上記のようなものだが、ではそのようなシステムはどのようにしたら作れるだろうか?ハードウェア的にはCPUと画面は図3にあるように、フレームバッファを介してつながっているのであった。(覚えてらっしゃいますよね?) [6]さて、こういうハードがあり、

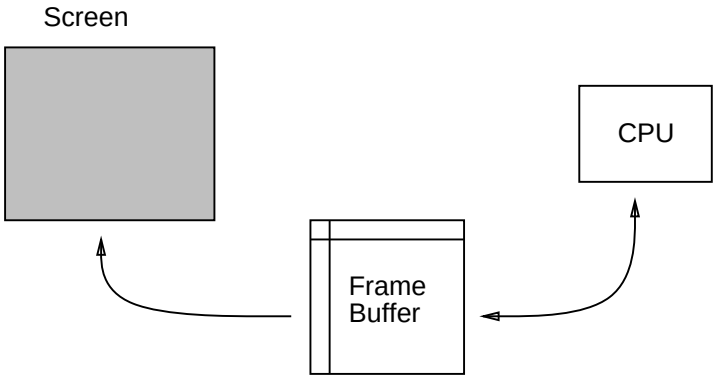

図 3: フレームバッファとビットマップ画面

[4]が、窓の切替えのたびにマウスを掴むのは面倒なのでキーボードだけで窓を切替える方がいいといってそういう機能を使う人も多い。

[5]マウスで窓の切替えをやる際、「窓の領域にカーソルが入ったらそれだけでそっちに切り替わる」という流儀と「切替えたい窓にカーソルが行くだけでなく、そこでマウスボタンを押すと始めて切り替わる」という流儀とがある。

[6]

おまけ:

フレームバッファというのは機能的には普通の主記憶と同様にアクセスできるのだが、書き込むとその場所に書き込んだビット列がそのまま対応する画面上の点の輝度の明暗に対応して現れるようになっている。例えばSunだと

```
screendump ファイル名
```

とやると、フレームバッファの内容が指定したファイルに保存される。その`od`を見ればフレームバッファの内容が見られる。逆に保存したファイル名を指定して

ソフトからアクセスできるとして、では前節で見たようなウィンドウシステムを作るにはどうしたらいいと思うか?

一つの方法は、窓を作り出すような各プログラムが「自分の窓はどこどこにあるから、その場所に窓の内容を描こう」という風に各窓のプロセスに任せるやり方である。SunView などはこの方式である。この様子を図 4 に示す。しかし、この方式だと各窓の重なり具合によって隠れてい

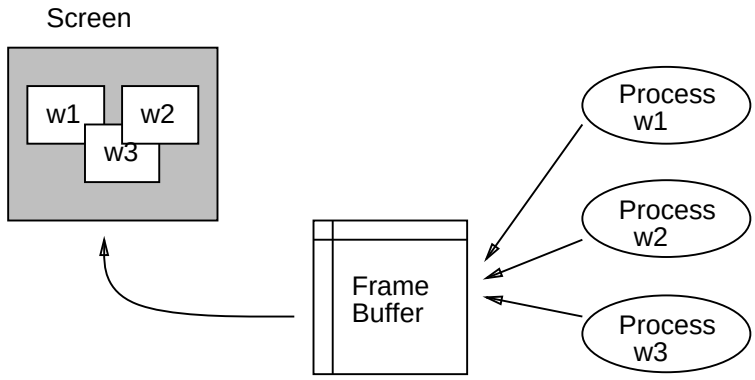

図 4: 直接描画方式のウィンドウシステム

るところを互いによけて描く必要があるし、これを含め様々な図形出力ルーチンも各プロセスごとに持つ必要があるので大変である。

そこで、現在ではウィンドウサーバと呼ばれるプロセスを 1 つ用意し、このプロセスが窓を作る各プロセス (クライアント、と呼ばれる) の依頼を受けてフレームバッファへの書き込みを管理する、「サーバ方式」が一般的である。X-Window もこの方式を取っている。この様子を図 5 に示す。この方式では、各種の描画ルーチンはサーバのみが持てばよく、また窓の重なりを考慮した

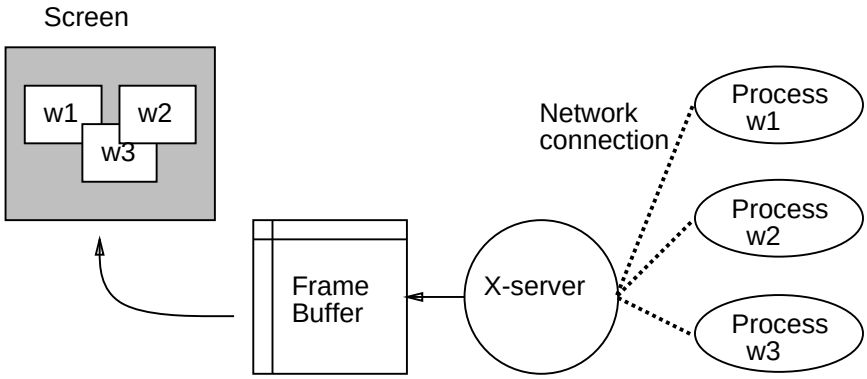

図 5: サーバ方式のウィンドウシステム

描画もサーバが一括しておこなえばよいので、各窓に対応するプロセスはずっと簡単である。各プロセスはサーバとネットワーク通信機能によってつながり、サーバに対して「窓を作って欲しい」、「窓のどこにどんな図形/文字列を描いて欲しい」などの要求を出す。従って、各プロセスは必ずしも画面のある計算機の上で動いていなくてもよい。サーバはフレームバッファに書き込むので、必ず画面のある計算機で動く。[7]

`screenload` ファイル名

とやると、保存された内容がフレームバッファに書かれるので画面が保存したときの様子に戻ってしまう。NEWS もコマンドはないけど類似のことができる。X 端末だと... フレームバッファは端末についてるのでこれはできない。残念。

[7]で、X 端末というのはフレームバッファを持ち、サーバのプログラムだけが動いている計算機、ということである。

12.4 ウィンドウプログラムの構造

ところで、上で述べた要求 (リクエスト) というのはクライアントからサーバへ情報を渡す通信だったが、逆にサーバからクライアントに情報を渡す通信も当然必要である (例えばどんなものか、わかりますか?)。これを「イベント」と呼ぶ。言い替えればイベントとは窓の上で起こったことをクライアントに通知する「信号」のようなものである。イベントの種類としては例えば次のようなものが代表的である。

窓が見えるようになったよ。
窓の大きさや位置が変化したよ。
窓の中でキーやマウスボタンがが押されたよ。
窓の中でマウスが動いたよ。

そして、ウィンドウを用いたプログラムというのはこれらのイベントに対処してそれぞれの動作を行なう (例えば、「終了」ボタンの上でマウスが「押された」というイベントが来たら、そのプログラムを終る、という風に)。従ってプログラムの構造はすごく簡単化して考えると

```
初期設定...
while(True) {
    Event e;
    ReadEvent(e);
    イベントの種類によってそれぞれの動作;
}
```

という形になっている。このようなプログラム構造を「イベントドリブン」になっている、などという。実はこの辺の構造はサーバ方式でなくても (たとえば SunView や Mac Toolbox のプログラムでも) 基本的に同様である。

12.5 X-Window

ここまではあくまでもウィンドウシステム一般の話だったつもりだが、とりあえず皆様が触って観察して見られるのは普段使っている X-Window なので、以下では皆様が実際に試して見られる、X の諸側面について取り上げる。ただし、例によって「こんなことができる、面白い」ではなく「内部がどういう構造になっているから、従って外部のこういうふるまいが説明できる」という立場に立って受け止めて欲しい。

12.5.1 クライアント

さて、X はサーバ方式のウィンドウシステムなのでクライアントプロセスはサーバと通信さえできればどのマシンで動いていてもよい。でも、ネットワーク中には色々な人のサーバが同時に動いているはずである。では、「どのサーバに窓を作るか」はどうやって決めるのだろうか。これには 2 通り方法がある。

1. プログラムを起動する時、「`-disp` ホスト名`:0.0`」というオプションを指定することで明示的に指定する。
2. 環境変数 `DISPLAY` に「ホスト名`:0.0`」なる文字列が入っていて、これによって定まる。

各プロセスはもちろん別の (Unix などが動いている) 計算機上にある。

オプションの指定があればそれは環境変数に優先する。その他によく指定するオプションとしては、クライアントの窓の位置や大きさを指定するものがある。これは「-geom 幅 x 高さ+X 座標+Y 座標」なるオプションで指定できる。座標の前の符合を+の代わりに-にすると、「画面の右/下端からの距離」の意味になる。

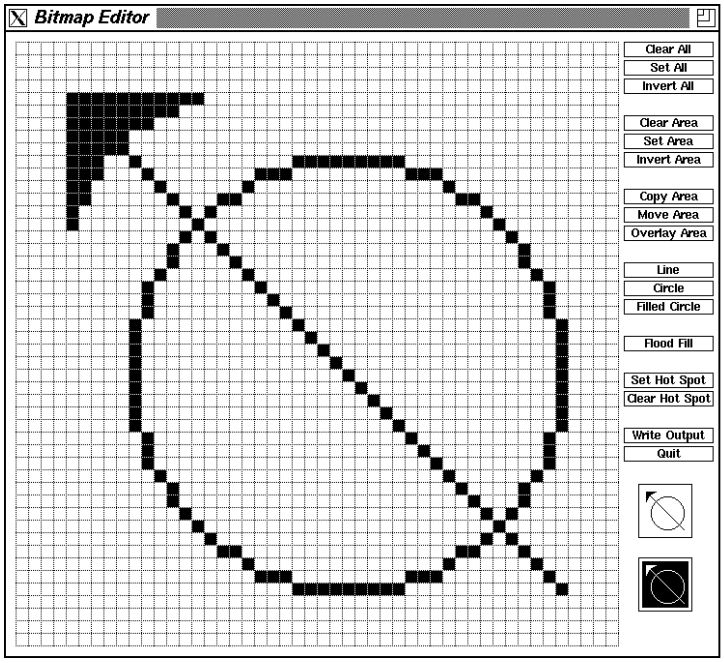

図 6: bitmap の窓

さて、それでは具体的にはどんなクライアントがあるかも簡単に列挙する。それぞれの詳しい説明は当然マニュアルページを参照されたい。

```
kterm  -- 端末のまねをする窓の漢字版。普段使っているのはこれ。
xterm  -- 同上、ただし元は英語のみ。日本語化されたのもある。
bitmap -- アイコンの絵などビットマップの絵を作成する。
idraw  -- MacDraw のそっくりさんのお絵描きツール。
xcalc  -- 電卓。
xbiff  -- メールが来ているかどうかを知らせる。
xclock, oclock  -- 時計。
ico, maze, puzzle, ...  -- 様々なお遊び。
xset   -- サーバの様々なオプションを設定する。
```

これらのクライアントは普通のプロセスだから、終らせるには (終り、というボタンがついているものもあるが) そのプロセスを殺せばよい。

ところで、アイコンや背景などに自分の好みの「絵」を入れたいと思ったらビットマップファイルを作る必要があるので、bitmap だけはもう少し詳しく説明しておこう。起動方法は

```
bitmap ファイル名 [幅 x 高さ]
```

による (画面、フォント、位置なども指定したければできる)。始まると指定した大きさの「方眼紙」が出てくるので、その上でマウスの左ボタンを押すとそこが黒になり、右ボタンで白くなる。その他丸、四角、線なども描ける。OK と思ったら Write でファイルを書き出し、Quit すると終る。bitmap によって作られたファイルは、アイコンやカーソルなどに使用することができる。

12.5.2 フォント

X に限らず、ウィンドウシステムでは画面上に様々なフォントが表示できる。X では現在どんなフォントが利用可能かを表示したり、あるフォントがどんな形の字かを見てみるのに次の指令

が使える。

```
xlsfonts                -- 利用可能なフォントの一覧を表示
xfd -fn フォント指定    -- 指定したフォントの文字を表示して見せる
```

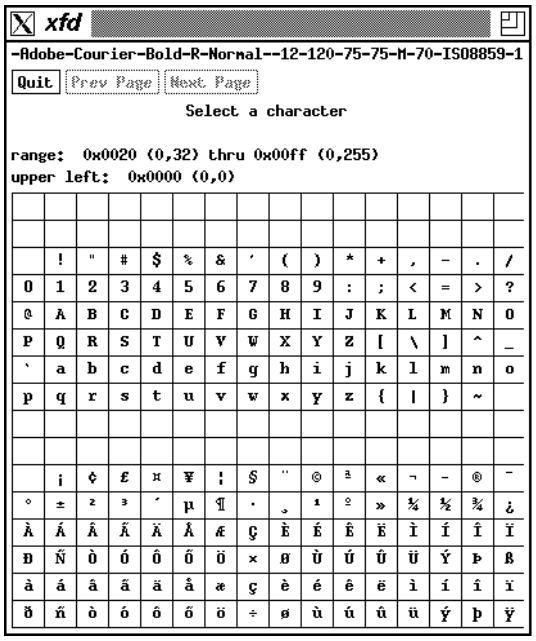

図 7: xfd の窓

一般にどのクライアントでもそれが表示に使用するフォントを「-fn フォント指定」で指定することができる。また kterm では合わせて漢字用フォントも「-fk フォント指定」で指定できる。図 7 は xfd で書体を表示させている例である。

「フォント指定」は、一般に次のような文字列である。

```
-Maker-Family-Weight-Slant-Kind--Dots-Points-Xres-Yres-Spacing-Width-...
```

ただし、

```
Maker     -- メーカ名
Family    -- 書体 (ゴチックとか明朝とか...)
Weight    -- 黒さ。medium と bold がある
Slant     -- r(立体)、i(イタリック)、o(オブリック) など
Kind      -- ほとんど normal しかないが。
Dots      -- 何ドットか。
Points    -- 何ポか。
Xres, Yres  -- 横/縦方向解像度。
Spacing   -- p(可変幅)、m(固定幅)
Width     -- 平均幅
```

さらに後ろの方には文字コード規格などの情報もある。これらを全部指定してると大変なので、*(任意) を使って、例えば

```
*-courier-bold-r-*--24-*
```

などのように指定する。これは「クーリエ書体のボールドの立体、24 ドットのてきとーなもの」という意味である。ただし、より古いフォントは単に短い「名前」しか持っていないものもある。例えば漢字の 14/16/24 ドットフォントは k14/k16/k24 という名前で、それに対応する半角フォントは a14/a16/a24 という名前である。

12.5.3 リソース

Unix では通常、各プログラムに対する初期設定をホームディレクトリの . で始まるファイルに入れておく。たとえば.cshrc などがそうである。しかし、X-Window 関連の場合はこの方法はあまりよくない (なぜか?)。そこで、X サーバの中にリソースデータベースというものを用意し、オプションの標準値をこの中に設定しておく。その内容を見たり設定したりするには xrdb 指令を使う。例えば次の通りである。

```
% xrdb ~/.Xdefaults          -- ファイルからデータベース設定
% xrdb -q                    -- データベースの内容表示
kterm*font:       a14
kterm*kanjiFont:  k14
% xrdb -m                    -- 端末から内容を追加
xterm*font:  *helv*bold*25*
^D
% xrdb -q
kterm*font:     a14
kterm*kanjiFont:        k14
xterm*font:    *helv*bold*25*
%
```

このようにした後で xterm を呼ぶと helvatica-bold-25pt フォントで表示する xterm の窓ができるはずである。

12.5.4 サーバの調整

上記のリソースは各種クライアントのオプションをまとめて設定するものだったが、この他にサーバの状態を変更する機能がある。

```
xset m 感度 閾値    -- マウスの感度を調整する
xset c ボリューム   -- 0〜100 の値でキークリック音の大きさを指定
xset r on/off       -- オートリピートの on/off
xset q              -- 設定値の現状値を表示
xsetroot -cursor カーサ マスク   -- カーソルの形状を変更
xsetroot -bitmap 背景  -- 背景を指定したビットマップに変更
xsetroot -gray          -- 背景を灰色にする
xsetroot -def           -- 設定を標準値に戻す
xphoon                  -- 月の形の背景にする
```

12.5.5 起動

フレームバッファを持ち、普段は X が動いていないマシン上で X を起動するには xinit という指令を使う。このプログラムはまず X サーバを起動し、続いてホームディレクトリにあるファイル.xinitrc の内容をシェルスクリプトとして実行する。典型的な.xinitrc の内容を次に示す。

```
xset m 5 2                  -- マウスの感度を好みの値に設定
xrdb $HOME/.Xdefaults       -- リソースデータベースの設定
oclock -geom 100x100-0+0 &  -- 時計を表示
twm &                       -- ウィンドマネージャを起動
kterm -geom 80x24+300+50 -C -e /usr/new/csh  -- 端末窓起動
```

このように、.xinitrc に起動したいクライアントを書いておけばそれは X を起動すると同時に自動的に開始されることになる。

また、OS が立ち上がると同時に X も動き始めるように設定されたシステム (X 端末もこの 1 種) では、これと併せて xdm というプログラムが動いていて、これが login プロンプトの画面を表示させている。この場合には利用者がやってきてユーザ名、パスワードを入力しそれが正しいと xdm はユーザのホームディレクトリにある.xsession というファイル (なければ代わりにシステムに備

わる標準のものが使われる) をシェルスクリプトとして実行する。結局、.xinitrc も.xsession も内容としては同様のものでよいわけだ。

また、xinit はこのシェルスクリプトが修了すると X サーバを停止して終るので、この例だと最後に書かれている端末窓が修了すると X も終って裸コンソールの状態に戻ることになる (xdm の場合も同様だが、ただし X サーバは停止させず login 窓を表示する状態に戻る)。従って、twm が終った時に X(ないし自分のセッション) も終るようにしたければ twm を最後に&なしで書けばよい。

12.5.6 ウィンドマネージャ

ところで、上で出てきた twm というのはウィンドウマネージャと呼ばれる特別なクライアントである。ウィンドウマネージャの仕事は、窓の位置を変更したり窓をアイコンにしたりといった窓の操作をユーザが行なうのを助けることである。例えば窓にタイトルバー (窓の名前を記した部分) がついているのも、実はウィンドウマネージャの機能の一部である。ウィンドウマネージャは、利用者がマウスで窓を操作したりメニューを出したりといった操作を行なうと、その情報を X サーバから教えてもらい、それに呼応して窓を動かしたりアイコンにしたりする (正確には窓を動かしたりアイコンにしたりするよう X サーバに依頼する)。この状況を図 8 に示す。実は、この

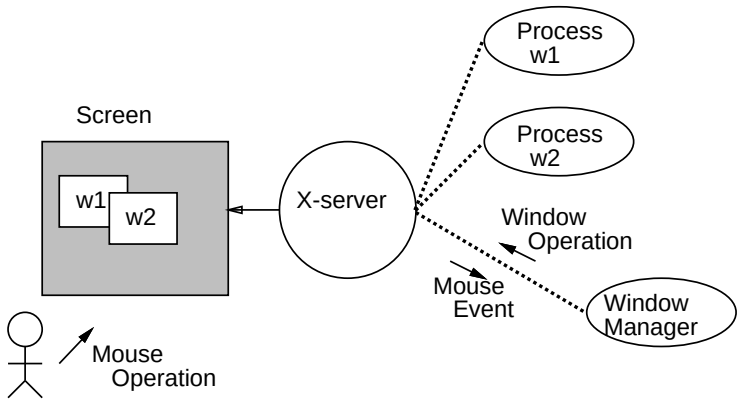

図 8: X におけるウィンドウマネージャの位置付け

ようにウィンドウマネージャが普通のプロセスである、というのは X の特徴の一つである。旧来のウィンドウシステム (Star、Macintosh など) では窓を動かしたりするのはウィンドウシステムそのものの機能として組み込まれていて、そのやり方を変更するのは不可能であった。一方、X ではウィンドウマネージャを取り買えると窓の操作のスタイルが変化する。

ここで示した例ではどれも twm が動かしてあったが、これらの他に NEWS などでは mwm と呼ばれるウィンドウマネージャが標準で備わっている。その他、rtl、uwm、gwm、awm など多数のウィンドウマネージャがこれまでに作られている。さらに、それぞれのウィンドウマネージャのふるまいも、例えば twm ならホームディレクトリの.twmrc に設定を書いておくことでカスタマイズできる。詳しくはマニュアルページを参照されたい。

12.5.7 X の挙動の観察

ここまでで一般の利用者が X を利用する時に目に触れるものについて一通り述べたが、この他に X の挙動を観察したりするのに役に立つ指令がいくつか用意されている。

まず、xlswins という指令を使うと、現在画面上にある窓の ID(プロセスでいえば PID のようなもの) と階層構造を調べることができる。また、これで調べた ID を利用してそれぞれの窓について次のような指令を活用することができる。

```
xwininfo -id 窓 ID        --- 窓の位置、大きさその他の情報を表示。
xkill -id 窓 ID           --- その窓を強制的に消去する (殺す)。
xwd -id 窓 ID >ファイル   --- その窓の内容をダンプする。
xwud -in ファイル         --- ダンプした内容を表示させる。
xpr -dev ps ファイル      --- ダンプした内容を PostScript 形式にする。
```

もちろん、PostScript 形式になったファイルは PostScript プリンタ (レーザーライタなど) に出力できる。これらの指令の詳細についてはマニュアルページを参照のこと。

また、xev という指令を動かすと画面に窓が現れ、その窓についてサーバが発生させたイベントを観察することができる。さらに恐ろしいことに、

```
xev -id 窓 ID
```

で、既にある任意の窓に対するイベントを覗き見することまでできる。

A 演習、課題

A.1 12 節の演習

12-1. (X 端末でなければ) ウィンドウサーバ、個別のクライアント、そしてウィンドウマネージャのプロセスを凍結すると、それぞれどんなことが起きるか。またそれは X-Window の構造から考えてどう説明できるか考えてみよ。例えば、クライアントプロセスを凍結した状態で別の窓を動かしてその一部を覆い隠した後取り除けると隠されていた部分はどうなるか? それはなぜか?

12-2. 他のマシンに rlogin して、そこで kterm その他のクライアントを起動し、自分のいるマシンに窓を開いてみよ。そうやって使う場合、rlogin した状態で使うのとどう異なるか、そういう使い方はどういう利点があるかを具体的に示せ。

12-3. `xlfonts` でどんなフォントがあるか調べ、それらがどんなフォントか `xfd` で観察し、どのようなフォントセットが用意されているかを自分なりにまとめよ。また、これらのフォントを使った kterm/xterm を起動して普段のフォントと使い心地を比べてみよ。[8]

12-4. 上記のようなフォント変更を、オプションで指定する代わりにリソースで指定してみよ。例えば、ただ「kterm」というと 24 ドットフォントのができるようにしてみよ (目の悪い人に好適)。また、フォント以外にも何かリソースを使用してプログラムのオプションを指定する、というのをやってみよ。

12-5. 普段自分が使っている状態で窓はいくつあるか考えてみよ。そして、`xlswins` で調べて予想が合っているかどうか確かめよ。また、画面にある全ての窓の (窓 ID を記入した) 配置図を作成してみよ。(または画面ダンプに記入してもよい。) また、何でそうなっているかも考察せよ。[9][10]

12-6. 上で調べた窓 ID をもとに、`xkill` で窓をいくつか殺してみよ。殺すとどんなことが起きるか? またそれはなぜか? [11]

12-7. `xev` を窓 ID 指定なしで使って、X では具体的にどのようなイベントがどのような時に送られるか観察してまとめよ。また窓 ID 指定しながら使って、具体的なクライアントプログラムの場合でも確認してみよ。

A.2 9 回目の課題

この回の報告を出される場合には 12-1〜12-7 から 2 個以上選択して下さい。

[8] 例えば、`kterm` の普段のフォントは半角が a14、漢字が k14 である。これを a16,a24,k16,k24 等ととりかえてみよ。半角と全角で大きさが不つりあいだとどうなるか? なお、サイトによってはこれらのフォント全部はないかも知れないし、もっと色々あるかも知れない。

[9] どの窓がどれか分からなければ、`xwd + xwud` で中身を別の窓として表示させてみれば分かる。ただしその窓がちゃんと開いて見える状態でないとうまく行かないかも知れないので注意。

[10] 画面ダンプは根窓の `xwd + xpr` でできるはず。

[11] ウィンドウマネージャが落ちて入力ができなくなることがあるかも知れないが、慌てず騒がず別の端末から入って自分のプロセスを全部 (実は適切な 1 個!) 殺せばセッションがリセットされるはず。