```
From unizh!uucp Tue Feb 26 22:02:43 1985
Received: by vlsivax.UUCP (4.12/4.7)
        id AA09429; Tue, 26 Feb 85 22:02:40 -0100
Received: by unizh.UUCP (4.12/4.7)
        id AA11156; Tue, 26 Feb 85 18:32:28+0100
Received: by cernvax.UUCP (4.12/4.7)
        id AA10346; Tue, 26 Feb 85 16:29:34 -0100
Received: by mcvax.UUCP; Tue, 26 Feb 85 15:26:00 -0100 (MET)
Return-Path: <decvax!ucbvax!Wirth.pa@Xerox.ARPA>
Received: by decvax.UUCP (4.12/1.0)
        id AA21492; Mon, 25 Feb 85 22:19:19 est
Received: from Xerox.ARPA (xerox.ARPA.ARPA) by UCB-VAX.ARPA (4.24/4.42)
        id AA04268; Mon, 25 Feb 85 12:30:05 pst
Message-Id: <8502252030.AA04268@UCB-VAX.ARPA>
Received: from Cabernet.MS by ArpaGateway.ms ; 25 FEB 85 12:20:44 PST
Date: 25 Feb 85 12:19:34 PST
From: unizh!cernvax!mcvax!decvax!ucbvax!Wirth.pa@XEROX.ARPA
Subject: New OS
To: BERKELEY!cernvax!unizh!ethz!Wille (M.Wille IfI)
Cc: Wirth.pa@XEROX.ARPA
Status: R
```

This is the promised set of ideas about the planned operating system. Please distribute copies to FP, HE, andJG. I am curious to have your reactions. Please confirm receipt of this message and the last one about the M2 compiler.
Stack overflow: I suggest as a compromise to offer a built-in function StackSpace(), giving the possibility to program a check at places considered crucial. The stack limit could be stored at 4(SB). Then the inline-code would be:

```
   SPRD SP, Ri
   SUBD Ri, 4(SB)
```

Process transfers will have to take care of setting the proper stack limit. Do you need advice about how to incorporate it? Best regards
NW.

Thoughts about a new operating system
N. Wirth 23.2.85
The following are some ideas about the new operating system planned for the computer based on the NS 32000 processor, which I shall now call Flintstone. The ideas are unordered and sometimes incoherent, but I hope consistent. They are to be taken as suggestions to be thought about. A lot needs to be added. Much of what follows was inspired by the use of Cedar, but I try to avoid the hideous complexities in its mechanism and usage.
One major difference between Medos and (I guess we need a name here; Logos, Simos, ... ?) is multitasking: When a program is started by the command tool, not just a procedure (on a next "level") is created, but a new process. This has strong consequences on resource management: it must be centralized in the kernel. The resources are:

- Processor
- Memory (units = areas)
- Disk (units = files)
- Screen (units = viewers)
- Devices.

Another major difference is that the network is integrated in the kernel.
User interface and screen management.
The command tool, the user interface of the OS, opens a viewer (window). Each created process may itself open viewers. The screen manager allocates new viewers according to parameters taken as hints. A standard pop-up menu in the title bar displays commands to change size and position, or to close the viewer. A client is unaware of its activation.
The contents of a viewer are managed by the process owning the viewer. Pop-up menus appearing inside the viewer are created by calling a procedure ShowMenu. A client asks for a next event through a Read procedure (merging of mouse and keyboard input?). The left mouse button sets a caret. Keyboard input is directed to the process in whose viewer the caret resides.
The screen is divided into a narrower left part (say 448*800) and a wider right part (say 576*800). The right part has the format of an A4

document. The left part permanently contains a viewer for the command tool and an area (not a viewer?) holding a table of closed viewers. If a viewer is closed (not terminated), its screen area is freed and its name (from the title bar) is entered into that table. This table assumes the role of the icon row in Cedar, and uses less space. Selecting the name (icon) reopens the viewer (by stealing space from others). A call for the creation of a viewer has parameters: indicating left or right side (special case: full screen?), the desired initial height, approximately, a repainting procedure, and the viewer's kind. I envision basically three kinds of viewers.

1. The viewer has a title bar and a scroll bar. The change of the cursor look when entering the scroll bar is handled by the screen manager; the client (typically an editor) is unaware of it. Scrolling involves repainting by calling a procedure Repaint, supplied by the client when the viewer was created.
2. The viewer has a title bar and a scroll bar. The contents are an unformatted text backed up by a file. Thus repainting upon scrolling is done by the manager accessing that file. No repainting procedure is supplied. Typically, this viewer shows the output of a program.
3. There is no scrollbar nor does the manager have any information about the viewer's contents. Hence no scrolling and repainting is done.

The editor appears as always present. It has access to viewers of the first two kinds, and can select information in those of kind 1 which were created by it and in all those of kind 2. Writing in viewers of kind 2 is sequential, however. So they appear as "non-editable" to the user. The standard font is a proportional font (say TR10 or TR12).

Storage management

Compared to Medos, storage management is more complex, because of the concurrent processes, but also simpler because of the homogeneous, linear address space supported by a memory mapping unit in hardware (MMU). Although this opens the door to a truly virtual memory, I do not recommend automatic transfers to and from backing store, not even for program code. A property of Cedar that annoys me considerably is that execution speed drops towards the evening. The reason is that code is automatically overlaid, so thrashing increases during a work session, because no code is ever deallocated. It is a very peculiar property of Cedar indeed that the basic operation of deletion is practically unknown for programs, for data, and even for files. (When I "delete" a message from the mail system, it actually is transferred to a list of "deleted messages"!)

The storage manager must provide the means for obtaining and for returning areas (which need not be physically contiguous because of the MMU). It seems to me that the managed units should not be too small. Cedar's general garbage collection mechanism is quite heavy, and in most cases one does not need that fine grained management at the "automatic level". (Can the MMU be used to give protection against stack overflow?)

Processor management

All process switching must be handled by the kernel. Device-driven processes must have priority. It is not necessary but may be desirable to introduce "soft" priorities, adapting the Medos scheme. Deadlock always poses a problem. We cannot assume fully deadlock-safe programs; so it should always be possible to break a deadlock. Some Cedar system programs occasionally cause a deadlock, and then the macine's boot button is the only recourse. That should not be.

File management

Here I see the least reason to depart radically from Medos, although the availability of much larger disks requires a more sophisticated directory structure and search scheme. We should plan for disks of up to 150 MByte capacity. Perhaps Farrell Ostler's system could be adopted? We should certainly make us of it as much as possible. Recoverability is an ongoing concern. Should each disk sector contain the file and sector numbers of the file to which it belongs?

Compatibility with Lilith and Medos

Although basically compatibility is provided by the common use of Modula-2, different sets of OS modules with different concepts and procedures can destroy the portability of programs written in the same language. But since some of the ideas sketched differ from the world of Medos, we cannot aim for total upward compatibility. How far can the base modules Terminal, FileSystem, Heap, Windows be adopted? The first task is to define a client interface at about this level. It must be our goal to be able to host previously written programs, perhaps after they had been adapted to the new system according to a reasonably small set of transformation rules.

I have just received and read with great interest your "Thoughts about a new operating system".
Hereafter I shall try to sketch out my own opinion on the subject.
You may well regard these explanations as comments to your suggestions.
If I judge correctly, our ideas don't actually diverge. Please take into consideration that (this time) I prefer quickly returning the ball to formulating well-ordered and well-balanced statements.
This implies both a sometimes moderate linguistic style and a rather disordered presentation of my ideas, disordered from a "vertical" as well as from a "horizontal" point of view.
I recently read the article "A Tour through Cedar" in IEEE Software, and I was impressed by both Cedar's general concepts and its complexity. The article stimulated me to think about the topic of integrated (operating) systems and programming environments. I must, however, admit that the depth of the resulting and subsequently outlined thoughts is strongly limited by the (rather chaotic) external circumstances under which I had to complete the previous Semester. Nevertheless, I hope that the following explanations will animate the discussion.

In retrospect, I can summarize my opinion about Cedar as follows:
1) As its most important aspects, I consider Cedar's multitasking or multiactivity concept and the permanent availability of the editor. To preclude any misunderstanding, it is not the possibly parallel execution of processes (background actions) which impresses me so much, but rather the possibility of having several activities simultaneously alive, more precisely the fact that a new activity can be started at any time and that it does not have to be terminated before a previous activity can be resumed. A most desirable consequence of this organization is the disappearance of "global modes". A "global mode" is entered whenever a program in Medos is being called. I regard this as a consequent extension of the road to avoiding "local modes", for example in editors.
Concerning the permanence of the editor, I would go even one step further than Cedar does. In fact, I would count the editor to the class of "services" rather than to the class of "activities". As a consequence, the editor (even a text formatter like Tioga) should not have to be explicitely "started" each time. Instead, upon opening a rectangle (within a viewer) of an appropriate type, an object of the respective type (for example unformatted or formatted text or even a figure) should become ready to be edited. Clearly, the class of possible object types could depend on the current system state or on the system configuration.

2) I believe that much of the complexity of Cedar has one of two origins (I might be wrong here): The inclusion of "automatisms" or "artificial intelligence" (this name is probably too pompous for what I mean) and a sometimes exaggerated pictorial representation of items (icons, scrollbars, marks etc.) on the display (this confirming Weizenbaums thesis "Verkindlichung der Welt der Erwachsenen").
In fact, I personally prefer a meaningful name or a simple symbol to identify (the state of) an object. As an example of "automatism" I mention the program editor which automatically sets up statement skeletons and indents automatically. Ironically enough this automatism neutralizes all efforts to free programmers from column boundedness like in old assemblers and COBOLs. By the way, would an artist accept an analogous help when he creates a painting? In the contrary, I suggest to enable the programmer to make profitable use of the greater variety of character styles and attributes which have been introduced with bitmap displays and printers.
Further examples of "automatisms" are the DWIM-concept (what do I mean with that?) and - if a date can be detected within a message - the automatic transfer of this message from the mailbox to the agenda (and, when the time has come, automatic ringing the alarm bell). A last (well-known to us) example may illustrate the increase in complexity of a data-structure, if just a touch of "artificial intelligence" comes into play. In fact, I came to the conclusion that the greater complexity of the Tioga editor compared to Lara is caused by the ambition to reflect a documents structure as regards content. Then, of course, a tree of a variable number of levels is mandatory. In contrast, the data-structure of Lara restricts on formatting aspects and completely disregards contents. Formatting aspects can be represented by exactly four universal levels.
My general opinion is that a system should optimally support its user in his manipulations, but, as the word states, let him do them by hand.

3) A third source of complexity of the Cedar system is its interpretative mode. In fact, I am not enthusiastic about such a mode. While I strongly

defend the permanence of the editor, I decline equally strongly the permanent presence of a debugger. Debugging should remain a timely limited activity instead of becoming a permanent state of affairs. Moreover, I got the impression that our debuggers present the relevant data in a much more convenient way than the Cedar debugger.

Back to our plans about a new operating system. In the light of the above summary of my point of view, I would propose a model similar to the following for a future operating system:

There exist services and activities. (Internally, both kinds are realized as programs). Services include local input-output handling (including keyboard, mouse (replaying mechanism?) and windows), editing, command interpreting and network communication. Dedicated servers (printer, mass-storage) may offer additional services.
These services are permanently active. Possibly, their current power depends on the system state or on the system configuration. The system state can implicitly change or explicitely be changed.
In contrast, activities (or activity-units) are started by pointing to their identification (a name or a symbol) on the display.
If an activity demands an input, this input has preliminarily been created in an arbitray window and must be selected at the time of starting the activity (comparable to a text selection when starting an edit-command). Both the identification item of the activity and the input have been created with the editor. It is, of course, desirable that the output of an activity can serve as input to the next activity (system integration).
During the execution of an activity, at well-defined points (it would be sufficient, if activities could be interrupted at places which separate logical activity-units) any new activity can be started and any old activity can be resumed.
I shall conclude this draft with a few comments on windows, menus and the mouse. I think that the current module Windows could well serve as the base of a new window system. However, I propose to remove all routines which concern the interior of windows (i.e. routines to draw frames and titlebars). A window as handled by this basic module should just be a rectangle (for example the full screen). The new module Windows should export the current dimension (x,y,w,h) of all active windows (for example the full screen).
In principle, I would not completely do without overlapping. However, I agree with you that the window allocation should be (better) supported by the window system (window base module). It is further recommandable (and in fact easy) to adjust the restore-algorithm such that it does not call the restore procedure for windows which are totally overlapped by others. Taking into account the large memory of new machines, one might discuss, if a new type "temporary window" should be introduced. Instances of this type would differ from normal windows solely in the fact that the space covered by such a window is saved as a bitmap (and can therefore later be restored quickly by the basic window module itself).
Besides from the basic window handler, the window system should include a collection of service modules which handle the interior of windows, for example a module which exports procedures to draw titlebars, scrollbars and further window attributes, a module which exports procedures to draw patterns, arbitrary lines and curves, and a module which exports procedures to write text in default and arbitrary style etc. (By the way: if a proportional font should become the default (a non-negligable increase in complexity!), then I strongly stand for a Meier-font instead of Timesroman!)
I think that a shortcoming of the current window package is the fact that (when using standard modules) each window must be denounced as text-window or graphic-window etc. In the new system, a window should not be confiscated by any module except the window base.
Personally, I like pop-up menus (as we have them now) better than the "buttons" of Cedar. Buttons occupy valuable space and may lead to an overfeeding of the user with (unused) information. In combination with the possible dependency on the window (zone), our existing pop-up menus are sufficiently powerful. Thus, the menu module could directly be taken over.
The mouse module should be slightly adjusted. First, the translation from mouse-coordinates to screen-coordinates can be both simplified and made more flexible. Simplification can be achieved by eliminating the "barrier-effect". As the mouse coordinate-range will be enlarged with the new machine, the translation routine may depend on a scaling-factor which indicates the translation ratio and thus controls the speed of the cursor when moving the mouse around.

Many thanks for your help both to find an apartment and to improve
my Lara paper. The money is on the way to you (to your Xerox address).
Could Xerox let me know what I have to do to complete the processing
of my visa! Kind regards