EUROPEAN PATENT SPECIFICATION

(45) Date of publication and mention of the grant of the patent:

(21) Application number: 90313048.2

(22) Date of filing: 30.11.1990

(54) Architecture for server extension
Architektur für Datenanbietungs nebenteil
Architecture pour poste serveur

(84) Designated Contracting States:
DE FR GB IT

(30) Priority: 30.11.1989 US 444728

(43) Date of publication of application:
05.06.1991 Bulletin 1991/23

(73) Proprietor: DIGITAL EQUIPMENT CORPORATION
Maynard, Massachusetts 01745 (US)

(72) Inventors:
• Annichiarico, Richard Francis
  Penacook, NH 03303 (US)
• Chesler, Robert Todd
  Nashua, NH 03063 (US)
• Jamison, Alan Quentin
  Milford, NH 03055 (US)

(74) Representative: Maggs, Michael Norman et al
Kilburn & Strode
30 John Street
London WC1N 2DD (GB)

(56) References cited:
• IBM TECHNICAL DISCLOSURE BULLETIN. vol. 25, no. 11B, April 1983, NEW YORK US pages 6003 - 6006; J.D. DIXON ET AL.: 'Device for collection of highly detailed statistics concerning usage of I/O devices'
• IBM TECHNICAL DISCLOSURE BULLETIN. vol. 19, no. 8, January 1977, NEW YORK US pages 2874 - 2876; A.J. HEIMSOETH ET AL.: 'On-line error and statistics logging in large database systems'
• EDN ELECTRICAL DESIGN NEWS. vol. 31, no. 4, February 1986, NEWTON, MASSACHUSETTS US pages 189 - 198; L. GOSS ET AL.: '1/4Cs with on-chip EEPROM provide system adaptability'
• IBM TECHNICAL DISCLOSURE BULLETIN. vol. 28, no. 12, May 1986, NEW YORK US pages 5573 - 5574; 'Collect host/ network traffic at VTAM control point'
• COMPUTER. vol. 8, no. 11, November 1975, LONG BEACH US pages 51 - 61; G.J. NUTT ET AL.: 'Tutorial: Computer system monitors'

Note: Within nine months from the publication of the mention of the grant of the European patent, any person may give notice to the European Patent Office of opposition to the European patent granted. Notice of opposition shall be filed in a written reasoned statement. It shall not be deemed to have been filed until the opposition fee has been paid. (Art. 99(1) European Patent Convention).
Description

Field of the Invention

This invention relates generally to architecture for a server extension used in a computer system and in particular to an architecture for a server extension for intercepting input events and protocol requests generated by an application program. The intercepted input events and protocol requests are subsequently used for any purpose, for example determining the time lapse between the input event and the protocol request for evaluating the performance of the server and/or the application program.

Background

The server is the part of a computer system's architecture that functions as the interface between the computer or central processing unit and the user. It is desirable to test or evaluate the performance of the server for various purposes. For example, long delays between a user input and the system response can be detected and subsequently eliminated. Similarly, it is desirable to test and evaluate application programs running on the central processor unit of the computer for many purposes.

Common methods of performance testing a server involve the use of video cameras or stop watches. In one such method the user or operator working at a terminal and the terminal screen are filmed by a video camera. Thereafter the video is replayed on a machine that displays time as a function of the film speed. Performance metrics are then calculated based upon the difference between the displayed time of a starting film event and an ending film event. The performance metrics for each timed event are then recorded. In another method, the step of calculating performance metrics comprises manually timing response times with a stop watch. In either method, the performance metrics for all of the timed events are then analyzed to determine the performance of the applications or server. This same approach is also used to determine the performance of the server for a single or specific input event. This procedure is time-consuming, tedious, and prone to error. Furthermore, this procedure requires actual operation by the user at a workstation. Thus, simulated user input is impossible. Furthermore, since actual user input is inconsistent from one operation to the next, it is impossible to have consistent input from run to run.

Another way of testing the performance of an application program or a server in response to input events or protocol requests is with remote terminal emulation (RTE). RTE dispenses with a live user or operator for sending input to the server or the application program and replaces the operator with software generated input events which are loaded onto the server input event queue. However, prior attempts at RTE on servers of the X window type systems have failed to provide a way to intercept output generated protocol requests by the server or the application. Furthermore, none of these attempts have supported receiving simulated input from a remote source, i.e., a source that sends input over a non-X transport link, such as network or asynchronous terminal line. Thus, these prior attempts at RTE cannot test system configurations involving networks, modems and other peripheral devices.

IBM Technical disclosure bulletin, Vol. 25, No. 11B, April 1983 New York, US, pages 6003-6008, by J. D. Dixon teaches a computer system having an input/output control unit. A statistical data collection unit corresponding to a server extension is attached to the input/output unit. Upon receipt of data, the input/output control unit executes a microprogram routine to handle the data. This document discloses the features set out in the preamble to claim 1.

SUMMARY OF THE INVENTION

The present invention pertains to architecture for a server extension used in a computer system, and in its broad form resides in apparatus and method as recited in claims 1 and 2 respectively. An extension is a software module that performs server functions and has access to the server variables but is not a permanent part of the server. The server extension intercepts user input events before they are received by the application program (also called a client), writes or stores information about the input, including the type of input and the time it was intercepted, into a client defined location. The client defined location can be any type of storage such as a sequential file. The storage can be local or at a remote location and be connected over a modem line, a terminal line or a network line. Furthermore, the server extension intercepts output protocol requests generated by the application program before they result in visible changes to the screen display of a terminal. An output protocol request is a signal generated by an application program and sent to the server, which causes the server to draw text or graphics on the screen of a terminal. The server extension also writes or stores information about the output protocol request including the type of request and the time it was intercepted, into a client defined location. The client defined location can be any type of storage such as a sequential file. The storage can be local or at a remote location and connected over a modem line, a terminal line or a network line. Finally, the server extension is capable of receiving input from user operated input devices such as the mouse or keyboard of a workstation or simulated input from the controlling client.
The server extension architecture as described herein intercepts input events and protocol requests which are used to enable the controlling client to monitor a server or the application program or to control a workstation for example testing the performance of applications, system configurations, and user interfaces. The data structures of the server extension architecture are configured in a portion or block of memory and mimic or are substantially identical to the data structures of the server which are configured in a different portion or block of memory. In addition the server extension architecture of the present invention is operated under the control of the client or application program. Thus, while the server extension architecture is described in relation to the well known X server operating system program, it can also be implemented with any other operating system.

The server extension architecture also provides RTE, remote terminal emulation and is capable of receiving simulated input from a remote source. For example the server extension architecture provides RTE capability for the X Window System which is a well known operating system documented in R.W. Scheifler, J. Gettys, and R. Newman, X Window System. Digital Press (1986). The server extension architecture also provides a quick, inexpensive means for software regression testing, and competitive or interoperability testing of applications among different hardware and software platforms. The server extension architecture also facilitates computer-based instructions where it is important to monitor a user's interaction at a terminal and be able to demonstrate correct interactions. It further facilitates trade show demonstrations where the demonstrator need not understand the application being demonstrated or needs to concentrate on the customer rather than the demonstration. As is apparent from the above examples the server extension architecture of the present invention can be used for a wide variety of purposes which will be apparent to those skilled in the field.

BRIEF DESCRIPTION OF THE DRAWINGS

A more detailed understanding of the invention may be had from the following description of a preferred embodiment, given by way of example and to be understood in conjunction with the accompanying drawing wherein:

FIG. 1 shows the general operation of the server extension of a preferred embodiment of this invention in use with a computer system.

FIG. 2 shows the data structures of the server extension of this invention as well as the data structures of the X server which the extension has access to.

FIG. 3 is a process flow diagram showing an application program implementation of the server extension in one mode of operation.

FIG. 4 is a process flow diagram showing an application program implementation of the server extension in another mode of operation.

DESCRIPTION OF THE PREFERRED EMBODIMENT

FIG. 1 shows the general operation of the server extension architecture of a preferred embodiment of the present invention in use with basic components of a computer system. A software configuration runs on a central processing unit that interacts with an X server 12. The X server is a well known operating system program that controls a user interface of the terminal or work station 14 and utilizes windows 16 and a mouse 18 in addition to standard user interface elements such as a keyboard 20 or text. The X server 12 can support "extensions," or software modules that have access to X server variables, but which are not permanent parts of the X server. The extension 22 of this embodiment is referred to as the "XTrap extension." Those skilled in the art will know how to apply the teachings of this embodiment to other operating systems and servers.

A plurality of application programs 24A through 24N can be run on the central processor unit (CPU) 10. One application program or client 24N running on the CPU 10 acts as the controlling client of the XTrap extension 22, which is a software extension of the X server 12. Known software modules or extensions that perform operating system functions are controlled by the server, the XTrap extension 22 is controlled by a single application program or client 24N. The controlling client 24N sets the parameters of the XTrap extension 22 and turns the XTrap extension 22 on and off. User input from the mouse 18 or the keyboard 20 is initially received by the X server 12. The present invention is also applicable to operating systems with other input devices, as well as to operating systems with only one type of input device. The XTrap extension 22 intercepts the input, reformats a copy of the input for use by its "write" routine, as more fully set forth below, and sends the original input to the server 12 for normal processing. Ultimately, the client 24A to 24N-1, the client being monitored or controlled, receives the original input and handles it according to its particular implementation.

Any client 24A through 24N-1 generates output in the form of protocol requests, usually to control some kind of screen output at the workstation 14. The controlling client 24N can simultaneously display the same output on a screen at another workstation (not illustrated). These output protocol requests cause the server to draw text, lines or other graphic elements on the display screen of the workstation 14. Although in the preferred embodiment the only commu-
nations between the application programs and the X server that XTrap intercepts are output protocol requests, the present invention could be used to intercept any communication between any client and the server. The X server 12 receives output protocol requests from an application program 24A-24N-1, and the XTrap extension 22 intercepts them before the X server processes them. As with input, the XTrap extension reformats a copy and sends the original back to the X server. Ultimately, the output reaches the screen of the workstation 14 to control the form of text, lines or erasures.

The XTrap extension 22 calls a "write" subroutine that is part of the code of the extension. XTrap has a different "write" routine for every communication channel over which XTrap can communicate with the controlling client. The controlling client can be at a remote location from XTrap, such as at the other end of a network line. This "write" routine sends the collected information on inputs and outputs to the controlling client 24N or to a sequential file.

Fig. 2 shows the architecture of the XTrap extension and the server. The present invention is not limited to use with an architecture of this type, but will work on any operating system architecture that calls subroutines to process input and protocol request. The X server processes keyboard data by sending it to the routine whose address is stored in the "Keyboard ProcessInputProc" storage location 28. The server handles mouse input in a similar way, i.e., by referencing the "Pointer ProcessInputProc" storage location 30. The X server processes output protocol requests by sending the appropriate information to the routine whose address is stored in a specified element of the "ProcVector" array 32.

The XTrap extension contains data structures that are one half the size as the X server data structures, however the size can be smaller or even identical. The XTrap data structures are also laid out in memory in the same or substantially the same configuration as the X server's data structures. A person skilled in the art will know how to use the teachings of this invention to provide means for mimicking the architecture of another operating system so that they call extension subroutines for processing input (and/or output) instead of its own, while at the same time providing a means for returning the operating system to its normal state after the use of the extension is completed. When XTrap is configured for intercepting keyboard input, the X server's storage location, "Keyboard ProcessInputProc" 28 contains the address of the XTrap routine that handles keyboard input, and the XTrap version of this storage space, called "keyboard_inputproc" 34, stores the address of the X server's input-handling routine. If XTrap is configured for intercepting mouse-input, the X-server's storage location, "Pointer ProcessInputProc" 30 contains the address of the XTrap routine that handles mouse input, and the XTrap version of this storage space, called "pointer-inputproc" 36, stores the address of the X-server's input handling routine.

As can be seen from the above configuration, XTrap will intercept keyboard and mouse input. As explained above, when the X server receives keyboard input, it calls the routine whose address is stored in "Keyboard ProcessInputProc" 28, and when it receives mouse input, it calls the routine whose address is stored in "Pointer ProcessInputProc" 30. Since during the implementation of XTrap these storage locations 28 and 30 contain the addresses of XTrap routines, the keyboard and mouse input is sent to the XTrap extension.

The XTrap extension intercepts output protocol requests by swapping the address of the XTrap extension output handling routine with the address of the X server output handling routine stored in data structure 32, referred to as "ProcVector". Each 4-byte value is an address of a routine that handles an output protocol request. In the preferred embodiment there are 256 4-byte addresses, but it is within the scope of the present invention to have smaller or larger arrays. Array element 42 contains the address for the routine that handles an "ImageText" request. This routine draws a single string of text on the screen. If XTrap is configured for intercepting the "ImageText" output protocol request, the value stored at this location is the address of an XTrap routine that draws a single string of text on the screen. This routine is called "ext_imagegettext" B. The address of the "ImageText" routine is stored at the corresponding element 44 of the "ext_procvector" array 38 which is XTrap's version of the X server's "ProcVector" data structure 32. Thus, when the X server receives a request from any client to draw a string of text, it calls the routine that has its address stored in the element 42 of "ProcVector" 32. While XTrap is active that address will be the address of the XTrap routine, "ext_imagegettext" B normally stored at element 44.

Other output protocol requests include for example, "PolyText", which draws more than one string of text on the screen, "MapWindow", which prepares the server for drawing a window on the screen, and "UnmapWindow", which erases windows from the screen. The corresponding routines in the XTrap data structure 38 are "ext_polygettext", "ext_unmapwindow" and "ext_mapwindow". Any other output protocol request can be intercepted by the XTrap extension in a similar fashion.

When the XTrap extension intercepts input or output, it creates a copy of the input or output information that is formatted for processing by XTrap's "write" routine that the client has configured. While XTrap is turned "on", the address of the "write" routine is stored in the "xtrap_write" vector 40. When XTrap is turned "off", "xtrap_write" holds the address of a null routine, i.e., a routine that does nothing but return control to XTrap. In either case, if XTrap is configured to intercept any input or output, it sends a copy of formatted input or output to the routine whose address is stored in "xtrap_write".

The data structure "ext_environment" 48 contains the XTrap internal variables. The variable "swap/*Gstate" is a
bit field which indicates which output protocol requests and input events are being intercepted. The "genflgs" variable is a bit field of general extension flags that enable and disable various features such as timestamp calculation, "window_id" processing, and output format (i.e., binary or ASCII). "Window_id" processing, when enabled, causes XTrap to report which window is the object of a "MapWindow" or "UnMapWindow" output protocol request. The variable "osflgs" is a bit field of implementation specific flags. The variable "imagcnt" holds the maximum number of characters that XTrap saves from a text string when XTrap intercepts an "ImageText" or a "PolyText" request. The saved characters are part of the information on output protocol requests that XTrap sends to the "write" routine. If "imagcnt" is negative, the characters are saved from the beginning of the string; if "imagcnt" is positive, the characters are saved from the end of the string.

The variable "polcnt" holds the maximum number of strings XTrap should process when XTrap intercepts a "PolyText" request. When the strings are processed, "imagcnt" characters are sent to the "write" routine. If "polcnt" is positive, strings are processed from the beginning of the string list; if "polcnt" is negative, strings are processed from the end of the string list. The variable "L_comm" contains an identifier for the communication channel being used for simulated input to XTrap. The variable "a_comm" contains an identifier for the communication channel being used for output from the XTrap extension. Finally, the variable "writeIo" holds the pointer to the "write" routine that is placed in "xtrap_write" when XTrap receives a START_IO request.

FIGS. 3 and 4 are process flow diagrams showing an application program or controlling client configuration and use of the XTrap server extension in two different modes of operation. FIG. 3 is a process flow diagram showing the operation of the XTrap server extension when recording input events or output protocol requests are generated during a live user session referred to as the record mode of operation. FIG. 4 is a process flow diagram showing the operation of the XTrap server extension using the RTE, remote terminal emulation capabilities to simulate a live user session using simulated input events and waiting for intercepted output events referred to as the playback mode of operation. The controlling client communicates with the XTrap extension by sending a data structure to the server via the XLIB xflush macro, which is documented in R.W. Scheffler, J. Gettys, and R. Newman, - X Window System, published by Digital Press (1988). The server then sends the data structure to the extension. Table 1 below shows the format of the data structure "xXtrapReq".

### Table 1

The xXtrapReq data structure (short version)

```c
typedef struct _xXtrapReq // 9 longwords in size */
{
    CARD8 reqtype; // Extension Major opcode */
    BYTE minor_opcode; // Minor opcode of request */
    CARD16 length; // Initialized with 0 */

    union
    { union of various request data structure }
}
```

The first field, "reqType," identifies which XTrap extension the "xXtrapReq" data structure should be passed to. The client collects this identifier by calling "xQueryExtension," which is documented in R.W. Scheffler, J. Gettys, and R. Newman, entitled X Window System of Digital Press (1988). The second field, "minor_opcode," contains the identifier for the specific XTrap extension request the client is sending. Referring again to FIG. 2, the "ext_request_vector" array 46 contains a dispatch table that XTrap uses to look up the "minor_opcode" and to call the appropriate request processing routine. The third field, "length" is always 9 for XTrap requests. The fourth field, "data" is a union configured differently for each request. It contains the data specific to the request being sent. The present invention is not limited to the use
of xflush for communicating with the extension.

Referring again to FIG. 3, the first step a client takes in configuring XTrap is to obtain the extension identifier by calling "xQueryExtension," which is documented in R.W. Schaeffer, J. Gettys, and R. Newman, entitled X Window System, of Digital Press (1988). Next, the client requests the XTRAP_RESET request by setting the various fields of the "xTrapReq" structure and sending it to the server via the xflush macro. The XTRAP_RESET request causes the extension to prepare itself for use by a new client. If a client is already using the extension and has previously configured itself as the controlling client, XTrap will check the "force" flag in the request. The "force" flag is part of the request-specific information that is sent to XTrap in the "RESET" union field of "xTrapReq." (See Table 2)

Table 2
The xTrapReq data structure

(long version - showing contents of the "data" union

```c
typedef struct_xTrapReq
{ /* 9 longwords in size */
    CARD8 reqType; /* Extension Major opcode */
    BYTE minor_opcode; /* Minor opcode of request */
    CARD16 length Bl6; /* Initialized with 9 */
    union
    { /*
        BYTE buffer[32]; /* General Buffer */
        RESET reset; /* Data for XTRAP_RESET */
        INFORM inforeq; /* Data for XTRAP_INFO */
        CONFIG config; /* Data for XTRAP_CONFIG */
        IO io; /* Data for CONFIG_IO */
        START start; /* Data for START_IO */
        XINPUT xinput; /* Data for SIMULATE_XEVENT */
        TCP_IO tcp_io; /* Data for CONFIG_IO */
        DMX_IO dnet_io; /* Data for CONFIG_IO */
    }
    data;
} xTrapReq;
```

If the "force" flag is set, XTrap will reset itself and remove itself from the control of the other client. If the "force" flag is not set, XTrap will return an error code to the client.

A client can prevent another client from gaining accidental control of an extension by setting the "control" flag during an XTRAP_RESET request. Setting the "control" flag causes the XTrap extension to ignore any subsequent XTRAP_RESET requests from another client unless the other client sets the "force" flag. The "control" flag, like the "force" flag is a part of the "RESET" union.

When the XTrap extension receives an XTRAP_RESET request, it resets XTrap's internal variables and flags to their default values, and undoes any configuration for intercepting input or output. All client programs send an XTRAP_RESET before they stop.

Once the XTrap extension is reset, the client sends an XTRAP_CONFIG request to set up the kind of input and
output the extension is to intercept. For the XTRAP_CONFIG request, the "data" union of xXTrapReq* has the structure shown in Table 3.

Table 3
The CONFIG union

#define MAX_LIST 20

typedef struct _config /* used by XTRAP_CONFIG */
{
    FLAGS  flags;
    BITS   bits;
    CARD8  opcode_count;
    CARD8  pad_byte;
    INT16  pad_word;
    CARD8  list[MAX_LIST];
} CONFIG;

The "flags" field is used to enable and disable input and output interception and is shown in Table 4.
typedef struct flags {
    union {
        unsigned int mask;
        struct {
            /*
             * This word is used in a request to specify
             * the types of configuration information that
             * should be changed or updated. It is used
             * as a valid data check.
             */
            unsigned v_kbd :1; /* keyboard input */
            unsigned v_ptr :1; /* pointer input */
            unsigned v_OPCODES :1; /* output array */
            unsigned v_gen :1; /* general */
            unsigned v_os :1; /* specific (<16) */
            unsigned v_poly :1; /* poly count */
            unsigned v_image :1; /* image count */
            unsigned pad0_9bits :9; /* pad to short */
        } bit;
    } use;
} FLAGS;

Setting the "v_kbd" flag causes XTrap to set the state of keyboard input interception according to the state of the "s_kbd" flag. If "s_kbd" is on, keyboard input interception is enabled. It is off, keyboard-input interception is disabled.
Mouse input is similarly controlled with the "v_ptr" and the "s_ptr" flags. The pair of flags, "v_opcodes" and "s_opcodes", act in the same way to enable or disable the interception of output protocol requests listed in the "list" field of the "CONFIG" structure shown in Table 3.

The XTRAP_CONFIG request can also configure the extension to take a millisecond timestamp of any intercepted input or output protocol request. XTRAP_CONFIG also provides some control over the form of output generated by XTrap, namely over the choice between ASCII format and binary format.

When the XTrap extension receives an XTRAP_CONFIG request, it performs the swap operation that exchanges the address of the XTrap input or output processing routine with the address of the X server's input or output processing routine, for each type of input and output protocol request selected by the client. For example, if the client sets flags "v_kbd" and "s_kbd" in its XTRAP_CONFIG request, then, referring to FIG. 2, XTrap will exchange the "KbdProc" pointer in vector "Keyboard ProcessInputProc" for the pointer "xtrap_keyboard" in "keyboard inputproc". If the client sets flag "v_kbd" but leaves flag "s_kbd" unset, the two pointers would be swapped. The XTRAP_RESET request also causes XTrap to unswap "Keyboard ProcessInputProc" and "xtrap_keyboard".

Next, the client sends a CONFIG_IO request to XTrap to specify over what kind of communication channel to send information on collected input and output to the controlling client. When XTrap receives this request, it sets its internal variable "write_io" to the value of a pointer to a "write" routine. This routine is used to send the information on intercepted input events and output protocol requests to the controlling client via the configured communication channel. In record mode, the client records the information in a sequential file. In playback mode, the controlling client uses the information received via the configured "write" routine to verify that simulated input was sent to the server. This allows the controlling client to ensure that the server will ignore input from the keyboard or the mouse when, for security or other reasons, this is necessary. The client also uses the information to monitor output protocol requests to ensure that it waits to send the next input event until after the last output protocol request has been processed. In playback mode the controlling client may also record the information to a sequential file. For example, if the last output protocol request was a request for a "$" prompt, the client must wait until the "$" appears on the screen before it sends the next input event.

In playback mode, the CONFIG_IO request is made to implement Remote Terminal Emulation (RTE). Table 5 shows the structure of the "data" union of "xTrapReq" for the CONFIG_IO request.
```c
#define IO_BUFFER_SIZE 28

typedef struct _io /* used by CONFIG_IO */
{
  CARD8 transport;
  CARD8 direction;
  CARD16 pad;
  BYTE buffer[IO_BUFFER_SIZE];
} IO;

/*
 * Values for use in the direction field.
 * The communication mode specified in the
 * transport field will be used to configure
 * the input and/or output channels specified
 * in the direction field.
 */
#define IO_DIR_IN  (1<<0) /* Configure for input */
#define IO_DIR_OUT  (1<<1) /* Configure for output */
#define IO_DIR_BOTH (IO_DIR_IN | IO_DIR_OUT)
```

The "transport" field contains a decimal value describing the type of communication channel being configured. Examples of communication channels include asynchronous terminal lines, sequential files, TCP/IP networks, VMS mailbox connection, XLIB transport and DECnet Task-to-Task Communication. Input sent via XLIB transport is sent via the xflush macro using a SIMULATE_XEVENT request or the MOVE_POINTER request, discussed below. It can be seen that by using sequential files as the source for keyboard and mouse input, an application program can be performance tested without a live user. Furthermore, software regression and comparative testing are simplified. The "direction" field specifies whether the CONFIG_IO request applies to input, output or both.

Once the XTrap extension is configured, the client makes the START_IO request/Get to start I/O over the communication media configured with the CONFIG_IO request and to initiate the "write" routine that sends input and output information to the controlling client. The only data specified with this request are (1) the number of characters the extension should save from an intercepted text string that is part of an output protocol request, and (2) the number of text strings the extension should process when the extension intercepts "PolyText8", output protocol request that request the server to draw more than one string of text on the screen. (See Table 6).
Table 6
The START union

```c
typedef struct _start
{
    INT16 imaq_count;
    INT16 poly_count;
} START;
```

When the controlling client has completed its remote control or monitoring of a workstation that requires XTrap implementation, the client should send a STOP_IO request to the XTrap server extension via the xflush macro. This causes character flow to and from the IO channels to stop. Before the client is ready to stop execution, it should send an XTRAP_RESET request to the XTrap extension in order to unswap any input or output processing routines.

The controlling client can send the XTRAP_INFO request whenever the controlling client needs to know the values of any internal variables.

The controlling client can use the SIMULATE_XEVENT and the MOVE_POINTER requests in playback mode to send input to the X server via XLIB. Table 7 shows the format of the *data* union of the *XTrapReq* structure for use with SIMULATE_XEVENT and MOVE_POINTER. SIMULATE_XEVENT is used to send keyboard events and mouse click events.

Table 7
The XINPUT union

```c
typedef struct _xinput
{
    CARD8 type;
    CARD8 detail;
    CARD16 x;
    CARD16 y;
    CARD16 pad_s;
} XINPUT;
```

MOVE_POINTER is used to send mouse motion events. The variable "type" is set to the value of an identifier for the type of event being sent, which, in the case of a MOVE_POINTER request, is a mouse motion event. The variable "detail" is set for the value of an identifier for the key or mouse button pressed or released in a SIMULATE_XEVENT request. The variables "x" and "y" are set to the destination screen coordinates of the mouse cursor.

The foregoing description of the invention has been presented for purposes of illustration and description. It is not
intended to limit the invention to the precise forms disclosed, and obviously many modifications and variations are possible and envisaged in light of the above teachings.

5

Claims

1. A server extension (22) comprising a software module, for use in a computer system having a host computer, including a server for executing application programs, (24A, 24N) a user interface (14) for displaying text or graphic figures, at least one application program executing on the host computer for generating output protocol requests, a controlling application program (24N) acting on the host computer to generate inputs and extension protocol requests, the server (12) comprising a portion of memory having addressable locations and a plurality of server routines each having an address, the server (12) having performance characteristics that are variable over a range, the server (12) receiving the inputs, transferring the inputs to the application program, receiving the output protocol requests and transferring the output protocol requests to the user interface (14), the server extension (22) being characterised by:

an extension portion of memory directly connected to the controlling application program (24N) for creating a record of the input and output data for use in evaluating whether the performance characteristics are within an acceptable range by monitoring one of the server (12) and the application program;

the extension portion of memory having specific memory locations for data storing that correspond to specific memory locations in the server portion of memory;

a plurality of extension routines each having an address, each of the extension routines corresponding to one of the server routines;

the addresses for the plurality of server routines being stored in the extension portion of memory and the addresses for the plurality of extension routines being stored in the server portion of memory;

the inputs and the output protocol requests each being directed to one of the server routines;

the extension (22) receiving the extension protocol requests from the controlling application program (24N) for configuring and controlling said extension; and

the extension (22) intercepting the inputs and the output protocol requests caused to be generated by the controlling application program (24N) and redirecting the inputs and the output protocol requests to one of the extension routines, said one of the extension routines corresponding structurally to the server routine to which the inputs and output protocol requests are directed, the extension reformating a copy of the inputs and the output protocol requests for monitoring the server (12), for monitoring the performance of the application program or for formatting display of the text or graphic figures on the user interface (14).

2. A method of operating a server extension (22) comprising a software module in a computer system having a user interface (14) for displaying text or graphic figures, a host computer, including a server (12) for executing application programs (24A, 24N), at least one application program executing on the host computer for generating output protocol requests, a controlling application program (24N) executing on the host computer to cause it to generate inputs and extension protocol requests, the server (12) comprising a portion of memory and having performance characteristics that are variable over a range, the server (12) having a plurality of server routines each having an address, the inputs and the output protocol requests each being directed to one of the server routines, the server extension being characterised by an extension portion of memory directly connected to the controlling application program, the server portion of memory having specific memory locations for data storing that correspond to specific memory locations in the extension portion of memory, the server extension (22) having a plurality of extension routines each having an address, each of the extension routines corresponding structurally to one of the server routines, the method being further characterised by:

receiving the extension protocol requests from the controlling application program (24N);

configuring the extension portion of memory in response to the extension protocol requests;

receiving the inputs and the output protocol requests from the server (12);

redirecting the inputs and the output protocol requests to one of the extension routines, said one of the extension routines corresponding to the server routine to which the inputs and output protocol requests are directed;

reformatting a copy of the inputs and the output protocol requests for monitoring the server (12), for monitoring the performance of the application program or for formatting display of the text or graphic figures on the user interface (14); and

creating a record of input and output data for use in evaluating whether the performance characteristics are within an acceptable range.
Patentansprüche

1. Servererweiterung (22), die ein Softwaremodul aufweist, zur Verwendung in einem Computersystem mit einem Hostcomputer, einschließlich eines Servers zum Ausführen von Anwenderprogrammen (24A, 24N), einer Benutzerschnittstelle (14) zum Anzeigen von Text- oder Graphikzeichen, wobei wenigstens ein Anwenderprogramm eine Ausführung auf dem Hostcomputer zum Erzeugen von Ausgabe-Protokollaufrufen durchführt, wobei ein steuerndes Anwenderprogramm (24N) auf dem Hostcomputer zum Erzeugen von Eingaben und Erweiterungsprotokollaufrufen handelt, wobei der Server (12) einen Teil eines Speichers mit adressierbaren Stellen und einer Vielzahl von Serverprogrammen aufweist, die jeweils eine Adresse haben, wobei der Server Leistungseigenschaften hat, die über einen Bereich variabel sind, wobei der Server (12) die Eingaben empfängt, die Eingaben zum Anwenderprogramm überträgt, die Ausgabe-Protokollaufrufe empfängt und die Ausgabe-Protokollaufrufe zur Benutzerschnittstelle (14) überträgt, wobei die Servererweiterung (22) gekennzeichnet ist durch:

- einen Erweiterungssteil eines Speichers, der direkt mit dem steuernden Anwenderprogramm (24N) verbunden ist, zum Erzeugen einer Aufzeichnung der Eingabe- und Ausgabedaten zur Verwendung bei einer Beurteilung, ob die Leistungseigenschaften innerhalb eines akzeptierbaren Bereichs sind, und zwar durch Überwachen entweder des Servers (12) oder des Anwenderprogramms;
- wobei der Erweiterungssteil des Speichers spezifische Speicherstellen für eine Datenspeicherung hat, die den spezifischen Speicherstellen im Serverteil des Speichers entsprechen;
- eine Vielzahl von Erweiterungsprogrammen, die jeweils eine Adresse haben, wobei jedes der Erweiterungsprogramme einem der Serverprogramme entspricht;
- wobei die Adressen für die Vielzahl von Serverprogrammen im Erweiterungssteil des Speichers gespeichert sind, und wobei die Adressen für die Vielzahl von Erweiterungsprogrammen im Serverteil des Speichers gespeichert sind;
- wobei die Eingaben und die Ausgabe-Protokollaufrufe jeweils zu einem der Serverprogramme gerichtet sind;
- wobei die Erweiterung (22) die Erweiterungsprotokollaufrufe vom steuernden Anwenderprogramm (24N) zum Konfigurieren und zum Steuern der Erweiterung empfängt; und
- wobei die Erweiterung (22) die Eingaben und die Ausgabe-Protokollaufrufe abfangt, die verursacht sind, durch das steuernde Anwenderprogramm (24N) erzeugt zu werden, und die Eingaben und die Ausgabe-Protokollaufrufe erneut zu einem der Erweiterungsprogramme richtet; wobei das eine der Erweiterungsprogramme strukturiert, dem Serverprogramm entspricht, zu dem die Eingaben und Ausgabe-Protokollaufrufe gerichtet sind, wobei die Erweiterung eine Kopie der Eingaben und der Ausgabe-Protokollaufrufe zum Überwachen des Servers (12), zum Überwachen der Leistungsfähigkeit des Anwenderprogramms oder zum Formatieren einer Anzeige der Text- oder Graphikzeichen an der Benutzerschnittstelle (14) neu formatiert.

2. Verfahren zum Betreiben einer Servererweiterung (22), die ein Softwaremodul in einem Computersystem mit einer Benutzerschnittstelle (14) zum Anzeigen von Text- oder Graphikzeichen aufweist, einen Hostcomputer einschließlich eines Servers (12) zum Ausführen von Anwenderprogrammen (24A, 24N), wobei wenigstens ein Anwenderprogramm eine Ausführung auf dem Hostcomputer zum Erzeugen von Ausgabe-Protokollaufrufen durchführt, wobei ein steuerndes Anwenderprogramm (24N) eine Ausführung auf dem Hostcomputer durchführt, um ihn zu verarbeiten, Eingaben und Erweiterungsprotokollaufrufe zu erzeugen, wobei der Server (12) einen Teil eines Speichers aufweist und Leistungseigenschaften hat, die über einen Bereich variabel sind, wobei der Server (12) eine Vielzahl von Serverprogrammen aufweist, die jeweils eine Adresse haben, wobei die Eingaben und die Ausgabe-Protokollaufrufe jeweils zu einem der Serverprogramme gerichtet sind, wobei die Servererweiterung durch einen Erweiterungssteil eines Speichers gekennzeichnet ist, der mit dem steuernden Anwenderprogramm verbunden ist, wobei der Serverteil des Speichers spezifische Speicherstellen zur Datenspeicherung aufweist, die spezifischen Speicherstellen dem Erweiterungssteil des Speichers entsprechen, wobei die Servererweiterung (22) eine Vielzahl von Erweiterungsprogrammen hat, die jeweils eine Adresse haben, wobei jedes der Erweiterungsprogramme strukturiert, einem der Serverprogramme entspricht, wobei das Verfahren weiterhin gekennzeichnet ist durch:

- Empfangen der Erweiterungsprotokollaufrufe vom steuernden Anwenderprogramm (24N);
- Konfigurieren des Erweiterungssteils des Speichers in Antwort auf die Erweiterungsprotokollaufrufen;
- Empfangen der Eingaben und der Ausgabe-Protokollaufrufe vom Server (12);
- erneutes Richten der Eingaben und der Ausgabe-Protokollaufrufe zu einem der Erweiterungsprogramme, wobei das eine der Erweiterungsprogramme dem Serverprogramm entspricht, zu dem die Eingaben und die Ausgabe-Protokollaufrufe gerichtet sind;
- erneutes Formatieren einer Kopie der Eingaben und der Ausgabe-Protokollaufrufe zum Überwachen des Ser-
EP 0 430 708 B1

vers (12), zum Überwachen der Leistungsfähigkeit des Anwenderprogramms oder zum Formatieren einer Anzeige der Text- oder Graphikzeichen auf der Benutzerschnittstelle (14); und Erzeugen einer Aufzeichnung von Eingabe- und Ausgabedaten zur Verwendung bei einer Beurteilung, ob die Leistungscharakteristiken innerhalb eines akzeptierbaren Bereichs sind.

Revendications

1. Extension de serveur (22) comprenant un module logiciel, pour une utilisation dans un système informatique ayant un ordinateur hôte, incluant un serveur pour exécuter des programmes d’application (24A, ..., 24N), une interface d’utilisateur (14) pour afficher du texte ou des graphiques, au moins un programme d’application s’exécutant sur l’ordinateur hôte pour produire des requêtes de protocole de sortie, un programme d’application de commande (24N) opérant sur l’ordinateur hôte pour produire des entrées et des requêtes de protocole d’extension, le serveur (12) comprenant une partie de mémoire ayant des emplacements adressables et une pluralité de sous-programmes de serveur ayant chacun une adresse, le serveur (12) ayant des caractéristiques de performances qui sont variables sur une plage, le serveur (12) recevant les entrées, transférant les entrées vers le programme d’application, recevant les requêtes de protocole de sortie et transférant les requêtes de protocole de sortie vers l’interface d’utilisateur (14), l’extension de serveur (22) étant caractérisée par:

- une partie d’extension de mémoire directement connectée au programme d’application de commande (24N) pour produire un enregistrement des données d’entrée et de sortie destiné à être utilisé pour évaluer si les caractéristiques de performances se trouvent à l’intérieur d’une plage acceptable en surveillant le serveur (12) ou le programme d’application;
- la partie d’extension de mémoire ayant des emplacements de mémoire spécifiques pour un stockage de données qui correspondent à des emplacements de mémoire spécifiques dans la partie de serveur de mémoire;
- une pluralité de sous-programmes d’extension ayant chacun une adresse, chacun des sous-programmes d’extension correspondant à l’un des sous-programmes de serveur;
- les adresses pour la pluralité de sous-programmes de serveur étant mémorisées dans la partie d’extension de mémoire et les adresses pour la pluralité de sous-programmes d’extension étant mémorisées dans la partie de serveur de mémoire;
- les entrées et les requêtes de protocole de sortie étant dirigées chacune vers l’un des sous-programmes de serveur;
- l’extension (22) recevant les requêtes de protocole d’extension provenant du programme d’application de commande (24N) pour configurer et commander ladite extension; et
- l’extension (22) interceptant les entrées et les requêtes de protocole de sortie qui sont produites par le programme d’application de commande (24N) et réacheminant les entrées et les requêtes de protocole de sortie vers l’un des sous-programmes d’extension, l’extension correspondant structurellement au sous-programme de serveur vers lequel les entrées et les requêtes de protocole de sortie sont dirigées, l’extension reformatant une copie des entrées et des requêtes de protocole de sortie pour surveiller le serveur (12), pour surveiller les performances du programme d’application ou pour formater l’affichage du texte ou des graphiques sur l’interface d’utilisateur (14).

2. Procédé pour faire fonctionner une extension de serveur (22) comprenant un module logiciel dans un système informatique ayant une interface d’utilisateur (14) pour afficher du texte et des graphiques, un ordinateur hôte, incluant un serveur (12) pour exécuter des programmes d’application (24A, ..., 24N), au moins un programme d’application s’exécutant sur l’ordinateur hôte pour produire des requêtes de protocole de sortie, un programme d’application de commande (24N) s’exécutant sur l’ordinateur hôte afin de l’amener à produire des entrées et des requêtes de protocole d’extension, le serveur (12) comprenant une partie de mémoire et ayant des caractéristiques de performances qui sont variables sur une plage, le serveur (12) ayant une pluralité de sous-programmes de serveur ayant chacun une adresse, les entrées et les requêtes de protocole de sortie étant dirigées chacune vers l’un des sous-programmes de serveur, l’extension de serveur étant caractérisée par une partie d’extension de mémoire directement connectée au programme d’application de commande, la partie de serveur de mémoire ayant des emplacements de mémoire spécifiques pour un stockage de données qui correspondent à des emplacements de mémoire spécifiques dans la partie d’extension de mémoire, l’extension de serveur (22) ayant une pluralité de sous-programmes d’extension ayant chacun une adresse, chacun des sous-programmes d’extension correspondant structurellement à l’un des sous-programmes de serveur, le procédé étant caractérisé également par les étapes consistant à:
recevoir les requêtes de protocole d'extension provenant du programme d'application de commande (24N);
configurer la partie d'extension de mémoire en réponse aux requêtes de protocole d'extension;
recevoir les entrées et les requêtes de protocole de sortie provenant du serveur (12);
réachemiser les entrées et les requêtes de protocole de sortie vers l'un des sous-programmes d'extension,
ledit un des sous-programmes d'extension correspondant au sous-programme de serveur vers lequel les en-
trées et les requêtes de protocole de sortie sont dirigées;
reformer une copie des entrées et des requêtes de protocole de sortie pour surveiller le serveur (12), pour
surveiller les performances du programme d'application ou pour formater l'affichage du texte ou des graphi-
ques sur l'interface d'utilisateur (14); et
produire un enregistrement de données d'entrée et de sortie destiné à être utilisé pour évaluer si les caracté-
ristiques de performances se trouvent à l'intérieur d'une plage acceptable.
Fig. 1
Fig. 2
CONTROLLING CLIENT PREPARES TO SEND RESET REQUEST

XTRAP CONTROLLED?

YES

FORCE ON?

NO

SEND ERROR TO CONTROLLING CLIENT

NO

RESET INTERNAL VARIABLES
DE-CONFIGURE PREVIOUS I/O INTERCEPTION

CONFIGURE I/O INTERCEPTION SWAP POINTERS

CONFIGURE COMMUNICATION CHANNEL, SELECT "WRITE" ROUTINE

TURN XTRAP "ON"

USER ENTERS INPUT AT MOUSE AND KEYBOARD

XTRAP INTERCEPTS INPUT CALLS "WRITE"; RETURNS INPUT TO SERVER

APPLICATION RECEIVES INPUT; Generates OUTPUT PROTOCOL REQUESTS

XTRAP INTERCEPTS OUTPUT CALLS "WRITE"; RETURNS OUTPUT TO SERVER

NO

HAS USER QUIT APPLICATION?

YES

TURN XTRAP "OFF";
RESET XTRAP

STOP

Fig. 3
CONTROLLING CLIENT Prepares to send reset request

XTRAP Controlled?

FORCE ON?

SEND ERROR TO
CONTROLLING CLIENT

NO

RESET INTERNAL
VARIABLES
DE-CONFIGURE PREVIOUS
I/O INTERCEPTION

CONFIGURE I/O INTERCEPTION
SWAP POINTERS

CONFIGURE COMMUNICATION
CHANNEL, SELECT "WRITE"
ROUTINE

TURN XTRAP "ON"

CONTROLLING CLIENT SENDS
SIMULATED INPUT TO SERVER

XTRAP INTERCEPTS INPUT;
VERIFY INPUT SOURCE;
RETURN INPUT TO SERVER IF
VALID

APPLICATION RECEIVES
INPUT; GENERATES OUTPUT
PROTOCOL REQUESTS

XTRAP INTERCEPTS OUTPUT
CALLS "WRITE"; RETURNS
OUTPUT TO SERVER

HAS APPLICATION
FINISHED?

NO

YES

STOP

TURN XTRAP "OFF";
RESET XTRAP

Fig. 4