

Exhibit 28

Office Action in Ex Parte Reexamination	Control No. 90/011,311	Patent Under Reexamination 5,915,131	
	Examiner Woo H. Choi	Art Unit 3992	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

- a Responsive to the communication(s) filed on 01 November 2010. b This action is made FINAL.
c A statement under 37 CFR 1.530 has not been received from the patent owner.

A shortened statutory period for response to this action is set to expire 2 month(s) from the mailing date of this letter. Failure to respond within the period for response will result in termination of the proceeding and issuance of an *ex parte* reexamination certificate in accordance with this action. 37 CFR 1.550(d). **EXTENSIONS OF TIME ARE GOVERNED BY 37 CFR 1.550(c)**. If the period for response specified above is less than thirty (30) days, a response within the statutory minimum of thirty (30) days will be considered timely.

Part I THE FOLLOWING ATTACHMENT(S) ARE PART OF THIS ACTION:

1. Notice of References Cited by Examiner, PTO-892. 3. Interview Summary, PTO-474.
2. Information Disclosure Statement, PTO/SB/08. 4. _____.

Part II SUMMARY OF ACTION

- 1a. Claims 1-20 are subject to reexamination.
1b. Claims _____ are not subject to reexamination.
2. Claims _____ have been canceled in the present reexamination proceeding.
3. Claims 11 and 14 are patentable and/or confirmed.
4. Claims 1-10, 12, 13 and 15-20 are rejected.
5. Claims _____ are objected to.
6. The drawings, filed on _____ are acceptable.
7. The proposed drawing correction, filed on _____ has been (7a) approved (7b) disapproved.
8. Acknowledgment is made of the priority claim under 35 U.S.C. § 119(a)-(d) or (f).
a) All b) Some* c) None of the certified copies have
1 been received.
2 not been received.
3 been filed in Application No. _____.
4 been filed in reexamination Control No. _____.
5 been received by the International Bureau in PCT application No. _____.
* See the attached detailed Office action for a list of the certified copies not received.
9. Since the proceeding appears to be in condition for issuance of an *ex parte* reexamination certificate except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte* Quayle, 1935 C.D. 11, 453 O.G. 213.
10. Other: _____

cc: Requester (if third party requester)

DETAILED ACTION

Reexamination

1. This is an *ex parte* reexamination of U.S. Patent Number 5,915,131 ('131 patent) requested by a third party requester. Claims 1-20 are subject to reexamination. The references discussed herein are as follows:

1. Teaff, Danny, et. al, "*The Architecture of High Performance Storage Systems (HPSS)*," 4th NASA Goddard Conference on Mass Storage Systems and Technologies, March 29-30, 1995 ("Teaff");
2. U.S. Patent No. 5,566,346 ("Andert").

Claim Rejections - 35 USC § 102

2. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

3. Claims 1-6 are rejected under 35 U.S.C. 102(a) as being anticipated by Teaff.

4. With respect to claim 1, Teaff discloses **a computer system** (p. 3, Figure 1) **comprising:**
a bus (Figure 1, control buses that connect network elements and workstations to the HPSS servers; also, a bus is an inherent component of a computer system);

at least one memory (Figure 1, network attached memories or other memories in servers and workstations) **coupled to the bus for storing data and programming instructions that include applications and an operating system; and**

a processing unit (Figure 1, servers and workstations) **coupled to the bus and running the operating system and applications by executing programming instructions** (see pp. 4-5, Modularity and APIs, and Portability and Standards), **wherein an application has a first plurality of tailored distinct programming interfaces** (p. 4, “The HPSS software components are loosely coupled, with open application program interfaces (APIs) defined at each component level”; see also Appendix A for a list of tailored APIs) **available to access a plurality of separate sets of computer system services** (p. 4, “Most users will access HPSS at its high level interfaces – client API, FTP, (both parallel and sequential), NFS, Parallel File system (PFS), with AFS/DFS, ...”) **through the operating system of the computer system via service requests** (see Figures 2 and 3, and the text associated with the figures; see Figure 6 for an example of security service requests handled through the operating system).

5. With respect to claim 3, see the discussion of claim 1 above. Teaff discloses that **the operating system comprises a plurality of servers** (see Figures 1 and 2, HPSS Servers, name servers, location servers, etc.; see also Appendix A), **and each of the first plurality of programming interfaces transfer service requests to one of the plurality of servers, wherein each of the plurality of servers responds to service requests from clients of the separate sets of I/O services** (see p. 4, Modularity and APIs and Figure 2; Appendix A).

6. With respect to claim 2, **each of the first plurality of tailored distinct programming interfaces are tailored to a type of I/O service provided by each set of I/O services** (p. 4, Modularity and APIs; see also Appendix A, APIs are tailored for each type of services).
7. With respect to claim 4, **service requests are transferred as messages in a messaging system** (p. 8, "HPSS uses the DCE Remote Procedure Call (RPC) mechanism for control messages and DCE Threads for multitasking").
8. With respect to claim 5, **each of the plurality of servers supports a message port** (see p. 10, Communication, "The control path communication between HPSS components is through DCE RCPs or Encima transaction RPCs. For data path communication, the HPSS Mover(s) currently utilize either Sockets or IPI-3 (over HIPPI) libraries.").
9. With respect to claim 6, **at least one of the plurality of servers is responsive to service requests from applications and from at least one other set of I/O services** (see p. 14, Physical Volume Library, "The volume mount service is provided to clients such as a Storage Server").
10. Claims 1-10, 12-13, and 15-20 are rejected under 35 U.S.C. 102(a) as being anticipated by Andert.
11. With respect to claim 1, Andert discloses **a computer system** (Figure 10) comprising: **a bus** (see Figures 4, 7, and 8);

at least one memory (Figure 10, 1008) coupled to the bus for storing data and programming instructions that include applications and an operating system; and a processing unit (1006) coupled to the bus and running the operating system (1014) and applications (1030) by executing programming instructions, wherein an application has a first plurality of tailored distinct programming interfaces (see Figure 1 and c8:30-44) available to access a plurality of separate sets of computer system services (c8:45-67) through the operating system (see Figure 5, applications 1030 access I/O services provided by Device Ensembles 1032 through the operating system 1014; see also c3:47-49) of the computer system via service requests (c1:59-62).

12. With respect to claim 3, see the discussion of claim 1 above. Andert discloses that **the operating system comprises a plurality of servers** (see Figures 1 and 2, IO services frameworks), **and each of the first plurality of programming interfaces transfer service requests to one of the plurality of servers, wherein each of the plurality of servers responds to service requests from clients of the separate sets of I/O services** (see Figure 1, c10:1-6).

13. With respect to claim 2, **each of the first plurality of tailored distinct programming interfaces are tailored to a type of I/O service provided by each set of I/O services** (see Figure 1; see also c8:45-67).

14. With respect to claim 4, **service requests are transferred as messages in a messaging system** (c10:1-6 and c17:36-40).

15. With respect to claim 5, **each of the plurality of servers supports a message port** (see c17:36-53, the message port limitation reads on the means for receiving messages from application programs and interrupt service means).

16. With respect to claim 6, **at least one of the plurality of servers is responsive to service requests from applications and from at least one other set of I/O services** (c9:1-11).

17. With respect to claim 7, **the operating system further comprises a plurality of activation models, wherein each of the plurality of activation models is associated with one of the plurality of servers to provide a runtime environment for the set of I/O services to which access is provided by said one of the plurality of servers** (c11:25-58).

18. With respect to claim 8, **at least one instance of a service is called by one of the plurality of servers for execution in an environment set forth by one of the plurality of activation models** (c11:25-58).

19. With respect to claim 9, Andert discloses **a computer system comprising:**
a bus (see claim 1 above);
at least one memory coupled to the bus for storing data and programming instructions that comprise applications and an operating system (see claim 1 above);

a processing unit coupled to the bus and running the operating system and applications by executing programming instructions, wherein the operating system provides computer system services through a tailored distinct one of a plurality of program structures (see claim 1 above), each tailored distinct program structure comprising:

a first programming interface for receiving service requests for a set of computer system I/O services of a first type (see s 1 and 3 above),

a first server coupled to receive service requests and to dispatch service requests to the computer system I/O services (see claim 3 above),

an activation model to define an operating environment in which a service request is to be serviced by the set of computer system I/O services (see claim 7 above), and

at least one specific instance of the set of computer system I/O services that operate within the activation model (see claim 8 above).

20. With respect to claim 10, the first programming interface is responsive to request from applications and from other program structures (see claim 6 above).

21. With respect to claim 12, the first server receives a message corresponding a service request from the first programming interface, maps the message into a function called by the client, and then calls the function (c17:36-54, subroutines in the I/O service framework are executed to provide data to the application program in response to the messages received).

22. With respect to claim 13, the message comprises a kernel message (c17:41-46).

23. With respect to claim 15, **two or more I/O services share code or data** (c9:1-11, code or data for SCSI service is shared by the SCSI framework and the Mass-Storage framework).
24. With respect to claim 16, **said two or more I/O services are different types** (see Figure 1).
25. With respect to claim 17, **the program structure further comprises a storage mechanism to maintain identification of available services to which access is provided via the first server** (c14:30-56).
26. With respect to claim 18, **a computer implemented method of accessing I/O services of a first type, said computer implemented method comprising the steps of:**
- generating a service request for a first type of I/O services** (see claims 1, 3, and 9 above);
 - a tailored distinct family server, operating in an operating system environment and dedicated to providing access to service requests for the first type of I/O service, receiving and responding to the service request based on an activation model specific to the first type of I/O services** (see claims 1, 3, and 9 above); **and**
 - a processor running an instance of the first type of I/O services that interfaces to the file server to satisfy the service request** (see claims 1, 3, and 9 above).

Art Unit: 3992

27. With respect to claim 19, **the service request is generated by an application** (see claim 7 above).

28. With respect to claim 20, **the service request is generated by an instance of an I/O service running in the operating system environment** (see claim 7 above).

Examiner's Statement of Reasons for Patentability/Confirmation

29. Claims 11 and 14 are deemed to be patentable and/or confirmed over the prior art of record for the following reasons:

30. Claim 11 recites the limitation **"the first programming interface comprises at least one library for converting functions into messages."** Requester alleges that Andert discloses the limitation at c10:1-6 ("A client 214 sends requests for services to a specific IO service framework 102 associated with the client. The IO service framework 102 uses its associated access manager 210 to load an appropriate device register with an appropriate command, such as "write byte," "buffer a block," or whatever the appropriate action might be.") (see Request, p. 67). Requester asserts that "[t]hese commands, "write a byte" or "buffer a block," must reside in a library." However, Requester provides no basis for this assertion. Andert discloses receiving messages from application programs. In contrast, the claim requires the programming interface to convert functions into messages (see the '131 patent specification, Figure 3, where a procedure call from an application 302 is converted into a message by the library 303 at the interface 301).

31. Claim 14 recites the limitation "said one of said at least one specific instances communicates to said another program structure of a second type using a message created using a library sent to the server of said another program structure." Andert does not disclose a message that is created using a library sent to the server.

Service of Papers

32. After filing of a request for ex parte reexamination by a third party requester, any document filed by either the patent owner or the third party requester must be served on the other party (or parties where two or more third party requester proceedings are merged) in the reexamination proceeding in the manner provided in 37 CFR 1.248. The document must reflect service or the document may be refused consideration by the Office. See 37 CFR 1.550(f).

Extensions of Time

33. Extensions of time under 37 CFR 1.136(a) will not be permitted in these proceedings because the provisions of 37 CFR 1.136 apply only to "an applicant" and not to parties in a reexamination proceeding. Additionally, 35 U.S.C. 305 requires that *ex parte* reexamination proceedings "will be conducted with special dispatch" (37 CFR 1.550(a)). Extensions of time in *ex parte* reexamination proceedings are provided for in 37 CFR 1.550(c).

Litigation Reminder

34. The patent owner is reminded of the continuing responsibility under 37 CFR 1.565(a) to apprise the Office of any litigation activity, or other prior or concurrent proceeding, involving the

Art Unit: 3992

patent throughout the course of this reexamination proceeding. The third party requester is also reminded of the ability to similarly apprise the Office of any such activity or proceeding throughout the course of this reexamination proceeding. See MPEP §§ 2207, 2282 and 2286.

Art Unit: 3992

All correspondence relating to this *ex parte* reexamination proceeding should be directed as follows:

By U.S. Postal Service Mail to:

Mail Stop *Ex Parte* Reexam
ATTN: Central Reexamination Unit
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

By FAX to: (571) 273-9900
Central Reexamination Unit

By hand to: Customer Service Window
Randolph Building
401 Dulany St.
Alexandria, VA 22314

Any inquiry concerning this communication or earlier communications from the Reexamination Legal Advisor or Examiner, or as to the status of this proceeding, should be directed to the Central Reexamination Unit at telephone number (571) 272-7705.

/Woo H. Choi/
Reexamination Specialist
Central Reexamination Unit 3992

EOK



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Reexamination of
U.S. Patent No. 5,915,131
to KNIGHT, *et al.*

Reexam Control No. 90/011,311
Filed November 1, 2010

For: **METHOD AND APPARATUS
FOR HANDLING I/O REQUESTS
UTILIZING SEPARATE
PROGRAMMING INTERFACES TO
ACCESS SEPARATE I/O SERVICES**

Confirmation No.: 1355

Art Unit: 3992

Examiner: CHOI, Woo

Atty. Docket No. 20142.0003.RXUS00

Mail Stop Ex Parte Reexamination
Central Reexamination Unit
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Patent Owner's Interview Summary

Patent Owner thanks the Examiner for the courtesies extended during the May 27, 2011 Examiner Interview. A discussion took place in which the Patent Owner argued that Andert did not disclose a tailored distinct programming interface; that neither reference disclosed a plurality of servers as part of the operating system; and that Andert's Device Access Manager does not define an execution environment, but only regulates device access. The discussion focused on the meaning of claim terms. Patent Owner acknowledged the broadest reasonable interpretation standard, but pointed out that claim terms still must have a meaning that is reasonable in light of the specification, and that it is that understanding of the claim limitation that must be disclosed in the prior art to constitute anticipation. No agreement was reached.

* * * * *

The undersigned representative requests any extension of time that may be deemed necessary to further prosecution of this application.

The undersigned representative authorizes the Commissioner to charge any additional fees under 37 C.F.R. § 1.16 or 1.17 that may be required, or credit any overpayment, to Deposit Account 14-1437, referencing Attorney Docket No. 20142.0003.RXUS00.

In order to facilitate the resolution of any issues or questions resented by this paper, the Examiner may directly contact the undersigned by phone to further discussion.

Respectfully submitted,

/tracy w. druce/

Tracy W. Druce, Esq.
Reg. No. 35,493

Brian K. McKnight, Esq.
Reg. No. 59,914

C. Gideon Korrell, Esq.
Reg. No. 60,131

Novak, Druce + Quigg LLP
1000 Louisiana Street, Suite 5300
Houston, Texas 77002
(713) 571-3400
(713) 456-2836 (fax)
tracy.druce@novakdruce.com

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Reexamination of
U.S. Patent No. 5,915,131
to KNIGHT, *et al.*

Reexam Control No. 90/011,311

Filed November 1, 2010

For: **METHOD AND APPARATUS
FOR HANDLING I/O REQUESTS
UTILIZING SEPARATE
PROGRAMMING INTERFACES TO
ACCESS SEPARATE I/O SERVICES**

Confirmation No.: 1355

Art Unit: 3992

Examiner: CHOI, Woo

Atty. Docket No. 20142.0003.RXUS00

Mail Stop Ex Parte Reexamination
Central Reexamination Unit
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

**Patent Owner's Response to Non-Final Office Action
Under 37 C.F.R. §§ 1.111 and 1.550 in *Ex Parte* Reexamination**

This paper is filed in response to the Non-Final Office Action mailed April 11, 2011 ("Office Action") in the above-captioned *ex parte* reexamination proceeding setting forth a shortened statutory period of two months in which to respond. Accordingly, this paper is timely filed on or before June 13, 2011 which is the next succeeding secular or business day which is not a Saturday, Sunday, or a Federal holiday after Saturday, June 11, 2011. 35 U.S.C. 21.

I. Status of Claims

Claims 1-20 of U.S. Patent No. 5,915,131 (the '131 patent) are pending in this reexamination proceeding. Claims 1-10, 12, 13, and 15-20 stand rejected, and claims 11 and 14 are confirmed. In this response no claims have been amended. Patent Owner submits that all claims are in condition for confirmation, and therefore requests reconsideration and issuance of a Notice of Intent to Issue a Reexamination Certificate (NIRC) confirming all claims.

II. Summary of Principal Arguments

There are fundamental differences between the invention claimed in the '131 patent and the Andert and Teaff references that form the basis of the Examiner's rejections. **First**, the Andert reference does not disclose, teach, or suggest "a first plurality of tailored distinct programming interfaces available to access a plurality of separate sets of I/O services provided through the operating system via service requests." **Second**, neither the Andert reference nor the Teaff reference discloses, teaches, or suggests "an operating system compris[ing] a plurality of servers." **Third**, the Andert reference does not disclose, teach, or suggest an "operating system [that] further comprises a plurality of activation models." **Fourth**, the Teaff reference does not disclose, teach, or suggest "[a] computer system comprising: a bus; at least one memory coupled to the bus . . .; and a processing unit coupled to the bus."

III. All Rejected Claims are Patentable Over the Andert and Teaff References

In the Office Action, claims 1-6 were rejected under 35 U.S.C. § 102 as being anticipated by both the Andert and Teaff references. Additionally, claims 7-10, 12-13, and 15-20 were rejected under 35 U.S.C. § 102 as being anticipated by Andert only. However, neither of the references anticipates the claims because neither reference

discloses every element recited in the claims.^{1,2} Accordingly, Patent Owner respectfully traverses the rejections.

A. Andert Does Not Disclose “a first plurality of tailored distinct programming interfaces”

Claim 3 recites, *inter alia*, “wherein an application has a first plurality of tailored distinct programming interfaces available to access a plurality of separate sets of I/O services provided through the operating system via service requests.” The relevant section of the Office Action is provided below for reference:

wherein an application has a first plurality of tailored distinct programming interfaces (see Figure 1 and c8:30-44) available to access a plurality of separate sets of computer system services (c8:45-67) through the operating system (see Figure 5, applications 1030 access I/O services provided by Device Ensembles 1032 through the operating system 1014; see also c3:47-49) of the computer system via service requests (c1:59-62).³

The Office Action asserts that Andert anticipates the claimed tailored distinct programming interfaces through Andert’s Figure 1 and at col. 8, lines 30-44, where it shows and describes various I/O Service Frameworks. However, the I/O Service Frameworks in Andert are not a *tailored* distinct programming interface for accessing computer system services as recited in the claim because Andert does not disclose that the interfaces to the I/O System Framework are tailored.

¹ *Verdegaal Bros. v. Union Oil Co. of California*, 814 F.2d 628, 631 (Fed. Cir. 1987) (“A claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference.”); *Bristol-Myers Squibb Co. v. Danbury Pharm., Inc.*, 26 F.3d 138 (Fed. Cir. 1994) (“There must be no difference between the claimed invention and the anticipating reference as viewed by a person of ordinary skill in the art.”) (citing *Scripps Clinic & Research Fdn. v. Genetech, Inc.*, 927 F.2d 1565, 1576 (Fed. Cir. 1991)); *Richardson v. Suzuki Motor Co.*, 868 F.2d 1226, 1236 (Fed. Cir. 1989) (“The identical invention must be shown in as complete detail as is contained in the . . . claim.”); MPEP 2131.

² Also, an ambiguous prior art reference is not anticipatory. *Mitsubishi Chemical Corp. v. Barr Labs., Inc.*, 718 F. Supp.2d 382,415-16 (citing *Eli Lilly & Co. v. Zenith Goldline Pharms., Inc.*, 364 F. Supp. 2d 820, 901 (S.D. Ind. 2005) (citing *In re Brink*, 57 C.C.P.A. 861, 419 F.2d 914, 918 (1970)), *aff’d*, 471 F.3d 1369 (Fed. Cir. 2006); *In re Turlay*, 49 C.C.P.A. 1288, 304 F.2d 893, 899 (1962) (“It is well established that an anticipation rejection cannot be predicated on an ambiguous reference.”); *In re Hughes*, 52 C.C.P.A. 1355, 345 F. 2d 184, 188 (1965) (“[A]n ambiguous reference . . . will not support an anticipation rejection.”).

³ Ctrl. No. 90/011,311, Office Action, mailed Apr. 11, 2011, at 5 (bold in original).

The plurality of I/O service frameworks disclosed in the Andert reference appear to provide applications access to the various I/O services offered through the operating system. The examples of I/O services disclosed by Andert include:

- (a) Mass Storage I/O services;
- (b) Keyboard processing services;
- (c) Mouse/Pointing device processing services;
- (d) SCSI services;
- (e) Serial communications port services;
- (f) Expansion bus management services;
- (g) Desktop bus I/O services;
- and (h) Power management services.⁴

By providing access to these operating system services, each of the I/O service frameworks presumably include interfaces, which might be considered distinct, but are not disclosed as being tailored, as those terms are used in the '131 patent.

The term "tailored" is used in both the claim and the specification without any special meaning beyond its dictionary definition.⁵ In the context of claim 3, "tailored" refers to a programming interface⁶ to a family of I/O services that is customized to meet the particular needs of that family.⁷ Such an interpretation is consistent with both the original prosecution of the application that matured into the '131 patent^{8,9} and the '131 patent's specification.^{10, 11}

Specifically, the '131 patent states:

⁴ Andert at 8:31-37.

⁵ See Exhibit B - The definition of "tailor" is "to cut, form, produce, alter, etc. so as to meet requirements or particular conditions; as her novel is *tailored* to popular tastes." WEBSTER'S NEW TWENTIETH CENTURY DICTIONARY 1858 (2d ed. 1983).

⁶ An "application programming interface" is referred to in the '131 patent specification as "FPI" or "family programming interface."

⁷ Serial No. 08/435,677, Response, dated Sept. 2, 1997, at 4-5.

⁸ Serial No. 08/435,677, Response, dated Sept. 2, 1997, at 2-5.

⁹ Serial No. 08/435,677, Office Action, mailed Jan. 2, 1998, at 3.

¹⁰ The '131 patent at 6:20-36.

¹¹ The Federal Circuit has recently explained that claim interpretation during patent reexaminations are only reasonable if consistent with the specification:

As a starting point, the proper interpretation of a claim limitation in reexamination must be assessed. While it is true that claim language is required to be given its broadest reasonable interpretation, it is also true that the interpretation must be 'consistent with the specification, . . . and that claim language should be read in light of the specification as it would be interpreted by one of ordinary skill in the art.'

In re Suitco Surface, 603 F.3d 1255, 1260 (Fed. Cir. 2010) (citing *In re Bond*, 910 F.2d 831, 833 (Fed.Cir.1990) (quoting *In re Sneed*, 710 F.2d 1544, 1548 (Fed.Cir.1983))).

For example, when an application generates data for a video device, a display FPI tailored to the needs of video devices is used to gain access to display services. Likewise, when an application desires to input or output sound data, the application gains access to a sound family of services through an FPI. Therefore, the present invention provides family programming interfaces tailored to the needs of specific device families.¹²

Andert, however, does not disclose a programming interface that is tailored to the needs of the associated service. Andert is silent regarding the details of any interface made available by its I/O service frameworks beyond disclosing that they “represent an end user’s interface to the I/O system,”¹³ and similar statements, which do not anticipate the claimed “tailored distinct programming interfaces.”

In fact, it is probable that Andert’s interface to its I/O service frameworks was not tailored. In the accompanying Declaration, David Wilson, Ph.D. explains that leading up to and at the time of the ‘131 patent’s filing, typical interfaces to I/O services were limited to only a generic set of commands.¹⁴

However, regardless of the reason, Andert does not provide any description of its I/O service frameworks that can be considered a disclosure of “tailored distinct programming interfaces.” As anticipation requires the reference to disclose each and every feature of the claim, Andert does not anticipate. Accordingly, Patent Owner submits that the rejection should be withdrawn.

While it has not been alleged in the current rejection, Patent Owner nevertheless notes that Andert does not anticipate the claim through any other application of the reference. Patent Owner notes that none of the other interfaces present in Andert could be considered to anticipate the claimed “tailored distinct programming interfaces” as none of the interfaces of Andert are disclosed as being tailored.

For example, Andert’s Device Access Manager cannot have, or be itself considered the claimed tailored programming interfaces. While Andert indicates that a “one-size-fits-all” interface abstraction is not desirable when discussing the Device

¹² The ‘131 patent at 6:29-36 (emphasis added).

¹³ Andert at 9:15-16.

¹⁴ See Exhibit A, Wilson Dec. ¶¶ 36-38.

Access Manager, Andert is discussing the functions of the Device Access Manager, not the interface to the Device Access Manager.¹⁵

Furthermore, even if Andert's Device Access Manager had or was a tailored interface, the reference fails to disclose other elements of the claim. If Andert's Device Access Manager was cited against the "tailored distinct programming interfaces" claim limitation, nothing in Andert would disclose the claimed server, applications, or the plurality of activation models (in claim 7).

In conclusion, the rejection improperly alleges anticipation by Andert of the claimed: "a first plurality of tailored distinct programming interfaces" because the asserted I/O Service Frameworks in Andert are not disclosed to be tailored. Further, no other interface in Andert is disclosed as being tailored. Accordingly, Patent Owner submits that Andert does not anticipate claim 3.

Additionally, claim 1 recites the identical limitation and claims 9 and 18 recite a similar limitation and are patentable for the reasons discussed above with respect to claim 3. Claim 9 recites "wherein the operating system provides computer system services through a tailored distinct one of a plurality of program structures, each tailored distinct program structure comprising: a first programming interface for receiving service requests for a set of computer system I/O services of a first type." Claim 18 recites "a tailored distinct family server, operating in an operating system environment and dedicated to providing access to service requests for the first type of I/O service." Just as claim 3 is not anticipated by Andert, the respective rejections of claims 1, 9, and 18 also fail to anticipate because Andert's I/O Service Frameworks are not disclosed as being tailored or distinct. Accordingly, Patent Owner submits that Andert does not anticipate claims 1, 3, 9, 18, or any of their respective dependent claims.

¹⁵ Andert at 11:26-53.

B. Andert Does Not Disclose “an operating system comprising a plurality of servers”

Claim 3 recites “wherein the operating system comprises a plurality of servers, and each of the first plurality of programming interfaces transfer service requests to one of the plurality of servers, wherein each of the plurality of servers responds to service requests from clients of the separate sets of I/O services.” The Office Action alleges that Andert anticipates this limitation in the following way:

Andert discloses that the operating system comprises a plurality of servers (see Figures 1 and 2, I/O services frameworks), and each of the first plurality of programming interfaces transfer service requests to one of the plurality of servers, wherein each of the plurality of servers responds to service requests from clients of the separate sets of I/O services (see Figure 1, c 10:1-6).¹⁶

These portions of the Andert reference cited in the Office Action do not make any mention of an operating system comprising a plurality of servers. Specifically, the Office Action cites to Figures 1 and 2 of Andert where it shows the I/O Service Frameworks which are presumably alleged to be the claimed servers. The Office Action however does not assert that Andert’s I/O Service Frameworks are part of an operating system as required by the claim. This is most likely because Andert explains that the I/O Service Frameworks are not part of the operating system.

Andert specifically illustrates its I/O Service Frameworks as being outside the operating system.¹⁷ For example, in Andert’s Figure 3, below, the client I/O services shown communicating with the device access manager are illustrated as being in a user mode, which as Dr. Wilson discusses in his included Declaration, is another way of specifying that they are outside of the operating system.¹⁸

¹⁶ Ctrl. No. 90/011,311, Office Action, mailed Apr. 11, 2011, at 5 (bold in original).

¹⁷ Andert, at FIG. 3.

¹⁸ See Exhibit A, Wilson Dec. ¶¶ 40-41.

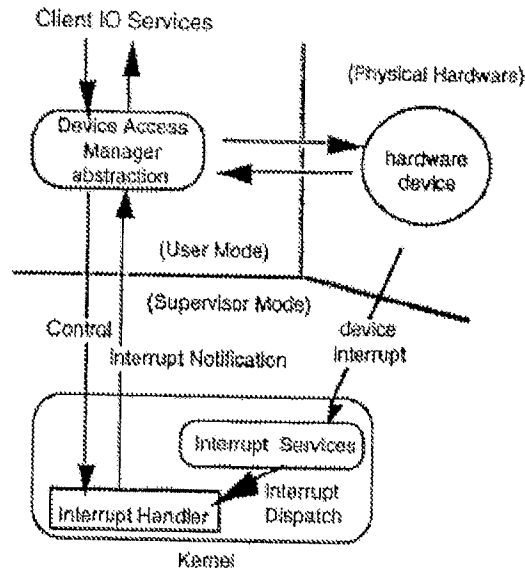


FIGURE 3

Dr. Wilson explains that both the '131 patent and Andert draw a clear boundary between its user mode and its supervisor mode (operating system mode), and that both references use the user mode terminology consistently and the terminology has the same meaning in both documents.¹⁹ As Andert never discloses its I/O Service Frameworks as being in the operating system, Patent Owner asserts that the rejection should be withdrawn.

While it has not been alleged in the current rejection, Andert's Interrupt Services, which are illustrated to be in Andert's Supervisor Mode, cannot be the claimed servers. As Dr. Wilson discusses in the accompanying Declaration, he understands the '131 patent servers to receive service requests via an API from applications.²⁰ Since the interrupt services do not receive service requests from applications, the interrupt services cannot be the claimed service.

In conclusion, the Office Action incorrectly alleges that Andert discloses "the operating system comprises a plurality of servers" because Andert does not disclose that its I/O Service Frameworks are in the operating system. In

¹⁹ See Exhibit A, Wilson Dec. ¶¶ 12, and 40-41; see also Exhibit A Wilson Dec. at ¶¶ 10-14, and 17-19.

²⁰ See Wilson Dec. ¶ 26.

fact, Andert specifically excludes the I/O Service Frameworks from the operating system. Accordingly, Patent Owner submits that Andert does not anticipate claim 3, or its dependent claims. Furthermore, independent claims 9 and 18 also recite limitations that require that servers be within the operating system. Claim 9 recites, “the operating system provides computer system services through a tailored distinct one of a plurality of program structures, each tailored distinct program structure comprising.. a first server” and claim 18 recites “a tailored distinct family server, operating in an operating system environment.”²¹ As claim 9 and 18 recite similar limitations as claim 3, they, and their respective dependent claims, are also patentable for the same reasons as claim 3 and its dependents. Accordingly, Patent Owner requests a NIRC confirming these claims.

C. Teaff Does Not Disclose “the operating system comprises a plurality of servers”

Claim 3 recites “wherein the operating system comprises a plurality of servers, and each of the first plurality of programming interfaces transfer service requests to one of the plurality of servers, wherein each of the plurality of servers responds to service requests from clients of the separate sets of I/O services.” The Office Action alleges that Teaff anticipates this limitation in the following way:

²¹ As discussed in his Declaration, Dr. Wilson explains that an operating system environment is the run time memory environment of the operating system. See Wilson Dec. ¶¶ 20-23.

each of the first plurality of programming interfaces transfer service requests to one of the plurality of servers, wherein each of the plurality of servers responds to service requests from clients of the separate sets of I/O services (see p. 4, Modularity and APIs and Figure 2; Appendix A).²²

However, none of the cited sections disclose that the “operating system comprises a plurality of servers.” Teaff’s page 4 cited in the Office Action describes that Teaff’s “HPSS architecture is highly modular,” and that the “HPSS software components are loosely coupled” with APIs. Nothing in this portion of Teaff or the Office Action points to an operating system comprising a plurality of servers. The same is true in Teaff’s Appendix A, which lists APIs to HPSS components. Figure 2 does illustrate a plurality of servers, but it does not show their relationship to the operating system. Further, Teaff’s Figure 2 shows servers as being HPSS components, and as discussed in detail below, Teaff’s HPSS application layer is separate from the operating system. Patent Owner submits that the rejection fails to anticipate claim 3 because it does not show that Teaff discloses that the “operating system comprises a plurality of servers,” as claimed.

The Teaff reference discloses what it calls a High Performance Storage System in which client requests are directed by an HPSS server to network-attached storage devices through a high-speed data transfer network.²³ In the HPSS system, clients gain access to the networked storage devices by accessing the HPSS server through an API. Teaff never discloses that its HPSS servers are part of the operating system, and never discloses that the HPSS system is itself an operating system. Rather Teaff actually indicates the opposite. As illustrated in Teaff’s Figure 3 below, HPSS is shown to be situated several layers above and indeed separate from the operating system.

²² Ctrl. No. 90/011,311, Office Action, mailed Apr. 11, 2011, at 3 (bold in original).

²³ Teaff, at 3.

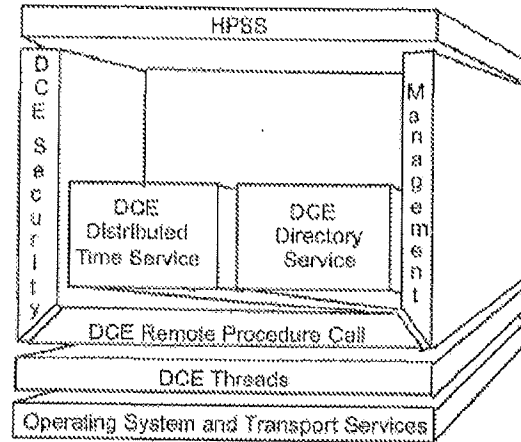


Figure 3 - HPSS DCE Architecture Infrastructure

Further, the HPSS itself cannot be considered an operating system. HPSS utilizes services from the layers below such as the DCE (distributed computing environment standard) and the operating system,^{24, 25} and such behavior is inconsistent with a view of the HPSS being considered an operating system.²⁶

In fact, Teaff does not anticipate the claim because including the servers in an operating system was antithetical to Teaff's explicit design objective.²⁷ Specifically Teaff states:

Another important design goal is portability to many vendor's platforms to enable OEM and multivendor support of HPSS. HPSS has been designed to run under Unix requiring no kernel modifications, and to use standards based protocols, interfaces, and services where applicable.²⁸

In the passage above, Teaff clearly states that it is designed to be portable to many vendor's platforms, and therefore Teaff would not want to limit the use of its servers to one specific operating system.

²⁴ See Teaff, p. 4 ("These requirements are accomplished using a client/server architecture, the use of OSF's DCE as its distributed infrastructure, support for distributed file system interfaces and multiple servers.")

²⁵ See Teaff, p. 7 ("HPSS uses OSF's DCE as the base infrastructure for its distributed architecture.")

²⁶ See Exhibit A, Wilson Dec. ¶ 45.

²⁷ See Exhibit A, Wilson Dec. ¶ 46.

²⁸ Teaff, at 5.

In conclusion, Teaff does not disclose “wherein the operating system comprises a plurality of servers, and each of the first plurality of programming interfaces transfer service requests to one of the plurality of servers, wherein each of the plurality of servers responds to service requests from clients of the separate sets of I/O services” as claimed. Further, the limitation is antithetical to Teaff’s specific design objectives to work in a layer under the operating system. Accordingly, Patent Owner submits that Teaff does not anticipate claim 3, or its dependents. Accordingly, Patent Owner requests a NIRC confirming these claims.

D. Andert Does Not Disclose “the operating system comprises a plurality of activation models”

Claim 7 recites, *inter alia*, “[t]he computer system defined in claim 3 wherein the operating system further comprises a plurality of activation models.” Regarding Andert, the Office Action asserts:

With respect to claim 7, the operating system further comprises a plurality of activation models, wherein each of the plurality of activation models is associated with one of the plurality of servers to provide a runtime environment for the set of I/O services to which access is provided by said one of the plurality of servers (c 11:25-58).²⁹

The Office Action asserts that Andert discloses the claimed activation model by way of Andert’s Device Access Manager. However, the activation model is claimed to be in the operating system while the Device Access Manager resides outside the operating system in user space.

Andert requires that the Device Access Manager reside outside of the operating system. Similar to the discussion above with respect to servers, Andert places its Device Access Manger in the user space as illustrated in its FIG. 3, below.

²⁹ Ctrl. No. 90/011,311, Office Action, mailed Apr. 11, 2011, at 6. (bold in original)

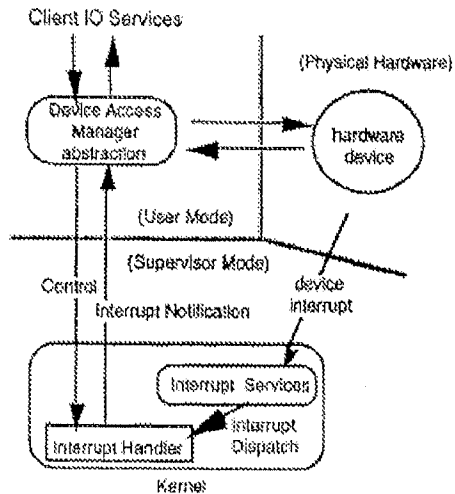


FIGURE 3

Accordingly, Andert explicitly places its Device Access Manager outside its operating system, unlike the claimed activation model, which is recited as being comprised in the operating system; therefore Andert does not anticipate this limitation of the claim.

More fundamentally, the activation model is just a model while the Device Access Manager is actual executable code. The activation model is defined by the claims as a model associated with one of the plurality of servers to provide a runtime environment for the set of I/O services. This is different from Andert's Device Access Manager that executes hardware access tasks. Andert describes the Device Access Manager as "user-mode abstractions that execute outside the kernel and are charged with all tasks performing tasks associated with hardware access."³⁰ Andert explains that some of the tasks performed by the Device Access Manager include: install and remove Interrupt Handlers,³¹ makes buffers ready for I/O,³² unlocks client's resident I/O buffers,³³ and informs clients through the I/O service framework of completed tasks.³⁴ The activation model, on the other hand, does not perform tasks; rather, as explained

³⁰ Andert, at 9:49-51.

³¹ Andert, at 10:42-43.

³² Andert, at 11:4-6.

³³ Andert, at 11:19-20.

³⁴ Andert, at 11:20-22.

above, it defines the runtime environment. It is something that executables reference to. Accordingly, the Device Access Manager does not disclose the claimed activation models.

In conclusion, the Office Action incorrectly alleges that Andert discloses “the operating system further comprises a plurality of activation models” because Andert does not disclose that its Device Access Manager, which is cited against the claimed activation models, is in the operating system. In fact, Andert specifically places the Device Access Manager in user space, and therefore outside the protection of the operating system. Accordingly, Patent Owner submits that Andert does not anticipate claim 7, or its dependent claim. Independent claims 9 and 18 also recite limitations that require that activation models be within the operating system. Claim 9 recites, “the operating system provides computer system services through a tailored distinct one of a plurality of program structures, each tailored distinct program structure comprising... an activation model to define an operating environment” and claim 18 recites “a tailored distinct family server, operating in an operating system environment and dedicated to providing access to service requests for the first type of I/O service, receiving and responding to the service request based on an activation model specific to the first type of I/O services.”³⁵ As claim 9 and 18 recite similar limitations as claim 7, they, and their dependent claims, are also patentable for the same reasons as claim 7 and its dependents. Accordingly, Patent Owner requests a NIRC confirming these claims.

³⁵ As discussed in his Declaration, Dr. Wilson explains that an operating system environment is the run time memory environment of the operating system. See Wilson Dec. ¶¶ 20-23.

E. Teaff Does Not Disclose “A computer system comprising: a bus.”

Claim 1 recites, *inter alia*, “A computer system comprising: a bus; at least one memory coupled to the bus . . .; and a processing unit coupled to the bus.” Regarding Teaff, the Office Action asserts:

Teaff discloses **a computer system** (p. 3, Figure 1) **comprising:**

a bus (Figure 1, control buses that connect network elements and workstations to the HPSS servers; also, a bus is an inherent component of a computer system);

at least one memory (Figure 1, network attached memories or other memories in servers and workstations) **coupled to the bus for storing data and programming instructions that include applications and an operating system; and**

a processing unit (Figure 1, servers and workstations) **coupled to the bus and running the operating system and applications by executing programming instructions** (see pp. 4-5, Modularity and APIs, and Portability and Standards), **wherein an application has a first plurality of tailored distinct programming interfaces** (p. 4, “The HPSS software components are loosely coupled, with open application program interfaces (APIs) defined at each component level”; see also Appendix A for a list of tailored APIs)³⁶

A bus, as that term was understood by persons of ordinary skill in the art when the ‘131 patent was filed, is a collection of parallel wires directly connected to a processor and other computer hardware components.³⁷

The Office Action asserts that Teaff discloses a bus in two possible applications of Teaff. The first application of Teaff matches the understanding of a person of ordinary skill in the art for a bus as stated above; that is, the recited bus is inherently disclosed as a component of a computer system. Even under this application of Teaff, it does not

³⁶ Ctrl. No. 90/011,311, Office Action, mailed Apr. 11, 2011, at 2-3. (bold in original)

³⁷ See Exhibit A, Wilson Dec. ¶¶ 27-30.

anticipate when the entire claim is considered, which is required for an anticipation rejection; that is, Teaff does not disclose “a processing unit” having all of the recited limitations, and that is coupled to the bus.

Under another application of Teaff, the Office Action asserts that Teaff’s control network is a bus, but this is inconsistent with a person of ordinary skill in the art’s understanding of a bus when the ‘131 patent was filed, and therefore, this application of Teaff also does not anticipate the claim. The rejection does not allege that every element of the claim resides on the same computer as would be required even if an inherently disclosed bus were asserted from one of Teaff’s computers. Still, every element of the claim must be disclosed in Teaff as being on that same computer.³⁸ The rejection fails to identify a specific computer that is disclosed as having a bus connected to a memory and to a processing unit having the characteristics recited in the claim. In fact, the rejection argues the opposite by citing to Teaff’s sections “Modularity and APIs”, and “Portability and Standards,” which discloses that HPSS software components are distributed (i.e, not on the same computer). In these sections Teaff does acknowledge flexibility in its distributed system in that components can be moved to other platforms, but Teaff never discloses a single computer that meets the claimed limitations, and accordingly does not anticipate the claim.

The rejection also improperly alleges that Teaff’s control network anticipates the claimed bus. At the time the ‘131 patent was filed, a network connection was not considered to be a bus.^{39,40} In his Declaration, Dr. Wilson discusses that at least until 1998, a bus was known by persons of ordinary skill in the art to be a collection of parallel wires directly connected to a processor and other computer hardware components.⁴¹ This understanding of what a bus was at the time the ‘131 patent was filed is consistent with the ‘131 patent’s specification; in particular, its description of the computer system

³⁸ See MPEP 2131 citing *In re Bond*, 910 F.2d 831, 15 USPQ2d 1566 (Fed. Cir. 1990) (“The elements must be arranged as required by the claim, but this is not an ipsissimis verbis test, i.e., identity of terminology is not required.”).

³⁹ See Exhibit A, Wilson Dec. ¶ 31.

⁴⁰ See, Exhibit C, Wikipedia - Bus (computing), [http://en.wikipedia.org/wiki/Bus_\(computing\)](http://en.wikipedia.org/wiki/Bus_(computing)) - (last modified on May 25, 2011).

⁴¹ See Exhibit A, Wilson Dec. ¶¶ 27-30.

illustrated in FIG. 1. Teaff's control network as described was not a bus as that term was understood by persons of ordinary skill in the art at the time the '131 patent was filed. Rather Teaff's control network is a network connection, which does not describe the claimed bus as those terms were understood at the time the '131 patent was filed.

In conclusion, Teaff does not anticipate claim 1 under either of the two alternative application of Teaff asserted in the Office Action. While a bus may be an inherent component of a computer, when an internal bus is alleged to anticipate the claim, Teaff fails to disclose the rest of the recited elements of the claim. Second, Teaff's control network is not a bus, and therefore does not anticipate that claim element.

IV. Conclusion

It is respectfully asserted that the rejections have been properly traversed. Patent Owner therefore respectfully requests that the Examiner reconsider all presently outstanding rejections and that they be withdrawn. Patent Owner believes that a full and complete reply has been made to the outstanding Office Action and, as such, the present reexamination proceeding is in condition for a Notice of Intent to Issue a Reexamination Certificate (NIRC). If Examiner believes, for any reason, that personal communication will expedite prosecution of this reexamination proceeding, the Examiner is invited to telephone the undersigned at the number provided.

Prompt and favorable consideration of this Response is respectfully requested.

* * * * *

The undersigned representative requests any extension of time that may be deemed necessary to further prosecution of this application.

The undersigned representative authorizes the Commissioner to charge any additional fees under 37 C.F.R. § 1.16 or 1.17 that may be required, or credit any overpayment, to Deposit Account 14-1437, referencing Attorney Docket No. 20142.0003.RXUS00.

In order to facilitate the resolution of any issues or questions resented by this paper, the Examiner may directly contact the undersigned by phone to further discussion.

Respectfully submitted,

/tracy w. druce/

Tracy W. Druce, Esq.
Reg. No. 35,493

Brian K. McKnight, Esq.
Reg. No. 59,914

C. Gideon Korrell, Esq.
Reg. No. 60,131

Novak, Druce + Quigg LLP
1000 Louisiana Street, Suite 5300
Houston, Texas 77002
(713) 571-3400
(713) 456-2836 (fax)
tracy.druce@novakdruce.com

**Exhibit A – Declaration of
David A. Wilson, Ph.D.**

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Reexamination of
U.S. Patent No. 5,915,131

to KNIGHT, *et al.*

Reexam Control No. 90/011,311

Filed November 1, 2010

For: **METHOD AND APPARATUS
FOR HANDLING I/O REQUESTS
UTILIZING SEPARATE
PROGRAMMING INTERFACES TO
ACCESS SEPARATE I/O SERVICES**

Confirmation No.: 1355

Art Unit: 3992

Examiner: CHOI, Woo

Atty. Docket No. 20142.0003.RXUS00

Attn: Central Reexamination Unit
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Declaration of David A. Wilson, Ph.D. Under 37 C.F.R. § 1.132

I, David A. Wilson, Ph.D., declare as follows:

1. I have been retained on behalf of Apple Inc. ("Apple"), the Patent Owner in the above captioned reexamination. I understand that this reexamination involves U.S. Patent No. 5,915,131 ("the '131 patent") entitled "Method and Apparatus for Handling I/O Requests Utilizing Separate Programming Interfaces to Access Separate I/O Services."

2. I have reviewed and am now familiar with the specification of the '131 patent filed on May 5, 1995.

3. I have reviewed and am now familiar with the Office Action dated April 11, 2011 issued by the U.S. Patent and Trademark Office ("USPTO") for the '131 patent ("Office Action"). I understand that claims 11 and 14 have been confirmed and claims 1-10, 12, 13, and 15-20 have been rejected in this reexamination proceeding.

4. I have reviewed and am now familiar with U.S Patent No. 5,566,346 to Andert et al. (“Andert”) which is a document applied under one 35 U.S.C. § 102 rejection set forth in the Office Action.

5. I have also reviewed and am now familiar with *The Architecture of the High Performance Storage System* by Danny Teaff et al. (“Teaff”) which is a document applied under another 35 U.S.C. § 102 rejection set forth in the Office Action.

6. In making the statements, and reaching my opinions and conclusions stated herein, I have considered Andert, Teaff, and the ‘131 patent, including portions of its file history, in the context of my own education, training, research, knowledge, and personal and professional experience.

7. My retention is through a consulting services agreement I have with Silicon Valley Expert Witness Group of Mountain View, CA to provide expert witness services in this matter. My compensation is in no way dependent on, nor affects the substance of my statements in this Declaration.

Qualifications

8. A detailed record of my professional qualifications, including a list of publications, awards, professional activities, and prior testimony is attached to this Declaration as Exhibit 1.

The '131 Patent

9. Claim 3 of the '131 patent is generally representative of the other rejected independent claims. Claim 3 recites a computer system comprising a bus, at least one memory coupled to the bus, and a processing unit coupled to the bus. The memory coupled to the bus is for storing data and programming instructions that include applications and an operating system. The operating system comprises a plurality of servers. A plurality of programming interfaces transfer service requests to one of the plurality of servers. Each of the plurality of servers responds to service requests from clients of the separate sets of input/output (I/O) services. The processing unit, also coupled to the bus, runs the operating system and applications by executing programming instructions. The application has a first plurality of tailored distinct programming interfaces available to access a plurality of separate sets of I/O services provided through the operating system via service requests.

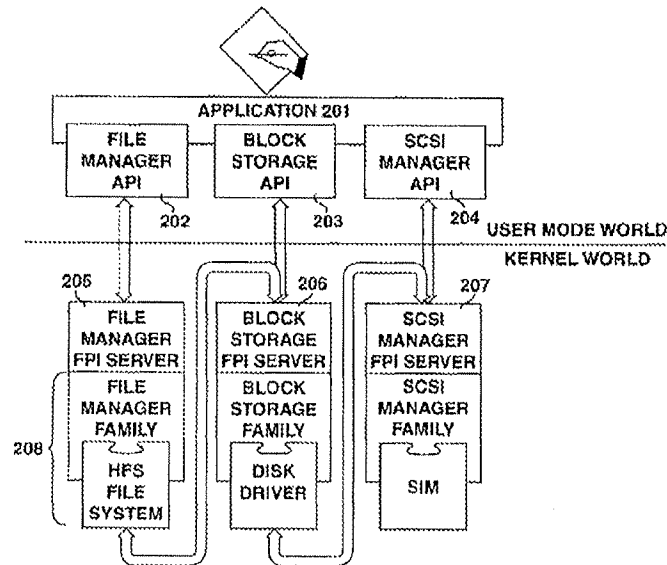


FIG. 2

Illustration of Claim 3 of the '131 Patent

10. Claim 3 can be better understood with reference to Figure 2 above of the '131 patent, together with the specification. As illustrated in Figure 2, the recited application exists in a "user mode world." This illustration is consistent with my

understanding of application programs which reside in the user domain of memory. This is also consistent with how the '131 patent uses the term "application" in both the specification and the claims.

11. The claimed plurality of servers that are in the operating system are illustrated in Figure 3 of the '131 patent above. The placement of these servers in the kernel world is consistent with the reference by the '131 patent claims and specification of the servers being located in the operating system. It is generally understood that some operating system components reside in the kernel domain memory space, and that computer system devices are only accessible through the kernel domain. The significance of this is, as the '131 patent explains, "[t]he user domain does not have direct access to data of the kernel domain, while the kernel domain can access data in the user domain." The '131 patent at 5:9-11.

12. It is my understanding that the claimed applications reside in the user domain, the operating system domain includes the kernel domain, and the operating system domain and the user domain are separate. Such a conclusion is consistent with the description and claims of the '131 patent, and also with my experience.

Application

13. My understanding that the claim term "application" refers to an application that is in the user domain is based on my reading of the claim, and is fully consistent with the rest of the '131 patent. Specifically, claim 3 requires that the computer system comprises applications and an operating system, and goes on to specify that the operating system comprises a plurality of servers, and that the application has a plurality of tailored distinct programming interfaces. *See* the '131 patent, claim 3. These tailored distinct programming interfaces are used by the applications to access the servers of the operating system. Based on my reading of claim 3, and my technical knowledge of computer programming, it is clear that the application and the operating system are different entities, and that the application located in user domain is not included in or a part of the operating system.

14. My understanding that the claimed application is in the user domain and separate from the operating system is the only understanding that is consistent with the

specification. In addition to Figure 2, above, showing the applications in “user mode world,” the specification repeatedly refers to the application as being in the user domain/user mode world. For example, Figures 3, 4, 7 and 8 consistently show the applications in the user mode world.

Operating System

15. Claim 3 also recites an operating system, and requires that the operating system comprise a plurality of servers, and further requires that the I/O services are provided through the operating system via service requests. Figure 2 illustrates an example of the plurality of operating system servers, including a File Manager FPI Server, a Block Storage FPI server, and a SCSI Manager FPI server.

16. The patent uses FPI as an acronym for “Family Programming Interface,” where the programming interfaces used to request services are defined for each family of I/O devices.

17. Each of these servers are shown in the “kernel world,” which refers to memory space with access limited by the operating system. *See* the ‘131 patent at 5:9-11 (“The user domain does not have direct access to data of the kernel domain, while the kernel domain can access data in the user domain.”); 5:5-6 (“In one embodiment, the computer system runs a kernel-based, preemptive, multitasking operation system”); 6:43-46 (“In one embodiment, a family 305 may provide two versions of its FPI library 303, one that runs in the user domain and one that runs in the operating system kernel domain.”) A further point illustrated by the quotes above is that the kernel domain is part of the operating system.

18. The ‘131 patent treats the operating system domain and the user domain as separate domains. *See, e.g.,* the ‘131 patent at 6:20-28 and 6:37-46 (discussing that separate FPI libraries can be accessed from the user domain or the operating system domain); and 5:8-9 (discussing that the computer system has separate protection domains). The fact that the ‘131 patent discusses placing separate FPI libraries in separate domains, depending on function, clearly shows that the ‘131 patent is treating these two domains as being separate.

19. Further the concept of a separate operating system domain (or mode) and user domain (or mode) is consistent with my understanding of computer systems at the time the '131 patent was filed. For example, in Andrew S. Tanenbaum's seminal treatise *Modern Operating Systems*, he explains:

[t]he operating system is that portion of the software that runs in **kernel mode** or **supervisor mode**. . . . Compilers and editors run in **user mode**. If a user does not like a particular compiler, he is free to write his own if he so chooses; he is not free to write his own disk interrupt handler, which is part of the operating system and is normally protected by hardware against attempts by users to modify it.

Exhibit - 2, Andrew S. Tanenbaum, *Modern Operating Systems*, Prentice-Hall, 1992, at pp. 1-3 (emphasis in original).

Most CPUs have two modes: kernel mode, for the operating system, in which all instructions are allowed; and user mode, for user programs, in which I/O and certain other instructions are not allowed.

Exhibit - 2, Andrew S. Tanenbaum, *Modern Operating Systems*, Prentice-Hall, 1992, at p. 19

20. Claim 18 and its dependent claims recite a variation on the "operating system comprises" limitation found in claim 3, but the effect of the limitation is the same. Claim 18 recites "operating in an operating system environment." An operating system environment refers to the memory execution environment of an operating system at runtime. My understanding of this term is based on the claim language itself, the specification and its consistent usage of the term "environment," and my knowledge as a person of ordinary skill in the art when the patent was filed.

21. Claim 18 recites "operating in an operating system environment" where the word "operating" indicates that it is a runtime environment. Additionally, the word environment indicates a runtime memory space. My understanding of the claim term is supported by the specification of the '131 patent where it is stated:

The operating system running on processor 103 takes care of basic tasks such as starting the system, handling interrupts, moving data to and from memory 104 and peripheral devices via input/output interface unit 140, and

managing the memory space in memory 104. In order to take care of such operations, the operating system provides multiple execution environments at different levels (e.g., task level, interrupt level, etc.). Tasks and execution environments are known in the art.

The '131 patent at 6:62-5:3.

In the passage above, the '131 patent explains the operating system provides multiple execution environments, which refers to protected memory spaces, and support for various software functions provided by the operating system. I note that the paragraph above states that environments are known in the art, which I provide an explanation of herein.

22. The use of the term “environment” to refer to memory space is also consistent with its use elsewhere in the '131 patent specification. Specifically, the '131 patent states:

Referring to FIG. 4, three instances of families 401-403 are shown operating in the kernel environment. Although three families are shown, the present invention may have any number of families.

In the user mode, two user-mode FPI libraries, Xlibu 404 and Zlibu 405, are shown that support the FPIs for families X and Z, respectively. In the kernel environment, two kernel-mode FPI libraries, Ylibk 406 and Zlibk, 407, for families Y and Z, respectively, are shown.

The '131 patent at 8:34-42.

In the passage above the '131 patent discusses operating in the kernel environment, which is also referring to the runtime memory environment of the kernel space.

23. Accordingly, based on my broad reading of the claim language, which is consistent with the specification, my understanding of the claim phrase “operating in an operating system environment” is that it means the memory execution environment of an operating system at runtime.

Server

24. As discussed above, claim 3 further recites that the operating system comprises a plurality of servers. Claim 3 also specifically recites that “the plurality of servers responds to service requests from clients of the separate sets of I/O services.” While the ‘131 patent discusses that the clients can be either an application in the user domain or another I/O service family, the claim specifically identifies that it is the application which accesses the I/O services, and thus claim 3 is directed to an application being the client that is requesting the service.

25. Tanenbaum defines applications as follows:

Finally, above the systems programs come the applications programs. These programs are written by the users to solve their particular problems, such as commercial data processing, engineering calculations, or game playing.

Exhibit – 2, Andrew S. Tanenbaum, *Modern Operating Systems*, Prentice-Hall, 1992, at p. 3

26. The function of the servers is to provide access to a distinct family of services. The ‘131 patent at 5:59-62. The servers provide access to these services by responding to service requests made by the application programs. The ‘131 patent at 6:47-49. Application programs initiate service requests by sending service request using the programming interface associated with the family of a particular service. The ‘131 patent at 9:1-3. The ‘131 patent explains that “[a]ccess to services is available only through an I/O family’s programming interface.” The ‘131 patent at 5:29-30. As shown in Figure 2 of the ‘131 patent (above), the servers reside in the kernel domain, and as explained in more detail above, access from applications in the user domain is prevented, except through a programming interface made available by the operating system. Thus, access to the family services by applications is only available through the programming interface for that family.

Bus

27. Claim 1 recites a bus, and a memory and processing unit coupled to the bus. A bus, as that term was understood by persons of ordinary skill in the art when the '131 patent was filed is a collection of parallel wires directly connected to a processor and other computer hardware components. Often the wires corresponded to one or more pins on the processor. In another book Tanenbaum provides the following explanation of a microcomputer bus from 1984:

The components of a microcomputer, the microprocessor, memory, and I/O controller chips, are connected by a collection of parallel wires called a bus. The lines (wires) in the bus can be classified as address, data, or control. They correspond closely but not exactly, to the microprocessor's pins. Several buses have been standardized by IEEE and other organizations to facilitate interconnection of processors, memories, and other devices from different vendors.

Exhibit – 3, Andrew S. Tanenbaum, *Structured Computer Organization*, Second Edition, Prentice-Hall, 1984, at p. 95.

28. The concept of a bus was still the same in 1995, when the '131 patent was filed, and even beyond to at least 1998 when Mark Edmead and Paul Hinsberg wrote their book *Windows NT Performance*, where they explain:

When a program executes, the operating system must first get the program from the hard drive. Part of the program is then transferred from the disk (or whatever media the program resides in). Most Intel-based systems provide the option of installing interface cards into different bus types. The system contains a bus controller that is responsible for servicing the request to and from the peripheral device...

Intel systems provide an 8-bit, 16-bit, or 32-bit bus. Obviously, the 32-bit bus is ideal implementation. The slower of the available bus architectures is the ISA (Industry Standard Architecture) bus. The ISA bus has a transfer rate of 5MB/sec. ISA includes both a 8-bit and 16-bit bus. A Peripheral Component Interconnect (PCI) bus, on the other hand, is a 32-bit bus, which means that it can address the full 4GB. The PCI bus is also a 133MB/sec bus, which is a significant improvement from the ISA bus architecture.

Exhibit – 4, Mark Edmead and Paul Hinsberg, *Windows NT Performance: Monitoring, Benchmarking, and Tuning*, New Riders Publishing, 1998, at pp. 10-11.

In the passage above, Edmead and Hinsberg discuss common buses such as the ISA bus and the PCI bus, which are both a collection of parallel wires directly connected to a processor and other computer hardware components.

29. The fact that a bus was known to be a collection of parallel wires directly connected to a processor and other computer hardware components in before and after 1995 is further exemplified by the '131 patent's description of its computer system illustration in FIG. 1. The '131 patent states:

The computer system of the present invention also includes an input/output (I/O) bus or other communication means 101 for communication information in the computer system. A data storage device 107, such as a magnetic tape and disk drive, including its associated controller circuitry, is coupled to I/O bus 101 for storing information and instructions. A display device 121, such as a cathode ray tube, liquid crystal display, etc., including its associated controller circuitry, is also coupled to I/O bus 101 for displaying information to the computer user, as well as a hard copy device 124, such as a plotter or printer, including its associated controller circuitry for providing a visual representation of the computer images. Hard copy device 124 is coupled with processor 103, main memory 104, non-volatile memory 106 and mass storage device 107 through I/O bus 101 and bus translator/interface unit 140. A modem 108 and an ethernet local area network 109 are also coupled to I/O bus 101.

The '131 patent at 8:34-42.

As explained in the passage above, the '131 patent also discusses and illustrates a processor and a memory coupled to an I/O bus.

30. As demonstrated above, buses were commonly known in the computing arts at least as early as 1984 and since that time they were known to be at least a physical connection between a processor and another device. More accurately, buses were a collection of parallel wires directly connected to a processor and other computer hardware components.

31. In 1995 when the '131 patent was filed, a person of ordinary skill in the art, such as myself, would not have considered a network connection to be a bus.

Activation Model

32. Claim 7 further recites that the operating system further comprises a plurality of activation models. Each of the plurality of activation models is associated with one of the plurality of servers to provide a runtime environment for the set of I/O services to which access is provided by said one of a plurality of servers. What I believe this to mean for a person ordinarily skilled in the art at the time of the filing of the '131 patent (May 5, 1995) is that the activation models are defining *when* and *how* tasks are performed by the operating system I/O servers (i.e., the runtime environment) in response to a I/O service request from an application program.

33. As recited in the claims, the activation models are a part of the operating system, and each is associated with one of the operating system I/O servers. The '131 patent specification provides a number of examples of activation models and explains how each controls how the respective server responds to service requests.

34. For example, the '131 patent explains that an operating system I/O server family associated with the "Single-Task [Activation] Model" runs a single monolithic task which is fed from a service request queue or a plug-in generated interrupt. The '131 patent at 10:53-63. As such, the tasks are executed asynchronously but no more than one at a time. The '131 patent explains that such a model is ideal for keyboard or mouse devices, or other devices that do not require more than one I/O request to be handled at once. The '131 patent at 11:37-53.

On the other hand, an operating system I/O server family associated with the “Task-Per-Request [Activation] Model” can process multiple I/O requests simultaneously. The ‘131 patent at 13:25-34. The servers associated with this activation model do this by blocking tasks until a request is complete. The ‘131 patent at 13:13-15. The ‘131 patent explains that the task-per-request activation model is ideal for operating system services such as a file manager. The ‘131 patent at 13:25-34. Thus, the activation models provide an operating system runtime environment that is tailored for each family to best utilize the unique characteristics of the I/O services or devices associated with that family.

Andert

35. Andert discloses an object-oriented input/output system for interfacing with a plurality of I/O devices. Andert at 1:52-55. Andert makes use of a plurality of different frameworks configured to interface between applications and I/O devices. Andert at 2:6-18. Andert states that these frameworks will assist developers in creating low-level I/O drivers by providing design and architectural guidance to the developer. Andert 5:38-40.

36. The state of the art around the time the Andert reference was filed was for an operating system to provide application programs with a single programming interface for accessing all of the I/O services available through the computer system. For example, the Macintosh OS offered an interface that included commands such as *OpenDriver*, *CloseDriver*, *FSRead*, *FSWrite*, *Control*, *Status*, and *KillIO*. Although some of these commands would not be applicable to certain I/O devices, the interface was the same for all devices nonetheless.

37. As another example of the state of the art when the Andert reference was filed, consider Tanenbaum’s description of “Input/Output in UNIX”, where each I/O device is treated like a file, so that “No special commands or system calls are needed. The usual READ and WRITE system calls will do just fine.”

7.3.4. Input/Output in UNIX

Like all computers, those running UNIX have I/O devices such as terminals, disks, printers, and networks connected to them. Some way is needed to allow programs to access these devices. Although various solutions are possible, the UNIX one is to integrate them into the file system as what are called special files. Each I/O device is assigned a path name, usually in */dev*. For example, the printer might be */dev/lp*, terminal 1 might be */dev/tty1*, and the network might be */dev/net*.

These special files can be accessed the same way as any other files. No special commands or system calls are needed. The usual READ and WRITE system calls will do just fine. For example, the command

```
cp file /dev/lp
```

copies the *file* to printer, causing it to be printed (assuming that this is permitted). Programs can open, read, and write special files the same way as they do regular files. In fact, *cp* in the above example is not even aware that it is printing. In this way, no special mechanism is needed for doing I/O.

Exhibit – 2, Andrew S. Tanenbaum, *Modern Operating Systems*, Prentice-Hall, 1992, at pp. 290-291.

38. This again shows the state of the art as not requiring or providing tailored distinct APIs for accessing I/O services.

39. The front page of Andert shows that it was first assigned to Taligent Inc., which was a joint venture between IBM and Apple, the makers of separate and incompatible computers and operating systems. Due to this cross-platform collaboration, an additional motive behind the Andert invention was to come up with a solution that avoided changes to or reliance upon a particular operating system. Such a motivation is apparent in the many places in Andert where it is explained generally that its system supports multiple operating systems. *See, e.g.*, Andert at 1:47, 2:16-17, 2:59, and 16:5-27. In addition, the Andert reference specifically explains that components of its system, such as the Device Access Managers, are placed in the user-mode. Andert at 9:48-51.

40. Andert draws a clear boundary between the operating system (supervisor mode) and outside the operating system (user mode). For example in Figure 3, below, Andert shows that the Client IO Services Interface and the Device Access Managers are in user mode, while the Interrupt Services are in the Supervisor Mode.

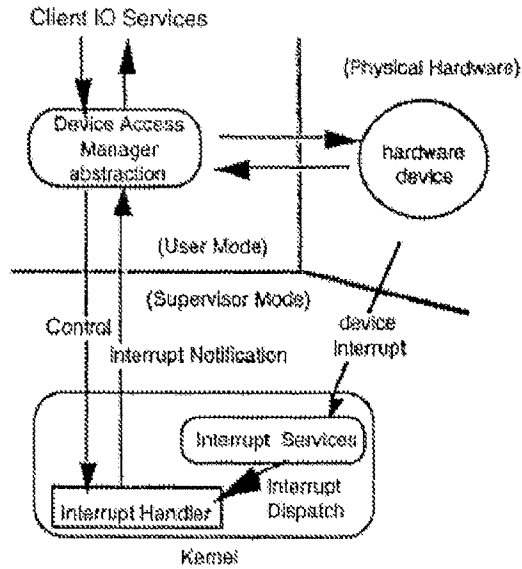


FIGURE 3

Andert Figure 3

41. Both the Andert reference and the '131 patent use the terminology "user mode." I understand both documents to be using this terminology consistently to distinguish that these services in the user mode are outside of the operating system. I further understand that the supervisor mode in Andert is used to delineate the operating system boundary. I further understand the use of the term kernel in both Andert and the '131 patent to have the same meaning in both documents. I further understand that the "Supervisor Mode" referred to in Andert Figure 3 above has the same meaning as "Kernel World" in the '131 patent.

Teaff

42. Teaff discloses a distributed high performance storage system (HPSS). The HPSS architecture uses a high-speed network for data transfer and a second network for control. Teaff, p. 2. The control network uses the Open Software Foundation's Distributed Computing Environment Remote procedure call technology. Teaff, p. 2.

43. Teaff discloses a layered system wherein the HPSS layer is built on a Distributed Computing Environment ("DCE") layer, which is built on top of an

Operating System layer. Figure 3, below, illustrates this arrangement. In this arrangement, the HPSS layer utilizes services from the DCE layers below it. *See* Teaff, p. 4 (“These requirements are accomplished using a client/server architecture, the use of OSF’s DCE as its distributed infrastructure, support for distributed file system interfaces and multiple servers...”); Teaff, p. 7 (“HPSS uses OSF’s DCE as the base infrastructure for its distributed architecture.”).

44. Figure 3 further indicates that HPSS does not directly access any Operating System services.

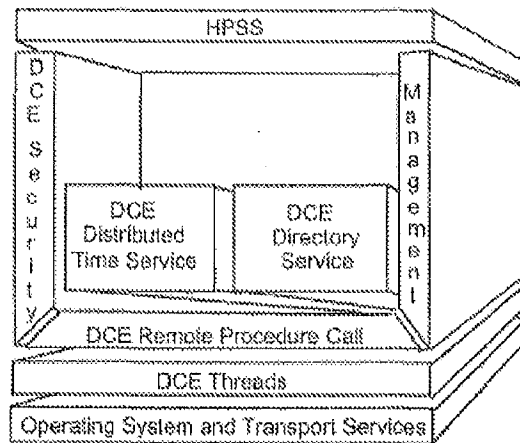


Figure 3 - HPSS DCE Architecture Infrastructure

Teaff Figure 3

45. Teaff’s arrangement demonstrates that the HPSS layer is not an operating system layer. The fact that the HPSS layer utilizes services from the layers below is inconsistent with the concept of an operating system, as discussed in paragraphs 15-19 above. As discussed in paragraph 15, the operating system provides services to the higher layers (applications), while limiting access to the operating system memory space. In paragraph 17, I discuss that the ‘131 patent’s discussion of operating systems is consistent with how a person of ordinary skill in the art understood the concept of an operating system when the ‘131 patent was filed. In paragraph 18, I further discuss that the operating system domain and user domain are separate domains. All of this is inconsistent with the HPSS layer being the operating system. Specifically, the HPSS

layer accesses services of the layers below it. Further, the HPSS layer cannot be both the operating system and the user domain because they are separate domains.

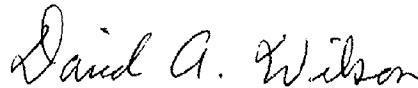
46. The fact that the HPSS layer is not an operating system layer is further supported by Teaff's express design objective to be portable to many different vendor's platforms. Specifically Teaff states:

Another important design goal is portability to many vendor's platforms to enable OEM and multivendor support of HPSS. HPSS has been designed to run under Unix requiring no kernel modifications, and to use standards based protocols, interfaces, and services where applicable.

Teaff, p. 5.

I hereby declare that all statements herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, 18 U.S.C. § 1001, and that such willful false statements may jeopardize the validity or enforceability of the '131 patent.

Date: June 13, 2011



David A. Wilson, Ph.D.

Exhibit - 1

David A. Wilson, Ph.D.



4181 Horizon Court
San Jose, CA 95148
(408) 532-1663
(650) 575-5687 (cell)
wilson99@pacbell.net

Expertise

- Object-Oriented Programming
- Java technologies on the server and client
- Java Swing programming
- C++, Smalltalk, Objective-C, and other object programming languages
- XML/SOAP Web Services
- Microsoft .NET Technologies
- Integrated Development Environments and programmer tools
- Distributed Computing, including CORBA and Java RMI
- Graphical User Interfaces and user interface design
- Software Development Processes and Practices
- Software Reuse
- Software Frameworks and Design Patterns
- Artificial Life technologies such as Genetic Algorithms
- Visual Programming Languages
- Digital Photography
- Organizing, summarizing, and presenting complex technical material
- Ultrasonic Imaging Systems for Medical Diagnosis

Professional Summary

Over twenty years experience at developing and teaching advanced technologies. Dr. Wilson has co-developed a number of innovative software products, including a dataflow-based visual programming language for doing complex numeric and financial calculations, two C++ applications frameworks, various Java tools for managing billing systems, and a Java-based prototype of an automated teller machine. He has also helped developed a number of innovative hardware products, including a real-time UV-Visible Spectrophotometer, and numerous real-time ultrasonic imaging systems for medical diagnosis. Dr. Wilson is recognized as an industry technologist and spokesman with excellent verbal and written communication skills.

Examples of Dave's Work

Employment History

Independent Consultant [1983 - present]

- Highly experienced technology consultant specializing in object-oriented development.
- Provides technical expertise to law firms specializing in intellectual property litigation.
- Software development using a variety of programming languages, including Java, C++, Objective-C, Smalltalk-80, and Object Pascal.
- Co-developed Spreadsheet 2000, an innovative application that provided the user with a visual programming language for performing complex numeric calculations and financial modeling.
- Co-developed QuickApp, a C++ applications framework used to develop a number of commercial Macintosh applications.
- Co-developed MicroGA – a C++ framework for solving optimization problems using the artificial life technique known as Genetic Algorithms.
- Projects for major clients described below.

Portal Software, Inc: Senior Architect [2000 - 2002]

- Architected and developed Storable Class Editor and other Java-based client applications that allow customization of Portal's real-time billing system.
- Architected and prototyped various systems monitoring and management tools.
- Architected and prototyped an XML/SOAP-based Web Service to support Portal's Infranet Content Connector.
- Member of the Architecture Steering Group.
- Defined the software development process used throughout the Product Development organization.
- Lived in Germany for five months, helping with technology transfer between Portal's European and US-based product development organizations.
- Helped drive Engineering's mentoring program; helped engineers with career and personal development.
- Developed and presented in-house training on Java, CORBA, O-O Frameworks, XML Web Services, and other subjects

Sun Microsystems: Contractor [1999 - 2000]

- Developed the first hands-on Jini and JavaSpaces programming "Code Camps". Invented a technique for dynamically assembling GUI applications from independent Jini services.
- Developed Sun's first "Code Camp" on Java performance tuning. Showed how to make Java clients and servers run faster using performance profiling tools and code optimization.
- Trained three other trainers on how to develop courses and give effective technical presentations.

Sun Microsystems: Contractor [1998]

- In late 1998, Sun introduced their “Java 2” technology with a special press presentation at the Java Business Expo in NYC.
- Developed a prototype "Accessible" Automated Teller Machine for that introduction that featured voice synthesis (for blind users), high-contrast screen display modes (for visually-impaired users), and support for 10 languages including Chinese, Japanese, and Korean.
- This system was demo'ed on stage by a blind user (and her Guide Dog).
- User preferences were programmed into a Java Ring using the JavaCard APIs.
- The user interface was programmed in Swing, while RMI servers delivered functionality to the client.

Apple Computer:Contractor [1994 - 1996]

- In charge of all programmer training for the OpenDoc Development Framework (ODF)
- Developed and taught 5-day class on using C++ and ODF to develop new OpenDoc components.
- Wrote many ODF sample programs that were shipped with the product itself.

Apple's Pink Project/Taligent, Inc.: Contractor [1989 - 1994]

- Programmer training and sample programs
- Hired as the first contractor for Apple's super-secret Pink project to develop C++ sample programs and assemble hands-on programming classes for new employees, and eventually, for third-party developers.
- Continued in this role when Apple and IBM formed their \$200 million Taligent joint venture.
- Developed a 3-day class on *Writing Reusable C++ Frameworks* , and presented it all over the US, in the UK, in Australia, and at IBM's Yamato Research Labs outside of Tokyo.

Xerox and ParcPlace Systems: Contractor [1988]

- Developed Smalltalk programming classes
- Developed the first ParcPlace programmer training classes for Smalltalk-80.

Apple Computer: Contractor [1984 - 1989]

- Developed Apple's classes on Macintosh programming.
- Trained other trainers to help build Apple's “Developer University”.
- Trained developers on using Pascal and C to do procedural programming using the Macintosh Toolbox.
- Developed Apple's classes on using Object Pascal and C++ for object-oriented programming using the MacApp applications framework.

SRI International: Member of Technical Staff; Director, Bio-Engineering Research Center [1978 - 1983]

- Developed research instruments for medical diagnosis and therapy for the National Cancer Institute and the National Heart, Lung, and Blood Institute.
- Managed a team of 15 researchers in developing new products for biomedical research and ultrasonic imaging.

Hewlett-Packard Laboratories: Member of the Technical Staff [1970 - 1978]

- Helped invent and develop a new real-time Ultraviolet-Visible Spectrophotometer for chemical analysis using an Acoustically-Tuned Optical Filter.
 - Researched and started H-P's first project on real-time, two-dimensional, B-Scan ultrasonic imaging for medical diagnosis. This eventually led to a series of products with which H-P became the world leader in cardiac ultrasonic imaging.
-

Litigation Support Experience

- Wilson Sonsini Goodrich & Rosati
 - Date: 2002 - 2003
 - Project: Patent infringement, prior-art research and analysis
 - Status: Ongoing
 - Howrey Simon Arnold & White
 - Date: 2000 - 2001
 - Client: Sun Microsystems
 - Project: Patent/technology survey
 - Status: Completed
 - Howrey Simon Arnold & White
 - Date: 1998 - 1999
 - Client: Sun Microsystems
 - Project: Patent/prior-art research
 - Status: Completed
 - Howrey Simon Arnold & White
 - Date: 1997
 - Client: Apple Computer (v. Articulate Systems)
 - Project: Patent infringement, created expert reports and depositions
 - Status: Completed
-

Education

Ph.D. Applied Physics	Stanford University
M.S. Applied Physics	Stanford University
B.S. Engineering Physics	Cornell University

U.S. Patents

- Patent #4,482,834
 - Issue Date: November 13, 1984
 - *Acoustic imaging transducer*
 - Assignee: Hewlett-Packard
- Patent #4,442,713

- Issue Date: April 17, 1984
- *Frequency varied ultrasonic imaging array*
- Assignee: SRI International
- Patent #4,446,740
 - Issue Date: May 8, 1984
 - *Frequency controlled hybrid ultrasonic imaging arrays*
 - Assignee: SRI International
- Patent #4,471,785
 - Issue Date: September 18, 1984
 - *Ultrasonic imaging system with correction for velocity inhomogeneity and multipath interference using an ultrasonic imaging array*
 - Assignee: SRI International

Commercial Software Products

- *Spreadsheet 2000*. A Macintosh application published in 1997 by Casady & Greene. ISBN 1-56482-141-2
- *Keep It Simple Spreadsheet*. A Macintosh application published in 1996 by Casady & Greene. ISBN 1-56482-101-3
- *QuickApp*. A C++ applications framework published by Emergent Behavior in 1993.

Publications (1982 to Present)

- Wilson, David A., *The Java Guidelines – Creating fast, maintainable, reliable, portable Java programs*, published internally by Visa International in 1999 for use by their engineering teams.
- Wilson, David A., et. al., *C++ Programming With MacApp*, Addison-Wesley, 1990, ISBN 0-201-57021-1.
- Wilson, David A., et. al., *Programming With MacApp*, Addison-Wesley, 1990, ISBN 0-201-55062-8.
- Wilson, David A., *Class Diagrams: A Tool for Design, Documentation, and Modeling*, Journal of Object-Oriented Programming, January/February 1990, pp. 38 – 44.
- Wilson, David, *The Sordid Truth About Apple: Why Don't Those Idiots Ever Do Anything Right*, MacTutor (now MacTech), December 1990.
- Wilson, David A., *Introduction to MacApp & Object Programming*, published by MacApp Developer's Association, April 1989.
- Wilson, David, *MacApp Objects*, MacTutor (now MacTech), September 1987, pp. 41 – 46.
- Goodin, Sue and Wilson, Dave, *Programming the New Macs*, MacTutor (now MacTech), May 1987.
- Wilson, David, *Resource Formats for Asm, Rmaker and Lisa*, MacTutor (now MacTech), June 1986.
- Wilson, David A., et. al., *Practical BASIC Programs, IBM Personal Computer Edition*, Osborne/McGraw-Hill, 1982, ISBN 0-931988-80-2.

Formal Conference & Panel Presentations, (1988 – Present)

- Wilson, David A., *A Framework for Assembling Client Applications from Modular Components*, Sun's JavaOne Conference 2002, San Francisco, March 2002.
- Wilson, David A., *Designing Object-Oriented Frameworks*, 7th IBM Conference on Object-

Oriented Software Development, July 1994.

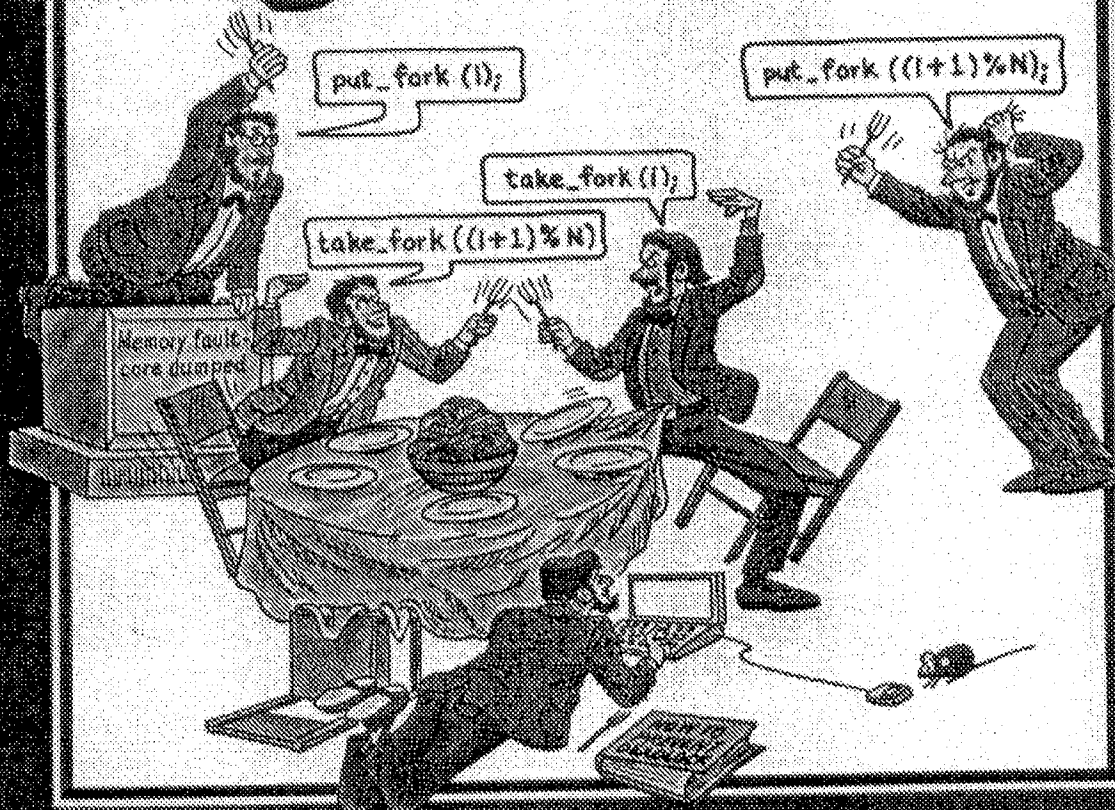
- Wilson, David A. and Wilson, Stephen D., *Writing Frameworks – Capturing Your Expertise About a Problem Domain*, Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA) 1993.
- Wilson, David A., *Developing a MacApp Application*, MacWorld Expo Tokyo, February 1991.
- Dan Shafer, David A. Wilson, Jeff McKenna, John R. Pugh, Adele Goldberg: Panel: *Teaching OOP*. OOPSLA 1988.
- Numerous Apple World-Wide Developer Conferences and Apple European Developer Conferences.

Professional Associations

- Member, IEEE

Exhibit 2

Modern Operating Systems



Andrew S. Tanenbaum

Tanenbaum, Andrew S.
Modern operating systems / Andrew S. Tanenbaum.
p. cm.
Includes bibliographical references and index.
ISBN 0-13-880187-0
1. Operating systems (Computers) I. Title.
QA76.76.D68T369 1992
005.4'3--dc20

91-45010
CIP

Acquisitions editor: Tom McIlwee
Production supervisor: Bayani Mendoza de Leon
Cover designer: Bruce Kenselaar
Prepress buyer: Linda Behrens
Manufacturing buyer: Dave Dickey
Supplements editor: Alice Dworkin
Interior designer: Andrew S. Tanenbaum



© 1992 by Prentice-Hall, Inc.
A Simon & Schuster Company
Englewood Cliffs, New Jersey 07632

The author and publisher of this book have used their best efforts in preparing this book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The author and publisher make no warranty of any kind, expressed or implied, with regard to these programs or the documentation contained in this book. The author and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

All rights reserved. No part of this book may be reproduced, in any form or by any means, without permission in writing from the publisher.

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

ISBN 0-13-880187-0

Prentice-Hall International (UK) Limited, London
Prentice-Hall of Australia Pty. Limited, Sydney
Prentice-Hall Canada Inc., Toronto
Prentice-Hall Hispanoamericana, S.A., Mexico
Prentice-Hall of India Private Limited, New Delhi
Prentice-Hall of Japan, Inc., Tokyo
Simon & Schuster Asia Pte. Ltd., Singapore
Editors Prentice-Hall do Brasil, Ltda., Rio de Janeiro

TRADEMARK INFORMATION

IBM PC is a registered trademark of International Business Machines Corporation.
UNIX is a registered trademark of AT&T (Bell Laboratories).
PDP II and VAX are registered trademarks of Digital Equipment Corporation.
MS-DOS is a trademark of Microsoft Corporation.
Atari is a trademark of Atari Corporation.
SPARC is a trademark of Sun Microsystems.
Macintosh is a trademark of Apple Computer, Inc.
Intel is a registered trademark of Intel Corporation.

1

INTRODUCTION

Without its software, a computer is basically a useless lump of metal. With its software, a computer can store, process, and retrieve information, find spelling errors in manuscripts, play adventure, and engage in many other valuable activities to earn its keep. Computer software can be roughly divided into two kinds: the system programs, which manage the operation of the computer itself, and the application programs, which solve problems for their users. The most fundamental of all the system programs is the operating system, which controls all the computer's resources and provides the base upon which the application programs can be written.

A modern computer system consists of one or more processors, some main memory (often known as "core memory," even though magnetic cores have not been used in memories for over a decade), clocks, terminals, disks, network interfaces, and other input/output devices. All in all, a complex system. Writing programs that keep track of all these components and use them correctly, let alone optimally, is an extremely difficult job. If every programmer had to be concerned with how disk drives work, and with all the dozens of things that could go wrong when reading a disk block, it is unlikely that many programs could be written at all.

Many years ago it became abundantly clear that some way had to be found to shield programmers from the complexity of the hardware. The way that has gradually evolved is to put a layer of software on top of the bare hardware, to manage all parts of the system, and present the user with an interface or virtual machine that is easier to understand and program. This layer of software is the operating system, and forms the subject of this book.

The situation is shown in Fig. 1-1. At the bottom is the hardware, which in many

cases is itself composed of two or more layers. The lowest layer contains physical devices, consisting of integrated circuit chips, wires, power supplies, cathode ray tubes, and similar physical devices. How these are constructed and how they work is the province of the electrical engineer.

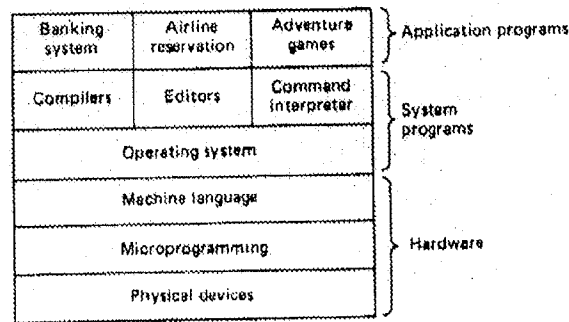


Fig. 1-1. A computer system consists of hardware, system programs, and application programs.

Next comes a layer of primitive software that directly controls these devices and provides a cleaner interface to the next layer. This software, called the **microprogram**, is usually located in read-only memory. It is actually an interpreter, fetching the machine language instructions such as **ADD**, **MOVE**, and **JUMP**, and carrying them out as a series of little steps. To carry out an **ADD** instruction, for example, the microprogram must determine where the numbers to be added are located, fetch them, add them, and store the result somewhere. The set of instructions that the microprogram interprets defines the **machine language**, which is not really part of the hard machine at all, but computer manufacturers always describe it in their manuals as such, so many people think of it as being the real "machine." On some machines the microprogram is implemented in hardware, and is not really a distinct layer.

The machine language typically has between 50 and 300 instructions, mostly for moving data around the machine, doing arithmetic, and comparing values. In this layer, the input/output devices are controlled by loading values into special device registers. For example, a disk can be commanded to read by loading the values of the disk address, main memory address, byte count, and direction (**READ** or **WRITE**) into its registers. In practice, many more parameters are needed, and the status returned by the drive after an operation is highly complex. Furthermore, for many I/O devices, timing plays an important role in the programming.

A major function of the operating system is to hide all this complexity and give the programmer a more convenient set of instructions to work with. For example, **READ BLOCK FROM FILE** is conceptually simpler than having to worry about the details of moving disk heads, waiting for them to settle down, and so on.

On top of the operating system is the rest of the system software. Here we find the command interpreter (shell), compilers, editors and similar application-independent programs. It is important to realize that these programs are definitely

not part of the operating system, even though they are typically supplied by the computer manufacturer. This is a crucial, but subtle, point. The operating system is that portion of the software that runs in **kernel mode** or **supervisor mode**. It is protected from user tampering by the hardware (ignoring for the moment some of the older microprocessors that do not have hardware protection at all). Compilers and editors run in **user mode**. If a user does not like a particular compiler, he[†] is free to write his own if he so chooses; he is not free to write his own disk interrupt handler, which is part of the operating system and is normally protected by hardware against attempts by users to modify it.

Finally, above the system programs come the application programs. These programs are written by the users to solve their particular problems, such as commercial data processing, engineering calculations, or game playing.

1.1. WHAT IS AN OPERATING SYSTEM?

Most computer users have had some experience with an operating system, but it is difficult to pin down precisely what an operating system is. Part of the problem is that operating systems perform two basically unrelated functions, and depending on who is doing the talking, you hear mostly about one function or the other. Let us now look at both.

1.1.1. The Operating System as an Extended Machine

As mentioned earlier, the **architecture** (instruction set, memory organization, I/O and bus structure) of most computers at the machine language level is primitive and awkward to program, especially for input/output. To make this point more concrete, let us briefly look at how floppy disk I/O is done using the NEC PD765 controller chip, which is used on the IBM PC and many other personal computers. (Throughout this book we will use the terms "floppy disk" and "diskette" interchangeably.) The PD765 has 16 commands, each specified by loading between 1 and 9 bytes into a device register. These commands are for reading and writing data, moving the disk arm, and formatting tracks, as well as initializing, sensing, resetting, and recalibrating the controller and the drives.

The most basic commands are READ and WRITE, each of which requires 13 parameters, packed into 9 bytes. These parameters specify such items as the address of the disk block to be read, the number of sectors per track, the recording mode used on the physical medium, the intersector gap spacing, and what to do with a deleted-data-address-mark. If you do not understand this mumbo jumbo, do not worry, that is precisely the point—it is rather esoteric. When the operation is completed, the controller chip returns 23 status and error fields packed into 7 bytes. As if this were not enough, the floppy disk programmer must also be constantly aware of whether the

[†]"He" should be read as "he or she" throughout the book.

1.4.1. Monolithic Systems

By far the most common organization, this approach might well be subtitled "The Big Mess." The structure is that there is no structure. The operating system is written as a collection of procedures, each of which can call any of the other ones whenever it needs to. When this technique is used, each procedure in the system has a well-defined interface in terms of parameters and results, and each one is free to call any other one, if the latter provides some useful computation that the former needs.

To construct the actual object program of the operating system when this approach is used, one compiles all the individual procedures, or files containing the procedures, and then binds them all together into a single object file with the linker. In terms of information hiding, there is essentially none—every procedure is visible to every other one (as opposed to a structure containing modules or packages, in which much of the information is local to a module, and only officially designated entry points can be called from outside the module).

Even in monolithic systems, however, it is possible to have at least a little structure. The services (system calls) provided by the operating system are requested by putting the parameters in well-defined places, such as in registers or on the stack, and then executing a special trap instruction known as a **kernel call** or **supervisor call**.

This instruction switches the machine from **user mode** to **kernel mode** (also known as **supervisor mode**), and transfers control to the operating system, shown as event (1) in Fig. 1-8. (Most CPUs have two modes: kernel mode, for the operating system, in which all instructions are allowed; and user mode, for user programs, in which I/O and certain other instructions are not allowed.)

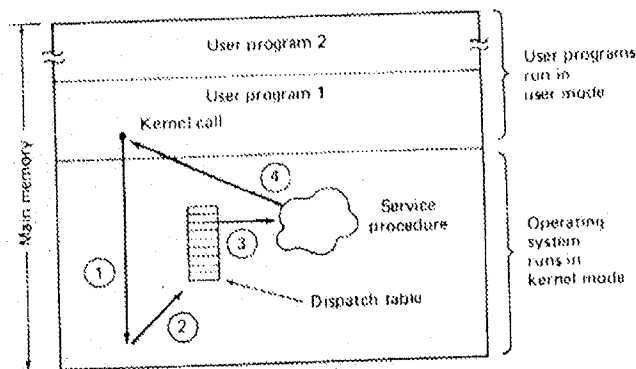


Fig. 1-8. How a system call can be made: (1) User program traps to the kernel. (2) Operating system determines service number required. (3) Operating system locates and calls service procedure. (4) Control is returned to user program.

The operating system then examines the parameters of the call to determine which system call is to be carried out, shown as (2) in Fig. 1-8. Next, the operating system indexes into a table that contains in slot k a pointer to the procedure that

the lock has been released. In order to successfully place a lock, every byte in the region to be locked must be available.

When placing a lock, a process must specify whether it wants to block or not in the event that lock cannot be placed. If it chooses to block, when the existing lock has been removed, the process is unblocked and the lock is placed. If the process chooses not to block when it cannot place a lock, the system call returns immediately, with the status code telling whether the lock succeeded or not.

Locked regions may overlap. In Fig. 7-11(a) we see that process *A* has placed a shared lock on bytes 4 through 7 of some file. Later, process *B* places a shared lock on bytes 6 through 9, as shown in Fig. 7-11(b). Finally, *C* locks bytes 2 through 11. As long as all these locks are shared, they can co-exist.

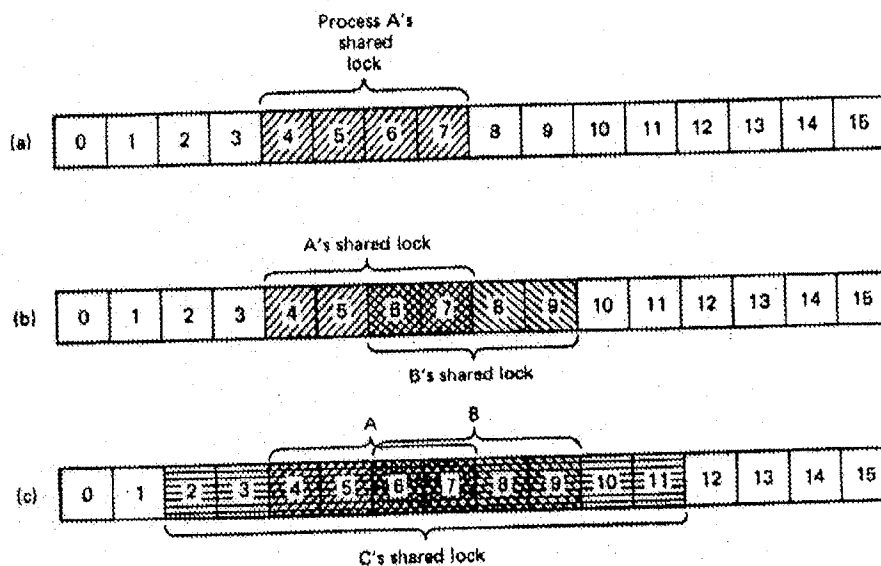


Fig. 7-11. (a) A file with one lock. (b) Addition of a second lock. (c) A third lock.

Now consider what happens if a process tries to acquire an exclusive lock to byte 9 of the file of Fig. 7-11(c), with a request to block if the lock fails. Since two previous locks cover this block, the caller will block and will remain blocked until both *B* and *C* release their locks.

7.3.4. Input/Output in UNIX

Like all computers, those running UNIX have I/O devices such as terminals, disks, printers, and networks connected to them. Some way is needed to allow programs to access these devices. Although various solutions are possible, the UNIX one is to integrate them into the file system as what are called **special files**. Each I/O device is assigned a path name, usually in */dev*. For example, the printer might be */dev/lp*, terminal 1 might be */dev/tty1*, and the network might be */dev/net*.

These special files can be accessed the same way as any other files. No special commands or system calls are needed. The usual READ and WRITE system calls will do just fine. For example, the command

```
cp file /dev/lp
```

copies the *file* to printer, causing it to be printed (assuming that this is permitted). Programs can open, read, and write special files the same way as they do regular files. In fact, *cp* in the above example is not even aware that it is printing. In this way, no special mechanism is needed for doing I/O.

An additional advantage is that the usual file protection rules apply automatically to I/O devices. If the protection bits for */dev* are set up to prohibit everyone except the superuser from directly accessing the files in it, then users are prohibited from doing direct I/O themselves. Restricted access to selected I/O devices can be given by installing *setuid* programs that have permission to read and write the files in */dev*, but do that in limited ways.

For example, a common way to manage access to the printer is to make */dev/lp* readable for no one and writable for only the superuser. A program called *lpr* is offered to allow users to print files. What *lpr* does is copy the files specified in its arguments to a spooling directory, where a daemon with access to */dev/lp* takes them out and prints them in order. It should be clear that */dev/lp* need not necessarily be owned by the superuser. All that matters is that the daemon has access to it. They could both be owned by *daemon* for example. By having the I/O devices integrated into the file system like this, great flexibility in access can be achieved.

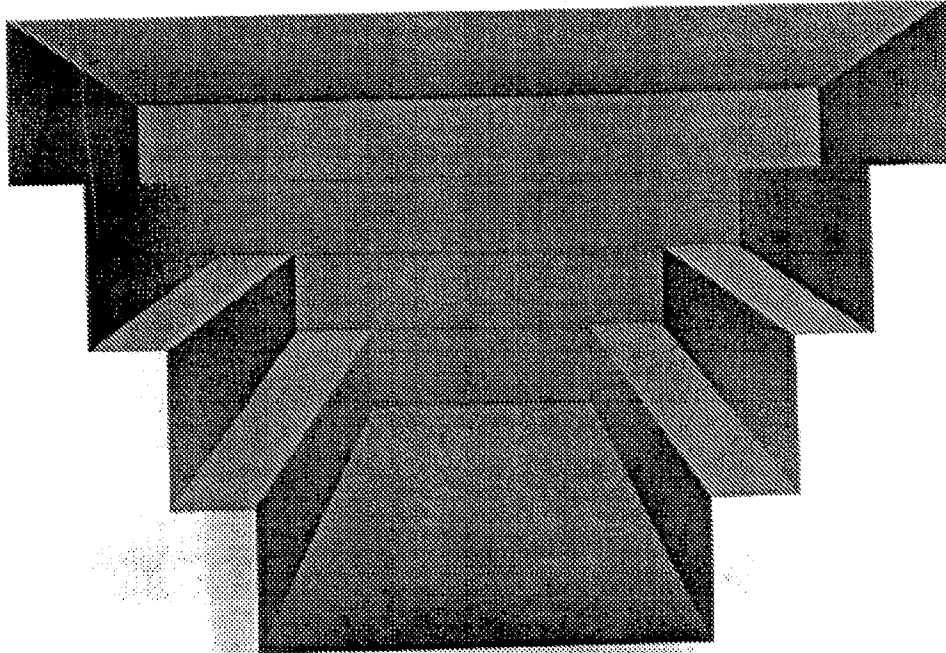
Special files are divided into two categories, block and character. A **block special file** is one consisting of a sequence of numbered blocks. The key property of the block special file is that each block can be individually addressed and accessed. In other words, a program can open a block special file and read, say, block 124 without first having to read blocks 0 to 123. Block special files are used for disks.

Character special files are normally used for devices that input or output a character stream. Terminals, printers, networks, mice, plotters, and most other I/O devices that accept or produce data for people use character special files. It is not possible (or even meaningful) to seek to block 124 on a mouse.

Although character special files cannot be randomly accessed, they often need to be controlled in ways that block special files do not. Consider, for example, a terminal. In addition to accepting read and write requests, a terminal has a number of special characteristics that must be managed. For example, when the user makes a typing error and wants to erase the last character typed, he presses some key. Some people prefer to use backspace, and others prefer DEL. Similarly, to erase the entire line just typed, many conventions abound. Some users prefer @, while others prefer CTRL-U, CTRL-C, or other character. Likewise, to interrupt the running program, some special key must be hit. Here too, different people have different preferences.

Rather than making a choice and forcing everyone to use it, UNIX allows all these special functions and many others to be customized by the user. A special system call is generally provided for setting these options. This system call also handles tab expansion, enabling and disabling of character echoing, conversion between carriage

Exhibit 3



STRUCTURED COMPUTER ORGANIZATION

SECOND EDITION

ANDREW S. TANENBAUM

Library of Congress Cataloging in Publication Data

Tanenbaum, Andrew S. (date)
Structured Computer Organization

Includes index.

1. Electronic digital computers—Programming.
I. Title
QA76.6.T38 1984 001.64'2 83-2916
ISBN 0-13-854489-1

To Suzanne, Barbara, Marvin and the memory of Sweetie π

*Production/editorial supervision: Nancy Milnamow
Manufacturing buyer: Gordon Osbourne*

© 1984 by Prentice-Hall, Inc., Englewood Cliffs, N. J. 07632

All rights reserved. No part of this book
may be reproduced in any form or by any means
without permission in writing from the publisher.

10 9 8 7 6

Printed in the United States of America

ISBN 0-13-854489-1

PRENTICE-HALL INTERNATIONAL, INC., *London*
PRENTICE-HALL OF AUSTRALIA PTY. LTD., *Sydney*
EDITORA PRENTICE-HALL DO BRASIL LTDA., *Rio de Janeiro*
PRENTICE-HALL OF CANADA, LTD., *Toronto*
PRENTICE-HALL OF INDIA PRIVATE LTD., *New Delhi*
PRENTICE-HALL OF JAPAN, INC., *Tokyo*
PRENTICE-HALL OF SOUTHEAST ASIA PTE. LTD., *Singapore*
WHITEHALL BOOKS LTD., *Wellington, New Zealand*

The control pins regulate the flow and timing of data to and from the microprocessor and have other miscellaneous uses. Their number and function vary considerably among microprocessors. Some pins regulate memory access, others deal with I/O, and some are needed for power, ground, and clock signals. We will describe the control pins in more detail below, after having described how the various components of a microcomputer are interconnected.

3.4.2. Microcomputer Buses

The components of a microcomputer, the microprocessor, memory, and I/O controller chips, are connected by a collection of parallel wires called a **bus**. The lines (wires) in the bus can be classified as address, data, or control. They correspond closely but not exactly, to the microprocessor's pins. Several buses have been standardized by IEEE and other organizations to facilitate interconnection of processors, memories, and other devices from different vendors (Boberg, 1980; Burr et al., 1979; Elmquist et al., 1979; Gilbert, 1982).

Figure 3-31 illustrates a simple microcomputer with a $2K \times 8$ EPROM for the program, a $2K \times 8$ RAM for data, and an I/O chip. In this figure, the 16 address lines (called the **address bus**), the 8 data lines (called the **data bus**), and the (unspecified number of) control lines (called the **control bus**) are shaded to indicate that more than one line is present.

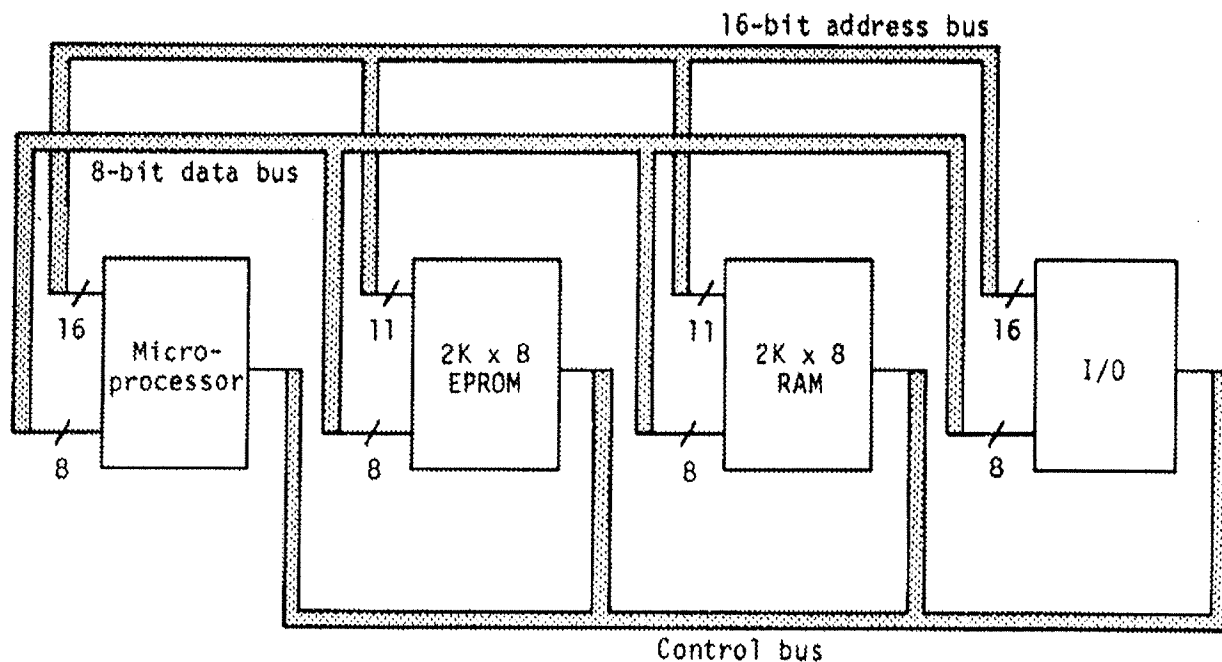


Fig. 3-31. A microcomputer with EPROM, RAM, and an I/O chip.

Another convention used in the figure is the short diagonal line with a number, telling how many bits are meant. For example, the microprocessor is connected to all

Exhibit 4

Windows NT[®] Performance: Monitoring, Benchmarking, and Tuning

New
Riders

Mark T. Edmead
Paul Hinsberg

Windows NT Performance Monitoring, Benchmarking, and Tuning

Mark T. Edwards, Paul Hirschberg

Copyright © 1998 by New Riders Publishing

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without written permission from the publisher, except for the inclusion of brief quotations in a review.

International Standard Book Number: 1-56205-942-4

Library of Congress Catalog Card Number: 98-86489

Printed in the United States of America

2001 00 99 98 4 3 2 1

Interpretation of the printing code: The rightmost double-digit number is the year of the book's printing; the rightmost single-digit, the number of the book's printing. For example, the printing code 98-1 shows that the first printing of the book occurred in 1998.

Composed in *Baskin* and *Roth Sans Serif* by *Macmillan Computer Publishing*

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. *New Riders Publishing* cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The authors and the publisher shall have neither liability or responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Executive Editor
Linda Ratts Engelmann

Acquisitions Editor
Karen Wachs

Development Editor
Christopher Cleveland

Managing Editor
Caroline Roop

Project Editor
Brad Herriman

Copy Editor
Kris Simmons

Indexer
Tim Wright

Technical Editor
Will Adams

Proofreader
Sheri Replin

Production
Steve Balle-Gifford
Louis Porter, Jr.

Network Resources

Networking is built in to Windows NT, which includes a basic peer-to-peer network server that communicates using the Server Message Block (SMB) protocol. This SMB protocol is what makes NT compatible with the old Microsoft Network and LAN Manager. As with many other operating systems, NT is designed to follow the Open Systems Interconnect (OSI) seven-layer model. Chapter 4, "Determining System Performance Objectives," discusses more on the specifics of each layer.

You should be familiar with two components of the NT networking architecture: the network redirector and the network server.

The redirector provides the facilities for one NT machine to access other resources on other machines on the network. The Windows NT redirector, for instance, can access remote files and printers. Because the redirector uses the SMB protocol, it is completely compliant with DOS, Windows, and even OS/2 systems. You can think of the redirector components as the "client" side of the client/server architecture. This means that as the client, the redirector wants to connect to a server resource somewhere on the network.

The network server is also compliant with the SMB protocol. You can think of the network server as the "server" side of the client/server architecture. This means that the network server is responsible for accepting and processing requests from client machines. Because NT is SMB-compliant, it can accept requests not only from other Windows NT systems, but also from other systems running LAN Manager software.

To communicate with different types of network systems, NT needs to allow for the installation and use of different transport protocols. In Windows NT, transport protocols are implemented as drivers. To avoid the problem of letting a particular transport protocol know which type of input the protocol driver expects, NT implements what is called a transport driver interface (TDI). TDI is a single programming interface that redirectors and other high-level network drivers use. It is the TDI that allows redirectors and network servers to be independent from the transport layer of the OSI model.

I/O Bus Architecture

When an application is loaded and running, the program will more than likely need to make some type of request to either get or receive data from a peripheral. In fact, this requirement exists even before the program runs. When a program executes, the operating system must first get the program from the hard drive. Part of the program is then transferred from the disk (or whatever media the program resides in). Most Intel-based systems provide the option of installing interface cards into different bus types. The system contains a bus controller that is responsible for servicing the request to and from the peripheral device.

The I/O architecture plays an important role in system optimization. Regardless of how much memory you have or how fast your processor is, if the data is moving to the peripheral at a slow rate, that will become your bottleneck.

Intel systems provide an 8-bit, 16-bit, or 32-bit bus. Obviously, the 32-bit bus is the ideal implementation. The slower of the available bus architectures is the ISA (Industry Standard Architecture) bus. The ISA bus has a transfer rate of 5MB/sec. ISA includes both a 8-bit and 16-bit I/O bus. A Peripheral Component Interconnect (PCI) bus, on the other hand, is a 32-bit bus, which means that it can address the full 4GB. The PCI bus is also a 133MB/sec bus, which is a significant improvement from the ISA bus architecture.

Be on the lookout for a faster PCI bus architecture. Currently, developers are working on a new PCI bus that will increase the speed to 166MB/sec.

Summary

To understand the available choices for optimizing the system, it is important to know the NT architecture. NT is installed on a computer system, and you have many choices for various computer system resources, such as memory, processor, hard disk, network, and I/O bus architecture.

Windows NT was designed with an eye toward modularity and robustness. Unfortunately, NT was also designed with backward compatibility in mind, which means it carries all the excess baggage necessary to run existing DOS and 16-bit Windows applications.

Subsequent chapters in this book address in detail optimization of the Windows NT system. This first chapter prepared you or refreshed your memory about the internal workings of NT. You will see how these NT features can sometimes adversely affect NT performance. Luckily, you can make changes to the NT system to overcome these deficiencies and make NT work as the system it was intended to be.

Hardware and Software Interaction

Before launching into simulations of the individual resources, let's examine briefly how a computer is divided into a series of components that work together to provide services to the user's applications. Figure 3.1 shows the layout of a typical Intel-based computer system.

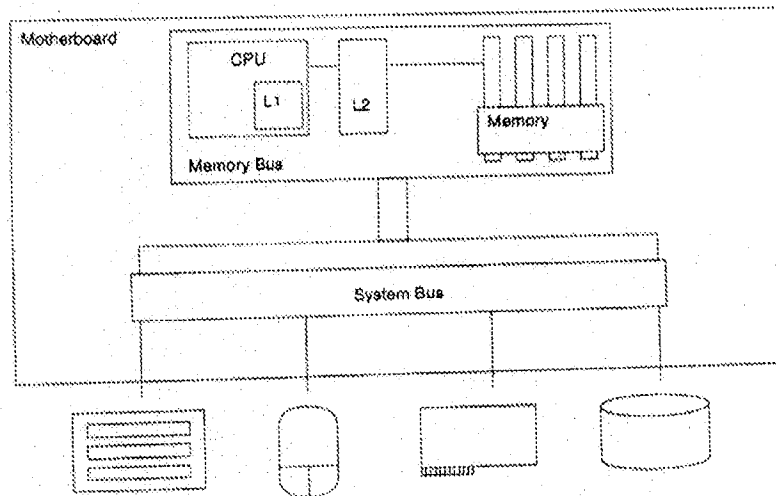


Figure 3.1 Layout of the typical Intel hardware computer system.

At the heart of the system is a central processing unit (CPU). This processor has its own Level-1 cache memory and, in addition, could have a secondary Level-2 cache. In Windows NT, the system can have more than one processor, each with its own Level-1 or Level-2 cache memory. All of the processors, however, share the same physical RAM. The RAM and the CPU communicate via a high-speed 32-bit bus.

Looking Ahead

Later in Chapter 6, "Optimizing CPU Performance," you will take a closer look at the CPU architecture. You will see how the architecture of the CPU depends on not only the manufacturer, such as Intel or AMD, but also the model. For example, the memory bus and cache structure of a Pentium Pro is different from that of a Pentium II. Also, Chapter 6 has a discussion on the newer technologies that are emerging. Technologies such as SDRAM and 100MHz memory bus systems will further enhance processor performance.

Connected to this 32-bit bus is an I/O controller that is responsible for the interface of the CPU to the attached peripherals. Possible peripherals connected in this manner include hard disks, CD-ROMs, network cards, mice, keyboards, and so on. Many of these devices are connected directly to the motherboard, but many require additional interface cards, such as a SCSI scanner device, which requires a SCSI card.

In the Intel platform architecture, the I/O bus architecture can be 8-bit, 16-bit, or 32-bit. This architecture depends on the type of I/O bus that was built in to the PC. Bus specifications include ISA, EISA, and PCI. The ISA is the oldest of the technologies and typically supports the 8-bit and 16-bit ranges. The EISA is a 32-bit functionality that isn't frequently used anymore. PCI is common and is usually built in to a system to run at 32-bit; however, the specification and technology can run at 64-bit. Examples of the interface cards that support these bus topologies are ISA, EISA, and PCI.

The CPU, memory, disks, peripherals, and I/O buses all work together to allow applications to load and run. If you look at a typical situation in which the user double-clicks an application, this is what happens:

1. The application resides on the hard disk. The mouse click informs the operating system that a click has occurred, and the OS interprets the click as a command to launch the application.
2. The application is executed. This means that the Process Manager (see Chapter 1, "Understanding Windows NT Architecture") allocates the resources necessary to run the application.
3. The Virtual Memory Manager (VMM) provides the application with up to 4GB of virtual memory to store code and data in memory. Part of the program instructions will reside in physical RAM, while the rest stay on the hard disk.
4. Parts of the program, which reside on the physical disk, are moved into memory via the I/O bus.
5. The instructions are moved from RAM to the CPU via the 32-bit bus. Instructions are stored in either the Level-1 or Level-2 cache (if present).
6. As the program executes, the VMM will return to the hard disk to load those parts not available in memory. The request will travel via the I/O controller to the hard disk device and back up to the memory.
7. The program instructions, executed by the processor, will contain instructions to do a variety of things such as draw text on the screen, write data to the network, read data from a floppy, and so on.
8. These requests will again travel through the I/O controller to the appropriate peripheral device.

The resource bottleneck can occur at any or all of these stages. The major areas that this book covers are memory, processor, disk, and network resource bottlenecks.

**Exhibit B – definition of “tailored”
from WEBSTER’S NEW TWENTIETH
CENTURY DICTIONARY 1858 (2d
ed. 1983).**

WEBSTER'S
NEW UNIVERSAL
UNABRIDGED
DICTIONARY

WEBSTER'S
NEW UNIVERSAL
UNABRIDGED
DICTIONARY

DELUXE
SECOND EDITION

BASED UPON THE BROAD FOUNDATIONS LAID DOWN BY

Noah Webster

EXTENSIVELY REVISED BY THE PUBLISHER'S EDITORIAL STAFF UNDER THE GENERAL SUPERVISION OF

JEAN L. McKECHNIE

INCLUDING ETYMOLOGIES, FULL PRONUNCIATIONS, SYNONYMS, AND AN ENCYCLOPEDIA SUPPLEMENT OF
GEOGRAPHICAL AND BIOGRAPHICAL DATA, SCRIPTURE PROPER NAMES, FOREIGN WORDS AND PHRASES,
PRACTICAL BUSINESS MATHEMATICS, ABBREVIATIONS, TABLES OF WEIGHTS AND MEASURES, SIGNS AND
SYMBOLS, AND FORMS OF ADDRESS

ILLUSTRATED THROUGHOUT

Dorset & Baber

WEBSTER'S NEW TWENTIETH CENTURY DICTIONARY

Second Edition

Copyright © 1983 and 1955, 1956, 1957, 1958, 1959, 1960, 1962, 1964,
1968, 1970, 1975, 1977, 1979 by Simon & Schuster, a Division of Gulf & Western Corporation
Full-Color Plates Copyright © 1972 by Simon & Schuster, a Division of Gulf & Western Corporation
All rights reserved
including the right of reproduction
in whole or in part in any form
Published by New World Dictionaries/Simon and Schuster
A Simon & Schuster Division of Gulf & Western Corporation
Simon & Schuster Building
Rockefeller Center
1230 Avenue of the Americas
New York, New York 10020
SIMON AND SCHUSTER, TREE OF KNOWLEDGE and colophon are trademarks
of Simon & Schuster.

*Dictionary Editorial Offices
New World Dictionaries
850 Euclid Avenue
Cleveland, Ohio 44114*

Manufactured in the United States of America

K 20 19 18

Library of Congress Catalog Card Number: 83-42537

ISBN 0-671-41819-X

Previous editions of this book were published by The World Publishing Company, William Collins + World Publishing Co., Inc. and William Collins Publishers, Inc.

tailed, *a.* having a (specified kind of) tail; caudate; principally in combination, as in *short-tailed*, *bobtailed*, *ring-tailed*, etc.
 2. in botany, having a taillike appendage.
 3. shaped like a tail.

tail end, 1. the rear or bottom end of anything.
 2. the concluding or last part of anything.
 3. [*pl.*] inferior samples of corn.

tail feath'er (feth'), *n.* a reatrix; one of the feathers of a bird's tail.

tail fin, the caudal fin of a fish.

tail'flow'er, *n.* any plant of the genus *Anthurium*.

tail gate, the gate at the lower end of a canal lock.

tail'gate, *n.* a tailboard, as of a wagon.

tail grape, any climbing shrub of the genus *Aralobryx*, native to tropical Africa and Asia, the fruit of which is supported by a recurving peduncle.

tail'ing, *n.* 1. [*pl.*] waste or refuse in various processes of milling, mining, distilling, etc.
 2. in building, the part of a projecting brick, stone, etc. embedded in a wall.
 3. a defect in printing calico caused by an imperfect process.
 4. in electricity, (a) a residual discharge affecting the receiver of a telegraph system, tending to make the signals run together; (b) a residual or return charge or current in the transmission of electromagnetic waves through a dielectric.

tail lamp, a taillight.

taille (tāl), *n.* [Fr. from *tailler*, to cut.]
 1. a tally. [Obs.]
 2. a French feudal tax imposed by the king or a lord.
 3. form or shape, especially of the bust.
 4. the waist of a dress or its fit, cut, etc.

tail'less, *a.* having no tail.

tail'leur' (tä-yēr'), *n.* [Fr. *tailleur*, to cut.] the dealer or banker in various French card games.

tail'lie, *n.* [Fr. *tailleur*, to cut.] in Scots law, tail.

tail'light, *n.* a light at the back of a vehicle to warn approaching vehicles of its presence at night; also *tail lamp*.

tail'loir' (tä-yvär'), *n.* [Fr. *taillier*, to cut.] in architecture, an abacus.

tail'lor, *n.* [Fr. *tailleur*, from *tailler*, to cut.]
 1. a person who makes, repairs, or alters clothes, especially suits, coats, etc.
 2. in zoology, (a) a tailorbird; (b) the silver-sides.
merchant tailor; see under *merchant*.
tailor's chair; a legless seat with a back rest, allowing the occupant to sit crosslegged.
tailor's muscle; the sartorius.

tail'lor, *v.i.*; tailored, *pl.*, *pp.*; tailoring, *ppr.*
 1. to work as a tailor.
 2. to deal with tailors, as for clothing.

tail'lor, *v.t.* 1. to make (clothes) by tailor's work.
 2. to fit or provide (a person) with clothes made by a tailor.
 3. to cut, form, produce, alter, etc. so as to meet requirements or particular conditions; as, her novel is *tailored* to popular tastes.
 4. to fashion (women's garments, etc.) with trim, simple lines like those of men's clothes.

tail'lor-bird, *n.* one of various birds, as those of the genus *Orthotomus* or allied genera, constructing a nest by sewing leaves together and lining the case with some soft substance.

tail'lor-ess, *n.* a woman tailor.

tail'lor-ing, *n.* 1. the business of a tailor.
 2. the workmanship or skill of a tailor.

tail'lor-mäde, *a.* made by or as by a tailor or according to his methods; specifically, (a) made with trim, simple lines; said of a woman's garment; (b) made to order or to meet particular conditions; as, furniture *tailor-made* for the small apartment.

tail'piece, *n.* 1. a piece or part added to or forming the end of something, as (a) in a lathe, the tailpin; (b) in mining, a snore piece.
 2. the small triangular piece of wood at the lower end of a violin, cello, etc., to which the strings are attached.
 3. a short beam or rafter with one end tailed in a wall and the other supported by a header.
 4. in printing, an ornamental design, engraving, etc. put at the end of a chapter or at the bottom of a page.
 5. in entomology, one of the parts making up a pygidium.

tail'pin, *n.* the adjustable screw on a rear spindle.

tail'pipe, *n.* 1. an exhaust pipe at the rear of an automotive vehicle.
 2. the exhaust duct of a jet engine.

tail plane, a horizontal supporting surface at the rear of an aircraft; a stabilizer.

tail'race, *n.* 1. the lower part of a millrace.
 2. the channel through which water flows after going over a water wheel.
 3. a water channel to carry away tailings from a mine.

tail'spin, *n.* the descent of an airplane with nose down and tail spinning in circles overhead; often used figuratively; also *tail spin*.

tail'stock, *n.* the adjustable part of a lathe, containing the dead center which holds the work.

tail switch'ing (swich'), a method of switching trains at stations by means of which they may be drawn out tail end first.

tail wa'ter, the water flowing from the buckets of a water wheel in motion.

tail wind, a wind blowing from behind an airplane, ship, etc. in motion.

tail'zie, *n.* [Fr. *taillier*, to cut.] in Scots law, tail.

tain, *n.* [ME. *tainc*, a thin plate. L. *tainia*, a band.] thin tin plate; also, tin foil for mirrors.

Tai'nō, *n.*; *pl.* Tai'nōs, 1. [*pl.*] a member of an extinct, aboriginal Indian tribe of the West Indies.
 2. its Arawakan language.

tain't, *v.t.*; tainted, *pl.*, *pp.*; tainting, *ppr.* [contr. from *attaint*; meaning influenced by Fr. *teindre*, pp. of *teindre*, from L. *tingere*, to wet, moisten.]
 1. to affect with something physically injurious, unpleasant, etc.; to infect; to spoil.
 2. to make morally corrupt or depraved; as, greed *tainted* his mind.
 3. to dye; to color. [Obs.]
 4. to sully or stain (a person's honor). [Obs.]
 5. to mollify. [Obs.]

tain't, *v.i.* 1. to become tainted.
 2. to be affected with incipient putrefaction, as meat.

tain't, *n.* 1. tincture; stain; color. [Obs.]
 2. an infectious or contaminating trace; infection; contamination.
 3. a trace of corruption, evil, disgrace, etc.

tain't, *n.* [shortened form of *attaint*.] a hit in tilting. [Obs.]

tain't, *v.t.* to make a thrust in tilting. [Obs.]

tain't, *v.t.* to touch as in tilting; also, to thrust, as a lancet. [Obs.]

tain't'less, *a.* without taint or infection; pure.

tain't'less-ly, *adv.* without taint; in a taintless manner or way.

tain't'ör, *n.* a dyer. [Rare.]

tain't'üre, *n.* taint; tinge; defilement; stain; spot. [Obs.]

tain't'worm, *n.* any parasitic worm injurious to plant or animal life.

Tai'ping (tī'), *a.* [Chin. *t'ai-p'ing*, great peace: designation of the dynasty that was to be established.] designating or of a rebellion (1850-1864) against the Manchu dynasty, led by Hung Siu-tuan.

ta'i'ra, *ta'y'ra*, *n.* [S.Am.] a South American carnivore, *Galera barbura*, resembling the weasel.

ta'irge, *v.t.* [ME. *taryen*, to tarry, delay; OFr. *targer*, to delay.] to censure. [Scot.]

ta'irn, *n.* a tarn. [Scot.]

ta'isch, *n.* [Gael.] the phantom or voice of a person about to die. [Scot.]

ta'it, *n.* [Australian.] an Australian marsupial, *Tarsipes rostratus*, about the size of the mouse, feeding principally upon honey and insects.

ta'j, *n.* [Per., from Ar.] a distinguishing head-dress; a high cap such as is worn by Mohammedan dervishes.

Tä'jik, *n.*; *pl.* Tä'jik, same as *Tadzhik*.

Tä'j Mä-häl', [Per., best of buildings.] the famous mausoleum at Agra, India, built (1630?-1648?) by Shah Jahan for his favorite wife.

tä'kä, *n.*; *pl.* tä'kä, [from Sans. *lōṅkō*, silver coins.] the monetary unit of Bangladesh.

tä'ke, *v.t.*; took, *pl.*; taken, *pp.*; taking, *ppr.* [ME. *taken*; (late) AS. *tacan*, from ON. *taka*.]
 1. to get by conquering; to capture; to seize.
 2. to trap or snare (a bird, animal, or fish).
 3. (a) to win, as a game, a part of a game, or a trick at cards; (b) to remove (an opponent's piece) from play by capturing.
 4. to get hold of; to grasp.
 5. to hit (a person) in or on some part.
 6. to affect; to attack; as, he was *taken* by violent shaking.

7. to catch (a person) in some act, especially a fault.
 8. to capture the fancy of; to charm.
 9. to get into one's hand or hold; to transfer to oneself.
 10. to eat, drink, swallow, etc. for bodily nourishment.
 11. to get benefit from by exposure to; as, she *took* the air. [Rare.]
 12. to enter into a special relationship with; as, she *took* students to add to her income; he *took* a wife.
 13. to buy; as, he *took* the first suit that the clerk offered.
 14. to rent or lease; as, we *took* a cottage for the summer.
 15. to get regularly by paying for; as, we *take* two daily newspapers.
 16. to assume as a responsibility, task, etc.; as, he *took* the job.
 17. to assume or adopt (a badge or symbol of duty, office, etc.); as, the president *took* the chair.
 18. to obligate oneself by; as, he *took* a vow.
 19. to become a member of; to join, as a party or side in a contest, disagreement, etc.
 20. to assume (something) as if granted or due one; as, he *took* the blame; she *took* her leave.
 21. to cheat; to trick. [Slang.]
 22. in grammar, to have or admit of according to usage, nature, etc.; to be used with in construction; as, a transitive verb *takes* an object.
 23. to choose; to select.
 24. to use or employ; to resort to; as, he *took* a whip to his son.
 25. to travel by; to get in or on as a means of traveling; as, she *took* a train.
 26. to go to (a place) for shelter, safety, etc.; as, the birds *took* cover.
 27. to deal with; to consider; as, he *took* the matter gravely.
 28. to occupy; as, *take* a chair.
 29. to require; to demand; to need; used impersonally, as, it *takes* money to make money. [Colloq.]
 30. to derive or draw, as a name, quality, etc., from.
 31. to extract, as for quotation; to excerpt; as, he *took* a verse from the Bible.
 32. to obtain by observation, experiment, study, etc.; as, he *took* a poll.
 33. to write down; to copy; as, *take* notes.
 34. to draw, photograph, etc. a likeness of; as, let me *take* your picture.
 35. to win, as a prize, reward, etc.
 36. to be the object of; to undergo; as, *take* punishment.
 37. to occupy oneself in; to enjoy; as, *take* a nap.
 38. to accept (something offered); as, *take* a bet; *take* advice.
 39. to have a specified reaction to; as, he *took* the joke in earnest.
 40. to confront and get over, through, etc.; as, the horse *took* the jump.
 41. to be affected by (a disease, etc.); as, he *took* cold.
 42. to absorb; to become impregnated with, as a dye, polish, etc.
 43. (a) to understand the remarks of (a person); (b) to comprehend the meaning of (words, remarks, etc.); (c) to understand or interpret in a specified way.
 44. to suppose; to presume; as, I *take* him to be an intelligent person.
 45. to have or feel, as an emotion, mental state, etc.; as, *take* pity, *take* notice.
 46. to hold and act upon, as an idea, resolution, etc.
 47. to do; to perform (an act); as, *take* a walk.
 48. to make or put forth as the result of thought, as a resolution or objection.
 49. to aim and execute (a specified action) at an object; as, he *took* a short jab at his opponent. [Colloq.]
 50. to conduct; to lead; as, this path *takes* you to the river.
 51. to carry; as, *take* your skates with you.
 52. to remove from a person or thing; to extract; as, the thief *took* the silver.
 53. to remove by death.
 54. to subtract; as, the storekeeper *took* a dollar from the price.
 55. to direct (oneself); to go.
to take amiss; (a) originally, to be wrong concerning; to mistake; (b) to misunderstand the reason behind (an act); to become offended at.
to take at one's word; to accept as true, correct, etc.

**Exhibit C – Wikipedia - Bus
(computing),**

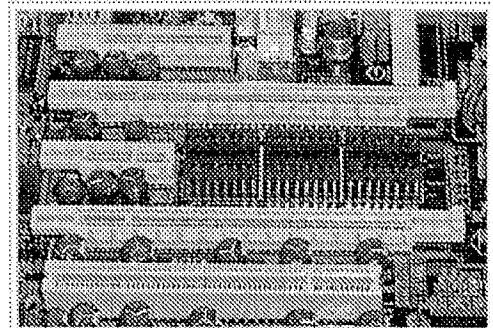
**[http://en.wikipedia.org/wiki/Bus_\(computing\)](http://en.wikipedia.org/wiki/Bus_(computing)) - (last modified on May
25, 2011)**

Bus (computing)

From Wikipedia, the free encyclopedia

In computer architecture, a **bus** is a subsystem that transfers data between components inside a computer, or between computers.

Early computer buses were literally parallel electrical wires with multiple connections, but the term is now used for any physical arrangement that provides the same logical functionality as a parallel electrical bus. Modern computer buses can use both parallel and bit-serial connections, and can be wired in either a multidrop (electrical parallel) or daisy chain topology, or connected by switched hubs, as in the case of USB.



4 PCI Express bus card slots (from top to bottom: x4, x16, x1 and x16), compared to a 32-bit conventional PCI bus card slot (very bottom)

Contents

- 1 History
 - 1.1 First generation
 - 1.2 Second generation
 - 1.3 Third generation
- 2 Description of a bus
- 3 Bus topology
- 4 Examples of internal computer buses
 - 4.1 Parallel
 - 4.2 Serial
 - 4.3 Self-repairable
- 5 Examples of external computer buses
 - 5.1 Parallel
 - 5.2 Serial
- 6 Examples of internal/external computer buses
- 7 See also
- 8 References
- 9 External links

History

First generation

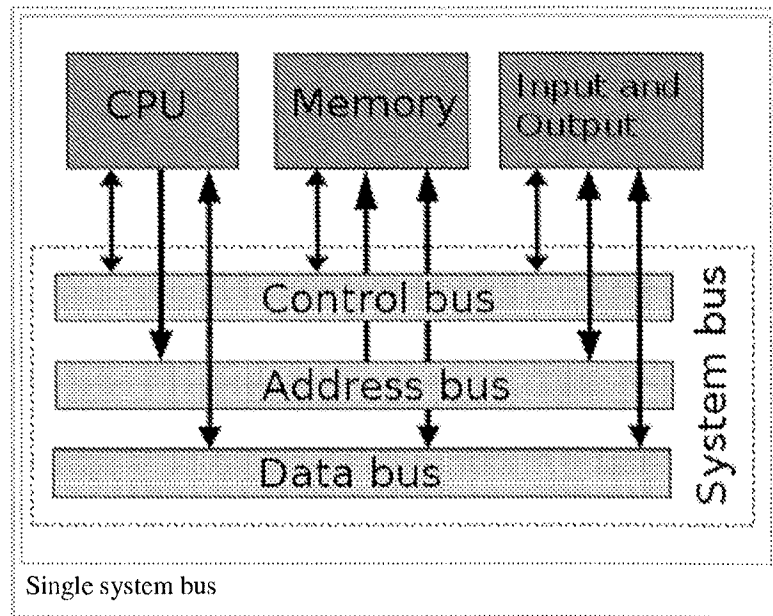
Early computer buses were bundles of wire that attached computer memory and peripherals. Anecdotally termed the "*digit trunk*",^[1] they were named after electrical power buses, or busbars. Almost always, there was one bus for memory, and another for peripherals,^[citation needed] and these were accessed by separate

instructions, with completely different timings and protocols.

One of the first complications was the use of interrupts. Early computer programs performed I/O by waiting in a loop for the peripheral to become ready. This was a waste of time for programs that had other tasks to do. Also, if the program attempted to perform those other tasks, it might take too long for the program to check again, resulting in loss of data. Engineers thus arranged for the peripherals to interrupt the CPU. The interrupts had to be prioritized, because the CPU can only execute code for one peripheral at a time, and some devices are more time-critical than others.

To provide modularity, memory and I/O buses can be combined into a unified system bus.^[2] Digital Equipment Corporation (DEC) further reduced cost for mass-produced minicomputers, and mapped peripherals into the memory bus, so that the input and output devices appeared to be memory locations. This was implemented in the Unibus of the PDP-11 around 1969.^[3]

Later computer programs began to share memory common to several CPUs. Access to this memory bus had to be prioritized, as well. The classic, simple way to prioritize interrupts or bus access was with a daisy chain.



Early microcomputer bus systems were essentially a passive backplane connected directly or through buffer amplifiers to the pins of the CPU. Memory and other devices would be added to the bus using the same address and data pins as the CPU itself used, connected in parallel. Communication was controlled by the CPU, which had read and written data from the devices as if they are blocks of memory, using the same instructions, all timed by a central clock controlling the speed of the CPU. Still, devices interrupted the CPU by signaling on separate CPU pins. For instance, a disk drive controller would signal the CPU that new data was ready to be read, at which point the CPU would move the data by reading the "memory location" that corresponded to the disk drive. Almost all early microcomputers were built in this fashion, starting with the S-100 bus in the Altair 8800 computer system.

In some instances, most notably in the IBM PC, although similar physical architecture can be employed, instructions to access peripherals (`in` and `out`) and memory (`mov` and others) have not been made uniform at all, and still generate distinct CPU signals, that could be used to implement a separate I/O bus.

These simple bus systems had a serious drawback when used for general-purpose computers. All the equipment on the bus has to talk at the same speed, as it shared a single clock.

Increasing the speed of the CPU becomes harder, because the speed of all the devices must increase as well. When it is not practical or economical to have all devices as fast as the CPU, the CPU must either enter a wait state, or work at a slower clock frequency temporarily,^[4] to talk to other devices in the

computer. While acceptable in embedded systems, this problem was not tolerated for long in general-purpose, user-expandable computers.

Such bus systems are also difficult to configure when constructed from common off-the-shelf equipment. Typically each added expansion card requires many jumpers in order to set memory addresses, I/O addresses, interrupt priorities, and interrupt numbers.

Second generation

"Second generation" bus systems like NuBus addressed some of these problems. They typically separated the computer into two "worlds", the CPU and memory on one side, and the various devices on the other. A *bus controller* accepted data from the CPU side to be moved to the peripherals side, thus shifting the communications protocol burden from the CPU itself. This allowed the CPU and memory side to evolve separately from the device bus, or just "bus". Devices on the bus could talk to each other with no CPU intervention. This led to much better "real world" performance, but also required the cards to be much more complex. These buses also often addressed speed issues by being "bigger" in terms of the size of the data path, moving from 8-bit parallel buses in the first generation, to 16 or 32-bit in the second, as well as adding software setup (now standardised as Plug-n-play) to supplant or replace the jumpers.

However these newer systems shared one quality with their earlier cousins, in that everyone on the bus had to talk at the same speed. While the CPU was now isolated and could increase speed without fear, CPUs and memory continued to increase in speed much faster than the buses they talked to. The result was that the bus speeds were now very much slower than what a modern system needed, and the machines were left starved for data. A particularly common example of this problem was that video cards quickly outran even the newer bus systems like PCI, and computers began to include AGP just to drive the video card. By 2004 AGP was outgrown again by high-end video cards and other peripherals and has been replaced by the new PCI Express bus.

An increasing number of external devices started employing their own bus systems as well. When disk drives were first introduced, they would be added to the machine with a card plugged into the bus, which is why computers have so many slots on the bus. But through the 1980s and 1990s, new systems like SCSI and IDE were introduced to serve this need, leaving most slots in modern systems empty. Today there are likely to be about five different buses in the typical machine, supporting various devices.

Third generation

"Third generation" buses have been emerging into the market since about 2001, including HyperTransport and InfiniBand. They also tend to be very flexible in terms of their physical connections, allowing them to be used both as internal buses, as well as connecting different machines together. This can lead to complex problems when trying to service different requests, so much of the work on these systems concerns software design, as opposed to the hardware itself. In general, these third generation buses tend to look more like a network than the original concept of a bus, with a higher protocol overhead needed than early systems, while also allowing multiple devices to use the bus at once.

Buses such as Wishbone have been developed by the open source hardware movement in an attempt to further remove legal and patent constraints from computer design.

Description of a bus

At one time, "bus" meant an electrically parallel system, with electrical conductors similar or identical to the pins on the CPU. This is no longer the case, and modern systems are blurring the lines between buses and networks.

Buses can be parallel buses, which carry data words in parallel on multiple wires, or serial buses, which carry data in bit-serial form. The addition of extra power and control connections, differential drivers, and data connections in each direction usually means that most serial buses have more conductors than the minimum of one used in the 1-Wire and UNI/O serial buses. As data rates increase, the problems of timing skew, power consumption, electromagnetic interference and crosstalk across parallel buses become more and more difficult to circumvent. One partial solution to this problem has been to double pump the bus. Often, a serial bus can actually be operated at higher overall data rates than a parallel bus, despite having fewer electrical connections, because a serial bus inherently has no timing skew or crosstalk. USB, FireWire, and Serial ATA are examples of this. Multidrop connections do not work well for fast serial buses, so most modern serial buses use daisy-chain or hub designs.

Most computers have both internal and external buses. An *internal bus* connects all the internal components of a computer to the motherboard (and thus, the CPU and internal memory). These types of buses are also referred to as a local bus, because they are intended to connect to local devices, not to those in other machines or external to the computer. An *external bus* connects external peripherals to the motherboard.

Network connections such as Ethernet are not generally regarded as buses, although the difference is largely conceptual rather than practical. The arrival of technologies such as InfiniBand and HyperTransport is further blurring the boundaries between networks and buses. Even the lines between internal and external are sometimes fuzzy, I²C can be used as both an internal bus, or an external bus (where it is known as ACCESS.bus), and InfiniBand is intended to replace both internal buses like PCI as well as external ones like Fibre Channel. In the typical desktop application, USB serves as a peripheral bus, but it also sees some use as a networking utility and for connectivity between different computers, again blurring the conceptual distinction.

Bus topology

In a network, the master scheduler controls the data traffic. If data is to be transferred, the requesting computer sends a message to the scheduler, which puts the request into a queue. The message contains an identification code which is broadcast to all nodes of the network. The scheduler works out priorities and notifies the receiver as soon as the bus is available.

The identified node takes the message and performs the data transfer between the two computers. Having completed the data transfer the bus becomes free for the next request in the scheduler's queue.

- Advantage: Any computer can be accessed directly and messages can be sent in a relatively simple and fast way.
- Disadvantage: A scheduler is required to organize the traffic by assigning frequencies and priorities to each signal.

See also: Bus network

Examples of internal computer buses

Parallel

- ASUS Media Bus proprietary, used on some ASUS Socket 7 motherboards
- Computer Automated Measurement and Control (CAMAC) for instrumentation systems
- Extended ISA or EISA
- Industry Standard Architecture or ISA
- Low Pin Count or LPC
- MBus
- MicroChannel or MCA
- Multibus for industrial systems
- NuBus or IEEE 1196
- OPTi local bus used on early Intel 80486 motherboards.
- Conventional PCI
- Parallel ATA (aka Advanced Technology Attachment, ATA, PATA, IDE, EIDE, ATAPI, etc.) disk/tape peripheral attachment bus
- Q-Bus, a proprietary bus developed by Digital Equipment Corporation for their PDP and later VAX computers.
- S-100 bus or IEEE 696, used in the Altair and similar microcomputers
- SBus or IEEE 1496
- SS-50 Bus
- STEbus
- STD Bus (for STD-80 [8-bit] and STD32 [16-/32-bit]), FAQ (<http://www.controlled.com/std/faq.html>)
- Unibus, a proprietary bus developed by Digital Equipment Corporation for their PDP-11 and early VAX computers.
- VESA Local Bus or VLB or VL-bus
- VMEbus, the VERSAmodule Eurocard bus
- PC/104
- PC/104 Plus
- PC/104 Express
- PCI-104
- PCIE-104

Serial

- 1-Wire
- HyperTransport
- I²C
- PCI Express or PCIe
- Serial ATA (SATA)
- Serial Peripheral Interface Bus or SPI bus
- UNI/O
- SMBus

Self-repairable

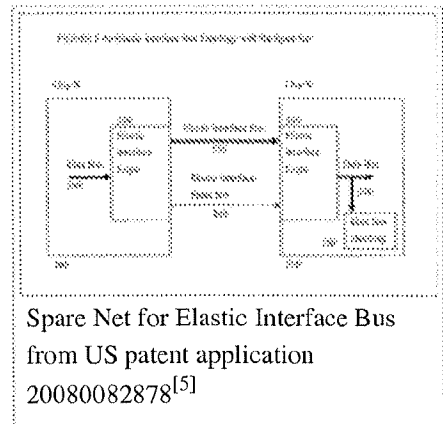
Self-repairable elastic interface buses have recently been invented by IBM. IBM has filed a patent application on these buses which is undergoing peer review on Peer-to-Patent. The public commentary

period closed on July 24, 2008.^[5] The IBM invention provides a spare net which the system switches to in the event that an alternate net doesn't function.

Examples of external computer buses

Parallel

- HIPPI High Performance Parallel Interface
- IEEE-488 (aka GPIB, General-Purpose Interface Bus, and HPIB, Hewlett-Packard Instrumentation Bus)
- PC Card, previously known as *PCMCIA*, much used in laptop computers and other portables, but fading with the introduction of USB and built-in network and modem connections



Serial

- USB Universal Serial Bus, used for a variety of external devices
- Controller area network ("CAN bus")
- EIA-485
- eSATA
- IEEE 1394 interface (FireWire)

Examples of internal/external computer buses

- Futurebus
- InfiniBand
- QuickRing
- Scalable Coherent Interface (SCI)
- SCSI Small Computer System Interface, disk/tape peripheral attachment bus
- Serial Attached SCSI (SAS) and other serial SCSI buses

See also

- Address bus
- Bus contention
- Control bus
- Front-side bus (FSB)
- Harvard architecture
- Network On Chip
- List of device bandwidths

References

- ↑ See the early Australian CSIRAC computer
- ↑ Linda Null; Julia Lobur (2006). *The essentials of computer organization and architecture* (<http://books.google.com/books?id=QGPHAI9GE-IC&pg=PA33>) (2nd ed.). Jones & Bartlett Learning.

- pp. 33,179–181. ISBN 9780763737696. <http://books.google.com/books?id=QGPHA19GE-IC&pg=PA33>.
- ^a C. Gordon Bell; R. Cady; H. McFarland; J. O'Laughlin; R. Noonan; W. Wulf (1970). "A New Architecture for Mini-Computers — The DEC PDP-11" (<http://research.microsoft.com/en-us/um/people/gbell/CGB%20Files/New%20Architecture%20PDP11%20SJCC%201970%20c.pdf>) . *Spring Joint Computer Conference*: 657–675. <http://research.microsoft.com/en-us/um/people/gbell/CGB%20Files/New%20Architecture%20PDP11%20SJCC%201970%20c.pdf>.
 - ^a Bray, Andrew C.; Dickens, Adrian C.; Holmes, Mark A. (1983). "28. The One Megahertz bus" (<http://www.nvg.org/bbc/doc/BBCAdvancedUserGuide-PDF.zip>) (zipped PDF). *The Advanced User Guide for the BBC Microcomputer*. Cambridge, UK: Cambridge Microcomputer Centre. pp. 442–443. ISBN 0-946827-00-1. <http://www.nvg.org/bbc/doc/BBCAdvancedUserGuide-PDF.zip>. Retrieved 2008-03-28.
 - ^a ^b Peer to Patent review page for "System and Method to Support Use of Bus Spare Wires in Connection Modules" (<http://www.peertopatent.org/patent/20080082878/overview>)

External links

- Chip Weems' Lecture 12: Buses (<http://www.cs.umass.edu/~weems/CmpSci635/635lecture12.html>)
- Computer hardware buses (<http://www.dmoz.org/Computers/Hardware/Buses/>) at the Open Directory Project
- Computer hardware buses and slots pinouts with brief descriptions (http://pinouts.ru/pin_Slots.shtml)

Retrieved from "http://en.wikipedia.org/wiki/Bus_(computing)"

Categories: Computer buses | Digital electronics | Motherboard

- This page was last modified on 25 May 2011 at 22:17.
 - Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. See Terms of Use for details.
- Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.