Design Issues in a Bare PC Web Server

Long He, Ramesh K. Karne, Alexander L. Wijesinha,
Sandeep Girumala, and Gholam H. Khaksari
Department of Computer & Information Sciences, Towson University, 7800 York Rd, MD 21252
lhe@towson.edu

Abstract

We present a bare PC Web server design that runs on any Intel 386 based PC without any hard disk and other supporting software. This server design is based on simplicity and minimal functionality. We identify design issues related to such server and provide our preliminary performance measurements in comparison with IIS and Apache servers. We have found that our Web server performs comparable to commercial servers and it can be enhanced to perform even better when it is optimized for certain design factors. This work also has motivated us to pursue further research in building bare PC Web servers that can be scalable to run large applications. In addition, as the Web server is self-contained with all necessary code, it inherently provides ubiquity on the network, which allows for moving the server dynamically on the network during emergency and catastrophic situations. Furthermore, bare PC software may also provide alternate ways of building secure systems for critical applications.

1. Introduction

Bare PC applications are not a new in concept. In fact, when computing started, all applications were designed for bare hardware, and were loaded and executed using toggle switches. Since then operating systems have provided elegant application services and transparency to hardware. Over the years, operating systems (OS) have grown in size, complexity, and hence are prone to errors and have become difficult to maintain. The unwarranted software and hardware releases have caused tremendous waste in cost, productivity, people resources, and environment damage [8]. The original purpose of OS and its useful intentions have faded away due to their inefficiency, complexity, and massive size. Some researchers suggest that operating systems should be exterminated [1] and their functionality should be moved to applications.

Web server designs are interleaved with operating systems as their functionality is driven by heavy system calls. Our bare-PC Web server, based on dispersed operating system computing (DOSC) [3] is the first step towards eliminating operating systems thus constructing applications that directly communicate with hardware. The DOSC concepts are based on dispersing the necessary OS functionalities to application programs, and making the application programmer aware of underlying hardware interfaces. Each application program in this environment will be denoted as an application object (AO) [6, 7], and each AO communicates with hardware using AO Interfaces [5], or API for the hardware. The AO interfaces do not change for a long period of time. When new interfaces such as "tasks" are added, the AO interfaces can be extended to add the new interfaces. Details of C++ API for direct communication to Intel 386-based processors are described in [4].

The bare PC Web server is designed as a single AO. This AO is self-contained, self-controlled, and self-executed (SSS) similar to autonomic world. The AO contains all the necessary codes including: boot, loader, memory management, task management, scheduling, and I/O. It is a single entity that can reside on a network, or a storage media. It carries its own device driver. An AO is an entity that is based on an application and inherits SSS properties.

2. Description

The bare PC DOSC Web server (DS) is currently running at http://dosc.towson.edu. It runs on a Pentium II processor with 350MHZ, and 128MB of memory. It contains 3COM 509B device driver running on bare PC. This server is designed for simplicity and minimal functionality.

The DS is designed to run the HTTP 1.1 protocol. The AO for DS requires HTTP, TCP, IP, ARP, and Ethernet networking protocols and their implementation stack. The dataflow for DS is shown in Figure 1. All tasks are shown in circles, processing blocks are shown in rectangles, and

locks indicate synchronization points. Network Interface Card (NIC) receives packets, and stores them in its local FIFO (first-in first-out). The "Ethernet Receive" task periodically checks (polling) for a received message and if a message arrives, it will insert it in an Ethernet circular buffer (ECB). Task scheduling is done every time a hardware timer causes an interrupt. The scheduling of tasks is done using a round robin algorithm. Once a packet is in ECB, a "Poll Task" will read this packet and process it appropriately as a single string until it is complete. An HTTP process will take this request from HTTP table and processes it until all data packets are sent to the client. As the "Poll Task" processes packets and put them into TCP and HTTP lists, HTTP processes will process them concurrently to serve client requests. We have separate tasks for serving HTTP requests after the TCP connections are established. TCP related messages are processed as part of a single task in "Poll Task". Also, resources such as HTTP list, TCP list, and ECB have contention with other tasks, thus they use locks for synchronization.

HTTP files are stored on a different network node. The server loads these files into memory using an ad-hoc (our own) ftp protocol. The DS hosts PI's Web site with 673 files on 40MB file storage. The Web clients can access our Web server from standard Web browsers. The size of the executable files for DS is 77,824 bytes, and number of lines of C++ code is 6587. The number of lines of ASM code is not measured and included in the above count.

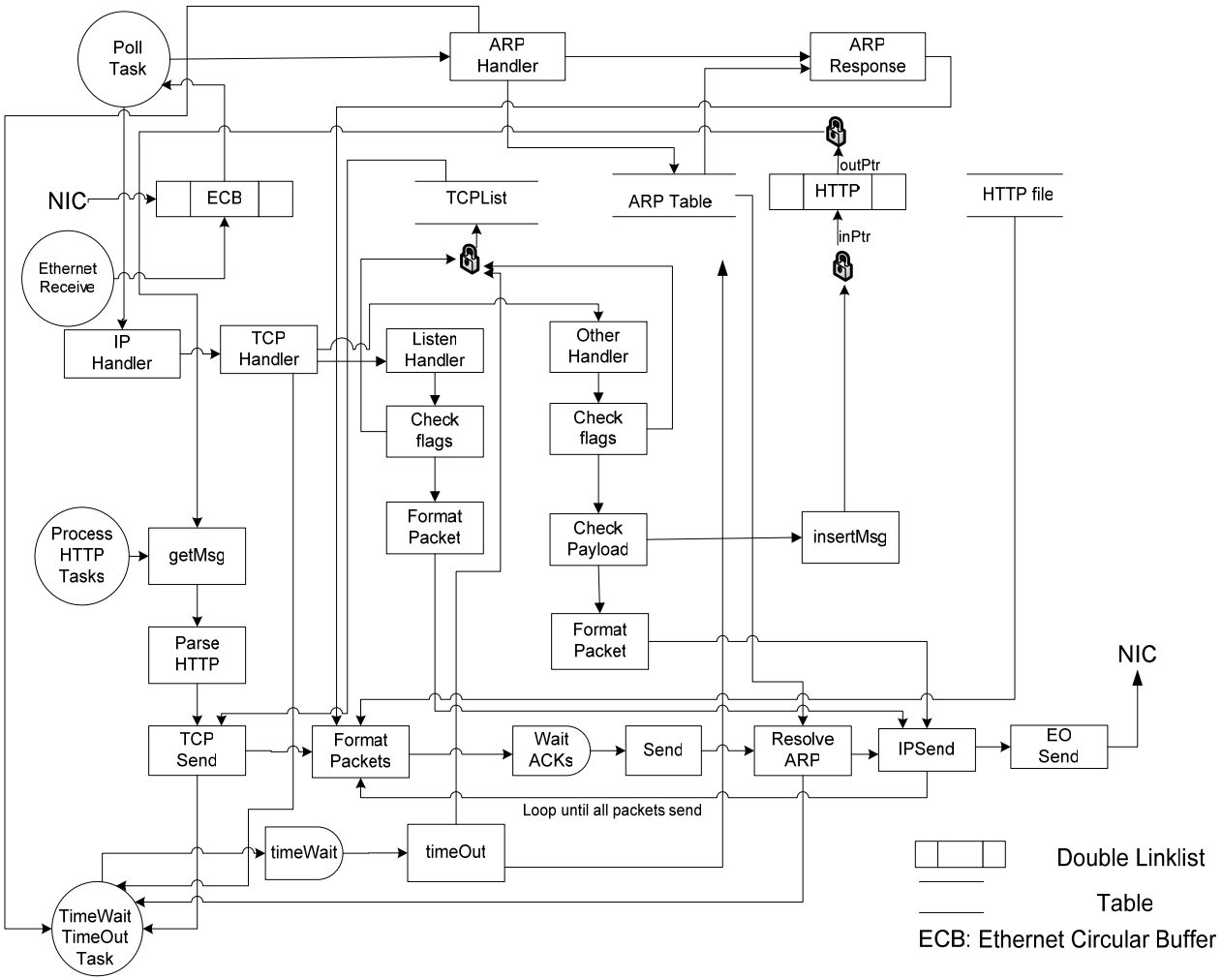

Figure 1

3. Design Issues

It is not possible to go through this vast literature in this paper; however, a comprehensive bibliography of this research is posted at author's web site http://www.towson.edu/~karne. As per our knowledge, there is no one looking at building servers on a bare machine environment. As Web servers are application specific entities, they are excellent candidates for DOSC implementation.

Current research directions in design issues on DS Web servers are mostly focus on operating system related improvements and network related optimizations. In this Web server architecture, there is no hard disk, paging, virtual memory, cache, and graphical user interfaces. We identified the following issues related to bare machine Web server and provide our research findings.

3.1. File Storage

Usually a large number of HTML files are stored in an HTTP Web server. Our DOSC Web server runs on a bare PC and it are not convenient to carry all the HTTP files in the AO. HTTP Files are stored on the network; DOSC Web server fetches the files during initialization. Currently, all files are fetched into memory as there is plenty of memory to support file storage. That character helps server's mobility when moved from one location to another on the Internet.

3.2. Memory

As simplicity is the main objective of DS in design, we avoid OS, paging, and virtual memory. We can fairly assume that there is full address space of 4GB memory in the system as our DOSC Web server only needs 128MB of memory for its operation and nearly all the memory is available for applications. For applications needing more memory, several bare PCs can be connected to achieve higher throughput. This scalable parallel bare PC architecture is currently under investigation.

3.3 Ethernet Driver

When an AO runs on a bare PC, it carries all the programs and controls necessary to execute on a bare PC including device driver interfaces. We have designed a device driver for 3COM509B Ethernet card that works with our server. This driver provides API calls in C++ to communicate directly to Network Interface Card (NIC) and has been developed as a C++ class object. This class can be instantiated in the program's required calls such as init, read, and can be written directly from other part of application. We select this particular NIC as its programming reference manual is available on the Web. We also have developed a new device driver for 3COM905CX NIC by collaborating with 3COM, which is more recent and has enhanced functionality. Our experiences with both drivers for NIC indicate that 3COM could have designed an extensible NIC to support downward compatibility. Each NIC and each I/O card has its own interface thus causing a myriad of drivers and interfaces in a PC environment. We also found that the C++ API designed for the above two drivers are very similar and they can be easily extended for the next generation of NIC devices.

3.4 Network Stack

Only the necessary functionalities required for Web server application are implemented in the network stack. The stack protocols including Ethernet, ARP, IP, UDP, TCP, and HTTP are designed and implemented as API calls. Each protocol is a C++ class object and can be instantiated anywhere in a C++ program. All classes can directly communicate with Ethernet class, that provides a robust foundation for testing new network architectures and protocols.

3.5 Task Scheduling

The DS tasks are provided as C++ API calls. Each task is a member function in a task object. We can design as many tasks as needed by simply adding new member functions in the task class, and control the task scheduling. Currently a simple round robin algorithm is used to schedule tasks. Tasks are scheduled when a hardware timer interrupt comes at a periodic rate of 55 milliseconds. We can also vary this clock rate by dividing the rate in multiples of 2.

3.6 Concurrency

We use locking mechanisms to implement synchronization. However there are many design options to exploit concurrency to achieve maximum throughput and minimum response time.

3.7 Response Times

Interrupt service routine (ISR) implementation for receiving and sending messages to NIC will improve response time. However, this approach needs to be

studied in combination of ISRs and Tasks. ISRs make the system more complex and hard to debug in a bare PC environment.

3.8 Throughput

In our driver design, we wait and check status of the transmitted message to make sure the packet be sent successfully. Using ISRs to process the status will improve the performance. However, that make the system more complex and ISRs will interfere with task processing. The HTML file size will also play a major role in the throughput outcome, which needs to be studied further.

3.9 Robustness

The DS design is very robust. Task scheduling, sending and receiving messages, concurrency and synchronization, network protocols, and API to bare machine are very flexible and easy to maintain. It also provides more robustness for designing new applications.

3.10 Pervasiveness

The DOSC concept is based on bare machine API and single programming language such as C++. If we can use Intel 386-based CPU architecture in pervasive devices, then the same Email client, POP server, and Web server code can run on any device. As the code is small, it is quite possible to make it fit in any devices. If we make pervasiveness in application as a requirement and use same architecture for a variety of devices, we will provide a pervasive solution.

3.11 Mobility

DS AO is a small and complete entity, which can execute on a bare PC. The AO can easily be made mobile under emergency or catastrophic situations. When an AO is mobile, it detects the situation, communicates with the remote bare PC, and moves to a remote node. We have all the mechanisms designed for the mobility of AO, and its implementation is in progress.

4. Measurements

Our testing platform for making measurements includes 3 similar PCs on a LAN each with 350MHZ Pentium II processor and 128MB memory. PC-1 is loaded with the DS server only. PC-2 is loaded with Windows 2000 Server OS, the Microsoft IIS server and the Apache Tomcat server, and Web stress tool "http_load" is [2] installed on PC-3 with Windows 2000. We compare DS with IIS and Apache servers, where each server hosts a Web site with 673 files including: HTML, PDF, and GIF files, totaling 40MB of storage. When tests are run, "http_load" uses a single file to access and measure the performance. At the end of a run, the "http_load" provides measured data, which is directly used in the following charts.

We first compare the throughput and first response time of DS with IIS and Apache server based on different fetches. We make measurements on file size 3593 bytes for 10 minutes. We select DS running on 55/48 ms and 55/16 ms time slot. The measurement result is shown in figure 2 and figure 3. The chart figure 2 shows DS server has the same throughput performance as IIS and Apache, figure 3 shows that the first response time of DS in 55/48 ms time slot is better than Apache and IIS.

We also use three files to make the measurements on DS, IIS and Apache server. The results are shown on figure 4. The file sizes are 3593 bytes, 9652 bytes and 18007 bytes. The measurements are based on 10 fetches / sec for 10 minutes. DS server runs with time slot of 55/48 ms. The measurement result shows that DS server has roughly the same throughput performance as IIS and Apache server with the change of file sizes.

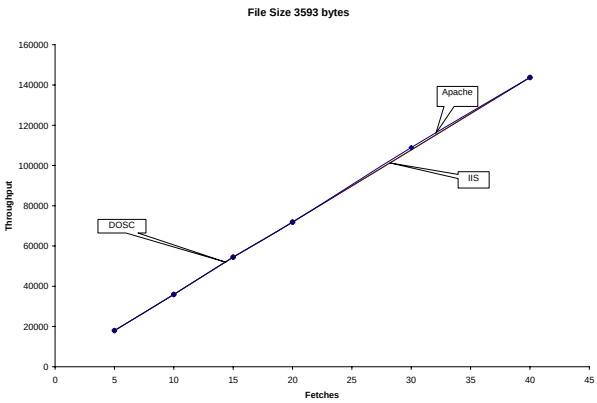

Figure 2

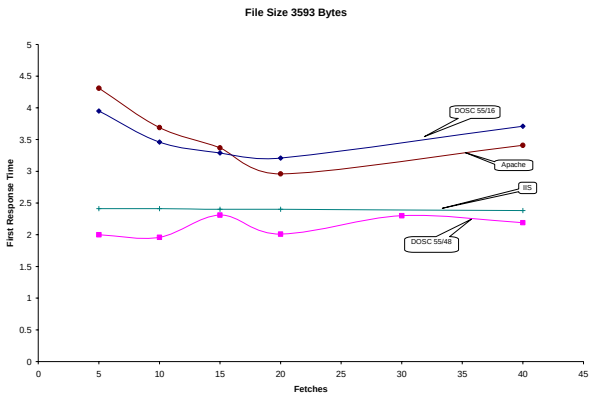

Figure 3

In DS ,the time slot can be changed by 55/1, 55/2, 55/4, 55/8, 55/10, 55/16, 55/20, 55/24, 55/28, 55/32, 55/36, 55/40, 55/44, 55/48, 55/56, 55/64 and 55/72 ms, we make measurement based on different time slot, the measurement result shows that the shorter the time slot, the better performance for DS. The measurement results are shown in figure 5. That measurement is based on one fetch on file size 3593 bytes for 10 minutes. It shows that with the time slots less than 55/20 ms, DS server's first response time is better than IIS and Apache server.

5. Further Research

The measurements on this DOSC server show promising results and provide insight into further research to build a Web server that runs on latest PCs with PCI (Peripheral Component Interface) bus. As NIC 3COM905CX provides better performance and different memory interfaces, we have designed a new device driver for 3COM905CX NIC. Currently, we integrate this driver with Web server AO. We will also conduct measurements with new changes made to AO This paper presented a novel concept for Web server design, which runs on a bare PC. A Web server description based on dispersed operating system concept is presented including its design details and interfaces. This Web server directly communicates to hardware through application programming interfaces for timer, CPU, tasks, memory, network card interfaces, and so on. When these interfaces become a standard, it has a potential of providing bare machine architecture for pervasive devices. Design issues related to building a bare PC Web server are identified and a brief description of each issue is presented. Performance measurements of this server and comparison with IIS and Apache server are presented. with the design issues studied in this paper. Our long range plans for this Web server is to study the scalability in a parallel environment, and assess this system for reliability and security.

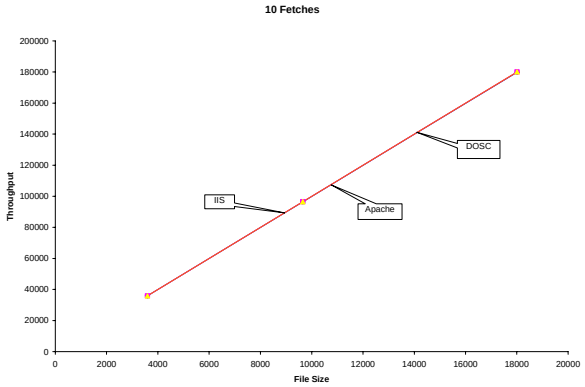

Figure 4

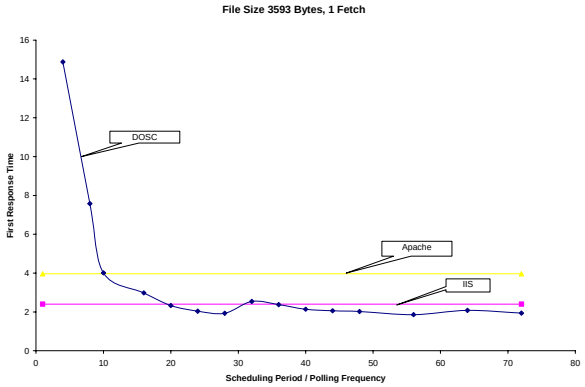

Figure 5

6. Conclusions

Further research in bare PC Web server research is outlined.

References

[1] Engler, D. R., M.F. Kaashoek, "Exterminate all operating system abstractions," FifthWorkshop on Hot Topics in Operating Systems, p. 78, 1995.

[2] http://www.acme.com/software/http_load.

[3] Karne, R.K., Jaganathan, K. V., Ahmed, T. "DOSC: Dispersed Operating System Computing", OOPSLA '05, 20th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, Onward Track, October 2005, Sandiego, CA, p55-p61.

[4] Karne, R.K., Jaganathan, K. V., Ahmed, T. "How to run C++ Applications on a bare PC", SNPD 2005, Proceedings of SNPD 2005, 6th ACIS International Conference, May 2005, p50-p55.

[5] Karne, R.K., Gattu, R., Dandu, R., and Zhang, Z., "Application-oriented Object Architecture: Concepts and Approach," 26th Annual International Computer Software and Applications Conference, IASTED International Conference, Tsukuba, Japan, NPDPA 2002, October 2002.

[6] Karne, R.K. "Application-oriented Object Architecture: A Revolutionary Approach," 6th International Conference, HPC Asia 2002, December 2002.

[7] Karne, R.K., "Object-oriented Computer Architectures for New Generation of Applications," Computer Architecture News, December 1995, Vol. 23, No. 5, pp. 8-19.

[8] Whitley, J., "A Study of Computer Obsolescence and Its Impact," M.S Thesis, Department of Computer and Information Sciences, Towson University, Towson, MD 21252, December 2001.