Available online at www.sciencedirect.com
SCIENCE DIRECT
Information and Software Technology 48 (2006) 471–483
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

An event-driven framework for inter-user communication applications

Chien-Chih Hsu, I.-Chen Wu*

Department of Computer Science and Information Engineering, National Chiao Tung University, 1001 Ta Hsueh Road, Hsinchu 300, Taiwan, ROC

Received 16 June 2004; revised 20 May 2005; accepted 24 May 2005
Available online 10 August 2005

Abstract

This paper presents an event-driven framework for inter-user communication applications, such as Internet gaming or chatting, that require frequent communication among users. This paper addresses two major blocking problems for event-driven programming for inter-user communication applications, namely output blocking and request blocking. For the former, an output buffering mechanism is presented to solve this problem. For the latter, a service requesting mechanism with helper processes is presented to solve this problem. The above two mechanisms are incorporated into the framework presented in this paper to facilitate application development. In practice, this framework has been applied to online game development.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Inter-user communication applications; Event-driven programming; Concurrent programming; Framework; Threads

1. Introduction

With the rapid growth of the Internet, applications involving real-time communication among clients have become increasingly important. These applications include chat rooms such as Yahoo! Chat [44] and EFnet chat network [11], Internet games such as Yahoo! Games [45], Warcraft III [4], and Counter-strike [41], and present and instant messaging systems such as ICQ [17] and MSN Messenger [23]. Consider an example of chat room or game system. One user types a message and others then can read that message in real time. Since these applications involve inter-user communication, this paper calls them *inter-user communication applications*.

For inter-user communication applications, servers are often used to handle inter-user communication. For example, game servers receive player events (or messages) and then respond (or pass messages) to other players. For inter-user communication applications, server developers generally must consider the following criteria.

1. *Minimize the client response time.* If the response time is unexpectedly long, interactions may not evolve as expected or users may run out of patience.
2. *Ensure high server stability.* Server crashes cause all clients connected to that server to become disconnected.
3. *Support as many clients concurrently as possible.* For example, support thousands of players on a single server.

The first criterion is essential for server programming in inter-user communication applications. To respond to users as rapidly as possible, servers usually hold connections to clients. Servers thus must handle client messages (or events) from all connections concurrently and server developers must handle concurrent events carefully.

Two main programming models exist for concurrent event handling, namely threading and event-driven programming. Threading is a general-purpose technique for managing concurrency. The advantages of threading compared to event-driven programming include: (a) support of context switching among threads, and (b) support of scalable performance on multiple CPUs.

However, some developers and researchers [27,32] have also observed that threading has some drawbacks compared to event-driven programming. Note that Ousterhout [27] described the following drawbacks:

* Corresponding author. Tel.: +886 3 573 1855; fax: +886 3 573 3777.
E-mail addresses: jjshie@csie.nctu.edu.tw (C.-C. Hsu), icwu@csie.nctu.edu.tw (I.-C. Wu).

0950-5849/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2005.05.005

1. *Difficult to program.* Threads generally require synchronous mechanisms (e.g. locks) to access shared data safely. However, incorrect locking may cause deadlocks, making independent module design difficult. Besides, another problem that also increases programming difficulty is that several standard libraries are not thread-safe [24].
2. *Hard to debug*. For threading, it is difficult for developers to debug the code due to data and timing dependencies. Besides, another problem that also increases debugging difficulty is that thread stack sizes are normally limited [22,24], causing processes crash when stacks overflow. In contrast, in event-driven programming, the lack of context switching among event handlers makes it quite easy to debug the code by recording and then replaying the sequence of events.
3. *Difficult to achieve good performance.* Coarse-grain locking yields low concurrency, while fine-grain locking tends to increase lock operations and thus reduce performance.

Since inter-user communication applications are often used to facilitate heavy inter-user communication among numerous clients (say, thousands of players in a game system), it makes the above drawbacks even worse. The first two drawbacks imply that it is hard for threading to satisfy the second criterion (above) of the inter-user communication applications; and the third drawback indicates that it is hard for threading to satisfy the third criterion. Thus, for the application developers who are more concerned with the second and third criteria and less concerned with the two threading advantages (described above), the event-driven programming model becomes attractive. Hence, this paper is motivated to study and design an event-driven framework for inter-user communication applications. Note that a framework [13,33] is defined as a set of collaborative classes that enable developers to reuse the architecture and implementation of a generic program for a set of domain specific applications.

Our framework is based on event-driven programming (rather than threading) for the following reason. In inter-user communication applications, the above three drawbacks of threading (or the second and third criteria) are important as described above, while the two drawbacks of event-driven programming are less important because they can be ignored or alternatively can be solved in this paper. First, regarding the two drawbacks of event-driven programming, this paper ignores the one, namely not supporting scalable performance on multiple CPUs, because for most inter-user communication applications servers can be separated into several processes to achieve scalable performance. In the case of casual games, such as Chess and Bridge, servers can naturally be separated into several processes, e.g. one for each game. Even for most massive multiplayer online games (MMOGs), such as Ultima Online [12], the server system can use several processes each dealing with a single game scene. Second, this paper focuses on overcoming the other drawback of event-driven programming: the need to pay attention to the blocking problem in event handling.

This paper addresses two major blocking problems and presents solutions or guidelines. The two blocking problems are described below.

1. *Output blocking:* This problem occurs on sending messages to clients with corresponding full kernel buffers. The buffer generally becomes full when network traffic is jammed. This problem frequently is neglected at the start of server development.
2. *Request blocking:* This problem occurs when a server waits for responses after sending requests to other servers. For example, when a game server attempts to read several game records from a remote database server.

This paper presents solutions for the above two blocking problems. An output buffering mechanism is presented to solve the output blocking problem, while a service requesting mechanism is presented to solve the request blocking problem. Meanwhile, for the second problem, several system and library calls that may cause the problem are also identified. Both mechanisms are incorporated into the event-driven framework presented in this paper.

Practically, the event-driven framework has been used in the CYC game system [39] that provides players with casual games, such as Chess, Bridge, Mahjong, etc. Currently, the CYC game system has supported up to 10,000 concurrent players.

The rest of this paper is organized as follows. Section 2 reviews the event-driven programming model. Section 3 describes the output blocking problem and presents solutions. Section 4 then describes the request blocking problem and presents solutions. Section 5 presents our experiments by applying our framework to the CYC game system and some performance analysis. Finally, Section 6 summarizes our work.

2. Event-driven programming

This section reviews the event-driven programming model. In this model, applications wait for specific events and dispatch occurring events to appropriate handlers for processing. In networked applications, event-driven based servers generally handle both input and output events. Input events occur when sockets are ready to read, while output events occur when sockets are ready to write.

Most event-driven based servers in the Unix environment use the `select` system call [10,18,34,36] to demultiplex input/output events. The `select`-based event-driven model has been induced as the Reactor design patterns in [31,32]. In this pattern, the core component named

Download English Version:

https://daneshyari.com/en/article/550789

Download Persian Version:

https://daneshyari.com/article/550789

Daneshyari.com