EXTENDING THE CHORUS MICRO-KERNEL TO SUPPORT CONTINUOUS MEDIA APPLICATIONS

Geoff Coulson, Gordon S. Blair, Philippe Robin and Doug Shepherd

Department of Computing,
Lancaster University,
Lancaster LA1 4YR, U.K.
e.mail: mpg@comp.lancs.ac.uk

Abstract. Currently, popular operating systems are unable to support the end-to-end real-time requirements of distributed continuous media. Furthermore, the integration of continuous media communications software into such systems poses significant challenges. This paper describes a design for distributed multimedia support in the Chorus micro-kernel operating system environment which provides the necessary soft real-time support while simultaneously running conventional applications. Our approach is to extend existing Chorus abstractions to include QoS configurability, connection oriented communications and real-time threads. The design uses the following key concepts: the notion of a flow to represent QoS controlled communication between two application threads, a close integration of communications and thread scheduling and the use of a split level scheduling architecture with kernel and user level threads. The paper shows how our design qualitatively improves performance over existing micro-kernel facilities by reducing the number of protection domain crossings and context switches incurred.

1 Introduction

A considerable amount of research has already been carried out in communications support for continuous media over high speed networks. However, much less work has been done in the area of general purpose operating system support for continuous media. Typically, end-system implementations have either been embedded in non real-time operating systems such as UNIX and suffered from poor performance, or have been implemented in specialised hardware/software environments unable to support general purpose applications.

The SUMO Project at Lancaster [1] is addressing this deficiency in the state of the art by extending a commercial micro-kernel (i.e. Chorus [2]) to support continuous media applications alongside standard UNIX applications (Chorus already supports UNIX applications through the provision of a UNIX subsystem). Chorus is a useful starting point for continuous media support as it includes a number of desirable real-time features. However, in common with other micro-kernels, it fails to adequately support continuous media in a number of key areas. First, communication in Chorus is message based whereas continuous media requires stream-oriented communications. Second, Chorus offers no quality of service (QoS) control over communications and only coarse grained relative priority based scheduling to control the QoS of processing activities. Finally, Chorus does not offer end-to-end real-time support spanning both the communications and scheduling components.
To overcome these deficiencies we introduce the concept of a ‘flow’. A flow characterises the production, transmission and eventual consumption of a single media stream as an integrated activity governed by a single statement of QoS. Realisation of the flow concept demands tight integration between communications, thread scheduling and device management components of the operating system. It also requires careful optimisation of control and data transfer paths within the system.

The rest of this paper is structured as follows. Section 2 provides the background on the Chorus micro-kernel necessary to understand the rest of the paper. Section 3 describes the programming interface to our multimedia facilities and section 4 presents some examples of its use. Following this, section 5 discusses the implementation of the multimedia support, concentrating on communications and scheduling issues. The examples of section 4 are also revisited to illustrate the qualitative efficiency gains produced by our design over standard micro-kernel facilities. Finally, section 7 discusses related work in the field and section 8 presents our conclusions and indicates our plans for future work.

2 Background on Chorus

Chorus, conceived at INRIA, France, is a micro-kernel based operating system which supports the implementation of conventional operating system environments through the provision of ‘sub-systems’ (for example a sub-system is available for UNIX SVR4). The micro-kernel is implemented using modern techniques such as multithreaded address spaces and inter-process communication with copy-on-write semantics. The basic Chorus abstractions are actors, threads and ports, all of which are named by globally unique and globally accessible unique identifiers. Actors are address spaces and containers of resources which may exist in either user or system space. Threads are units of execution which run code in the context of an actor. By default, they are scheduled according to either a pre-emptive priority based scheme or round robin timeslicing. Ports are message queues used to hold incoming and outgoing messages. They can be aggregated into port groups to support multicast messaging and may be migrated between actors. Inter-process communication is datagram based and supports both request/reply messages (via the ipcCall() and ipcReply() system calls and one shot messages (via ipcSend() and ipcReceive()).

Chorus has several desirable real-time features and has been widely used for embedded real-time applications. Real-time features include pre-emptive scheduling, page locking, system call timeouts, and efficient interrupt handling. Chorus also incorporates a framework, called scheduling classes, which allows system implementers to add new scheduling policy modules to the system. These modules are upcalled each time a scheduling event occurs. Modules impose their scheduling decisions by manipulating a global table of thread priorities.

Unfortunately, Chorus’ real-time support is not sufficient for the requirements of distributed multimedia applications, principally because there is no support for QoS control and resource reservation:-

- although it is possible to specify thread scheduling constraints relative to other threads, absolute statements of requirement for individual threads cannot be made,
- the exclusive use of connectionless communications makes it impossible to pre-specify communications resource allocation.

In addition, Chorus suffers from a lack of communications/scheduling integration. This means that there is no way to provide timely scheduling in concert