A C++ Library for Computer Music Programming on Real-Time Mach

Andrew Choi
Department of Computer Science
University of Hong Kong
Pokfulam Road, Hong Kong
choi@cs.kau.hk

Abstract

The combination of the OSF Mach microkernel, the McLinux Unix server, and the set of GNU software tools represents a free yet powerful programming environment for the development of real-time interactive computer music software on desktop PCs. A C++ library is being built in this programming environment to support real-time audio and MIDI input and output, and to provide buffer and event management. The OSF Mach microkernel provides all real-time scheduling and synchronization support.

1 Introduction

As the speed and power of general-purpose microprocessors increase, the development of computer music software on systems built on them becomes more feasible and appealing. Compared to ones with DSP processors, these systems are attractive because of the availability of programming environments and software tools that are familiar to the programmer, and not requiring the understanding of additional details of the DSP processors and architecture.

Desktop multimedia and music systems built around personal computers are certainly capable of processing MIDI and digital audio data. In applications for which this data can be pre-computed, timing accuracy is achieved by placing time-critical code in device drivers and interrupt handlers. Despite this, programmers of real-time interactive computer music software must work within or around the limitations of the operating systems of these personal computers. Programming libraries for interactive music systems such as W [Davey and Ruby 1995], MOOD [Anderson and Bilmes 1991], and MidiShare [O'Carley and Legay 1989] provide their own schedulers. However, the response time of real-time software constructed with these libraries can still be no better than the latency in the underlying operating system, which is typically not designed to support real-time processing.

The FTS real-time executive runs on a number of specialized machines as well as Unix workstations [Dechelle 1995]. It provides a simple scheduling policy that periodically executes the tasks on a static list. It will be interesting to study whether the real-time performance of FTS is hindered by being run on the Unix operating system. [Dannenberg et al. 1996] discusses the features in an operating system that are important for supporting real-time interactive computer music software and evaluate various operating systems for that purpose. [Pope 1995] summarizes the evolution of computer music workstations and software environments.

2 OSF Mach

Earlier this year, the Open Software Foundation made available their version of the Mach microkernel and the McLinux Unix server for Intel x86 machines. Versions of OSF Mach and McLinux for Apple Power Macs have also been announced for release later this year. OSF Mach is based on Carnegie Mellon University's Mach 3.0 [Rau et al. 1989] but has incorporated many improvements [Wells 1994]. One of these is real-time support, using results from their real-time project. New features for real-time support include:

- a preemptive microkernel [Swartzendruber 1993],
- coexistence of multiple scheduling policies and control over the scheduling policies and priorities of threads [Hayross et al. 1994],
- synchronization primitives with real-time support [CaraDonna et al. 1994], and
- various services such as clocks, alarms, wired memory, and a real-time threads library.

Making the microkernel preemptible is the most important enhancement to Mach for real-time programming. Although other versions of Mach all support the other features listed above to various extents, they have much higher interrupt latencies than OSF Mach and are therefore not as suitable for supporting real-time applications.

http://www.gr.osf.org/mclinux/

1 The code names RT1, RT2, etc. are used in OSF literature to refer to releases of these real-time microkernels. This is not to be confused with the RT Mach project at CMU.
With a non-preemptive microkernel, if a high-priority thread becomes runnable when a low-priority thread is executing a kernel call, the former will not get control of the CPU until the latter completes the kernel call. As a result, high-priority threads may be made to wait for as long as the longest time required by a kernel call.

[Dannenberg et al. 1993] and [Dannenberg et al. 1994] report scheduling delays of as much as 149.6 ms and 288.6 ms for the Mach 3.0 microkernel and the RC-Mach microkernel [Tokuda et al. 1990], respectively, when a mixture of non-real-time and real-time tasks are run concurrently. Such delays in OSF Mach are not than 1 ms [Weis 1994]. Our experiments with MIDI message input and output confirm their claim.

3 The MkMusic Library

MkMusic is a small library that provides low-level support for writing real-time MIDI and DSP applications under OSF Mach. It implements a number of abstract data types so that programmers can avoid dealing directly with Mach objects and primitives. The emphasis of the library is on low-level support such as threads, programming, communication, and synchronization. High-level constructs such as those provided by the virtual time and music layers of MOOD [Anderson and Bilmes 1991] may be built on it.

A MkMusic application runs as a single multitreaded Mach task. Each thread is in fact an rthread (real-time thread) supported by the OSF Rthreads library [Cara-Donna 1994]. All the threads of a task share their single address space. Concurrent access to data used by the application is controlled by semaphores, but most applications need not manipulate semaphores directly because of the use of high-level abstract data types provided by MkMusic.

Using the Rthreads library, an application can create and destroy threads, and set and change their scheduling policies and priorities. Setting a high priority for the time-critical threads of the application ensures that their timing accuracy is not affected by non-time-critical threads running concurrently in the system at low priorities. Since the OSF Rthreads library already provides a simple interface, MkMusic applications use it directly to manage threads.

4 Buffer Queues and Event Queues

MkMusic provides two types of queues for transmission of data among threads. Message queues serve as first-in-first-out (FIFO) buffers of messages. Two types of message queues are defined: one for MIDI messages and one for blocks of audio samples. A size is specified when a queue is created and determines the maximum number of messages or blocks that can be stored in it at any one time. Access to a message queue follows the simple producer-consumer model (see e.g. [Mackay et al. 1987]). A write operation proceeds immediately if the queue is not empty; in the case, it all the slots are occupied, the issuing thread is blocked until one becomes empty. A read operation proceeds immediately if one or more messages are available in the queue; otherwise the issuing thread is blocked until one becomes available. Multiple producers and consumers are allowed to access the same queue concurrently and are protected from each other by semaphores associated with the queue. Queues of size two are equivalent to double buffers, which are commonly used in interrupt handlers for inpipe and output devices.

Event queues are priority queues containing event messages with the times of their delivery. Events in one event queue are prioritized in a "first in, first out" or "first deadline" order. MkMusic supports two types of event queues: one for MIDI events and one for events for delivery of sample blocks. The write operation to an event queue has the same semantics as for message queues: it may need to wait for an empty slot to become available in the queue. Writing an event whose time has already passed results in an error. The thread reading an event queue will be blocked until the time of the earliest event in the queue. When that time arrives, the message is delivered and the blocked thread becomes runnable. This design is similar to the "stream model" used in [Letz et al. 1995].

In the current implementation of the library, events must be added in increasing time order. A future implementation should remove this restriction.

5 Predefined Queues and I/O Threads

MkMusic provides four predefined queues for handling MIDI and audio I/O: _m_in, _m_out, _a_in, and _a_out. The two input queues are defined to be message queues, since input messages are made available in the application as soon as they arrive. The two output queues are defined to be event queues since output messages are typically scheduled to occur at later times. A special specifier _auto allows output events to be sent immediately. At initialization time, MkMusic initializes the MIDI and audio devices and automatically creates four rthreads to receive input from and to send output to these devices. These rthreads run under the FIFO scheduling policy and at the highest priority in OSF Mach so that I/O is handled as promptly as possible. The C++ statements to access the predefined queues are:

```
_m_in >> mm;
_m_out << am;
_a_in >> am;
```
where \( m, n, a, \) and \( a_0 \) are a MIDI message, a
MIDI event, an audio sample block message, and an
audio sample block event, respectively.

6 Sample MkMusic Applications

A graphical notation is used below to illustrate the
different applications that can be built using
rhythms, buffer queues, and event queues. At
initialization time, an application may create user queues by calling
the MkMusic library and creates user threads by calling
the Rhythms library. When they are executed,
rhythms read from and write to queues in a manner
indicated by the arrows in the diagrams. Figure 1
shows the elements of the graphical notation: a pre-
defined rhythm, a user rhythm, a message queue, and
an event queue.

![Figure 1. Elements of Graphical Notation](image)

The block diagram of a MIDI recorder application is
shown in figure 2. The only user rhythm in this
application associates a time-stamp with each received
message and appends this to a MIDI file.

![Figure 2. A MIDI recorder application](image)

Figure 3 shows the block diagram of a MIDI player
application. A user rhythm reads messages from a
MIDI file, converts the time-stamp associated with
each message to absolute time, and sends each mes-
sage as an event to the \( m_{\text{out}} \) event queue. The
fixed size of \( m_{\text{out}} \) serves to regulate the flow of
data from the MIDI file to the MIDI output. The
user rhythm attempts to keep the event queue filled
so that messages queue up in it ready for output.

![Figure 3. A MIDI player application](image)

The block diagram of an interactive MIDI application,
such as one for experimenting with machine improvi-
sation, is shown in figure 4. One or more rhythms
can read from the \( m_{\text{in}} \) message queue and respond
by generating a number of MIDI notes, or by starting
other rhythms that may perform a sequence of
actions. All of the rhythms may write MIDI notes to
the \( m_{\text{out}} \) event queues to schedule notes to be played
at a later time.

![Figure 4. A pitch-to-MIDI system.](image)

A pitch-to-MIDI system (figure 5) demonstrates how
an application may process audio input data. The
predefined rhythm started by MkMusic for the sound
audio input packs the input samples into fixed-size
blocks and places them in the \( m_{\text{in}} \) message queue.
These sample blocks become available to be read
from the queue at regular time intervals, since the
predefined rhythm runs at the highest priority in OSF
Mach. A user rhythm reads the signal blocks and
performs fundamental frequency estimation and note
detection on them. When a note is detected, the user
rhythm sends a MIDI event to the \( m_{\text{out}} \) event queue
with the time specifier \( \text{TIME} \) so that the note is
played immediately.

![Figure 5. A pitch-to-MIDI system.](image)

A software synthesizer represents an application that
requires direct access to the synchronizations prim-
itives in OSF Mach, in addition to using the message
and event queues in MkMusic. One possible organi-
sation of a software synthesizer is as shown in figure 6.
A number of user rhythms handle the generation of
notes: each request to play a note is assigned an
rhythm, which is responsible for computing the sam-
ples of the note for its entire duration. There are as
many rhythms as the number of polyphonic notes
supported by the synthesizer, which is limited by the
speed of the processor. An rhythm corresponding to
an active note executes periodically to fill a buffer
with a block of samples of the note. An output
rhythm accesses the buffers corresponding to all active
notes, sums them, and schedules the resulting sample
block for output by sending it to the \( m_{\text{out}} \) event
queue. All user rhythms are scheduled for periodic
execution with a period equal to the time required to
play each sample block. Additional clock and syn-
chronization services in Mach are accessed to cause
7 Current State of the Implementation

Since the Intel x86 versions of OSF Mach and MkMusic on which McKmusic is being developed has only recently been released, a number of important components are still missing from them. Most notably the microkernel has no device drivers for sound cards and MIDI cards. Work is currently underway to port drivers for these devices from Linux. Another omission which also greatly hinders development is that the gdb debugger can not yet run on OSF Mach.

The current implementation of McKmusic accesses MIDI devices through one of the serial communication ports of the PC. The implementation of message queues and event queues have been fully tested with MIDI messages and events, and the library is able to handle these with high timing accuracy - the maximum delay for MIDI input and output is never more than 0.5 ms, even when other, non-time-critical tasks that perform graphics output and disk accesses are running concurrently on the system with lower priorities. The testing for audio sample block messages and events will commence when the device driver for a sound card becomes operational.

8 Summary

OSF Mach is highly suitable as an operating system platform for the implementation of interactive real-time music software due to its excellent real-time support. Other features of the Mach microkernel yet to be exploited are its efficient and flexible interprocess communication implementation and its multiprocessor support. These features open up a variety of possibilities of being able to easily expand a computer music system as the computational demands of applications increase.

References
