Real-Time Media Player on ChronOS

Final Project Report

ECE 5550G - (Advanced) Real-Time Systems

	Name	Email-id
1.	Parth Shroff	parths11@vt.edu
2.	Anamika Sen	anamikas@vt.edu

About the Project:

This project consists of rewriting an open source multimedia application in order to exploit real-time scheduling policies implemented in ChronOS. We have chosen **Mplayer 1.2.1** for this project. Real-time scheduling policies are used greatly by the Multimedia applications to deliver the content at a constant rate to the user. However, because multimedia application don't usually rely on a real-time scheduler many times the audio or video is not delivered as smoothly as expected. The actual performance degradation depends on whether there are other currently running applications in the system. In order to make multimedia applications run smoothly this project proposes to rewrite an open source multimedia player to use ChronOS real-time scheduling API and, therefore, avoiding deadline misses.

Objective:

The aim of this project is to rewrite an open-source multimedia application in order to exploit real-time scheduling policies implemented in ChronOS. Through our project we plan to achieve the following goals:

- Investigate how to rewrite an open-source multimedia application ie. Mplayer in order to exploit real-time scheduling policies.
- Implement a prototype that exploits ChronOS scheduling.

Evaluate the implementation (with multiple scheduling algorithms).

About M-player:

'MPlayer - The Movie Player' is a free and open-source media player software initially written for Linux. The program is now available for other major operating systems like

Windows and OS X. [4] Development of MPlayer began in 2000. MPlayer can play a wide variety of media formats, supported by FFmpeg libraries, and can also save all streamed content to a file locally. Being an open-source software, the source code is freely available. The documentation of the player was well-written and thus we chose to go ahead with this player.

Approach to the Problem:

In this project, we rewrite M-player 1.2.1 to use ChronOS real-time scheduling API in order to make multimedia applications run smoothly. Our solution would try to avoid deadline misses or more specifically frame misses. We tweaked the source code of M-player for the project modifying it to incorporate RMA and EDF scheduling policies with ChronOS APIs.

Implementation:

The approach of our implementation was as follows:

- **Compiled and installed MPlayer along with ChronOS:**
  The major task was to compile and build MPlayer and get it running on the kernel. There were a lot of dependencies in this process which had to be resolved. After installing and debugging these dependencies we were able to successfully compile MPlayer. However, we did face problem enabling the audio but as we were only interested in the video frames, we did not tackle that issue for the current project.

- **Resolving dependencies:**
  There were various missing dependencies like **x11, gtk 2.0, yasm** etc. All were installed and included for a successful build of MPlayer. The cause of dependencies were the libraries and APIs that MPlayer uses for operations like multimedia-encoding, decoding, scheduling, playbacks etc.

- **Identify source code part responsible for playback feature and for decoding frames.**
  Since we had to tweak the source code, it was imperative to understand the code structure and working of MPlayer, mainly its functionalities, workflow and procedure calls. The audio and video player are built around a main periodic loop in which the decoding of the audio and video runs and passes the results of this decoding to the audio and video cards respectively. In mplayer.c file we found this loop which sends frame to decoding function. This loop starts from while (!mpctx->eof) which is in the

mplayer.c file.

- **How to incorporate MPlayer with ChronOS so that the MPlayer can make system calls with the ChronOS interface.**
  We tried to understand how **sched_test_app** uses system calls to schedule different tasks. The various library dependencies of sched_test_app were to be identified included in mplayer.c so that the audio and video frames decoding tasks can be scheduled based on real time schedulers like RMA and EDF.

  There are 2 system calls in ChronOS for communication between application and the real-time systems.
  begin_rt_seg: Begins a real-time segment for a given thread.
  end_rt_seg: Ends a real-time segment and returns a thread to the standard Linux Fair Group scheduler.

- **Test the functionality of our implementation by running the videos using different scheduling algorithms.**
  Frame decoding is a task in MPlayer. Thus each frame decoding will correspond to a task scheduling based on real time scheduling algorithms in ChronOS. The tasks which were not able to complete in sched_test_app corresponds to the number of frames which were not decoded.This is nothing but the number of frames dropped.

Experimental Environment:

All the experiments were conducted on UBUNTU 14.04.1 in VMWare Player 14 on intel i7-7700hq, 2.8GHz machine having 16 GB RAM, 2 TB of hard disk, 256 GB of SSD and running 64-bit windows 10 operating system.
The ubuntu server image used did not have a GUI which was a problem as we had to play a media player and thus without the GUI we could not have got the actual video output. So, in order to have the gui to display the actual video output and also not lose out on the functionality of the server, we installed the open source remote desktop protocol xRDP on our server so that we can access it using the remote desktop connection on windows. We then installed the XFCE4 desktop environment to get GUI we sought.

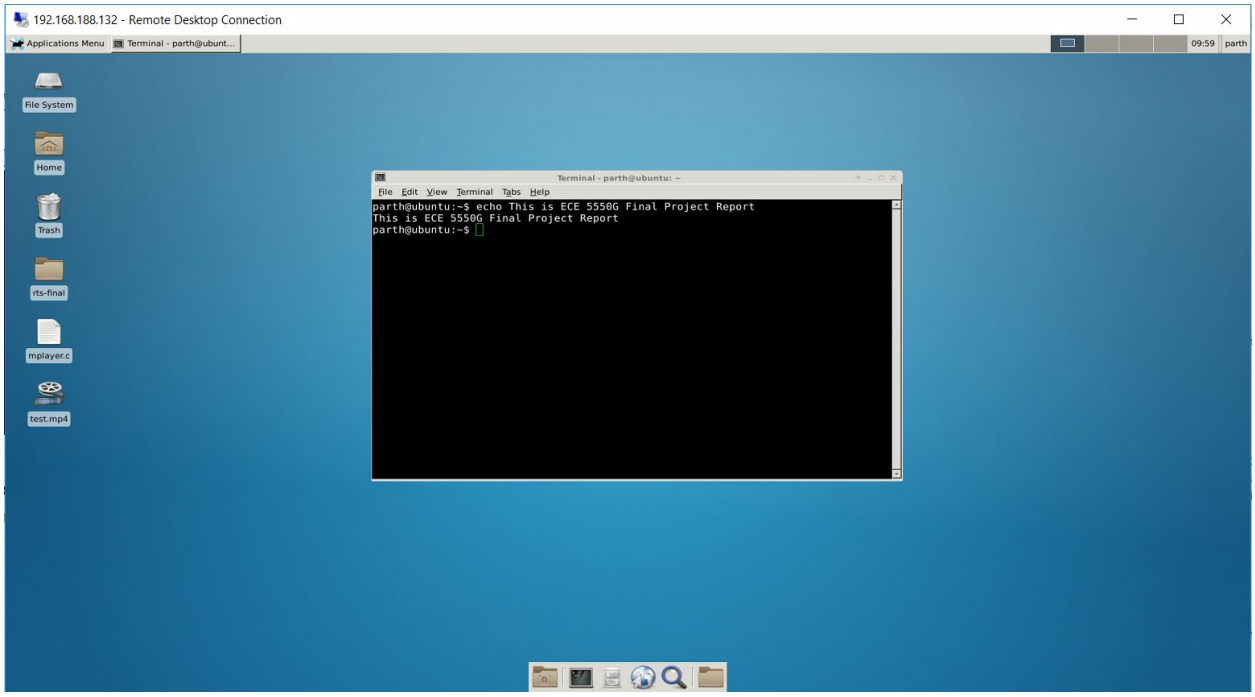

Figure 1: The XFCE4 desktop environment on ubuntu server accessed using remote desktop connection on windows

Now that we were able to enable the GUI, we were interested in playing multiple instances of the media player simultaneously as we wanted to investigate how the no. of frames(tasks) dropped were affected with the increase in the number of instances and in general the number of frames dropped in non real-time mplayer as compared to real-time media player.

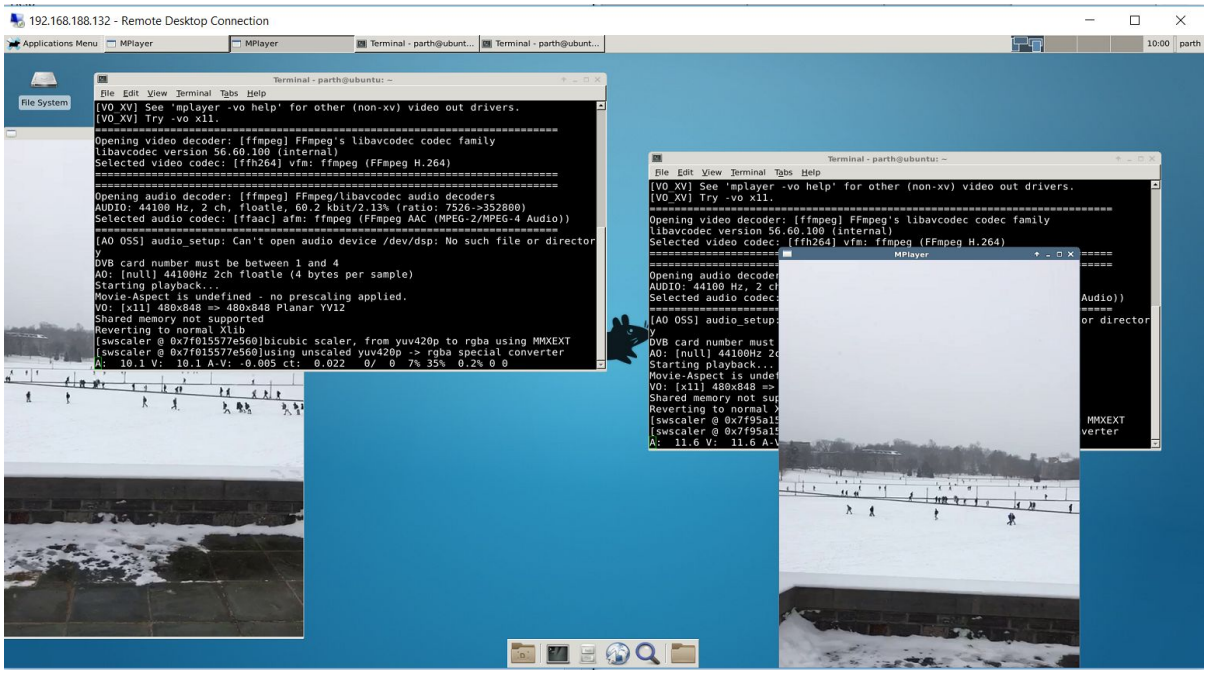

Figure 2: Multiple instances of the MPlayer executed

While playing a video we also observed that certain statistics of the video being played were displayed on the terminal window.

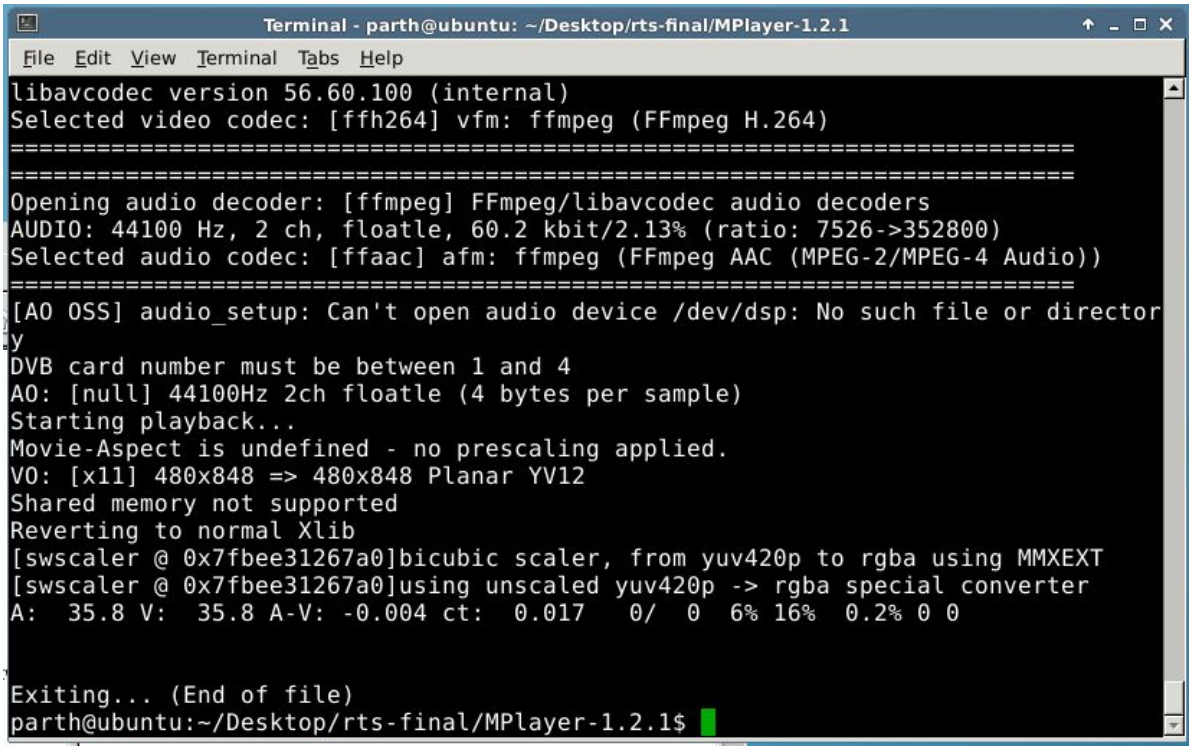

Figure 3: Terminal output of the MPlayer

As we can see in the figure 3, there are a number of statistics displayed on the command line when a video is played. We were able to get an explanation for these numbers on the FAQ page of he MPlayer.

A: audio position in seconds
V: Video position in seconds
A-V: audio-video difference in seconds(delay)
Ct: Total A-V sync correction done
frames played (counting from last seek) / frames decoded (counting from last seek)
video codec CPU usage in percent
video_out CPU usage
audio codec CPU usage in percent
frames needed to drop to maintain A-V sync

Outcomes:

The function mp_msg(MSGT_TV, MSGL_INFO,%s: %d frames successfully processed, %d frames dropped") helped us in calculating the value of the no. of frames dropped. We then played multiple instances of the mplayer to get the statistics in the following graph:

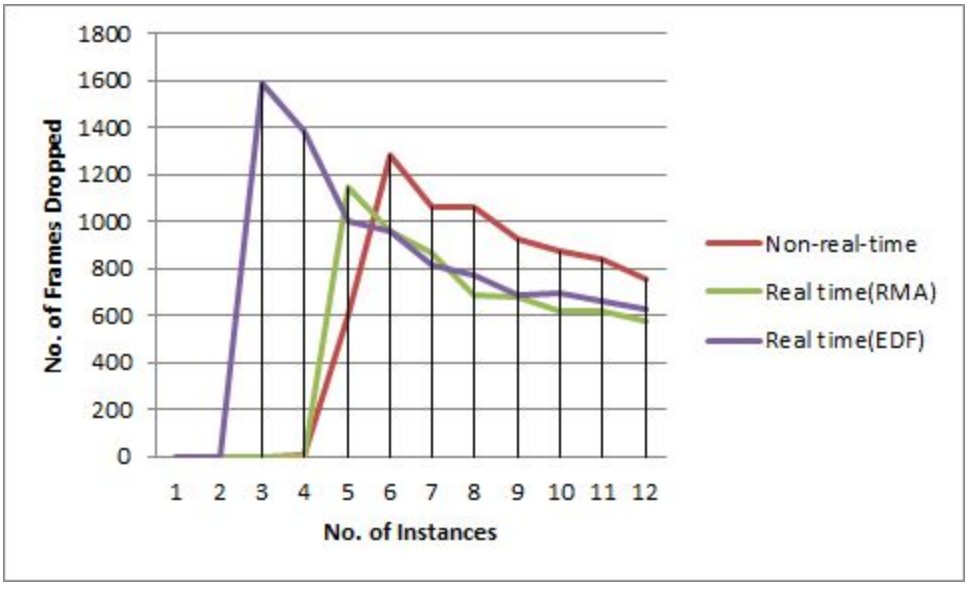

Figure 4: no. of instances v/s no. of frames dropped for basic MPlayer and RMA and EDF real-time MPlayer

From the above, graph we can observe that the frames dropped to maintain the audio-video sync are less in case of the real-time implementation of the MPlayer as compared to the

original MPlayer. For the different scheduling techniques, there is not much difference in the performance for RMA and EDF but still in most of the cases the number of frames dropped in case of RMA are less than EDF though the difference is not very substantial in most cases.

Conclusion:

We can observe from our experiments, that on an average the performance of the real-time implementation of the media player is indeed better as compared to the original media player. Our experiment was also extended to compare the results of the two scheduling schemes for the real-time implementation of the media player.

Future Work:

There is huge amount of future work that can be done in respect to this project. A number of other scheduling algorithms can be tested for the real-time implementation. Also, a comparative study can be made with the other media players and their real-time implementation. Also. there is a scope to see the effect of the video file format to investigate if the real-time implementation works better with certain video formats and codecs

Acknowledgements:

We would like to thank Dr. Tam Chantem for encouraging us to do this project. We would also like to thank the TA of the course, Mr. Sandeep Bijinemula for all the support and constant help by providing all the necessary files and documentation for installing the ChronOS and incorporating the ChronOS APIs in our project.

References:

1. http://www.irmosproject.eu/
2. Integrating multimedia applications in hard real-time systems. L. Abeni; G. Buttazzo, Real Time Systems Symposium, 1998. Proceedings., The 19th IEEE. pp 4-13 1998
3. Isovic, D.; Fohler, G. "*Quality aware MPEG-2 stream adaptation in resource constrained systems*", Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on, On page(s): 23 – 32
4. https://en.wikipedia.org/wiki/MPlayer