A VIDEO PROCESSING APPLICATION IN LITMUS-RT

S M Farhanur Rahman

Computer Science

University of Texas at Dallas

Richardson, TX 75080

[sxr190032@utdallas.edu](mailto:sxr190032@utdallas.edud)

Abstract

When using a regular non real-time video application, a common problem that users may face is their video lagging. Creating a video player that runs as a real-time task could solve that issue and give the user much smoother experience with their videos. To develop such a video player, real-time interfaces provided by Litmus-RT can be used, alongside FFMPEG and SDL libraries, to ensure that the data is processed and displayed faster. This solution can also be tested within Litmus-RT by using its evaluation interfaces to trace out the real-time jobs within the video player.

**1. Introduction**

Litmus-RT is a real-time extension to the Linux operating system. It modifies the kernel to support sporadic task model, modular scheduler plugins, and reservation-based scheduling. Clustered, partitioned, and global schedulers are included, and semi-partitioned scheduling is supported as well [1]. More information can be found on their website, litmus-rt.org. As stated on their website, the primary purpose of Litmus-RT is to provide a useful experimental platform for applied real-time systems research and, to that end, it provides abstractions and interfaces within the kernel that simplify the prototyping of real-time scheduling algorithms. This paper aims to use some of those interfaces to develop a video processing application and evaluate it using tracing tools that have also been provided in the modified kernel.

Before we go into the paper, it would be beneficial to look at what a movie file is so that things would be clearer when talking about the implementation of the video processing application. Movie files have a few basic components [2]. First, the file itself is called a container, and the type of container determines where the information in the file goes. Examples of containers are AVI and QuickTime. Next, there are a large number of streams, such as audio stream and video stream. A "stream" simply means a succession of data elements that are made available over time. The data elements in a stream are called frames. Each stream is encoded by a different kind of codec, which defines how the actual data is coded and decoded. Examples of codecs are DivX and MP3. Packets are then read from the stream. Packets are pieces of data that can contain bits of data that are decoded into raw frames that we can finally manipulate for our application. For our purposes, each packet contains complete frames, or multiple frames in the case of audio.

The way that the rest of this paper has been structured is as follows: Section 2 has a small discussion about works that are related to video processing. Section 3 describes my implementation of the application. Section 4 presents the evaluation tools in Litmus-RT and shows the real-time traces of the application. Lastly, the paper concludes with Section 5.

**2. Related Work**

As introduced, the goal of this paper is to develop a video processing application; one way to do that is using the FFMPEG framework. That is the approach that this paper will take, and there are various tutorials that exists on the internet to showcase how to use the framework. However, most of them use the old libraries from the framework that are now deprecated. The solution developed for this paper uses the up to date FFMPEG libraries. A high-level overview of the implementation will be discussed in the following section.

**3. Implementation**

This section describes the implementation of a video processing application in C using FFMPEG and SDL libraries. All the steps laid out in the following paragraphs have been carried out in the Litmus environment after the Litmus-RT extension was properly installed in the Linux Ubuntu distribution according to the instructions provided on their website [1].

The first step was to download the SDL and FFMPEG libraries. For my implementation, I used SDL1.2. Although version 2.2 was available, I stuck with version 1.2 due to a simple tutorial I found showing the steps to integrate with FFMPEG [2]. As for FFMPEG, I used a more up-to-date version: 4.1.3. Once those were downloaded, it was possible to use their provided libraries to make a video player application.

In Litmus-RT, there is a *liblitmus* interface that provides a C language API for interacting with the real-time operating system in order to build custom real-time tasks. The skeleton for the code can be found in */liblitmus/bin* under the file name *base\_task.c*, which creates single threaded tasks. There is also a multithreaded task option, but that won’t be the focus of this work.

After opening the file, the first thing it contained were the header files. It was necessary to add the header files for the new aforementioned libraries so that our code could make use of their API. Jumping to the main method, the first thing I needed to do, apart from what was already provided in the skeleton, was initialize SDL. Then I opened the video file from my file system and start looking for the audio and video streams in the file using FFMPEG. Once found, I then retrieved the Codec for both the audio and video so that SDL could use that information to play the video on its own window.

Furthermore, to handle the audio, a queue had to be implemented. The queue stores packets from the stream and a thread initiated by SDL continuously checks the queue for data to decode and play. However, right before the data is played, it needs to be resampled so that the proper output comes out from the speaker and not just noise. As for handling the video, a decoder again decodes the output data and when a whole frame has been decoded, the YUV display in the SDL window is updated to show the proper image.

Now to make this application a real-time task, the skeleton code provided a job function that periodically releases a task to complete. In my implementation, that job is to read the data, determine what kind of stream it is, and handle it accordingly. For a video stream, I found a complete frame and updated the SDL display. If it is an audio stream, I pushed that packet to the queue so that the audio thread would handle it. Lastly, the job also checked if a signal such as Ctrl-C is sent so that it may exit the application.

**4. Evaluation**

This section will be discussing the tools used to evaluate the real time task. Revisiting the code, the skeleton of the *base\_task.c* file defined some constant values as shown in Figure 1. These values determine how the jobs are released by assigning the period, deadline, and execution cost of a job. In the later figures, we’ll see this job being executed by different scheduling algorithms, namely GSN-EDF and P-FP. Litmus-RT provides a wider range of scheduling algorithms but for this paper we’ll only focus on those two.

A screenshot of a cell phone

Description automatically generated

Figure 1. Real-Time Constraints for Task

To evaluate the real-time aspect of the video processing application, the feather-trace interface of Litmus-RT is used; specifically, the *ST* visualizing tool. In order to create a trace, the first thing I needed to do was run the *st-trace-schedule* command and then execute the video processing application. When the video application finishes playing its video, I stop the trace program and it generates a binary file. This new file can then be used to create a time diagram and also look at job statistics. Executing *st-draw* creates a pdf with the schedules plotted on a time diagram and part of that is shown in Figure 2 and Figure 3.

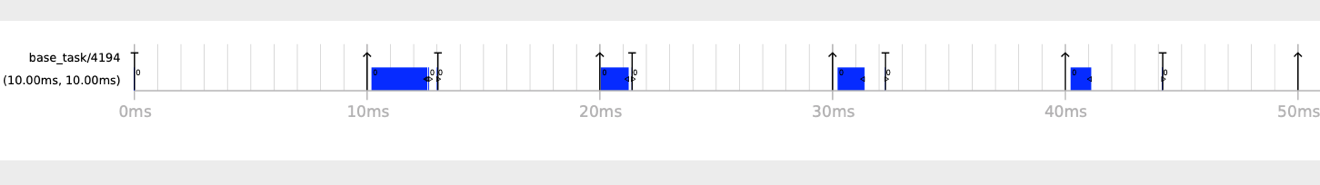


Figure 2. GSN-EDF Trace

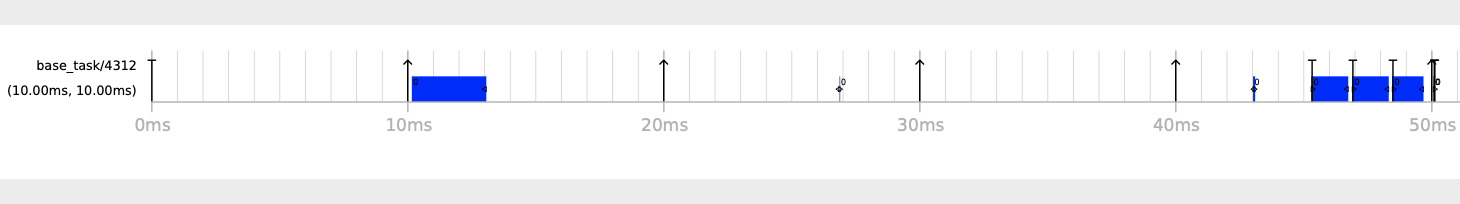


Figure 3. P-FP Trace

To gain more insights on the jobs being executed, feather-trace also provides the command *st-job-stats*. As shown in Figure 4 and Figure 5, it includes the period, response time, lateness and tardiness of the job; it also shows whether or not its deadline was missed, or if the job was forced. ACET stands for the Average Case Execution Time of a job. The number of preemptions and migrations the job caused is also shown.

A circuit board

Description automatically generated

Figure 4. GSN-EDF Statistics

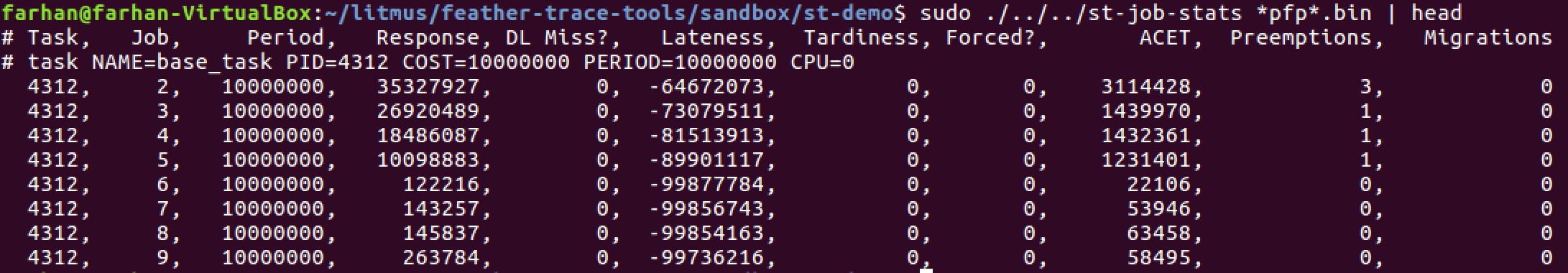


Figure 5. P-FP Statistics

**5. Conclusions**

The purpose of this project was to create a video player that would execute in real-time. This was done by using the FFMPEG and SDL libraries in the *base\_task.c* file that was provided in the *liblitmus* interface. Finally, running the *ST* evaluation tools proved that the video processing was done in real-time.

**References**

[1] J. Calandrino, H. Leontyev, A. Block, U. Devi, and J. Anderson, "LITMUSRT: A Testbed for Empirically Comparing Real-Time Multiprocessor Schedulers ", Proceedings of the 27th IEEE Real-Time Systems Symposium, pp. 111–123, December 2006.

B. Brandenburg, “Scheduling and Locking in Multiprocessor Real-Time Operating Systems”, PhD thesis, UNC Chapel Hill, 2011.

[2] “Tutorial 01: Making Screencaps.” *An Ffmpeg and SDL Tutorial*, Dranger, dranger.com/ffmpeg/tutorial01.html.