VIDEO PROCESSING APPLICATION USING FFMPEG AND SDL LIBRARIES IN LITMUS^RT KERNEL

A detailed project report by Samuel Priyadarshan Selvakumar Kingslin – (**SXS200367**)

1. INTRODUCTION

The main idea behind this project is to create a video processing application with the help of libraries such a FFMPEG and SDL. The problem with conventional video players such as VLC or QuickTime video players is that they handle non-real-time application and process them. However, this can be a problem when it comes to a smoother video play, which should be ultimate intuition behind any conventional video processing application. The goal behind this project is to use Dynamic Priority Scheduling Algorithm such as EDF to schedule the video, making it a real-time application. This can be done with the help of the LITMUS^RT kernel, which is a testbed for Multiprocessor Scheduling. However, this project will only be focusing on the multiprocessor. This project includes a small demo video [5] that can be found on the reference.

2. VIDEO PROCESSING

For this project, an open-source tutorial by dranger has been followed. It explains the general workflow behind the working of the video processing in a step-by-step fashion. The intuition behind the implementation is to run this code as a real-time application for the non-real-time event as the video can be highly sporadic in nature. The versions and implementation of this video processing application will be discussed later in this project as it was the most crucial part when it came to implementing the idea. But for now, the motivation behind the video processing will be discussed.

As mentioned earlier, the main libraries that make this application possible are FFMPEG and SDL. To start with, it is essential to understand the framework of a video and this code. The code written in C can be called as the container. For example, conventional videos such as MP4 or MKV can be called as a **container**. The containers will generally have a stream of data which can be called as the **frame**. These frames depend on how many frames are included in the video, which makes the video look smoother depending on its proportionality to the frames. These frames have both video and audio **codecs**. Examples of video codecs are MPEG, DivX while the examples of audio codecs can be ACC, DTS, etc. They play a vital role in encoding and decoding the stream, formatting the file into the specific standards. These stream, in return the decoded **packets**, which can be the actual raw data that can be processed.

Starting with FFMPEG, the libraries provided by it can give libraries that enable the basic decoding of the input file. In this project a sample video file of format 'MPEG' with a resolution of 960x400 with audio codec of MP2 and video codec MPEG-2 has been used.

3. IMPLEMENTATION

This is the most critical section of the project, and a great amount of time was spent just on the set-up for the video processing application to open the sample video file and process it without a hassle. All the steps involved will be explained briefly in this session. To make sure that the video processing works with LITMUS^RT kernel, it is essential for it to work in gcc. So to start off, the video processing application was run in Ubuntu 20.04 with gcc 4.9.0. As the LITMUS^RT kernel VirtualBox image uses ubuntu 14.04, which is archaic, the libraries used in the video processing had to be older libraries as newer libraries expect higher version of gcc to process. Upon finally compiling the open-source code, the implementation had to be replicated in LITMUS^RT kernel. But unfortunately, it din't work as the codec was not supported in 14.04. To go around this, the LITMUS^RT kernel had to be compiled and booted on Ubuntu 20.04, where it is working extremely fine. However, after weeks of struggle on that, the kernel dint boot at all. After carefully going over the manual over again, it came to light that LITMUS^RT kernel has only been tested only in Ubuntu 14.04 and 16.04. Hence the entire process had to be replicated all over again on Ubuntu 16.04. But this time, the kernel was booted on Ubuntu 16.04 instead of loading the FFMPEG and SDL libraries as it was a smarter choice and time saving. This step worked as planned and finally the video processor was able to run on the booted kernel.

The implementation uses the open-source code used in the dranger tutorial. The main function calls just one function *avcodec_decode_video2(pCodecCtx, pFrame, &frameFinished, &packet);* that processes the file and returns the decoded frame. However, on browsing the library header files included with the code (*libavcodec/avcodec.h*), it was evident that, this function could actually be replaced by *avcodec_send_packet()* & *avcodec_receive_frame()* which is required as it can come handy to test the processing. For integrating it with litmus, liblitmus has a set of predefined templates that can come in handy. For testing the functionality of the code, the base_task.c file under /bin was used as a template, which can be used to make the real-time implementation. This function call can be called in *int job()* which is specified in the main function. With the completion of each frame, it is returned to the main code and then this is looped until all the frames have been received and processed. Any errors in-between can cause and error and the execution is aborted. This is mainly handled by the int ret variable which acts a flag along with every execution. There is another flag called the int do_exit, which lets the main function if the job has completed its execution. This video file that has been used has been called 3059 times under **GSN-EDF**. The details about this scheduler will be discussed in the upcoming sections.

4. FLOW CHART

Figure. 1 shown below represents the basic workflow of the template API provided in liblitmus. The function call used in the main function is put into the job() function. In addition to this, the job number is printed each time the job() function is called.

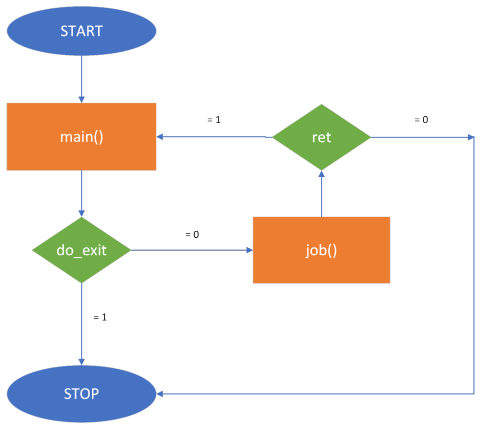


Figure. 1

5. ALGORITHM

The base_task.c template file, which has been used for this project, uses two main functions namely the main() and job(). The basic workflow of the algorithm has been explained in Figure. 1. Upon calling the *avcodec_send_packet(),* the video packet is sent for decoding. If the packet is not sent properly, it crashes the program thereby exiting. If the packet is successfully received, it then moves to forward by receiving the frames from the decoded packet by calling *avcodec_receive_frame()*. The RAW frame data obtained in this step must then be displayed in a window, thereby achieving the goal. This is made possible with the help of the SDL libraries that are called where the processed decoded data can be put to display. The *SDL_DisplayYUVOverlay()*. This function gets the processed data and then is responsible for displaying the decoded frame on the video player.

This process is repeaded for all the remaining frames and once all the frames have been displayed on the video player, the ***job()*** must be stopped. This is achieved by returning the value of ret at any point of the loop to do_exit variable, which controls the do while loop of the template code. Most importantly, the SDL libraries shouldn't take over the main() function and this is implemented by a simple definition. Finally, the code should also make sure to exit the looping in the event of pressing Ctrl+C. This has been implemented using a simple case statement that can override the execution of the program.

6. EXECUTION

The code is run under GSN-EDF scheduler which is a global scheduler. By default the standard Linux scheduler is set. This must be manually changed before the code could be executed. This is set by the command <setsched GSN-EDF> on the Linux command. In the base_task.c, the task_mode(LITMUS_RT_TASK) is being called to ensure that the entire processing happens as a real-time and not as a non-real-time function. The application is run with the default timings provided with the template. However, changing these parameters keeping the utilization under consideration should produce a feasible schedule under EDF as discussed in the lectures. This code can do processing on multiprocessors just by setting the 'param.cpu' to how much ever CPU it has to be tested on. The base_mt_task.c template can provide the provision to carry out the operations using multi-thread. But that will not be the scope of this project. The results obtained in this project will be on a single processor run using a single thread. This code is then compiled by doing a simple Makefile that can link all the libraries to the desired file thereby giving the output file.

7. TRACING

With LITMUS^RT, tracing can be a piece of cake with the commands found on feather-trace. The developers of this kernel came up with algorithms that can convert the bin file into graphical representation that con come in handy for debugging and analysis. To go by this tracing, the <st-trace-schedule> command must be invoked. This can start the tracing for the any real-time application unless cancelling the tracing by hitting the return button. Once this command is invoked on the terminal, another terminal must be opened where the file is executed. Upon completing the execution, the <wait> command is used to stop all the RT launches.

Upon completing the task, the <st-trace-schedule> command must be stopped which saves the logs as a bin file. This bin file contains all the analysis done during the execution of the file when run as a real-time application. This bin file is then analyzed and graphically represented as a timing graph using the command <st-draw>. This runs a python script that can measure all the deadlines and plot them as a graph that helps with analysis. The plot for this project can be seen in Figure 2.

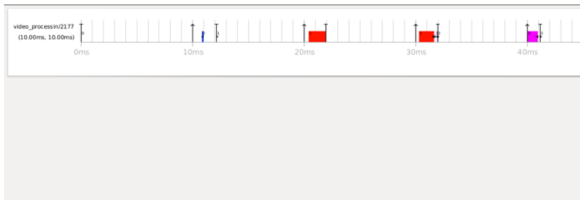
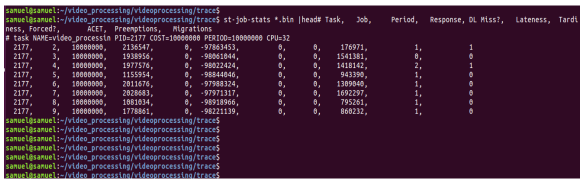


Figure. 2

It is evident in the picture that the scheduler can access various resources indicated in different color. The tracing also gives the provision to see which resource is being accessed and when which is very convenient. As the project is done on a virtual machine, the overheads that occur with the processor can go as high as up to 2ms which is not desired as EDF is a scheduler with minimal overheads. However, there's no way around this if working on a Virtual Machine as it becomes a trade-off.

Finally, the parameters are used to analyze the deadline hit / miss of the jobs run as real-time along with basic characteristics such as preemption, response, lateness, Average Case Execution Time (ACET) are obtained. These results are obtained by the command <st-job-stats>. These results can be very useful in analyzing the real-time operation of the program. Figure.3 gives all the information for this project video file.



```
samuel@samuel:~/video_processing/videoprocessing/trace$
samuel@samuel:~/video_processing/videoprocessing/trace$ st-job-stats *.bin |head# Task,   Job,     Period,   Response, DL Miss?,   Lateness,  Tardi
ness, Forced?,       ACET,  Preemptions,   Migrations
# task NAME=video_processin PID=2177 COST=10000000 PERIOD=10000000 CPU=32
  2177,     2,   10000000,    2136547,        0,  -97863453,         0,        0,     176971,          1,            1
  2177,     3,   10000000,    1938956,        0,  -98061044,         0,        0,    1541381,          0,            0
  2177,     4,   10000000,    1977576,        0,  -98022424,         0,        0,    1418142,          2,            1
  2177,     5,   10000000,    1155954,        0,  -98844046,         0,        0,     943390,          1,            0
  2177,     6,   10000000,    2011676,        0,  -97988324,         0,        0,    1309040,          1,            0
  2177,     7,   10000000,    2028683,        0,  -97971317,         0,        0,    1692297,          1,            0
  2177,     8,   10000000,    1081034,        0,  -98918966,         0,        0,     795261,          1,            0
  2177,     9,   10000000,    1778861,        0,  -98221139,         0,        0,     860232,          1,            0
samuel@samuel:~/video_processing/videoprocessing/trace$
samuel@samuel:~/video_processing/videoprocessing/trace$
samuel@samuel:~/video_processing/videoprocessing/trace$
samuel@samuel:~/video_processing/videoprocessing/trace$
samuel@samuel:~/video_processing/videoprocessing/trace$
samuel@samuel:~/video_processing/videoprocessing/trace$
samuel@samuel:~/video_processing/videoprocessing/trace$
```

Figure. 3

Other schedulers such as P-FP, PFAIR, PSN-EDF were also tried but it gave an unknown error -22 and hence the further steps couldn't be moved any further with the tracing using other schedulers.

8. SCHEDULER

In this section, the details regarding the EDF scheduler will be discussed. EDF stands for Earliest Deadline First. As the name states, the jobs with deadlines that come the earliest will be executed with the highest priority while the jobs with their deadlines later will execute with lower priority, depending on their deadline. EDF is a dynamic priority scheduler. Any sporadic task with a feasible schedule can be scheduled under EDF which has been discussed in Liu's book and the lectures. The most significant thing to note is that, under EDF, for the scheduler to produce a feasible schedule, the total utilization of the jobs must be less than or equal to one. Else, under EDF scheduling, the job will for sure miss its deadline. Not much details have been documented regarding GSN-EDF in the official manual of the LITMUS^RT. However, as the name suggests, the scheduling should happen as a global agent and should not be partitioned.

9. SOFTWARE SETUP

The following versions are used:

- GCC – 5.4.0
- Ubuntu – 16.04 (booted with litmus^RT)
- FFMPEG – 4.2
- SDL – 2.2.0.22

10. CONCLUSION

To conclude, the video processing application has been designed and then tested as a real-time application under GSN-EDF scheduler. The video processing application has been made possible with the libraries provided by FFMPEG and SDL.

REFERENCES

[1] https://www.litmus-rt.org/tutor16/manual.pdf

[2] http://dranger.com/ffmpeg/tutorial03.html

[3] https://www.litmus-rt.org/tutorial/tutorial-slides.pdf

[4] https://microcontrollerslab.com/earliest-deadline-first-scheduling/

[5] https://drive.google.com/file/d/1az8YRhRzVekwHQaPEWt9rMpw_Y1_nmuu/view?usp=sharing