ECEN5623:Real-Time Embedded Systems

Summer 2017

Extended Lab Report

Time lapse project

Aasheesh Dandupally and Omkar Prabhu

Introduction

This report contains design,testing, analysis and results of Extended lab time lapse project for ECEN5623 Real time embedded systems for Summer 2017. The project aims to meet the requirements defined below to create a time lapse video from captured images.

Functional requirements

Minimum requirements

1) Resolution of the camera is 640\*480

2) Frames are acquired at 1Hz

3) Image file format is PPM P3(for triggering motion detection) and P6(capture and other services) RGB format

4) Timestamps are embedded in the PPM header using date command in Linux

5) Output of “uname –a” is embedded in the PPM header of every frame

Minimum extra requirements

1. Image processing- Sharpening of image

Target requirements

1. Compression of PPM files to JPEG files
2. Transfer acquired images over Ethernet

Stretch requirements

1. Acquire images at 10Hz

High Level System and Software design

This section would cover the high level system and software design. The block diagrams are presented in the following order 1) Minimum requirements 2) Minimum extra requirements 3) Target requirements.

Hardware design

Our time lapse acquisition system is running on Laptop with native Linux Ubuntu 16.04 distribution. The camera device used for this project is Logitech C270.

System design

The goal of the system is to design a time lapse acquisition system which would capture images at 1Hz. After capturing 2000 frames, the images are embedded with timestamps and system information in the header of the acquired images. After adding the metadata in the images, they undergo image processing which sharpens the image. After image processing, the images are compressed from PPM format to JPEG format to save space on filesystem. A typical PPM image is ~920 KB and after compression, the image in JPEG format is around ~60KB. After acquiring the compressed images, they are sent over Ethernet to another system via NFS so that our acquisition system can save memory on filesystem.

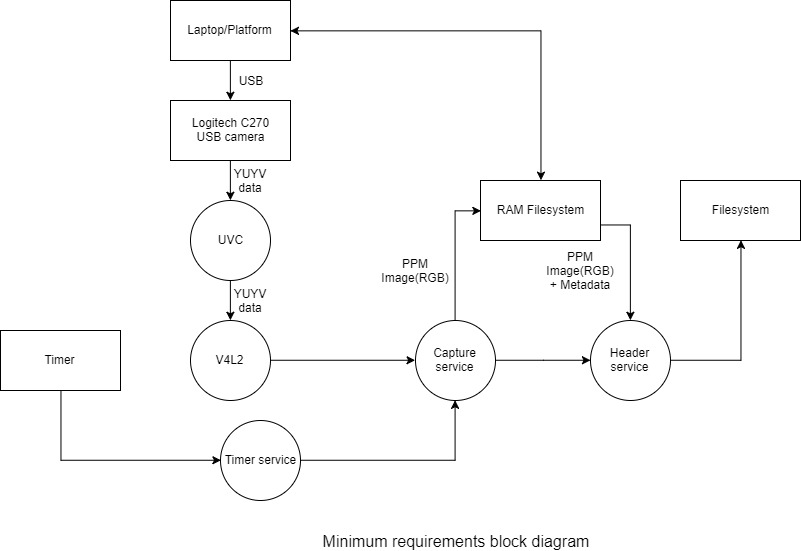
Software design

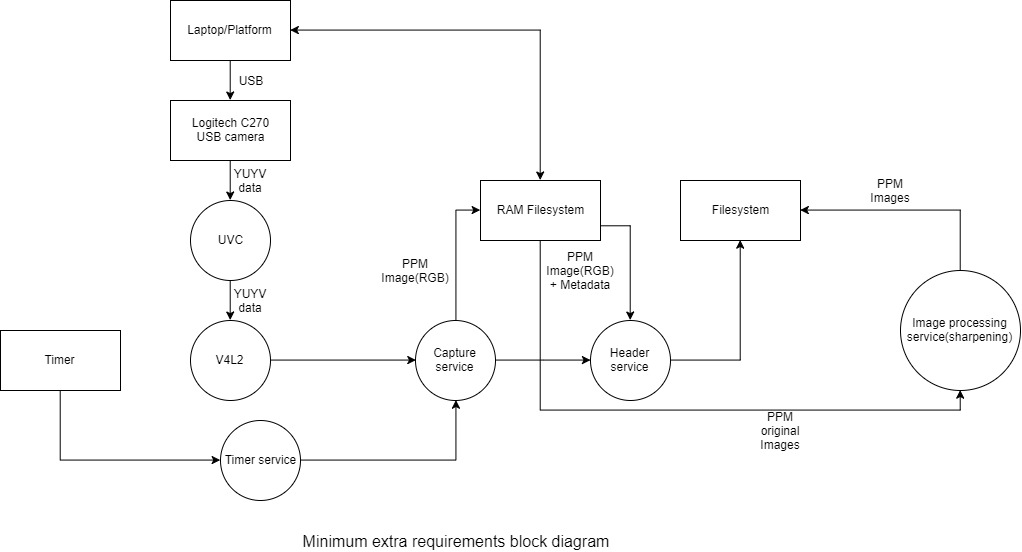
We are using a realtime clock as a timer to trigger capturing images at 1Hz.Here we have given highest priority to timer thread which keeps track of time and once it reaches 1 second it passes control to capture thread using semaphore. Images are acquired and stored in ram file system to make it more efficient. After acquiring images, header service is called which would insert metadata like timestamp and system information in the PPM header. Linux system calls “date” and “uname -a” are used to get the metadata and file descriptors are used to insert the metadata in the PPM header. RAM filesystem is used to save images which would improve the performance of the services. After that, the image processing service called sharpening service is called which would sharpen the image and this is achieved by blurring the image and then subtracting that from the original image. openCV API’s are used to achieve the same. After acquiring the sharpened images, the compression service is called which would convert the acquired PPM images into JPEG files.This is achieved using openCV API’s as well. After compression, the images are sent over Ethernet to another system via NFS.

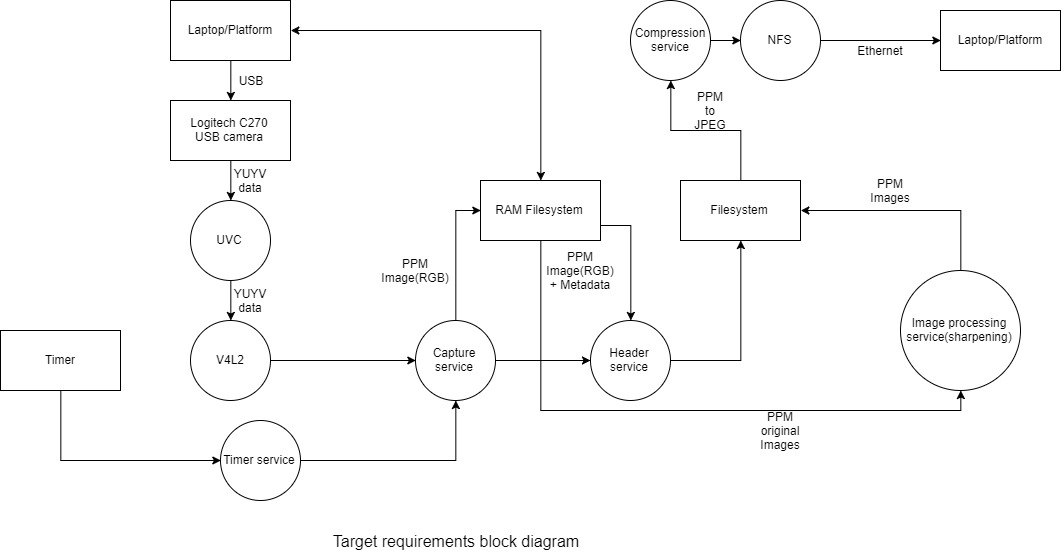
The command used to create a RAM filesystem is as follows

$ sudo mount -t tmpfs -o size=512m tmpfs /mnt/ramfs

This would create a RAM filesystem of fixed size 512MB on /mnt/ramfs







Real time requirements and services

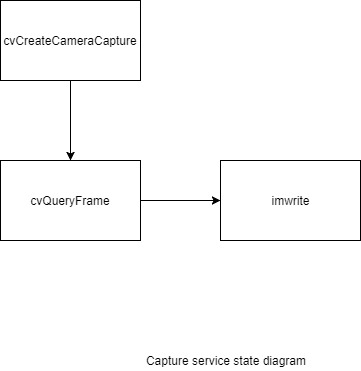
Timer service:- This service would enable the capture service to acquire frames at a given frequency I.e 1Hz

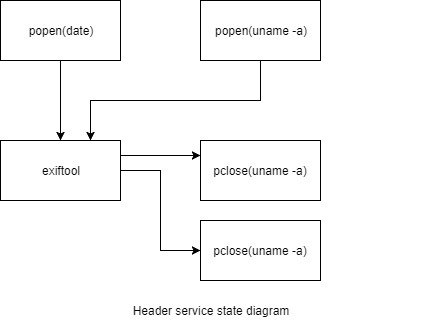
Capture service:- This service acquires frames from camera and saves them in PPM format on the RAM filesystem

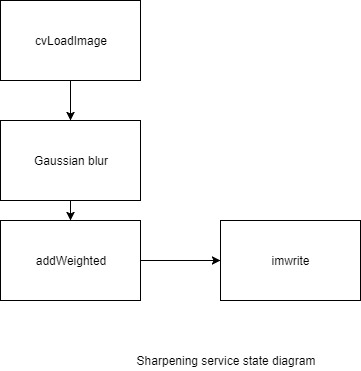
Header service:- This service inserts the timestamps and system information in PPM images

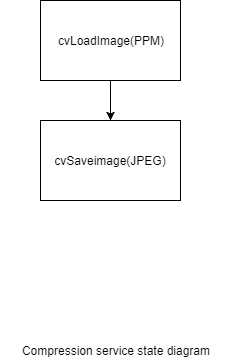
Image processing service:- This service performs sharpening of the image by applying gaussian blur and subtracting the the blur image with original

Compression service:- This service reads the PPM images and converts them JPEG images and saves them to a Network filesystem









Testing and Analysis

We are using binary semaphores for synchronization between threads. All the threads are following SCHED FIFO SMP.

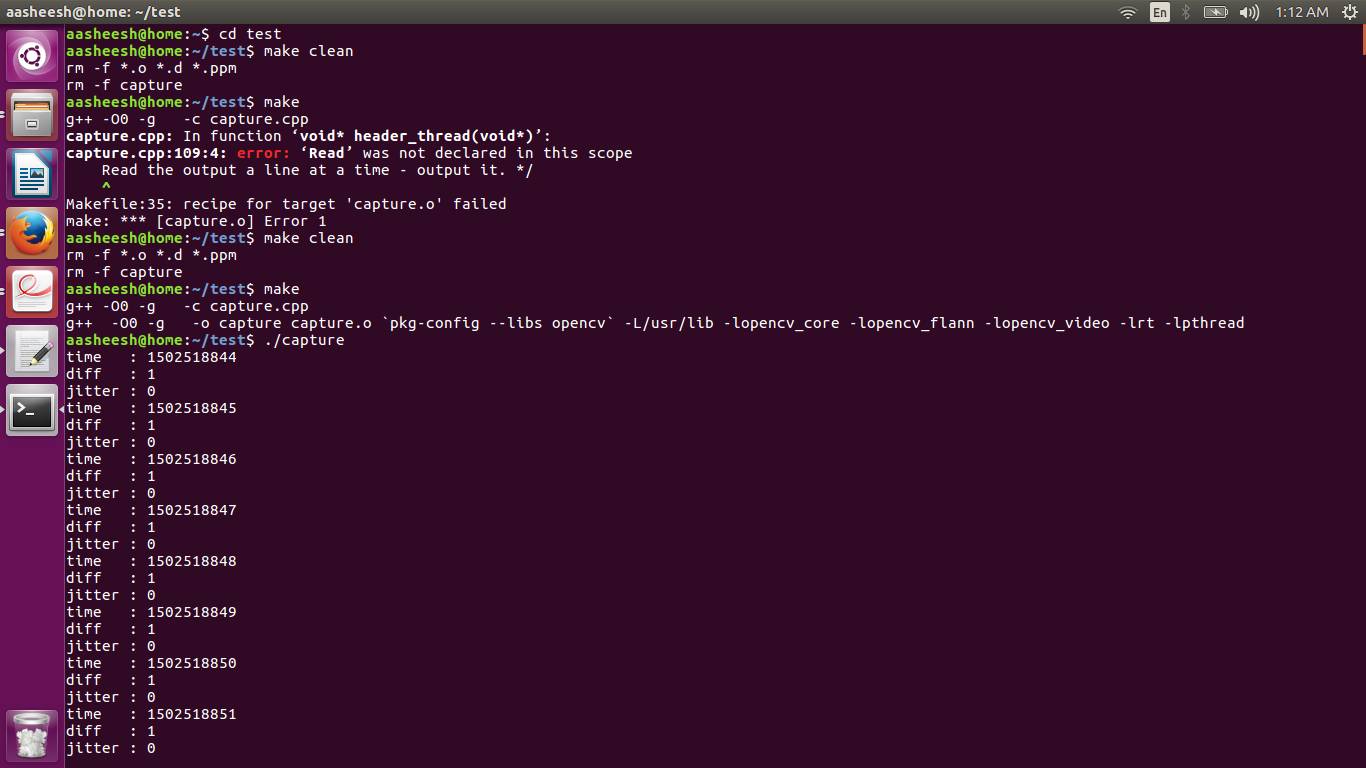
In our code there are 4 services as of now as we have explained. In that frame acquisition is the real time service. For accurate timer mechanism we are planning to add one more thread which would be of highest priority and that is timer service where we are assigning timer to one core that is to hardware so that it would be deterministic and it would trigger capturing service by semaphore. We are using push back method in compression. For sharpening we are using guassian blurr API and in other services such as adding header we are adding fprintf statements to print in file.

Deadline analysis

Here we have kept deadline equal to time period since it is an RM analysis. So deadline is 1 second for both the services.

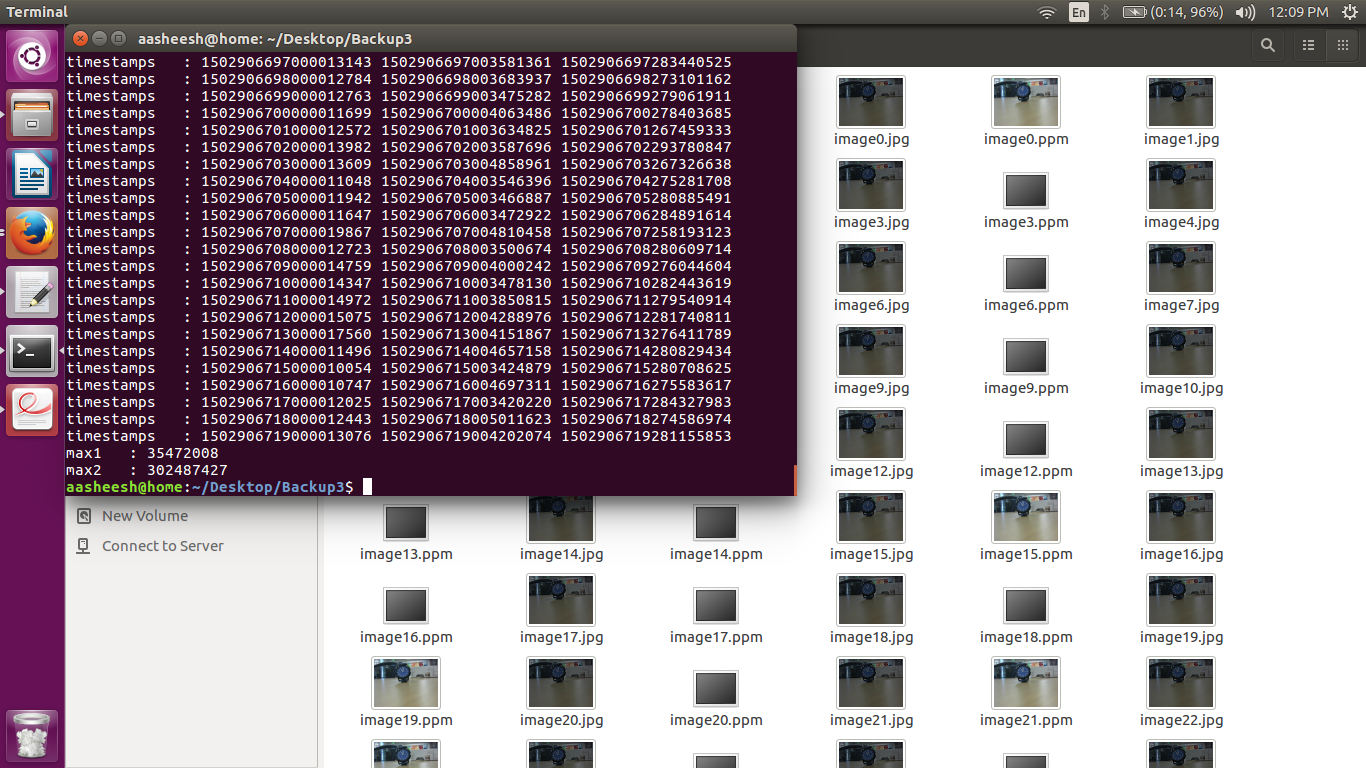
Jitter Analysis

The Accumulated jitter is 0 seconds in 2000 frames and average jitter is close to 0 for that we have taken few screenshots which we have added here. For that we have created arrays in code which stores information about jitter and with the help of that we are printing jitter and calculating average jitter





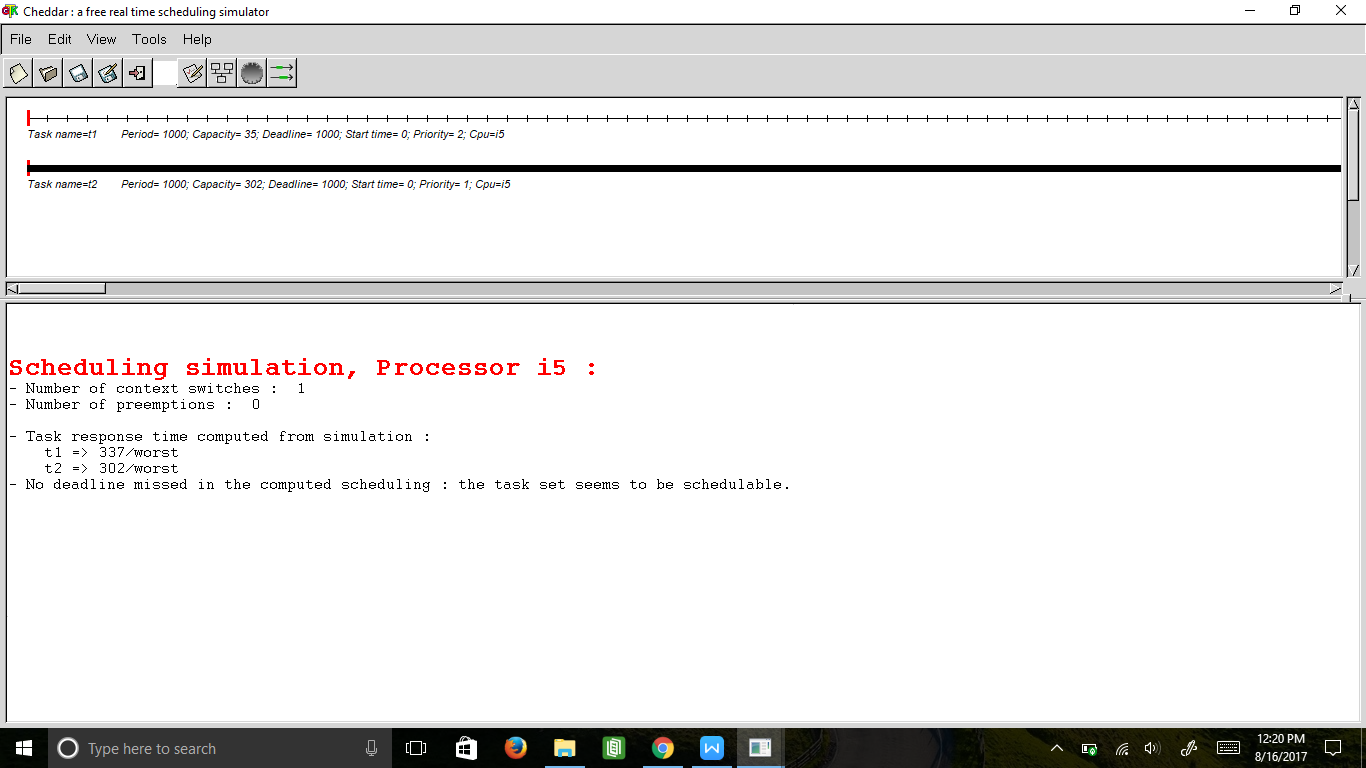
Worst case execution time



Scheduling point/Completion tests



Cheddar Analysis



Safety margin analysis

Since it is soft real time service we are keeping 5% margin

Conclusion

The time lapse acquisition system shows that a soft real time system can be implemented on native linux with POSIX threads which provide schedulability and synchronization. In which use of hardware timer is important due to it’s deterministic nature. Hence we are getting 0 jitter over 2000 frames thus system is accurate. We have also confirmed its schedulablity using tests and done cheddar analysis which proves that it is feasible.

References

Reading and writing images

<http://docs.opencv.org/2.4/modules/highgui/doc/reading_and_writing_images_and_video.html>

Image sharpening adapted from

<https://stackoverflow.com/questions/4993082/how-to-sharpen-an-image-in-opencv>

Header service reading system calls adapted from

<https://stackoverflow.com/questions/646241/c-run-a-system-command-and-get-output>

POSIX calls and scheduling adapted from examples by Professor Samuel Siewart and Beginning Linux Programming 4Th edition Chapter 12 POSIX programming

Appendix

/\*

\*

\* Adapted from example by Sam Siewert

\*

\*/

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <iostream>

#include <pthread.h>

#include <time.h>

#include <semaphore.h>

#include "opencv2/imgproc/imgproc.hpp"

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

using namespace cv;

using namespace std;

CvCapture\* capture;

sem\_t bin\_sem1;

sem\_t bin\_sem2;

sem\_t bin\_sem3;

sem\_t bin\_sem4;

sem\_t bin\_sem5;

sem\_t bin\_sem6;

sem\_t bin\_sem7;

#define HRES 640

#define VRES 480

#define BILLION 1E9;

time\_t timestamp[2000];

time\_t diff[2001];

signed int jitter[2000];

static double timestamp1(void)

{

int i,ret = 0;

struct timespec curr\_time;

double time;

int8\_t arr1[10],arr2[10];

ret = clock\_gettime(CLOCK\_REALTIME, &curr\_time);

{

if(ret != 0)

printf("clock\_gettime failed!");

}

time = ((curr\_time.tv\_sec) + (curr\_time.tv\_nsec)) / BILLION;

return time;

}

void \*timer\_thread(void \*arg)

{

struct timespec curr\_time;

time\_t cu\_timer;

int count = 1;

while(1)

{

sem\_wait(&bin\_sem5);

clock\_gettime(CLOCK\_REALTIME, &curr\_time);

time\_t aq\_timer = curr\_time.tv\_sec;

while(1)

{

clock\_gettime(CLOCK\_REALTIME, &curr\_time);

cu\_timer = curr\_time.tv\_sec;

if(cu\_timer - aq\_timer == 1)

{

sem\_post(&bin\_sem4);

count++;

break;

}

}

if(count == 2000)

{

break;

}

}

}

void \*capture\_thread(void \*arg)

{

int i,ppm=1,sleep = 1000000;

char a[15];

CvCapture\* capture;

IplImage\* frame;

struct timespec curr\_time;

cvSetCaptureProperty(capture, CV\_CAP\_PROP\_FRAME\_WIDTH, HRES);

cvSetCaptureProperty(capture, CV\_CAP\_PROP\_FRAME\_HEIGHT, VRES);

capture = cvCreateCameraCapture(2);

for(i=0;i<4;i++)

{

frame = cvQueryFrame(capture);

}

for(i=0;i<2000;i++)

{ sem\_wait(&bin\_sem4);

frame = cvQueryFrame(capture);

sem\_post(&bin\_sem5);

//clock\_gettime(CLOCK\_REALTIME, &curr\_time);

//timestamp[i] = curr\_time.tv\_sec;

sprintf(a,"/mnt/ramfs/image%d.ppm",i);

cvSaveImage(a,frame,&ppm);

sem\_post(&bin\_sem1);

}

cvReleaseCapture(&capture);

}

void \*header\_thread(void \*arg)

{

FILE \*fp1;

FILE \*fp2;

int i=0;

char uname[1000],date[100],out[1000],move[100];

char a[15];

IplImage\* src;

for(i=0;i<2000;i++)

{

sem\_wait(&bin\_sem1);

/\* Open the command for reading. \*/

fp1 = popen("date", "r");

if (fp1 == NULL) {

printf("command fail\n" );

exit(1);

}

/\* Open the command for reading. \*/

fp2 = popen("uname -a", "r");

if (fp2 == NULL) {

printf("command fail\n" );

exit(1);

}

fgets(date, sizeof(date), fp1);

fgets(uname, sizeof(uname), fp2);

date[strcspn(date, "\n")] = 0;

uname[strcspn(uname, "\n")] = 0;

sprintf(out,"exiftool '-Comment=%s %s' /mnt/ramfs/image%d.ppm",date,uname,i);

system(out);

sprintf(move,"mv /mnt/ramfs/image%d.ppm /home/aasheesh/Desktop/Backup3/",i);

system(move);

/\* close \*/

pclose(fp1);

pclose(fp2);

sem\_post(&bin\_sem2);

}

pthread\_exit(NULL);

}

void \*sharpening\_thread(void \*arg)

{

int i, j, kernel\_length = 31,jpg\_index = 2;

Mat dst\_frame;

IplImage\* src;

char a[15],b[15],remove[100];

IplImage\* result;

for(i=0;i<2000;i++)

{ sem\_wait(&bin\_sem2);

sprintf(a,"/mnt/ramfs/image%d.ppm\_original",i);

src = cvLoadImage(a);

Mat mat\_frame(src);

GaussianBlur(mat\_frame, dst\_frame, Size(0, 0), 3);

addWeighted(mat\_frame, 1.5, dst\_frame, -1, 0, dst\_frame);

sprintf(b,"sharp\_image%d.ppm",i);

imwrite(b, dst\_frame);

sprintf(remove,"rm /mnt/ramfs/image%d.ppm\_original",i);

system(remove);

sem\_post(&bin\_sem3);

}

}

void \*compression\_thread(void \*arg)

{

int i, png\_index = 2;

char a[15],b[15];

IplImage\* comp\_frame;

double start,end;

for(i = 0; i < 2000; i++)

{ sem\_wait(&bin\_sem3);

sprintf(a,"sharp\_image%d.ppm",i);

comp\_frame = cvLoadImage(a);

sprintf(b,"image%d.jpg",i);

cvSaveImage(b,comp\_frame,&png\_index);

end = timestamp1();

}

}

int main(void)

{

sem\_init(&bin\_sem1,0,0);

sem\_init(&bin\_sem2,0,0);

sem\_init(&bin\_sem3,0,0);

sem\_init(&bin\_sem4,0,1);

sem\_init(&bin\_sem5,0,0);

int ret,max\_priority,min\_priority;

pthread\_t thread0,thread1,thread2,thread3,thread4;

pthread\_attr\_t thread0\_attr,thread1\_attr,thread2\_attr,thread3\_attr,thread4\_attr;

struct sched\_param scheduling\_value;

cpu\_set\_t core0,core1,core2,core3;

CPU\_ZERO(&core0);

CPU\_SET(0,&core0);

CPU\_ZERO(&core1);

CPU\_SET(1,&core1);

CPU\_ZERO(&core2);

CPU\_SET(2,&core2);

CPU\_ZERO(&core3);

CPU\_SET(3,&core3);

ret = pthread\_attr\_init(&thread0\_attr);

if (ret != 0) {

printf("Attribute creation failed");

}

ret = pthread\_attr\_init(&thread1\_attr);

if (ret != 0) {

printf("Attribute creation failed");

}

ret = pthread\_attr\_init(&thread2\_attr);

if (ret != 0) {

printf("Attribute creation failed");

}

ret = pthread\_attr\_init(&thread3\_attr);

if (ret != 0) {

printf("Attribute creation failed");

}

ret = pthread\_attr\_init(&thread4\_attr);

if (ret != 0) {

printf("Attribute creation failed");

}

ret = pthread\_attr\_setschedpolicy(&thread0\_attr, SCHED\_FIFO);

if (ret != 0) {

printf("Setting Scheduling policy failed");

}

ret = pthread\_attr\_setschedpolicy(&thread1\_attr, SCHED\_FIFO);

if (ret != 0) {

printf("Setting Scheduling policy failed");

}

ret = pthread\_attr\_setschedpolicy(&thread2\_attr, SCHED\_FIFO);

if (ret != 0) {

printf("Setting Scheduling policy failed");

}

ret = pthread\_attr\_setschedpolicy(&thread3\_attr, SCHED\_FIFO);

if (ret != 0) {

printf("Setting Scheduling policy failed");

}

ret = pthread\_attr\_setschedpolicy(&thread4\_attr, SCHED\_FIFO);

if (ret != 0) {

printf("Setting Scheduling policy failed");

}

max\_priority = sched\_get\_priority\_max(SCHED\_FIFO);

min\_priority = sched\_get\_priority\_min(SCHED\_FIFO);

scheduling\_value.sched\_priority = max\_priority;

ret = pthread\_attr\_setschedparam(&thread0\_attr, &scheduling\_value);

if (ret != 0) {

printf("Setting scheduling priority failed");

}

scheduling\_value.sched\_priority = max\_priority-10;

ret = pthread\_attr\_setschedparam(&thread1\_attr, &scheduling\_value);

if (ret != 0) {

printf("Setting scheduling priority failed");

}

scheduling\_value.sched\_priority = max\_priority-30;

ret = pthread\_attr\_setschedparam(&thread2\_attr, &scheduling\_value);

if (ret != 0) {

printf("Setting scheduling priority failed");

}

scheduling\_value.sched\_priority = max\_priority-40;

ret = pthread\_attr\_setschedparam(&thread3\_attr, &scheduling\_value);

if (ret != 0) {

printf("Setting scheduling priority failed");

}

scheduling\_value.sched\_priority = max\_priority-20;

ret = pthread\_attr\_setschedparam(&thread4\_attr, &scheduling\_value);

if (ret != 0) {

printf("Setting scheduling priority failed");

}

pthread\_attr\_setaffinity\_np(&thread0\_attr, sizeof(cpu\_set\_t), &core0);

pthread\_attr\_setaffinity\_np(&thread1\_attr, sizeof(cpu\_set\_t), &core1);

pthread\_attr\_setaffinity\_np(&thread2\_attr, sizeof(cpu\_set\_t), &core2);

pthread\_attr\_setaffinity\_np(&thread3\_attr, sizeof(cpu\_set\_t), &core3);

pthread\_attr\_setaffinity\_np(&thread4\_attr, sizeof(cpu\_set\_t), &core1);

ret = pthread\_create(&thread0, &thread0\_attr, timer\_thread, NULL);

if (ret != 0) {

printf("Thread creation failed");

}

ret = pthread\_create(&thread1, &thread1\_attr, capture\_thread, NULL);

if (ret != 0) {

printf("Thread creation failed");

}

ret = pthread\_create(&thread2, &thread2\_attr, sharpening\_thread, NULL);

if (ret != 0) {

printf("Thread creation failed");

}

ret = pthread\_create(&thread3, &thread3\_attr, compression\_thread, NULL);

if (ret != 0) {

printf("Thread creation failed");

}

ret = pthread\_create(&thread4, &thread4\_attr, header\_thread, NULL);

if (ret != 0) {

printf("Thread creation failed");

}

ret = pthread\_join(thread0,NULL);

if (ret != 0) {

printf("Thread join failed");

}

ret = pthread\_join(thread1,NULL);

if (ret != 0) {

printf("Thread join failed");

}

ret = pthread\_join(thread2,NULL);

if (ret != 0) {

printf("Thread join failed");

}

ret = pthread\_join(thread3,NULL);

if (ret != 0) {

printf("Thread join failed");

}

ret = pthread\_join(thread4,NULL);

if (ret != 0) {

printf("Thread join failed");

}

/\*int i;

int j=1;

signed int total\_jitter = 0;

time\_t total\_exe = 0;

time\_t match[10];

float avg\_jitter;

float avg\_exe;

for(i=1;i<2000;i++)

{

diff[i] = timestamp[i] - timestamp[i-1];

if(diff[i]==0)

{

match[j] == timestamp[i];

j++;

}

jitter[i] = diff[i]-1;

printf("time : %ld\n",timestamp[i-1]);

printf("diff : %ld\n",diff[i]);

printf("jitter : %d\n",jitter[i]);

}

for(i=1;i<2000;i++)

{

total\_exe = total\_exe+diff[i];

total\_jitter = total\_jitter+jitter[i];

avg\_jitter = total\_jitter/1999;

avg\_exe = total\_exe/1999;

}

/\*for( i=1;i<=2000;i++)

{

printf("%ld\n",match[i]);

printf("%d\n",j);

}

printf("total\_jitter : %d\n",total\_jitter);

printf("avg\_jitter : %f\n",avg\_jitter);

printf("avg\_exe\_time : %f\n",avg\_exe);

\*/

}