REAL TIME DATA PROCESSING

BY

T. LAVANYA SITA **G. SUSHMA REDDY** **K. CHAITANYA**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING,
OSMANIA UNIVERSITY COLLEGE OF ENGINEERING, HYDERABAD

SUPERVISED BY

REPORT OF PROJECT UNDER THE INDUSTRIAL ATTACHMENT PROGRAM

CONTENTS

ABSTRACT

Real-Time Systems are those whose proper operation depends not only on the correctness of the results but also their timeliness. To meet the growing needs of real-time applications, several operating systems have evolved which incorporate features suitable for real-time applications. Our project deals with developing a program that reads an image from an external device (simulated by a periodic timer) and processes and displays it before the arrival of the next image.

This has been implemented on both the Linux platform (which supports soft real-time applications) and the IRIX platform (which has multiprocessing capabilities and provides enhanced graphics and real-time features). We have studied the real-time features available in these operating systems and tried to incorporate them into our program to deal with high data rates without loss of data.

INTRODUCTION

REAL-TIME SYSTEMS: Real-time systems are those that must respond to external stimulus within a short time interval depending on the nature of the problem being solved.

A real-time system must be capable of working in harsh environments like rapidly changing computational loads rich in electromagnetic noise and elementary particle radiation.

Real-time tasks can be classified into two types:

1) By predictability of their arrival

- Periodic tasks are the tasks that are done repetitively.
- Aperiodic tasks are those that occur occasionally

2) By consequences of their not being executed on time

- Critical tasks (hard real-time): This generally includes embedded systems that control devices. Examples of this category are systems that control aircraft, nuclear reactors, chemical power plants, jet engines, etc. Disastrous results ensue when the system does not respond in time.
- Non Critical tasks (soft real time): These are systems where nothing catastrophic happens even when some deadlines are missed. An example of this category is multi-media.

ISSUES IN REAL-TIME SYSTEMS

BOUNDED RESPONSE TIME: A real-time system provides bounded and usually fast response to specific external events, allowing applications to schedule a particular thread to run within a specified time limit after the occurrence of an event.

SCHEDULING: In the conventional time-sharing environment, every process yields its CPU at the end of its time slice. This is not applicable in a real-time environment as it may result in loss of data. So we must ensure, by choosing a suitable scheduling mechanism, that other processes do not preempt these real-time processes.

PRIORITY MANAGEMENT: To ensure that there is no data loss we must see that whenever there is an input available, the input must be read into the input buffer. Therefore the read module is given the highest priority. The processing block must be given the next higher priority so that the data that has been read is processed before output. In other words, the processing is done only when the read block and the process block are idle.

TASK SYNCHRONISATION: Whenever data is available, an indication is sent to the read module to start input. When the read module finishes reading, it sends an indication to the process block to start processing. After processing the data, the process block sends an indication to the output block to output the data in the output buffer.

ASYNCHRONOUS I/O : Using this facility a programmer can queue a read or write request to a device and optionally receive a queued signal when the request is complete. The read( ) or write( ) call will return when the request is queued rather than blocking the process pending completion of the I/O. Optionally, process priority can be used to establish the order in which queued requests are completed.

MEMORY LOCKING A real-time application can avoid the overhead of page fault processing by locking ranges of its text and data into memory.

INTER PROCESS COMMUNICATION: The communication (transfer of data) between the three processes is through shared buffers. The simulator and the read block share a common buffer from which the read block inputs data. The read block and the process block share buffers through which the data is communicated between them. The process block shares a buffer with the output block to send data for output.

PERFORMANCE MEASURES FOR REAL-TIME SYSTEMS

1) Reliability is the probability that the system will not undergo failure over any part of a prescribed interval.

2) Availability is the fraction of time for which the system is up.

3) Throughput is the average number of instructions per unit time that the system can process.

4) Performability of a real-time system is dependent on performance of the process that it controls. A real-time process has several accomplishment levels (levels of performance as seen by the user). Performability is the probability that the computer system will allow each accomplishment level to be met.

5) Hard deadlines should be met in order to avoid catastrophic failures.

PROBLEM DEFINITION:

Data (image) is received from an external device at regular intervals and buffered. Processing is carried out on receiving an entire block of data. Once processed the result of processing is displayed.

The major problem is to ensure that no data is lost. Since there is limited storage, processing should be efficient so that every block of data is processed before the buffers overflow.

DESIGN

The entire job has essentially three aspects.

1. Reading data from an external device.
2. Processing the data read
3. Displaying (output) of this processed data.

These three functions must be incorporated into three different processes running in parallel. This leads to several important issues mentioned below.

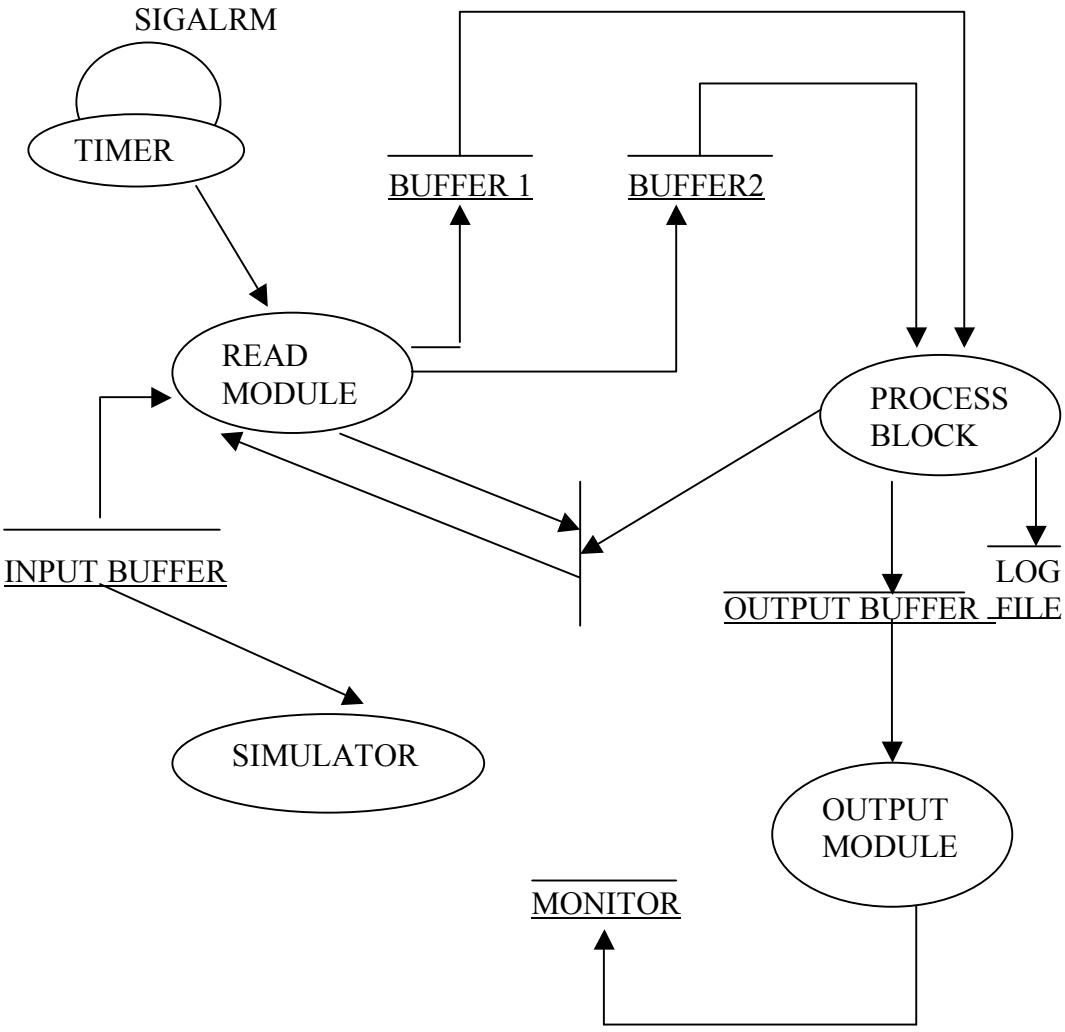
- Scheduling.
- Priority management.
- Task synchronization.
- Inter process communication.
- Bounded response time
- Asynchronous I/O
- Memory Locking.

The modules involved are:

1. Simulator module: It initializes the data which is used by the application.
2. Timer module: This simulates an external device by sending a signal to the read process.
3. Read module: This module reads an image from the input buffer (i.e. a shared memory segment created and initialized by the simulator). It then copies the read data to buffer1/buffer2 and issues a signal to the processing module.
4. Processing module: This module is scheduled to run when it receives a signal from the read module. It processes the received image and sends a signal to the output module.
5. Output module: The output module displays the processed image on receiving a signal from the process block

DATA FLOW DIAGRAMS

DFD (LEVEL 1)



FEATURES OF LINUX

Linux is not designed to support hard realtime applications where a response time of less that 1 milli second is garunteed. It is basically designed to maximize the average case performance instead of the worst case performance. However, there are recently emerged versions of Linux (the rtlinux for instance) that are suitable for hard realtime applications. We confine our study to our conventional Linux (version 6.1).

Scheduling And Priority Management In Linux:

Linux uses a simple priority based scheduling algorithm to choose between the current processes in the system. There are two types of Linux processes viz 1) normal (time sharing) and real time.

Linux supports real time processes and these are scheduled to have a higher priority than all of the other non real- time processes in the system. And also allows scheduler to give each real time process a relative priority. The priority of the real-time processes can be altered using system calls.

Real-time processes may have two types of policy, namely

1) ROUND ROBIN: In Round robin scheduling each runnable real time process is run in turns

2) FIRST-IN FIRST-OUT: In First-in First-out each runnable process is run in the order that it is in the run queue.

Scheduling Classes

SCHED_FIFO: A preemptive priority based scheduling. Each process managed under this scheduling policy possesses the CPU as long as 1) it does not block itself and 2) there comes no other process into a higher priority wait queue. There exist a FIFO queue for each priority level, and every process which becomes runnable is inserted into the queue behind all other processes.

SCHED_RR: A preemptive priority based round robin scheduling strategy with quanta. Each process has a time quantum and the process becomes preempted when the time quantum expires; it is inserted at the end of the queue for the same priority level if it runs longer than the time quantum.

SCHED_OTHER: This policy is based on time-sharing scheduling. Here all the processes have a static priority 0. The priorities between the SCHED_OTHER processes can be set with the nice command.

The system calls used are

int sched_setscheduler(pid_t pid, int policy, const struct sched_param *p);

```
struct sched_param{
        int sched_priority;
        ……..
        };
```

This system call sets the scheduler policy and the static priorities of the process. Priorities between 1&99 can be assigned to processes. To use this system call, the process must have superuser privileges or must have its effective-id equal to its user-id.

Memory Locking

A real-time application can avoid the overhead of page fault processing under IRIX by locking ranges of its text and data into memory.The mlock() and mlockall() system calls can be used for this purpose.

Process Synchronization

Task synchronization can be achieved with signals. The following system calls have been used to deal with signals.

int kill(pid_t pid, int signum);

This is used to send a specified signal with signal number 'signum' to process with id 'pid'.

void (*signal(int signum, void (*handler)(int)))(int);

The signal system call installs a new signal handler for the signal with number signum. The signal handler is set to a handler, which may be a user specified function or a default function.

Interprocess Communication

Shared Memory is used as communication between processes. The system calls used here are as follows.

int shmget(key_t key, int size, int shmflg);

shmget( ) allocates a shared memory segment of size 'size'. It returns an identifier of the shared memory segment associated to the value of the argument 'key'.

char * shmat(int shmid, char* shmaddr, int shmflg);

This function attaches the shared memory segment to the pointer 'shmaddr'.

char* shmdt(char* shmaddr);

This function detaches the shared memory from the pointer 'shmaddr'.

FEATURES 0F IRIX

The IRIX operating system is basically an improvement on the UNIX operating system with several enhancements like advanced graphics and visual computing platform and multiprocessor capability. It provides a rich set of real-time programming features that are collectively referred to as the REACT extensions.

Various components of REACT include: bounded response time, clocks, timers, signals, virtual memory control, asynchronous I/O, threads, scheduling policies, real-time priority band, processor isolation, process binding, and interrupt redirection.

These features of IRIX make it suitable for hard real-time applications unlike Linux.

Discussed below are some of the important features of the IRIX operating system.

Bounded Response Time

A real-time system provides bounded and usually fast response to specific external events, allowing applications to schedule a particular thread to run within a specified time limit after the occurrence of an event.

IRIX guarantees deterministic response of one millisecond on certain uni-processor systems. This real-time strategy guarantees the highest priority thread will execute within one millisecond from the time it was made runnable.

On certain multi-processor machines (OCTANE, Origin200, Origin2000, Onyx2, Origin 3000 series, and Onyx3), the one millisecond bounded response time guarantee is controlled by the systune variable rtcpus. rtcpus represents a threshold at which the scheduler functionality that is required to meet this guarantee is enabled. The threshold is based on the number of physical cpus in the system. If rtcpus is set greater than or equal to the number of physical processors, the bounded response guarantee is enabled. If rtcpus is set below the number of physical processors in the

machine, the bounded response time guarantee is NOT enabled. The default value for rtcpus is 0, which means that by default, the guarantee is not enabled. In order to enable the guarantee, rtcpus must be set equal to or greater than the number of cpus in the system

Timers

Timer expiration interrupts are dispatched to IRIX interrupt threads for handling. The priority at which these threads are scheduled is determined by the scheduling policy and priority of the thread which sets the timer: If the thread setting the timer is running with a timeshare scheduling policy, then the associated interrupt thread will be scheduled at real-time priority one.

If the thread setting the timer is running with a real-time scheduling policy, then the priority of the associated interrupt thread will be the priority of the setting thread plus one. Priority 255, being the maximum real-time band priority, is an exception. If the thread setting the timer is running at priority 255, then the interrupt thread will also be scheduled at priority 255. Hence, real-time applications depending on system services shouldn't use priority 255. Once the timer expires, the interrupt thread will be scheduled ahead of the thread which set the timer.

The system calls used to invoke the timer are

int timer_create();

int timer_settime (timer_t timerid, int flags,const struct itimerspec *value,
struct itimerspec *ovalue);

Signals

IRIX implements queued signals which provide signal priorities and queuing of signals such that exactly as many signals are received as were sent.

Memory Locking

The mlockall system call can be used to lock down a process's entire virtual address space. Since it is not always desirable to lock down the entire virtual address space, IRIX provides the following system calls to lock and unlock a specified range of addresses in memory: pin/munpin and mlock/munlock. The major difference between the two sets is that mpin/munpin maintains a per page lock counter and mlock/munlock does not.

System calls :

int mlockall(int flags);
int munlockall(void);

Asynchronous I/O

IRIX implements the interface to asynchronous I/O. By using this facility one can queue a read or write request to a device and optionally receive a queued signal when the request completes. The read( ) or write( ) call will return when the request is queued rather than block the process pending completion of the I/O. Optionally, process priority can be used to establish the order in which queued requests are completed.

System calls:

int aio_write(aiocb_t *aiocbp);

int aio_write64(aiocb64_t *aiocbp);

```
struct aiocb_t
{
        int aio_nbytes;
        int aio_fildes;
        char *aio_buf;
        int aio_sigevent;
        ……
}
```

Threads

Threads (pthreads) in IRIX support both process and system scope threads. System scope threads enable pthread applications to obtain predictable scheduling behavior on a system level by using the kernel scheduler directly, bypassing the user-level pthread scheduler.

System calls:

int pthread_create(pthread_t *thread, pthread_attr_t *attr,
void *(*start)(void *), void *arg);

int pthread_kill(pthread_t thread, int sig);

Real-time Scheduling

IRIX supports the real-time scheduler interfaces, including sched_setscheduler and sched_setparam.

These interfaces provide privileged applications with the control necessary for managing the cycles of the system processor(s). Real-time scheduling policies, such as round-robin and first-in-first-out, may be selected along with a real-time priority.

Real-time Priority Band

A real-time thread may select one of a range of 256 priorities (0-255) in the real-time priority band, using interfaces sched_setparam( ) or sched_setscheduler( ). The higher the numeric value of the priority, the more important the thread.

Many soft real-time applications simply need to execute ahead of timeshare applications, in which case priority range 0 through and including 89 is best suited. Since timeshare applications are not priority scheduled, a thread running at the lowest real-time priority (0) will still execute ahead of all timeshare applications. Hard real-time applications may use priorities 240 through and including 254 for the most deterministic behavior and the lowest latencies.

Device Driver Interrupt Thread Priorities

As of IRIX 6.4, device drivers employ interrupt threads to handle device interrupts. Interrupt threads have default priorities in the range 200 through and including 239.

Processor Control

Using the sysmp( ) call or the mpadmin and runon commands a programmer may control the distribution of processes among the processors in a real-time system. It is possible to bind a particular process onto a processor and conversely, it is possible to restrict a processor to only run those processes that are explicitly bound to it.

IMPLEMENTATION

IMPLEMENTATION OF MAJOR DESIGN ISSUES :

- *SCHEDULING & PRIORITY MANAGEMENT*: -The scheduling class that is used to schedule the various processes in this application is SCHED_FIFO, which ensures no data loss. Each process is associated with a specific priority according to its requirements. The scheduling class and the priorities within this class are set for each module using the 'sched_setparam( )' system call.

- *TASK SYNCHRONIZATION*: Task synchronization, one of the major design issues, is achieved through SIGNALS. The following are the various signals that enable task synchronization in this application.
    1. SIGALRM: This signal is issued every second to indicate the arrival of input data and is captured by the read process which then reads the supplied input.
    2. SIGQUIT: On completion of reading the entire data for an image, the read process sends a signal (SIGQUIT) to the processing process.
    3. SIGPIPE: The signal SIGPIPE is generated by the processing process to the output process after the processing process has finished with its processing.

- *INTERPROCESS COMMUNICATION:* This aspect of design is implemented using the concept of shared memory. The various shared memory segments used in this application are designated as follows:
    1. Input buffer, between simulator and read process.
    2. Buffer1, between read process and processing process.
    3. Buffer2, between read process and processing process.
    4. Output buffer, between processing and output processes.

- *RESPONSE TIME :* The response time of the application can be improved by incorporating the following features:
  1. Bounded response time: This facility is available in IRIX OS. This real time strategy guarantees the highest priority thread will execute within a specified time from the time it was made runnable.
  2. Asynchronous I/O: This is possible in IRIX. aio_read( ) and aio_write( ) system calls are used to achieve asynchronous I/O
  3. Memory locking: This is implemented using the system calls, mlockall( ) and munlock( ).

MODULE SPECIFICATION

SIMULATOR: This is run before all the other processes are invoked. Its purpose is to create a shared memory segment (the input buffer) and initialize it with a fixed pattern. This pattern can be described as follows. Let the dimensions of the image being displayed be 2N * 2N. The input buffer has a size of 6 * N * N bytes and every third byte represents red color and the remaining bytes represent blue.

TIMER: It simulates an external device. It indicates the availability of data to the read process at regular intervals. It interrupts the currently running process by issuing a signal SIGALRM.

READ MODULE: This module is assigned the highest priority (priority 3). It is activated on receiving SIGALRM from the timer.

- INPUT: It reads data (an entire image) at regular intervals from the input buffer. It reads N* N bytes (size of the image) in a circular fashion, starting from the last byte unaccessed in the previous reading (that is, if it encounters the end of the buffer, it proceeds to the beginning of the buffer and continues from there).

- OUTPUT: The data read is written to BUFFER1 and BUFFER2 alternately. Once the buffer is filled, it sends SIGQUIT to the processing module and goes into a pause state.

PROCESSING MODULE: This module is assigned second highest priority (priority 2). It is scheduled when it receives a SIGQUIT signal and when the read module is in pause state (waiting for SIGALRM signal).

- INPUT: It reads data from BUFFER1 and BUFFER2 alternately.
- PROCESSING:

1) Histogram: It counts the frequency of occurrence of various colors (red and blue) and stores it in a log file.
2) Stretching: It widens the range of pixel intensities. Pixel intensities are multiplied by a factor (F) which is determined as follows.

   F = (desired range/current range)
3) Inversing: Pixel intensities are complemented.

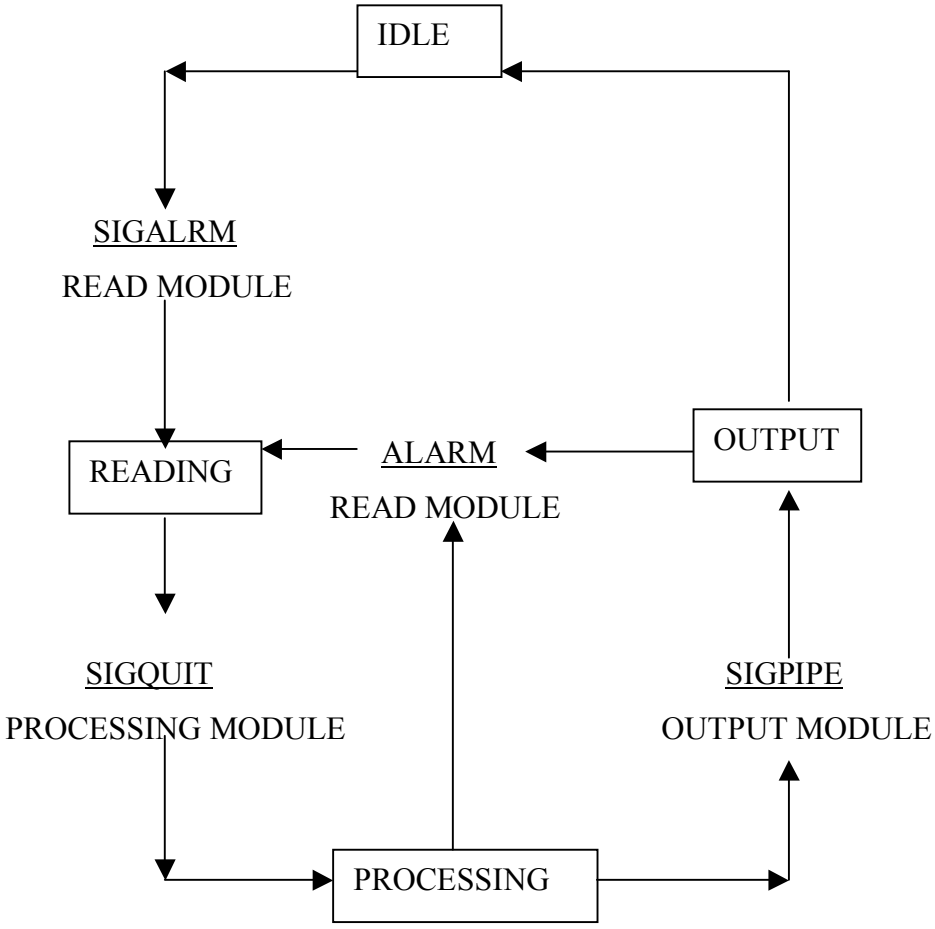
Finally a signal (SIGQUIT) is sent to the output process.

- OUTPUT: The frequency of occurrence of various colors is written onto a log file and the contents of BUFFER1 / BUFFER2 are copied onto the OUPUT BUFFER.

OUTPUT MODULE: It is assigned the least priority (priority 1). The output module is activated on receiving SIGPIPE from the processing module and when the read and the processing modules are in a pause state. It initializes the GUI and creates a new top-level window.

- INPUT: It reads the image from the OUTPUT BUFFER.
- PROCESSING: Each time, the image is copied onto the pixmap array that is associated with a widget.
- OUTPUT: The widget corresponding to the OUTPUT BUFFER is displayed on the screen.

STATE TRANSITION DIAGRAM



TESTING

- It must be ensured that no data is lost at the required data rate. To simulate this data rate, an alarm is set so that it issues a signal to the read process every second and the size of the image is equalized with the numeric value of the required data rate. The system is held under scrutiny to check for data loss.
- The efficient operation of the program even in the presence of other processes shall be tested. To ensure this, several dummy processes are made to run in the background and the output is held under scrutiny to check for data loss.

OBSERVATIONS

LINUX

THE MAXIMUM DATA RATE FOR DIFFERENT PROCESSING ACTIVITIES
(The results shown below were recorded with the timer value set at 1 sec.)

IMAGE SIZE	MAXIMUM DATA RATE	PROCESSING INVOLVED
1088 x 1088	1.13 Mbytes/sec	Stretching, inverting, recording the frequency count and displaying the image
3840 x 3840	14.03 Mbytes/sec	Stretching, inverting, recording the frequency count

IRIX

MAXIMUM DATA RATE FOR DIFFERENT IMAGE SIZES
(The processing done is stretching, inversion and recording the frequency count)

IMAGE SIZE	MAXIMUM DATA RATE
1024 X 1024	26 Mbytes/sec
1088 X 1088	24 Mbytes/sec
2048 X 2048	20 Mbytes/sec
3840 X 3840	17 Mbytes/sec

CONCLUSIONS

1. We have studied the real-time features of various operating systems.
2. We have experimented with these features and observed how the response time can be improved.
3. We have studied how the response time varies with each of these features.
4. Although IRIX has more features to accommodate real-time applications, we found that Linux can produce reasonable results in a soft real-time application like image processing. In other words, in the worst case, there may be some data loss due to lack a deterministic response time during context switching. But the average performance for the image size chosen is not far behind the IRIX operating system.

BIBLIOGRAPHY

1. Walter S. Heath: Real-Time Software Techniques

2. C. M. Krishna & Kang G.Shin : Real-Time Systems

3. Eric Harlow: Developing Linux Applications with GDK & GTK+

4. Linux Manual pages

5. IRIX Manual pages.