Stabilizing Execution Time of User Processes by Bottom Half Scheduling in Linux

Kyong Jo Jung, Seok Gan Jung, Chanik Park
System Software Laboratory
Pohang University of Science and Technology
Kyungbuk, Republic of Korea
{braiden,javamaze,cipark}@postech.ac.kr

Abstract

The CPU time allocated to user processes is rendered inaccurate by an unexpectedly and frequently occurring interrupt and a bottom half that consumes most interrupt processing time. Additionally, when the time consumed in the kernel mode greatly fluctuates with interrupt processing, the scheduler cannot distribute CPU time to user processes normally. This problem can dramatically distort the stable execution time of user processes. In addition, such time-sensitive applications as multimedia players cannot provide consistent quality. To overcome this stolen-time problem, we propose a bottom half scheduling approach that dynamically restricts the maximum time consumed by bottom halves. In this paper, we implement our proposed scheme in Linux 2.4. In addition, we show that the fluctuation of CPU time allocated to user processes by stolen-time can be shrunk with our proposed scheme by means of experiments using a multimedia application.

1. Introduction

With the rapid growth of hardware technologies, it is now commonplace that real-time applications such as packet routing and voice recognition software run on general purpose operating systems. For these reasons, a number of studies have been investigated to support the time constraints essential to soft real-time applications on commodity operating systems, such as Linux.

On a general operating system, the stolen-time caused by the interrupt processing time is one of the biggest problems to be addressed to support time-sensitive applications. This problem can dramatically distort the stable execution time of user processes. Consequently, a time-sensitive application such as a multimedia player cannot provide consistent quality. Specifically, the interrupt handling that is mainly managed in the bottom halves can steal the execution time from a currently executing process. The Linux 2.4 divides interrupt processing into two phases. One is the "Top Half," which executes critical operations such as acknowledging an interrupt to the PIC(Programmable Interrupt Controller) immediately, and the other is the "Bottom Half," which executes the remaining routine with all interrupts enabled. As an interrupt signal occurs, its corresponding Interrupt Service Routine (ISR) handles it. An interrupt signal invokes its associated Interrupt Service Routine. Since an ISR disables all hardware interrupts during its execution, it should include only the essential part of its entire service routine, thereby keeping the interrupt-disabled period as short as possible. After processing a critical operation, the ISR activates the bottom half. The bottom halves activated by ISR are immediately executed by the dispatcher before switching to the user mode. For example, in heavy network traffic, the number of incoming interrupt signals is frequent and the execution time of bottom halves is highly variable. This unpredictable variation of bottom halves influences the execution time of a currently executing user process. Thus, the execution time of the user processes should fluctuate. In addition, it can lead to scheduling anomalies where the scheduler does not allocate the available CPU time to user processes. In extreme cases, if the in-coming interrupt rate is high enough to cause the system to spend all of its time handling interrupts, nothing else will occur, and the system through-

Proceedings of the 12th 16th Euromicro Conference on Real-Time Systems (ECRTS'04)
1068-3070/04 $20.00 © 2004 **IEEE**

put will drop to zero. We call this condition the *receive livelock problem*: the system is not deadlocked, but it makes no progress in any of its tasks.[4][8]

To show the impact of stolen-time by the bottom halves, we performed some experiments using mplayer[12]. Our experiments employ a sample movie file that is encoded in MPEG-4 at the rate of 25 frames per second and performed under heavy network traffic. The inter-frame delay is referred to as the time difference required to decode two adjacent frames in the movie file. An inter-frame delay of 40 msec is expected within a sufficient CPU resource. However, in our experiment, since the execution time of mplayer was stolen by the bottom halves, the average inter-frame delay is about 100 msec and the variation range is high. In order to test how bottom half scheduling can solve this problem, the experiment was repeated by restricting the maximum processing time of the bottom half in the average time of the first experiment. As shown by the second plot in Figure 1, the average of inter-frame delay is approximately 40 msec. In this experiment, it is important that the average delay as well as the fluctuation of the inter-frame shrinks by restricting the maximum processing time of the bottom halves, although the average time consumed by the bottom halves is almost identical in the two experiments. This experimental result shows that a scheduler cannot distribute CPU time to user processes normally when the time consumed in kernel mode is highly variable. We define the *scheduling anomaly* as this state and describe it experimentally in section 4.

In this paper, to overcome the above-mentioned problem, we propose a bottom half scheduling method that keeps the threshold value obtained from the previous processing time for the bottom halves and schedules the handling time of the bottom halves based on the estimated threshold time. In addition, the experimental result performed on Linux 2.4 shows that our scheme can shrink extensive variation of the execution time of bottom halves and prevent process scheduling anomalies with the proposed bottom half scheduling method that mitigates the wide difference between the allocated CPU time and the available CPU time for a process.

Much research has been done on the stolen time problem caused by unpredictably occurring interrupts and the processing time for the bottom half. In [1][2][5] and [6], previous authors proposed approaches that compensate

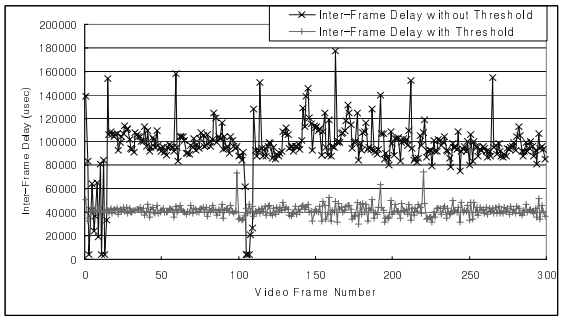


Figure 1. Impact of the stolen-time by bottom halves

for stolen time to user process on scheduling scheme based on *CPU reservation*. In [4][7], a method that controls the bottom halves with user processes by modifying network subsystem were proposed. These approaches can be applied to a *reservation based scheduler* or a modified network subsystem. In this paper, we propose an enhanced approach without modifying the scheduling mechanism or the network subsystem architecture.

The remainder of this paper is organized as follows. In section 2, we describe our proposed scheme and algorithm for bottom half scheduling. In section 3, we show the impact of the bottom half scheduler using the proposed algorithm through the experimental results. Finally, we conclude this paper in section 4.

2. Proposed Scheme

2.1. Architecture of Interrupt Handling in Linux

In Linux 2.4, the architecture of interrupt handling is divided into two phases. The first is the ISR or interrupt handler. The second is the bottom halves.

As shown in Figure 2, the in-coming interrupt signal is passed to ISR in terms of its category. Because ISR disables other interrupt signals during the processing interrupt, ISR handles only critical operations and activates the bottom halves. Afterwards, the bottom half activated by ISR is executed by the dispatcher and copes with deferred operations, such as receiving or sending data through the device.

Linux 2.4 has three types of deferrable functions (or bottom halves) by means of its function : *softirq, tasklet, and bottom*

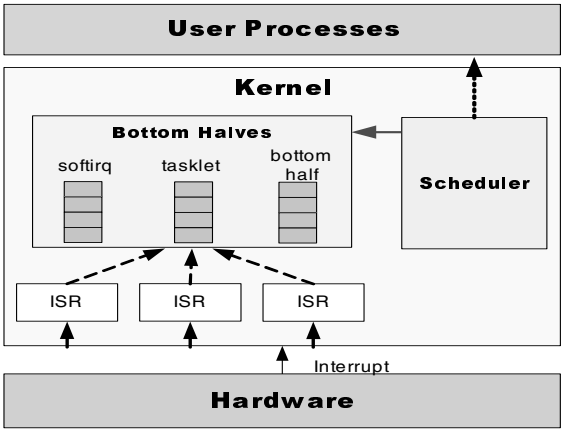


Figure 2. Structure of Bottom Halves in Linux 2.4

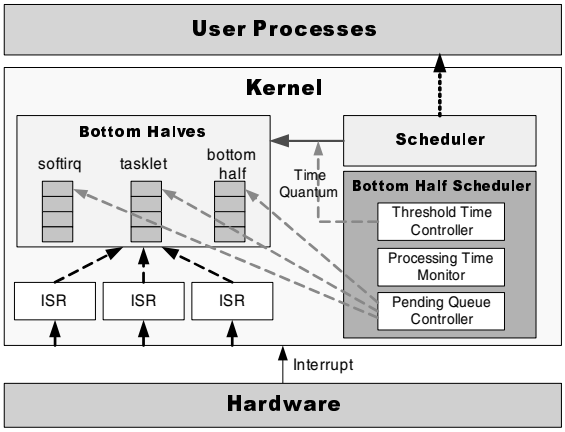


Figure 3. Architecture of Bottom Half Scheduler

half. In this paper, the "bottom half" denotes all types of these deferrable functions. A device driver or a kernel service uses one of these three kinds of bottom halves according to its purpose.

A kernel thread called *ksoftirq_CPUn* exists in each CPU in Linux 2.4. It determines if the pending bottom halves exist before switching to the user mode, and executes them.

By means of a bottom half processing mechanism, the activated bottom half copes with all deferred operations at the executed time. In the worst case, it might consume the entire time of a *tick*(in Linux, 10 msec). Since the amount of deferred operation is unpredictable and a range of variation is wide, the CPU time allocated to user process suffer from high variation and the process scheduler cannot schedule normally.

2.2. Proposed New Architecture

As seen in Figure 3, the proposed Bottom Half Scheduler contains three types of modules.

- *Processing Time Monitor*
- *Threshold Time Controller*
- *Pending Queue Controller*

The *Processing Time Monitor* measures the CPU time consumed by each bottom half. To prevent the bottom halves from consuming too much or intensely fluctuating the CPU time, the *Threshold Time Controller* computes the threshold time that the bottom halves can consume. A *Pending Queue Controller* removes the remaining pending jobs when the total execution time of the bottom halves reaches the threshold time established by the *Threshold Time Controller*. Afterwards, it is reinserted at the head of the pending queue of the bottom halves when the kernel activates the bottom halves the next time.

The time consumed by the bottom halves fluctuates in spite of a fixed workload(see Figure 5 (a) in section 4 for the result). Consequently, the *Bottom Half Scheduler* defers execution of the bottom halves until the next time activated by *ksoftirq_CPUn* kernel thread when the time consumed by them reaches the established threshold time. Therefore, the Bottom Half Scheduler prevents an intensive increment of consumed time. In this manner, excess beyond the established threshold time is processed when the jobs of the bottom halves is below the threshold time. Thus, the variation of CPU time consumed by the bottom halves can be reduced.

2.3. Design of Threshold Time Controller

As mentioned in the previous section, the variation of processing time consumed by the bottom halves disturbs the scheduler in a stable distributing CPU time to user processes. A purpose of the *Threshold Time Controller* is to restrict the execution time of the bottom halves within the average time. The Bottom halves not-yet-disposed by the threshold time are postponed in the next activated time so that the time consumed by the bottom halves is below the threshold time.

Notation	Description
$T_{thr}(n)$	In n^{th} sampling time, threshold time of bottom halves
G	Control gain
${}^{+}\Delta T_i^{thr}$	In i^{th} sampling time, expected processing time variation of bottom halves not-yet-disposed by threshold time
${}^{-}\Delta T_i^{thr}$	In i^{th} sampling time, time variation consumed by bottom halves
T_i^{pro}	In i^{th} sampling time, time consumed by bottom halves
T_i^{rmn}	In i^{th} sampling time, average expected processing time of bottom halves not-yet-disposed by threshold time
N_i^{pro}	In i^{th} sampling time, the number of processed job
N_i^{rmn}	In i^{th} sampling time, the number of bottom halves not-yet-disposed by threshold time
AW	size of average time window

Table 1. Notations

Algorithm 1: The bottom half threshold time assignment scheme based on the processing time of the bottom halves not-yet-disposed by threshold time and the processed bottom halves

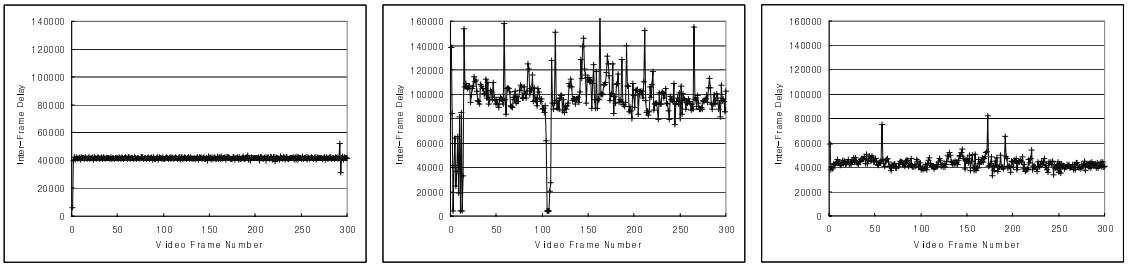
input : $N_i^{pro}, T_i^{pro}, N_i^{rmn}$
output : $T_{thr}(i)$
begin
  if $(N_i^{pro} > 0)$ **then**
    $T_i^{rmn} \leftarrow N_i^{rmn} \times (T_i^{pro} \,/\, N_i^{pro})$
  else
    $T_i^{rmn} \leftarrow N_i^{rmn} \times$ Default processing time per packet
  end
  ${}^{+}\Delta T_i^{thr} \leftarrow (\sum_{AW} T_{rmn} \,/\, AW) - T_i^{rmn}$
  ${}^{-}\Delta T_i^{thr} \leftarrow (\sum_{AW} T_{pro} \,/\, AW) - T_i^{pro}$
  $T_{thr}(i) \leftarrow T_{thr}(n-1) + G({}^{+}\Delta T_i^{thr} + {}^{-}\Delta T_i^{thr})$
end

Table 1 describes the notations used by the threshold control algorithm. Using the notations of Table 1, the algorithm to estimate the maximum time that can be consumed by the bottom halves is described as algorithm 1.

If the threshold time is too low, the number of bottom halves not-yet-disposed by the threshold time is continuously increased. On the other hand, if the threshold time is too high, the time consumed by the bottom halves fluctuates as the one without bottom half scheduling. Thus, the `Threshold Time Controller` adjusts the threshold time so that a variation in both the consumed time and the expected processing time of the bottom halves not-yet-disposed by the threshold time to zero. When the threshold time of ${}^{+}\Delta T_i^{thr}$ and ${}^{-}\Delta T_i^{thr}$ is zero we keep the time consumed by the bottom halves controlled without dropped jobs and high variation.

3. Performance Evaluation

In this section, we describe our experimental result using our proposed scheme on Linux 2.4. The implementation of the `Bottom Half Scheduler` was performed based on kernel version 2.4.20. Since a network device is one of the most high-bandwidth devices, we implemented



(a) Inter-frame delay on no traffic

(b) Inter-frame delay without Bottom Half Scheduler under high network traffic

(c) Inter-frame delay with Bottom Half Scheduler under high network traffic

Figure 4. Comparison with inter-frame delay : Whereas, in Figure (a), average inter-frame delay is nearly 40 msec. In Figure (b), it is 96.2 msec under high network traffic. In Figure (c), it is 43.2 msec by preventing fluctuation of time consumed by the bottom half.

		no network load	without scheduling	with scheduling
Decoding Time		12 sec	29.03 sec	14.52 sec
Network Throughput		n/a	10.006 MB/sec	10.046 MB/sec
Average Packet Processing Time per msec		n/a	454.489 usec	497.697 usec
CPU Usage(%)	User Domain	10.9	7.5	24.8
	System Domain	1.3	92.4	75.1

Table 2. Comparison with total decoding time, network throughput, average packet processing time and CPU usage

and evaluated it only for network packet processing. The *Processing Time Monitor* and *Threshold Time Controller* is loaded as a dynamic kernel module. In addition, a partial kernel code of packet processing is modified to keep the packet from being processed when the time consumed by the bottom half exceeds the threshold time.

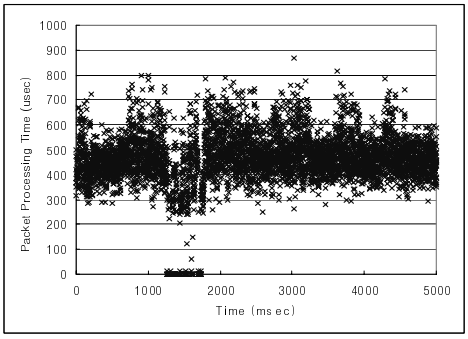
3.1. Experimental Setup

The experimental setup consists of two 2GHz Pentium-4 machines, each configured with 512MB of RAM and connected through a 100Mbps Ethernet. For the network load, one machine sent UDP packets of 50 bytes using the ttcp[11] benchmark tool.

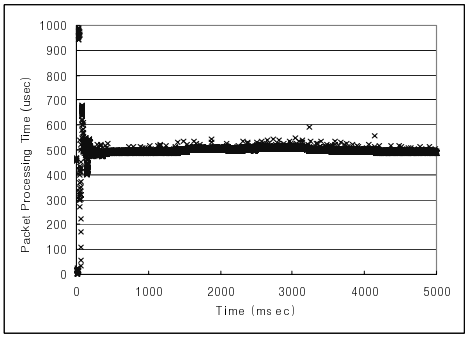
We performed experiments on a multimedia application, mplayer[12]. We chose an inter-frame delay as the latency metric for mplayer. For measuring the inter-frame delay, we modified the source code of the mplayer. The mplayer is an audio/video player that can handle several media formats. The mplayer synchronizes audio and video streams by using time-stamps that are associated with frames. The audio card is used as a timing source. When a video frame is decoded, its time-stamp is compared with the time-stamp of the currently playing audio frame. If the video time-stamp is smaller than the audio time-stamp, the video frame is late and the video is immediately displayed. Otherwise, the mplayer sleeps until the time difference between the video and audio time-stamps and then displays the video.

3.2. Effects of Bottom Half Scheduling

If time consumed by the bottom halves strongly fluctuates, the inter-frame delay is influenced by variation that is caused by the stolen-time. Thus, we presented the effect of our scheme by measur-



(a) Packet processing time consumed by bottom halves without scheduling



(b) Packet processing time consumed by bottom halves with scheduling

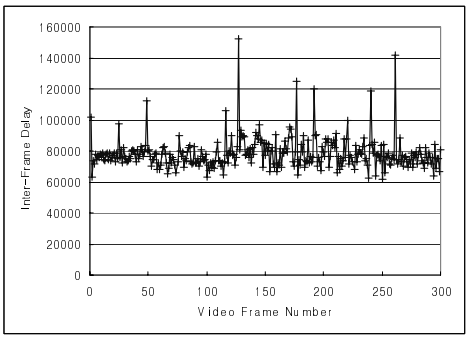
Figure 5. Comparison with time consumed by bottom half per msec : Without Bottom Half Scheduler, in Figure (a), range of consumed time is from nearly 200 usec to maximum 850 usec. With Bottom Half Scheduler, in Figure (b), the range is between 450 usec and 550 usec after starting phase(until approximately 300 msec).

ing the inter-frame delay of mplayer.

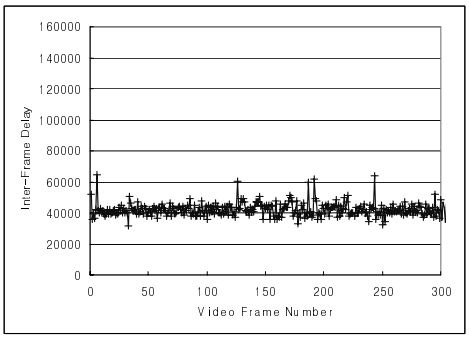
In our experiment, we chose sample video data that was encoded in MPEG-4 at a rate of 25 frames per second. If the system resource is sufficient, the decoding interval should be 40 msec. We evaluated inter-frame delay through three types of experiments. First, we evaluated inter-frame delay without network traffic. As expected, Figure 4(a) shows that inter-frame delay is approximately 40 msec. In our second experiment, the inter-frame delay was highly increased by packet processing time that was consumed by bottom halves. Figure 4(b) shows that its maximum inter-frame delay is about 160 msec and the average is 96.2 msec.

Next, we evaluated the inter-frame delay with the `Bottom Half Scheduler` under high network traffic. We used a value of 0.25 for gain G and 100 for average time window AW. These values were derived from a heuristic approach. In the third experiment, Figure 4(c) shows that the maximum inter-frame delay is about 60 msec and the average is 43.2 msec. Figure 5 shows the packet processing time consumed by the bottom halves. In Figure 5, the X axis is the sampling time whose interval is 1 msec, and the Y axis is the packet processing time (usec) consumed by the bottom halves per 1 msec. Whereas Figure 5(a) shows that the range of the processing time is wide from 250 usec to maximum 850 usec per msec, Figure 5(b) shows that the variation range is much narrower from 480 usec to 580 usec, although it is highly variable in the starting phase until approximately 300 msec.

As shown in Table 2, the average time consumed by the bottom halves is similar in the two experiments. However, the CPU usage rate of the user domain in experiment with `Bottom Half Scheduler` is approximately three times more than it without the bottom half scheduling. We estimated that scheduling anomalies cause this result, as noted in section 1. In other words, the process scheduler cannot schedule user processes exactly under the high jitter of execution time in the kernel mode, although the average time consumed by the bottom halves is similar. In Figure 5(a), if a scheduler distributes the available CPU time to user processes exactly when the time consumed by the bottom halves is below the average consumed time, only the jitter of inter-frame delay should be shrunk in Figure 4(c) compared with Figure 4(b). However, because the scheduler cannot distribute the available CPU time to user processes under the highly variable processing time in the kernel mode, the CPU usage of user domain is decreased below the amount required by application(see the 5th row of Table 2). Consequently, the average time as well as jitter of the inter-frame delay shrinks in Figure 4(c).



(a) Inter-frame delay without Bottom Half Scheduler playing through NFS



(b) Inter-frame delay with Bottom Half Scheduler playing through NFS

Figure 6. Comparison with inter-frame delay under high network traffic playing through NFS : Without Bottom Half Scheduler, in Figure(a), range of the inter-frame delay is nearly between 60msec and 150msec. With Bottom Half Scheduler, in Figure(b), it is nearly 30 msec and 60 msec.

3.3. Impact of Bottom Half Scheduling on Network Processing

In this section, we show the impact of the proposed bottom half scheduling on network processing. To show the impact on network processing, we performed an additional experiment. For the experiment, we used another machine that contains sample video data. A machine that plays video data is connected to the machine that contains video data through NFS(Network File System). The other machine sent UDP packets for network load, as in the previous experiment.

As shown in Figure 6, the experimental result is similar to the result of the previous experiment. Although the Bottom Half Scheduler defers packet processing when the time consumed by the bottom halves reaches the established threshold time, the average time of packet processing is almost identical compared with the case without the Bottom Half Scheduler. In other words, the proposed Bottom Half Scheduler does not give rise to delays in network processing since the `Threshold Time Controller` adjusts the threshold time to the average time of packet processing.

4. Conclusion and Future Work

The stolen-time problem is that the execution time allocated to a user process is stolen by the interrupt processing time handled mainly by the bottom halves. Since the time consumed by the bottom halves is unpredictable and strongly fluctuates, the stolen-time should distort the stable execution time of user processes. Thus, time-sensitive applications such as multimedia player cannot provide consistent quality. In addition, we defined a scheduling anomaly as a state where the scheduler cannot distribute the available CPU time to user processes normally because of a high variation of processing time in the kernel mode. We showed that, due to highly fluctuating stolen-time, a scheduler cannot distribute CPU to user processes in a stable manner. To overcome the problem, we proposed a bottom half scheduling scheme that estimates the threshold time in terms of the previous time consumed by bottom halves and controls handling time of bottom halves based on the estimated threshold time. Experimental results conducted in Linux 2.4 revealed that our scheme can prevent a high variation of time consumed by the bottom halves and stabilize the execution time of a user process. Also, we show that the proposed mechanism does not cause a delay in network processing despite deferring packet processing when the time consumed by the bottom halves reaches the established threshold time.

While we proposed an algorithm and scheme to schedule the bottom halves, it was limited by a part of network packet processing. We plan to unify all the bottom halves to schedule in future work. We are also interested in investigating a more sophisticated algorithm to adapt to workload variations.

References

[1] Luca Abeni, Luigi Palopoli, Giuseppe Lipari, and Jonathan Walpole. Analysis of a reservation-based feedback scheduler. Proceedings of the IEEE Real-Time Systems Symposium, 2002

[2] John Regehr and John A. Stankovic. Augmented CPU reservations: Towards predictable execution on general-Purpose operating systems. Proceedings of the Real-Time Technology and Applications Symposium, 2001

[3] J. Mogul, K. Ramakrishnan. Eliminating receive livelock in an nterrupt-driven kernel. ACM Transactions on Computer Systems, 1997

[4] Kevin Jeffay, F. Donelson Smith, Arun Moorthy, and James Anderson. Proportional share scheduling of operating system services for real-time applications. Proceedings of the IEEE Real-Time Systems Symposium, 1998

[5] Luca Abeni and Giuseppe Lipari. Compensating for interrupt process times in real-time multimedia systems. Real-Time Linux Workshop Work in Progress, 2001

[6] Luca Abeni. Coping with interrupt execution time in RT kernels: A non-intrusive approach. IEEE Real-Time Systems Symposium Work in Progress, 2001

[7] Peter Druschel and Gaurav Banga. Lazy Receiver Processing (LRP): A network subsystem architecture for server systems. Proceedings of the USENIX Symposium on Operating System Design and Implementation, 1996

[8] A. Indiresan, A. Mehra, and K.G.Shin. Receive livelock elimination via dynamic interrupt rate control. Technical report, University of Michigan, 1997

[9] G.F.Franklin, J.D.Powell and M.L.Workman. Digital control of dynamic systems(3rd Ed.). Addison-Wesley, 1998.

[10] C.Lu, J.A.Stankovic, G.Tao, and S.H.Son. Feedback control real-time scheduling: Framework, modeling, and algorithms. Real-Time Systems journal, Special Issue on Control-Theoretical Approaches to Real-Time Computing, 2002

[11] Test TCP(TTCP) benchmark tool. http://www.ccci.com/tools/ttcp/

[12] Mplayer - Movie player for Linux. http://www.mplayerhq.hu