# A design and evaluation of Two-way Request Latency Test Program

| | | | | | |
|---|---|---|---|---|---|
| Rev | Date | Description | | | |
| | Issue | | Author | Check | App |
| Rev | Date | Description | | | |
| | Issue | | Author | Check | App |
| Rev | Date 2010/11/30 | Description | | | |
| 0 | Issue Professional Service | | Author T.Oshima | Check | App |
| Title | A design and evaluation of Two-way Request Latency Test Program | | No Confidential | | Revision |
| | | | | | Page 1 |

## Introduction.

This paper describes the design of Two-way Request Latency Test Program which evaluates real-time OS synthetically.

## Purpose.

The purpose of this program is not the fragmentary performance of real-time OS, but is an actual quality assessment, and does not evaluate fragmentary performance.
Conventionally, speed of response (Response Time) exists in what the maker has released as a numerical value which evaluates real-time OS.

**Timer Example1**

The result of having executed the program which measures the starting interval of the timer handler for 1ms for about 10 seconds, and having evaluated the real-time performance by Red Hat software and RedHawk on the same hardware (digital logic). (In order to change a priority, it performs by root account)

- RedHatEL 5.4 kernel 2.6.18-164 with PREEMPT_RT
- RedHawk 5.4 Kernel 2.6.31.6

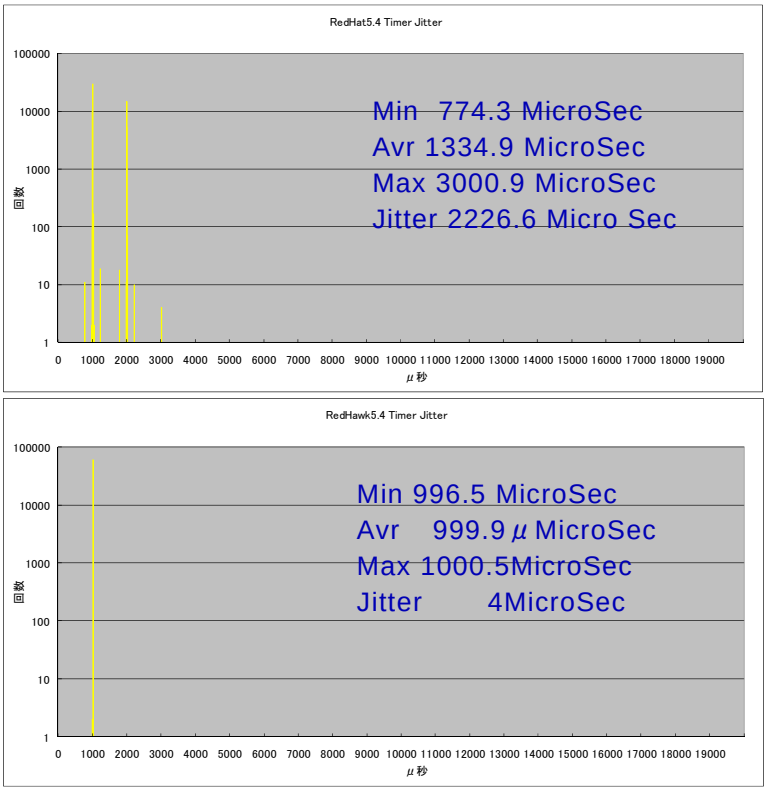



Figure 1 Timer Example1

**Timer Example2**

The result of having executed the program which measures the starting interval of a timer handler by Ubuntu and RedHawk, respectively for about 30 minutes, and having evaluated the real-time performance on the same hardware (Panasonic). (In order to change a priority, it performs by root account)

- Ubuntsu 9.04 Kernel 2.6.29.6 With PREEMPT_RT
- RedHawk 5.4 Kernel 2.6.31.6

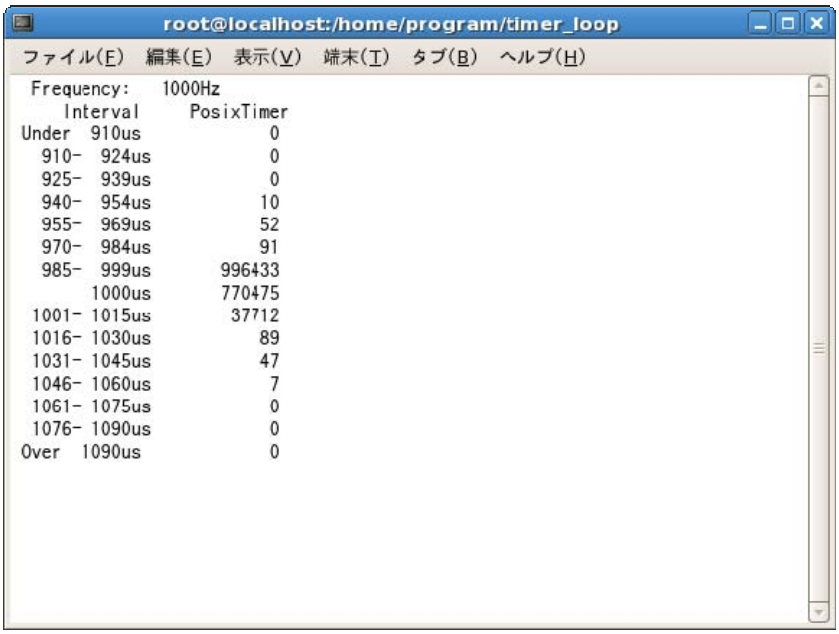




Figure 2 Timer Example2

**A design and evaluation of Two-way Request Latency Test Program**

When evaluating the performance of real-time OS synthetically, there is response performance of TCP/IP as an important point.

This "Two-way Request Latency Test Program" (henceforth, TRLTP) designed as follows based on this point.

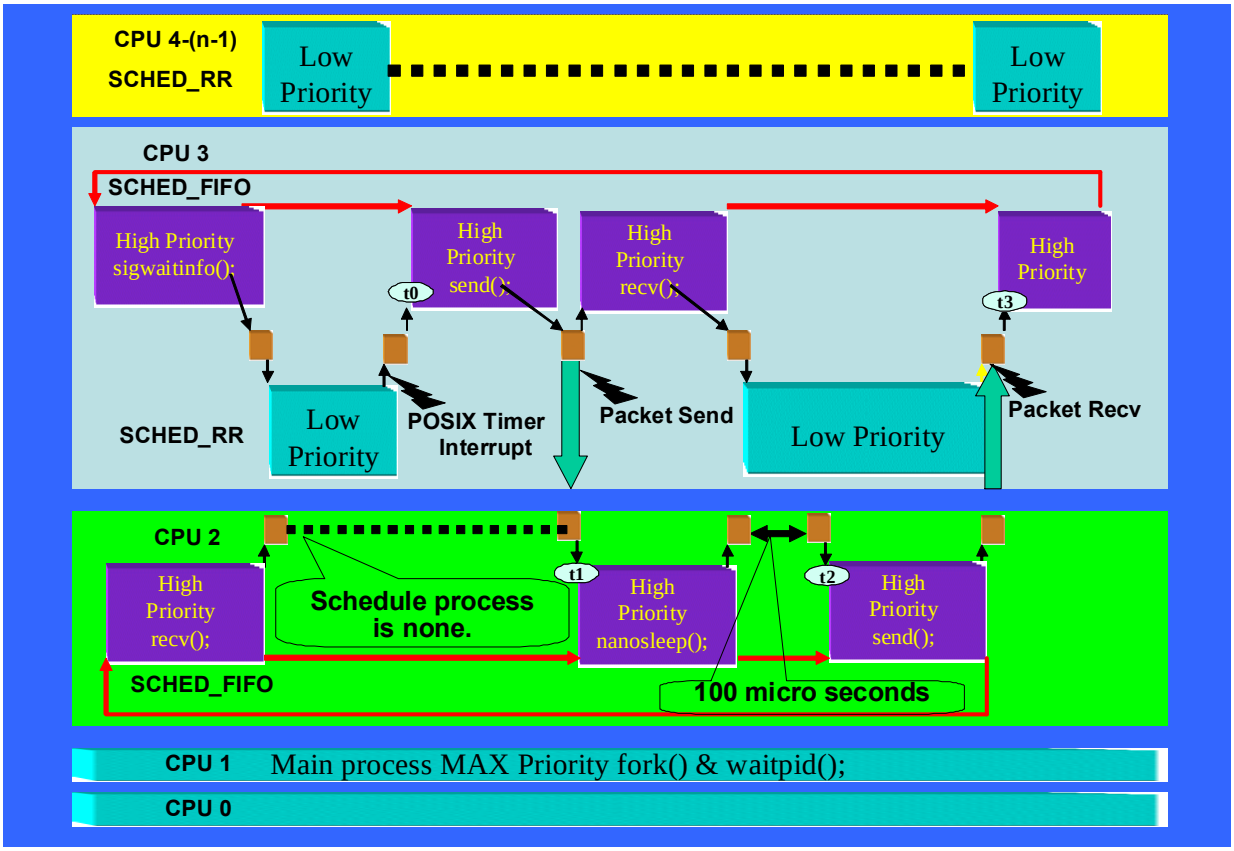



**Fig 3 Structure of a TRLTP program**

A program generates four processes and performs the following operation.

●**main Process** Starting process.
server, client, and an idle process are started and it waits for the end of each process.

1. MAX priority of SCHED_FIFO

●**client Process** Main Process

1. Generate the POSIX timer of 1000 Hz and measure the jitter (t0).
2. synchronizing with the POSIX timer of 1000 Hz -- a server process -- TCP/IP -- 8 bytes of null -- transmit data. And waiting and time stamp measurement (t3) in the time are performed for the response from a server process.
3. MAX priority-1 of SCHED_FIFO

●**server Process** The high priority auxiliary process which receives the request from a client process and is run.

1. Wait for the reception from a client process by select.
2. Perform after-return reception from select and perform time stamp measurement (t1) in

the time.
3. Perform POSIX nanosleep for 100 microseconds.
4. Perform time stamp measurement (t2) in the time after returning from nanosleep().
5. Transmit to a client process by using the time stamp of t1 and t2 as data.
6. MAX priority-2 of SCHED_FIFO

**●Idle Process**

The low priority auxiliary process which receives the request from a client process and is performed.

Whenever it repeats execution, a process increases and goes by a program option.

1. POSIX from client process It waits for a real-time signal.
2. Open a file, perform double precision fft calculation of 65536 points, write out a result to a file and eliminate it.
3. Repeat processing of 2 until there are directions.
4. MIN priority of SCHED_RR

By a program, it is the minimum (microsecond) about the statistical data of the following values. Average (microsecond) Maximum (microsecond) When it outputs as a deviation and points by an option l file name, it combines with the histogram data and records on a file.

However, a desired value is used for calculation of the deviation of interval timer time and the time of nanosleep, without a center value carrying out an average.

| | |
|---|---|
| POSIX Timer Interval Time | The value between t0 [n-1] and t0 [n] |
| Send Time | The value between t0 [n] and t1 [n] |
| nanosleep(100 Micro Sec) | The value between t1 [n] and t2 [n] |
| Receive Time | The value between t2 [n] and t3 [n] |
| Response Time | The value between t0 [n] and t3 [n] |
| Total Jitter | The absolute value which subtracted t1 [n] and t2 [n] from the value between t0 [n] and t3 [n], and subtracted 100 microseconds from the difference of t1 [n] and t2 [n] is applied.<br>The value which added the absolute value between t0 [n-1] and t0 [n] by the same operation.<br>When this asks for a deviation, it is the disposal for expressing not the difference from an average but a difference with a desired value. |

In addition, although all programs use POSIX-API and the command depending on OS is not used, only CPU assignment is using non-POSIX-API common to Linux.
(Notes: Since there is no definition of CPU assignment in POSIX-API, use sched_setaffinity())

| | | |
|---|---|---|
| With no assignment | CPU0 | A demon etc. is usually performed. |
| main process | CPU1 | MAX priority of SCHED_FIFO. |
| server process | CPU2 (echo back) | MAX priority-2 of SCHED_FIFO. |
| client process | CPU3 (measurement target) | MAX priority-1 of SCHED_FIFO. |
| idle process | CPUs 3, 4, 5, and 6, 7...31 | MIN priority of SCHED_RR. |

As a meaning, one load process is divided into the same CPU as (1) measurement process, and it hits it.
(2) Assign the best state to echo back.

Moreover, in order to eliminate a difference with the counter value of RedHawk RCIM, the clock counter with a built-in CPU is read by the tscrd command.

The result of a TRLTP benchmark test.

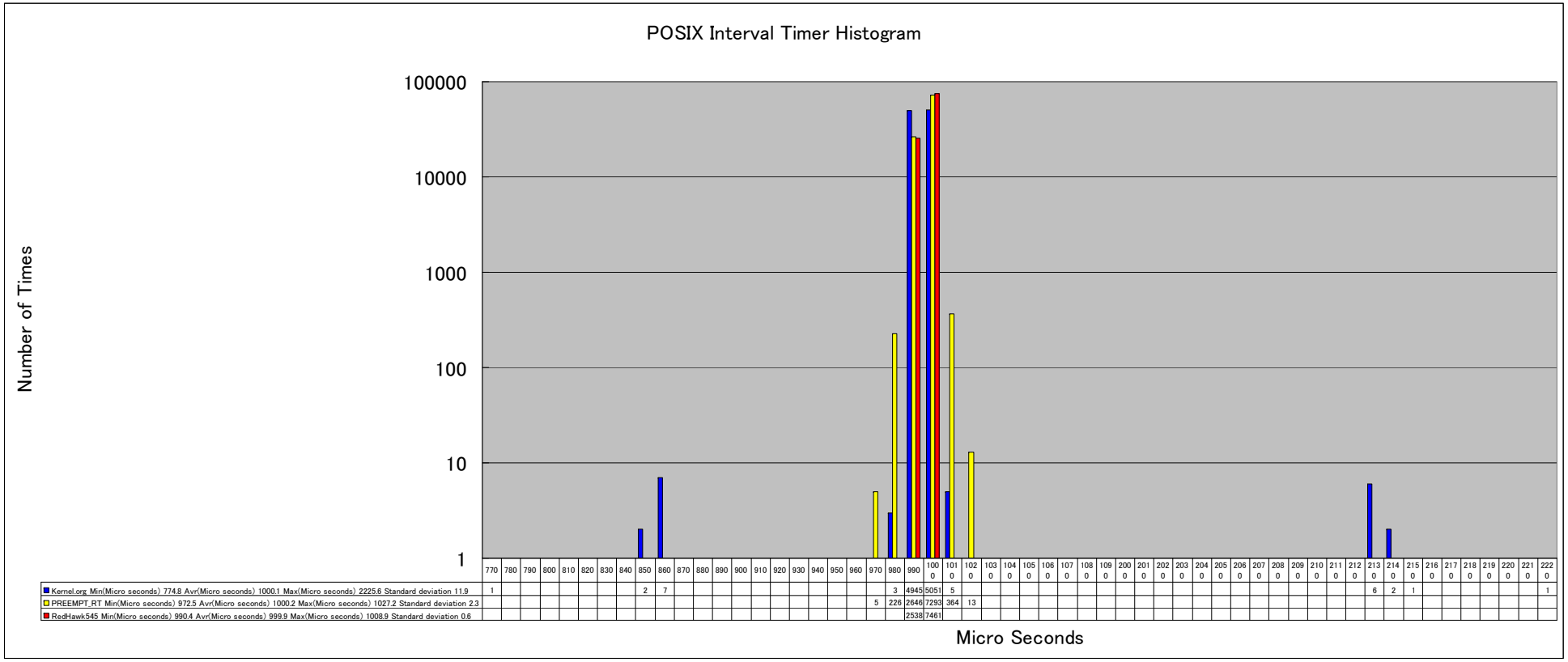
Below, it is Dell T5500. The result of

(1) kernel.org [ which was performed by 6 G bytes of 4-Core 2-CPU main memory ] (RHEL, SUSE, etc.),
(2) PREEMPT_RT application Linux (MRG, SLRT, Ubuntu, etc.), and
(3) RedHawk5.4.5 is shown.

All results are the performances of the same same machine and binary object and 64-bit OS, and the unit of time is mu second. Moreover, 15 low priority processes were performed in the state of existing.

(Notes: Red Hat software Enterprise, Red Hat software MRG, and RedHawk can be performed by the same binary object.)

**Histogram of POSIX Interval Timer**

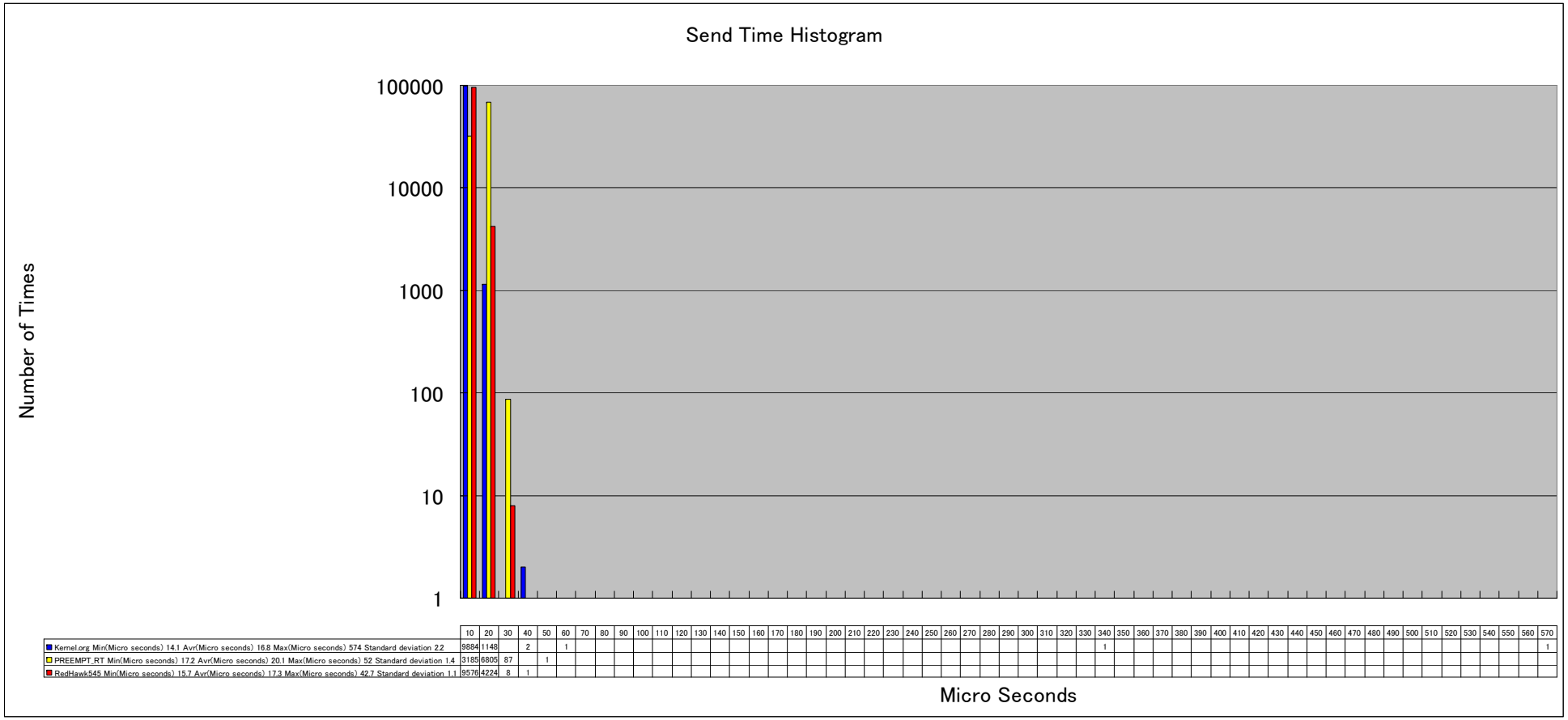



POSIX Interval Timer is measuring the Preempt-time under sigwaitinfo() execution.
Although they show the almost equivalent performance, RedHawk and PREEMPT_RT have a smaller jitter of RedHawk, when a histogram is seen.

By Kernel.org to which PREEMPT-patch is not applied, since the influence from the low priority process assigned to the same CPU is received greatly, it is shown that a jitter is large.

| | Minimum | Average | Maximum | Deviation |
|---|---|---|---|---|
| **Kernel.org** | 774.8 | 1000.1 | 2225.6 | 11.9 |
| **PREEMPT_RT** | 972.5 | 1000.2 | 1027.2 | 2.3 |
| **RedHawk545** | 990.4 | 999.9 | 1008.9 | 0.6 |

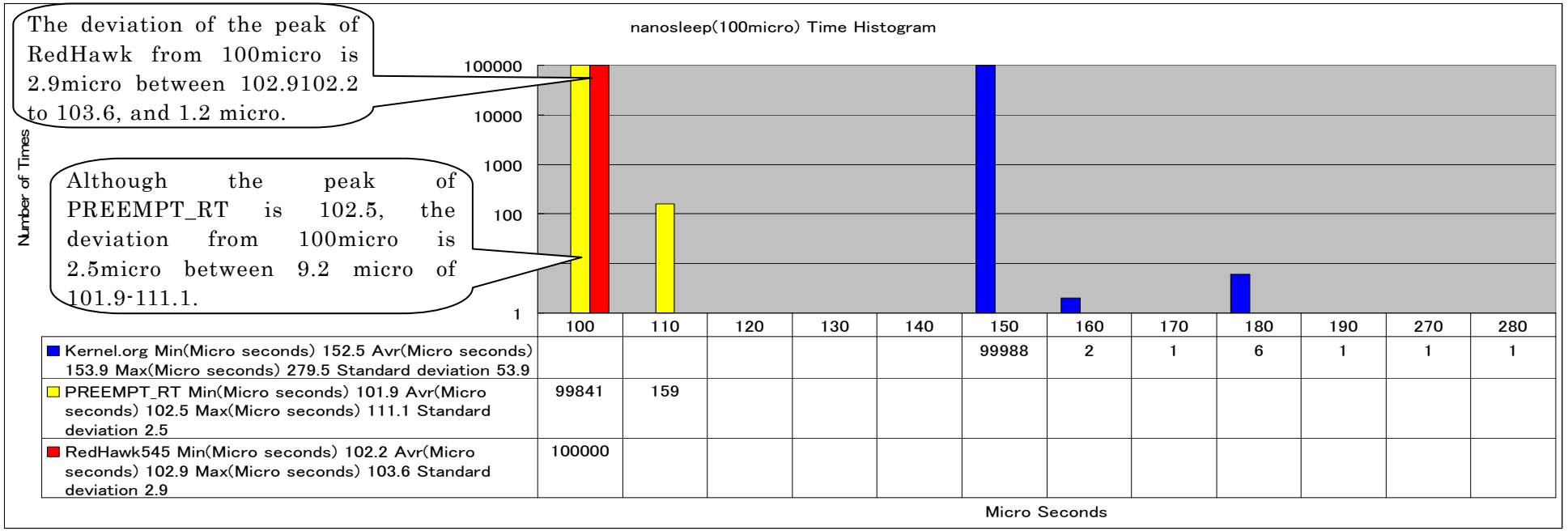
**Histogram of Send Time**




Although transmitting time is the transmitting time to a echo back process (server), probably because a load process exists in the same CPU as a transmitting process, by schedule delay of an IRQ demon and Kernel.org, a jitter looks very greatly.

| | Minimum | Average | Maximum | Deviation |
|---|---|---|---|---|
| **Kernel.org** | 14.1 | 16.8 | 574.0 | 2.2 |
| **PREEMPT_RT** | 17.2 | 20.1 | 52.0 | 1.4 |
| **RedHawk545** | 15.7 | 17.3 | 42.7 | 1.1 |

**Histogram of nanosleep**

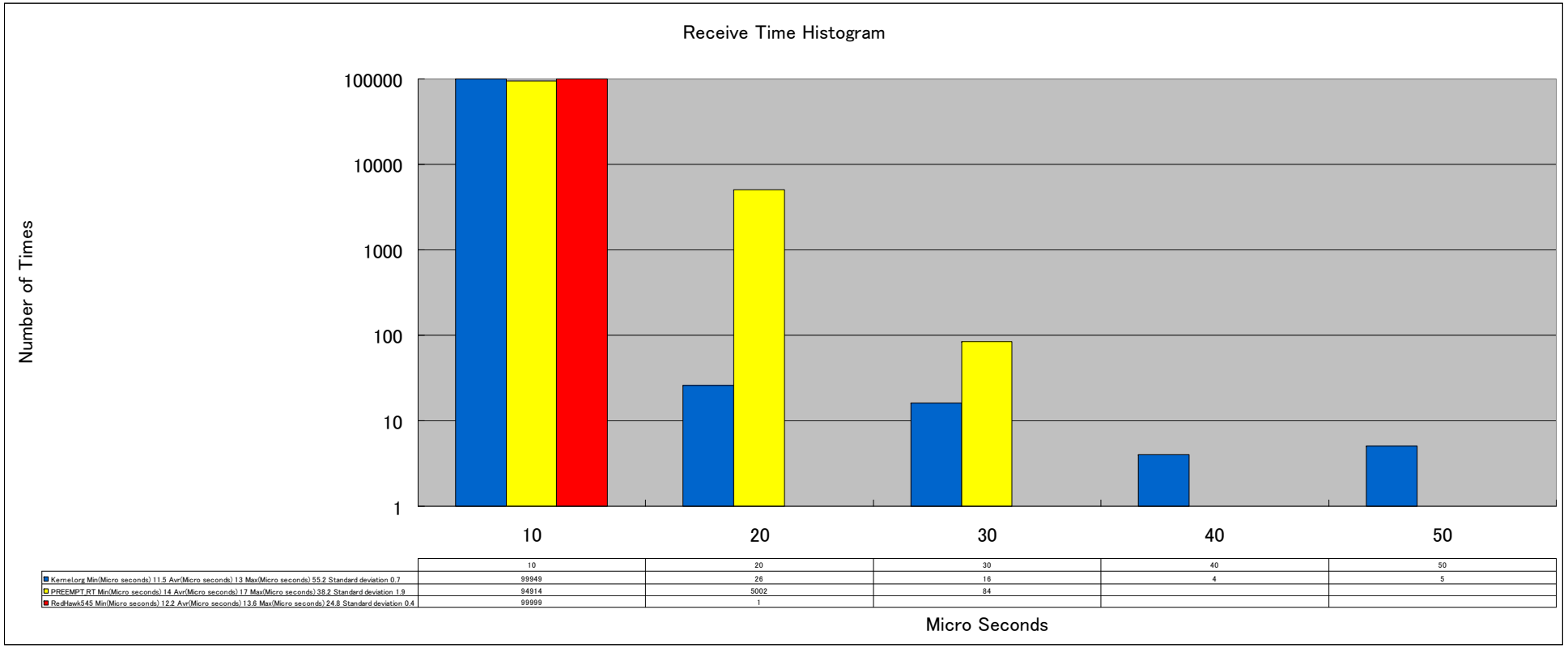



Although the result of nanosleep is no-load on the design which expresses the character of OS very much, in order to return only in time for 150 microseconds or more to the demand for 100 microseconds, the use for 100 microseconds is not borne by kernel.org.

However, PREEMPT_RT which is OS which applied the PREEEMPT_RT patch to kernel.org shows the very good result as compared with kernel.org. A result in which the overhead for 2.9 microseconds exists by average value in kernel.org, and exists in RedHawk by a 2.5micro overhead was brought. When this graph and deviation are compared, that PREEMPT_RT shows the good performance by the deviation has less distribution of RedHawk at a histogram than PREEMPT_RT. Since this is asking for the distribution to 100 microseconds of desired values, it is because a result with more sufficient PREEMPT_RT is brought. If it asks for the distribution to an average, a result with little distribution of RedHawk will be brought, but about nanosleep, PREEMPT_RT can say that the good performance is shown.

| | Minimum | Average | Maximum | Deviation |
|---|---|---|---|---|
| Kernel.org | 152.5 | 153.9 | 279.5 | 53.9 |
| PREEMPT_RT | 101.9 | 102.5 | 111.1 | 2.5 |
| RedHawk545 | 102.2 | 102.9 | 103.6 | 2.9 |

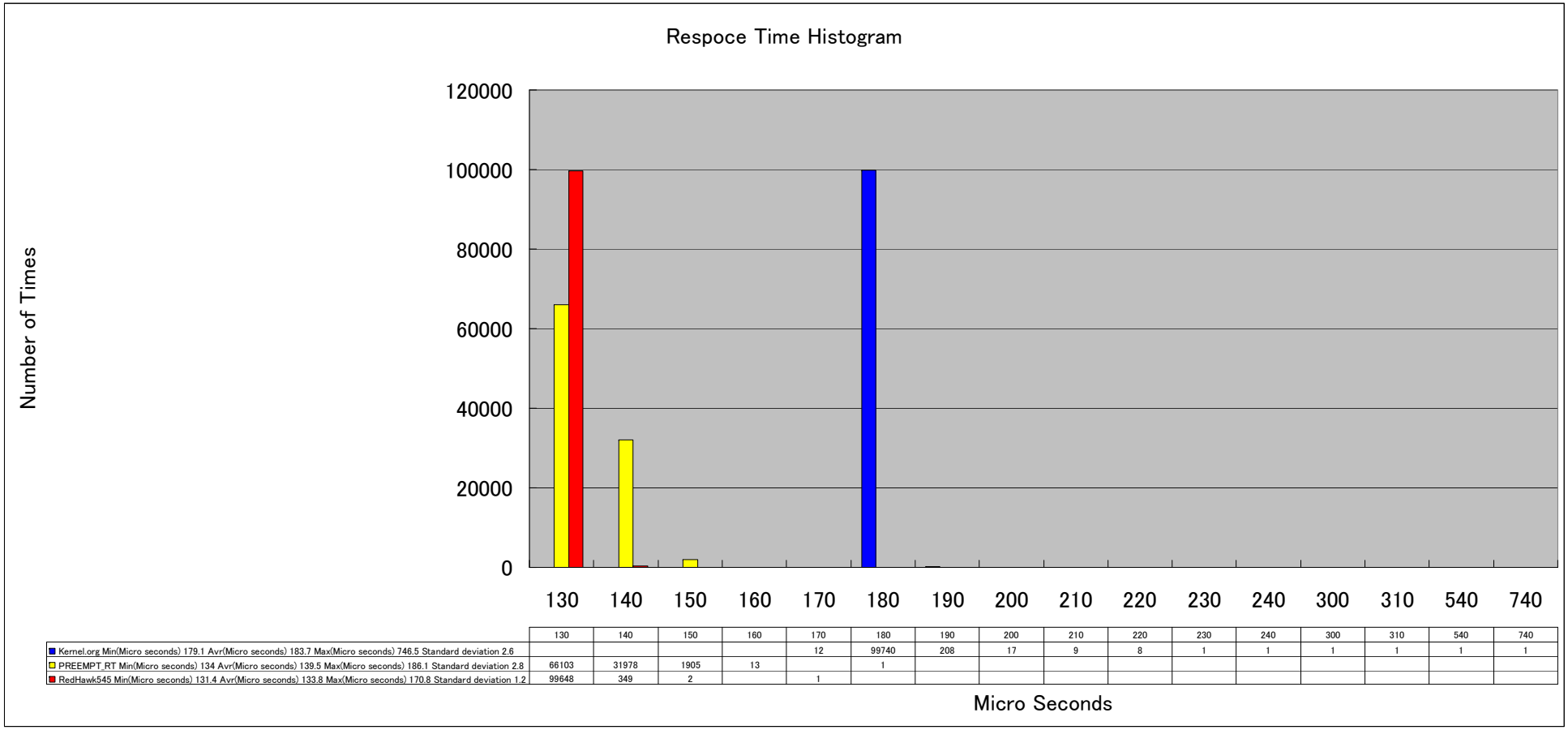
**Histogram of Receive Time**




Receiving time expresses the response delay from a echo back process (server). Compared with transmitting time, there are few differences between OS's. This can be considered to be because for CPU of a sending end to be no-load.
Therefore, when the waiting for the end of transmitting exists at the time of transmission if it combines with a previous transmitting result and thinks, and the process that a priority is low exists in the same CPU, the process that a priority is low is till the end of transmitting,If it performs and PREEMPT-Patch is not applied, he can understand that transmitting delay will be caused.

| | Minimum | Average | Maximum | Deviation |
|---|---|---|---|---|
| **Kernel.org** | 11.5 | 13.0 | 55.2 | 0.7 |
| **PREEMPT_RT** | 14.0 | 17.0 | 38.2 | 1.9 |
| **RedHawk545** | 12.2 | 13.6 | 24.8 | 0.4 |

**Histogram of Response Time**

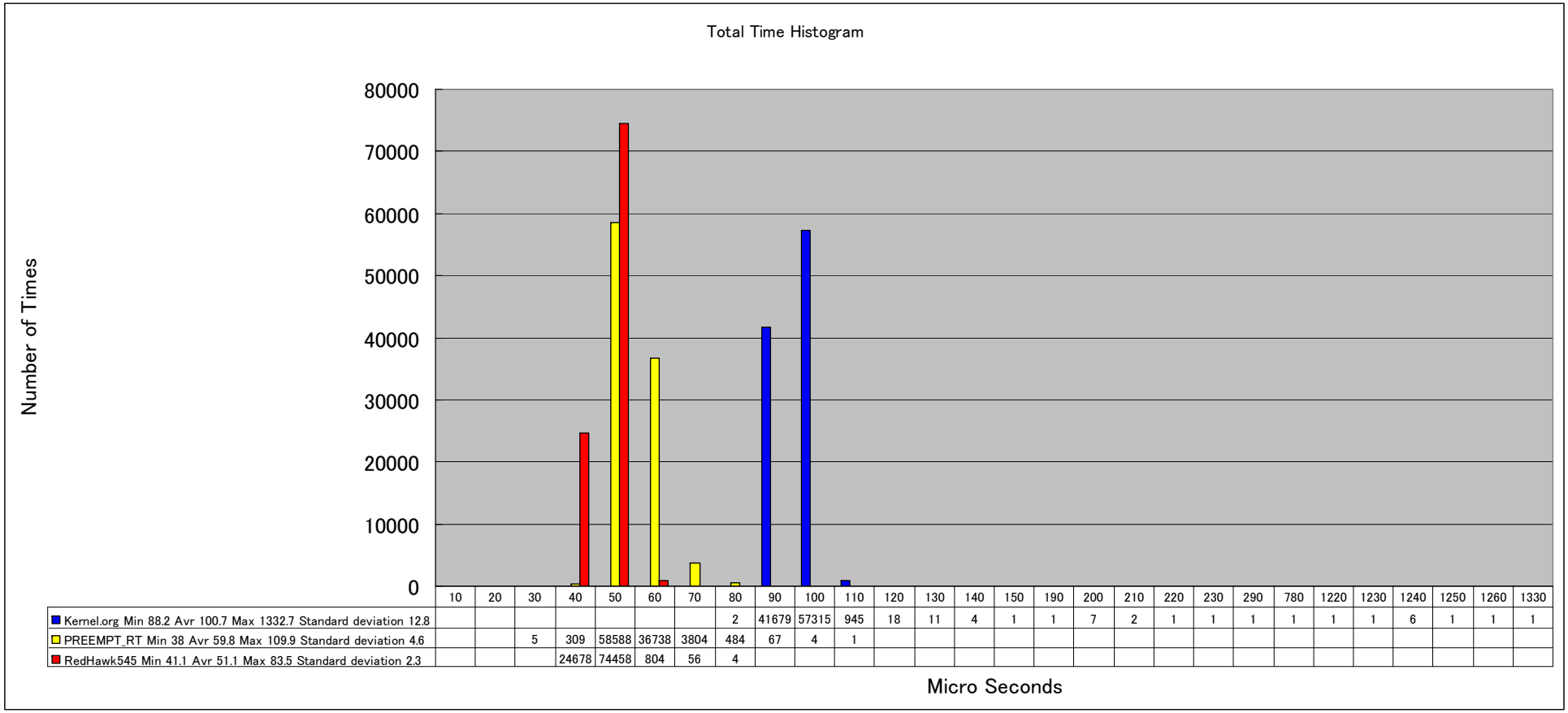



| | 130 | 140 | 150 | 160 | 170 | 180 | 190 | 200 | 210 | 220 | 230 | 240 | 300 | 310 | 540 | 740 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Kernel.org | | | | | 12 | 99740 | 208 | 17 | 9 | 8 | 1 | 1 | 1 | 1 | 1 | 1 |
| PREEMPT_RT | 66103 | 31978 | 1905 | 13 | | 1 | | | | | | | | | | |
| RedHawk545 | 99648 | 349 | 2 | | 1 | | | | | | | | | | | |

This is all the results of transmission, nanosleep (100 microseconds), and reception.
At the lowest, 5 times of system calls and the jitter at the time of five re-schedules are included.

| | Minimum | Average | Maximum | Deviation |
|---|---|---|---|---|
| **Kernel.org** | 179.1 | 183.7 | 746.5 | 2.6 |
| **PREEMPT_RT** | 134.0 | 139.5 | 186.1 | 2.8 |
| **RedHawk545** | 131.4 | 133.8 | 170.8 | 1.2 |

**Histogram of Total Time**




| | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 110 | 120 | 130 | 140 | 150 | 190 | 200 | 210 | 220 | 230 | 290 | 780 | 1220 | 1230 | 1240 | 1250 | 1260 | 1330 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Kernel.org Min 88.2 Avr 100.7 Max 1332.7 Standard deviation 12.8 | | | | | | | | 2 | 41679 | 57315 | 945 | 18 | 11 | 4 | 1 | 1 | 7 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 6 | 1 | 1 | 1 |
| PREEMPT_RT Min 38 Avr 59.8 Max 109.9 Standard deviation 4.6 | | | 5 | 309 | 58588 | 36738 | 3804 | 484 | 67 | 4 | 1 | | | | | | | | | | | | | | | | | |
| RedHawk545 Min 41.1 Avr 51.1 Max 83.5 Standard deviation 2.3 | | | | 24678 | 74458 | 804 | 56 | 4 | | | | | | | | | | | | | | | | | | | | |

It is the result of evaluating the time of all above. The performance of OS is directly expressed only with this figure.

| | Minimum | Average | Maximum | Deviation |
|---|---|---|---|---|
| **Kernel.org** | 88.2 | 100.7 | 1332.7 | 12.8 |
| **PREEMPT_RT** | 38.0 | 59.8 | 109.9 | 4.6 |
| **RedHawk545** | 41.1 | 51.1 | 83.5 | 2.3 |

Thus, he can understand that only a total jitter is seen and real-time synthesis performance can be evaluated, and it is shown that the design of this program is right.

## Conclusion

.

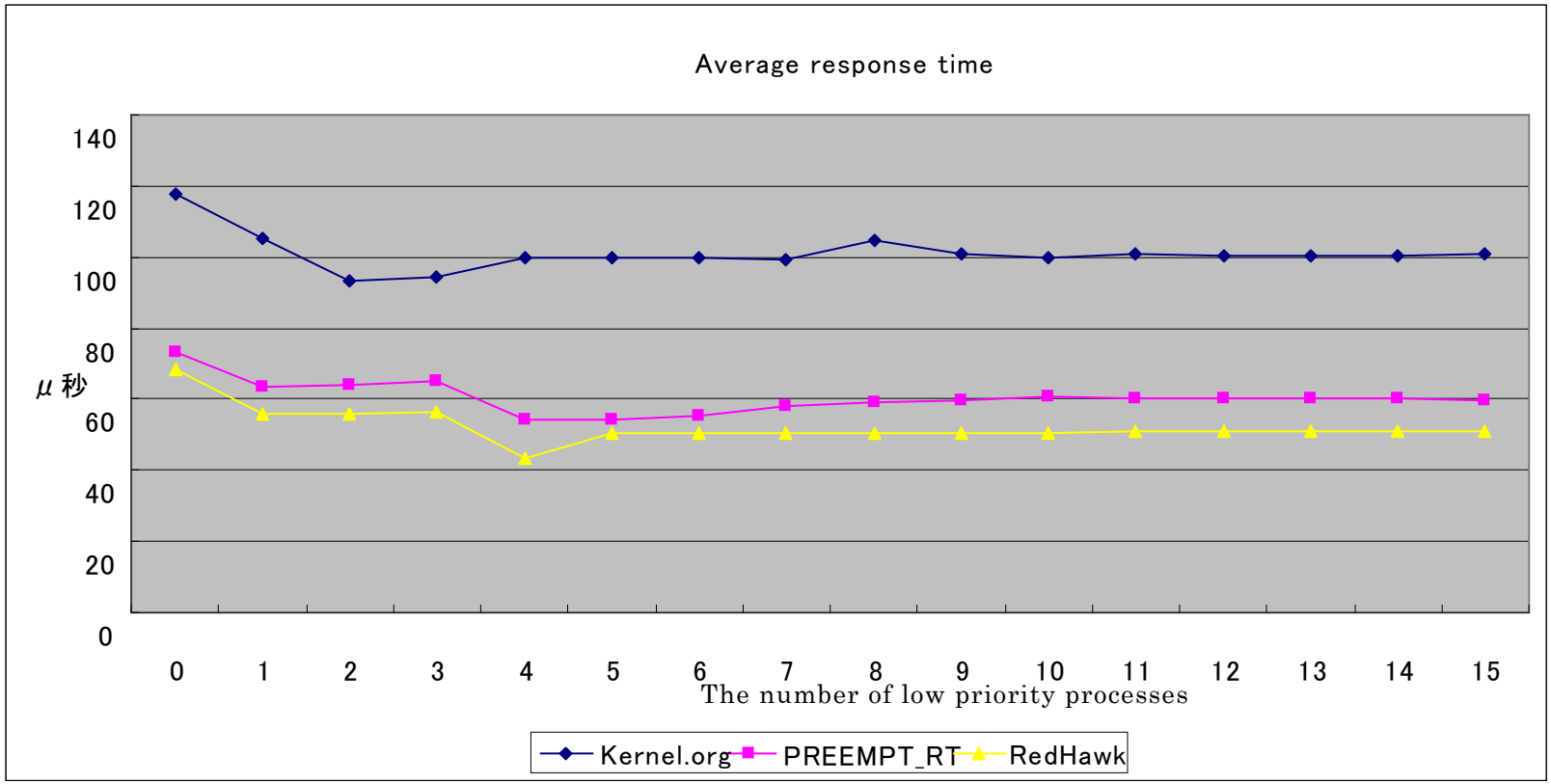
Below, the value of the minimum at the time of increasing the number of low priority processes, an average, the maximum, and a deviation is shown.

**Total response time**

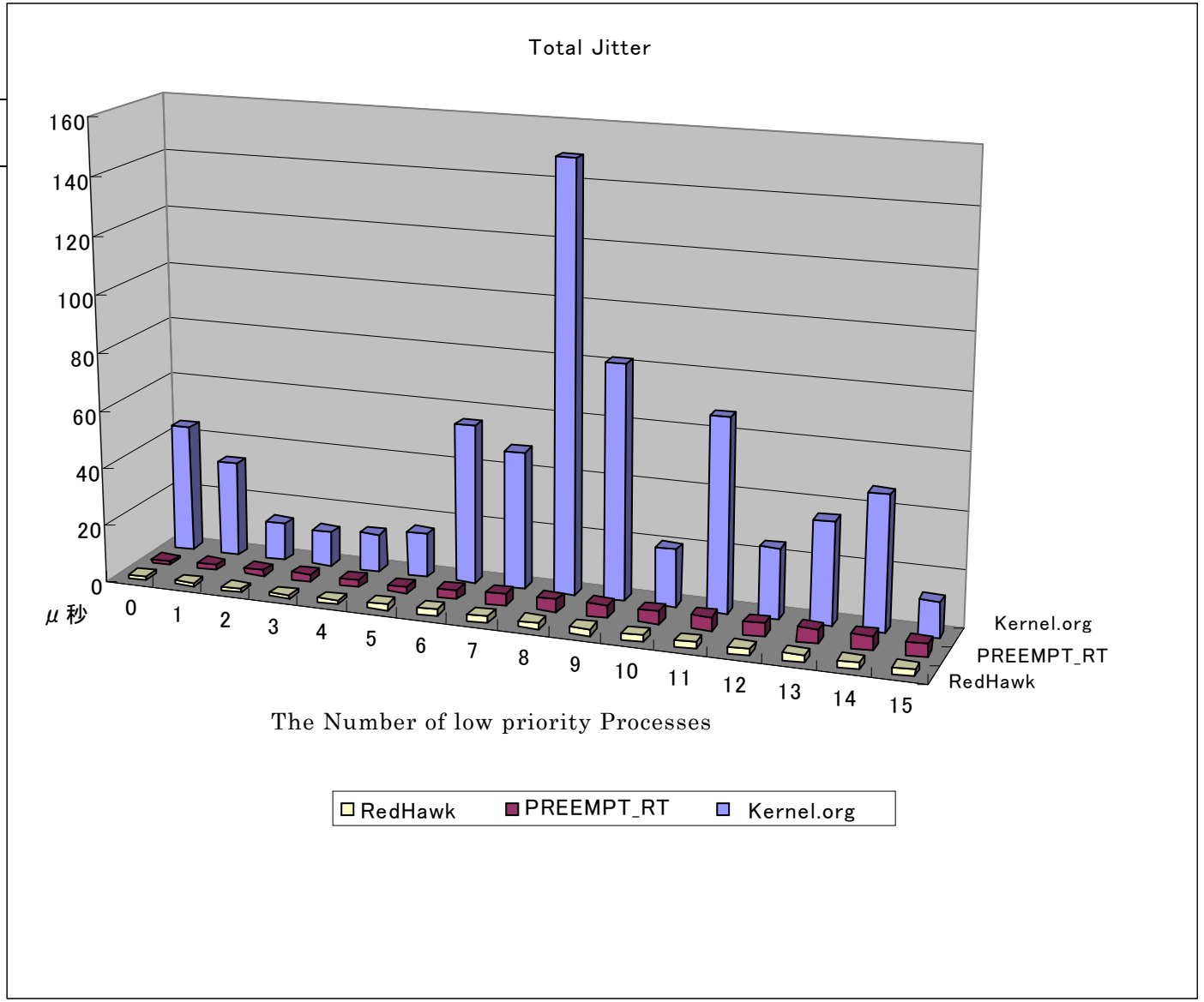
| Minimum | | | Average | | | Maximum | | | Deviation | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Kernel.org | PREEMPT_RT | RedHawk | Kernel.org | PREEMPT_RT | RedHawk | Kernel.org | PREEMPT_RT | RedHawk | Kernel.org | PREEMPT_RT | RedHawk |
| 96.1 | 54.4 | 54.6 | 118 | 73.4 | 68.5 | 11451.9 | 91 | 76.3 | 44.7 | 1.1 | 1.3 |
| 80.1 | 53.5 | 42.2 | 105.3 | 63.5 | 56 | 6927.6 | 100.3 | 77.8 | 33.2 | 1.9 | 1.1 |
| 81.5 | 51.5 | 42.7 | 93.5 | 64.1 | 56 | 1361.4 | 77.1 | 76.6 | 13.4 | 2.1 | 1 |
| 80 | 51 | 43 | 94.2 | 65.1 | 56.2 | 1221.4 | 95.4 | 82.2 | 12.3 | 2.6 | 1 |
| 78.1 | 47.8 | 38.1 | 99.6 | 54.3 | 43.4 | 2317.4 | 87.2 | 72 | 13.3 | 2.6 | 1.3 |
| 83.8 | 44.3 | 40.9 | 99.8 | 54 | 50.6 | 1748.5 | 82.6 | 80 | 15.6 | 2.2 | 2.2 |
| 82.4 | 44.9 | 44.7 | 99.8 | 55.2 | 50.6 | 7031.6 | 88.9 | 85.1 | 55.9 | 3.1 | 2.3 |
| 83.3 | 41 | 43.1 | 99.3 | 58.3 | 50.6 | 6571.4 | 91.1 | 84.8 | 48.2 | 4.1 | 2.3 |
| 57.4 | 38.4 | 42.8 | 104.9 | 59.2 | 50.6 | 7083.6 | 95.6 | 83.5 | 149.3 | 4.3 | 2.2 |
| 85.6 | 37.9 | 44 | 100.7 | 59.5 | 50.7 | 6460.3 | 99.8 | 83 | 82.2 | 4.5 | 2.3 |
| 86.9 | 38.7 | 45.3 | 100 | 60.6 | 50.7 | 5085.7 | 103.6 | 83.3 | 20.1 | 4.7 | 2.4 |
| 85.4 | 38.1 | 44.7 | 101.2 | 60 | 50.9 | 6770.7 | 103.8 | 83.8 | 67.5 | 4.8 | 2.3 |
| 86.7 | 38.6 | 43.3 | 100.3 | 60.3 | 50.9 | 6532.5 | 103.7 | 82.7 | 24.6 | 4.9 | 2.3 |
| 87.7 | 38.3 | 45.5 | 100.2 | 60.2 | 51 | 7487.8 | 115.2 | 80.1 | 35.7 | 4.9 | 2.4 |
| 85.8 | 37.5 | 44.8 | 100.2 | 60.5 | 51.1 | 12843.5 | 104.8 | 85.4 | 46.9 | 5 | 2.3 |
| 88.2 | 38 | 41.1 | 100.7 | 59.8 | 51.1 | 1332.7 | 109.9 | 83.5 | 12.8 | 4.6 | 2.3 |

As a measure of real-time performance, determinacy is important. that is, "jitter is small " it is important. This has a deviation of a TRLTP examination equal to a small thing.

Therefore, it is shown in order of (1)RedHawk and (2)PREEMPT_RT that real-time performance is high, and the result that real-time nature is that there is nothing (with no determinacy) is shown in kernrl.org.




It can read that the upper graph shows average response time, and the same tendency is shown although performances differ.
That is, when the number of low priority processes is increased and it goes, average response time is a point which has brought the result that it was contradictory apparently of approaching a fixed value and being in the reduction instead of the increase from an unloaded condition.
It has a cause that context switching and the number of times of interruption will increase this if the number of processes increases, and the number of times of scheduling within a kernel increases.
namely, "the number of times of a demand to a kernel" =" re-schedule " it is it will answer at high speed by things.
This fact shows that it is cautious of the quality assessment of OS. That is, it is because it is possible that the performance may be mistaken if load is increased recklessly.




Right graph is a jitter to average response time.

Although RedHawk shows the good performance for 3 or less microseconds irrespective of the number of processes, Kernel.org does not have correlation.

That is, in Kernel.org, it is that computability is lacking, and if there is no real-time nature, it will come to a conclusion.

Although PREEMPT_RT shows the fixed performance, when the number of low priority clients is increased, PREEMPT_RT has it in the tendency for performance to fall and go relatively.

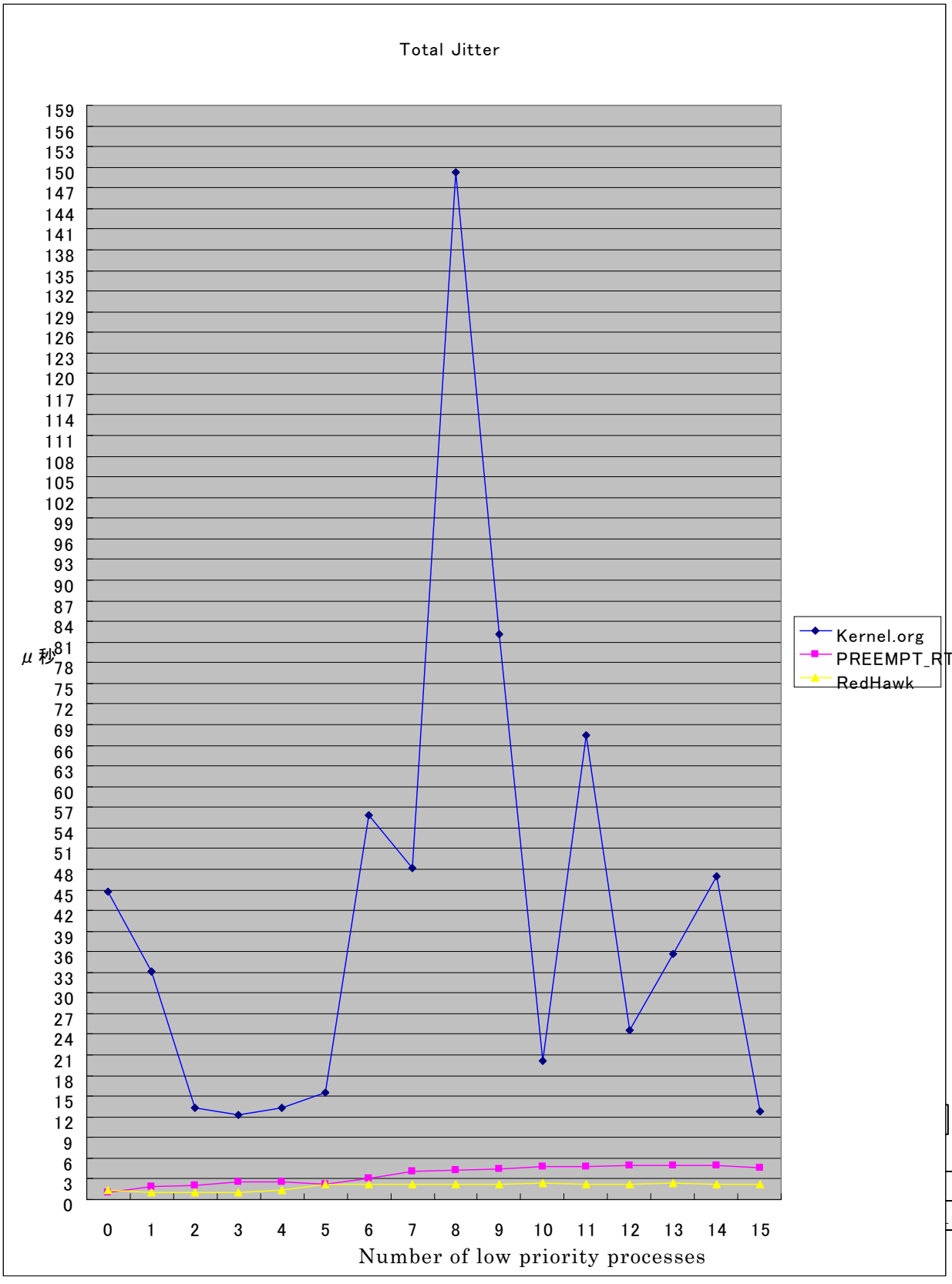
The graph of the following page changes a right bar graph into a line graph. If the number of low priority clients is increased, PREEMPT_RT is in the tendency for performance to fall and go relatively, and change of the number of low priority processes has taken place from a point of 5 by this examination, as stated previously.

This is related to reaching the CPU maximum carried in this system by 5, when a load process is assigned sequentially from CPU3.

That is, in PREEMPT_RT, when the number of processes assigned to a core based CPU increases, even if it is a process of a low priority, if it influences by the response of the process of a high priority, it can come to a conclusion.

In RedHawk, the influence can read a small thing in the following graph.

thus, Kernel which applied the thing PREEMPT_RT patch where RedHawk is designed contribute to improvement in a real-time response, and which does not have a response guarantee -- it is concluded that it has checked that it was the performance distinguished from Linux 2.6 system.



Source code

Below, all the source codes are shown

```
/*
 * ****************************************************************************
 *   Copyright(c) 2010 Concurrent Computer Corporation
 *                          ALL RIGHTS RESERVED
 *
 *   This software is furnished under a license and may be used and copied only
 *   in accordance with the terms of such license and with the inclusion of the
 *   above copyright notice. This software or any other copies thereof, may not
 *   be provided or otherwise made available to anyone other than the licensee.
 *
 *                   Title to and ownership of this software remains with
 *                   the Concurrent Computer Corporation (CONCURRENT).
 *
 *          The information in this document is subject to change without
 *          notice and should not be construed as a commitment by CONCURRENT.
 *
 *          CONCURRENT assumes no responsibility for the use or reliability of
 *          its software on equipment that is not supplied by CONCURRENT.
 * Version 3.0 ALL POSIX API VERSION 32bit & 64bit & TSC
 * ****************************************************************************
 */
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <time.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>
#include <locale.h>
#include <nl_types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <sys/uio.h>
#include <sys/time.h>
#include <poll.h>
#include <netinet/tcp.h>
#include <signal.h>
#include <sched.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>

static sig_atomic_t idle_flag=1, loop_flag=1;
char *inet_ntoa(struct in_addr in);
struct sockaddr_in          tcp_srv_addr;
struct servent              tcp_serv_info;
struct hostent              tcp_host_info;
#define TSC
#define MAX_HISTORY         200000
#define MAXCOUNT            100001
#define SERV_HOST_ADDR    "127.0.0.1"
#define   CLI_HOST_ADDR    "127.0.0.1"
#define SERV_PORT           20000
#define MAX_PACKET          (65536)
#define SERVER      0x01
#define CLIENT      0x02
#define ASYNC       0x04
#define IDLE        0x08
#define TCP         0x10

#define PACKET_SIZE         1024
#define END_MARK 0xFF

static int DEBUG=0;
static int CYCLE=1000;
```

```
static int PORT=SERV_PORT;

FILE *logfp = NULL;

static int ASYNC_FLAG = 0;
static int TestLoop = 1;
static pid_t IdlePID[128];
#define     PI          3.14159265358979323846
#define N 65536

typedef struct
{
            double r;
            double i;
} complex;

static int last_n = 0;
static int *bitrev = NULL;
static double *sintbl = NULL;
static double *isintbl = NULL;

static void make_sintbl2(
            int         n,
            double      sintbl[],
            double      isintbl[])
{
            int         i,n2,n4,n8;
            double      c,s,dc,ds,t;

            n2 = n / 2; n4 = n / 4; n8 = n / 8;
            t = sin(PI / n);
            dc = 2 * t * t; ds = sqrt(dc*(2-dc));
            t = 2 * dc; c = sintbl[n4] = 1; s = sintbl[0] = 0;
            isintbl[n4] = -1; isintbl[0] = 0;
            for (i=1;i<n8;i++)
            {
                        c -= dc; dc += t * c;
                        s += ds; ds -= t * s;
                        sintbl[i] = s; sintbl[n4-i] = c;
                        isintbl[i] = -s; isintbl[n4-i] = -c;
            }
            if (n8!=0) { sintbl[n8] = sqrt(0.5); isintbl[n8]=-sintbl[n8];}
            for ( i=0; i<n4; i++)
            {
                        sintbl[n2-i] = sintbl[i];
                        isintbl[n2-i] = -sintbl[i];
            }
            for ( i=0; i<n2+n4; i++)
            {
                        sintbl[i+n2] = -sintbl[i];
                        isintbl[i+n2] = sintbl[i];
            }
}

static int make_bitrev2(
            int         n,
            int         bitrev[])
{
            int         i,j,k,n2;

            n2 = n/2; i = j = 0;
            for( ; ; ) {
                        bitrev[i] = j;
                        if (++i >= n )          break;
                        k = n2;
                        while (k<=j) { j -= k; k /= 2; }
                        j += k;
            }
            return(0);
}

static int make_table(int n)
{
            register int            i,j,k,n4,stage;
```

```
	if (n==0)
	{
		if ( sintbl != NULL ) {free(sintbl);free(isintbl);}
		if ( bitrev != NULL ) free(bitrev);
		return(0);
	}
	if (n<0)
	{
		n= -n;
	}
	n4 = n / 4;
	if (n!=last_n || n== 0 )
	{
		last_n = n;
		if ( sintbl != NULL ) {free(sintbl);free(isintbl);}
		if ( bitrev != NULL ) free(bitrev);
		if (n==0) return(0);

		sintbl = (double *)malloc((n+n4) * sizeof(double));
		isintbl = (double *)malloc((n+n4) * sizeof(double));
		bitrev = (int *)malloc(n * sizeof(int));

		if (sintbl == NULL ||isintbl == NULL|| bitrev == NULL)
		{
			fprintf(stderr,"memory allocation error¥n");
			return(1);
		}
		make_sintbl2(n,sintbl,isintbl);
		make_bitrev2(n,bitrev);
	}
	return(0);
}

static int small_fft( int n, complex v[])
{
	int		i,j,k,ik,h,d,k2,n4,inverse;
	double	s,c,dx,dy;
	register complex	t;

	if (n<0) {
		n= -n;	inverse = 1;
	} else {
		inverse = 0;
	}
	n4 = n / 4;
	for(i=0; i < n; i++){
		j = bitrev[i];
		if(i<j){
			t = v[i]; v[i] = v[j]; v[j] = t;
		}
	}
	for (k=1;k<n;k=k2) {
		h = 0; k2 = k + k; d = n / k2;
		for (j=0;j<k;j++) {
			c = sintbl[h+n4];
			if (inverse) {
				s = -sintbl[h];
			} else {
				s =  sintbl[h];
			}
			for (i=j;i<n;i+=k2) {
				ik = i + k;
				dx = s * v[ik].i + c * v[ik].r;
				dy = c * v[ik].i - s * v[ik].r;
				v[ik].r = v[i].r - dx; v[i].r += dx;
				v[ik].i = v[i].i - dy; v[i].i += dy;
			}
			h += d;
		}
	}
	if (!inverse) {
		for (i=0;i<n;i++) { v[i].r /= (double)n; v[i].i /= (double)n; }
	}
	return(0);
}
```

```
static void delete_posix_timer(timer_t timer_id)
{
        static struct itimerspec itspec;

        /*      Set the initial delay and period of the timer.
        This also arms the timer */
        clock_gettime(CLOCK_REALTIME,&itspec.it_value);
        itspec.it_interval.tv_sec = 0;
        itspec.it_interval.tv_nsec =    0;
        timer_settime( timer_id, TIMER_ABSTIME, &itspec, NULL);
         timer_delete(timer_id);
}

static timer_t create_posix_timer(int signum,int sec,int nsec)
{
        static struct sigevent event;
        static timer_t timer;
        static struct itimerspec itspec;

        /* Create the POSIX timer to generate signum */
        event.sigev_notify = SIGEV_SIGNAL;
        event.sigev_signo = signum;
        if (timer_create((clockid_t)CLOCK_REALTIME,&event,&timer)==(-1))
        {
                fprintf(stderr, "create_posix_timer() Cannot create timer - %s¥n",
                strerror(errno));
                return ((timer_t)-1);
        }
        /*      Set the initial delay and period of the timer.
        This also arms the timer */
        clock_gettime(CLOCK_REALTIME,&itspec.it_value);
        itspec.it_value.tv_sec += 3;
        itspec.it_interval.tv_sec = sec;
        itspec.it_interval.tv_nsec =    nsec;
        if (timer_settime( timer, TIMER_ABSTIME, &itspec, NULL)==(-1))
        {
                return ((timer_t)-1);
        }
        return(timer);
}

#if 1
unsigned long long int rdtsc() {
unsigned int lo, hi;
/* We cannot use "=A", since this would use %rax on x86_64 */
__asm__ __volatile__ ("rdtsc" : "=a" (lo), "=d" (hi));
return (unsigned long long int)hi << 32 | lo;
}
double cpu_clock;
double getMHz(void)
{
         double mhz;
        FILE *f = fopen("/proc/cpuinfo", "r");
        if (f == 0){
                perror("can't open /proc/cpuinfo¥n");
                exit(1);
        }

        for ( ; ; ){
                int ret;
                char buf[1000];

                if (fgets(buf, sizeof(buf), f) == NULL){
                        fprintf(stderr, "FATAL: cannot locate cpu MHz in " "/proc/cpuinfo¥n");
                        exit(1);
                }
                ret = sscanf(buf, "cpu MHz              : %lf", &mhz);
                if (ret == 1){
                        fclose(f);
                        return (mhz);
                }
        }
}
```

```
static void readtime(struct timespec *nowtime)
{
#ifdef TSC
        unsigned long long T;

        T = rdtsc();
        nowtime->tv_nsec = T & 0xFFFFFFFF;
        nowtime->tv_sec = (T>>32) & 0xFFFFFFFF;
#else
        clock_gettime(CLOCK_REALTIME,nowtime);
#endif
}
static void adjust(struct timespec *start,struct timespec *finish, double *realtime)
{
#ifdef TSC
        unsigned long long t0,t1;
        t0 = (unsigned long long)start->tv_sec;
        t0 <<=32;
        t0 |= 0xFFFFFFFF & (unsigned long long)(start->tv_nsec);
        t1 = (unsigned long long)finish->tv_sec;
        t1 <<=32;
        t1 |= 0xFFFFFFFF & (unsigned long long)(finish->tv_nsec);
        *realtime = (double)(t1-t0)/cpu_clock;
#else
        int     sec,nsec;

        sec = finish->tv_sec - start->tv_sec;
        nsec = finish->tv_nsec - start->tv_nsec;
        if (nsec<0)
        {
                sec --;
                nsec += 1000000000;
        }
        *realtime = (double)(nsec/1000)+(double)(sec*1000000);
#endif
}
#else
#define readtime(nowtime)           clock_gettime(CLOCK_REALTIME,nowtime)
static void adjust( struct timespec *start, struct timespec *finish, double *realtime)
{
        int     sec,nsec;

        sec = finish->tv_sec - start->tv_sec;
        nsec = finish->tv_nsec - start->tv_nsec;
        if (nsec<0)
        {
                sec --;
                nsec += 1000000000;
        }
        *realtime = (double)(nsec/1000)+(double)(sec*1000000);
}
#endif

static int server(int option)
{
         int          listenfd,sockfd,fromlen,len,cli_len;
         struct sockaddr_in serv_addr,from;
         struct sockaddr_in cli_addr;
         static unsigned char serv_buff[MAX_PACKET];
         static int count=1;
         struct timespec t0,t1,t2,diff;
         double real;
         fd_set rfd;
         int ret,n;
         int on = 1;
         int sumlen = 0;
         struct timespec seltimeout;
        time_t now;
        struct tm *ptm;
         static struct sched_param myparam;
        fd_set readfds,writefds,execptfds;
        struct timeval timeout={0,0};
         int        nfds=0;
```

```
static sigset_t set,oset;
 struct pollfd      fdarray[1];
 struct timespec w100={0,100000};

 mlockall(MCL_CURRENT|MCL_FUTURE);
 myparam.sched_priority = sched_get_priority_max(SCHED_FIFO)-2;
sched_setscheduler(getpid(),SCHED_FIFO,&myparam);
 {
 cpu_set_t mask;
 int ret,cpu;
              unsigned int cpusetsize;

              cpusetsize = sizeof(cpu_set_t);
             CPU_ZERO(&mask);
            CPU_SET(2,&mask);
            ret = sched_setaffinity(getpid(),cpusetsize,&mask);
            if (ret<0)
            {
                        printf("sched_setaffinity(%08X) is error %s¥n",mask,strerror(errno));
            }
 }

 {
              if ((listenfd = socket (AF_INET, SOCK_STREAM, 0)) < 0)
              {
                        fprintf(stderr,"server: can't create TCP socket¥n");
                        return(-1);
              }
              if(setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on)) < 0)
              {
                        fprintf(stderr,"server: can't set socket option¥n");
                        return(-1);
              }
              bzero((char *)&serv_addr,sizeof(serv_addr));
              serv_addr.sin_family = AF_INET;
              serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
              serv_addr.sin_port = htons(PORT);
              if (bind(listenfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr))<0)
              {
                        fprintf(stderr,"server: can't bind TCP socket¥n");
                        close(listenfd);
                        return(-1);
              }
              if (listen(listenfd,1)<0)
              {
                        fprintf(stderr,"server: can't listen TCP socket¥n");
                        close(listenfd);
                        return(-1);
              }
              cli_len = sizeof(cli_addr);
              if ((sockfd = accept(listenfd,(struct sockaddr*)&cli_addr,&cli_len))<0)
              {
                        fprintf(stderr,"server: can't accept TCP socket¥n");
                        close(listenfd);
                        return(-1);
              }
              if(setsockopt(sockfd, IPPROTO_TCP, SO_PRIORITY, &on, sizeof(on)) < 0)
              {
                        fprintf(stderr,"server: can't set socket option¥n");
                        return(-1);
              }
 }

 sigemptyset(&set);
sigaddset(&set,SIGALRM);
if (sigprocmask(SIG_UNBLOCK,&set,&oset)==(-1))
{
            fprintf(stderr, "%s: Cannot set sigprocmask - %s¥n", strerror(errno));
            exit(1);
}
nfds=1;
fdarray[0].fd = sockfd;
fdarray[0].events = POLLIN;

 for(n=0;n<TestLoop;n++)
```

```
        {
                count=0;
                while(loop_flag)
                {
                        if (poll(fdarray,nfds,1000)>0)

                        {
                                sched_yield(); len = read(sockfd,serv_buff,sizeof(serv_buff));
                                if(len == sizeof(struct timespec))
                                {
                                        readtime(&t1);
                                        memcpy(serv_buff,(void*)&t1,sizeof(struct timespec));
nanosleep(&w100,0);
                                        readtime(&t1);
                                        memcpy(&serv_buff[sizeof(struct
timespec)],(void*)&t1,sizeof(struct timespec));
                                        write(sockfd,serv_buff,sizeof(struct timespec)*2);
sched_yield();
                                        count++;
                                }
                                else
                                {
                                        break;
                                }
                        }
                }
        }
        close(sockfd);
        if(logfp) fclose(logfp);

        return(0);
}

void calc(FILE *logfp,struct timespec *t0,struct timespec *t1,char *tag,double target )
{
        static double real[MAXCOUNT],min,max,avr,s,x;
        static int his[MAX_HISTORY];
         register int count;

        min=1000000000;
        s=avr =max=0;
        memset((void*)his,0,sizeof(his));
        memset((void*)real,0,sizeof(real));
        for(count=0;count<MAXCOUNT-1;count++)
        {
                adjust(&t0[count],&t1[count],&real[count]);
                his[(int)(real[count]+0.5)/10]++;
                avr += real[count];
                if (min> real[count]) min =    real[count];
                if (max< real[count]) max =    real[count];
        }
        avr /= (double)(MAXCOUNT-1);
        for(count=0;count<MAXCOUNT-1;count++)
        {
                 if (target==0.0)
                        x=real[count]-avr;
                 else
                        x=real[count]-target;
                 s+= (x * x);
         }
         s = sqrt(s/(double)(MAXCOUNT-2));
        printf("%s¥tmin        %8.1f(MicroSec)        avr        %8.1f(MicroSec)        max        %8.1f(MicroSec)
S %8.1f¥n",tag,min,avr,max,s);
        if(logfp)
        {

         fprintf(logfp,"%s¥t,Min,%8.1f,(MicroSeconds),Avr,%8.1f,(MicroSeconds),Max,%8.1f,(MicroSeconds),Sta
ndard deviation,%8.1f¥n",tag,min,avr,max,s);
                fprintf(logfp,"Time(MicroSeconds),Number of Times¥n",count*10,his[count]);
                for(count=0;count<MAX_HISTORY;count++)
                {
                        if (his[count]) fprintf(logfp,"%d,%d¥n",count*10,his[count]);
                }
                fprintf(logfp,"¥n¥n");
        }
```

```
}

static int client(void *vp,int option)
{
            unsigned char *host;
            int          sockfd,fromlen,len;
            struct sockaddr_in serv_addr,dest;
            static unsigned char cli_sbuf[MAX_PACKET];
            int err,signum,n;
           struct hostent *hp;
           unsigned long inaddr;
            int on = 1;
           static struct sigaction newact;
           static sigset_t set,oset;
           static timer_t timer_id;
           static siginfo_t info;
           static struct itimerspec itspec;
           static struct timespec t0[MAXCOUNT];
           static struct timespec t1[MAXCOUNT];
           static struct timespec t2[MAXCOUNT];
           static struct timespec t3[MAXCOUNT];
            static struct sched_param myparam;
            union sigval val;
            struct timespec w0={1,10000000};
           static double interval[MAXCOUNT],min,max,avr,s,x;
           static double total[MAXCOUNT];
           static int his[MAX_HISTORY];
            register int count;

            nanosleep(&w0,NULL);
            mlockall(MCL_CURRENT|MCL_FUTURE);
           myparam.sched_priority = sched_get_priority_max(SCHED_FIFO)-1;
           sched_setscheduler(getpid(),SCHED_FIFO,&myparam);
            {
            cpu_set_t mask;
            int ret,cpu;
                        unsigned int cpusetsize;

                        cpusetsize = sizeof(cpu_set_t);

                       CPU_ZERO(&mask);
                      CPU_SET(4,&mask);
                      ret = sched_setaffinity(getpid(),cpusetsize,&mask);
                      if (ret<0)
                      {
                                  printf("sched_setaffinity(%08X) is error %s¥n",mask,strerror(errno));
                      }
            }
           signum = SIGRTMIN+1;
           if (sigprocmask(0,NULL,&set)==(-1))
           {
                      fprintf(stderr, "Cannot get sigprocmask - %s¥n",strerror(errno));
                      exit(1);
           }
           sigaddset(&set,signum);
           if (sigprocmask(SIG_SETMASK,&set,&oset)==(-1))
           {
                      fprintf(stderr, "%s: Cannot set sigprocmask - %s¥n", strerror(errno));
                      exit(1);
           }



            host = (unsigned char *)vp;
           bzero((char *)&serv_addr,sizeof(serv_addr));

           serv_addr.sin_family = AF_INET;
           if((inaddr = inet_addr(host)) != (-1) ){
                      memcpy((char *)&serv_addr.sin_addr,(char *)&inaddr,
                      sizeof(inaddr));
           } else {
                      if((hp = gethostbyname (host))==NULL){
                                  fprintf(stderr,"udp_open: host name error: %s¥n",host);
                                  return(-1);
                      }
```

```
            memcpy((char *)&serv_addr.sin_addr,hp->h_addr,
                        hp->h_length);
}

{
serv_addr.sin_port = htons(PORT);
if ((sockfd = socket (AF_INET, SOCK_STREAM, 0)) < 0)
{
            fprintf(stderr,"client: can't create TCP socket¥n");
            return(-1);
}
            if(setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on)) < 0)
            {
                        fprintf(stderr,"client: can't set socket option¥n");
                        return(-1);
            }
            if(setsockopt(sockfd, IPPROTO_TCP, TCP_NODELAY, &on, sizeof(on)) < 0)
            {
                        fprintf(stderr,"client: can't set socket option¥n");
                        return(-1);
            }
            if(setsockopt(sockfd, IPPROTO_TCP, SO_PRIORITY, &on, sizeof(on)) < 0)
            {
                        fprintf(stderr,"client: can't set socket option¥n");
                        return(-1);
            }
            if (connect(sockfd,(void*)&serv_addr,sizeof(struct sockaddr_in))<0)
{
            fprintf(stderr,"client: can't connect TCP socket¥n");
            return(-1);
}
}
if (ASYNC_FLAG) fcntl(sockfd, F_SETFL, O_ASYNC);
memcpy((void*)&dest,(void*)&serv_addr,sizeof(struct sockaddr_in));
len = sizeof(struct timespec);

for(n=0;n<TestLoop;n++)
{
            if (n)
            {
                        val.sival_int=n;
                        sigqueue(IdlePID[n-1],SIGRTMIN,val);
                        if (DEBUG) printf("sigqueue %d¥n",IdlePID[n-1]);
                        nanosleep(&w0,NULL);
            }
            if (CYCLE==1)
            {
            timer_id = create_posix_timer(signum,1,0);
            }
            else
            {
            timer_id = create_posix_timer(signum,0,(1000*1000*1000)/CYCLE);
            }
if (timer_id<0)
{
            fprintf(stderr, "%s: Cannot set posix timer - %s¥n", strerror(errno));
            exit(1);
}
            printf("********************************************** ¥n");
            printf("****   Low priority Client Number %4d:     **** ¥n",n);
            printf("****   Posix Timer Cycle %4d Hz              **** ¥n",CYCLE);
            printf("********************************************** ¥n");
            if (logfp)
            {
                        fprintf(logfp,"********************************************** ¥n");
                        fprintf(logfp,"****   Low priority Client Number %4d:     **** ¥n",n);
                        fprintf(logfp,"****   Posix Timer Cycle %4d Hz              **** ¥n",CYCLE);
                        fprintf(logfp,"********************************************** ¥n");
            }
            count=0;
err = sigwaitinfo(&set,&info);
            do
            {
            err = sigwaitinfo(&set,&info);
            readtime(&t0[count]);
```

```
                        memset(cli_sbuf,0,sizeof(struct timespec));
                        err = write(sockfd,cli_sbuf,sizeof(struct timespec)); sched_yield();
                        if (err<0)
                        {
                                if (errno == EINTR)
                                {
                                        printf("intr ");
                                }
                                fprintf(stderr,"write error %s¥n",strerror(errno));
                                break;
                        }
                        sched_yield();
                len = read(sockfd,cli_sbuf,sizeof(struct timespec)*2);
                readtime(&t3[count]);
                        memcpy(&t1[count],&cli_sbuf[0],sizeof(struct timespec));
                        memcpy(&t2[count],&cli_sbuf[sizeof(struct timespec)],sizeof(struct timespec));
                count++;
                } while ((loop_flag)&&(count<MAXCOUNT));
                cli_sbuf[0] = END_MARK;
                err = write(sockfd,cli_sbuf,1); sched_yield();
        delete_posix_timer(timer_id);

                calc(logfp,&t0[0],&t0[1],"POSIX Timer Interval Time ",0.0);
                calc(logfp,&t0[0],&t1[0],"Send Time                 ",0.0);
                calc(logfp,&t1[0],&t2[0],"nanosleep(100 Micro Sec)  ",100.0);
                calc(logfp,&t2[0],&t3[0],"Receive Time              ",0.0);
                calc(logfp,&t0[0],&t3[0],"Response Time             ",0.0);
        min=1000000000;
        s=avr =max=0;
        memset((void*)his,0,sizeof(his));
        memset((void*)interval,0,sizeof(interval));
        memset((void*)total,0,sizeof(total));
        for(count=1;count<MAXCOUNT;count++)
        {
                adjust(&t0[count-1],&t1[count],&interval[count]);
                adjust(&t0[count],&t3[count],&total[count]);
                adjust(&t1[count],&t2[count],&x);
                        total[count] -= x; total[count] += fabs(x-100);
                        interval[count] = fabs(interval[count] - (double)CYCLE);//
                        total[count] += interval[count] ;
                his[(int)(total[count]+0.5)/10]++;
                avr += total[count];
                if (min> total[count]) min =    total[count];
                if (max< total[count]) max =    total[count];
        }
        avr /= (double)(MAXCOUNT-1);
        for(count=0;count<MAXCOUNT-1;count++)
        {
                        x=total[count]-avr;
                        s+= (x * x);
                }
                s = sqrt(s/(double)(MAXCOUNT-2));
        printf("Total Jitter Time           ¥tmin %8.1f(MicroSec) avr %8.1f(MicroSec) max %8.1f(MicroSec)
S %8.1f¥n",min,avr,max,s);
        if(logfp)
        {
                fprintf(logfp,"Total
Jitter¥t,Min,%8.1f,(MicroSeconds),Avr,%8.1f,(MicroSeconds),Max,%8.1f,(MicroSeconds),Standard
deviation,%8.1f¥n",min,avr,max,s);
                fprintf(logfp,"Time(MicroSeconds),Number of Times¥n",count*10,his[count]);
                for(count=0;count<MAX_HISTORY;count++)
                {
                        if (his[count]) fprintf(logfp,"%d,%d¥n",count*10,his[count]);
                }
                fprintf(logfp,"¥n¥n");
        }

        }

        close(sockfd);
        if(logfp) fclose(logfp);

        return(0);
}
```

```
void main_sigint_handler(int no)
{
printf("main_sigint_handler¥n");
        loop_flag=0;
}
void idle_sigint_handler(int no)
{
        idle_flag=0;
}

static int idle_process(int no)
{
        static struct sched_param myparam;
        double x,y;
       static sigset_t set,oset;
       static siginfo_t info;
        char fname[256];
        FILE * fp;
        register int i,n;
        static complex c1[N],c2[N];

        n=N;

        sprintf(fname,"/tmp/dummy%d",no);
        mlockall(MCL_CURRENT|MCL_FUTURE);
        myparam.sched_priority = sched_get_priority_min(SCHED_RR);
       sched_setscheduler(getpid(),SCHED_RR,&myparam);
        {
        cpu_set_t mask;
        int ret,cpu;
                unsigned int cpusetsize;

                cpusetsize = sizeof(cpu_set_t);

               CPU_ZERO(&mask);
                for(cpu=3;cpu<32;cpu++)
                CPU_SET(cpu,&mask);
              ret = sched_setaffinity(getpid(),cpusetsize,&mask);
              if (ret<0)
              {
                        printf("sched_setaffinity(%08X) is error %s¥n",mask,strerror(errno));
              }
        }
        signal(SIGINT,idle_sigint_handler);
        if (DEBUG) printf("suspend %d(pid=%d)¥n",no,getpid());
        sigemptyset(&set);
       sigaddset(&set,SIGRTMIN);
       if (sigprocmask(SIG_BLOCK,&set,&oset)==(-1))
       {
              fprintf(stderr, "%s: Cannot set sigprocmask - %s¥n", strerror(errno));
              exit(1);
       }

       sigwaitinfo(&set,&info);
        if (DEBUG) printf("wakeup %d¥n",getpid());
        while(idle_flag)
        {
                fp = fopen(fname, "w"); unlink(fname);
                for(i=0;i<n;i++)
                {
                        c1[i].r = c2[i].r = 6 * cos( 6 * PI * i / n)
                                        + 4 * sin(18 * PI * i / n);
                        c1[i].i = c2[i].i = 0;
                }
                small_fft(n,c2);
                small_fft(-n,c2);
                for(i=0;i<n;i++)
                {
                        if (fp) fprintf(fp,"(%f,%f)¥n", c1[i].r , c1[i].i );
                }
                if(fp) fclose(fp);
                sched_yield();
        }
        if (DEBUG) printf("exit %d¥n",getpid());
```

```
		exit(0);
}

static void do_main(int option,unsigned char *host)
{
		pid_t ServerPID;
		pid_t ClientPID;
		int status,i,n;
		union sigval val;

		if (option & IDLE)
		{
			for (n=0;n<TestLoop-1;n++)
			{
				IdlePID[n]=fork();
				if (IdlePID[n]<0)
				{
					fprintf(stderr,"server fork error¥n");
					exit(0);
				}
				if (IdlePID[n]==0)
				{
					idle_process(n);
					exit(0);
				}
			}

		}
		if (option & SERVER)
		{
			ServerPID=fork();
			if (ServerPID<0)
			{
				fprintf(stderr,"server fork error¥n");
				exit(0);
			}
			if (ServerPID==0)
			{
					server(TCP);
				exit(0);
			}
		}
		if (option & CLIENT)
		{
			ClientPID=fork();
			if (ClientPID<0)
			{
				fprintf(stderr,"client fork error¥n");
				exit(0);
			}
			if (ClientPID==0)
			{
					client(host,TCP);
				exit(0);
			}
		}
		if (option & CLIENT)
		{
			waitpid(ClientPID,&status,0);
		}
		if (option & SERVER)
		{
			waitpid(ServerPID,&status,0);
		}
		if (option & IDLE)
		{
			for (n=0;n<TestLoop-1;n++)
			{
				val.sival_int=n;
				sigqueue(IdlePID[n],SIGINT,val);
				if (DEBUG) printf("sigqueue %d¥n",IdlePID[n-1]);
				waitpid(IdlePID[n],&status,0);
			}
		}
		exit(0);
```

```
}

static void intr_handler(int sig,siginfo_t *info, void *dummy)
{
        printf("signal number is %d¥n",sig);
}

int main( int argc, char **argv )
{
        extern int optind;
        extern char *optarg;
        register int c, option = SERVER|CLIENT;
         unsigned char host[256];
         static struct sigaction newact;
         static struct sched_param myparam;

         cpu_clock = getMHz();
         mlockall(MCL_CURRENT|MCL_FUTURE);
         myparam.sched_priority = sched_get_priority_max(SCHED_FIFO);
        sched_setscheduler(getpid(),SCHED_FIFO,&myparam);
         signal(SIGINT,main_sigint_handler);
         sigaddset(&newact.sa_mask,SIGRTMIN);
        newact.sa_sigaction = intr_handler;
        newact.sa_flags = SA_SIGINFO|SA_RESTART;
        if (sigaction(SIGIO, &newact, 0)==(-1))
        {
                fprintf(stderr, "Cannot set sigaction - %s¥n",strerror(errno));
                return(1);
        }
         strcpy((char*)host,SERV_HOST_ADDR);
        while((c = getopt(argc, argv, "DC:asc:p:l:i:")) != EOF)
        {
                switch(c)
                {
                        case 'p':
                                PORT=atoi(optarg);
                                  printf("PORT %d¥n",PORT);
                                break;
                        case 'C':
                                CYCLE=atoi(optarg);
                                  if (CYCLE<=0) CYCLE = 1;
                                  if (CYCLE>=10000) CYCLE = 10000;
                                  printf("CYCLE %d Hz¥n",CYCLE);
                                break;
                        case 'D':
                                  DEBUG =1;
                                break;
                        case 'a':
                                  ASYNC_FLAG |=ASYNC;
                                break;
                        case 'c':
                                option|=CLIENT;
                                strcpy(host,optarg);
                                  printf("address %s¥n",host);
                                break;
                        case 's':
                                option|=SERVER;
                                break;
                          case 'l':
                                  logfp = fopen(optarg, "w");
                                  break;
                          case 'i':
                                  option|=IDLE;
                                  TestLoop=atoi(optarg);
                                  break;

                        default:
                                  printf("¥tD:DEBUG¥n¥t¥tC  Cycle  :default  1000Hz¥n¥ts  server¥n¥tc
client¥n");
                                break;
                }
        }
         make_table(N);
         do_main(option,host);
         make_table(0);
```

```
}
```

### Makefile

```
CMPCMD = cc -c $(CCOPT) $(USRDEF) $(SETSMP) -I .
LIBCMD  = ar ruv
LNKCMD  = cc $(CCOPT)
TRLTP           = trltp
EXTLIB          = -lrt
LNKOBJ          = $(TRLTP).o
TEMPFILES       = core core.*
CFLAGS          = -O    -D_GNU_SOURCE

LDFLAGS         =

$(TRLTP):$(LNKOBJ)
                $(LNKCMD) $(LDFLAGS) -o $(TRLTP)    $(LNKOBJ) $(LNKLIB) $(EXTLIB) -lm

.c.o:
                $(CMPCMD) $(CFLAGS) $<

$(LNKOBJ):      $(SYSINC) Makefile

clean:
                -rm -f $(TEMPFILES) $(LNKOBJ) $(TRLTP)
```

## Bibliography

(1) “The Use of POSIX in Real-time Systems, Assessing its Effectiveness and Performance”
Kevin M. Obenland, The MITRE Corporation, 1820 Dolley Madison Blvd. McLean, VA 22102, obenland@mitre.org
http://www.mitre.org/work/tech_papers/tech_papers_00/obenland_posix/obenland_posix.pdf

(2) “An Empirical Evaluation of OS Support for Real-time CORBA Object Request Brokers”
David L. Levine, Sergio Flores-Gaitan, and Douglas C. Schmidt flevine,sergio,schmidtg@cs.wustl.edu
Department of Computer Science, Washington University St. Louis, MO 63130, USA
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.43.2228&rep=rep1&type=pdf