リアルタイム OS を総合的に評価する
Two-way Request Latency Test Program の設計
（POSIX-OS の性能評価）

Rev	Date	Description			
	Issue		Author	Check	App
Rev	Date	Description			
	Issue		Author	Check	App
Rev	Date	Description			
0	Issue	Professional Service	Author	T.Oshima	Check	App
Title	POSIX-OS の性能評価		No		Revision
					Page 1

始めに

本稿では、リアルタイム OS を総合的に評価する Two-way Request Latency Test Program の設計について述べる。

目的

本プログラムの目的は、リアルタイム OS の断片的な性能ではなく、実使用環境に則した性能評価であり、断片的な性能を数値化するものではない。

従来、リアルタイム OS を評価する数値として、メーカーが公表しているものに、応答速度(Response Time)が存在する。

この応答速度も、メーカーにより様々な箇所の値が存在し、その解釈も異なっている。

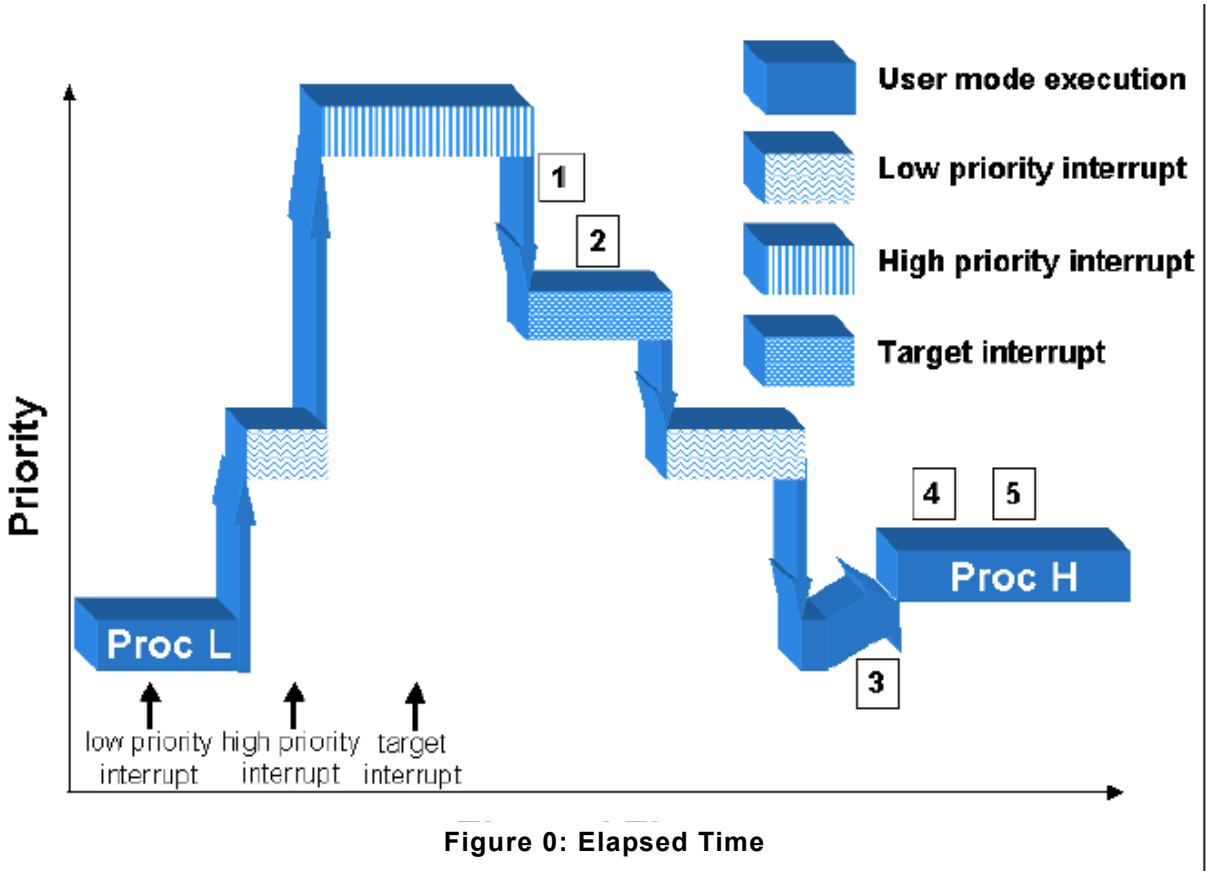


Figure 0: Elapsed Time

1. 割込みコントローラが割り込みを通知し、ＣＰＵが割り込み例外を通知する。
2. 割込みルーチンが実行され、割り込みを待っているプロセス（ターゲットプロセス）が呼び起こす。
3. 現在実行しているプロセスは中断し、そして、ターゲットプロセスを実行するために、コンテキストスイッチが行なわれる。
4. 割り込みを待ってブロックされていたターゲットプロセスは、カーネル空間からユーザ空間にコンテキストスイッチされる。
5. ターゲットプロセスはユーザ空間で実行される。

（図は、Concurrent Computer 社が公表している Application Response Time）

しかし、エンドユーザが必要とする応答時間は、イベントが発生しそのイベントの発生をアプリケーションプログラムが認知するまでの時間であり、個別の時間を評価することは、エンドユーザにとって意味は小さいと考える。

ここでは、AEGIS で使用されたベンチマーク試験[1][2]を参考に、リアルタイム OS の評価プログラムの設計を行うことを考える。

参考文献(1),(2)のベンチマーク試験の分析

参考文献(1)のベンチマーク試験では、下記の６つの項目を個別のベンチマークプログラムとして実施している。

Benchmark	Description	Aspect tested	Parameters
Timer Jitter	Create a periodic thread and measure the deviation between desired and actual expiration	Measures the response time of the operating system	Timer period: (1,10,100 ms)
Response	Execute a fixed processing load and measure its execution time over a number of runs	Determine if a thread can respond in a deterministic fashion	Type of processing: (add,copy,whetstone)
Bintime	Call a time of day clock and measure interval between calls	Measures the maximum kernel blocking time	None
Sync	Measure the latency of thread to thread or process to process synchronization	Measures the context switching time between threads and processes	Type of semaphore: (POSIX named/unnamed semaphore, pthread mutex, lynx semaphore); process to process or thread to thread
Message passing	Measure the latency of sending data form thread to thread or from process to process	Measures the possible throughput of data between processes and threads	Data buffer size; process to process or thread to thread
RT Signals	Measure the latency of real-time signals between two processes	Measures the latency of POSIX real-time signals	None

参考文献(2)のベンチマーク試験では、下図に示すように、IP を使用したエンドーエンドのベンチマークプログラムとして実施している。

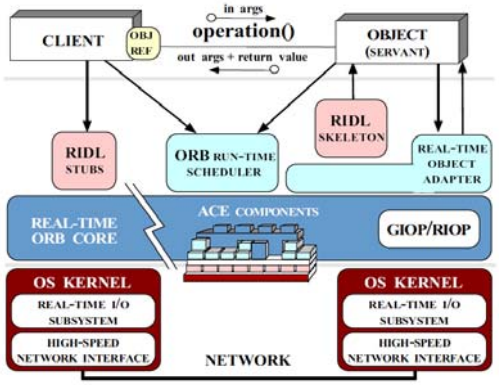


Figure 1: Components in the TAO Real-time ORB Endsystem

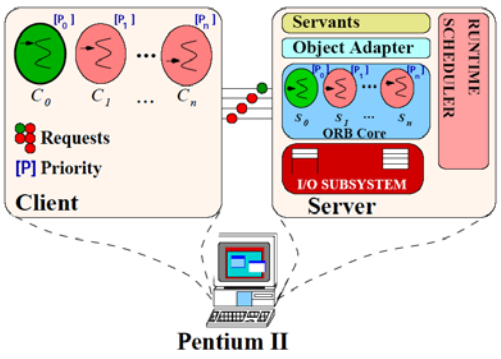


Figure 2: ORB Endsystem Latency and Jitter Test Configuration

何れのベンチマーク試験の項目も、今後の防衛関係のプログラム開発に必要な項目で有ることに異論はないと考える。

しかし、この AEGIS ベンチマーク試験と同様な試験プログラムを行い、評価するためには相応の知識と環境を用意するコストが要求されるため､本設計では､これらの試験を簡易的な、単一プログラムとして開発することを考える。

参考文献(1)に示された最初の３本のベンチマーク（Timer Jitter、Response と Bintime）は、オペレーティングシステムの決定論(Determinism)の計測に使用する。
リアルタイムシステムにおいて、最も重要な性能は、決定論(Determinism)であり、同じ処理であれば、常に同じ処理時間を保証する性能のことである。
残りの３本のベンチマークはオペレーティングシステムの同期、メッセージパッシングとリアルタイム・シグナルをテストしている。
リアルタイムシステムは、最小化された同期と通信待ち時間は非常に重要で、リアルタイム処理の待ち時間が最小化され、全体のオーバーヘッドが、小さいことが重要である。

Timer Jitter テスト
参考文献(1)の Timer Jitter テストの構造を下図に示す。
試験は、インターバルタイマを作り、その繰り返し時間の時刻をタイムスタンプで記録する。ジッターは、望ましい時限完了時間の偏差と定義される。
このタイムスタンプには、高い精度時刻機構を与える POSIX clock_gettime() 関数を使う。

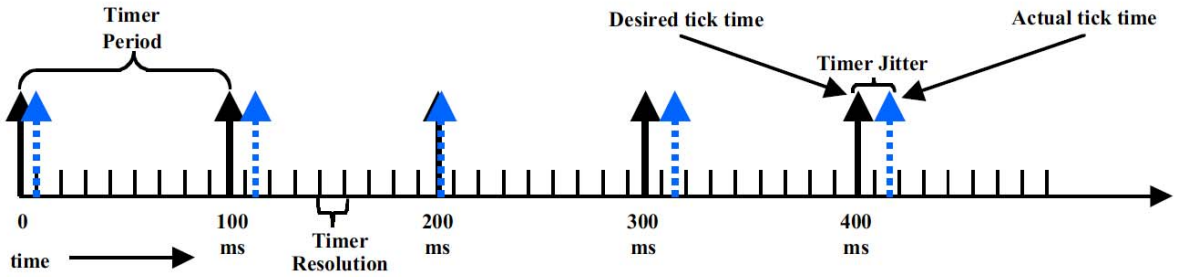


Figure 3: Timer Jitter

この機能は、POSIX 準拠のすべてのリアルタイム OS に備わっている機能であるため、繰り返し時間を高速にすることで、現行機種のベンチマーク試験にそのまま流用できると考えられる。(参考文献は 1990 年代の計算機であるため 2010 年の速度に合わない。)
もし、タイマーがハードウェアによって実現される精度の高いものであれば、このジッターのヒストグラムは、正の値と負の値が正規に分布し、それがソフトウェアによって実現される精度の良くないものであれば、おそらく正の値にシフトしているはずである。
clock_gettime()は、POSIX の定義であるが、システムコールであるためコンテキストスイッチを伴い、オーバーヘッドが存在する。しかし現在の x86 プロセッサでは、CPU に与えられたクロックをカウントする命令 tscrd が存在し、ボードに実装されたクロックチップに左右される事のない、より正確な測定が可能である。
この tscrd を使用した場合には、OS のオーバーヘッドは存在しない。また、linux においては異なる CPU 間で各クロックを同期させる機構を標準で持っているため、ベンチマークテストには最適である。

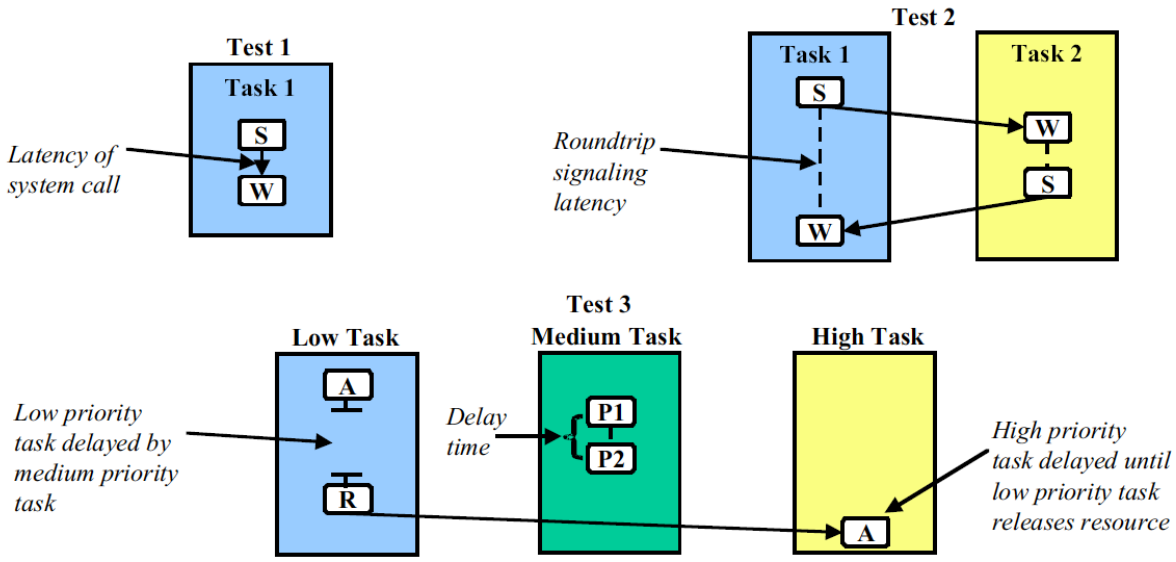
Response テスト
２番目の決定性ベンチマーク（Response）は、１０ミリ秒固定ブロックの実際の実行時間を測っている。　参考文献(1)におけるベンチマーク試験では、加算,コピー, 合成ウエットストーンベンチマークの３種を行っている。
このベンチマークで、問題にしている処理の実行時間変動は、主に以下の理由で発生する。

- 主記憶のページングの遅れ
- キャッシュのミスヒット
- 数値演算におけるパイプラインストール現象
- 優先度プロセス切り替えに伴う、コンテキストスイッチ時間の副作用
- システムクロックによるサービスのオーバーヘッドとその副作用

したがって、処理時間とキャッシュのミスヒットを多発させる fft プログラムを使用することにより、この影響を調べることが出来ると考える。

Bintime テスト
最大のカーネルブロッキング時間の測定。
参考文献(1)のベンチマークプログラムは、高優先順位リアルタイムスレッドが繰り返し Time of Day システムコールを行い、それぞれの呼び出しによって必要とされる時間を計算している。
それぞれの呼び出しによって必要とされる時間は、カーネルでブロックされ、システムコールを行なう度に成り立つ。
システムコールは常にループによって行われ、絶え間が無いため、ベンチマークによって報告された最大時間と平均待ち時間の間の偏差は､カーネルでブロックされた最大時間を与える。

Sync テスト
参考文献(1)で行われている、３つの異なった同期テストを下図に示す。
Test1 は、単一タスクで（Ｓ）を行い、そして次にセマフォ（Ｗ）を待つ。
このテストはセマフォシステムコールの待ち時間を測る。
Test2 は、２つのスレッド間で単一のセマフォの通知時間を計測する。
異なったタスクはスレッドあるいはプロセスでのいずれかである。
Test2 で得られた roundtrip 時間の半分から、Test1 の時間を引き算することによって、コンテキスト切替え時間を決定するために使うことができる。
最後のテストは､オペレーティング･システムが優先性反転(プライオリティインバージョン)を扱う能力を評価する。



低レベル優先度タスクが（Ａ）高優先度タスクが、後に必要するリソースを獲得するとき、プライオリティインバージョンが発生する。
図では、高優先度タスクが、リソースにアクセスする事を阻止され、そして、独立した中間の優先度タスクが CPU を独占しているため、低優先度タスクがリソースを解放することが無期限に遅れている。
この問題を解決する１つの典型的な方法は、低レベル優先度タスクが実行できるように、高優先度タスクの優先度を継承することを可能にし、情報資源Ｒをリリースする事である。
なお、このプライオリティインバージョンは、対応機能を持っていない場合にデッドロックを発生させる。

Message passing テスト
　Message passing テストは、同じプロセスあるいは異なったプロセス間で、POSIX メッセージキューを使って、データの待ち時間とスループットを計測する。

RT Signal テスト
RT Signal テストは、POSIX リアルタイムキューの待ち時間を測定する。

ORB テスト
Figure 4 は、参考文献(2)のベンチマークテストがどのように、 C0 から Cn クライアントを使うか示す。
　最高優先クライアント、すなわち C0 は、デフォルト OS リアルタイム優先度 P0 で実行され、２０ヘルツで処理を実行し、毎秒２０回の CORBA two way コールを呼び出す。
　残りのクライアント C1～Cn は、異なったより少ない OS スレッドプライオリティ P1～Pn を持っていて、１０ヘルツの処理を実行し、毎秒１０回の CORBA two way コールを呼び出す。

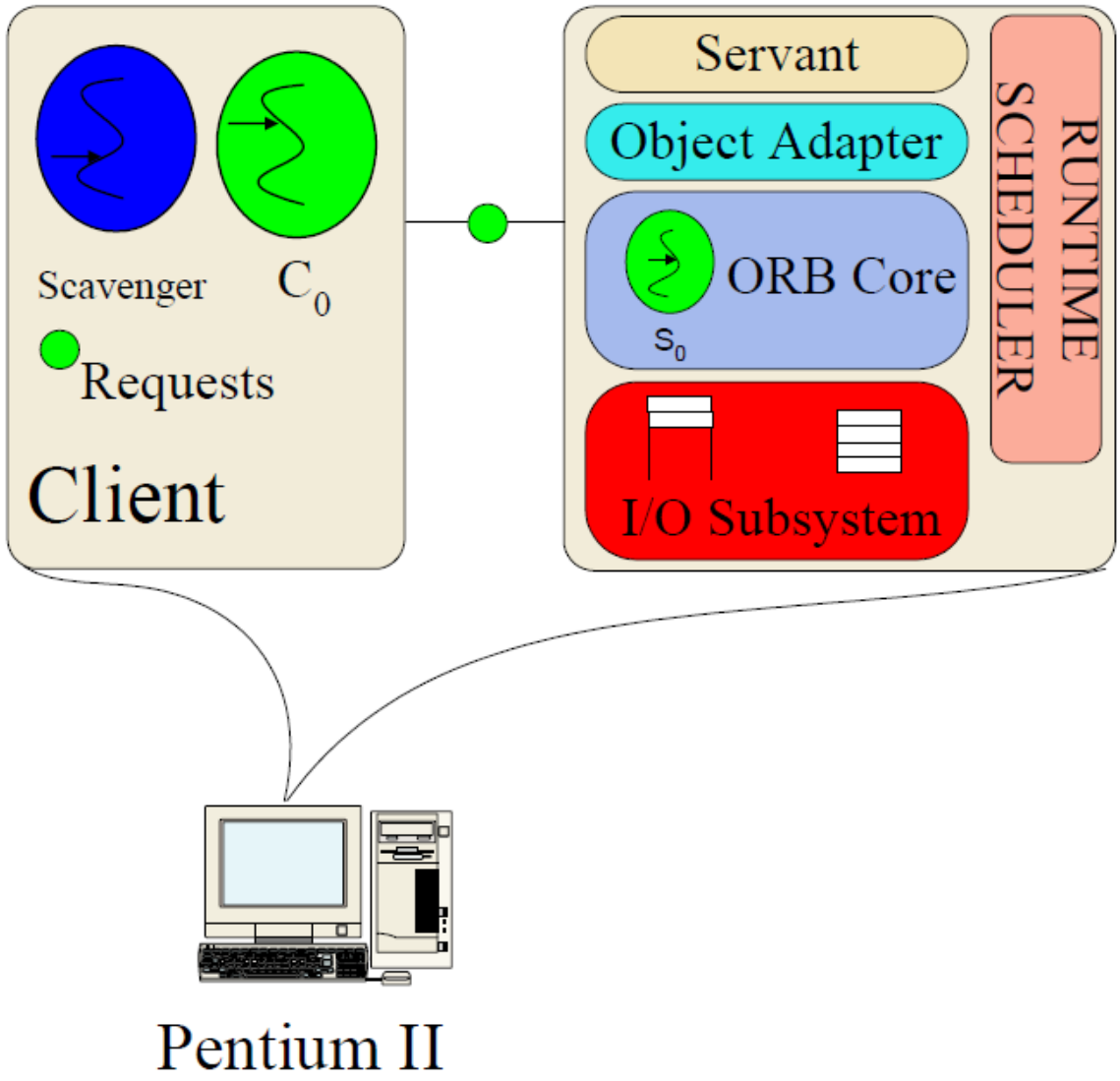


Figure 4:ORB Client/Server(C/S) Request Utilization Benchmark Configuration

ベース試験プログラム

コンカレントで、現在用意されている、試験プログラムは、いくつか存在するが、本稿に対して有用なプログラムは、以下のものがあると考えられる。

項目	プログラム名
Timer Jitter	timer.c, tswitch.c
Response	fft.c
Bintime	timer.c
Sync	sem.c, tswitch.c
Message passing	mq.c
RT Signals	sigq.c

Figure 5 :**ベースとなる試験プログラム**

timer.c

1000Hz の POSIX タイマーを生成し、その統計的な最小値、平均値、最大値を表示するとともに、nanosleep()関数の精度を試験する。

POSIX 仕様のリアルタイムカウンタ clock_gettime()を使用し、μ 秒精度の計測を行う。

参考文献(1)で行われている Timer Jitter テストと Bintime テストに相当する。

tswitch.c

1000Hz POSIX タイマーを生成し、その統計的な最小値、平均値、最大値を表示するとともに、２つの p スレッドの mutex 起動時間を試験する。

POSIX 仕様のリアルタイムカウンタ clock_gettime()を使用し、μ 秒精度の計測を行う。

参考文献(1)で行われている Timer Jitter テストと Bintime テストに加え Sync テストの Test2 に相当する。

sem.c

POSIX セマフォのサンプルプログラム。

参考文献(1)で行われている Sync テストの Test1 に相当する。

mq.c

POSIX メッセージキューのサンプルプログラム。

参考文献(1)で行われている Message passing テストに相当する。

sigq.c

POSIX リアルタイムシグナルのサンプルプログラム。

参考文献(1)で行われている RT signal テストに相当する。

実行例 1

同一ハードウェア(デジタルロジック)上にて上流ベンダーおよび RedHawk でそれぞれタイマーハンドラーの起動間隔を測定するプログラムを約 10 秒間実行し、リアルタイム性能を評価した結果。(優先度を変更するため、root アカウントにて実行)

- Upstream Vender 5.4 kernel 2.6.18-164 (SMP パッチ適用)
- RedHawk 5.4 Kernel 2.6.31.6 (SMP / PREEMT / CCUR 独自パッチ適用)

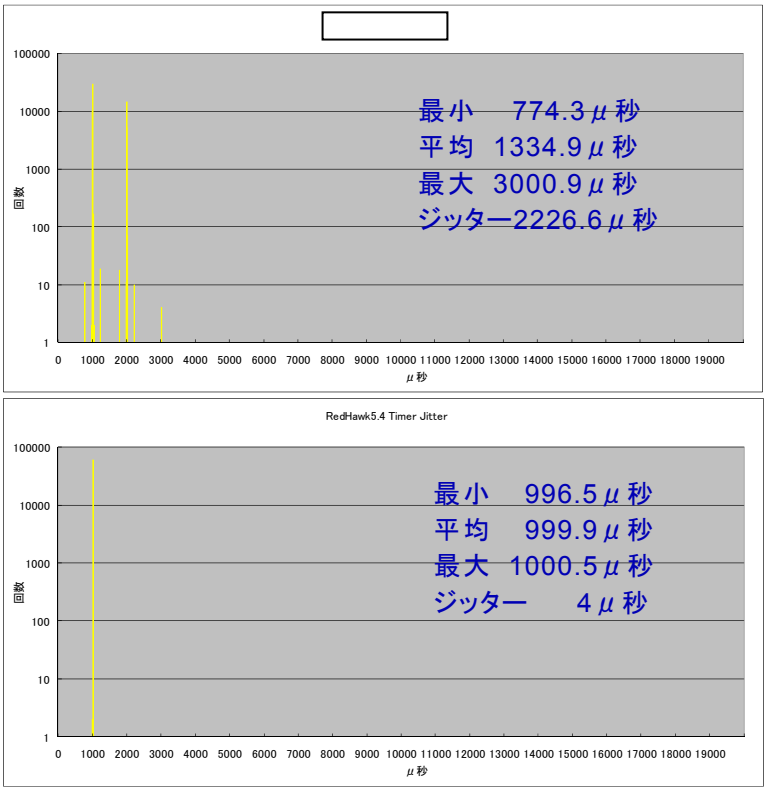


Figure 6 :実行例 1

実行例２

同一ハードウェア(パナソニック)上にて Ubuntu および RedHawk でそれぞれタイマーハンドラーの起動間隔を測定するプログラムを約 30 分間実行し、リアルタイム性能を評価した結果。(優先度を変更するため、root アカウントにて実行)

- Ubuntsu 9.04 Kernel 2.6.29.6 (SMP / PREEMT / PREEMT_RT パッチ適用)
- RedHawk 5.4 Kernel 2.6.31.6 (SMP / PREEMT / CCUR 独自パッチ適用)

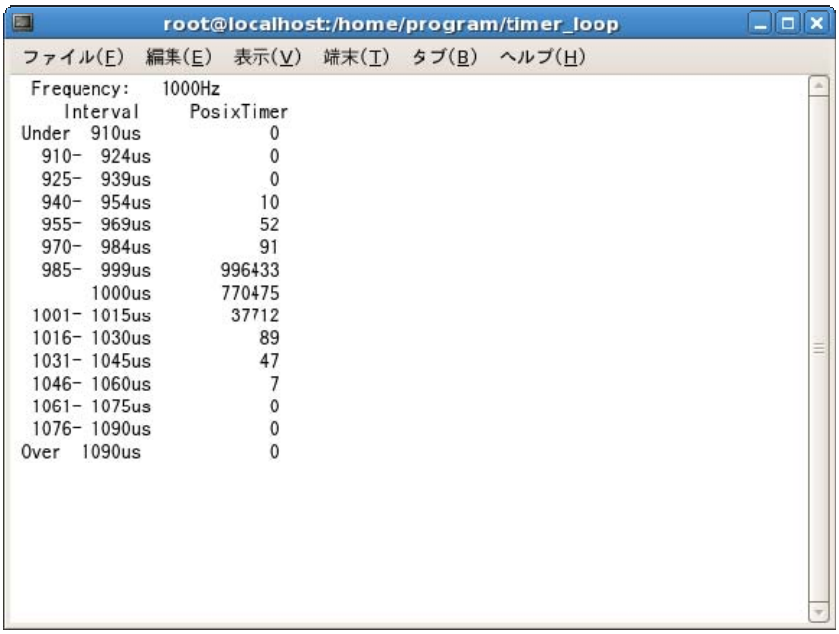


Figure 7: 実行例２

この様に、RedHawk、標準 kernel.org と PREEEMPT_RT パッチを適用した OS との性能差を読み取ることが出来るため 1000Hz のタイマー試験において、ヒストグラムと、標準偏差は有用であることが確認された。

Two-way Request Latency Test Program の設計と評価

AEGIS に必要とされたリアルタイム OS の性能を総合的に評価する際に重要な点として、ORB 試験に示されるように、TCP/IP の応答性能があり、その決定性にはジッターを評価している点がある。

本稿で設計する Two-way Request Latency Test Program (以下 TRLTP)は、この点を踏まえ、以下のように設計を行う。

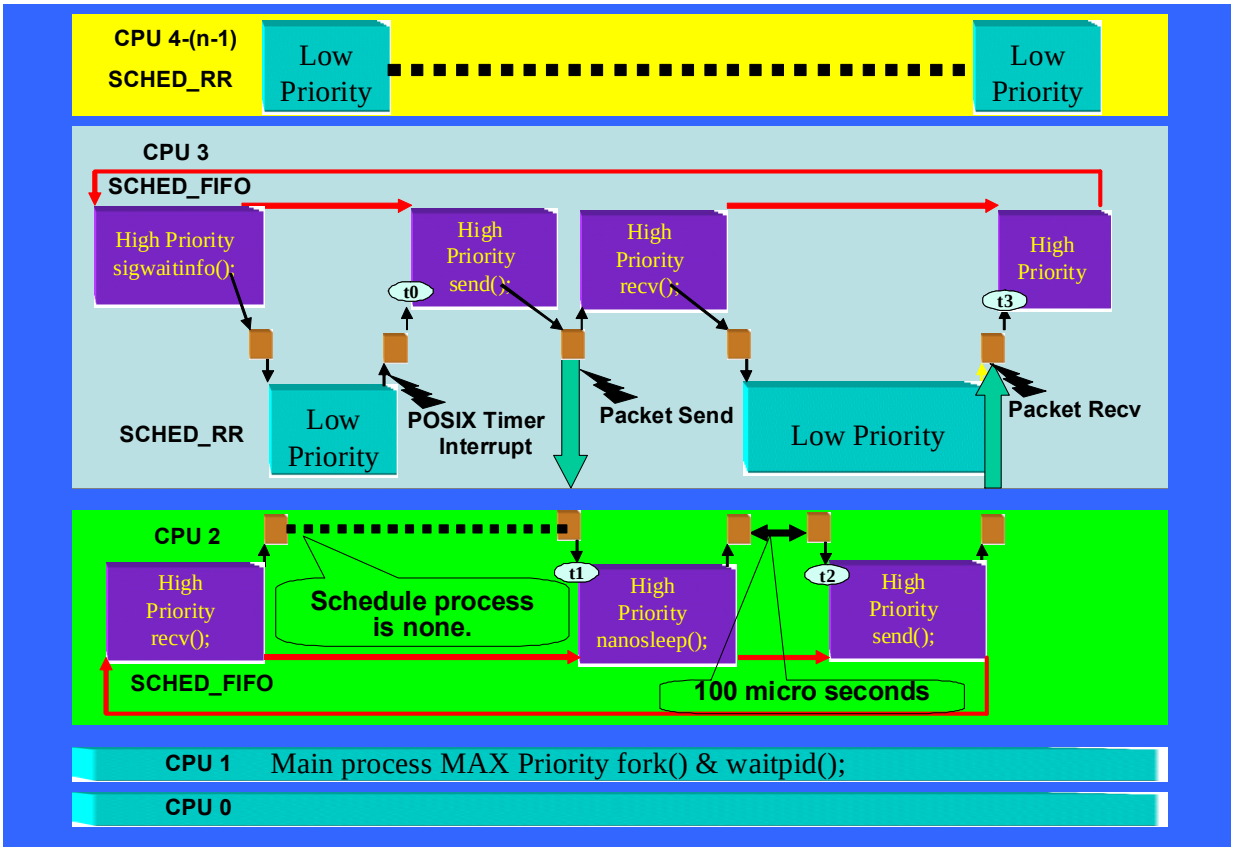


図 1　TRLTP プログラムの構成

プログラムは、4 つのプロセスを生成し、以下の動作を行う。

●main プロセス 起動プロセス
server,client,idle プロセスを起動し、各プロセスの終了を待つ。

1. SCHED_FIFO の MAX 優先度

●client プロセス：本プログラムの主プロセス

1. 1000Hz の POSIX タイマーを生成し、そのジッターを計測する。(t0)
2. 1000Hz の POSIX タイマーに同期して、server プロセスに、TCP/IP によって 8 バイトのヌルデータを送信し、server プロセスからの応答を待ち、その時点でのタイムスタンプ計測(t3)を行う。
3. SCHED_FIFO の MAX 優先度－1

●server プロセス：client プロセスからの依頼を受けて実行される高優先度補助プロセス。

1. client プロセスからの受信を select で待つ。

2. select からの復帰後受信を行い、その時点でのタイムスタンプ計測(t1)を行う。
3. 100μ 秒の POSIX nanosleep()を行う。
4. nanosleep()からの復帰後、その時点でのタイムスタンプ計測(t2)を行う。
5. t1,t2 のタイムスタンプをデータとして、client プロセスに送信する。
6. SCHED_FIFO の MAX 優先度－２

●idle プロセス

client プロセスからの依頼を受けて実行される低優先度補助プロセス。
プログラムオプションで、実行を繰り返すたび、プロセスが増えて行く。

1. client プロセスからの POSIX リアルタイムシグナルを待つ。
2. ファイルをオープンし、65536 ポイントの倍精度 fft 計算を行い、結果をファイルに書き出し、消去する。
3. ２の処理を指示があるまで繰り返す。
4. SCHED_RR の MIN 優先度

プログラムでは、以下の値の統計データを、最小(μ 秒) 平均(μ 秒) 最大(μ 秒) 偏差として出力し、オプション-l ファイル名で指示した場合、そのヒストグラムデータと併せてファイルに記録する。

ただし、インターバルタイマー時間と nanosleep の時間の偏差の計算には、平均を中心値とはせずに、目標値を使用する。

POSIX Timer Interval Time	t0[n-1]と t0[n]間の値
Send Time	t0[n]と t1[n]間の値
nanosleep(100 Micro Sec)	t1[n]と t2[n]間の値
Receive Time	t2[n]と t3[n]間の値
Response Time	t0[n]と t3[n]間の値
Total Jitter	t0[n]と t3[n]間の値から t1[n]と t2[n]を減算し、 t1[n]と t2[n]の差から 100μ 秒を減算した絶対値を加える。 同様の操作で、t0[n-1]と t0[n]間の絶対値を加算した値 これは、偏差を求める際に、平均からの差ではなく、目標値との差を表現するための処置である。

なお、プログラムはすべて、POSIX-API を使用し、OS に依存した命令を使用していないが、CPU 割り当てのみ、Linux 共通の非 POSIX-API を使用している。
（注：POSIX-API には CPU 割り当ての定義がないため、sched_setaffinity()を使用する）

割り当て無し	CPU0	通常は、デーモンなどを実行する。
main プロセス	CPU1	SCHED_FIFO の MAX 優先度
server プロセス	CPU2(エコーバック)	SCHED_FIFO の MAX 優先度-2
client プロセス	CPU3(計測ターゲット)	SCHED_FIFO の MAX 優先度-1
idle プロセス	CPU3,4,5,6,7...31	SCHED_RR の MIN 優先度

意味としては、
①計測プロセスと同じ CPU に負荷プロセスを１つ割り当る。
②エコーバックには、最も良い状態を割り当てる

また、RedHawk RCIM のカウンタ値との差を排除するために、CPU 内蔵のクロックカウンタを tscrd 命令で読み出している。

ソースコード
以下に、すべてのソースコードを示す。

```
/*
 * ****************************************************************************
 *   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_MONOTONIC,&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_MONOTONIC,&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_MONOTONIC,&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_MONOTONIC,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_MONOTONIC,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);
}
```

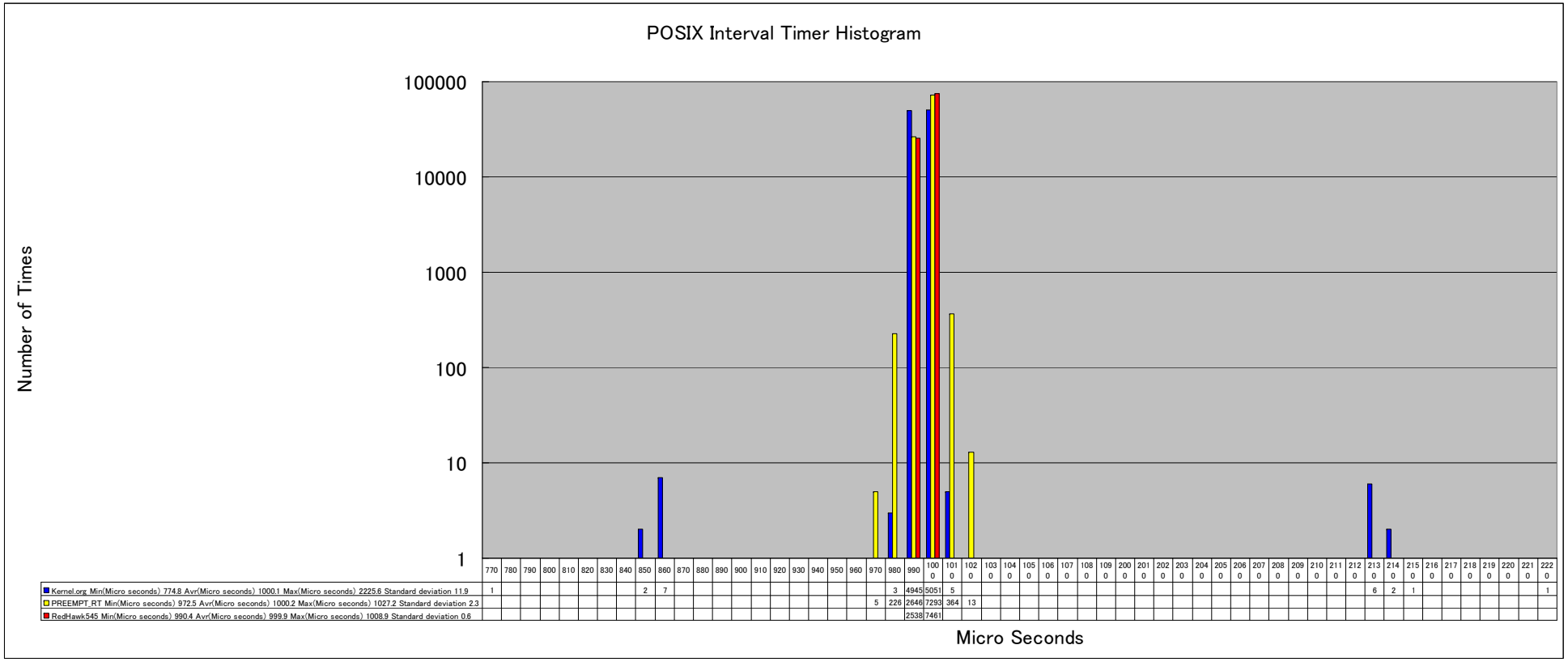
Makefile

```
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)
```

TRLTP ベンチマーク試験の結果

以下に、Dell T5500 4-Core 2-CPU メインメモリ 6G バイトで行った
①kernel.org(RHEL,SUSE 等),
②PREEMPT_RT 適用 Linux(MRG,SLRT,Ubuntu 等),
③RedHawk5.4.5
の結果を示す。
結果は、すべて、同一マシン、同一バイナリオブジェクト、64 ビット OS の性能であり、時間の単位は μ 秒である。また、低優先度プロセスは 15 個存在する状態で行った。

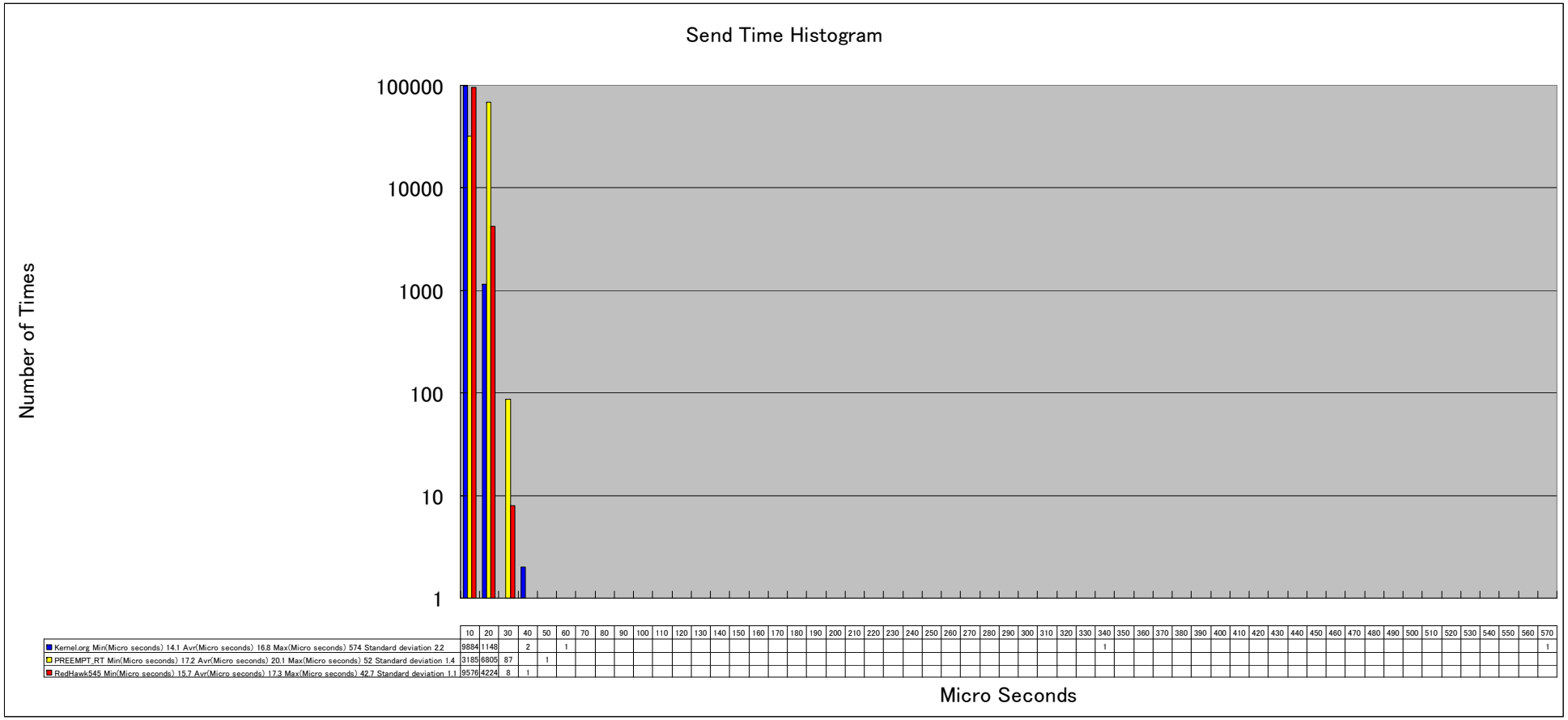
POSIX Interval Timer ヒストグラム



POSIX Interval Timer は、sigwaitinfo()実行中のプリエンプション時間を計測している。
RedHawk と PREEMPT_RT は、ほぼ同等の性能を示しているが、ヒストグラムをみると RedHawk のジッターがより小さい。
プリエンプトパッチが適用されていない、Kernel.org では、同一 CPU に割り当てられた、低優先度プロセスからの影響を大きく受けているため、ジッターが大きいことが示されている。

	最小	平均	最大	偏差
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

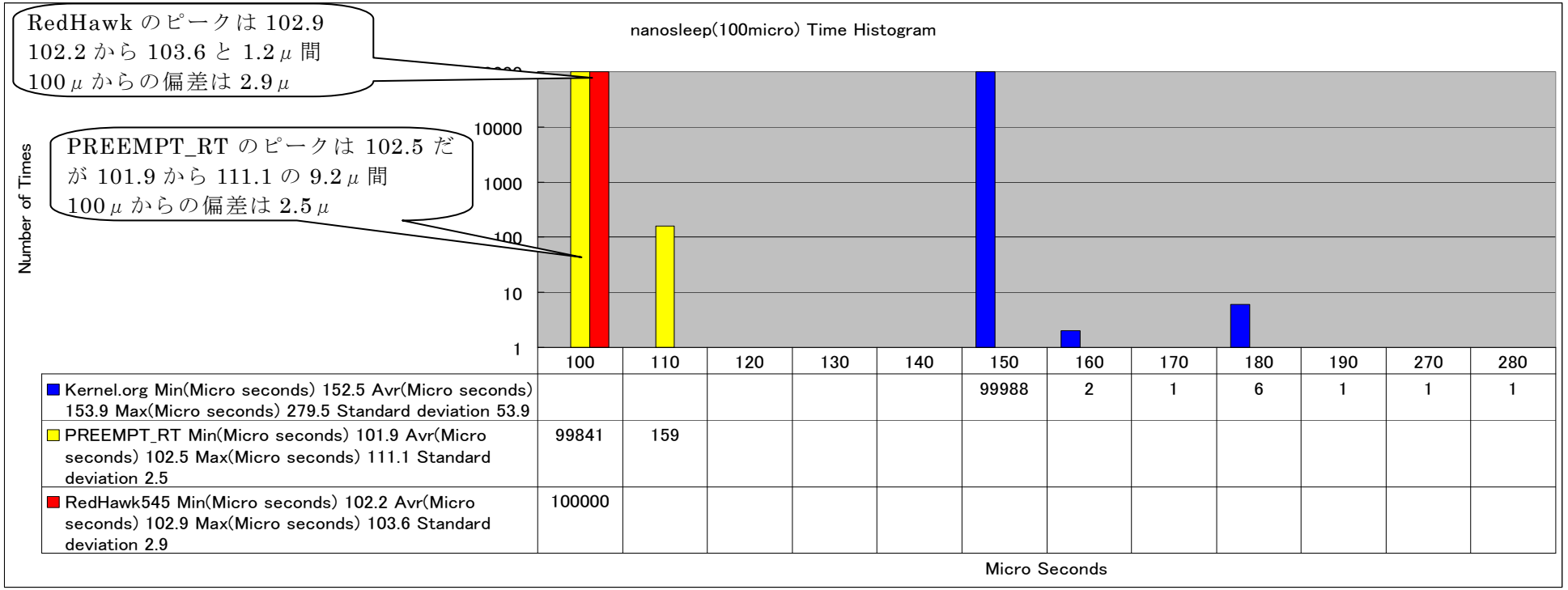
Send Time ヒストグラム



送信時間は、エコーバックプロセス(server)に対する、送信時間であるが、送信プロセスと同じ CPU に負荷プロセスが存在するためか、あるいは IRQ デーモンのスケジュール遅延か、Kernel.org ではジッターが非常に大きく見える。

	最小	平均	最大	偏差
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

nanosleep ヒストグラム



nanosleep の結果は、非常に OS の性格を現している、設計上無負荷であるにもかかわらず、kernel.org では、100μ 秒の要求に対して、150μ 秒以上の時間でしか戻ってこないため、100μ 秒の使用に耐えない。

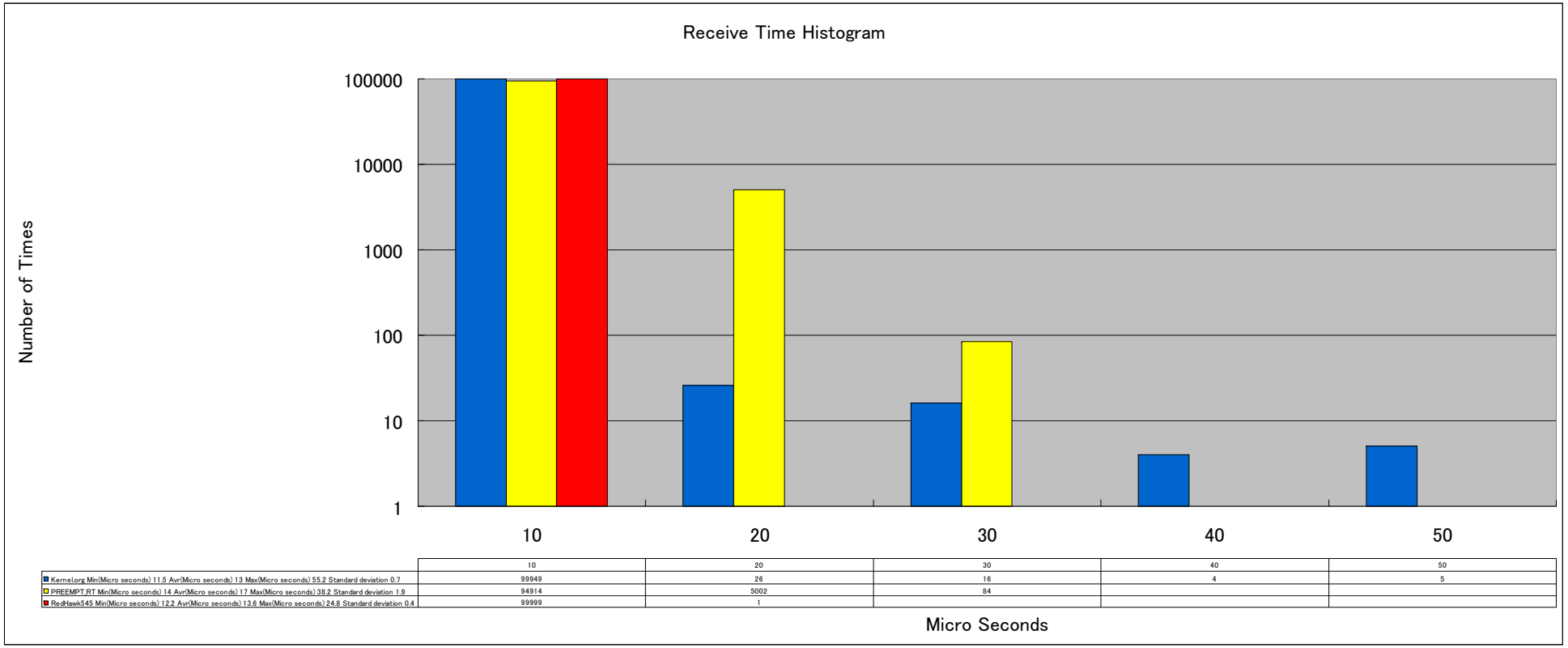
しかし、kernel.org に対して PREEEMPT_RT パッチを適用した OS である PREEMPT_RT では、kernel.org に比較して、非常に良い結果を示している。kernel.org では、平均値で 2.5μ のオーバーヘッドで、RedHawk では 2.9μ 秒のオーバーヘッドが存在する結果となった。このグラフと偏差を比較すると、偏差では、PREEMPT_RT が良い性能を示しているのにヒストグラムでは PREEMPT_RT より RedHawk の分散が少ない。

これは、目標値 100μ 秒に対しての分散を求めているため、PREEMPT_RT の方が良い結果になるためである。

平均に対しての分散を求めると RedHawk の分散が少ない結果になるが、nanosleep に関しては、PREEMPT_RT が良い性能を示していると言える。

	最小	平均	最大	偏差
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

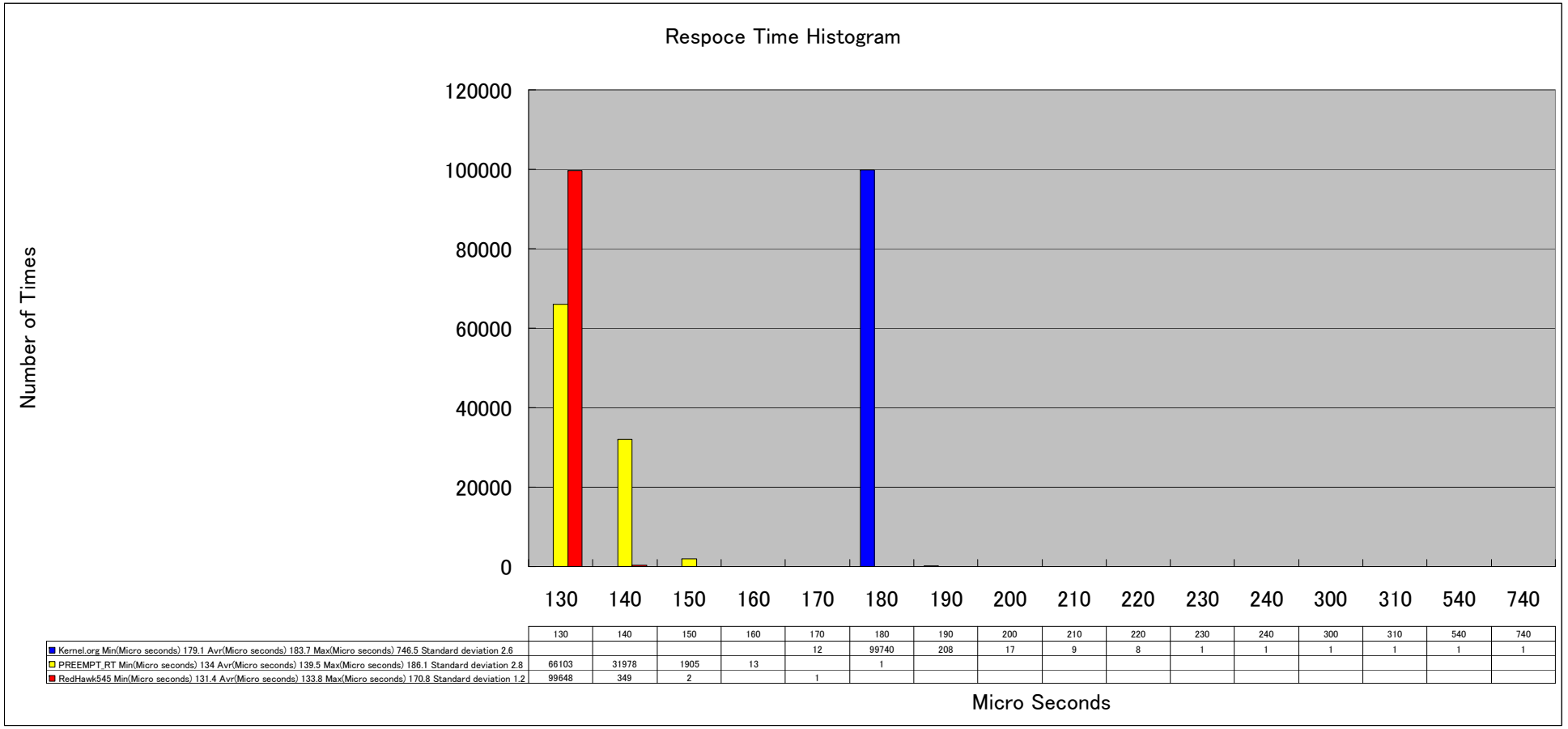
Receive Time ヒストグラム



受信時間は、エコーバックプロセス(server)からの応答遅延を現している。送信時間に比べると OS 間に差異は少ない。
これは、送信側の CPU が無負荷であることが原因であると考えることが出来る。
したがって、先の送信結果と併せて考えると、送信時に送信終了待ちが存在し、同一 CPU に優先度の低いプロセスが存在すると、優先度の低いプロセスが送信終了まで、実行されプリエンプトパッチが適用されていないと、送信遅延を引き起こす事となることが理解できる。

	最小	平均	最大	偏差
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

Response Time ヒストグラム

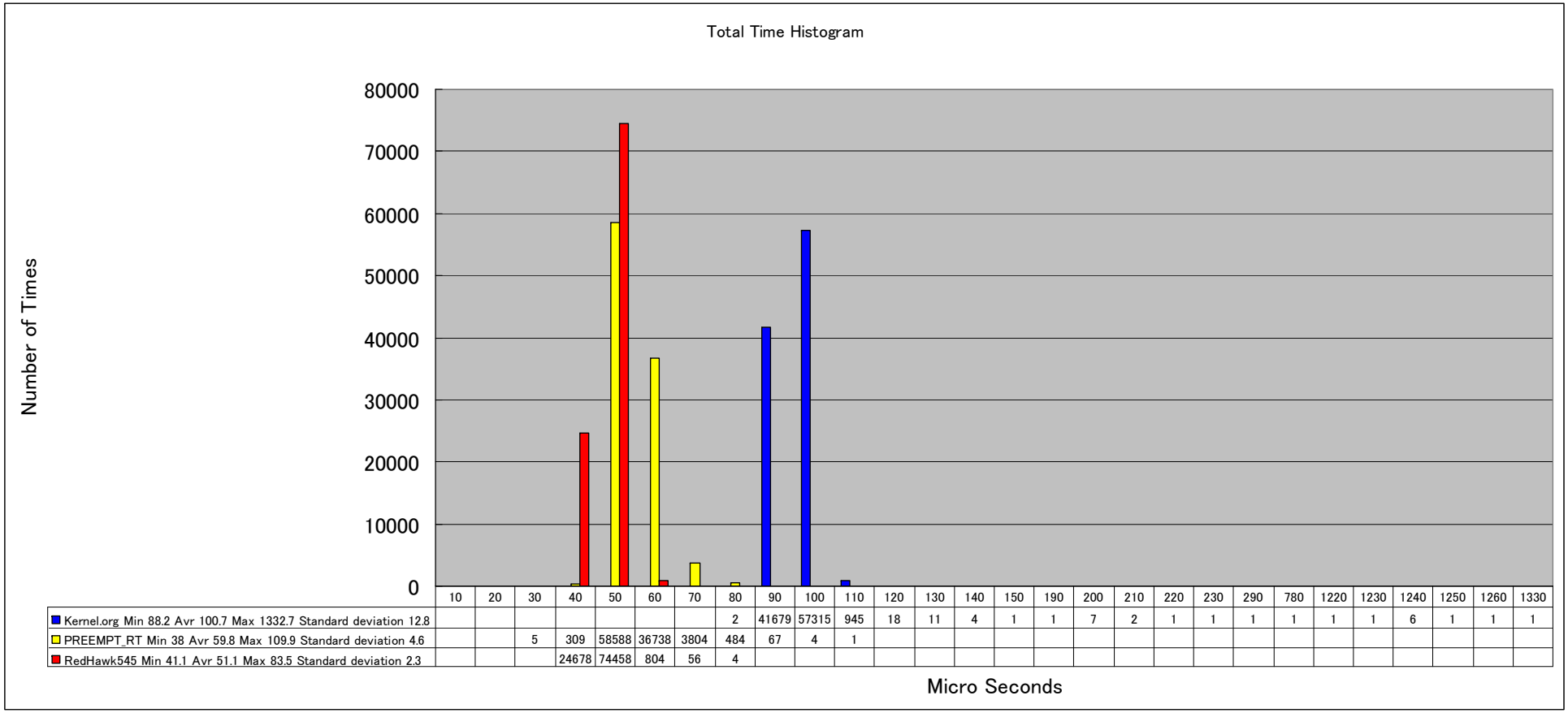


	130	140	150	160	170	180	190	200	210	220	230	240	300	310	540	740
Kernel.org Min(Micro seconds) 179.1 Avr(Micro seconds) 183.7 Max(Micro seconds) 746.5 Standard deviation 2.6					12	99740	208	17	9	8	1	1	1	1	1	1
PREEMPT_RT Min(Micro seconds) 134 Avr(Micro seconds) 139.5 Max(Micro seconds) 186.1 Standard deviation 2.8	66103	31978	1905	13		1										
RedHawk545 Min(Micro seconds) 131.4 Avr(Micro seconds) 133.8 Max(Micro seconds) 170.8 Standard deviation 1.2	99648	349	2		1											

これは、送信、nanosleep(100μ秒)、受信のすべての結果である。
最低でも、５回のシステムコール、５回の再スケジュール時のジッターを含んでいる。

	最小	平均	最大	偏差
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

Total Time ヒストグラム



上記総ての時間を数値化した結果であり。この数値だけで OS の性能が端的に現されている。

	最小	平均	最大	偏差
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

このように、トータルジッターだけを見てリアルタイム総合性能を評価できることが理解でき、本プログラムの設計が正しいことを示している。

まとめ

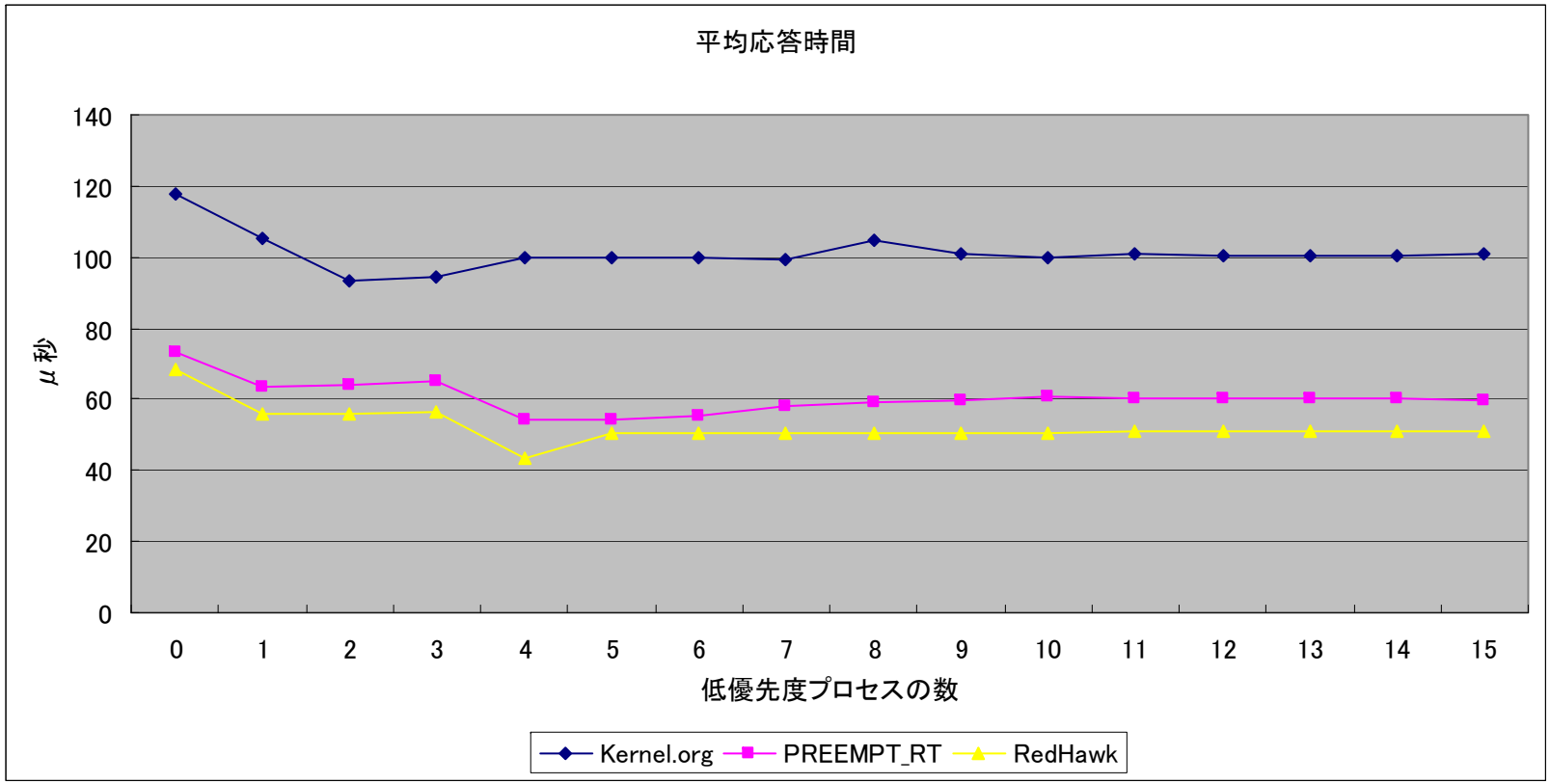
以下に、低優先度プロセスの数を増やした場合の、最小、平均、最大、偏差の値を示す。

トータル応答時間

最小			平均			最大			偏差		
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

リアルタイム性能の尺度として、デターミニズム（決定性）、つまり“ジッターが小さい事“と考えると、本 TRLTP 試験の偏差が小さいこと考えることが出来る。

したがって①RedHawk、②PREEMPT_RT の順でリアルタイム性能が高いことが示され、kernrl.org にはリアルタイム性が皆無（決定性なし）であるとの結果が示されている。



上のグラフは平均応答時間を示したものであり、性能は異なるが同じ傾向を示していることが読み取れる。

つまり、低優先度プロセスの数を増やして行くと、平均的な応答時間は一定の値に近づき、無負荷状態からの増加ではなく減少にあるという、一見矛盾した結果になっている点である。

これは、プロセスの数が増加するとコンテキストスイッチや割り込みの回数が増加し、カーネル内でのスケジューリング回数が増加する事に原因がある。

すなわち、“カーネルに対する要求回数”＝“再スケジュール“であることで、高速に応答することになる。

この事実は、OS の性能評価に注意が必要であることを示している。つまり、むやみに負荷を増やすと、その性能を見誤ることになりかねないと考えることが出来るからである。

右のグラフは、平均応答時間に対するジッターである。

プロセス数にかかわらず、RedHawk は、３μ秒以下の良い性能を示しているが、Kernel.org は相関関係が無い。

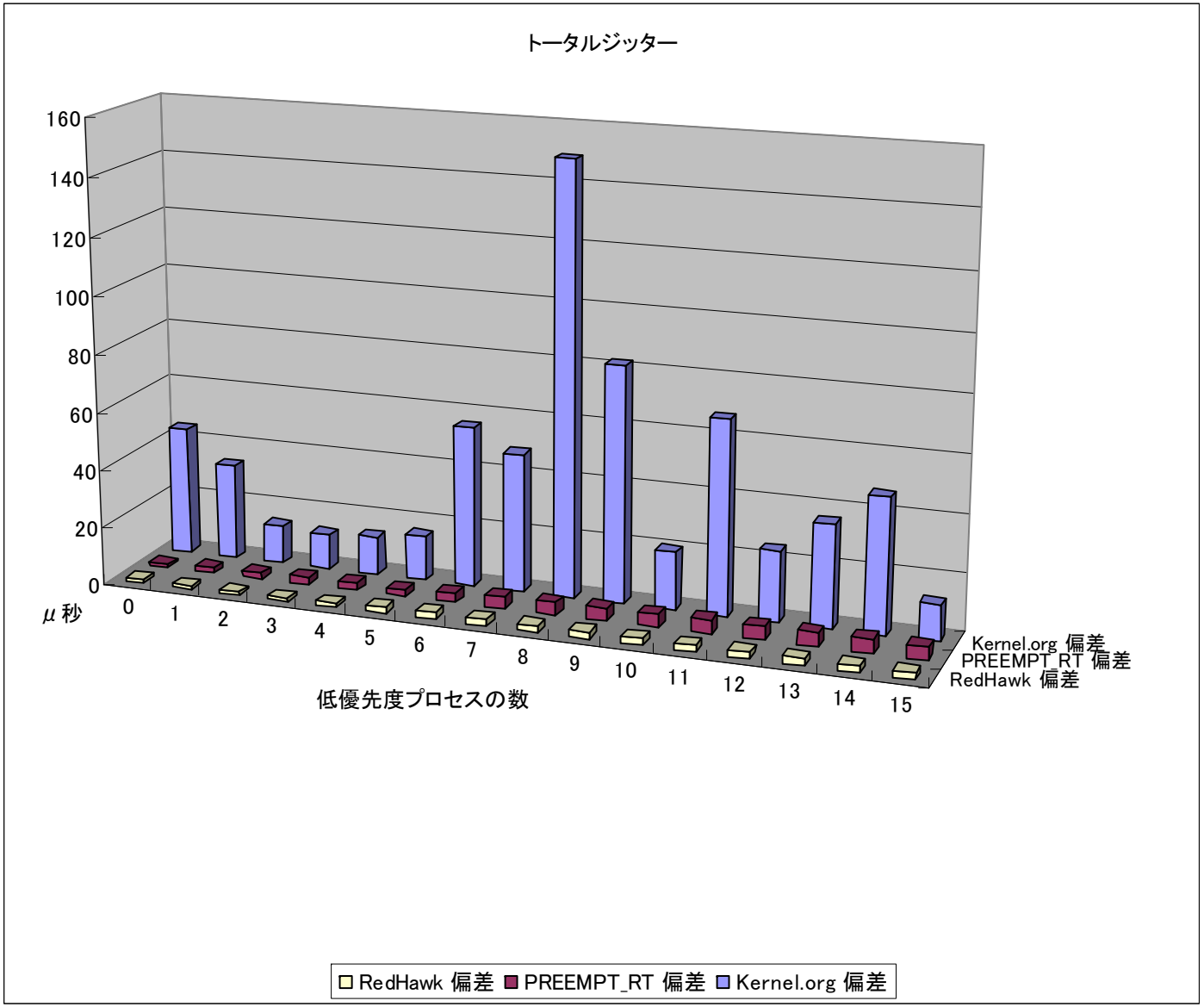
つまり、Kernel.org では、計算可能性（データミニズム）が欠如していることであり、リアルタイム性は無いと結論づけられる。

PREEMPT_RT は、一定の性能を示しているが、低優先度クライアント数を増加していくと PREEMPT_RT は、相対的に性能が落ちて行く傾向に有る。

次ページのグラフは、右の棒グラフを折れ線グラフに変更したものである。先に述べたとおり、低優先度クライアント数を増加していくと PREEMPT_RT は、相対的に性能が落ちて行く傾向にあり、この試験では低優先度プロセスの数が５の点から変化が起こっている。

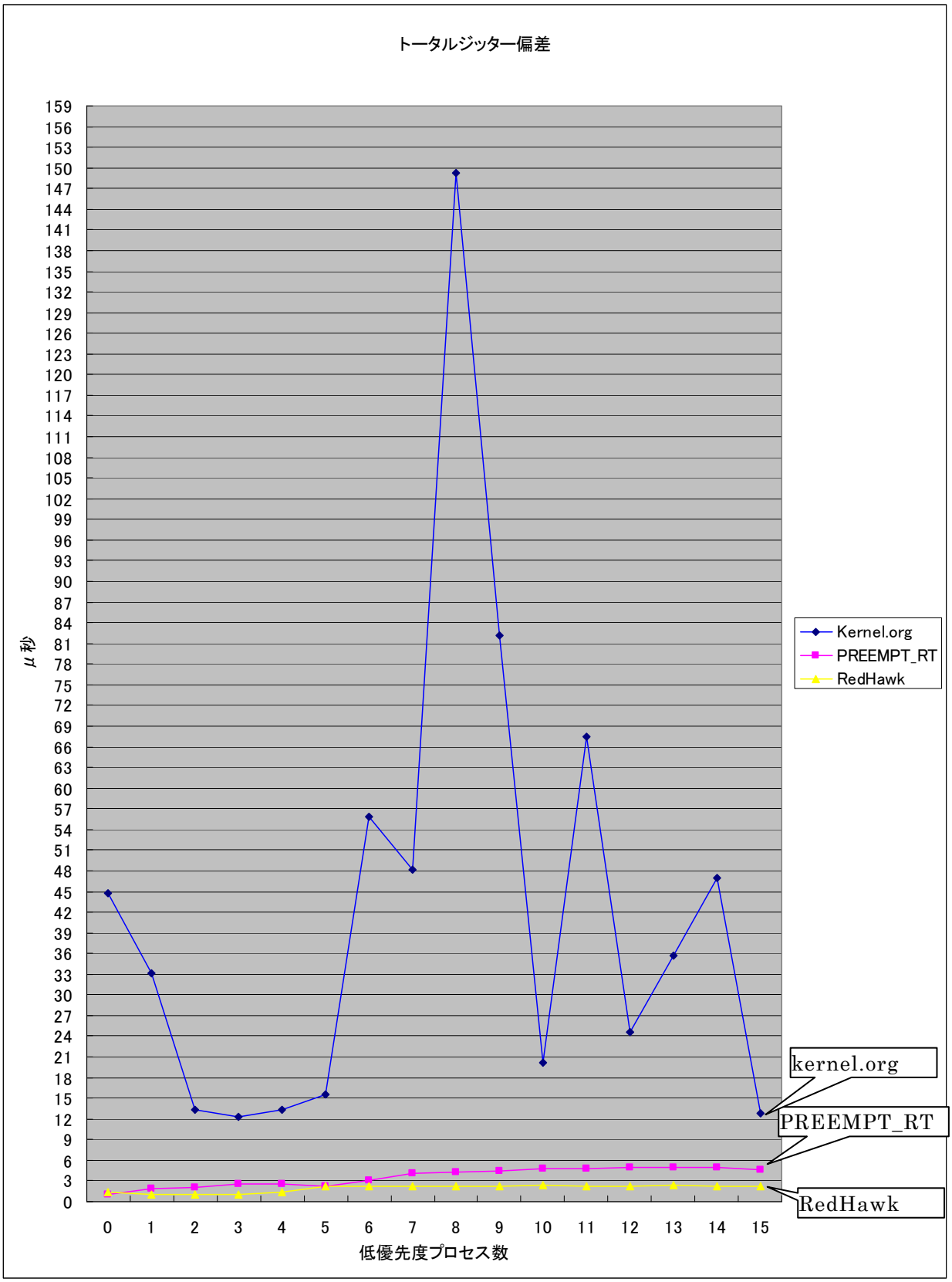
これは、CPU3 から順に負荷プロセスを割り当てると、5 で今回のシステムに搭載された CPU 上限に達することと関係がある。

つまり、PREEMPT_RT においては、CPU コアに割り当てるプロセス数が増加すると、それが低優先度のプロセスであっても、高優先度のプロセスの応答性により影響してゆくと結論づけることが出来る。



RedHawk の場合には、以下のグラフからその影響は小さいことを読み取ることができる。

このように、RedHawk は、リアルタイム応答性の向上に寄与するように設計され応答性保証の無いの PREEMPT_RT パッチを当てただけの Kernel2.6 系 Linux とは、一線を画する性能であることを確認出来たと結論づけられる。



参考文献

(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