Project DOUBT
ご紹介と参加者の募集

Project DOUBT First Speaker
奥山 健一

2004 年 11 月 19 日

Project DOUBT's URL

- 何は無くとも URL を…

http://developer.osdl.jp/projects/doubt/

Overview

- OSDL って何？
- Project DOUBT って何？

OSDLって何？

- 本家の URL:　http://osdl.org/
- 日本の URL:　http://osdl.jp/
- Open Source Development Laboratory　の略
- もっと判りやすくいうと：
  Linus Torvalds
  Andrew Morton
  にお給料を出している団体。
- 本拠地：　Oregon 州　Beaverton（ Portland　の近く）
  ただし、Linus も Andrew もここにはいません。
  彼らはシリコンバレーにいます。ので、OSDL は全然影響力がありません（T-T）。ちょっと無さ過ぎるぐらいに無い。

で、Beaverton ってどこ？

- オレゴン州の北のほうにある田舎町
- San Francisco から北へ550 マイルぐらい。
- 典型的なアメリカ北部。
- Portland という街ならびに空港と、ちんちん電車で繋がってます。
  アメリカらしからぬ(?)自然に優しい村です。

Beaverton のどこ？

- Beaverton Central
  という駅の「目の前」。
  ビルの 4F 。

OSDL JP は？

- 今は横浜ビジネスパークテクニカルセンターの 2F にあります。
- でも 11 月一杯で引っ越すそうです。
- 新しい場所は有楽町だそうな。
- 残念ながら写真がありません…

OSDL 概略

- 2000 年に IBM, HP, Intel, NEC が出資して創設した。
- 今は 61(?) 社が参加している
  日本でも Miracle Linux, 三菱電機 , 富士通 ,NEC, 東京工科大学 , 東芝 ,Turbo Linux, 日立 , 早稲田大学 ,NTT とかが参加している
- 一応、企業間中立な NPO 組織
- IBM, Intel にとって Linux 開発者である Linus 君があまりにも制御しづらいので、どうにかしようとしてできた組織、と言った方が実態にあってるかもしれない (^^;)
- でも、その「本音の所」自身はうまく行ってない (^.^) 。
- Open Source なグループに対し、リモートで実験するためのハードウェア環境を提供したり、Linux に欠けている機能をまとめたり、という事をしてます。

OSDL ミッション

- Linux 業界の中核を成す存在として認識され、以下のことを通して、エンタープライズコンピューティングにおける Linux の採用加速に取り組む。
  - Linux 開発コミュニティにおけるエンタープライズクラスのテストと技術サポート
  - Linux 業界のリソースを最もニーズのある分野への投資にフォーカスし、成長の阻害要因を除去
  - Linux 開発コミュニティとの効果的な協調作業について、ベンダーやエンドユーザーなどメンバーに対する実践的ガイダンス

OSDL の活動

今、主にやっているのは以下の通り

- 3 つの Working Groups
- 4 つの Special Interest Groups(SIG)
- 25 以上の Open Projects

当たり前ですが、全部英語です。

日本語のページなんぞありゃしません(^^;)

3 つの Working Groups

- Carrier Grade Linux
  - 交換機制御用に Linux OS を使う上で必要な機能の定義しよう、というWG
- Data Center Linux
  - Web Service など、バックエンドに Database を持つような世界に Linux を使う上で、必要な機能の定義をしよう、というWG
- Desktop Linux
  - 皆さんが「普段使うオフィスユース」などに Linux を使うために必要な機能の定義をしよう、というWG (Windows Killer)

WG での決定を通じて、Linux Distro の標準化や、
ツールの普及を目指す。Community を刺激する。

OSDL メンバー会社だけが参加できる。

4 つの Special Interest Group(SIG)

- Security SIG
- Storage SIG
- Hotplug SIG
- Binary Testing SIG

- それぞれに興味のある人が、興味のあるトピックについて参加できる。
- 出資会社以外も、個人でも参加 OK 。
- 各 SIG の興味対象は WG のテーマに沿ったもので、調査等が必要になると作られる。
- 以外と名が体を現していないことがある
（Storage なのに NFSv4 が対象に入ってるとか ）

たくさんの Project

- OSDL が OK を出した Open Source プロジェクト (出資会社の提案なら大抵 OK が出ます) 用のリソースを提供してくれます。
- Project DOUBT もこの一つ。
- 他にもこんなにたくさん…

Kernel Compiles
Robust Mutexes
OpenAIS (SA Forum Interfaces)
SA Forum Interfaces - Test Project
Linux Ethernet Bridging
Iproute2
TIPC
Event Service

Membership Service
Fast Reboot (kexec)
Kmsgdump
AIO
Linux Security Module
DAC960
PLM Development
STP Development

Live Patching (Pannus)
Boot Image Fallback
High Availability Linux Clusters (linux-ha)
DigSig - Digital Signatures for ELF binaries
Prioratus - Prioritized protocol processing

で、ようやっと

Project DOUBT

DOUBT の精神

We want to **trust** system.

We do not want to **believe in** system.

我々はシステムを信頼したいのだ

信仰したいのではない

DOUBT の精神

システムを信仰するってどういうこと？

- もし、システムがテストされていないのに、それが動作すると思っているなら、あなたはそのシステムを信仰している
- システムが動作する証拠があるなら、そのシステムは信頼できる
- 我々はシステムを信頼したいよってシステムは
  テストされなくてはいけない。

DOUBT の Misson

- システムテストを収拾し、ないものについては作成する
  - 対象は主に Linux だが、限定する必要は無い
- 収拾したテストを、可能な限り多くのシステムで実験し、結果を公表する
- デバッグではなく、バグの検出に主眼を置く
  (ので、バグがあることが判ればよいのであって、どこにバグがあるのかは全然判らなくても構わない)
- テストの自動化は主眼としない
- テストの **coverage(完全性)** は気にしません。

DOUBT の Target

- 他のプロジェクトがすでにやっている部分についてのテストは、やりません。気にもしません。
  - Linux Kernel Test をはじめとしていくつものテストプロジェクトは世の中に存在する。
    - そのほぼ全てが **Syntax Test** を主眼においている
  - しかし、次のような条件を満たすプロジェクトは無かった :

1) Semantic Test を主眼にする
2) 各テストの目標が明確で、ユーザーが目的ごとにテストを選べる

そこで、DOUBT は :

- Semantic Test を主眼とする /Syntax Test は通ったものとする

Semantic Test ってなに？

- システムテスト全般の内、Syntax Test じゃないもの全部

Syntax Test:

- 文法的なテスト
- 関数などのインターフェースが、ドキュメントどおりに用意されているかどうかテストする。
  - 関数があることを確認する
  - 正常な入力の場合は、正常な結果が得られるか確認する
  - 異常な入力の場合は、エラーが出ることを確認する

Semantic Test:

- 意味論的なテスト
- たとえば、write(2) というシステムコールが本当にデータを HDD に書きに行っているか確認するテスト

Syntax Test

- Syntax が一致していないと、それ以降の何も調べられない
  (プログラムのインターフェースが間違っているのだから、
  その挙動が正しいも、間違っているも、何も無い)
- Linux Kernel の場合、Linux Kernel Test (LKT) というプロジェクトが、最も有名
  (Linux Kernel はリリース前に LKT でテストされています)

- Syntax Test は Countable Test と言って Coverage が計測できる種類のテストです。
  - 完全にテストしきるための組み合わせは数え上げることができる
    (全部実行することができるほどの数に収まる、という意味じゃない)

じゃあ Semantic Test は数え切れないの？

- 数え切れません。なぜなら総数が判らないからです。

- Alan Turing が 1936 年に提唱した **Turing Machine** の **停止決定不能問題** という重要な定理があります。
- この定理は Semantic Test の総数が判らないことを数学的に証明しました。

この定理は重要なので、ちょっと寄り道しましょう。

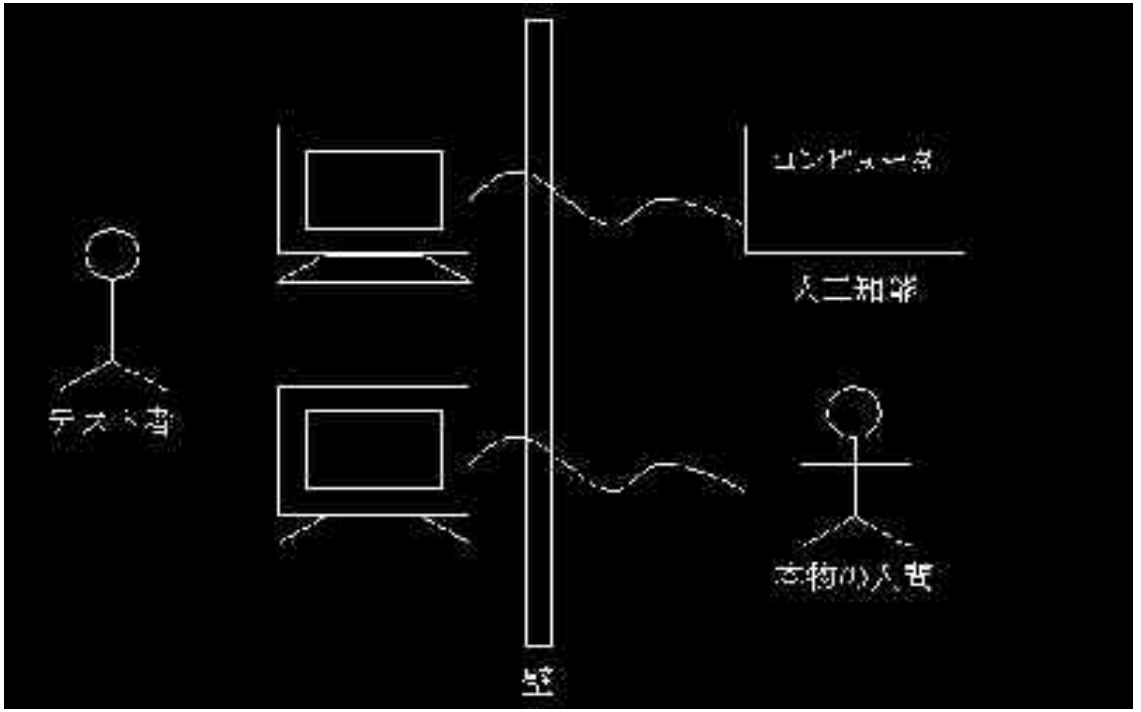
Alan Turing って誰？

Alan Mathison Turing 博士

- 1912 年 6 月 23 日、ロンドン生まれ
- 1954 年 6 月 7 日、マンチェスターで自殺
- 計算機の数学的有用性と、その限界を数学的に解析した、計算機科学の祖
- Turing Machine と Turing Test が最も有名な 2 大成果

Turing Test

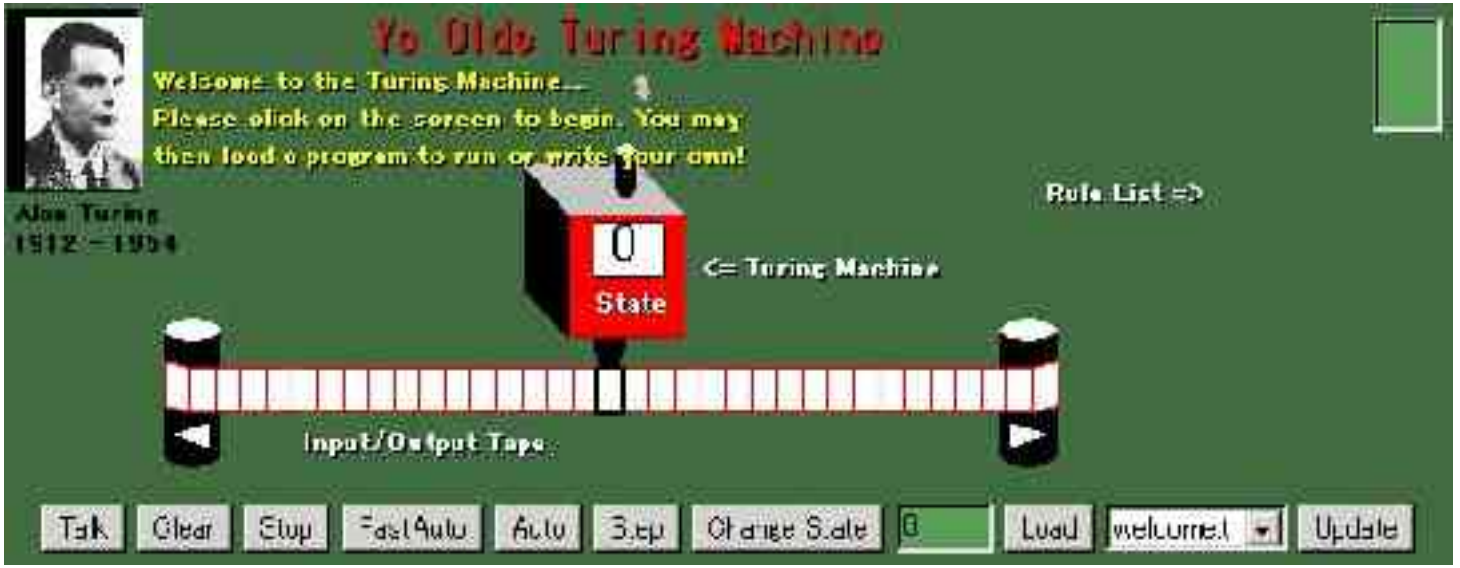
- 人工知能がパスするべきテストとして Alan Turing が提唱した
- それまでは人造物に知能がある / ないの判定基準は無かった
- 人間が、通信相手が機械か、人間か、区別できなければ合格
- 人工無能 という Turing Test にパスするプログラムが流行



Turing Machine

- 1936 年「計算可能数についての決定問題への応用」という論文で出してきた究極の計算機
  1. 無限に長い、升目で区切られた、紙テープ
  2. その中に格納された情報を読み書きするヘッド
  3. 機械の内部状態を記憶するメモリ

  という部品からできている



Turing Machine cont.

- 内部状態と、ヘッドから読み出した情報の組み合わせで
  1. 右か左に1枡動く
  2. ヘッドの位置の情報を読む
  3. ヘッドの位置に情報を書く
  4. 機械の内部状態を変える

  のどれかをやる
- 最初は Start という状態
- End という状態になるまで続ける
- 「内部状態」がどう変化するかは予め与える。
  これがプログラム
- 実行にかかる時間は気にしてはならないという前提

Turing Machine cont.

- 「計算可能数についての決定問題への応用」
  という論文のキーポイントは？
  - 数学において証明可能な事は、全てチューリングマシンで証明可能である(万能マシン)

  つまり
  - 計算機の能力が上がり
  - 情報がデジタル化できたなら
  - 数学的に可能な事は計算機で何でもできる

  という事

  (過去半世紀に渡る、計算機への投資は全部この論文から始まった)

Turing Machine の限界

- Turing Machine には停止決定不能問題 (Halting Problem) という有名な定理が存在する

任意のプログラム P が
正しい (いつか必ず停止する) か、
バグっている (いつまでも停止しない) か
を判定するプログラム D は書けない

という定理　（究極のデバッガ D は作れない、という事)

停止決定不能問題の証明

- 究極のデバッガ D() があるとする。
  D は引数にプログラム p を受け取り、止まる / 止まらない を返す
- 次のようなプログラムを書く。

```
Q( void )
{
    while ( D(Q) ==  止まる  ) ;
}
```

- プログラム Q は D() によって停止すると判定されるか、停止しないと判定されるか？
  - 止まると判定すると while 文が無限に回るので止まらない。
  - 止まらないと判定すると while 文がすぐ終了し、止まる。

停止決定不能問題について判っていること

- ごく、簡単なプログラムは停止決定が判断できます
  - P() { end; }　　きっと止まるだろう
  - Q() { while(1) ; }　　きっと止まらない
- だからどこかに決定不能境界があるに違いありません。

- Syntax Test は、少なくともテスト総数が判る類のテストである、という事は判っています。
- でもそれぐらいしか判っていません
  (正確には、この問題は意外といやらしくて、
  決定不能境界発見自体、かなり困難である事はわかっている)

テストの自動化をしない理由

- 任意のプログラムについてバグの有無は判らない
- でも簡単なプログラムだと判る

  P() { exit; }　　はきっと止まるだろう

  Q() { while (1) ; }　　はきっと止まらない

- テストプログラムが複雑だと、システムにバグがあるのか、テストプログラムにバグがあるのか、判らない
  - プログラムは可能な限り単純でなくてはいけない
- 自動化のような、複雑になる元凶は入れないほうが良い

Coverage を気にしない理由

- 任意のプログラムのバグの有無を判定するプログラムは書けない
- Linux Kernel はあまりにも大きすぎてバグの有無を判定できるほど単純とは思えない
  - という事は、n 回バグが発見されたプログラムに、n+1 個目のバグがあるかどうかは判らない。
  - ということは、バグの総数は判らない
  - バグの総数が判らないのに、 Coverage なんて判るわけない。

というわけで、DOUBT では Coverage を気にしません。

Coverage を気にしない理由

- Syntax テストのような種類のテストは、Coverage を示せます
  - テストするのが、全テスト種類の中の一部でしかない
  - テスト対象を数え上げることができる

- 他のプロジェクトでも Coverage を示せるものがあれば、そういう部署は、他のプロジェクトに任せます。
  我々はそういう所はテストの対象にしません

- それでもカバーできないテスト領域は山のようにあるので、
  気にしません

終わりはあるのか？

- C 言語のようにチューリングマシンを記述できるプログラミング言語のことを Turing 完全 といいます。Turing 完全な言語でかかれた Linux Kernel の正しさを証明するのは…

  天才が現れるのを待つしかありません。

- とりあえず、プロジェクトが終わる事はありえなさそうです。

テストを作成する手順

- 作業を 5 つの段階に分割しました
  1. どのような間違え様があるかを考える
  2. その間違えが、どのような症状となって現れるはずか、考える
  3. 症状を検出する方法を考える
  4. 検出方法を実装するテストツールを作成する
  5. さまざまなシステムでテストツールを実行し、結果を公表

- 各段階に、ばらばらに参加できるようにしました
  - Open Source コードを提供できない人でも 1-3 はできる
  - プログラムを Open にできるけど、1-3 が思いつかない人もいます

こういう人たちが協力できるように、作業を意図的に分解しました

でもなんか難しそう…

- diskio というテストを例にとって、
  難しいかどうか、見てみましょう。

Step1. 間違え様を思いつく

- Linux のファイルシステムは、書けと言ってもちゃんと disk に書いてない気がする。

- sync;sync;sync ってやっても、その直後の umount でものすごい IO が発生することがある。
- ベンチマークテストの結果が FreeBSD に比べて鬼のように速い !! FFS と ext2 なんて基本構造からして同じなのに。
  そんなに大きく速度が変わるなんてありえない

怪しい

Step2. 症状を考える

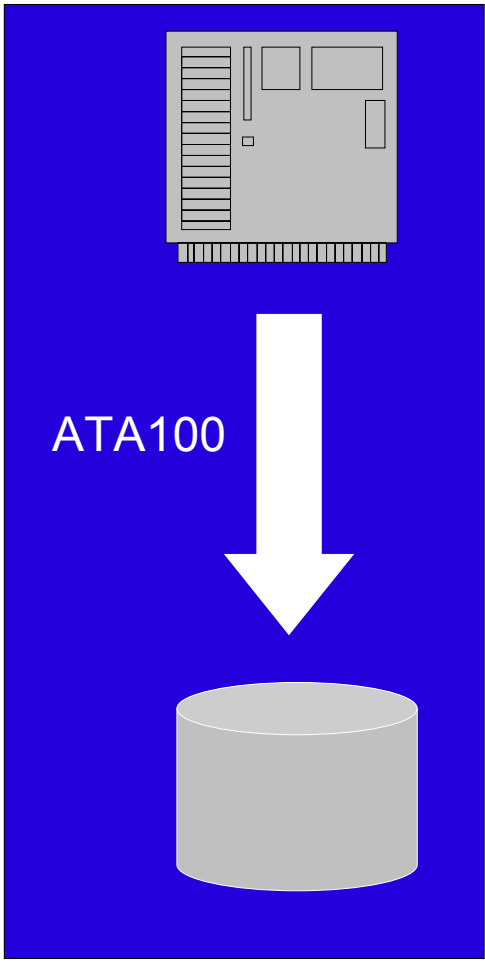
- 書けと言ってもちゃんと書いてないんだから、きっと応答速度がすごく速いに違いない。
  - write(2) システムコールの応答時間を見れば、何か判るかもしれない
- すごく速いの基準を何か見つければよい

Step3. 検出方法を考える

- HDD にまじめに書いているならば、マザーボードから HDD にデータを流すための ATA ケーブル上をデータが流れなくてはいけない
- ATA 100 は 100Mbyte/sec 出る、という事だ。
- 8kbyte のデータを流した場合、

  8k(byte) /100M (byte/sec)
  = 8/100k sec
  = **78.125 usec**

  かかる。書き込み時間の**最短時間**がこれ未満になったら**不合格**としよう。



Step3. 検出方法を考える cont.

- マイクロ秒単位の時間をどうやって計測する？

- Pentium には Time Slice Counter という64bit のカウンターがある。CPU tick 1tick ごとに 1 づつ数を増やしていく(reset 直後が 0)
- いまどきの CPU は 3GHz とかで動くから、0.3nsec 単位で 1 づつ増える。これを使えば マイクロ秒ぐらいなら楽勝に違いない !!

Step4. テストツールを作ってみた

```
int                     fd;
long long     start_t, end_t

fd        = open( “target”, O_RDWR|O_CREAT|O_SYNC,0777 );
start_t = rdtsc();
write( fd, dumpbuf, 8192 );
end_t   = rdtsc();
close( fd );
printf( “%llu\n”, end_t – start_t );
```

- エラー処理などを全部取り外すと、この程度。
- 同期方式として fsync()などを使う方法も書いてみた。

Step5. 実験結果

Kernel	journal mode	command	time(microseconds)		
			average	minimum	maximum
2.6.5	Journal	fdatasync	5489.1	1110.0	17446.9
		fsync	5610.5	1109.5	24296.4
		odsync	4936.1	1108.0	17008.7
		osync	4647.9	1110.4	16825.6
		sync	2282.8	2034.3	13107.4
		write	9.4	5.9	28.0
2.6.5-bk1	Journal	fdatasync	5316.8	7.7	52122.6
		fsync	4229.5	7.6	64134.5
		odsync	4549.0	2064.5	56366.1
		osync	5246.8	2096.9	90962.2
		sync	2335.8	2050.3	115195.9
		write	9.4	5.8	18.8
2.6.8.1	Journal	fdatasync	4903.7	7.9	21773.9
		fsync	4985.0	13.8	18425.8
2.6.9-rc1	Journal	fdatasync	5012.4	8.5	18451.9
		fsync	3331.6	8.7	24501.1

- 明らかに速すぎるケースが発見された!!
- ありゃ、FC3 のベースである 2.6.8.1 にも問題があるよ…。

他にわかりやすい例は？

- Wbtest (EMC の Brett Russ さん提供)
  - HDD に write cache があっても、同期書き込みの場合は write through で書き込まなくてはいけない。が Linux はやってないっぽい。
  - Write cache のついている HDD と Write cache のない HDD を 1 台づつ用意し、交互に書き込みを行う (何を書くのかは予め決めておく)
  - 途中のどこかで電源を抜こう
  - Write cache つきの HDD が Write cache なしの HDD と比べて大幅に書き込みがずれていたら、Write through では書いてないに違いない

実験結果は公開できないらしいが、

テスト方法は公開してもらえました。

結局のところ

- 完全網羅性を保障するアルゴリズムが無い以上、テストと言うのは視点 / 発想の問題です
  - 今まで誰も思いつかなかったことを思いつくのが大事
  - なまじっかな実装の知識など、無い方が良いアイディアを思いつきます
  - 本来どうあるべきか
    本来どうあって欲しいか
    を徹底して考えましょう

ただ、ちょっとまって
それ以前に

簡単に検出できるバグすら取れていない

Linux って実は危ない？

- ただの利用者に過ぎない人へ
  - 危ないです。 *BSD を使ってください (^o^)

- これから Linux Kernel 界で一旗挙げようという人へ
  - 活躍の場がたくさん残っている、
    と考えるべきでしょう
  - Fedora Core は Distro 開発用 Distro ですので、
    そういうのを積極的に使いたがる人にこそ、
    DOUBT で活躍して欲しい！

DOUBT へご招待

- 5 つのステップ全てに参加する必要はありません。
  - 簡単だと思った所から参加してください
  - 思いついたものから参加してください

ただし、これ、英語のプロジェクトです。

- Mailing List も英語です
- WebPage も英語です
- でも、「簡単なことは簡単に表現できる」ので心配しないで。

最後に

- もう一度 URL を:

http://developer.osdl.jp/projects/doubt

ML への参加方法も、このページに書いてあります。
皆様のご参加をお待ちしております。