Vol.2023-SE-215 No.4
2023/11/30

コンパイラー警告を利用した データ型変更時の影響範囲確認方法の提案

大江秀幸 [1,a)]　松下誠 [2,b)]　井上克郎 [3,c)]

概要：UNIX ベースの 32bit システムでは，2038 年に時刻情報のオーバーフローを起こすことが知られている．時刻情報が 64bit に拡張されたシステムでは特に問題とはならないが, 32bit システムで開発されたプログラムで, かつ 2038 年以降も稼働する場合では問題となる．これまで筆者らは，このようなシステムを 32bit システムのままで 2038 年以降も稼働させる，低コストなプログラムの修正方法を提案してきた．本稿では，そうしたシステムで 2038 年問題を完全に解決すべく 64bit 化を検討する際に，アプリケーションの中で修正が必要な箇所を簡単に抽出する方法を提案する．また，この方法で発見した FreeBSD 13.2-RELEASE の amd64 環境に存在する問題点について報告する．

キーワード：2038 年問題，32bit システム，UNIX，ソースコード解析

Checking the Scope of Influence When Changing Data Types Using Compiler Warnings

HIDEYUKI OE[1,a)]　MAKOTO MATSUSHITA[2,b)]
KATSURO INOUE[3,c)]

Abstract: On UNIX-based 32-bit systems, it is known that a time information overflow will occur in 2038. This is not a problem for systems with time information extended to 64-bit, but it is a problem for programs developed on 32-bit systems that will continue to be used after 2038. The authors have proposed a low-cost method of modifying such programs so that they can be used after 2038 on 32-bit systems. In this paper, we propose a method for easily identifying the parts of an application that need to be modified when considering the conversion to 64-bit systems to completely solve the 2038 problem. We also report on problems found in the amd64 environment of FreeBSD 13.2-RELEASE using this method.

Keywords: The year 2038 problems, 32bit system, Unix , Source code analysis

1. はじめに

2038 年問題は, 1970 年 1 月 1 日 0 時 0 分を起点として毎秒カウントアップする Unix 時間が，32bit 符号付き整数の最大値を超える際（2038 年 1 月 19 日）に起きる問題である．この問題の原因は，Unix 時間を 32bit 符号付き整数で扱うため，上記の日付でオーバーフローが発生することである．最近では，Unix をはじめ各種の OS で 64bit への拡張が進んでいる[1][2]ため，64bit 環境で新たに開発するアプリケーションソフトでは 2038 年問題は起こらない. しかし，Unix 時間を 32bit で扱う環境で開発され，既に稼働しているシステムについては，2038 年問題が内在している．特に 2038 年以降も稼働する長寿命なシステムでは対策が必要となる．

一般に，長期間にわたって稼働している 32bit のシステムを 64bit 化する場合, ハードウェア等のシステム要件や，OS，ソフトウェアライブラリ，ドライバ等の対応状況なども含め，影響範囲の見極めが困難なため，対策にはコストがかかる．

筆者らは，この問題をより低コストで解決するため，32bit のまま起点時間を 1998 年に 28 年間ずらす方法を示した[3]．この方法は，システム全体を大きく改変する可能性のある 64bit 化よりは低コストで実現できるが，2038 年から 28 年後の 2066 年には再度対策が必要となる．2066 年以降も稼働するシステムを対象とする場合には，システムを 64bit に拡張することが望ましい．その場合，time_t 型変数を 64bit 化するだけでなく，アプリケーションプログラム（以下，アプリケーション）の中で時刻の計算や，時刻情報を一時保存するために利用している変数も含めて 64bit 化が必要となる．それらの変数の利用箇所を漏れなく特定するのは，プログラムの規模によっては困難となる．

本稿では，2038 年問題の対策として 64bit 化を行うアプリケーションのこうした修正箇所を，一般にアプリケーション開発に用いる環境だけで簡単に抽出する方法を提案する．また，この方法で発見した FreeBSD 13.2-RELEASE の amd64 環境に存在する問題点について報告する．

1 大阪工業大学 非常勤講師／(株) Ｐｌｕｓ　Ｐｒｉｓｍ
Plus Prism Co.,Ltd Suita-shi, Osaka 565-0824, Japan
2 大阪大学
Osaka University, Suita-shi, Osaka 565-0871, Japan
3 南山大学
Nanzan University, Nagoya-shi, Aichi 466-8673, Japan

a) hideyuki.oe@plusprism.co.jp
b) matusita@ist.osaka-u.ac.jp
c) inoue@ist.osaka-u.ac.jp

2. 提案手法

2.1 問題点と課題

ハードウェアスペック等のシステム要件が許す前提で，アプリケーションの 2038 年問題に time_t 型の 64bit 化で対応する場合，まず OS を含むシステム全体を 64bit（LP64 データモデル[4]）に置き換える．この置き換えの際に，2038 年問題の原因となる time_t 型の定義は 64bit に拡張される．

拡張後は，該当システムで動作するアプリケーションのコンパイル時にも，time_t 型は 64bit として扱われる．このとき，アプリケーション内部で変数を別の型（例えば 32bit である int 型）の変数に代入する場合等に問題が起きる可能性がある．この場合，データ境界を越えて代入が行われる恐れがある．

アプリケーション内部で時刻情報をどのように扱うかは，設計に依存する．そのためソースコード上のどこで時刻情報を扱っているかを事前に知ることは，一般には困難である．

2.2 対象とする環境

FreeBSD 13.2-RELEASE を対象とし，/usr/bin 以下の各種コマンド（cat から uuidgen コマンドまで，全部で 38 個）をアプリケーションと見立ててこの問題を検討する．ソースコードのコンパイル環境は，/usr/src 以下に配置する．FreeBSD ではアーキテクチャーごとにインストールイメージが用意されている．本稿で対象とするアーキテクチャーは，i386（32bit），および amd64（64bit）とする．これらの環境の/usr/bin 以下のコマンドのソースコードは全く同一であり，データ型はアーキテクチャーに合わせて表 1 のように定義される．

表 1　アーキテクチャーによるデータ型と bit 幅
Table 1　Data type and bit width in each architecture.

型名	i386（ILP32）	amd64（LP64）
int	32bit	32bit
long	32bit	64bit
long long	64bit	64bit
time_t	32bit	64bit

2.3 手法の概要

システムの 64bit 化に伴い，time_t 型は 32bit から 64bit に拡張される．このとき，time_t 型を 32bit として扱う当該変数への代入，参照処理時に桁溢れやデータ型の不整合に起因する問題が発生する恐れがある．問題解決のためには，ソースコード上でこの処理を行う部分を見つけて修正する必要がある．

修正箇所特定のため，i386 アーキテクチャーで 2038 年まで問題なく動作するアプリケーションで，time_t 型変数のみを 64bit に拡張する．i386 アーキテクチャー環境では，time_t 型変数値を一時保存するのに，32bit の変数を利用するのが妥当である．time_t 型を 64bit 化すると，例えば当該変数の値を一時変数に代入する箇所でオーバーフローが起きる可能性が高い．このような箇所では，システムを 64bit 化したとしても，バグを含むことになる．

本研究では、このような箇所を簡便に発見するために，64bit 化する前の i386 環境のコンパイル結果に対して、time_t 型のみを 64bit 化した環境でのコンパイル結果を比較することを提案する．64bit 化により増加した警告による指摘された箇所付近のコードを詳細に検討することにより，必要な修正箇所が容易に発見できることが期待される．

具体的には，代入サイズのミスマッチを警告するコンパイルスイッチを Makefile に追記し，以下の２つの環境でコンパイルする．

① time_t 型のみを 64bit 化した i386 環境
② 純粋な i386 環境

コンパイル結果には，それぞれサイズミスマッチ警告を含むが，このときにどちらか片側にだけ現れるサイズミスマッチ警告が，time_t 型を変更したことに起因する警告である．これは①と②の警告メッセージを比較した際に差分として現れ，修正候補の箇所となる．32bit と 64bit でソースコードは共通のため，得られた修正候補箇所から 64bit ソースコードを用い，オーバーフローの可能性を考慮して修正の要否を検討する．図 1 に修正箇所特定方法の概要図を示す．

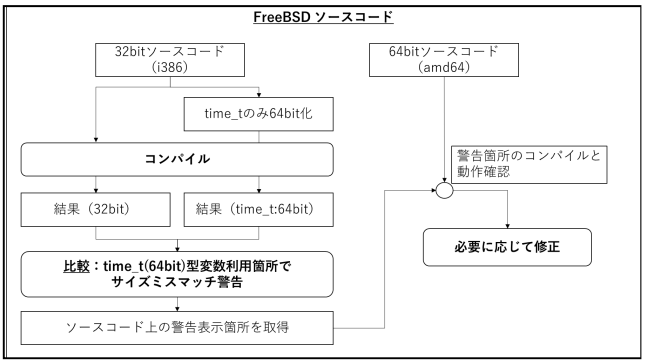

図 1　修正箇所の特定方法
Figure 1　Method of identifying modification parts

このようにして，一般にアプリケーション開発時に利用する環境だけを用いて修正箇所を抽出できる．

2.4 比較環境の構築

32bit の FreeBSD が動作する環境を構築し、ソースコードを入手する．FreeBSD では，Makefile を/usr/src/share/mk ディレクトリに格納している．このディレクトリ中の src.opts.mk ファイルで，/usr/src/bin 以下のソースコードを

コンパイルする際のオプションを設定できる．また FreeBSD では，コンパイラー警告が発生すると，その時点でコンパイル動作を止めている．警告を元に修正箇所の分析を行うには，警告が発生してもコンパイルを続行させる必要がある．このため，”src.opts.mk”ファイルに記述を追加してコンパイルが止まらないようにする．設定例を図 2 に示す．

```
CFLAGS += -Wshorten-64-to-32
MK_WERROR= no
```

図 2　“src.opts.mk”へのコンパイルオプションの追記例
Figure 2　Example of adding compile switch to “src.opts.mk”

2.5 i386 環境での time_t 型の 64bit 化

time_t 型は/usr/include の各種ヘッダファイルの中で定義されており，データ幅の設定はハードウェアスペックに応じたコンパイルオプションで行っている．図 3 にヘッダファイルでの time_t の定義の概略図を示す．

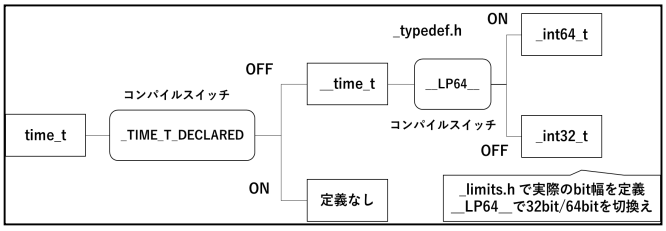

図 3　time_t 型の定義
Figure 3　Definition of “time_t”

time_t 型を定義する各ヘッダファイルでは，time_t 型の二重定義を防ぐため，_TIME_T_DECLARED の定義を用いている．time_t 型の定義前に#ifndef _TIME_T_DECLARED として別のヘッダファイルで time_t 型が定義済みでないかを確認し，未定義の場合には time_t 型の定義と同時に _TIME_T_DECLARED を定義する．つまりコンパイル時に最初に_TIME_T_DECLARED を定義した箇所で，time_t 型が定義されていることになる．このことを利用して，time_t 型の定義を 64bit に変更することができる．

コンパイルオプションに「-include ファイル名」を指定すると，指定したファイルを最初にコンパイラーが処理する．このことを利用して，time_t 型の bit 幅を所望のものに変更する．図 4 に，time_t 型を 64bit にする指定ファイルの内容を示す．

```
1 #ifndef _TIME_T_DECLARED
2 typedef long long      time_t;
3 #define _TIME_T_DECLARED
4 #endif
```

図 4　64bit time_t 型の宣言
Figure 4　Declaration of 64bit “time_t”

2.6 コンパイルの実施

FreeBSD 13.2-RELEASE のソースコード（usr/src/bin）を対象としてコンパイルを実行する．前節の time_t 型定義のための図 4 の内容のヘッダファイルを，/home/user1/my_time_t.h として配置し，/usr/src/share/mk/src.opts.mk に以下の表記を追加する．

CFLAGS += -include “/home/user1/my_time_t.h”

このようにして/usr/src/bin/で make を実行すると，time_t 型だけを 64bit に変更したオブジェクトファイルが生成される．make 実行の際に，以下の記述にてコンパイル時の標準出力をファイル化する．

make >&err_64

次に，/usr/src/share/mk/src.opts.mk の記述から-include /home/user1/my_time.h を削除する．

これにより，time_t 型が 32bit のオブジェクトファイルが生成される．64bit でのコンパイル時同様，コンパイル時の標準出力を以下のようにしてファイル化する．

make >&err_32

3. 手法適用結果

3.1 コンパイル結果の比較

2.6 節で作成した 2 つのファイル，err_64 と err_32 を diff ツール等で比較することで，サイズ修正の影響箇所を簡単に抽出できる．WinMerge[5]を用いた比較結果を図 5，図 6，および図 7 に示す．

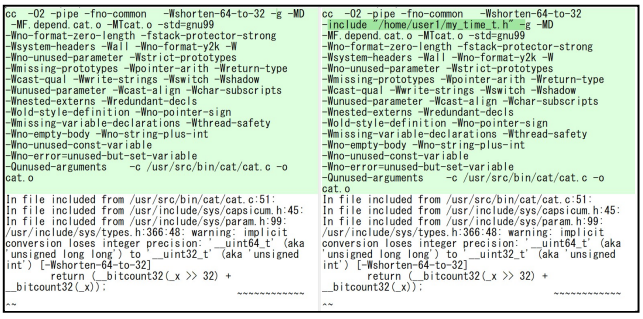

図 5　cat.c コンパイル時の比較
Figure 5　Comparison of the compilation results of the cat.c

図 5 では，cat.c のコンパイル時の差分を示している．図の左側が 32bit の time_t でコンパイルした際の表示，右側が 64bit の time_t でコンパイルした際の表示である．cat.c では，コンパイルオプションに-include を追加している部

分以外で差分はなく，time_t 型の変更によるプログラム上の影響はないことがわかる．

一方で図 6 に示した date.c については，左の 32bit 側で警告が表示されるのに対して，右の 64bit 側では警告がなくなっている．変数 tval が time_t 型で定義されているのに対して，strtoq 関数の戻り値が 64bit のため，32bit 側でのみ警告が表示されている．

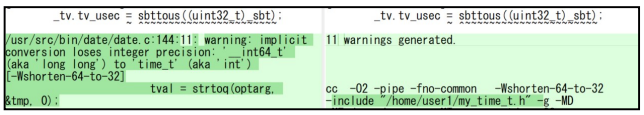

図 6　date.c コンパイル時の比較

Figure 6　Comparison of the compilation results of the date.c

また図 7 の vary.c では，32bit 側には無かった警告が右の 64bit 側で表示さる．ここでは，domktime 関数の戻り値が int 型（32bit）で定義されているのに対して， time_t 型の値（ret）を戻り値とするため，警告表示されている．

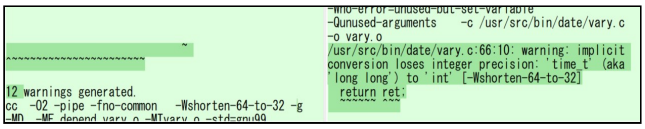

図 7　vary.c コンパイル時の比較

Figure 7　Comparison of the compilation results of the vary.c

このように，time_t 型のサイズ変更による影響箇所が，警告表示を比較することにより明らかになる．2038 年問題に対する修正を検討する場合，time_t 型を 64bit 化した際に表示される警告が問題となる．このため，64bit 側でのみ警告表示される箇所の修正要否を検討すれば，64bit 環境での time_t 型の 64bit 化による影響箇所，および要修正箇所が明らかになる．

3.2　time_t を 64bit 化した場合の修正候補箇所

FreeBSD 13.2-RELEASE のソースコードのうち，/usr/bin 以下のコマンド全てにつき，time_t 型の定義を 64bit に変更してコンパイルを実行し，64bit 環境でのみ警告表示される箇所を確認した．結果，以下のファイルで該当する警告が見られた．

date/vary.c
ps/print.c
sh/eval.c

これらのファイルでは，システム全体を 64bit 化したとしても，他の変数に時刻値を代入する際にオーバーフローが発生する恐れがあり，64bit 化を検討する場合には，修正候補として確認が必要な箇所となる．

3.3　修正候補箇所の確認

以下に警告が表示された箇所を示す．

(1)　date/vary.c

```
/usr/src/bin/date/vary.c:66:10: warning: implicit conversion loses
integer precision: 'time_t' (aka 'long long') to 'int' [-Wshorten-64-to-32]
  return ret;
  ~~~~~~ ^~~
1 warning generated.
```

時刻調整の機能の実現に必要な domktime 関数の戻り値が int 型（32bit）であり，当該関数の戻り値である ret は time_t 型（64bit）であることから，型の不一致が起きている．これは amd64 環境の FreeBSD 13.2-RELEASE でも同様であり，64bit の時刻情報を 32bit に切り詰めて処理している．ただし，domktime 関数の戻り値（int 型）は時刻情報としては扱っておらず，異常値（-1）かそれ以外かの判断に用いている．このため，符号なし 32bit の最大値である 0xffffffff が時刻情報として返った場合には，異常値の-1 として誤判断する恐れがある．修正の要否，修正する場合の方法については 3.4 節で述べる．

(2)　ps/print.c

```
/usr/src/bin/ps/print.c:571:37: warning: implicit conversion loses integer
precision: 'time_t' (aka 'long long') to 'long' [-Wshorten-64-to-32]
        secs = k->ki_p->ki_rusage.ru_stime.tv_sec;
               ~ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^~~~~~
/usr/src/bin/ps/print.c:585:37: warning: implicit conversion loses integer
precision: 'time_t' (aka 'long long') to 'long' [-Wshorten-64-to-32]
        secs = k->ki_p->ki_rusage.ru_utime.tv_sec;
               ~ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^~~~~~
/usr/src/bin/ps/print.c:604:13: warning: implicit conversion loses integer
precision: 'long long' to 'int' [-Wshorten-64-to-32]
        days = val / (24 * 60 * 60);
               ~ ~~~~^~~~~~~~~~~~~~~~
/usr/src/bin/ps/print.c:606:14: warning: implicit conversion loses integer
precision: 'long long' to 'int' [-Wshorten-64-to-32]
        hours = val / (60 * 60);
                ~ ~~~~^~~~~~~~~~~
/usr/src/bin/ps/print.c:608:13: warning: implicit conversion loses integer
precision: 'long long' to 'int' [-Wshorten-64-to-32]
        mins = val / 60;
               ~ ~~~~^~~~
```

これらの警告は，systime，および usertime 関数に含まれる，以下の文が原因で表示されている．

```
secs = k->ki_p->ki_rusage.ru_stime.tv_sec;
secs = k->ki_p->ki_rusage.ru_utime.tv_sec;
```

この文の，ローカル変数 secs は long 型（i386 環境で 32bit）で宣言されており，tv_sec 値は time_t 型（64bit）であることから型の不一致が起きている．また，elapsed 関数に含まれる，以下の文でも警告が表示される．

```
days = val / (24 * 60 * 60);
hours = val / (60 * 60);
mins = val / 60;
```

ここで，days，hours，および mins は int 型のローカル変数（32bit）である．time_t 型の変数である val（64bit）を用いた計算結果を代入するために型の不一致が起きている．修正の要否，修正する場合の方法については 3.4 節で述べる．

(3) sh/eval.c

```
/usr/src/bin/sh/eval.c:1362:31: warning: implicit conversion loses
integer precision: 'long long' to 'long' [-Wshorten-64-to-32]
        shumins = ru.ru_utime.tv_sec / 60;
                  ~ ~~~~~~~~~~~~~~~~~~~^~~~
/usr/src/bin/sh/eval.c:1364:31: warning: implicit conversion loses
integer precision: 'long long' to 'long' [-Wshorten-64-to-32]
        shsmins = ru.ru_stime.tv_sec / 60;
                  ~ ~~~~~~~~~~~~~~~~~~~^~~~
/usr/src/bin/sh/eval.c:1368:31: warning: implicit conversion loses
integer precision: 'long long' to 'long' [-Wshorten-64-to-32]
        chumins = ru.ru_utime.tv_sec / 60;
                  ~ ~~~~~~~~~~~~~~~~~~~^~~~
/usr/src/bin/sh/eval.c:1370:31: warning: implicit conversion loses
integer precision: 'long long' to 'long' [-Wshorten-64-to-32]
        chsmins = ru.ru_stime.tv_sec / 60;
```

これらの警告は，timescm 関数に含まれる以下の処理で表示される．

```
shumins = ru.ru_utime.tv_sec / 60;
shsmins = ru.ru_stime.tv_sec / 60;
chumins = ru.ru_utime.tv_sec / 60;
chsmins = ru.ru_stime.tv_sec / 60;
```

ここで，shumins，shsmins，chumins および chsmins は long 型（i386 環境で 32bit）のローカル変数であり，time_t 型（64bit）の変数である tv_sec を用いた計算結果を代入するために型の不一致が起きている．修正の要否，修正する場合の方法については 3.4 節で述べる．

3.4 修正検討

前節で導かれた修正候補箇所に対し，実際に 64bit 環境で修正が必要かを検討した．

(1) date/vary.c

date コマンドで時刻調整オプション（-v）を指定すると domktime 関数が呼ばれる．これは現在時刻を基準として，調整時刻を±で指定し，時刻の調整を行うオプションである．調整後の時刻情報が 1970 年 1 月 1 日 0 時 0 分 0 秒（UTC）からちょうど 0xffffffff 秒後の場合，つまり，2106 年 2 月 7 日 6 時 28 分 15 秒（UTC）の場合に，domktime 関数が 0xffffffff を返すため，呼出し元である年月日等の値を調整する関数でこの戻り値を int 型の「-1」と誤認識する．このため図 8 に示す通り，date コマンドは，”Cannot apply date adjustment” というメッセージを表示して調整が失敗する．この現象の確認には，現在時刻から 2106 年 2 月 7 日 6 時 28 分 15 秒（UTC）の差分を指定する必要があり，通常動作では条件設定が困難なため，テスト環境で変数値を設定して現象を確認した．なお，このときに与えたパラメータは，テスト環境で書き換えているため，指定した+4000w（4000 週間後）はコマンドの実行には利用していない．

```
user1@Prism02:/usr/obj/usr/src/amd64.amd64/bin/date $ ./date -v+4000w
+4000w: Cannot apply date adjustment
usage: date [-jnRu] [-I[date|hours|minutes|seconds]] [-f input_fmt]
            [-r filename|seconds] [-v[+|-]val[y|m|w|d|H|M|S]]
            [[[[[[cc]yy]mm]dd]HH]MM[.SS] | new_date] [+output_fmt]
```

図 8 date コマンドでのエラー時の表示
Figure 8 Display in case of error with date command

この現象が起こる 1 秒後の 2106 年 2 月 7 日 6 時 28 分 16 秒（UTC）に変数値を書き換えた場合，domktime 関数は-1 を返さないためエラーにならない．表示時刻は別で取得した 64bit の値を用いるため，正常に処理を実行し，図 9 に示すように，2106 年 2 月 7 日 6 時 28 分 16 秒（UTC）を表示する．

```
user1@Prism02:/usr/obj/usr/src/amd64.amd64/bin/date $ ./date -v+4000w
Sun Feb  7 06:28:16 UTC 2106
```

図 9 date コマンドでの正常表示
Figure 9 Normal display with date command

ここで、domktime 関数の戻り値を int 型から time_t 型に変更すると，2106 年 2 月 7 日 6 時 28 分 15 秒（UTC）においてもエラーとならないように修正される．図 10 に修正したソースコードを示す．左側が元のソースコード，右側が修正後のソースコードである．

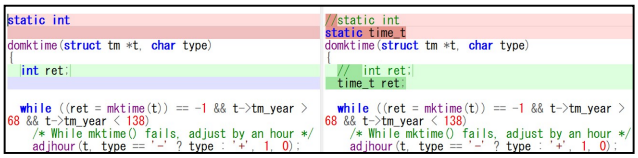

図 10　vary.c の修正
Figure 10 Modification of vary.c

図 11 に修正後の date コマンドの表示結果を示す．

```
user1@Prism02:/usr/obj/usr/src/amd64.amd64/bin/date $ ./date -v+4000w
Sun Feb  7 06:28:15 UTC 2106
```

図 11　date コマンド修正後の正常表示
Figure 11 Normal display after modification of date command

(2)　ps/print.c

FreeBSD 13.2-RELEASE の 64bit 環境では，long 型は 64bit で定義されるため，64bit の time_t 型と矛盾しない．このため systime，usertime 関数では型不一致によるオーバーフローは起こらず，修正は不要である．

一方で，elapsed 関数では 32bit の int 型を用いているため，変数 val（プロセスの経過時間）の値によってオーバーフローを起こす恐れがある．変数 val は elapsed 関数内で 1 日の秒数（86,400 秒）を基準として割り算をした余りから時間，分，秒を計算している．そのため，時間，分，秒については各単位の最大値（24 時間，60 分，および 60 秒）を超えることはない．しかし，日の計算においては，val の値が符号付整数型 32bit の最大値である 0x7fffffff の 86,400 倍を超えた場合に変数 days がオーバーフローを起こす．val の値が 0x7fffffff*24*60*60（5,881,580 年 7 月 11 日 0 時 0 分 0 秒 UTC）の場合と，0x80000000*24*60*60（5,881,580 年 7 月 11 日 0 時 0 分 1 秒 UTC）の場合の ps コマンドの出力を図 12 と図 13 に示す．

```
user1@Prism02:/usr/obj/usr/src/amd64.amd64/bin/ps $ ./ps -U user1 -o usertime,command,etime
USERTIME COMMAND                    ELAPSED
 0:00.00 su (sh)          2147483647-00:00:00
 0:00.00 ./ps -U user1 -o 2147483647-00:00:00
```

図 12　経過時間（最大値）の正常表示
Figure 12 Normal display of elapsed time (maximum value)

```
user1@Prism02:/usr/obj/usr/src/amd64.amd64/bin/ps $ ./ps -U user1 -o usertime,command,etime
USERTIME COMMAND                    ELAPSED
 0:00.00 su (sh)         -2147483648-00:00:00
 0:00.00 ./ps -U user1 -o -2147483648-00:00:00
```

図 13　経過時間（オーバーフロー時）の異常表示
Figure 13 Abnormal display of elapsed time (at overflow)

変数 val の値が 0x80000000*24*60*60（5,881,580 年 7 月 11 日 0 時 0 分 1 秒 UTC）の場合，オーバーフローが起こり，経過時間の情報がマイナスで表示される．500 万年以上先のことであり，現実的な問題としては考え難いが，処理上は問題を含んでいる．

そのため，変数 days を long 型に修正し，その表示のため asprintf 関数に与えている書式指定子を”%3d”から”%3ld”に修正した．修正内容を図 14 に示す．左側が元のソースコード，右側が修正後のソースコードである．

図 14　print.c の修正
Figure 14 Modification of print.c

結果，図 15 に示す通り現象の改善が確認できた．

```
user1@Prism02:/usr/obj/usr/src/amd64.amd64/bin/ps $ ./ps -U user1 -o usertime,command,etime
USERTIME COMMAND                    ELAPSED
 0:00.00 su (sh)          2147483648-00:00:00
 0:00.00 ./ps -U user1 -o 2147483648-00:00:00
```

図 15　修正後の正常表示
Figure 15 Normal display after correction

(3)　sh/eval.c

/ps/print.c の usertime，systime 関数と同様に，64bit 環境での time_t 型と long 型は矛盾しない．そのため，修正の必要はない．

4.　議論

本稿の手法を用いなくても，システムを 64bit 化する際のコンパイル時の警告表示で，対策が必要な箇所を知ることはできる．しかし，一般にシステムの 64bit 化では時刻情報以外にも各種のソフトウェアライブラリやドライバ，他システムとのインターフェースなど，同様の検討や修正が必要なことが想定される．このため，警告は time_t 型の 64bit 化に起因するもの以外にも多数表示され，修正箇所を絞り込むのは困難である．例えば i386 のビルド環境で -Wshorten-64-to-32 コンパイルオプションを有効にした場合の/usr/src/bin 以下の警告の数は，2.6 節で採取した err_32 の警告表示行数を数えることで求められる．数えた結果，図 16 に示す通り，警告表示箇所は 718 か所にのぼった．

```
user1@Prism03:/usr/src/bin $ grep "implicit conversion loses" err_32 | wc -l
     718
```

図 16　/usr/src/bin での警告表示行数
Figure 16 Number of lines of warning displayed in /usr/src/bin

本稿の手法では，32bit システムのまま time_t 型変数だけを 64bit に変更して比較確認するため，time_t 型変更時の影響箇所だけをシステム全体の 64bit 化から切り離して簡単に確認することができる．また，外部システムやモジュールとのインターフェースで 32bit の time_t 型を用いない場合では，32bit システムのまま，アプリケーションで 64bit の time_t 型が利用できるため，システムの 64bit 化に先行して 2038 年問題に対する修正作業を実施することができる．

以下に，本稿の手法だけでは，2038 年問題の修正箇所の抽出が困難と考えられる問題を挙げる．

4.1 値で制限するプログラム

アプリケーション内部で，time_t 型変数以外の値を用いて 2038 年を超えないように制御するプログラムがある場合は，この手法単独での解決は難しい．time_t 型変数を用いている場合には，本稿の手法で影響箇所を特定し，年数制限処理を修正することができるが，例えば struct tm 型等の別の型の変数値で制限する場合には，時刻情報のデータフロー解析と併用する必要がある．

4.2 time_t 値を間接参照するプログラム

間接参照の問題については，time_t 型変数の利用箇所からプログラムスライシング等により解析して，必要な修正を施す必要がある．

5. 関連研究

コンパイラーの警告出力を，プログラム改変時の影響箇所の特定に用いている例は見当たらない．プログラム中に故意にエラーを埋め込み，プログラムが期待した異常動作を行うかを確認するエラーシーディング[6]の考え方と似ているが，本稿の手法は，テスト品質の確認や異常動作を期待しておらず，手法として異なっている．

組込み機器開発における 2038 年問題への対応事例等[3] [7]では，2038 年問題に対し，起点時刻（epoch）である 1970 年 1 月 1 日 0 時 0 分 0 秒（UTC）を 28 年遅らせて，1998 年 1 月 1 日 0 時 0 分 0 秒（UTC）とすることで 32bit システムのまま問題発生を 2066 年まで遅らせる手法を提案した．また，32bit UNIX システムの 2038 年問題に対するプログラム修正法の提案[8]では，その手法を実現する一般的な手順を紹介した．UNIX の 2038 年問題に対する問題箇所特定ツール[9]では,その手順を自動化するツールについて紹介されている．本稿では，問題の根本解決のため，システムの 64bit 化を検討する際に，time_t 型のサイズ変更で直接影響を受ける箇所を簡単に見つける方法を提案した．

Avoiding Year 2038 Problem on 32-bit Linux by Rewinding Time on Clock Synchronization[10]では，32bit システムで基準時間を巻き戻すことにより，低コストで 2038 年問題に対策し，ntpd と linuxptp を修正した事例を紹介している．本稿では 64bit 化を前提とした修正箇所の特定作業を省力化する方法を提案している．

ユーザレベルアプリケーションにおける 2038 年問題の検知と解析[11]では，ユーザレベルアプリケーションを対象に，LLVM-IR を用いた time_t 変数のデータフロー解析を行うツールを作成し，C ベースの GitHub プロジェクトを対象にその効果について報告している．コンパイラー基盤を用いた解析という意味では本稿と類似しているが，本稿では time_t 型を実際に 64bit 化したものと 32bit のものでコンパイラー出力を比較しており，その点で手法としては異なっている．

32bit を超える time_t 型をもつ環境における 2038 年問題の検出手法の提案[12]では，time_t 型変数のオーバーフロー問題を 64bit 化等により解決する際に，32bit の time_t 型と混在する場合の問題に着目し，time_t 型の値が一貫して 32bit を超えることを検査するツールを紹介している．本稿では time_t 型の変数以外の，一時的に利用する変数を含めて，アプリケーションへの影響箇所を検査する方法を提案している．

6. おわりに

2038 年問題の直接の原因は，time_t 型変数がオーバーフローするためであるが，time_t 型変数値をアプリケーションがどのように扱うかは設計に依存するため，修正箇所の特定は困難な場合が多い．修正箇所を特定するには，静的解析ツールやプログラムスライシング等の手法を利用して，time_t 型変数の bit 幅の変更がアプリケーションのソースコードやデータのどこに影響するかを調べる必要がある．影響箇所の特定後は，修正が必要かどうかを判断し，修正が必要な場合には修正方法の検討を行うことになる．

本稿では，この影響箇所の特定作業を，一般的な開発環境であるコンパイラーを利用して簡単に行う方法を提案した．対象としたのは，FreeBSD 13.2-RELEASE のソースコードの一部であるが，その他の箇所や Linux 等でも同様な問題が含まれる恐れがある．提案した手法は，これらの問題にも有効である．また，本手法は time_t 型の変更に限らず，リファクタリング等で変数の型を変更する場合や，データ変更時の影響範囲の解析にも応用できると考えている．今後は手法の適用範囲を拡大していきたい．

2038 年問題では，time_t 型以外の変数値で制限するプログラムや，エイリアス問題を含む，time_t 値を間接的に参照するプログラムには，この手法だけでの解決は難しい．こうしたプログラムの効率的な解決方法も探っていきたい．

参考文献

[1] Apple：64-bit Transition on macOS, Apple Developers News and Update, https://developer.apple.com/news/?id=0411018a (April 11, 2018).
[2] Raspberry Pi：Raspberry Pi OS (64-bit), https://www.raspberrypi.com/news/raspberry-pi-os-64-bit/ (Feb 2nd, 2022).
[3] 大江秀幸, 松下誠, 井上克郎：組込み機器開発における 2038 年問題への対応事例, デジタルプラクティス Vol.10 No.3, (2019).
[4] 河西 朝雄 : [標準] C 言語重要用語解説 ＜ANSI C/ISO C99 対応＞, 株式会社技術評論社, (2012).
[5] WinMerge：https://winmerge.org/
[6] Martin L. Shooman：Software Engineering: Design, Reliability, and Management, McGraw-Hill, (1983).
[7] Hideyuki Oe, Makoto Matsushita, Katsuro Inoue：A Practical Approach to the Year 2038 Problem for 32-bit Embedded Systems, AsiaBSDCon 2020, (2020).
[8] 大江秀幸, 松下誠, 井上克郎：32bit UNIX システムの 2038 年問題に対するプログラム修正法の提案, 情報処理学会論文誌 62 (4), pp.1051-1055, (2021).
[9] 水上陽向, 松下誠, 井上克郎：UNIX の 2038 年問題に対する問題箇所特定ツール, ソフトウェアエンジニアリングシンポジウム 2021 論文集, pp.275-282, (2021).
[10] Ryo Okabe, Jun Yabuki, Masakatsu Toyama : Avoiding Year 2038 Problem on 32-bit Linux by Rewinding Time on Clock Synchronization, 2020 25th IEEE International Conference on ETFA, (2020).
[11] 鈴木慶汰, 窪田貴文, 河野健二：ユーザレベルアプリケーションにおける 2038 年問題の検知と解析, 情報処理学会研究報告 Vol.2019-OS-147 No.10, (2020).
[12] 星名藍乃介, 穐山空道, 上原哲太郎：32bit を超える time_t 型をもつ環境における 2038 年問題の検出手法の提案, マルチメディア, 分散, 協調とモバイルシンポジウム 2023 論文集, pp. 309-318, (2023).

正誤表

下記の箇所に誤りがございました．お詫びして訂正いたします．

訂正箇所	誤	正
6 ページ 図 10