修 士 論 文

# 特定アプリケーションのための RTOSの最適化に関する研究

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

LI　JIN

2016年5月

修 士 論 文

# 特定アプリケーションのための RTOSの最適化に関する研究


指導教員　田中　清史 准教授

審査委員主査　田中　清史准教授
審査委員　金子 峰雄 教授
審査委員　井口　寧教授

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

1310206LI　JIN



提出年月: 2016 年 5 月

## 概 要

現在、組込み開発分野における RTOS が広く利用される。RTOS を利用することにより、リアルタイム処理や資源管理の効率化が容易に実現できる。しかし、RTOS の機能は固定されており、開発対象のアプリケーションが使用しない機能を多く含むことになる。このことが、システムの実行オーバヘッドや実行バイナリのサイズに影響を与える可能性がある。RTOS はあらゆる組込みシステムで使用されることを想定している。従って、アプリケーションの実行時に起こり得る様々なエラーをチェックするコードを含む。一般的に組込みシステムでは、アプリケーションは限定される。従って、特定のアプリケーションでは、RTOS が提供する全てのエラーチェックが必要となるわけではない。また、カーネルデータの排他アクセスが必要ない場合がある。エラーチェックと同様に、ロックが必要となるわけではない。

本研究では、アプリケーションのソースコードを解析することにより、利用するシステムコール内の不必要なエラーコードを検出する方法を検討し、具体的な検出方式を提案した。次に各エラーチェックの要・不要を判断した結果を「define.h」というファイルに書き込む。ヘッダーファイルに書き込まれた値により、システムコール内のエラーチェックコードを除去することができる。続いて、アプリケーションのソースコードを解析することにより、利用するシステムコール内の不必要なロックコードを検出する方法を検討し、具体的な検出方式を提案した。次に各ロックコードの要・不要を判断した結果を「define2.h」というファイルに書き込む。ヘッダーファイルに書き込まれた値により、システムコール内のロックコードを除去することができる。本手法を利用することにより、実行オーバヘッドとバイナリサイズが削除されることを評価により示す。

# 目 次

# 第1章　はじめに

## 1.1　背景

多くの組込みシステムのアプリケーションにおいて、時間制約を満たすことが要求される。このためには、タスク間で優先度を考慮したスケジューリングを行うことが重要である。RTOS を利用することにより、時間制約を満たすスケジューリングが容易に実現可能である。また組込みシステムでは CPU やメモリの資源が限られるため、RTOS による資源管理が重要である。

RTOS はあらゆる組込みシステムで使用されることを想定している。従って、アプリケーションの実行時に起こり得る様々なエラーをチェックするコードを含む。例えば、不正な ID 番号の使用, 対象オブジェクトが不在, 不正なパラメータ値の使用などのエラーチェックを行う。これらのエラーチェックは、システム稼働時に行われるため、実行オーバヘッドが発生することになる。一般的に組込みシステムでは、アプリケーションは限定される。従って、特定のアプリケーションでは、RTOS が提供する全てのエラーチェックが必要となるわけではない。このことから、アプリケーションに従って不必要なエラーチェックコードを削除することにより、実行オーバヘッドを低減することと、バイナリコードのサイズを削減することが可能である。

システムコールは複数のタスクから同時に呼び出されることがある。重要なカーネルデータの一貫性を保障するために、システムコール内でロック機構が多用される。しかし、特定のアプリケーションでは、カーネルデータの排他アクセスが必要ない場合がある。前述のエラーチェックと同様に、ロックが不要となる箇所についてはロックのためのコードを削除することが効果的である。

RTOS はソースコードで提供される場合とオブジェクトコードで提供される場合がある。ソースコードで提供される場合は、アプリケーションに合わせてエラーチェックコードやロックコードの削除が可能である。一方、オブジェクトコードで提供される場合は変更が不可能である。このため、エラーチェック/ロックコードを含むシステムコールと含まないシステムコールの両方が提供され、アプリケーションから選択して使用することが望まれる。

本研究では、アプリケーションを解析し、エラーコード/ロックコードの要不要を判断し、自動的に RTOS をコンフィグレーションする方式を提案する。提案する方式を利用することにより、システムの開発者は容易に最適化された RTOS を利用できることが特色である。また、評価において提案方式の有効性を示すことにより、今後の組込みシステ

ム開発分野における RTOS の最適化の重要性を示す。

## 1.2 目的

現在、多くの組込みシステム開発で RTOS の利用が増加している。RTOS はリアルタイム処理や資源管理の効率化のために重要な役割を果たしている。しかし、RTOS の機能は固定化されており、開発対象のアプリケーションが使用しない機能を多く含むことになる。このことが、システムの実行オーバヘッドや実行バイナリのサイズに影響を与える可能性がある。　本研究はアプリケーションに対応して RTOS を自動的に最適化することを目的とする。対象 RTOS として、国内で広く利用されている ITRON 仕様 OS[1] をターゲットとする。ITRON は多数のシステムコールから構成され、各システムコール内で各種エラーのチェックコードやカーネルデータの一貫性保障のためのロックコードを多数含む。これらのコードは、たとえアプリケーショにとって不必要な場合でも実行されることになる。アプリケーションを解析し、システムコール内の不必要なコードを除去することにより、実行オーバヘッドとバイナリサイズの削減が可能となる。

## 1.3 論文構成

本論文は 4 章で構成される。第 2 章では，提案手法について述べる．第 3 章では評価環境について述べる．第 4 章で本論文をまとめる．

# 第2章　提案手法

本章では、本研究の提案手法に関して説明する。最初に、アプリケーションのソースコード群を解析することにより、各システムコール内の各エラーチェックの要・不要を判断する方法を説明する。次に、判断の結果に従って、エラーチェックコードの削除方法を述べる。また、アプリケーションのソースコード群を解析することにより、各システムコール内のロックコードの要・不要を判断する方法を説明する。次に、判断の結果に従って、ロックコードの削除方法を述べる。

## 2.1　各対象システムコールとエラーチェックの説明

### 2.1.1　対象システムコール

- タスク管理機能のためのシステムコール

  cre_tsk() : タスクの生成
  act_tsk() : タスクの起動

- 固定長メモリプール機能のためのシステムコール

  cre_mpf(): 固定長メモリプールの生成
  get_mpf(): 固定長メモリブロックの獲得
  rel_mpf(): 固定長メモリブロックの返却

- メールボックス機能のためのシステムコール

  cre_mbx(): メールボックスの生成
  snd_mbx(): メールボックスへの送信
  rcv_mbx(): メールボックスからの受信

- セマフォ機能のためのシステムコール

  cre_sem(): セマフォの生成
  sig_sem(): セマフォ資源の返却
  wai_sem(): セマフォ資源の獲得

### 2.1.2 対象エラーチェック

最適化の対象とするエラーチェックは以下の通りである

- タスク管理機能についてのエラーコード

  E_ID: 不正 ID 番号（指定されたタスク ID が不正あるいは使用できない）
  E_NOEXS: オブジェクト未生成（対象タスクが未登録）
  E_RSATR: 予約属性（指定されたタスク属性が不正あるいは使用できない）
  E_PAR：パラメータエラー（システムコールのその他の引数が不正）

- 固定長メモリプール機能についてのエラーコード

  E_ID: 不正 ID 番号（指定された固定長メモリプール ID が不正あるいは使用できない）
  E_RSATR: 予約属性（指定された固定長メモリプール属性が不正あるいは使用できない）
  E_NOEXS: オブジェクト未生成（対象固定長メモリプールが未登録）
  E_PAR: パラメータエラー（システムコールのその他の引数が不正）
  E_OBJ: オブジェクト状態エラー（指定された対象固定長メモリプールが登録済み）

- メールボックス機能についてのエラーコード

  E_ID: 不正 ID 番号（指定されたメールボックス ID が不正あるいは使用できない）
  E_RSATR: 予約属性（指定されたメールボックス属性が不正あるいは使用できない）
  E_NOEXS: オブジェクト未生成（対象メールボックスが未登録）
  E_PAR: パラメータエラー（システムコールのその他の引数が不正）
  E_OBJ: オブジェクト状態エラー（指定された対象メールボックスが登録済み）

- セマフォ機能についてのエラーコード

  E_ID: 不正 ID 番号（指定されたセマフォID が不正あるいは使用できない）
  E_RSATR: 予約属性（指定されたセマフォ属性が不正あるいは使用できない）
  E_NOEXS: オブジェクト未生成（対象セマフォが未登録）
  E_PAR: パラメータエラー（システムコールのその他の引数が不正）
  E_OBJ: オブジェクト状態エラー（指定された対象セマフォが登録済み）

## 2.2 提案手法の手順

### 2.2.1 各システムコール内の各エラーチェックの要・不要の判断

1. タスク管理機能の E_ID エラーチェックの要・不要を判断する手順を示す。
まず、アプリケーションのソースファイル（appl.c）を解析し、この中の“act_tsk”という

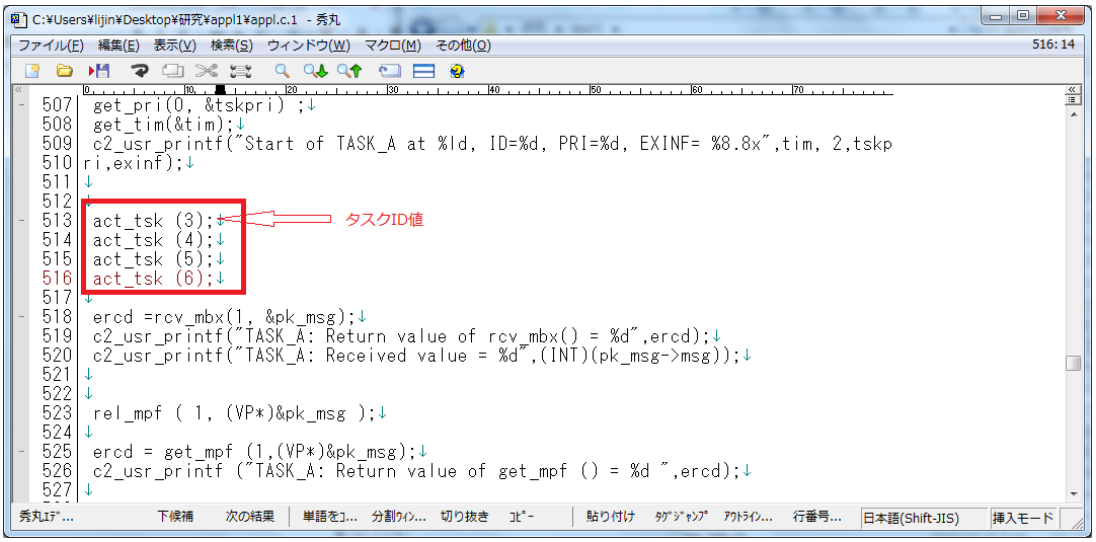


図 2.1: act_tsk の文字列が含まれている行

文字列が含まれている行を探す（図 2.1）。次に、文字列の中の引数の値を獲得して（この引数の値はタスク ID 値を意味する）保存する。保存した引数の種類を判断する。この引数が正数値を示す場合、エラーチェックの要・不要を判断できる。この引数が変数（例えば“act_tsk(i)”など）を示す場合、エラーチェックが必要である。この引数がマクロ（例えば“act_tsk(TSK_A)”など) を示す場合、このマクロが表す正数値を探して、エラーチェックの要・不要を判断できる。

エラーチェックの要・不要は以下のように判断する。

- エラーチェックが必要となる場合：
  これは E_ID エラーチェックコードを削除できない場合である。
  taskID(タスク ID 値) が 1 より小さい、また、TMAX_TSKID（タスク ID の最大値）以上となる場合。すなわち、以下の条件を満たす時である。

  $taskID < 1 \,||\, taskID >= TMAX\_TSKID$

- エラーチェックが不要となる場合：
  これは E_ID エラーチェックコードを削除できる場合である。

  $taskID >= 1 \;\&\&\; taskID < TMAX\_TSKID$

2. タスク管理機能の E_NOEXS エラーチェックの要・不要を判断する手順を示す。
まず、アプリケーションのコンフィグレーションファイル（system.cfg）を解析し、この中の“CRE_TSK”という文字列が含まれている行数を数える。数えた値は x という変数

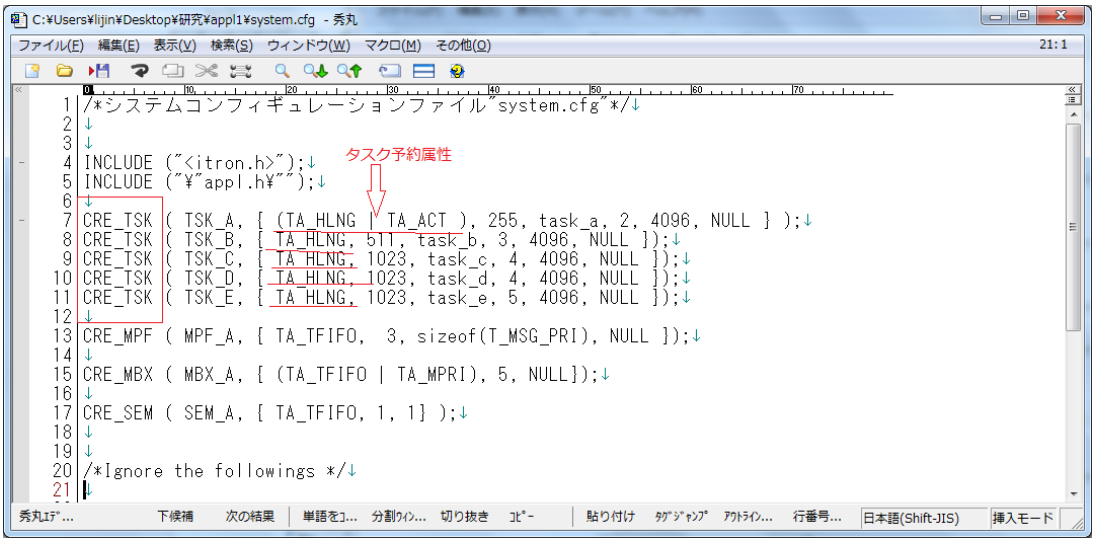


図 2.2: CRE_TSK の文字列が含まれている行におけるタスク管理機能の予約属性の獲得

に入れる。(x + 1 の値は登録したタスク ID 値の最大値を意味する)。次に、アプリケーションのソースファイル（appl.c）を解析し、この中の“act_tsk”という文字列が含まれている行を探す。最後に、その行の中で、文字列の引数の値を獲得する（この引数の値はタスク ID 値を意味する)。

- エラーチェックが必要となる場合：
  これは E_NOEXS エラーチェックコードを削除できない場合である。
  taskID(タスク ID 値) が 1 より小さい、または x+1(登録したタスク ID 値の最大値) より大きい場合。すなわち、以下の条件を満たす時である。

  $taskID < 1 \mid\mid taskID > (x + 1)$

- エラーチェックが不要となる場合：
  これは E_NOEXS エラーチェックコードを削除できる場合である。

  $taskID >= 1 \;\&\&\; taskID <= (x + 1)$

3. タスク管理機能の E_RSATR エラーチェックの要・不要を判断する手順を示す。
まず、アプリケーションのコンフィグレーションファイル（system.cfg）を解析し、この中の“CRE_TSK”という文字列が含まれている行を探す（図 2.2)。次に、その行の中でタスク予約属性を表す文字列を獲得して、buff2 というバッファに入れる。(タスクの属性には $((TA\_HLNG \mid\mid TA\_ASM) \mid (TA\_ACT))$ の指定ができる)。
タスク管理機能の予約属性は 7 種類ある。7 種類の予約属性を w1[] から w7[] までの配列に用意する。

$w1[] =$ "$TA\_HLNG$"
$w2[] =$ "$TA\_ASM$"
$w3[] =$ "$TA\_ACT$"
$w4[] =$ "$TA\_HLNG \mid TA\_ACT$"
$w5[] =$ "$TA\_ACT \mid TA\_HLNG$"
$w6[] =$ "$TA\_ASM \mid TA\_ACT$"
$w7[] =$ "$TA\_ACT \mid TA\_ASM$"

- エラーチェックが必要となる場合：
  これは E_RSATR エラーチェックコードを削除できない場合である。
  buff2 に入っているタスク予約属性の文字列が w1[] から w7[] までの配列に入る 7 種類の予約属性の中の一つではなかった場合。すなわち、以下の条件を満たす時である。

  $strcmp(buff2, w1) != 0 \;\&\&\; strcmp(buff2, w2) != 0 \;\&\&$
  $strcmp(buff2, w3) != 0 \;\&\&\; strcmp(buff2, w4) != 0 \;\&\&$
  $strcmp(buff2, w5) != 0 \;\&\&\; strcmp(buff2, w6) != 0 \;\&\&$
  $strcmp(buff2, w7) != 0$

- エラーチェックが不要となる場合：
  これは E_NOEXS エラーチェックコードを削除できる場合である。

  $strcmp(buff2, w1) == 0 \;||\; strcmp(buff2, w2) == 0 \;||$
  $strcmp(buff2, w3) == 0 \;||\; strcmp(buff2, w4) == 0 \;||$
  $strcmp(buff2, w5) == 0 \;||\; strcmp(buff2, w6) == 0 \;||$
  $strcmp(buff2, w7) == 0$

4. タスク管理機能の E_PAR エラーチェックの要・不要を判断する手順を示す。
まず、アプリケーションのコンフィグレーションファイル（system.cfg）を解析し、この中の“ CRE_TSK ”という文字列が含まれている行を探す（図 2.3）。その行の中で、タスクのパラメータを意味する文字列を獲得する。次に、ch[] という配列を用意して、ch[3] にはタスクの起動番地 (task) というタスクのパラメータを保存し、ch[4] にはタスクの起動時優先度 (itskpri) というタスクのパラメータを保存する。 ch[5] にはタスクのスタック領域のサイズ (stksz) というタスクのパラメータを保存し、ch[6] にはタスクのスタック領域の先頭番地 (stk) というタスクのパラメータを保存する。続いて、ch[4] に保存されている文字列を整数値に変換し、itskpri という変数に書き込み、ch[5] に保存されている文字列を整数値に変換し、stksz という変数に書き込む。

- エラーチェックが必要となる場合：
  これは E_PAR エラーチェックコードを削除できない場合である。
  ch[3](タスクの起動番地) は"NULL" という文字列と同じ場合。また、itskpri(タスクの

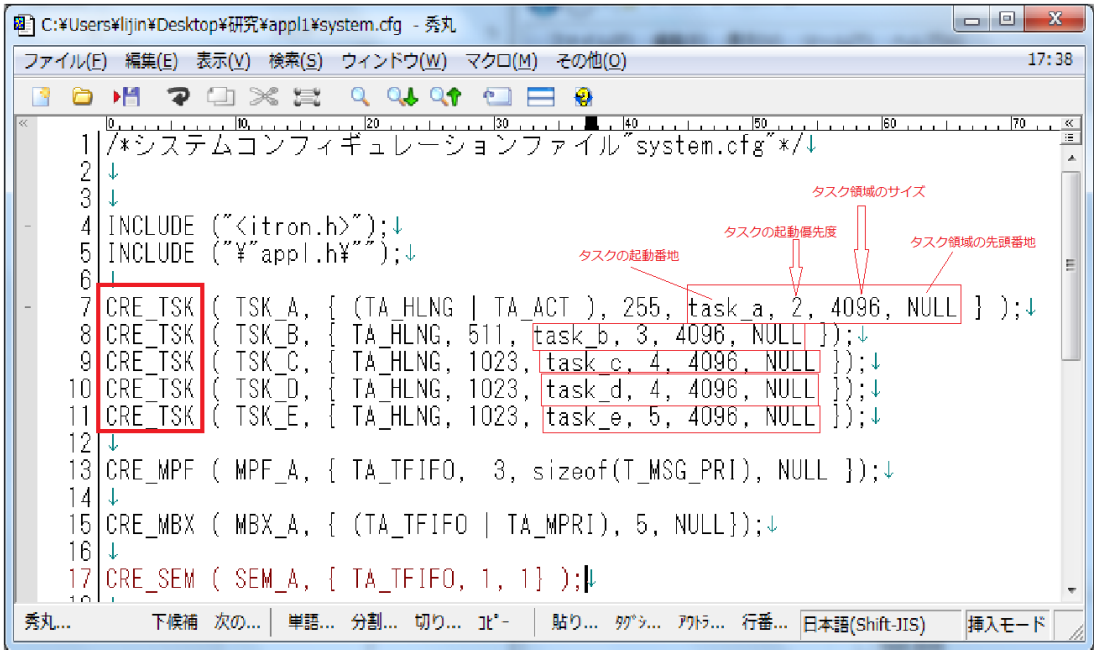


図 2.3: CRE_TSK の文字列が含まれている行におけるタスクのパラメータの獲得

起動時優先度) の値が 0 以下、TMAX_TPRI(タスクの起動時優先度の最大値) より大きい場合。もしくは、stksz(タスクのスタック領域のサイズ) の値が TMAX_STKSZ(タスクのスタック領域のサイズの最大値) より大きい場合。もしくは、ch[6](タスクのスタック領域の先頭番地) は"NULL" という文字列と違う場合。すなわち、以下の条件を満たす時である。

$strcmp(ch[3],"NULL") == 0 \ \ ||$
$((tpri <= 0) \ \ || \ \ (tpri > TMAX\_TPRI)) \ \ ||$
$stksz > TMAX\_STKSZ \ \ ||$
$strcmp(ch[6],"NULL") != 0$

- エラーチェックが不要となる場合：
  これは E_PAR エラーチェックコードを削除できる場合である。

  $strcmp(ch[3],"NULL") != 0 \ \ \&\&$
  $((tpri > 0) \ \ \&\& \ \ (tpri <= TMAX\_TPRI)) \ \ \&\&$
  $stksz <= TMAX\_STKSZ \ \ \&\&$
  $strcmp(ch[6],"NULL") == 0$

5. 固定長メモリプール機能の E_ID エラーチェックの要・不要を判断する手順を示す。まず、アプリケーションのソースファイル（appl.c）を解析し、この中の“get_mpf”という文字列が含まれている行を探し（図 2.4）、その文字列の引数の値を獲得する（この引数

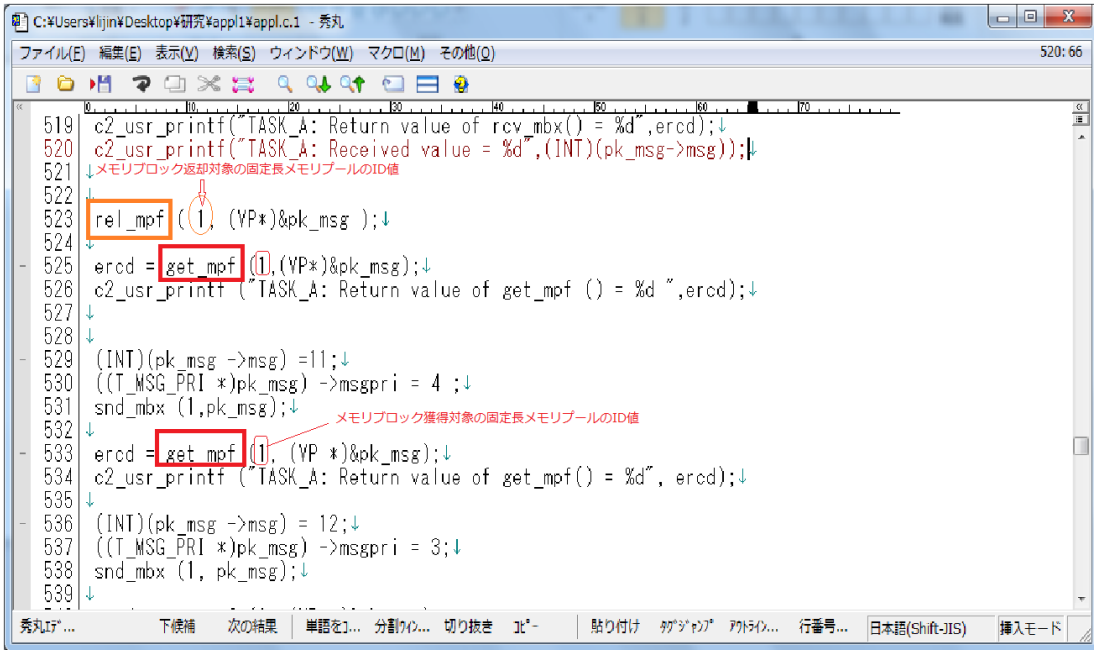


図 2.4: get_mpf と rel_mpf の文字列が含まれている行

の値はメモリブロック獲得対象の固定長メモリプールの ID 値 (get_mpfID) を意味する)。次に、同様に“ rel_mpf ”という文字列が含まれている行を探し（図 2.4)、その文字列の引数の値を獲得する（この引数の値はメモリブロック返却対象の固定長メモリプールの ID 値 (rel_mpf ID) を意味する)。

エラーチェックの要・不要は以下のように判断する。

- エラーチェックが必要となる場合：
  これは E_ID エラーチェックコードを削除できない場合である。
  get_mpfID(メモリブロック獲得対象の固定長メモリプールの ID 値) が 1 より小さい、または TMAX_MPFID(固定長メモリプールの ID 値の最大値) 以上となる場合。もしくは，rel_mpf(メモリブロック返却対象の固定長メモリプールの ID 値) が 1 より小さい、または TMAX_MPFID(固定長メモリプールの ID 値の最大値) 以上となる場合。すなわち、以下の条件を満たす時である。

  $(get\_mpfID < 1 \ || \ get\_mpfID >= TMAX\_MPFID) \ ||$
  $(rel\_mpfID < 1 \ || \ rel\_mpfID >= TMAX\_MPFID)$

- エラーチェックが不要となる場合：
  これは E_ID エラーチェックコードを削除できる場合である。

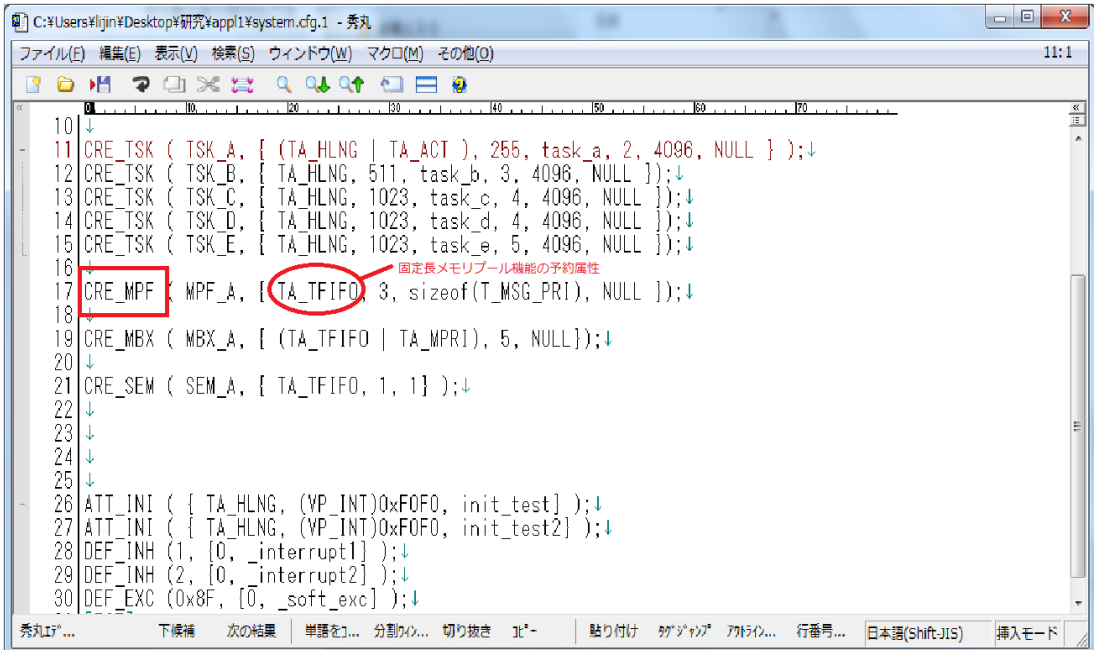


図 2.5: CRE_MPF の文字列が含まれている行における固定長メモリプールの予約属性の獲得

$$(get\_mpfID >= 1 \ \ \&\& \ \ get\_mpfID < TMAX\_MPFID) \ \ \&\&$$
$$(rel\_mpfID >= 1 \ \ \&\& \ \ rel\_mpfID < TMAX\_MPFID)$$

6. 固定長メモリプール機能の E_RSATR エラーチェックの要・不要を判断する手順を示す。
まず、アプリケーションのコンフィグレーションファイル（system.cfg）を解析し、この中の“CRE_MPF”という文字列が含まれている行を探す（図 2.5）。次に、その行の中で固定長メモリプール機能の予約属性を表す文字列を獲得して、buff2 というバッファに入れる。（固定長メモリプールの予約属性には $(TA\_TFIFO \ || \ TA\_TPRI)$ の指定ができる）。

固定長メモリプール機能の予約属性は 2 種類ある。2 種類の予約属性を w1[] から w2[] までの配列に用意する。

w1[] = ”TA_TFIFO”
w2[] = ”TA_TPRI”

- エラーチェックが必要となる場合：
  これは E_RSATR エラーチェックコードを削除できない場合である。
  buff2 に入っている固定長メモリプールの予約属性の文字列が w1[] から w2[] までの

配列に入る 2 種類の予約属性の中の一つではなかった場合。すなわち、以下の条件を満たす時である。

$strcmp(buff2, w1) != 0 \ \ \&\& \ \ strcmp(buff2, w2) != 0$

- エラーチェックが不要となる場合：
これは E_RSATR ラーチェックコードを削除できる場合である。

$strcmp(buff2, w1) == 0 \ \ || \ \ strcmp(buff2, w2) == 0$

7. 固定長メモリプール機能の E_NOEXS エラーチェックの要・不要を判断する手順を示す。
まず、アプリケーションのコンフィグレーションファイル（system.cfg）を解析し、この中の“ CRE_MPF ”という文字列が含まれている行数を数える。数えた値は x という変数に入れる ( x の値は登録した固定長メモリプールの ID 値の最大値を意味する)。次に、アプリケーションのソースファイル（appl.c）を解析し、この中の“ get_mpf ”という文字列が含まれている行を探す（図 2.4）。その行の中で、文字列の引数の値を獲得する（この引数の値はメモリブロック獲得対象の固定長メモリプールの ID 値 (get_mpfID) を意味する）。同様に“ rel_mpf ” という文字列が含まれている行を探し（図 2.4），文字列の引数の値を獲得する（この引数の値はメモリブロック返却対象の固定長メモリプールの ID 値 (rel_mpf ID) を意味する）。

- エラーチェックが必要となる場合：
これは E_NOEXS エラーチェックコードを削除できない場合である。
get_mpfID(メモリブロック獲得対象の固定長メモリプールの ID 値) が 1 より小さい、x(登録した固定長メモリプールの ID 値の最大値) より大きい場合。また、rel_mpf(メモリブロック返却対象の固定長メモリプールの ID 値) は 1 より小さい、x(登録した固定長メモリプールの ID の値の最大値) より大きい場合。すなわち、以下の条件を満たす時である。

$(get\_mpfID < 1 \ \ || \ \ get\_mpfID > x) \ \ ||$
$(rel\_mpfID < 1 \ \ || \ \ rel\_mpfID > x)$

- エラーチェックが不要となる場合：
これは E_NOEXS エラーチェックコードを削除できる場合である。

$(get\_mpfID >= 1 \ \ \&\& \ \ get\_mpfID =< x) \ \ \&\&$
$(rel\_mpfID >= 1 \ \ \&\& \ \ rel\_mpfID =< x)$

8. 固定長メモリプール機能の E_PAR エラーチェックの要・不要を判断する手順を示す。
まず、アプリケーションのコンフィグレーションファイル（system.cfg）を解析し、この

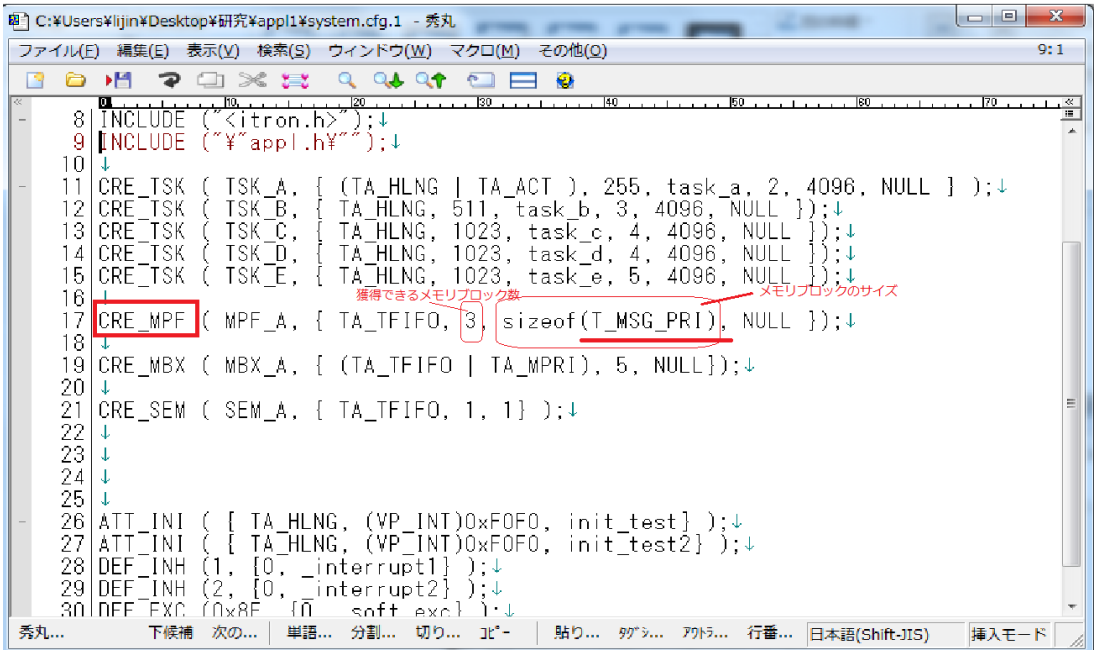


図 2.6: CRE_MPF の文字列が含まれている行における固定長メモリプールのパラメータの獲得

中の“CRE_MPF”という文字列が含まれている行を探す。その行の中で，固定長メモリプール機能のパラメータを意味する文字列を獲得する（図 2.6）。次に、ch[] という配列を用意して、ch[2] には獲得できるメモリブロック数 (blkcnt) という 固定長メモリプールのパラメータを保存し、ch[3] にはメモリブロックのサイズ (blksz) という固定長メモリプールのパラメータを保存する。次に、ch[2] に保存されている文字列を整数値に変換し、pk_cmpf_blkcnt という変数に書き込む。続いて、ch[3] に保存されている文字列において，sizeof という文字列の中に含まれる文字列を判断して pk_cmpf_blksz という変数に値を書き込む。もし”T_MSG” という文字列が含まれるなら 8、 ”T_MSG_PRI”という文字列なら 12、そして他の文字列なら 0 を書き込む。

- エラーチェックが必要となる場合：
  これは E_PAR エラーチェックコードを削除できない場合である。
  pk_cmpf_blkcnt(獲得できるメモリブロック数の値) が 0 以下、または TMAX_MPFBCNT(獲得できるメモリブロック数の最大値) より大きい場合。もしくは、pk_cmpf_blksz(メモリブロックのサイズの値) が 0 以下、または TMAX_MPFBSZ(メモリブロックのサイズの最大値) より大きい場合。すなわち、以下の条件を満たす時である。

  $((pk\_cmpf\_blkcnt <= 0) \ || \ (pk\_cmpf\_blkcnt > TMAX\_MPFBCNT)) \ ||$
  $((pk\_cmpf\_blksz <= 0) \ || \ (pk\_cmpf\_blksz > TMAX\_MPFBSZ))$

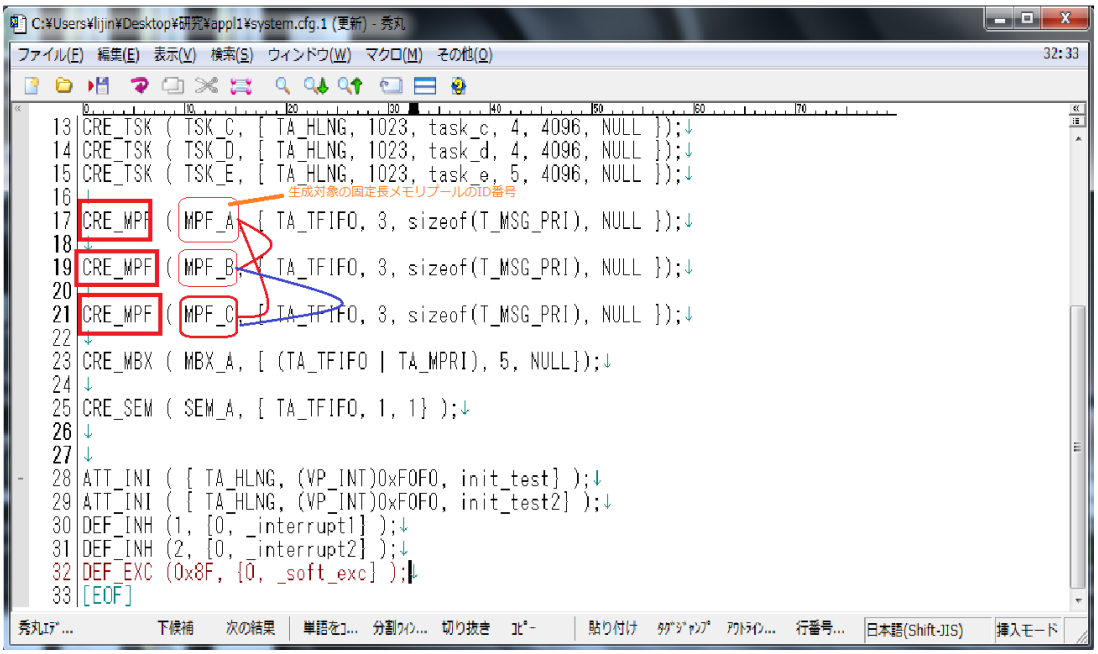


図 2.7: CRE_MPF の文字列が含まれている行における固定長メモリプールのオブジェクト状態の判断

- エラーチェックが不要となる場合：
  これは E_PAR エラーチェックコードを削除できる場合である。

$$((pk\_cmpf\_blkcnt > 0) \ \&\& \ (pk\_cmpf\_blkcnt <= TMAX\_MPFBCNT))\&\& \\ ((pk\_cmpf\_blksz > 0) \ \&\& \ (pk\_cmpf\_blksz <= TMAX\_MPFBSZ))$$

9. 固定長メモリプール機能の E_OBJ エラーチェックの要・不要を判断する手順を示す。まず、history [] という配列を用意する。アプリケーションのコンフィグレーションファイル（system.cfg）を解析し、この中の“ CRE_MPF ”という文字列が含まれている行を探す。その行から生成対象の固定長メモリプールの ID 番号を表す文字列を獲得し (図 2.7 の MPF_A)、history [] に保存する (history[1] = “ MPF_A ”)。次に、“ CRE_MPF ”の文字列が含まれる次の行を探して、その行から生成対象の固定長メモリプールの ID 番号を表す文字列 (MPF_B) を獲得して history [] に保存する (history[2] = “ MPF_B ”)。順次、次の“ CRE_MPF ”の文字列が含まれる行を探して、その行から生成対象の固定長メモリプールの ID 番号を表す文字列 (MPF_C) を獲得し、history [] に保存する (history[3] = “ MPF_C ”)。history [] に保存された文字列を比較して、同じ文字があるかどうかを調べる。

- E_OBJ エラーチェックが必要となる場合：

これはE_OBJエラーチェックコードを削除できない場合である。
上述する方法で、history [] に保存された文字列を比較し、同じ文字列がある場合である。すなわち、以下の条件を満たす時である。

$$strcmp(history[0], history[1]) == 0 \ ||$$
$$strcmp(history[0], history[2]) == 0 \ ||$$
$$strcmp(history[1], history[2]) == 0$$

- E_OBJ エラーチェックが不要となる場合：
これはE_OBJエラーチェックコードを削除できる場合である。

$$strcmp(history[0], history[1]) != 0 \ \&\&$$
$$strcmp(history[0], history[2]) != 0 \ \&\&$$
$$strcmp(history[1], history[2]) != 0$$

10. メールボックス機能のE_IDエラーチェックの要・不要を判断する手順を示す。
まず、アプリケーションのソースファイル（appl.c）を解析し、この中の“snd_mbx”という文字列が含まれている行を探し（図2.8）、その文字列の引数の値を獲得する（引数の値は送信対象のメールボックスのID値(snd_mbxID)を意味する）。次に、同様に“rcv_mbx”という文字列が含まれている行を探し（図2.8）、その文字列の引数の値を獲得する（引数の値は受信対象のメールボックスのID値(rcv_mbxID)を意味する）。

- エラーチェックが必要となる場合：
これはE_IDエラーチェックコードを削除できない場合である。
snd_mbxID（送信対象のメールボックスのID値）が1より小さい、またはTMAX_MBXID（メールボックスID値の最大値）以上となる場合。もしくは、rcv_mbxID(受信対象のメールボックスのID値)が1より小さい、またはTMAX_MBXID(メールボックスID値の最大値)以上となる場合。すなわち、以下の条件を満たす時である。

$$(snd\_mbxID < 1 \ \ || \ \ snd\_mbxID >= TMAX\_MBXID) \ \ ||$$
$$(rcv\_mbxID < 1 \ \ || \ \ rcv\_mbxID >= TMAX\_MBXID)$$

- エラーチェックが不要となる場合：
これはE_ID エラーチェックコードを削除できる場合である。

$$(snd\_mbxID >= 1 \ \ \&\& \ \ snd\_mbxID < TMAX\_MBXID) \ \ \&\&$$
$$(rcv\_mbxID >= 1 \ \ \&\& \ \ rcv\_mbxID < TMAX\_MBXID)$$

11. メールボックス機能のE_RSATRエラーチェックの要・不要を判断する手順を示す。
まず、アプリケーションのコンフィグレーションファイル（system.cfg）を解析し、この

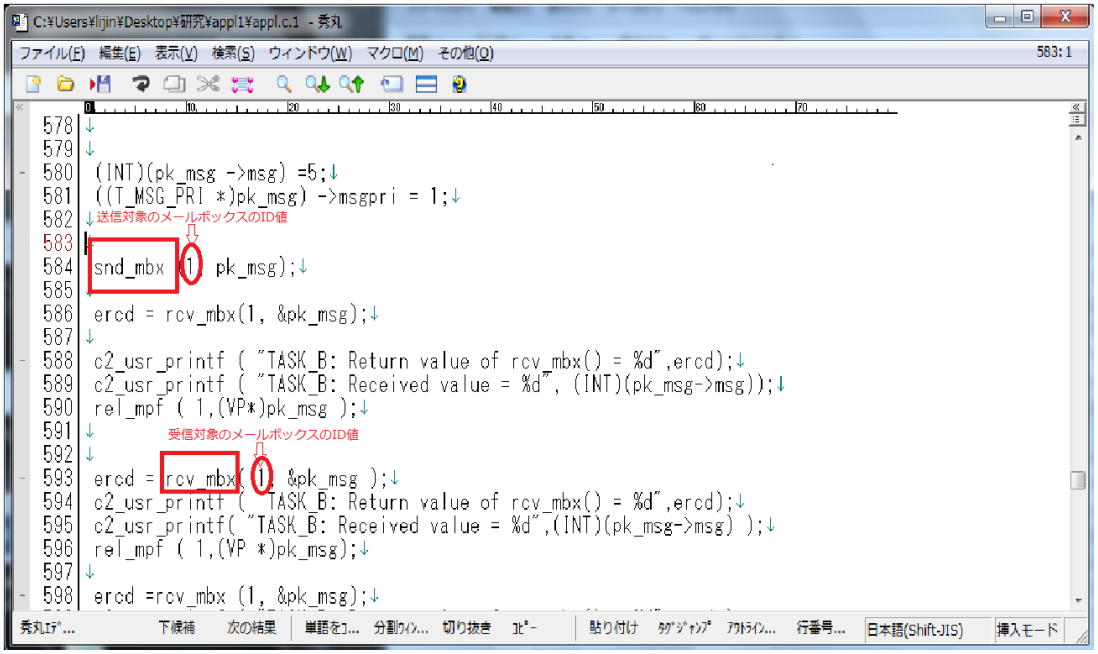


図 2.8: snd_mbx と rcv_mbx の文字列が含まれている行

中の“CRE_MBX”という文字列が含まれている行を探す（図 2.9）。次に、その行の中で、メールボックス機能の予約属性を表す文字列を獲得して、buff2 というバッファに入れる。（タスクの予約属性には $((TA\_TFIFO \mid\mid TA\_TPRI) \mid (TA\_MFIFO \mid\mid TA\_MPRI))$ の指定ができる）。

メールボックス機能の予約属性は 12 種類ある。12 種類の予約属性を w1[] から w12[] までの配列に用意する。

$w1[] = "TA\_TFIFO"$
$w2[] = "TA\_TPRI"$
$w3[] = "TA\_MFIFO"$
$w4[] = "TA\_MPRI"$
$w5[] = "TA\_TFIFO \mid TA\_MFIFO"$
$w6[] = "TA\_TFIFO \mid TA\_MPRI"$
$w7[] = "TA\_TPRI \mid TA\_MFIFO"$
$w8[] = "TA\_TPRI \mid TA\_MPRI"$
$w9[] = "TA\_MFIFO \mid TA\_TFIFO"$
$w10[] = "TA\_MFIFO \mid TA\_TPRI"$
$w11[] = "TA\_MPRI \mid TA\_TFIFO"$
$w12[] = "TA\_MPRI \mid TA\_TPRI"$

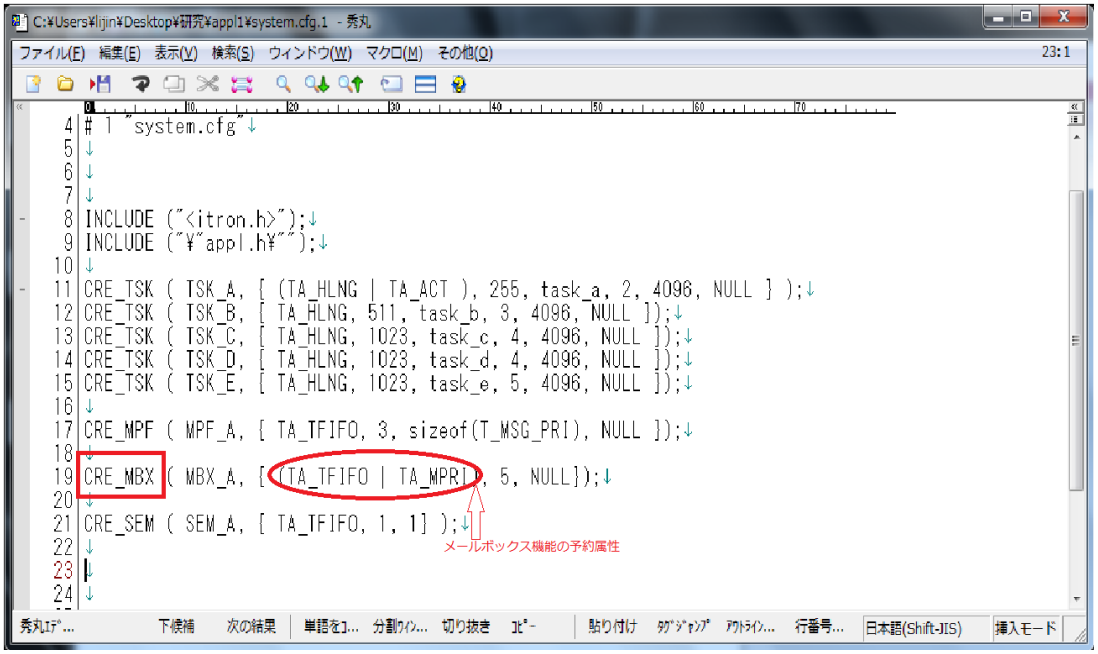


図 2.9: CRE_MBX の文字列が含まれている行におけるメールボックスの予約属性の獲得

- エラーチェックが必要となる場合：
  これは E_RSATR エラーチェックコードを削除できない場合である。
  buff2 に入っているメールボックス予約属性の文字列が w1[] から w12[] までの配列に入る 12 種類の予約属性の中の一つではなかった場合。すなわち、以下の条件を満たす時である。

  $$strcmp(buff2, w1) != 0 \;\; \&\& \;\; strcmp(buff2, w2) != 0 \;\; \&\&$$
  $$strcmp(buff2, w3) != 0 \;\; \&\& \;\; strcmp(buff2, w4) != 0 \;\; \&\&$$
  $$strcmp(buff2, w5) != 0 \;\; \&\& \;\; strcmp(buff2, w6) != 0 \;\; \&\&$$
  $$strcmp(buff2, w7) != 0 \;\; \&\& \;\; strcmp(buff2, w8) != 0 \;\; \&\&$$
  $$strcmp(buff2, w9) != 0 \;\; \&\& \;\; strcmp(buff2, w10) != 0 \;\; \&\&$$
  $$strcmp(buff2, w11) != 0 \;\; \&\& \;\; strcmp(buff2, w12) != 0$$

- エラーチェックが不要となる場合：
  これは E_RSATR エラーチェックコードを削除できる場合である。

  $$strcmp(buff2, w1) == 0 \;\; || \;\; strcmp(buff2, w2) == 0 \;\; ||$$
  $$strcmp(buff2, w3) == 0 \;\; || \;\; strcmp(buff2, w4) == 0 \;\; ||$$
  $$strcmp(buff2, w5) == 0 \;\; || \;\; strcmp(buff2, w6) == 0 \;\; ||$$

$strcmp(buff2, w7) == 0 \quad || \quad strcmp(buff2, w8) == 0 \quad ||$
$strcmp(buff2, w9) == 0 \quad || \quad strcmp(buff2, w10) == 0 \quad ||$
$strcmp(buff2, w11) == 0 \quad || \quad strcmp(buff2, w12) == 0$

12. メールボックス機能の E_NOEXS エラーチェックの要・不要を判断する手順を示す。まず、アプリケーションのコンフィグレーションファイル（system.cfg）を解析し、この中の“CRE_MBX”という文字列が含まれている行数を数える。数えた値は x という変数に入れる ( x の値は登録したメールボックスの ID 値の最大値を意味する)。次に、アプリケーションのソースファイル（appl.c）を解析し、この中の“snd_mbx”という文字列が含まれている行を探す（図 2.8)。その行の中で、文字列の中の引数の値を獲得する（この引数の値は送信対象のメールボックスの ID 値 (snd_mbxID) を意味する)。同様に“rcv_mbx”という文字列が含まれている行を探し、文字列の引数の値を獲得する (この引数の値は受信対象のメールボックスの ID 値 (rcv_mbxID) を意味する)。

- エラーチェックが必要となる場合：
  これは E_NOEXS エラーチェックコードを削除できない場合である。
  snd_mbxID(送信対象のメールボックスの ID 値) が 1 より小さい、x(登録したメールボックスの ID 値の最大値) より大きい場合。また、rcv_mbx(受信対象のメールボックスの ID 値) は 1 より小さい、x(登録したメールボックスの ID 値の最大値) より大きい場合。すなわち 、以下の条件を満たす時である。

  $(snd\_mbxID < 1 \ || \ snd\_mbxID > x) \ ||$
  $(rcv\_mbxID < 1 \ || \ rcv\_mbxID > x)$

- エラーチェックが不要となる場合：
  これは E_NOEXS エラーチェックコードを削除できる場合である。

  $(snd\_mbxID >= 1 \ \&\& \ snd\_mbxID <= x) \ \&\&$
  $(rcv\_mbxID >= 1 \ \&\& \ rcv\_mbxID <= x)$

13. メールボックス機能の E_PAR エラーチェックの要・不要を判断する手順を示す。まず、アプリケーションのコンフィグレーションファイル（system.cfg）を解析し、この中の“CRE_MBX”という文字列が含まれている行を探す（図 2.10)。その行の中で、メールボックス機能の予約属性とパラメータを意味する文字列を獲得する。次に、ch[] という配列を用意する。ch[1] にはメールボックス機能の予約属性を意味する文字列を保存し、ch[2] には送信されるメールボックスの優先度の最大値 (maxmpri) というメールボックスのパラメータを保存し、 ch[3] には優先度別のメッセージキュ―ヘッダ領域の先頭番地 (mprihd) というメールボックスのパラメータを保存する。続いて、ch[1] の中に” TA_MPRI ”という文字列が含まれるなら、ch[2] に保存されている文字列を整数値に変換し、maxmpri という変数に書き込む。

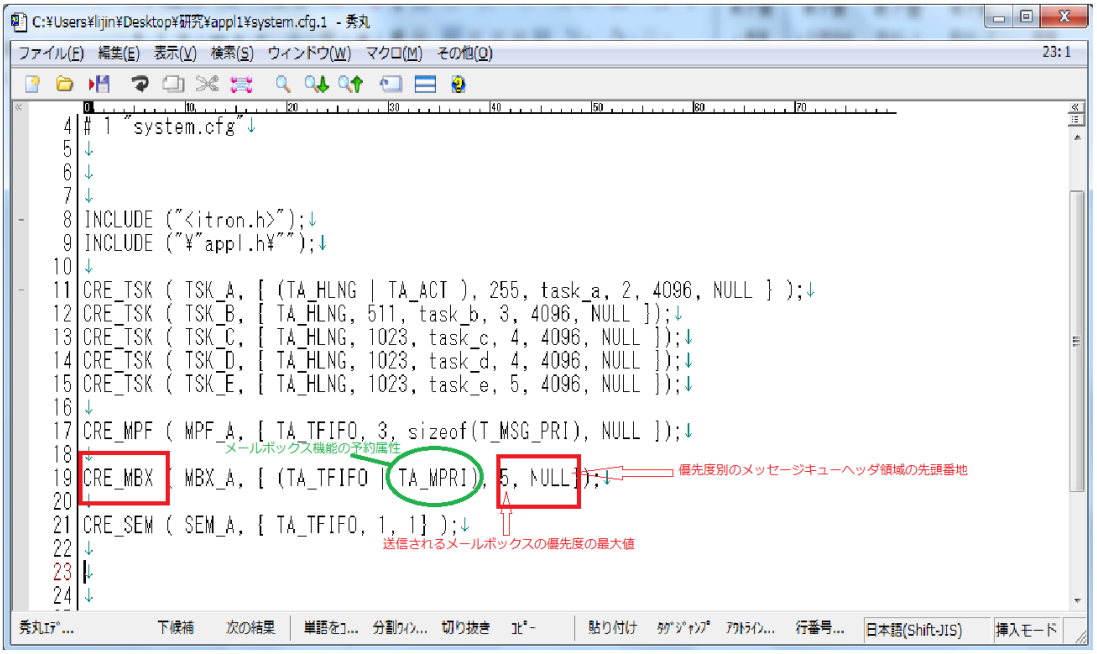


図 2.10: CRE_MBX の文字列が含まれている行におけるメールボックスのパラメータの獲得

- エラーチェックが必要となる場合：
  これは E_PAR エラーチェックコードを削除できない場合である。
  メールボックス機能の予約属性を表す文字列の中に” TA_MPRI ”という文字列が含まれるなら、maxmpri(送信されるメッセージの優先度の最大値) が 0 より小さい、またはTMAX_MPRI(メッセージ優先度の最大値) より大きい場合。もしくは、ch[3](優先度別のメッセージキューヘッダ領域の先頭番地) が NULL ではない場合。すなわち、以下の条件を満たす時である。

  $(maxmpri < 0 \ || \ maxmpri > TMAX\_MPRI) \ ||$
  $strcmp(ch[3], "NULL") != 0$

- エラーチェックが不要となる場合：
  これは E_PAR エラーチェックコードを削除できる場合である。

  $(maxmpri > 0 \ \&\& \ maxmpri <= TMAX\_MPRI) \ \&\&$
  $strcmp(ch[3], "NULL") == 0$

14. メールボックス機能の E_OBJ エラーチェックの要・不要を判断する手順を示す。まず、history [] という配列を用意する。次に、アプリケーションのコンフィグレーションファイル（system.cfg）を解析し、この中の“ CRE_MBX ”という文字列が含まれている行を探す。その行から生成対象のメールボックスの ID 番号を表す文字列を獲得し (図

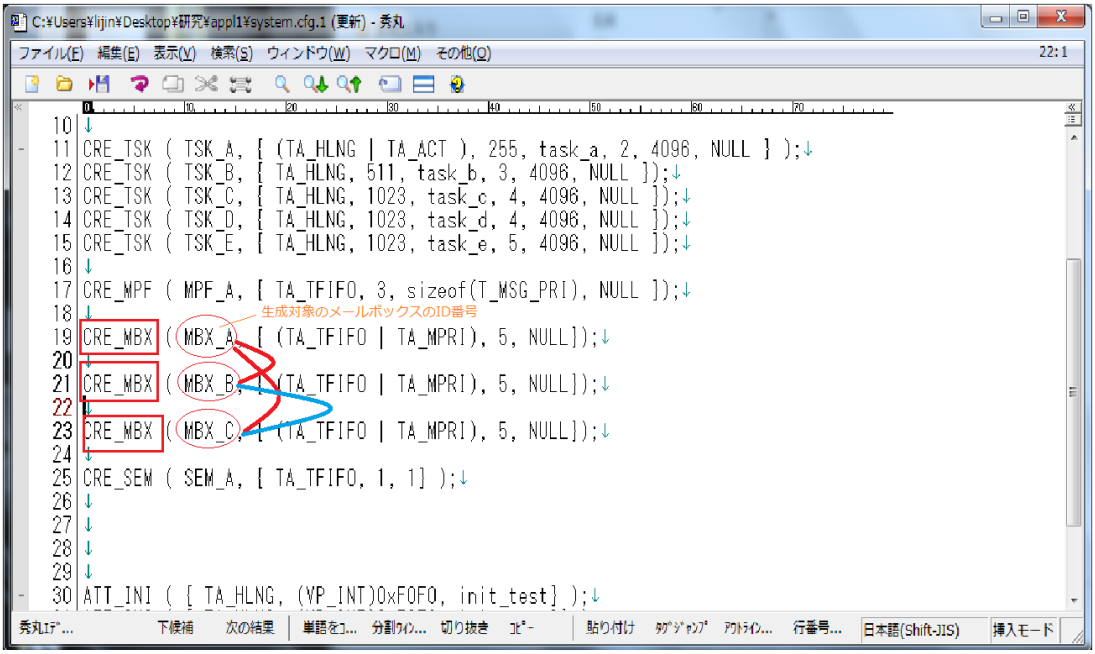


図 2.11: CRE_MBX の文字列が含まれている行におけるメールボックスのオブジェクト状態の判断

2.11 の MBX_A)、history [] に保存する (history[0] = “ MBX_A ”)。次に、“ CRE_MBX ”の文字列が含まれる次の行を探して、その行から生成対象のメールボックスの ID 番号を表す文字列 (MBX_B) を獲得して history [] に保存する (history[1] = “ MBX_B ”)。順次、次の“ CRE_MBX ”の文字列が含まれる行を探して、その行から生成対象のメールボックスの ID 番号を表す文字列 (MBX_C) を獲得し、history [] に保存する (history[2] = “ MBX_C ”)。history [] に保存された文字列を比較して、同じ文字があるかどうかを調べる。

- エラーチェックが必要となる場合：
  これは E_OBJ エラーチェックコードを削除できない場合である。
  上述する方法で、history [] に保存された文字列を比較し、同じ文字列がある場合である。すなわち、以下の条件を満たす時である。

  $$strcmp(history[0], history[1]) == 0 \ ||$$
  $$strcmp(history[0], history[2]) == 0 \ ||$$
  $$strcmp(history[1], history[2]) == 0$$

- エラーチェックが不要となる場合：
  これは E_OBJ エラーチェックコードを削除できる場合である。

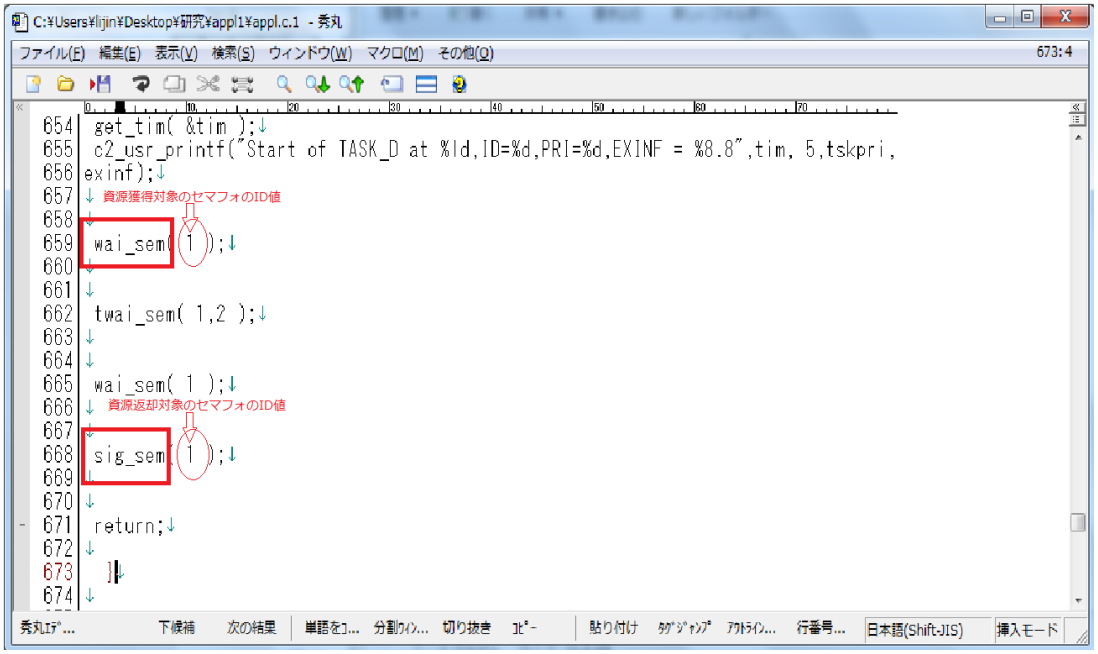


図 2.12: sig_sem と wai_sem の文字列が含まれている行

$$strcmp(history[0], history[1]) \; ! = 0 \;\&\&$$
$$strcmp(history[0], history[2]) \; ! = 0 \;\&\&$$
$$strcmp(history[1], history[2]) \; ! = 0$$

15. セマフォ機能の E_ID エラーチェックの要・不要を判断する手順を示す。
まず、アプリケーションのソースファイル（appl.c）を解析し、この中の“sig_sem”という文字列が含まれている行を探す（図 2.12）。次に、その文字列の引数の値を獲得する (この引数の値は資源返却対象のセマフォの ID 値 (sig_semID) を意味する)。次に、同様に“wai_sem”という文字列が含まれている行を探し、その文字列の引数の値を獲得する (この引数の値は資源獲得対象のセマフォの ID 値 (wai_semID) を意味する)。

エラーチェックの要・不要は以下のように判断する。

- エラーチェックが必要となる場合：
  これは E_ID エラーチェックコードを削除できない場合である。
  sig_semID(資源返却対象のセマフォの ID 値) が 1 より小さい、または MAX_SEMID(セマフォID 値の最大値) 以上となる場合。もしくは、wai_sem(資源獲得対象のセマフォの ID 値) が 1 より小さい、TMAX_SEMID(セマフォID 値の最大値) 以上となる場合。すなわち、以下の条件を満たす時である。

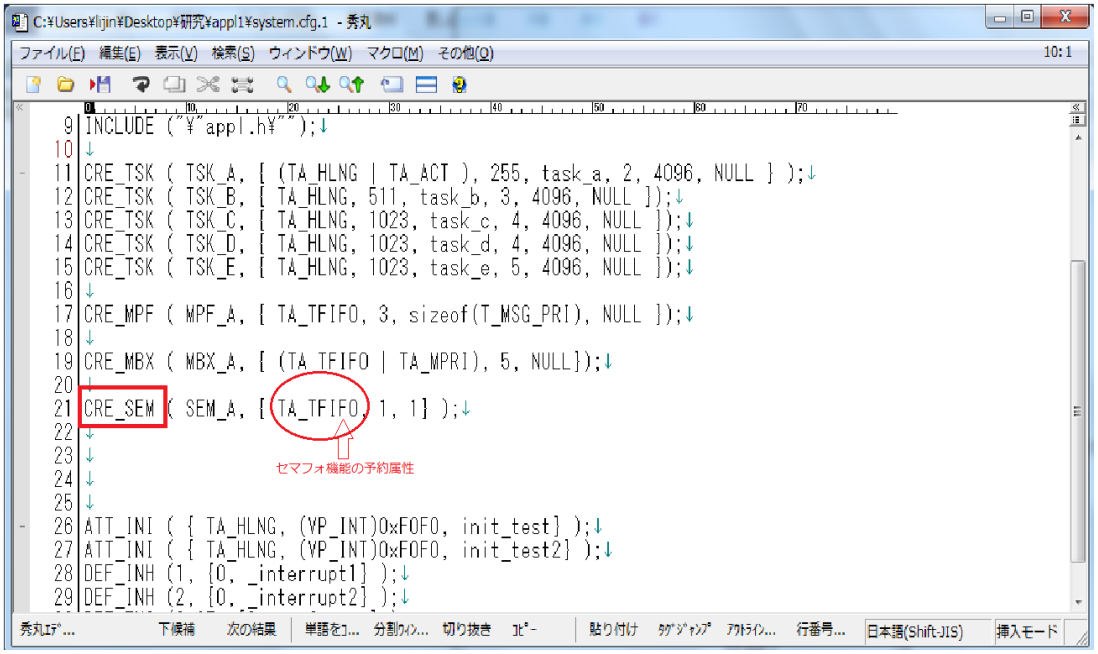


図 2.13: CRE_SEM の文字列が含まれている行におけるセマフォの予約属性の獲得

$$(sig\_semID < 1 \ \ || \ \ sig\_semID >= TMAX\_SEMID) \ \ ||$$
$$(wai\_semID < 1 \ \ || \ \ wai\_semID >= TMAX\_SEMID)$$

- エラーチェックが不要となる場合：
  これは E_ID エラーチェックコードを削除できる場合である。

$$(sig\_semID >= 1 \ \ \&\& \ \ sig\_semID < TMAX\_SEMID) \ \ \&\&$$
$$(wai\_semID >= 1 \ \ \&\& \ \ wai\_semID < TMAX\_SEMID)$$

16. セマフォ機能の E_RSATR エラーチェックの要・不要を判断する手順を示す。
まず、アプリケーションのコンフィグレーションファイル（system.cfg）を解析し、この中の“ CRE_SEM という文字列が含まれている行を探す（図 2.13）。次に、その行の中でセマフォ機能の予約属性を表す文字列を獲得して、buff2 というバッファに入れる。（セマフォ機能の予約属性には $(TA\_TFIFO \ || \ TA\_TPRI)$ の指定ができる）。

セマフォ機能の予約属性は 2 種類である。2 種類の予約属性を w1[] から w2[] までの配列に用意する。
char w1[] = ”TA_TFIFO”
char w2[] = ”TA_TPRI”

- エラーチェックが必要となる場合：
  これは E_RSATR エラーチェックコードを削除できない場合である。

buff2 に入っているセマフォの予約属性の文字列が w1[] から w2[] までの配列に入る 2 種類の予約属性の中の一つではなかった場合。すなわち、以下の条件を満たす時である。

$strcmp(buff2, w1) != 0 \ \ \&\& \ \ strcmp(buff2, w2) != 0$

- エラーチェックが不要となる場合：
これは E_RSATR エラーチェックコードを削除できる場合である。

$strcmp(buff2, w1) == 0 \ \ || \ \ strcmp(buff2, w2) == 0$

17. セマフォ機能の E_NOEXS エラーチェックの要・不要を判断する手順を示す。
まず、アプリケーションのコンフィグレーションファイル（system.cfg）を解析し、この中の“ CRE_SEM ”という文字列が含まれている行数を数える。数えた値は x という変数に入れる ( x の値は登録したセマフォの ID 値の最大値を意味する)。次に、アプリケーションのソースファイル（appl.c）を解析し、この中の“ sig_sem ”という文字列が含まれている行を探す（図 2.12）。その行の中で、文字列の引数の値を獲得する (この文字列の引数の値は資源返却対象のセマフォの ID 値 (sig_semID) を意味する)。同様に“ wai_sem ”という文字列が含まれている行を探し、文字列の引数の値を獲得する (この引数の値は資源獲得対象のセマフォの ID 値 (wai_semID) を意味する)。

- エラーチェックが必要となる場合：
これは E_NOEXS エラーチェックコードを削除できない場合である。
sig_semID(資源返却対象のセマフォの ID 値) が 1 より小さい、x(登録したセマフォの ID 値の最大値) より大きい場合。また、wai_semID(資源獲得対象のセマフォの ID 値) が 1 より小さい、x (登録したセマフォの ID 値の最大値) より大きい場合。すなわち、以下の条件を満たす時である。

$(sig\_semID < 1 \ \ || \ \ sig\_semID > x) \ \ ||$
$(wai\_semID < 1 \ \ || \ \ wai\_semID > x)$

- エラーチェックが不要となる場合：
これは E_NOEXS エラーチェックコードを削除できる場合である。

$(sig\_semID >= 1 \ \ \&\& \ \ sig\_semID <= x) \ \ \&\&$
$(wai\_semID >= 1 \ \ \&\& \ \ wai\_semID <= x)$

18. セマフォ機能の E_PAR エラーチェックの要・不要を判断する手順を示す。
まず、アプリケーションのコンフィグレーションファイル（system.cfg）を解析し、“CRE_SEM ”という文字列が含まれている行を探し（図 2.14）、この行の中で、セマフォのパラメータを意味する文字列を獲得する。次に ch[] という配列を用意して、ch[2] にはセマフォの資源数の初期値 (isemcnt) というセマフォのパラメータを保存し, ch[3] にはセマフォの最

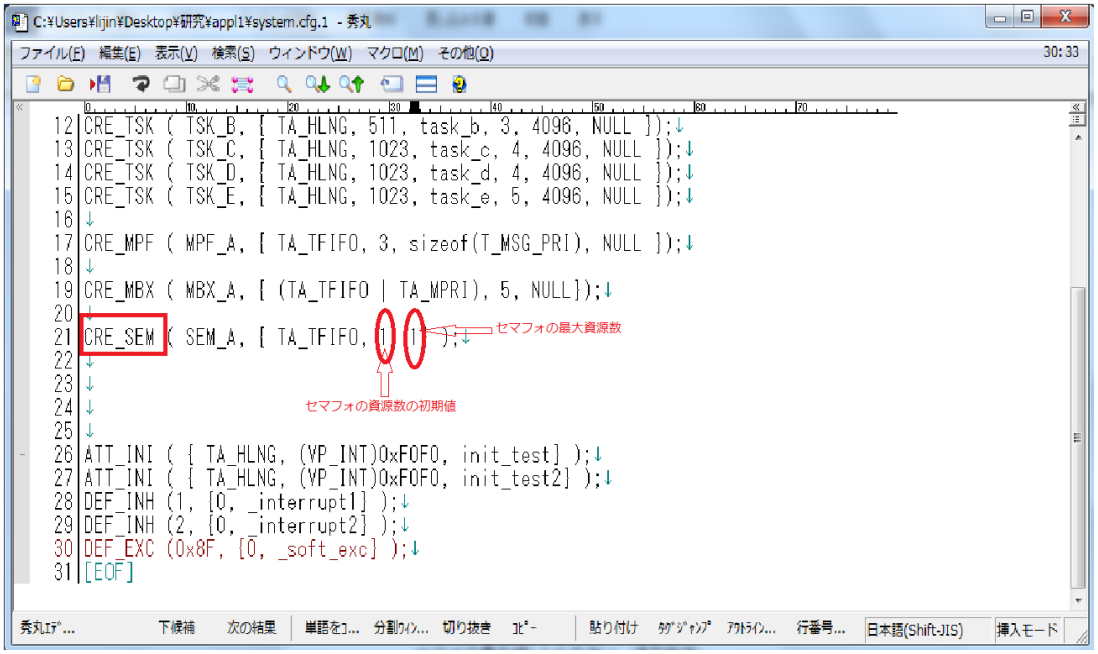


図 2.14: CRE_SEM の文字列が含まれている行におけるセマフォのパラメータの獲得

大資源数 (maxsem) というセマフォのパラメータを保存する。続いて、ch[2] に保存されている文字列を整数値に変換し、isemcnt という変数に書き込む。 ch[3] に保存されている文字列を整数値に変換し、maxsem という変数に書き込む。

- エラーチェックが必要となる場合：
  これは E_PAR エラーチェックコードを削除できない場合である。
  isemcnt(セマフォの資源数の初期値) が maxsem(セマフォの最大資源数) より大きい場合。また、maxsem(セマフォの最大資源数) が 0 に等しい、あるいは TMAX_MAXSEM(セマフォの最大資源数の最大値) より大きい場合。すなわち、以下の条件を満たす時である。

  $isemcnt > maxsem \ \ || \\ ((maxsem == 0) \ \ || \ \ (maxsem > TMAX\_MAXSEM))$

- エラーチェックが不要となる場合：
  これは E_PAR エラーチェックコードを削除できる場合である。

  $isemcnt <= maxsem \ \ \&\& \\ ((maxsem\ != 0) \ \ \&\& \ \ (maxsem <= TMAX\_MAXSEM))$

19. セマフォ機能の E_OBJ エラーチェックの要・不要を判断する手順を示す。
まず、history [] という配列を用意する。次に、アプリケーションのコンフィグレーション

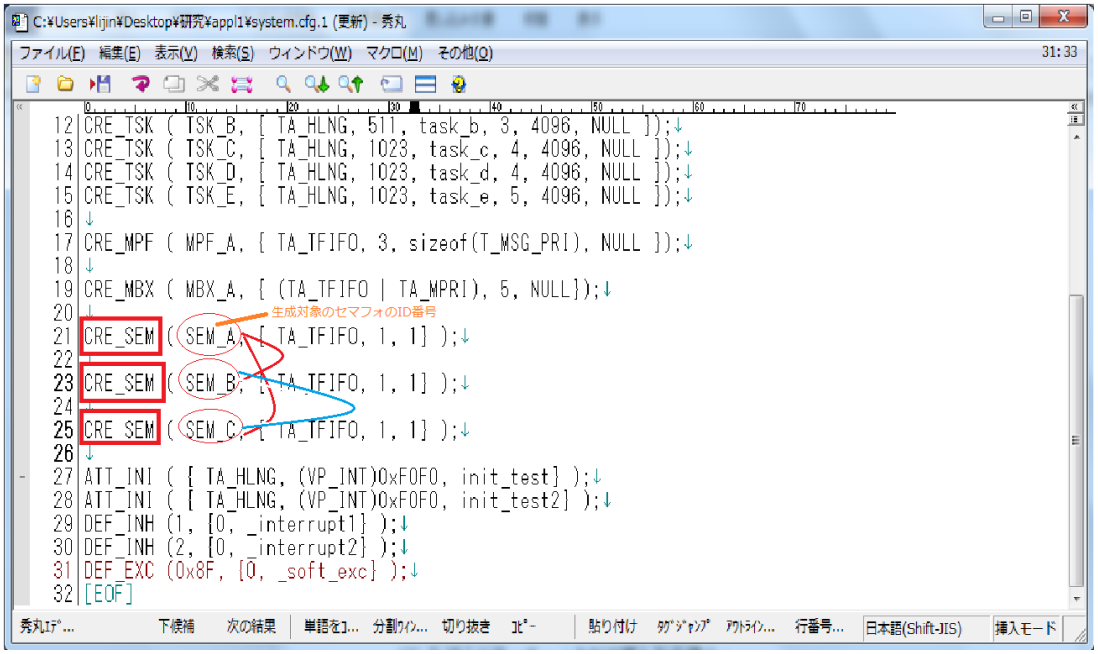


図 2.15: CRE_SEM の文字列が含まれている行におけるセマフォのオブジェクト状態の判断

ファイル（system.cfg）を解析し、この中の“CRE_SEM”という文字列が含まれている行を探す。その行から生成対象のセマフォの ID 番号を表す文字列を獲得し (図 2.15 の SEM_A)、history [] に保存する (history[0] = “SEM_A”)。次に、“CRE_SEM”の文字列が含まれる次の行を探して、その行から生成対象のセマフォの ID 番号を表す文字列 (SEM_B) を獲得して history [] に保存する (history[1] = “SEM_B”)。順次、次の“CRE_SEM”の文字列が含まれる行を探して、その行から生成対象のセマフォの ID 番号を表す文字列 (SEM_C) を獲得し、history [] に保存する (history[2] = “SEM_C”)。history [] に保存された文字列を比較して、同じ文字があるかどうかを調べる。

- エラーチェックが必要となる場合：
  これは E_OBJ エラーチェックコードを削除できない場合である。
  上述する方法で、history [] に保存された文字列を比較し、同じ文字列がある場合である。すなわち、以下の条件を満たす時である。

  $$strcmp(history[0], history[1]) == 0 \;||$$
  $$strcmp(history[0], history[2]) == 0 \;||$$
  $$strcmp(history[1], history[2]) == 0$$

- エラーチェックが不要となる場合：

これは E_OBJ エラーチェックコードを削除できる場合である。

$strcmp(history[0], history[1]) != 0$ &&
$strcmp(history[0], history[2]) != 0$ &&
$strcmp(history[1], history[2]) != 0$

### 2.2.2 ヘッダーファイル 1 の作成

各エラーチェックの要・不要を判断した結果を define.h というファイルに書き込む。
タスク管理機能の E_ID エラーを例として、内容は以下の通りである。

タスク管理機能の E_ID エラーチェックコードを削除できない場合：

#define TSK_E_ID_CHECK 1

タスク管理機能の E_ID エラーチェックコードを削除できる場合：

#define TSK_E_ID_CHECK 0

上記と同様に、タスク管理機能の他のエラーチェックコードおよび他の対象システムコール（メールボックス機能、固定長メモリプール機能、セマフォ機能）内の各エラーチェックコードの要・不要を判断した結果を define.h というファイルに書き込む。
例として、define.h に書き込む内容は：
#define TSK_E_ID_CHECK 1
#define TSK_E_NOEXS_CHECK 0
#define TSK_E_RSATR_CHECK 0
#define TSK_E_PAR_CHECK 0
#define MPF_E_ID_CHECK 1
#define MPF_E_NOEXS 1
#define MPF_E_RSATR_CHECK 1
#define CRE_MPF_E_PAR_CHECK 1
#define MPF_E_OBJ_CHECK 1
#define MBX_E_ID_CHECK 0
#define MBX_E_RSATR_CHECK 0
#define CRE_MBX_E_PAR_CHECK 1
#define MBX_E_OBJ_CHECK 1
#define MBX_E_NOEXS 1
#define SEM_E_ID_CHECK 1

#define SEM_E_NOEXS　1
#define SEM_E_RSATR_CHECK　0
#define CRE_SEM_E_PAR_CHECK　0
#define SEM_E_OBJ_CHECK　1

### 2.2.3　エラーチェックコードの除去

ヘッダーファイルに書き込まれた値により、システムコール内のエラーチェックコードを除去することができる。
タスク管理機能のタスクの生成 (cre_tsk()) というシステムコール内の E_ID エラーチェックコードの削除を例として、エラーチェックコードを削除する手順を示す。

元々のタスクの E_ID エラーチェックコードは以下の通りである。

if ($tskid < 1$ || $tskid >= TMAX\_TSKID$) { /* タスク ID の範囲 */
return E_ID ;
}

E_ID エラーチェックコードの前と後ろに以下のようなマクロ記述を追加する。

# if TSK_E_ID_CHECK == 1

if ($tskid < 1$ || $tskid >= TMAX\_TSKID$) { /* タスク ID の範囲 */
return E_ID ;
}

#endif

TSK_E_ID_CHECK == 1 の場合、E_ID エラーチェックコードを実行する。
TSK_E_ID_CHECK == 0 の場合、E_ID エラーチェックコードを実行しない。

上記と同様に、タスク管理機能の他のエラーチェックコードおよび他の対象システムコール（メールボックス機能、固定長メモリプール機能、セマフォ機能）内の各エラーチェックコードをヘッダファイルの情報に従って削除できる。

以上の手順から、アプリケーションに従って不必要なエラーチェックコードを削除することにより、実行オーバヘッドを低減することと、バイナリコードのサイズを削減するこ

とが可能である。

### 2.2.4 各システムコール内のロックコードの要・不要の判断

cre_tsk( ) システムコール内のロックコードの要・不要を判断する手順を示す。
アプリケーションのコンフィグレーションファイル (system.cfg) 内で記述された CRE_TSK() に対応する“ cre_tsk（）”はシステム初期化時に OS のタスクが自動的に実行する。
OS のタスクは最高優先度で実行され、実行中に割り込みが発生しないため、これらの cre_tsk() 内のロックコードは削除しても問題はない。しかし、アプリケーションのソースファイル (appl.c) 内で別途“ cre_tsk（）”を実行している場合は、cre_tsk() 実行中に割り込みが発生する可能性があるため、ロックコードは必要である。
以上から、appl.c 内で cre_tsk() が呼ばれている場合はロックコードが必要、呼ばれていない場合はロックコードが不必要であると判断できる。
アプリケーションのソースファイル (appl.c) を解析し、“ cre_tsk ”の文字列を探す。ある場合、count という変数に 1 を加える。

- ロックコードが必要となる場合：
  これはロックコードを削除できない場合である。
  アプリケーションのソースファイル (appl.c) 内で“ cre_tsk ”が呼ばれている場合。すなわち、以下の条件を満たす時である。
  $count > 0$
- ロックコードが不要となる場合：
  これはロックコードを削除できる場合である。
  $count <= 0$

act_tsk( ) システムコール内のロックコードの要・不要を判断する手順を示す。
まず、ch[] という配列を用意する。アプリケーションのコンフィグレーションファイル (system.cfg) を解析し、この中の“ CRE_TSK ”という文字列が含まれている行を探す。その行からタスクの関数の名前を表す文字列を獲得し（図 2.16 の task_a）、ch[] に保存する (ch[3]=“ task_a ”)。次に、アプリケーションのソースファイル (appl.c) を解析し、この中に、保存されたタスクの関数の名前に一致する関数を探す（図 2.17 の task_a の関数）。このタスク関数の中に“ act_tsk ”という文字列があるかどうかを探す。ある場合、count1 という変数に 1 を加える。同様の方法でループして、アプリケーションのコンフィグレーションファイル (system.cfg) を解析し、“ CRE_TSK ”という文字列が含まれている次の行を探して、その行からタスクの関数の名前を表す文字列を獲得し（図 2.16 の task_b、task_c、task_d）、ch[] に保存する。続いて、アプリケーションのソースファイル (appl.c) を解析し、この中に、保存されたタスクの関数の名前に一致する関数を探す（task_b、task_c、task_d

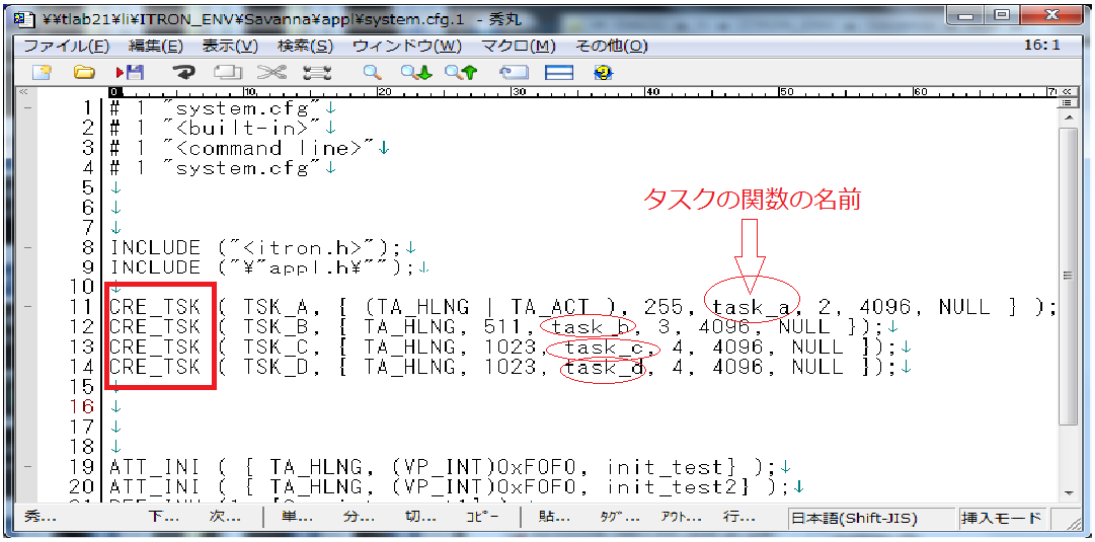


図 2.16: CRE_TSK の文字列が含まれている行におけるタスクの関数の名前を表す文字列の獲得

に一致する関数)。タスク関数の中に“ act_tsk ”という文字列があるかどうかを探す。ある場合、count1 という変数に 1 を加える。

- ロックコードが必要となる場合：
  これはロックコードを削除できない場合である。
  “ act_tsk ”が二つ以上のタスク関数の中で実行されている場合。すなわち、以下の条件を満たす時である。
  $count1 \quad >= \quad 2$

- ロックコードが不要となる場合：
  これはロックコードを削除できる場合である。
  $count1 \quad < \quad 2$

cre_mpf( ) システムコール内のロックコードの要・不要を判断する手順を示す。
アプリケーションのコンフィグレーションファイル (system.cfg) 内で記述された CRE_MPF() に対応する“ cre_mpf（）”はシステム初期化時に OS のタスクが自動的に実行する。
OS のタスクは最高優先度で実行され、実行中に割り込みが発生しないため、これらの cre_mpf() 内のロックコードは削除しても問題はない。しかし、アプリケーションのソースファイル (appl.c) 内で別途“ cre_mpf（）”を実行している場合は、cre_mpf() 実行中に割り込みが発生する可能性があるため、ロックコードは必要である。
以上から、appl.c 内で cre_mpf() が呼ばれている場合はロックコードが必要、呼ばれていない場合はロックコードが不必要であると判断できる。

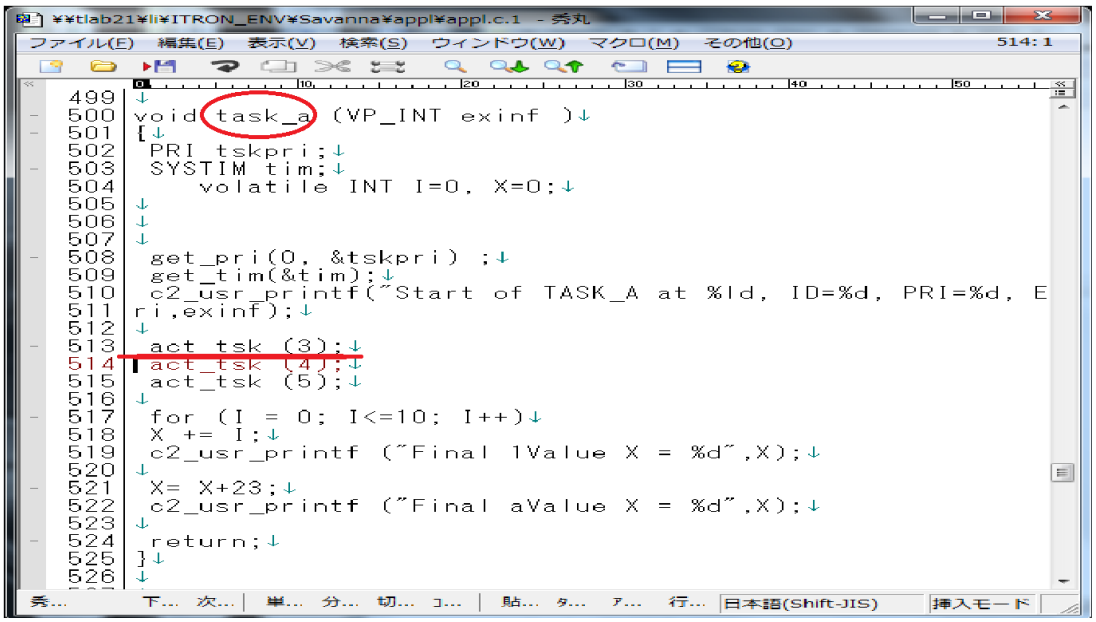


図 2.17: task_a のタスク関数の中に“ act_tsk ”の探す

アプリケーションのソースファイル (appl.c) を解析し、“ cre_mpf ”の文字列を探す。ある場合、count という変数に 1 を加える。

- ロックコードが必要となる場合：
  これはロックコードを削除できない場合である。
  アプリケーションのソースファイル (appl.c) 内で“ cre_mpf ”が呼ばれている場合。
  すなわち、以下の条件を満たす時である。
  $count > 0$

- ロックコードが不要となる場合：
  これはロックコードを削除できる場合である。
  $count <= 0$

cre_mbx( ) システムコール内のロックコードの要・不要を判断する手順を示す。
アプリケーションのコンフィグレーションファイル (system.cfg) 内で記述された CRE_MBX() に対応する“ cre_mbx（）”はシステム初期化時に OS のタスクが自動的に実行する。
OS のタスクは最高優先度で実行され、実行中に割り込みが発生しないため、これらの cre_mbx() 内のロックコードは削除しても問題はない。しかし、アプリケーションのソースファイル (appl.c) 内で別途“ cre_mbx（）”を実行している場合は、cre_mbx() 実行中に割り込みが発生する可能性があるため、ロックコードは必要である。
以上から、appl.c 内で cre_mbx() が呼ばれている場合はロックコードが必要、呼ばれてい

ない場合はロックコードが不必要であると判断できる。
アプリケーションのソースファイル (appl.c) を解析し、“ cre_mbx ”の文字列を探す。ある場合、count という変数に 1 を加える。

- ロックコードが必要となる場合：
  これはロックコードを削除できない場合である。
  アプリケーションのソースファイル (appl.c) 内で“ cre_mbx ”が呼ばれている場合。すなわち、以下の条件を満たす時である。
  $count > 0$
- ロックコードが不要となる場合：
  これはロックコードを削除できる場合である。
  $count <= 0$

cre_sem( ) システムコール内のロックコードの要・不要を判断する手順を示す。
アプリケーションのコンフィグレーションファイル (system.cfg) 内で記述された CRE_SEM() に対応する“ cre_sem（）”はシステム初期化時に OS のタスクが自動的に実行する。
OS のタスクは最高優先度で実行され、実行中に割り込みが発生しないため、これらの cre_sem() 内のロックコードは削除しても問題はない。しかし、アプリケーションのソースファイル (appl.c) 内で別途“ cre_sem（）”を実行している場合は、cre_sem() 実行中に割り込みが発生する可能性があるため、ロックコードは必要である。
以上から、appl.c 内で cre_sem() が呼ばれている場合はロックコードが必要、呼ばれていない場合はロックコードが不必要であると判断できる。
アプリケーションのソースファイル (appl.c) を解析し、“ cre_sem ”の文字列を探す。ある場合、count という変数に 1 を加える。

- ロックコードが必要となる場合：
  これはロックコードを削除できない場合である。
  アプリケーションのソースファイル (appl.c) 内で“ cre_sem ”が呼ばれている場合。すなわち、以下の条件を満たす時である。
  $count > 0$
- ロックコードが不要となる場合：
  これはロックコードを削除できる場合である。
  $count <= 0$

snd_mbx( ) システムコール内のロックコードの要・不要を判断する手順を示す。
主なコードを以下に示す。

```
  for (i=0;i<mbx_count;i++) {
  mbx_sender[i]=-1;
//初期値の設定、“snd_mbx”と mbx_names[i] に保存したメールボックスのペアの文字列はタスクに出現しないと意味する
     for (j=0;j <task_count;j++)
     if (send_count[j][i] > 0) {
//j 番号のタスクの中に“snd_mbx”と mbx_names[i] に i 番号のメールボックスのペアの文字列が出現する場合
        if (mbx_sender[i]==-1)
//“snd_mbx”と mbx_names[i] に保存した i 番号のメールボックスのペアの文字列は他のタスクに出現しなかった場合
           mbx_sender[i]=j; // このペアが所属のタスク番号を mbx_sender[i] に保存する
        else {

           snd_mbx_lock_count ++;
// snd_mbx_lock_count の初期値は 0、0 ではない場合、ロックコードが必要と意味する
              mbx_sender[i]=-2;
//“snd_mbx”と mbx_names[i] に i 番号のメールボックスのペアの文字列は二つ以上のタスクに出現すると意味する          }
        }
}
for (i=0;i <mbx_count;i++)
     if (mbx_sender[i] > =0) {
//“snd_mbx”と mbx_names[i] に保存した i 番号のメールボックスのペアの文字列はタスクに出現する場合
          for (j=0;j<task_count;j++)
           if ( (j!=mbx_sender[i]) && (recv_count[j][i] > 0) )
             if (priorities[mbx_sender[i]] >priorities[j])
//“rcv_mbx”と i 番号のメールボックスのペアが所属のタスクの優先度と“snd_mbx”と同じ番号のメールボックスのペアが所属のタスクの優先度を比較する
              snd_mbx_lock_count ++; //   ロックが必要
          }
```

まず、tasks[i][j](i はタスク番号、 j は CRE_TSK のパラメータ番号)、mbx_names[i](i はメールボックス番号)、priorities[i](i はタスク番号)、mbx_sender[i](i はタスク番号) という配列と task_count、mbx_count という変数を用意する。アプリケーションのコンフィグレーションファイル (system.cfg) を解析し、この中の“CRE_TSK”という文字列が含まれている行を探す。その行からタスクの関数の名前と優先度を表す文字列を獲得し、tasks[i][3]、tasks[i][4] にそれぞれ保存する。task_count をインクリメントしてタスク数を保存する。次

に、“ CRE_MBX ”という文字列が含まれている行を探す。その行からメールボックスの名前を表す文字列を獲得して mbx_names[i] に保存する。mbx_count をインクリメントしてメールボックス数を保存する。続いて、tasks[i][j] に保存されているタスク優先度を表す文字列を整数値に変換し、priorities[i] の配列に書き込む。次に、send_count[i][j](i はタスク番号、 j はメールボックス番号) と recv_count[i][j](i はタスク番号、 j はメールボックス番号) という配列を用意する。アプリケーションのソースファイル (appl.c) を解析し、この中に、保存されたタスクの関数の名前に一致する関数を探す。このタスク関数の中に“snd_mbx”と mbx_names[i] に保存したメールボックスのペアの文字列を探す。出現する回数を send_count[i][j] に保存する。続いて、“rcv_mbx”と mbx_names[i] に保存したメールボックスペアの文字列を探し、出現する回数を recv_count[i][j] に保存する。次に、send_count[i][j] の中に保存する値および mbx_sender[i] の値によって、snd_mbx ”と mbx_names[i] に保存したメールボックスのペアの文字列が二つ以上のタスク関数に出現するかを判断する。まず、mbx_sender[i] の初期値を設定する（mbx_sender[i]=-1）。send_count[i][j] の値が 0 より大きい場合、mbx_sender[i] の値が-1 の場合、このペアはタスクに初めて出現することを表す。このペアが含まれるのタスク番号を mbx_sender[i] に保存する。-1 ではない場合、このペアは他のタスクに出現していたことを表す。この場合、mbx_sender[i] に-2 の値を入れる。最後に、“snd_mbx” と mbx_names[i] のメールボックスのペアを含むタスクの優先度と“ rcv_mbx ”と同じメールボックスのペアが含まれるタスクの優先度を比較する。

- ロックコードが必要となる場合：
  これはロックコードを削除できない場合である。
  “ snd_mbx ”と mbx_names[i] に保存したメールボックスのペアの文字列が二つ以上のタスク関数に出現する。もしくは、“ rcv_mbx ”とメールボックスのペアが含まれるタスクの優先度が“ snd_mbx ”と同じ番号のメールボックスのペアが含まれるタスクの優先度より高い。すなわち、以下の条件を満たす時である。
  $mbx\_sender[i] = -2$ ||
  $(priorities[mbx\_sender[i]] > priorities[j])$

- ロックコードが不要となる場合：
  これはロックコードを削除できる場合である。
  $mbx\_sender[i] != -2$ &&
  $(priorities[mbx\_sender[i]] < priorities[j])$

rcv_mbx( ) システムコール内のロックコードの要・不要を判断する手順を示す。
まず、tasks[i][j](i はタスク番号、 j は CRE_TSK のパラメータ番号)、mbx_names[i](i はメールボックス番号)、priorities[i](i はタスク番号)、mbx_recv[i](i はタスク番号) という配列と task_count、mbx_count という変数を用意する。アプリケーションのコンフィグレーションファイル (system.cfg) を解析し、この中の“ CRE_TSK ”という文字列が含まれている行を探す。その行からタスクの関数の名前と優先度を表す文字列を獲得し、tasks[i][3]、

tasks[i][4] にそれぞれ保存する。task_count をインクリメントしてタスク数を保存する。次に、“ CRE_MBX ”という文字列が含まれている行を探す。その行からメールボックスの名前を表す文字列を獲得して mbx_names[i] に保存する。mbx_count をインクリメントしてメールボックス数を保存する。続いて、tasks[i][j] に保存されているタスク優先度を表す文字列を整数値に変換し、priorities[i] の配列に書き込む。次に、send_count[i][j](i はタスク番号、 j はメールボックス番号) と recv_count[i][j](i はタスク番号、 j はメールボックス番号) という配列を用意する。アプリケーションのソースファイル (appl.c) を解析し、この中に、保存されたタスクの関数の名前に一致する関数を探す。このタスク関数の中に“snd_mbx”と mbx_names[i] に保存したメールボックスのペアの文字列を探す。出現する回数を send_count[i][j] に保存する。続いて、“rcv_mbx”と mbx_names[i] に保存したメールボックスペアの文字列を探し、出現する回数を recv_count[i][j] に保存する。次に、recv_count[i][j] の中に保存する値および mbx_recv[i] の値によって、rcv_mbx ”と mbx_names[i] に保存したメールボックスのペアの文字列が二つ以上のタスク関数に出現するかを判断する。まず、mbx_recv[i] の初期値を設定する（mbx_recv[i]=-1)。recv_count[i][j] の値が 0 より大きい場合、mbx_recv[i] の値が-1 の場合、このペアはタスクに初めて出現することを表す。このペアが含まれるのタスク番号を mbx_recv[i] に保存する。-1 ではない場合、このペアは他のタスクに出現していたことを表す。この場合、mbx_recv[i] に-2 の値を入れる。最後に、“rcv_mbx”と mbx_names[i] のメールボックスのペアを含むタスクの優先度と“snd_mbx”と同じメールボックスのペアが含まれるタスクの優先度を比較する。

- ロックコードが必要となる場合：
  これはロックコードを削除できない場合である。
  “ rcv_mbx ”と mbx_names[i] に保存したメールボックスのペアの文字列が二つ以上のタスク関数に出現する。もしくは、“ snd_mbx ”とメールボックスのペアが含まれるタスクの優先度が“ rcv_mbx ”と同じ番号のメールボックスのペアが含まれるタスクの優先度より高い。すなわち、以下の条件を満たす時である。
  $mbx\_recv[i] = -2$ ||
  $(priorities[mbx\_recv[i]] > priorities[j])$
- ロックコードが不要となる場合：
  これはロックコードを削除できる場合である。
  $mbx\_recv[i]! = -2$ &&
  $(priorities[mbx\_recv[i]] < priorities[j])$

### 2.2.5 ヘッダーファイル 2 の作成

各システムコール内のロックコードの要・不要を判断した結果を define2.h というファイルに書き込む。

タスク管理機能のタスクの生成 (cre_tsk()) のシステムコールを例として、内容は以下の通りである。

cre_tsk システムコールのロックコードを削除できない場合：

#define CRE_TSK_LOCK　1

cre_tsk のシステムコールのロックコードを削除できる場合：

#define CRE_TSK_LOCK　0

上記と同様に、タスク管理機能の他のシステムコールおよび他の対象システムコール（メールボックス機能、固定長メモリプール機能、セマフォ機能のためのシステムコール）内のロックコードの要・不要を判断した結果を define2.h というファイルに書き込む。
例として、define2.h に書き込む内容は：
#define CRE_TSK_LOCK　0
#define ACT_TSK_LOCK　0
#define CRE_MPF_LOCK　1
#define REL_MPF_LOCK　1
#define GET_MPF_LOCK　1
#define CRE_MBX_LOCK　1
#define SND_MBX_LOCK　0
#define RCV_MBX_LOCK　0
#define CRE_SEM_LOCK　0
#define SIG_SEM_LOCK　0
#define WAI_SEM_LOCK　1

### 2.2.6　ロックコードの除去

ヘッダーファイルに書き込まれた値により、システムコール内のロックコードを除去することができる。
タスク管理機能のタスクの生成 (cre_tsk()) のシステムコール内のロックコードの削除を例として、ロックコードを削除する手順を示す。

元々のタスクのロックコードは以下の通りである。
_kernel_loc_cpu ();
または

_kernel_unl_cpu ();

ロックコードの前と後ろに以下のようなマクロ記述を追加する。

```
#if CRE_TSK_LOCK == 1

  _kernel_loc_cpu ();

#endif

#if CRE_TSK_LOCK == 1

  _kernel_unl_cpu ();

#endif
```

CRE_TSK_LOCK == 1 の場合、タスクの生成 (cre_tsk()) のシステムコール内のロックコードを実行する。
CRE_TSK_LOCK == 0 の場合、タスクの生成 (cre_tsk()) のシステムコール内のロックコードを実行しない。

上記と同様に、タスク管理機能の他のシステムコールおよび他の対象システムコール（メールボックス機能、固定長メモリプール機能、セマフォ機能ためのシステムコール）内のロックコードをヘッダファイル 2 の情報に従って削除できる。

以上の手順から、アプリケーションに従って不必要なロックコードを削除することにより、実行オーバヘッドを低減することと、バイナリコードのサイズを削減することが可能である。

# 第3章　評価

## 3.1　評価環境

本章では提案した方式を実装する。実装したシステムに作成したタスクセットを適用し、アプリケーションの実行性能 (バイナリコードのサイズおよび実行時間と実行命令数) を評価する。

### 3.1.1　CPU シミュレータ

本研究では、クロックサイクルベースの CPU シミュレータを使用して提案手法を評価するために、SPARC v8 命令セットアーキテクチャ[2] の CPU シミュレータを使用する。シミュレータに入力されたバイナリコードはタスクのコードと OS コードである。従って、この評価では正確な定量的評価を可能とする。
シミュレーションを用いたマシンの仕様は以下の通りである。
プロセッサ　: Intel(R) Xeon(R) CPU E3-1280 V2 @ 3.60GHz
キャッシュ: 8MB
メモリ:32GB
OS: CentOS 64 ビット
リリース:6.3

## 3.2　実験結果

### 3.2.1　エラーチェックコードの削除によるバイナリコードのサイズの減少

対象システムコール (タスク管理機能、固定長メモリプール機能、メールボックス機能、セマフォ機能) 内の各エラーチェックコードをヘッダファイルの情報に従って、削除できない場合と削除できる場合を比べた際、削除できる場合のバイナリコードのサイズが減っていることを以下に示す。

表 3.1: エラーチェックコードの削除によるタスク管理機能のバイナリコードのサイズの減小結果

| | 値 | cre_tsk | act_tsk |
|---|---|---|---|
| すべてのエラーチェック必要 | 1 | 848 | 472 |
| TSK_E_ID_CHECK | 0 | 840 | 456 |
| TSK_E_NOEXS_CHECK | 0 | | 432 |
| TSK_E_RSATR_CHECK | 0 | 844 | |
| TSK_E_PAR_CHECK | 0 | 780 | |
| すべてのエラーチェック不要 | 0 | 752 | 412 |

- タスク管理機能の cre_tsk
  エラーチェックコードが必要な場合：
  バイナリコードのサイズは 848 である。
  エラーチェックが不要となる場合：
  TSK_E_ID_CHECK が 0 になる場合はバイナリコードのサイズは 840(0.94%削除) になる。
  TSK_E_RSATR_CHECK が 0 になる場合はバイナリコードのサイズは 844(0.47%削除) になる。
  TSK_E_PAR_CHECK が 0 になる場合はバイナリコードのサイズは 780(8.01%削除) になる。
  TSK_E_ID_CHECK、TSK_E_RSATR_CHECK、 TSK_E_PAR_CHECK が 0 になる場合はバイナリコードのサイズは 752(11.32%削除) になる。 (表 3.1 に示す)

- タスク管理機能の act_tsk
  エラーチェックコードが必要な場合：
  バイナリコードのサイズは 472 である。
  エラーチェックコードが不要となる場合：
  TSK_E_ID_CHECK が 0 になる場合はバイナリコードのサイズは 456(3.39%削除) になる。
  TSK_E_NOEXS_CHECK が 0 になる場合はバイナリコードのサイズは 432(8.47%削除) になる。
  TSK_E_ID_CHECK、 TSK_E_NOEXS_CHECK、TSK_E_RSATR_CHECK が 0 になる場合はバイナリコードのサイズは 412(12.71%削除) になる。(表 3.1 に示す)

- 固定長メモリプール機能の cre_mpf

エラーチェックコードが必要な場合：
バイナリコードのサイズは 424 である。
エラーチェックコードが不要となる場合：
MPF_E_ID_CHECK が 0 になる場合はバイナリコードのサイズは 412(2.83%削除) になる。
MPF_E_RSATR_CHEC が 0 になる場合はバイナリコードのサイズは 408(3.77%削除) になる。
CRE_MPF_E_PAR_CHECK が 0 になる場合はバイナリコードのサイズは 380(10.38%削除) になる。
MPF_E_OBJ_CHECK が 0 になる場合はバイナリコードのサイズは 392(7.20%削除) になる。
MPF_E_ID_CHECK、 MPF_E_RSATR_CHECK、 CRE_MPF_E_PAR_CHECK、 MPF_E_OBJ_CHECK が 0 になる場合はバイナリコードのサイズは 316(25.47%削除) になる。(表 3.2 に示す）

- 固定長メモリプール機能の rel_mpf
  エラーチェックコードが必要な場合：
  バイナリコードのサイズは 444 である。
  エラーチェックコードが不要となる場合：
  MPF_E_ID_CHECK が 0 になる場合はバイナリコードのサイズは 432(2.70%削除) になる。
  MPF_E_NOEXS_CHECK が 0 になる場合はバイナリコードのサイズは 460(9.91%削除) になる。
  MPF_E_ID_CHECK、 MPF_E_NOEXS_CHECK が 0 になる場合はバイナリコードのサイズは 384(13.51%削除) になる。(表 3.2 に示す）

- 固定長メモリプール機能の get_mpf
  エラーチェックコードが必要な場合：
  バイナリコードのサイズは 416 である。
  エラーチェックコードが不要となる場合：
  MPF_E_ID_CHECK が 0 になる場合はバイナリコードのサイズは 404(2.88%削除) になる。
  MPF_E_NOEXS_CHECK が 0 になる場合はバイナリコードのサイズは 380(8.65%削除) になる。
  MPF_E_ID_CHECK、MPF_E_NOEXS_CHECK が 0 になる場合はバイナリコードのサイズは 368(11.54%削除) になる。(表 3.2 に示す）

表 3.2: エラーチェックコードの削除による固定長メモリプール機能のバイナリコードのサイズの減小結果

| | 値 | cre_mpf | rel_mpf | get_mpf |
|---|---|---|---|---|
| すべてのエラーチェック必要 | 1 | 424 | 444 | 416 |
| MPF_E_ID_CHECK | 0 | 412 | 432 | 404 |
| MPF_E_NOEXS_CHECK | 0 | | 400 | 380 |
| MPF_E_RSATR_CHECK | 0 | 408 | | |
| CRE_MPF_E_PAR_CHECK | 0 | 380 | | |
| MPF_E_OBJ_CHECK | 0 | 392 | | |
| すべてのエラーチェック不要 | 0 | 316 | 384 | 368 |

- メールボックス機能の cre_mbx
  エラーチェックコードが必要な場合：
  バイナリコードのサイズは 484 である。
  エラーチェックコードが不要となる場合：
  MBX_E_ID_CHECK が 0 になる場合はバイナリコードのサイズは 472(2.48%削除) になる。
  MBX_E_RSATR_CHECK が 0 になる場合はバイナリコードのサイズは 472(2.48%削除) になる。
  CRE_MBX_E_PAR_CHECK が 0 になる場合はバイナリコードのサイズは 420(13.22%削除) になる。
  MBX_E_OBJ_CHECK が 0 になる場合はバイナリコードのサイズは 448(7.43%削除) になる。
  MBX_E_ID_CHECK、 MBX_E_RSATR_CHECK、CRE_MBX_E_PAR_CHECK、MBX_E_OBJ_CHECK が 0 になる場合はバイナリコードのサイズは 352(27.27%削除) になる。(表 3.3 に示す）

- メールボックス機能の snd_mbx
  エラーチェックコードが必要な場合：
  バイナリコードのサイズは 560 である。
  エラーチェックコードが不要となる場合：
  MBX_E_ID_CHECK が 0 になる場合はバイナリコードのサイズは 548(2.14%削除) になる。
  MBX_E_NOEXS_CHECK が 0 になる場合はバイナリコードのサイズは 524(6.43%削除) になる。
  MBX_E_ID_CHECK、 MBX_E_NOEXS_CHECK が 0 になる場合はバイナリコード

表 3.3: エラーチェックコードの削除によるメールボックス機能のバイナリコードのサイズの減小結果

| | 値 | cre_mbx | snd_mbx | rcv_mbx |
|---|---|---|---|---|
| すべてのエラーチェック必要 | 1 | 484 | 560 | 456 |
| MBX_E_ID_CHECK | 0 | 472 | 548 | 444 |
| MBX_E_RSATR_CHECK | 0 | 472 | | |
| CRE_MBX_E_PAR_CHECK | 0 | 420 | | |
| MPF_E_OBJ_CHECK | 0 | 448 | | |
| MBX_E_NOEXS_CHECK | 0 | | 524 | 424 |
| すべてのエラーチェック不要 | 0 | 352 | 508 | 412 |

のサイズは 508(9.29%削除) になる。(表 3.3 に示す)

- メールボックス機能の rcv_mbx
  エラーチェックコードが必要な場合：
  バイナリコードのサイズは 456 である。
  エラーチェックコードが不要となる場合：
  MBX_E_ID_CHECK が 0 になる場合はバイナリコードのサイズは 444(2.63%削除) になる。
  MBX_E_NOEXS_CHECK が 0 になる場合はバイナリコードのサイズは 424(6.87%削除) になる。
  MBX_E_ID_CHECK、 MBX_E_NOEXS_CHECK が 0 になる場合はバイナリコードのサイズは 412(9.64%削除) になる。(表 3.3 に示す)

- セマフォ機能の cre_sem
  エラーチェックコードが必要な場合：
  バイナリコードのサイズは 328 である。
  エラーチェックコードが不要となる場合：
  SEM_E_ID_CHECK が 0 になる場合はバイナリコードのサイズは 316(3.65%削除) になる。
  SEM_E_RSATR_CHECK が 0 になる場合はバイナリコードのサイズは 312(4.88%削除) になる。
  CRE_SEM_E_PAR_CHECK が 0 になる場合はバイナリコードのサイズは 272(17.07%削除) になる。

SEM_E_OBJ_CHECK が 0 になる場合はバイナリコードのサイズは 288(11.83%削除) になる。
SEM_E_ID_CHECK、 SEM_E_RSATR_CHECK、 CRE_SEM_E_PAR_CHECK、 SEM_E_OBJ_CHECK が 0 になる場合はバイナリコードのサイズは 200(39.02%削除) になる。(表 3.4 に示す）

- セマフォ機能の sig_sem
  エラーチェックコードが必要な場合：
  バイナリコードのサイズは 396 である。
  エラーチェックコードが不要となる場合：
  SEM_E_ID_CHECK が 0 になる場合はバイナリコードのサイズは 388(2.02%削除) になる。
  SEM_E_NOEXS_CHECK が 0 になる場合はバイナリコードのサイズは 360(9.09%削除) になる。
  SEM_E_ID_CHECK、 SEM_E_NOEXS_CHECK が 0 になる場合はバイナリコードのサイズは 344(13.13%削除) になる。(表 3.4 に示す）

- セマフォ機能の wai_sem
  エラーチェックコードが必要な場合：
  バイナリコードのサイズは 344 である。
  エラーチェックコードが不要となる場合：
  SEM_E_ID_CHECK が 0 になる場合はバイナリコードのサイズは 332(3.49%削除) になる。
  SEM_E_NOEXS_CHECK が 0 になるはバイナリコードのサイズは 312(9.30%削除) になる。
  SEM_E_ID_CHECK、 SEM_E_NOEXS_CHECK が 0 になる場合はバイナリコードのサイズは 300(12.80%削除) になる。(表 3.4 に示す）

### 3.2.2 エラーチェックコードの削除による実行時間と実行命令数の減少

対象システムコール (タスク管理機能、固定長メモリプール機能、メールボックス機能、セマフォ機能) 内の各エラーチェックコードを削除できない場合と削除できる場合を比べた際、三つの実験によって、実行時間と実行命令数が減っていることを以下に示す。

表 3.4: エラーチェックコードの削除によるセマフォ機能のバイナリコードのサイズの減小結果

| | 値 | cre_sem | sig_sem | wai_sem |
|---|---|---|---|---|
| すべてのエラーチェック必要 | 1 | 328 | 396 | 344 |
| SEM_E_ID_CHECK | 0 | 316 | 388 | 332 |
| SEM_E_NOEXS_CHECK | 0 | | 360 | 312 |
| SEM_E_RSATR_CHECK | 0 | 312 | | |
| CRE_SEM_E_PAR_CHECK | 0 | 272 | | |
| SEM_E_OBJ_CHECK | 0 | 288 | | |
| すべてのエラーチェック不要 | 0 | 200 | 344 | 300 |

実験 1 においてタスク管理機能についてのプログラムを実行した。実行したプログラムについて、主なコードを以下に示す。

```
void task_a (VP_INT exinf )
{
    /*    省略    */
act_tsk (TSK_B);   //  task_b の起動により、act_tsk が実行される (一回目)
act_tsk (TSK_C);  // task_c の起動により、act_tsk が実行される (二回目)
act_tsk (TSK_D);  // task_d の起動により、act_tsk が実行される (三回目)
   /*    処理    */
}

void task_b (VP_INT exinf)
{
    /*    省略    */
}

void task_c (VP_INT exinf )
   /*    省略    */
}

void task_d (VP_INT exinf )
{
   /*    省略    */
}
```

```
/*システムコンフィギュレーションファイル"system.cfg"*/

 /*    省略    */

CRE_TSK ( TSK_A, { (TA_HLNG | TA_ACT ), 255, task_a, 2, 4096, NULL } );
//task_a の登録により、cre_tsk が実行される (一回目)
CRE_TSK ( TSK_B, { TA_HLNG, 511, task_b, 3, 4096, NULL });
//task_b の登録により、cre_tsk が実行される (二回目)
CRE_TSK ( TSK_C, { TA_HLNG, 1023, task_c, 4, 4096, NULL });
//task_c の登録により、cre_tsk が実行される (三回目)
CRE_TSK ( TSK_D, { TA_HLNG, 1023, task_d, 4, 4096, NULL });
//task_d の生成により、cre_tsk が実行される (四回目)
/*    省略    */
```

プログラムの説明:
タスクとして task_a 、task_b、task_c 、task_d を用意する。コンフィギュレーションファイル内で“CRE_TSK”によって各タスクを登録することにより、システム初期化時に cre_tsk システムコールが実行される。 task_a において、task_b、task_c 、task_d の起動を行う。(それぞれ、act_tsk (TSK_B)、 act_tsk (TSK_C) 、act_tsk (TSK_D)。)

このアプリケーションに対して提案手法によってエラーチェックコードの要・不要を判断した結果は以下の通りである。
define.h
#define TSK_E_ID_CHECK 0
#define TSK_E_NOEXS_CHECK 0
#define TSK_E_RSATR_CHECK 0
#define TSK_E_PAR_CHECK 0
#define MPF_E_ID_CHECK 1
#define MPF_E_NOEXS 1
・
・
・
・

タスク管理機能について、エラーチェックコードが必要な場合 (FULL) の実行時間と実行命令数を以下に示す。

- cre_tsk

エラーチェックコードが必要な場合 (FULL)：
一回目:( task_a の登録のための cre_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
開始時刻は 1,944 であり、完了時刻は 2,882 である。よって実行時間は 938 である。
開始命令は 1,014 であり、完了命令は 1,400 である。よって実行命令数は 386 である。
二回目:( task_b の登録のための cre_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
開始時刻は 2,919 であり、完了時刻は 3,187 である。よって実行時間は 268 である。
開始命令は 1,417 であり、完了命令は 1,623 である。よって実行命令数は 206 である。
三回目:( task_c の登録のための cre_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
開始時刻は 3,224 であり、完了時刻は 3,502 である。よって実行時間は 278 である。
開始命令は 1,640 であり、完了命令は 1,846 である。よって実行命令数は 206 である。
四回目:( task_d の登録のための cre_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
開始時刻は 3,537 であり、。完了時刻は 3,805 である。よって実行時間は 268 である。
開始命令は 1,861 であり、完了命令は 2,067 である。よって実行命令数は 206 である。

一回目から四回目まで実行した四回分の平均実行時間は 438 である。
一回目から四回目まで実行した四回分の平均実行命令数は 251 である。(表 3.5 に示す)

- act_tsk
  エラーチェックコードが必要場合 (FULL)：
  一回目:(task_b の起動のための act_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
  開始時刻は 6,811 であり、完了時刻は 7,291 である。よって実行時間は 480 である。
  開始命令は 3,898 であり、完了命令は 4,206 である。よって実行命令数は 308 である。
  二回目:(task_c の起動のための act_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
  開始時刻は 7,294 であり、完了時刻は 7,612 である。よって実行時間は 318 である。
  開始命令は 4,209 であり、完了命令は 4,525 である。よって実行命令数は 316 である。
  三回目:(task_d の起動のための act_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
  開始時刻は 7,615 であり、完了時刻は 7,951 である。よって実行時間は 336 である。
  開始命令は 4,528 であり、完了命令は 4,852 である。よって実行命令数は 324 である。

一回目から三回目まで実行した三回分の平均実行時間は 378 である。
一回目から三回目まで実行した三回分の平均実行命令数は 316 である。(表 3.5 に示す)

タスク管理機能について、エラーチェックコードが不要となる場合 (OPT) の実行時間と実行命令数を以下に示す。

- cre_tsk
  エラーチェックが不要となる場合 (OPT) :
  一回目:( task_a の生成のための cre_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
  開始時刻は 1,954 であり、完了時刻は 2,840 である。よって実行時間は 886 である。
  開始命令は 1,014 であり、完了命令は 1,378 である。よって実行命令数は 364 である。
  二回目:(task_b の生成のための cre_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
  開始時刻は 2,877 であり、完了時刻は 3,123 である。よって実行時間は 246 である。
  開始命令は 1,395 であり、完了命令は 1,579 である。よって実行命令数は 184 である。
  三回目:(task_c の生成のための cre_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
  開始時刻は 3,160 であり、完了時刻は 3,416 である。よって実行時間は 256 である。
  開始命令は 1,596 であり、完了命令は 1,780 である。よって実行命令数は 184 である。
  四回目:(task_d の生成のための cre_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
  開始時刻は 3,451 であり、完了時刻は 3,697 である。よって実行時間は 246 である。
  開始命令は 1,795 であり、完了命令は 1,979 である。よって実行命令数は 184 である。

  一回目から四回目まで実行した四回分の平均実行時間は 409(6.62%削除) である。
  一回目から四回目まで実行した四回分の平均実行命令数は 229(8.76%削除) である。
  (表 3.6 に示す)

- act_tsk
  エラーチェックが不要となる場合 (OPT) :
  一回目:(task_b の起動のための act_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
  開始時刻は 6,681 であり、完了時刻は 7,140 である。よって実行時間は 459 である。
  開始命令は 3,788 であり、完了命令は 4,085 である。よって実行命令数は 297 である。
  二回目:(task_c の起動のための act_tsk の処理が始まってから、終わるまでの実行時

表 3.5: タスク機能についての実験 1FULL

| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
|---|---|---|---|---|---|---|---|
| cre_tsk | 1 回目 | 1,944 | 2,882 | 938 | 1,014 | 1,400 | 386 |
| | 2 回目 | 2,919 | 3,187 | 268 | 1,417 | 1,623 | 206 |
| | 3 回目 | 3,224 | 3,502 | 278 | 1,640 | 1,846 | 206 |
| | 4 回目 | 3,537 | 3,805 | 268 | 1,861 | 2,067 | 206 |
| | 4 回分の平均実行時間:438 | | | | 4 回分の平均実行命令数:251 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| act_tsk | 1 回目 | 6,811 | 7,291 | 480 | 3,898 | 4,206 | 308 |
| | 2 回目 | 7,294 | 7,612 | 318 | 4,209 | 4,525 | 316 |
| | 3 回目 | 7,615 | 7,951 | 336 | 4,528 | 4,852 | 324 |
| | 3 回分の平均実行時間:378 | | | | 3 回分の平均実行命令数:316 | | |

間と実行命令数の値である)。
開始時刻は 7,143 であり、完了時刻は 7,450 である。よって実行時間は 307 である。
開始命令は 4,088 であり、完了命令は 4,393 である。よって実行命令数は 305 である。
三回目:(task_d の起動のための act_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
開始時刻は 7,453 であり、完了時刻は 7,778 である。よって実行時間は 325 である。
開始命令は 4,396 であり、完了命令は 4,709 である。よって実行命令数は 313 である。

一回目から三回目まで実行した三回分の平均実行時間は 364(3.70%削除) である。
一回目から三回目まで実行した三回分の平均実行命令数は 305(3.48%削除) である。
(表 3.6 に示す)

実験 2 において固定長メモリプール機能、メールボックス機能についてのプログラムを実行した。実行したプログラムについて、主なコードを以下に示す。

```
void task_a (VP_INT exinf )
{
   /*    省略    */
   ercd = get_mpf (MPF_A, (VP *)&pk_msg); //MPF_A により、固定長メモリブロックの獲得 (一回目)
   /*  メッセージを用意、優先度を付ける  */
   snd_mbx (MBX_A,pk_msg); //MBX_A により、メールボックスへの送信 (一回目)
```

表 3.6: タスク機能についての実験 1OPT

| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
|---|---|---|---|---|---|---|---|
| cre_tsk | 1 回目 | 1,954 | 2840 | 886 | 1,014 | 1,378 | 364 |
| | 2 回目 | 2,877 | 3,123 | 246 | 1,395 | 1,579 | 184 |
| | 3 回目 | 3,160 | 3,416 | 256 | 1,596 | 1,780 | 184 |
| | 4 回目 | 3,451 | 3,697 | 246 | 1,795 | 1,979 | 184 |
| | 4 回分の平均実行時間:409 | | | | 4 回分の平均実行命令数:229 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| act_tsk | 1 回目 | 6,681 | 7,140 | 459 | 3,788 | 4,085 | 297 |
| | 2 回目 | 7,143 | 7,450 | 307 | 4,088 | 4,393 | 305 |
| | 3 回目 | 7,453 | 7,778 | 325 | 4,396 | 4,709 | 313 |
| | 3 回分の平均実行時間:364 | | | | 3 回分の平均実行命令数:305 | | |

```
   ercd = get_mpf (MPF_B, (VP *)&pk_msg); //MPF_B により、固定長メモリブロックの獲得 (二回目)
   /*  メッセージを用意、優先度を付ける  */
   snd_mbx (MBX_B, pk_msg);  //MBX_B により、メールボックスへの送信 (二回目)

   ercd  = get_mpf (MPF_C, (VP *)&pk_msg); //MPF_C により、固定長メモリブロックの獲得 (三回目)
   /*  メッセージを用意、優先度を付ける  */
   snd_mbx (MBX_C, pk_msg); //MBX_C により、メールボックスへの送信 (三回目)

void task_b (VP_INT exinf)
{
   /*    省略    */
   rcv_mbx(MBX_A, &pk_msg);  //MBX_A により、メールボックスからの受信 (一回目)
   /*  メッセージを読み出す  */
   rel_mpf ( MPF_A,(VP*)pk_msg ); //MPF_A により、固定長メモリブロックの返却 (一回目)

   rcv_mbx(MBX_B, &pk_msg);   MBX_B により、メールボックスからの受信 (二回目)
   /*  メッセージを読み出す  */
   rel_mpf ( MPF_B,(VP*)pk_msg );//MPF_B により、固定長メモリブロックの返却 (二回目)

   rcv_mbx(MBX_C, &pk_msg); // MBX_C により、メールボックスからの受信 (三回目)
   /*  メッセージを読み出す  */
```

```
   rel_mpf ( MPF_C,(VP*)pk_msg ); //MPF_C により、固定長メモリブロックの返却
(三回目)
}

/*システムコンフィギュレーションファイル"system.cfg"*/

 /*    省略    */

CRE_TSK ( TSK_A, { (TA_HLNG | TA_ACT ), 255, task_a, 2, 4096, NULL } );
CRE_TSK ( TSK_B, { TA_HLNG, 511, task_b, 3, 4096, NULL });


CRE_MPF ( MPF_A, { TA_TFIFO,  3, sizeof(T_MSG_PRI), NULL });
//MPF_A の登録により、cre_mpf が実行される (一回目)
CRE_MBX ( MBX_A, { (TA_TFIFO | TA_MPRI), 5, NULL});
//MBX_A の登録により、cre_mbx が実行される (一回目)

CRE_MPF ( MPF_B, { TA_TFIFO,  3, sizeof(T_MSG_PRI), NULL });
//MPF_B の登録により、cre_mpf が実行される (二回目)
CRE_MBX ( MBX_B, { (TA_TFIFO | TA_MPRI), 5, NULL});
//MBX_B の登録により、cre_mbx が実行される (二回目)

CRE_MPF ( MPF_C, { TA_TFIFO,  3, sizeof(T_MSG_PRI), NULL });
//MPF_C の登録により、cre_mpf が実行される (三回目)
CRE_MBX ( MBX_C, { (TA_TFIFO | TA_MPRI), 5, NULL});
//MBX_C の登録により、cre_mbx が実行される (三回目)

/*    省略    */
```

プログラムの説明:
タスクとして task_a、task_b、固定長メモリプールとして MPF_A、MPF_B、MPF_C、メールボックスとして MBX_A、MBX_B、MBX_C を用意する。コンフィギュレーションファイル内で“ CRE_MPF ”と“ CRE_MBX ”によって各固定長メモリプール、各メールボックスを登録することにより、システム初期化時に cre_mpf システムコールと cre_mbx システムコールが実行される。 task_a において、MPF_A に対して、get_mpf (固定長メモリブロックの獲得) を行い、メッセージを用意して、優先度を付ける。MBX_A を使用して、snd_mbx (メールボックスへの送信) を行う。同様に MPF_ B に対して、get_mpf を行い、メッセージを用意して、優先度を付ける。MBX_B を使用して、snd_mbx を行う。続いて、MPF_C に対して、get_mpf を行い、メッセージを用意して、優先度を付ける。MBX_C を使

用して、snd_mbx を行う。また、task_b において、MBX_A を使用して、rcv_mbx (メールボックスからの受信) を行い、メッセージを読み出す。MPF_A に対して、rel_mpf (固定長メモリブロックの返却) を行う。同様に、MBX_B を使用して、rcv_mbx を行い、メッセージを読み出す。MPF_B に対して、rel_mpf を行う。最後に、MBX_C を使用して、 rcv_mbx を行い、メッセージを読み出す。MPF_C に対して、rel_mpf を行う。

このアプリケーションに対して提案手法によってエラーチェックコードの要・不要を判断した結果は以下の通りである。
define.h

・
・
・

```
#define MPF_E_ID_CHECK　0
#define MPF_E_NOEXS　0
#define MPF_E_RSATR_CHECK　0
#define CRE_MPF_E_PAR_CHECK　0
#define MPF_E_OBJ_CHECK　0
#define MBX_E_ID_CHECK　0
#define MBX_E_RSATR_CHECK　0
#define CRE_MBX_E_PAR_CHECK　0
#define MBX_E_OBJ_CHECK　0
#define MBX_E_NOEXS　0
#define SEM_E_ID_CHECK　1
#define SEM_E_NOEXS　1
```

・
・
・
・

固定長メモリプール機能について、エラーチェックコードが必要な場合 (FULL) の実行時間と実行命令数は以下に示す。

- cre_mpf
  エラーチェックコードが必要場合 (FULL)：
  一回目:(MPF_A の登録のための cre_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 3,217 であり、完了時刻は 3,549 である。よって実行時間は 332 である。
  開始命令は 1,623 であり、完了命令は 1,803 である。よって実行命令数は 180 である。

二回目:（MPF_B の登録のための cre_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
開始時刻は 4,014 であり、完了時刻は 4,216 である。よって実行時間は 202 である。
開始命令は 2,058 であり、完了命令は 2,238 である。よって実行命令数は 180 である。
三回目:(MPF_C の登録のための cre_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
開始時刻は 4,519 であり、完了時刻は 4,721 である。よって実行時間は 202 である。
開始命令は 2,491 であり、完了命令は 2,671 である。よって実行命令数は 180 である。

一回目から三回目まで実行した三回分の平均実行時間は 245 である。
一回目から三回目まで実行した三回分の平均実行命令数は 180 である。(表 3.7 に示す)

- get_mpf
  エラーチェックコードが必要場合 (FULL)：
  一回目:(MPF_A により,get_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)
  開始時刻は 8,494 であり、完了時刻は 8,708 である。よって実行時間は 214 である。
  開始命令は 5,057 であり、完了命令は 5,169 である。よって実行命令数は 112 である。
  二回目:(MPF_B により、get_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)
  開始時刻は 8,981 であり、完了時刻は 9,095 である。よって実行時間は 114 である。
  開始命令は 5,312 であり、完了命令は 5,424 である。よって実行命令数は 112 である。
  三回目:(MPF_C により、get_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)
  開始時刻は 9,258 であり、完了時刻は 9,372 である。よって実行時間は 114 である。
  開始命令は 5,567 であり、完了命令は 5,679 である。よって実行命令数は 112 である。

  一回目から三回目まで実行した三回分の平均実行時間は 147 である。
  一回目から三回目まで実行した三回分の平均実行命令数は 112 である。(表 3.7 に示す)

- rel_mpf
  エラーチェックコードが必要場合 (FULL)：
  一回目:(MPF_A により、rel_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)
  開始時刻は 12,206 であり、完了時刻は 12,427 である。よって実行時間は 221 であ

る。
開始命令は 7,375 であり、完了命令は 7,485 である。よって実行命令数は 110 である。
二回目:(MPF_B により、rel_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)
開始時刻は 12,593 であり、完了時刻は 12,714 である。よって実行時間は 121 である。
開始命令は 7,631 であり、完了命令は 7,741 である。よって実行命令数は 110 である。
三回目:(MPF_C により、rel_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)
開始時刻は 12,877 であり、完了時刻は 12,998 である。よって実行時間は 121 である。
開始命令は 7,894 であり、完了命令は 8,004 である。よって実行命令数は 110 である。

一回目から三回目まで実行した三回分の平均実行時間は 154 である。
一回目から三回目まで実行した三回分の平均実行命令数は 110 である。(表 3.7 に示す)

固定長メモリプール機能について、エラーチェックコードが不要となる場合 (OPT) の実行時間と実行命令数を以下に示す。

- cre_mpf
  エラーチェックが不要となる場合 (OPT) :
  一回目:(MPF_A の登録のための cre_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 3,153 であり、完了時刻は 3,423 である。よって実行時間は 270 である。
  開始命令は 1,579 であり、完了命令は 1,737 である。よって実行命令数は 158 である。
  二回目:(MPF_B の登録のための cre_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 3,819 であり、完了時刻は 3,999 である。よって実行時間は 180 である。
  開始命令は 1,963 であり、完了命令は 2,121 である。よって実行命令数は 158 である。
  三回目:(MPF_C の登録のための cre_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 4,273 であり、完了時刻は 4,453 である。よって実行時間は 180 である。
  開始命令は 2,345 であり、完了命令は 2,503 である。よって実行命令数は 158 である。

  一回目から三回目まで実行した三回分の平均実行時間は 210(14.29%削除) である。
  一回目から三回目まで実行した三回分の平均実行命令数は 158(12.22%削除) である。

(表 3.8 に示す)

- get_mpf
  エラーチェックが不要となる場合 (OPT)：
  一回目:(MPF_A により、get_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)
  開始時刻は 8,104 であり、完了時刻は 8,289 である。よって実行時間は 185 である。
  開始命令は 4,827 であり、完了命令は 4,930 である。よって実行命令数は 103 である。
  二回目:(MPF_B により、get_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)
  開始時刻は 8,544 であり、完了時刻は 8,649 である。よって実行時間は 105 である。
  開始命令は 5,065 であり、完了命令は 5,168 である。よって実行命令数は 103 である。
  三回目:(MPF_C により、get_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)
  開始時刻は 8,804 であり、完了時刻は 8,909 である。よって実行時間は 105 である。
  開始命令は 5,303 であり、完了命令は 5,406 である。よって実行命令数は 103 である。

  一回目から三回目まで実行した三回分の平均実行時間は 132(10.20%削除) である。
  一回目から三回目まで実行した三回分の平均実行命令数は 103(8.04%削除) である。
  (表 3.8 に示す)

- rel_mpf
  エラーチェックが不要となる場合 (OPT)：
  一回目:(MPF_A により、rel_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)
  開始時刻は 11,737 であり、完了時刻は 11,936 である。よって実行時間は 199 である。
  開始命令は 7,086 であり、完了命令は 7,184 である。よって実行命令数は 98 である。
  二回目:(MPF_B により、rel_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)
  開始時刻は 12,094 であり、完了時刻は 12,203 である。よって実行時間は 109 である。
  開始命令は 7,322 であり、完了命令は 7,420 である。よって実行命令数は 98 である。
  三回目:(MPF_C により、rel_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)
  開始時刻は 12,358 であり、完了時刻は 12,467 である。よって実行時間は 109 である。

開始命令は 7,565 あり、完了命令は 7,663 である。よって実行命令数は 98 である。

一回目から三回目まで実行した三回分の平均実行時間は 139(9.74%削除) である。
一回目から三回目まで実行した三回分の平均実行命令数は 98(10.90%削除) である。
(表 3.8 に示す)

メールボックス機能について、エラーチェックコードが必要な場合 (FULL) の実行時間と実行命令数を以下に示す。

- cre_mbx
  エラーチェックコードが必要場合 (FULL) :
  一回目:(MBX_A の登録のための cre_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 3,578 であり、完了時刻は 3,996 である。よって実行時間は 418 である。
  開始命令は 1,812 であり、完了命令は 2,050 である。よって実行命令数は 238 である。
  二回目:(MBX_B の登録のための cre_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 4,233 であり、完了時刻は 4,501 である。よって実行時間は 268 である。
  開始命令は 2,245 であり、完了命令は 2,483 である。よって実行命令数は 238 である。
  三回目:(MBX_C の登録のための cre_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 4,738 であり、完了時刻は 4,986 である。よって実行時間は 248 である。
  開始命令は 2,678 であり、完了命令は 2,916 である。よって実行命令数は 238 である。

  一回目から三回目まで実行した三回分の平均実行時間は 311 である。
  一回目から三回目まで実行した三回分の平均実行命令数は 238 である。(表 3.7 に示す)

- snd_mbx
  エラーチェックコードが必要場合 (FULL) :
  一回:(MBX_A により、snd_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 8,746 であり、完了時刻は 8,977 である。よって実行時間は 231 である。
  開始命令は 5,187 であり、完了命令は 5,308 である。よって実行命令数は 121 である。
  二回目:(MBX_B により、snd_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)

開始時刻は 9,123 であり、完了時刻は 9,244 である。よって実行時間は 121 である。
開始命令は 5,442 であり、完了命令は 5,563 である。よって実行命令数は 121 である。
三回目:(MBX_C により、snd_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
開始時刻は 9,399 であり、完了時刻は 9,520 である。よって実行時間は 121 である。
開始命令は 5,696 であり、完了命令は 5,817 である。よって実行命令数は 121 である。

一回目から三回目まで実行した三回分の平均実行時間は 158 である。
一回目から三回目まで実行した三回分の平均実行命令数は 121 である。(表 3.7 に示す)

- rcv_mbx
  エラーチェックコードが必要場合 (FULL) :
  一回目:(MBX_A により、rcv_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 11,950 であり、完了時刻は 12,181 である。よって実行時間は 231 である。
  開始命令は 7,239 であり、完了命令は 7,360 である。よって実行命令数は 121 である。
  二回目:(MBX_B により、rcv_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 12,441 であり、完了時刻は 12,569 である。よって実行時間は 128 である。
  開始命令は 7,489 あり、完了命令は 7,617 である。よって実行命令数は 128 である。
  三回目:(MBX_C により、rcv_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 12,718 であり、完了時刻は 12,853 である。よって実行時間は 135 である。
  開始命令は 7,745 であり、完了命令は 7,880 である。よって実行命令数は 135 である。

  一回目から三回目まで実行した三回分の平均実行時間は 165 である。
  一回目から三回目まで実行した三回分の平均実行命令数は 128 である。(表 3.7 に示す)

メールボックス機能について、エラーチェックコードが不要となる場合 (OPT) の実行時間と実行命令数を以下に示す。

- cre_mbx
  エラーチェックが不要となる場合 (OPT)：
  一回目: (MBX_A の登録のための cre_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 3,452 であり、完了時刻は 3,801 である。よって実行時間は 349 である。
  開始命令は 1,746 であり、完了命令は 1,955 である。よって実行命令数は 209 である。
  二回目:(MBX_B の登録のための cre_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 4,016 であり、完了時刻は 4,255 である。よって実行時間は 239 である。
  開始命令は 2,128 であり、完了命令は 2,337 である。よって実行命令数は 209 である。
  三回目:(MBX_C の登録のための cre_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 4,470 であり、完了時刻は 4,689 である。よって実行時間は 219 である。
  開始命令は 2,510 であり、完了命令は 2,719 である。よって実行命令数は 209 である。

  一回目から三回目まで実行した三回分の平均実行時間は 269(13.50%削除) である。
  一回目から三回目まで実行した三回分の平均実行命令数は 209(12.18%削除) である。
  (表 3.8 に示す)

- snd_mbx
  エラーチェックが不要となる場合 (OPT)：
  一回目:(MBX_A により、snd_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 8,337 であり、完了時刻は 8,550 である。よって実行時間は 213 である。
  開始命令は 4,948 であり、完了命令は 5,061 である。よって実行命令数は 116 である。
  二回目:(MBX_B により、snd_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 8,687 であり、完了時刻は 8,800 である。よって実行時間は 113 である。
  開始命令は 5,186 であり、完了命令は 5,299 である。よって実行命令数は 113 である。
  三回目:(MBX_C により、snd_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 8,946 であり、完了時刻は 9,059 である。よって実行時間は 113 である。
  開始命令は 5,423 であり、完了命令は 5,536 である。よって実行命令数は 113 である。

  一回目から三回目まで実行した三回分の平均実行時間は 146(8.22%削除) である。
  一回目から三回目まで実行した三回分の平均実行命令数は 114(5.79%削除) である。
  (表 3.8 に示す)

- rcv_mbx
  エラーチェックが不要となる場合 (OPT)：
  一回目:(MBX_A により、rcv_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 9,766 であり、完了時刻は 9,979 である。よって実行時間は 213 である。
  開始命令は 5,999 であり、完了命令は 6,112 である。よって実行命令数は 113 である。
  二回目:(MBX_B により、rcv_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 11,950 であり、完了時刻は 12,070 である。よって実行時間は 120 である。
  開始命令は 7,188 であり、完了命令は 7,308 である。よって実行命令数は 120 である。
  三回目:(MBX_C により、rcv_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 12,207 であり、完了時刻は 12,334 である。よって実行時間は 127 である。
  開始命令は 7,424 であり、完了命令は 7,551 である。よって実行命令数は 127 である。

  一回目から三回目まで実行した三回分の平均実行時間は 153(7.27%削除) である。
  一回目から三回目まで実行した三回分の平均実行命令数は 120(6.25%削除) である。
  (表 3.8 に示す)

実験 3 においてセマフォ機能についてのプログラムを実行した。実行したプログラムについて、主なコードを以下に示す。

```
void task_a (VP_INT exinf)
{
   /*    省略    */
   wai_sem( SEM_A ); //SEM_A により、セマフォ資源の獲得 (一回目)
   /*    処理    */
   sig_sem( SEM_A ); //SEM_A により、セマフォ資源の返却 (一回目)

   wai_sem( SEM_B ); //SEM_B により、セマフォ資源の獲得 (二回目)
   /*    処理    */
   sig_sem( SEM_B );  //SEM_B により、セマフォ資源の返却 (二回目)
}
```

表 3.7: 固定長メモリプール機能とメールボックス機能についての実験 2FULL

| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
|---|---|---|---|---|---|---|---|
| cre_mpf | 1 回目 | 3,217 | 3,549 | 332 | 1,623 | 1,803 | 180 |
| | 2 回目 | 4,014 | 4,216 | 202 | 2,058 | 2,238 | 180 |
| | 3 回目 | 4,519 | 4,721 | 202 | 2,491 | 2,671 | 180 |
| | 3 回分の平均実行時間:245 | | | | 3 回分の平均実行命令数:180 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| get_mpf | 1 回目 | 8,494 | 8,708 | 214 | 5,057 | 5,169 | 112 |
| | 2 回目 | 8981 | 9,095 | 114 | 5,312 | 5,424 | 112 |
| | 3 回目 | 9,258 | 9,372 | 114 | 5,567 | 5,679 | 112 |
| | 3 回分の平均実行時間:147 | | | | 3 回分の平均実行命令数:112 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| rel_mpf | 1 回目 | 12,206 | 12,427 | 221 | 7,375 | 7,485 | 110 |
| | 2 回目 | 12,593 | 12,714 | 121 | 7,631 | 7,741 | 110 |
| | 3 回目 | 12,877 | 12,998 | 121 | 7,894 | 8,004 | 110 |
| | 3 回分の平均実行時間:154 | | | | 3 回分の平均実行命令数:110 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| cre_mbx | 1 回目 | 3,578 | 3,996 | 418 | 1,812 | 2,050 | 238 |
| | 2 回目 | 4,233 | 4,501 | 268 | 2,245 | 2,483 | 238 |
| | 3 回目 | 4,738 | 4,986 | 248 | 2,678 | 2,916 | 238 |
| | 3 回分の平均実行時間:311 | | | | 3 回分の平均実行命令数:238 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| snd_mbx | 1 回目 | 8,746 | 8,977 | 231 | 5,187 | 5,308 | 121 |
| | 2 回目 | 9,123 | 9,244 | 121 | 5,442 | 5,563 | 121 |
| | 3 回目 | 9,399 | 9,520 | 121 | 5,696 | 5,817 | 121 |
| | 3 回分の平均実行時間:158 | | | | 3 回分の平均実行命令数:121 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| rcv_mbx | 1 回目 | 11,950 | 12,181 | 231 | 7,239 | 7,360 | 121 |
| | 2 回目 | 12,441 | 12,569 | 128 | 7,489 | 7,617 | 128 |
| | 3 回目 | 12,718 | 12,853 | 135 | 7,745 | 7,880 | 135 |
| | 3 回分の平均実行時間:165 | | | | 3 回分の平均実行命令数:128 | | |

表 3.8: 固定長メモリプール機能とメールボックス機能についての実験 2OPT

| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
|---|---|---|---|---|---|---|---|
| cre_mpf | 1 回目 | 3,153 | 3,423 | 270 | 1,579 | 1,737 | 158 |
| | 2 回目 | 3,819 | 3,999 | 180 | 1,963 | 2,121 | 158 |
| | 3 回目 | 4,273 | 4,453 | 180 | 2,345 | 2,503 | 158 |
| | 3 回分の平均実行時間:210 | | | | 3 回分の平均実行命令数:158 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| get_mpf | 1 回目 | 8,104 | 8,289 | 185 | 4,827 | 4,930 | 103 |
| | 2 回目 | 8,544 | 8,649 | 105 | 5,065 | 5,168 | 103 |
| | 3 回目 | 8,804 | 8,909 | 105 | 5,303 | 5,406 | 103 |
| | 3 回分の平均実行時間:132 | | | | 3 回分の平均実行命令数:103 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| rel_mpf | 1 回目 | 11,737 | 11,936 | 199 | 7,086 | 7,184 | 98 |
| | 2 回目 | 12,094 | 12,203 | 109 | 7,322 | 7,420 | 98 |
| | 3 回目 | 12,358 | 12,467 | 109 | 7,565 | 7,663 | 98 |
| | 3 回分の平均実行時間:139 | | | | 3 回分の平均実行命令数:98 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| cre_mbx | 1 回目 | 3,452 | 3,801 | 349 | 1,746 | 1,955 | 209 |
| | 2 回目 | 4,016 | 4,255 | 239 | 2,128 | 2,337 | 209 |
| | 3 回目 | 4,470 | 4,689 | 219 | 2,510 | 2,719 | 209 |
| | 3 回分の平均実行時間:269 | | | | 3 回分の平均実行命令数:209 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| snd_mbx | 1 回目 | 8,337 | 8,550 | 213 | 4,948 | 5,061 | 116 |
| | 2 回目 | 8,687 | 8,800 | 113 | 5,186 | 5,299 | 113 |
| | 3 回目 | 8,946 | 9,059 | 113 | 5,423 | 5,536 | 113 |
| | 3 回分の平均実行時間:146 | | | | 3 回分の平均実行命令数:114 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| rcv_mbx | 1 回目 | 9,766 | 9,979 | 213 | 5,999 | 6,112 | 113 |
| | 2 回目 | 11,950 | 12,070 | 120 | 7,188 | 7,308 | 120 |
| | 3 回目 | 12,207 | 12,334 | 127 | 7,424 | 7,551 | 127 |
| | 3 回分の平均実行時間:153 | | | | 3 回分の平均実行命令数:120 | | |

```
void task_b (VP_INT exinf)
{
   /*    省略    */
   wai_sem ( SEM_C ); //SEM_C により、セマフォ資源の獲得 (三回目)
   /*    処理    */
   sig_sem ( SEM_C ); //SEM_C により、セマフォ資源の返却 (三回目)
}

/*システムコンフィギュレーションファイル"system.cfg"*/

 /*    省略    */

CRE_TSK ( TSK_A, { (TA_HLNG | TA_ACT ), 255, task_a, 2, 4096, NULL } );
CRE_TSK ( TSK_B, { TA_HLNG, 511, task_b, 3, 4096, NULL });

CRE_SEM ( SEM_A, { TA_TFIFO, 1, 1} );
//SEM_A の登録により、,cre_sem が実行される (一回目)
CRE_SEM ( SEM_B, { TA_TFIFO, 1, 1} );
//SEM_B の登録により、,cre_sem が実行される (二回目)
CRE_SEM ( SEM_C, { TA_TFIFO, 1, 1} );
//SEM_C の登録により、,cre_sem が実行される (三回目)

/*    省略    */
```

プログラムの説明:
タスクとして task_a、task_b、セマフォとして SEM_A、SEM_B、SEM_C を用意する。コンフィギュレーションファイル内で“CRE_SEM”によって各セマフォを登録することにより、システム初期化時に cre_sem システムコールが実行される。 task_a において、SEM_A に対して、wai_sem (セマフォ資源の獲得) を行い、ある処理をして、sig_sem (セマフォ資源の返却) を行う。同様に SEM_B に対して、セマフォ資源の獲得、返却を行う。また、task_b において、SEM_C を利用して、セマフォ資源の獲得、セマフォ資源の返却を行う。

このアプリケーションに対して提案手法によってエラーチェックコードの要・不要を判断した結果は以下の通りである。
define.h
・
・
・

#define SEM_E_ID_CHECK　0
#define SEM_E_NOEXS　0
#define SEM_E_RSATR_CHECK　0
#define CRE_SEM_E_PAR_CHECK　0
#define SEM_E_OBJ_CHECK　0
・
・
・
・

セマフォ機能について、エラーチェックコードが必要な場合 (FULL) の実行時間と実行命令数を以下に示す。

- cre_sem
  エラーチェックコードが必要場合 (FULL)：
  一回目:(SEM_A の登録のための cre_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 3,167 であり、完了時刻は 3,399 である。よって実行時間は 232 である。
  開始命令は 1,563 であり、完了命令は 1,685 である。よって実行命令数は 122 である。
  二回目:(SEM_B の登録のための cre_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 3,146 であり、完了時刻は 3,548 である。よって実行時間は 132 である。
  開始命令は 1,692 であり、完了命令は 1,814 である。よって実行命令数は 122 である。
  三回目:(SEM_C の登録のための cre_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 3,565 であり、完了時刻は 3,687 である。よって実行時間は 122 である。
  開始命令は 1,821 であり、完了命令は 1,943 である。よって実行命令数は 122 である。

  一回目から三回目まで実行した三回分の平均実行時間は 162 である。
  一回目から三回目まで実行した三回分の平均実行命令数は 122 である。(表 3.9 に示す)

- sig_sem
  エラーチェックコードが必要場合 (FULL)：
  一回目:(SEM_A により、sig_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 7,546 であり、完了時刻は 7,714 である。よって実行時間は 168 である。

開始命令は 4,325 であり、完了命令は 4,423 である。よって実行命令数は 98 である。
二回目:(SEM_B により、sig_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
開始時刻は 7,846 であり、完了時刻は 7,944 である。よって実行時間は 98 である。
開始命令は 4,545 であり、完了命令は 4,643 である。よって実行命令数は 98 である。
三回目:(SEM_C により、sig_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
開始時刻は 8,994 であり、完了時刻は 9,092 である。よって実行時間は 98 である。
開始命令は 5,459 であり、完了命令は 5,557 である。よって実行命令数は 98 である。

一回目から三回目まで実行した三回分の平均実行時間は 121 である。
一回目から三回目まで実行した三回分の平均実行命令数は 98 である。(表 3.9 に示す)

- wai_sem
  エラーチェックコードが必要場合 (FULL) :
  一回目:(SEM_A により、wai_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 7,185 であり、完了時刻は 7,367 である。よって実行時間は 182 である。
  開始命令は 4,084 であり、完了命令は 4,186 である。よって実行命令数は 102 である。
  二回目:(SEM_B により、wai_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 7,707 であり、完了時刻は 7,809 である。よって実行時間は 102 である。
  開始命令は 4,426 であり、完了命令は 4,528 である。よって実行命令数は 102 である。
  三回目:(SEM_C により、wai_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 8,602 であり、完了時刻は 8,714 である。よって実行時間は 112 である。
  開始命令は 5,107 であり、完了命令は 5,209 である。よって実行命令数は 102 である。

  一回目から三回目まで実行した三回分の平均実行時間は 132 である。
  一回目から三回目まで実行した三回分の平均実行命令数は 102 である。(表 3.9 に示す)

セマフォ機能について、エラーチェックコードが不要となる場合 (OPT) の実行時間と実行命令数を以下に示す。

- cre_sem
  エラーチェックが不要となる場合 (OPT)：
  一回目:(SEM_A の登録のための cre_sem の処理が始まってから、終わるまでの実行時間実行時間と実行命令数の値である)
  開始時刻は 3,073 であり、完了時刻は 3,204 である。よって実行時間は 167 である。
  開始命令は 1,519 であり、完了命令は 1,616 である。よって実行命令数は 97 である。
  二回目:(SEM_B の登録のための cre_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 3,257 であり、完了時刻は 3,364 である。よって実行時間は 107 である。
  開始命令は 1,623 であり、完了命令は 1,720 である。よって実行命令数は 97 である。
  三回目:(SEM_C の登録のための cre_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 3,381 であり、完了時刻は 3,478 である。よって実行時間は 97 である。
  開始命令は 1,727 であり、完了命令は 1,824 である。よって実行命令数は 97 である。

  一回目から三回目まで実行した三回分の平均実行時間は 124(23.46%削除) である。
  一回目から三回目まで実行した三回分の平均実行命令数は 97(20.49%削除) である。
  (表 3.10 に示す)

- sig_sem
  エラーチェックが不要となる場合 (OPT)：
  一回目:(SEM_A により、sig_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 7,286 であり、完了時刻は 7,446 である。実行時間は 160 である。
  開始命令は 4,165 であり、完了命令は 4,255 である。よって実行命令数は 90 である。
  二回目:(SEM_B により、sig_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 7,570 であり、完了時刻は 7,660 である。よって実行時間は 90 である。
  開始命令は 4,369 であり、完了命令は 4,459 である。よって実行命令数は 90 である。
  三回目:(SEM_C により、sig_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 8,712 であり、完了時刻は 8,802 である。よって実行時間は 90 である。
  開始命令は 5,267 であり、完了命令は 5,357 である。よって実行命令数は 90 である。

  一回目から三回目まで実行した三回分の平均実行時間は 113(6.61%削除) である。
  一回目から三回目まで実行した三回分の平均実行命令数は 90(8.16%削除) である。
  (表 3.10 に示す)

- wai_sem
  エラーチェックが不要となる場合 (OPT)：
  一回目:(SEM_A により、wai_sem の処理が始まるから、終わるまで実行時間と実行命令数の値である)
  開始時刻は 6,953 であり、完了時刻は 7107 である。よって実行時間は 154 である。
  開始命令は 3,932 であり、完了命令は 4,026 である。よって実行命令数は 94 である。
  二回目:(SEM_B により、wai_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 7,439 であり、完了時刻は 7,533 である。よって実行時間は 94 である。
  開始命令は 4,258 であり、完了命令は 4,352 である。よって実行命令数は 94 である。
  三回目:(SEM_C により、wai_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 8,328 であり、完了時刻は 8,432 である。よって実行時間は 104 である。
  開始命令は 4,923 であり、完了命令は 5,017 である。よって実行命令数は 94 である。

  一回目から三回目まで実行した三回分の実行した三回の平均実行時間は 117(11.36%削除) である。
  一回目から三回目まで実行した三回分の平均実行命令数は 94(7.84%削除) である。
  (表 3.10 に示す)

### 3.2.3 ロックコードの削除によるバイナリコードのサイズの減少

対象システムコール (タスク管理機能、固定長メモリプール機能、メールボックス機能、セマフォ機能) 内のロックコードをヘッダファイルの情報に従って、削除できない場合と削除できる場合を比べた際、削除できる場合のバイナリコードのサイズが減っていることを以下に示す。

- タスク管理機能の cre_tsk
  ロックコードが必要な場合：
  バイナリコードのサイズは 848 である。
  ロックコードが不要となる場合：
  バイナリコードのサイズは 836(1.41%削除) になる。 (表 3.11 に示す)

表 3.9: セマフォ機能についての実験 3FULL

| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
|---|---|---|---|---|---|---|---|
| cre_sem | 1 回目 | 3,167 | 3,399 | 232 | 1,563 | 1,685 | 122 |
| | 2 回目 | 3,146 | 3,548 | 132 | 1,692 | 1,814 | 122 |
| | 3 回目 | 3,565 | 3,687 | 122 | 1,821 | 1,943 | 122 |
| | 3 回分の平均実行時間:162 | | | | 3 回分の平均実行命令数:122 | | |
| sig_sem | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| | 1 回目 | 7,546 | 7,714 | 168 | 4,325 | 4,423 | 98 |
| | 2 回目 | 7,846 | 7,944 | 98 | 4,545 | 4,643 | 98 |
| | 3 回目 | 8,994 | 9,092 | 98 | 5,459 | 5,557 | 98 |
| | 3 回分の平均実行時間:121 | | | | 3 回分の平均実行命令数:98 | | |
| wai_sem | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| | 1 回目 | 7,185 | 7,367 | 182 | 4,084 | 4,186 | 102 |
| | 2 回目 | 7,707 | 7,809 | 102 | 4,426 | 4,528 | 102 |
| | 3 回目 | 8,602 | 8,714 | 112 | 5,107 | 5,209 | 102 |
| | 3 回分の平均実行時間:132 | | | | 3 回分の平均実行命令数:102 | | |

表 3.10: セマフォ機能についての実験 3OPT

| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
|---|---|---|---|---|---|---|---|
| cre_sem | 1 回目 | 3,073 | 3,240 | 167 | 1,519 | 1,616 | 97 |
| | 2 回目 | 3,257 | 3,364 | 107 | 1,623 | 1,720 | 97 |
| | 3 回目 | 3,381 | 3,478 | 97 | 1,727 | 1,824 | 97 |
| | 3 回分の平均実行時間:124 | | | | 3 回分の平均実行命令数:97 | | |
| sig_sem | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| | 1 回目 | 7,286 | 7,446 | 160 | 4,165 | 4,255 | 90 |
| | 2 回目 | 7,570 | 7,660 | 90 | 4,369 | 4,459 | 90 |
| | 3 回目 | 8,712 | 8,802 | 90 | 5,267 | 5,357 | 90 |
| | 3 回分の平均実行時間:113 | | | | 3 回分の平均実行命令数:90 | | |
| wai_sem | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| | 1 回目 | 6,953 | 7,107 | 154 | 3,932 | 4,026 | 94 |
| | 2 回目 | 7,439 | 7,533 | 94 | 4,258 | 4,352 | 94 |
| | 3 回目 | 8,328 | 8,432 | 104 | 4,923 | 5,017 | 94 |
| | 3 回分の平均実行時間:117 | | | | 3 回分の平均実行命令数:94 | | |

- タスク管理機能の act_tsk
  ロックコードが必要な場合：
  バイナリコードのサイズは 472 である。
  ロックコードが不要となる場合：
  バイナリコードのサイズは 444(5.93%削除) になる。 (表 3.11 に示す)

- 固定長メモリプール機能の cre_mpf
  ロックコードが必要な場合：
  バイナリコードのサイズは 424 である。
  ロックコードが不要となる場合：
  バイナリコードのサイズは 388(8.49%削除) になる。 (表 3.11 に示す)

- 固定長メモリプール機能の rel_mpf
  ロックコードが必要な場合：
  バイナリコードのサイズは 444 である。
  ロックコードが不要となる場合：
  バイナリコードのサイズは 408(6.30%削除) になる。 (表 3.11 に示す)

- 固定長メモリプール機能の get_mpf
  ロックコードが必要な場合：
  バイナリコードのサイズは 416 である。
  ロックコードが不要となる場合：
  バイナリコードのサイズは 384(7.69%削除) になる。 (表 3.11 に示す)

- メールボックス機能の cre_mbx
  ロックコードが必要な場合：
  バイナリコードのサイズは 484 である。
  ロックコードが不要となる場合：
  バイナリコードのサイズは 448(7.43%削除) になる。 (表 3.11 に示す)

- メールボックス機能の snd_mbx
  ロックコードが必要な場合：
  バイナリコードのサイズは 560 である。
  ロックコードが不要となる場合：
  バイナリコードのサイズは 528(5.71%削除) になる。 (表 3.11 に示す)

- メールボックス機能の rcv_mbx
  ロックコードが必要な場合：
  バイナリコードのサイズは 456 である。
  ロックコードが不要となる場合：
  バイナリコードのサイズは 432(5.26%削除) になる。 (表 3.11 に示す)

- セマフォ機能の cre_sem
  ロックコードが必要な場合：
  バイナリコードのサイズは 328 である。
  ロックコードが不要となる場合：
  バイナリコードのサイズは 292(10.98%削除) になる。 (表 3.11 に示す)

- セマフォ機能の sig_sem
  ロックコードが必要な場合：
  バイナリコードのサイズは 396 である。
  ロックコードが不要となる場合：
  バイナリコードのサイズは 360(9.09%削除) になる。 (表 3.11 に示す)

- セマフォ機能の wai_sem
  ロックコードが必要な場合：
  バイナリコードのサイズは 344 である。
  ロックコードが不要となる場合：
  バイナリコードのサイズは 320(6.98%削除) になる。 (表 3.11 に示す)

### 3.2.4 ロックコードの削除による実行時間と実行命令数の減少

対象システムコール (タスク管理機能、固定長メモリプール機能、メールボックス機能、セマフォ機能) 内のロックコードを削除できない場合と削除できる場合を比べた際、三つの実験によって、実行時間と実行命令数が減っていることを以下に示す。

実験 4 においてタスク管理機能についてのプログラムを実行した。実行したプログラムは実験 1 のプログラムと同じものである。

タスク管理機能について、ロックコードが必要な場合 (FULL) の実行時間と実行命令数を以下に示す。

表 3.11: ロックコードの削除による各対象システムコールのバイナリコードのサイズの減小結果

| | ロックコード必要 | ロックコード不要 |
|---|---|---|
| cre_tsk | 848 | 836 |
| act_tsk | 472 | 444 |
| cre_mpf | 424 | 388 |
| rel_mpf | 444 | 408 |
| get_mpf | 416 | 384 |
| cre_mbx | 484 | 448 |
| snd_mbx | 560 | 528 |
| rcv_mbx | 456 | 432 |
| cre_sem | 328 | 292 |
| sig_sem | 396 | 360 |
| wai_sem | 344 | 320 |

- cre_tsk
  ロックコードが必要な場合 (FULL)：
  一回目:( task_a の登録のための cre_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
  開始時刻は 1,944 であり、完了時刻は 2,882 である。よって実行時間は 938 である。
  開始命令は 1,014 であり、完了命令は 1,400 である。よって実行命令数は 386 である。
  二回目:( task_b の登録のための cre_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
  開始時刻は 2,919 であり、完了時刻は 3,187 である。よって実行時間は 268 である。
  開始命令は 1,417 であり、完了命令は 1,623 である。よって実行命令数は 206 である。
  三回目:( task_c の登録のための cre_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
  開始時刻は 3,224 であり、完了時刻は 3,502 である。よって実行時間は 278 である。
  開始命令は 1,640 であり、完了命令は 1,846 である。よって実行命令数は 206 である。
  四回目:( task_d の登録のための cre_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
  開始時刻は 3,537 であり、。完了時刻は 3,805 である。よって実行時間は 268 である。
  開始命令は 1,861 であり、完了命令は 2,067 である。よって実行命令数は 206 である。

  一回目から四回目まで実行した四回分の平均実行時間は 438 である。
  一回目から四回目まで実行した四回分の平均実行命令数は 251 である。(表 3.12 に

示す)

- act_tsk
  ロックコードが必要場合 (FULL)：
  一回目:(task_b の起動のための act_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
  開始時刻は 6,811 であり、完了時刻は 7,291 である。よって実行時間は 480 である。
  開始命令は 3,898 であり、完了命令は 4,206 である。よって実行命令数は 308 である。
  二回目:(task_c の起動のための act_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
  開始時刻は 7,294 であり、完了時刻は 7,612 である。よって実行時間は 318 である。
  開始命令は 4,209 であり、完了命令は 4,525 である。よって実行命令数は 316 である。
  三回目:(task_d の起動のための act_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
  開始時刻は 7,615 であり、完了時刻は 7,951 である。よって実行時間は 336 である。
  開始命令は 4,528 であり、完了命令は 4,852 である。よって実行命令数は 324 である。

  一回目から三回目まで実行した三回分の平均実行時間は 378 である。
  一回目から三回目まで実行した三回分の平均実行命令数は 316 である。(表 3.12 に示す)

タスク管理機能について、ロックコードが不要となる場合 (OPT) の実行時間と実行命令数を以下に示す。

- cre_tsk
  ロックコードが不要となる場合 (OPT)：
  一回目:( task_a の生成のための cre_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
  開始時刻は 1,954 であり、完了時刻は 2,803 である。よって実行時間は 849 である。
  開始命令は 1,014 であり、完了命令は 1,351 である。よって実行命令数は 337 である。
  二回目:(task_b の生成のための cre_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
  開始時刻は 2,840 であり、完了時刻は 3,072 である。よって実行時間は 232 である。
  開始命令は 1,368 であり、完了命令は 1,528 である。よって実行命令数は 160 である。
  三回目:(task_c の生成のための cre_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
  開始時刻は 3,109 であり、完了時刻は 3,341 である。よって実行時間は 232 である。

開始命令は 1,545 であり、完了命令は 1,705 である。よって実行命令数は 160 である。
四回目:(task_d の生成のための cre_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
開始時刻は 3,376 であり、完了時刻は 3,598 である。よって実行時間は 222 である。
開始命令は 1,720 であり、完了命令は 1,880 である。よって実行命令数は 160 である。

一回目から四回目まで実行した四回分の平均実行時間は 384(12.33%削除) である。
一回目から四回目まで実行した四回分の平均実行命令数は 204(18.73%削除) である。
(表 3.13 に示す)

- act_tsk
  ロックコードが不要となる場合 (OPT) :
  一回目:(task_b の起動のための act_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
  開始時刻は 6,555 であり、完了時刻は 6,985 である。よって実行時間は 430 である。
  開始命令は 3,662 であり、完了命令は 3,920 である。よって実行命令数は 258 である。
  二回目:(task_c の起動のための act_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
  開始時刻は 6,988 であり、完了時刻は 7,256 である。よって実行時間は 268 である。
  開始命令は 3,923 であり、完了命令は 4,189 である。よって実行命令数は 266 である。
  三回目:(task_d の起動のための act_tsk の処理が始まってから、終わるまでの実行時間と実行命令数の値である)。
  開始時刻は 7,259 であり、完了時刻は 7,545 である。よって実行時間は 286 である。
  開始命令は 4,192 であり、完了命令は 4,466 である。よって実行命令数は 274 である。

  一回目から三回目まで実行した三回分の平均実行時間は 328(13.23%削除) である。
  一回目から三回目まで実行した三回分の平均実行命令数は 266(15.82%削除) である。
  (表 3.13 に示す)

実験 5 において固定長メモリプール機能、メールボックス機能についてのプログラムを実行した。実行したプログラムは実験 2 のプログラムと同じものである。(get_mpf と rel_mpf のロックコードの要・不要の判断は未実装のため、全て不要であると仮定する。)

固定長メモリプール機能、ロックコードが必要な場合 (FULL) の実行時間と実行命令数を以下に示す。

表 3.12: タスク機能についての実験 4FULL

| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
|---|---|---|---|---|---|---|---|
| cre_tsk | 1 回目 | 1,944 | 2,882 | 938 | 1,014 | 1,400 | 386 |
| | 2 回目 | 2,919 | 3,187 | 268 | 1,417 | 1,623 | 206 |
| | 3 回目 | 3,224 | 3,502 | 278 | 1,640 | 1,846 | 206 |
| | 4 回目 | 3,537 | 3,805 | 268 | 1,861 | 2,067 | 206 |
| | 4 回分の平均実行時間:438 | | | | 4 回分の平均実行命令数:251 | | |

| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
|---|---|---|---|---|---|---|---|
| act_tsk | 1 回目 | 6,811 | 7,291 | 480 | 3,898 | 4,206 | 308 |
| | 2 回目 | 7,294 | 7,612 | 318 | 4,209 | 4,525 | 316 |
| | 3 回目 | 7,615 | 7,951 | 336 | 4,528 | 4,852 | 324 |
| | 3 回分の平均実行時間:378 | | | | 3 回分の平均実行命令数:316 | | |

表 3.13: タスク機能についての実験 4OPT

| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
|---|---|---|---|---|---|---|---|
| cre_tsk | 1 回目 | 1,954 | 2,803 | 849 | 1,014 | 1,351 | 337 |
| | 2 回目 | 2,840 | 3,072 | 232 | 1,368 | 1,528 | 160 |
| | 3 回目 | 3,109 | 3,341 | 232 | 1,545 | 1,705 | 160 |
| | 4 回目 | 3,376 | 3,598 | 222 | 1,720 | 1,880 | 160 |
| | 4 回分の平均実行時間:384 | | | | 4 回分の平均実行命令数:204 | | |

| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
|---|---|---|---|---|---|---|---|
| act_tsk | 1 回目 | 6,555 | 6,985 | 430 | 3,662 | 3,920 | 258 |
| | 2 回目 | 6,988 | 7,256 | 268 | 3,923 | 4,189 | 266 |
| | 3 回目 | 7,259 | 7,545 | 286 | 4,192 | 4,466 | 274 |
| | 3 回分の平均実行時間:328 | | | | 3 回分の平均実行命令数:266 | | |

- cre_mpf
  ロックコードが必要場合 (FULL)：
  一回目:(MPF_A の登録のための cre_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 3,217 であり、完了時刻は 3,549 である。よって実行時間は 332 である。
  開始命令は 1,623 であり、完了命令は 1,803 である。よって実行命令数は 180 である。
  二回目:（MPF_B の登録のための cre_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 4,014 であり、完了時刻は 4,216 である。よって実行時間は 202 である。
  開始命令は 2,058 であり、完了命令は 2,238 である。よって実行命令数は 180 である。
  三回目:(MPF_C の登録のための cre_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 4,519 であり、完了時刻は 4,721 である。よって実行時間は 202 である。
  開始命令は 2,491 であり、完了命令は 2,671 である。よって実行命令数は 180 である。

  一回目から三回目まで実行した三回分の平均実行時間は 245 である。
  一回目から三回目まで実行した三回分の平均実行命令数は 180 である。(表 3.14 に示す)

- get_mpf
  ロックコードが必要場合 (FULL)：
  一回目:(MPF_A により、get_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)
  開始時刻は 8,494 であり、完了時刻は 8,708 である。よって実行時間は 214 である。
  開始命令は 5,057 であり、完了命令は 5,169 である。よって実行命令数は 112 である。
  二回目:(MPF_B により、get_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)
  開始時刻は 8,981 であり、完了時刻は 9,095 である。よって実行時間は 114 である。
  開始命令は 5,312 であり、完了命令は 5,424 である。よって実行命令数は 112 である。
  三回目:(MPF_C により、get_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)
  開始時刻は 9,258 であり、完了時刻は 9,372 である。よって実行時間は 114 である。
  開始命令は 5,567 であり、完了命令は 5,679 である。よって実行命令数は 112 である。

  一回目から三回目まで実行した三回分の平均実行時間は 147 である。
  一回目から三回目まで実行した三回分の平均実行命令数は 112 である。(表 3.14 に示す)

- rel_mpf
  ロックコードが必要場合 (FULL)：
  一回目:(MPF_A により、rel_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)
  開始時刻は 12,206 であり、完了時刻は 12,427 である。よって実行時間は 221 である。
  開始命令は 7,375 であり、完了命令は 7,485 である。よって実行命令数は 110 である。
  二回目:(MPF_B により、rel_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)
  開始時刻は 12,593 であり、完了時刻は 12,714 である。よって実行時間は 121 である。
  開始命令は 7,631 であり、完了命令は 7,741 である。よって実行命令数は 110 である。
  三回目:(MPF_C により、rel_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)
  開始時刻は 12,877 であり、完了時刻は 12,998 である。よって実行時間は 121 である。
  開始命令は 7,894 であり、完了命令は 8,004 である。よって実行命令数は 110 である。

  一回目から三回目まで実行した三回分の平均実行時間は 154 である。
  一回目から三回目まで実行した三回分の平均実行命令数は 110 である。(表 3.14 に示す)

固定長メモリプール機能について、ロックコードが不要となる場合 (OPT) の実行時間と実行命令数を以下に示す。

- cre_mpf
  ロックコードが不要となる場合 (OPT)：
  一回目:(MPF_A の登録のための cre_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 3,207 であり、完了時刻は 3,485 である。よって実行時間は 278 である。
  開始命令は 1,623 であり、完了命令は 1,759 である。よって実行命令数は 136 である。
  二回目:(MPF_B の登録のための cre_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 3,950 であり、完了時刻は 4,108 である。よって実行時間は 158 である。
  開始命令は 2,014 であり、完了命令は 2,150 である。よって実行命令数は 136 である。

三回目:(MPF_C の登録のための cre_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
開始時刻は 4,411 であり、完了時刻は 4,569 である。よって実行時間は 158 である。
開始命令は 2,403 であり、完了命令は 2,539 である。よって実行命令数は 136 である。

一回目から三回目まで実行した三回分の平均実行時間は 198(19.18%削除) である。
一回目から三回目まで実行した三回分の平均実行命令数は 136(24.44%削除) である。
(表 3.15 に示す)

- get_mpf
  ロックコードが不要となる場合 (OPT):
  一回目:(MPF_A により、get_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)
  開始時刻は 8,494 であり、完了時刻は 8,646 である。よって実行時間は 152 である。
  開始命令は 5,057 であり、完了命令は 5,117 である。よって実行命令数は 60 である。
  二回目:(MPF_B により、get_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)
  開始時刻は 8,919 であり、完了時刻は 8,981 である。よって実行時間は 62 である。
  開始命令は 5,260 であり、完了命令は 5,320 である。よって実行命令数は 60 である。
  三回目:(MPF_C により、get_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)
  開始時刻は 9,144 であり、完了時刻は 9,206 である。よって実行時間は 62 である。
  開始命令は 5,463 であり、完了命令は 5,523 である。よって実行命令数は 60 である。

  一回目から三回目まで実行した三回分の平均実行時間は 92(37.41%削除) である。
  一回目から三回目まで実行した三回分の平均実行命令数は 60(46.43%削除) である。
  (表 3.15 に示す)

- rel_mpf
  ロックコードが不要となる場合 (OPT):
  一回目:(MPF_A により、rel_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)
  開始時刻は 12,206 であり、完了時刻は 12,367 である。よって実行時間は 161 である。
  開始命令は 7,375 であり、完了命令は 7,435 である。よって実行命令数は 60 である。
  二回目:(MPF_B により、rel_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)

開始時刻は 12,533 であり、完了時刻は 12,604 である。よって実行時間は 71 である。
開始命令は 7,581 であり、完了命令は 7,641 である。よって実行命令数は 60 である。
三回目:(MPF_C により、rel_mpf の処理が始まってから、終わるまでの実行時間と実行命令数の値)
開始時刻は 12,767 であり、完了時刻は 12,838 である。よって実行時間は 71 である。
開始命令は 7,794 あり、完了命令は 7,854 である。よって実行命令数は 60 である。

一回目から三回目まで実行した三回分の平均実行時間は 101(34.42%削除) である。
一回目から三回目まで実行した三回分の平均実行命令数は 60(45.45%削除) である。
(表 3.15 に示す)

メールボックス機能について、ロックコードが必要な場合 (FULL) の実行時間と実行命令数を以下に示す。

- cre_mbx
  ロックコードが必要場合 (FULL)：
  一回目:(MBX_A の登録のための cre_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 3,578 であり、完了時刻は 3,996 である。よって実行時間は 418 である。
  開始命令は 1,812 であり、完了命令は 2,050 である。よって実行命令数は 238 である。
  二回目:(MBX_B の登録のための cre_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 4,233 であり、完了時刻は 4,501 である。よって実行時間は 268 である。
  開始命令は 2,245 であり、完了命令は 2,483 である。よって実行命令数は 238 である。
  三回目:(MBX_C の登録のための cre_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 4,738 であり、完了時刻は 4,986 である。よって実行時間は 248 である。
  開始命令は 2,678 であり、完了命令は 2,916 である。よって実行命令数は 238 である。

  一回目から三回目まで実行した三回分の平均実行時間は 311 である。
  一回目から三回目まで実行した三回分の平均実行命令数は 238 である。(表 3.14 に示す)

- snd_mbx
  ロックコードが必要場合 (FULL)：
  一回:(MBX_A により、snd_mbx の処理が始まってから、終わるまでの実行時間と実

行命令数の値である)
開始時刻は 8,746 であり、完了時刻は 8,977 である。よって実行時間は 231 である。
開始命令は 5,187 であり、完了命令は 5,308 である。よって実行命令数は 121 である。
二回目:(MBX_B により、snd_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
開始時刻は 9,123 であり、完了時刻は 9,244 である。よって実行時間は 121 である。
開始命令は 5,442 であり、完了命令は 5,563 である。よって実行命令数は 121 である。
三回目:(MBX_C により、snd_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
開始時刻は 9,399 であり、完了時刻は 9,520 である。よって実行時間は 121 である。
開始命令は 5,696 であり、完了命令は 5,817 である。よって実行命令数は 121 である。

一回目から三回目まで実行した三回分の平均実行時間は 158 である。
一回目から三回目まで実行した三回分の平均実行命令数は 121 である。(表 3.14 に示す)

- rcv_mbx
  ロックコードが必要場合 (FULL) :
  一回目:(MBX_A により、rcv_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 11,950 であり、完了時刻は 12,181 である。よって実行時間は 231 である。
  開始命令は 7,239 であり、完了命令は 7,360 である。よって実行命令数は 121 である。
  二回目:(MBX_B により、rcv_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 12,441 であり、完了時刻は 12,569 である。よって実行時間は 128 である。
  開始命令は 7,489 あり、完了命令は 7,617 である。よって実行命令数は 128 である。
  三回目:(MBX_C により、rcv_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 12,718 であり、完了時刻は 12,853 である。よって実行時間は 135 である。
  開始命令は 7,745 であり、完了命令は 7,880 である。よって実行命令数は 135 である。

  一回目から三回目まで実行した三回分の平均実行時間は 165 である。
  一回目から三回目まで実行した三回分の平均実行命令数は 128 である。(表 3.14 に示す)

メールボックス機能について、ロックコードが不要となる場合 (OPT) の実行時間と実行命令数を以下に示す。

- cre_mbx
  ロックコードが不要となる場合 (OPT)：
  一回目: (MBX_A の登録のための cre_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 3,568 であり、完了時刻は 3,929 である。よって実行時間は 361 である。
  開始命令は 1,812 であり、完了命令は 2,003 である。よって実行命令数は 191 である。
  二回目:(MBX_B の登録のための cre_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 4,166 であり、完了時刻は 4,387 である。よって実行時間は 221 である。
  開始命令は 2,198 であり、完了命令は 2,389 である。よって実行命令数は 191 である。
  三回目:(MBX_C の登録のための cre_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 4,624 であり、完了時刻は 4,825 である。よって実行時間は 201 である。
  開始命令は 2,584 であり、完了命令は 2,775 である。よって実行命令数は 191 である。

  一回目から三回目まで実行した三回分の平均実行時間は 261(16.08%削除) である。
  一回目から三回目まで実行した三回分の平均実行命令数は 191(19.75%削除) である。
  (表 3.15 に示す)

- snd_mbx
  ロックコードが不要となる場合 (OPT)：
  一回目:(MBX_A により、snd_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 8,746 であり、完了時刻は 8,919 である。よって実行時間は 173 である。
  開始命令は 5,187 であり、完了命令は 5,260 である。よって実行命令数は 73 である。
  二回目:(MBX_B により、snd_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 9,065 であり、完了時刻は 9,138 である。よって実行時間は 73 である。
  開始命令は 5,394 であり、完了命令は 5,467 である。よって実行命令数は 73 である。
  三回目:(MBX_C により、snd_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 9,293 であり、完了時刻は 9,366 である。よって実行時間は 73 である。

開始命令は 5,600 であり、完了命令は 5,673 である。よって実行命令数は 73 である。

一回目から三回目まで実行した三回分の平均実行時間は 106(32.91%削除) である。
一回目から三回目まで実行した三回分の平均実行命令数は 73(39.67%削除) である。
(表 3.15 に示す)

- rcv_mbx
  ロックコードが不要となる場合 (OPT)：
  一回目:(MBX_A により、rcv_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 11,960 であり、完了時刻は 12,111 である。よって実行時間は 151 である。
  開始命令は 7,239 であり、完了命令は 7,310 である。よって実行命令数は 71 である。
  二回目:(MBX_B により、rcv_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 12,371 であり、完了時刻は 12,449 である。よって実行時間は 78 である。
  開始命令は 7,439 であり、完了命令は 7,517 である。よって実行命令数は 78 である。
  三回目:(MBX_C により、rcv_mbx の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 12,598 であり、完了時刻は 12,683 である。よって実行時間は 85 である。
  開始命令は 7,645 であり、完了命令は 7,730 である。よって実行命令数は 85 である。

  一回目から三回目まで実行した三回分の平均実行時間は 105(36.36%削除) である。
  一回目から三回目まで実行した三回分の平均実行命令数は 78(39.06%削除) である。
  (表 3.15 に示す)

実験 6 においてセマフォ機能についてのプログラムを実行した。実行したプログラムは実験 3 のプログラムと同じものである。(sig_sem と wai_sem のロックコードの要・不要の判断は未実装のため、全て不要であると仮定する。)

セマフォ機能について、ロックコードが必要な場合 (FULL) の実行時間と実行命令数を以下に示す。

- cre_sem
  ロックコードが必要場合 (FULL)：
  一回目:(SEM_A の登録のための cre_sem の処理が始まってから、終わるまでの実行

表 3.14: 固定長メモリプール機能とメールボックス機能についての実験 5FULL

| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
|---|---|---|---|---|---|---|---|
| cre_mpf | 1 回目 | 3,217 | 3,549 | 332 | 1,623 | 1,803 | 180 |
| | 2 回目 | 4,014 | 4,216 | 202 | 2,058 | 2,238 | 180 |
| | 3 回目 | 4,519 | 4,721 | 202 | 2,491 | 2,671 | 180 |
| | 3 回分の平均実行時間:245 | | | | 3 回分の平均実行命令数:180 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| get_mpf | 1 回目 | 8,494 | 8,708 | 214 | 5,057 | 5,169 | 112 |
| | 2 回目 | 8981 | 9,095 | 114 | 5,312 | 5,424 | 112 |
| | 3 回目 | 9,258 | 9,372 | 114 | 5,567 | 5,679 | 112 |
| | 3 回分の平均実行時間:147 | | | | 3 回分の平均実行命令数:112 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| rel_mpf | 1 回目 | 12,206 | 12,427 | 221 | 7,375 | 7,485 | 110 |
| | 2 回目 | 12,593 | 12,714 | 121 | 7,631 | 7,741 | 110 |
| | 3 回目 | 12,877 | 12,998 | 121 | 7,894 | 8,004 | 110 |
| | 3 回分の平均実行時間:154 | | | | 3 回分の平均実行命令数:110 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| cre_mbx | 1 回目 | 3,578 | 3,996 | 418 | 1,812 | 2,050 | 238 |
| | 2 回目 | 4,233 | 4,501 | 268 | 2,245 | 2,483 | 238 |
| | 3 回目 | 4,738 | 4,986 | 248 | 2,678 | 2,916 | 238 |
| | 3 回分の平均実行時間:311 | | | | 3 回分の平均実行命令数:238 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| snd_mbx | 1 回目 | 8,746 | 8,977 | 231 | 5,187 | 5,308 | 121 |
| | 2 回目 | 9,123 | 9,244 | 121 | 5,442 | 5,563 | 121 |
| | 3 回目 | 9,399 | 9,520 | 121 | 5,696 | 5,817 | 121 |
| | 3 回分の平均実行時間:158 | | | | 3 回分の平均実行命令数:121 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| rcv_mbx | 1 回目 | 11,950 | 12,181 | 231 | 7,239 | 7,360 | 121 |
| | 2 回目 | 12,441 | 12,569 | 128 | 7,489 | 7,617 | 128 |
| | 3 回目 | 12,718 | 12,853 | 135 | 7,745 | 7,880 | 135 |
| | 3 回分の平均実行時間:165 | | | | 3 回分の平均実行命令数:128 | | |

表 3.15: 固定長メモリプール機能とメールボックス機能についての実験 5OPT

| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
|---|---|---|---|---|---|---|---|
| cre_mpf | 1 回目 | 3,207 | 3,485 | 278 | 1,623 | 1,759 | 136 |
| | 2 回目 | 3,950 | 4,108 | 158 | 2,014 | 2,150 | 136 |
| | 3 回目 | 4,411 | 4,569 | 158 | 2,403 | 2,539 | 136 |
| | 3 回分の平均実行時間:198 | | | | 3 回分の平均実行命令数:136 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| get_mpf | 1 回目 | 8,494 | 8,646 | 152 | 5,057 | 5,117 | 60 |
| | 2 回目 | 8,919 | 8,981 | 62 | 5,260 | 5,320 | 60 |
| | 3 回目 | 9,144 | 9,206 | 62 | 5,463 | 5,523 | 60 |
| | 3 回分の平均実行時間:92 | | | | 3 回分の平均実行命令数:60 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| rel_mpf | 1 回目 | 12,206 | 12,367 | 161 | 7,375 | 7,435 | 60 |
| | 2 回目 | 12,533 | 12,604 | 71 | 7,581 | 7,641 | 60 |
| | 3 回目 | 12,767 | 12,838 | 71 | 7,794 | 7,854 | 60 |
| | 3 回分の平均実行時間:101 | | | | 3 回分の平均実行命令数:60 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| cre_mbx | 1 回目 | 3,586 | 3,929 | 361 | 1,812 | 2,003 | 191 |
| | 2 回目 | 4,166 | 4,387 | 221 | 2,198 | 2,389 | 191 |
| | 3 回目 | 4,624 | 4,825 | 201 | 2,584 | 2,775 | 191 |
| | 3 回分の平均実行時間:261 | | | | 3 回分の平均実行命令数:191 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| snd_mbx | 1 回目 | 8,746 | 8,919 | 173 | 5,187 | 5,260 | 73 |
| | 2 回目 | 9,065 | 9,138 | 73 | 5,394 | 5,467 | 73 |
| | 3 回目 | 9,293 | 9,366 | 73 | 5,600 | 5,673 | 73 |
| | 3 回分の平均実行時間:106 | | | | 3 回分の平均実行命令数:73 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| rcv_mbx | 1 回目 | 11,960 | 12,111 | 151 | 7,239 | 7,310 | 71 |
| | 2 回目 | 12,371 | 12,449 | 78 | 7,439 | 7,517 | 78 |
| | 3 回目 | 12,598 | 12,683 | 85 | 7,645 | 7,730 | 85 |
| | 3 回分の平均実行時間:105 | | | | 3 回分の平均実行命令数:78 | | |

時間と実行命令数の値である)
開始時刻は 3,167 であり、完了時刻は 3,399 である。よって実行時間は 232 である。
開始命令は 1,563 であり、完了命令は 1,685 である。よって実行命令数は 122 である。
二回目:(SEM_B の登録のための cre_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
開始時刻は 3,146 であり、完了時刻は 3,548 である。よって実行時間は 132 である。
開始命令は 1,692 であり、完了命令は 1,814 である。よって実行命令数は 122 である。
三回目:(SEM_C の登録のための cre_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
開始時刻は 3,565 であり、完了時刻は 3,687 である。よって実行時間は 122 である。
開始命令は 1,821 であり、完了命令は 1,943 である。よって実行命令数は 122 である。

一回目から三回目まで実行した三回分の平均実行時間は 162 である。
一回目から三回目まで実行した三回分の平均実行命令数は 122 である。(表 3.16 に示す)

- sig_sem
  ロックコードが必要場合 (FULL):
  一回目:(SEM_A により、sig_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 7,546 であり、完了時刻は 7,714 である。よって実行時間は 168 である。
  開始命令は 4,325 であり、完了命令は 4,423 である。よって実行命令数は 98 である。
  二回目:(SEM_B により、sig_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 7,846 であり、完了時刻は 7,944 である。よって実行時間は 98 である。
  開始命令は 4,545 であり、完了命令は 4,643 である。よって実行命令数は 98 である。
  三回目:(SEM_C により、sig_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 8,994 であり、完了時刻は 9,092 である。よって実行時間は 98 である。
  開始命令は 5,459 であり、完了命令は 5,557 である。よって実行命令数は 98 である。

  一回目から三回目まで実行した三回分の平均実行時間は 121 である。
  一回目から三回目まで実行した三回分の平均実行命令数は 98 である。(表 3.16 に示す)

- wai_sem
  ロックコードが必要場合 (FULL):

一回目:(SEM_A により、wai_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
開始時刻は 7,185 であり、完了時刻は 7,367 である。よって実行時間は 182 である。
開始命令は 4,084 であり、完了命令は 4,186 である。よって実行命令数は 102 である。
二回目:(SEM_B により、wai_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
開始時刻は 7,707 であり、完了時刻は 7,809 である。よって実行時間は 102 である。
開始命令は 4,426 であり、完了命令は 4,528 である。よって実行命令数は 102 である。
三回目:(SEM_C により、wai_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
開始時刻は 8,602 であり、完了時刻は 8,714 である。よって実行時間は 112 である。
開始命令は 5,107 であり、完了命令は 5,209 である。よって実行命令数は 102 である。

一回目から三回目まで実行した三回分の平均実行時間は 132 である。
一回目から三回目まで実行した三回分の平均実行命令数は 102 である。(表 3.16 に示す)

セマフォ機能について、ロックコードが不要となる場合 (OPT) の実行時間と実行命令数を以下に示す。

- cre_sem
  ロックコードが不要となる場合 (OPT):
  一回目:(SEM_A の登録のための cre_sem の処理が始まってから、終わるまでの実行時間実行時間と実行命令数の値である)
  開始時刻は 3,157 であり、完了時刻は 3,329 である。よって実行時間は 172 である。
  開始命令は 1,563 であり、完了命令は 1,635 である。よって実行命令数は 72 である。
  二回目:(SEM_B の登録のための cre_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 3,346 であり、完了時刻は 3,428 である。よって実行時間は 82 である。
  開始命令は 1,642 であり、完了命令は 1,714 である。よって実行命令数は 72 である。
  三回目:(SEM_C の登録のための cre_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 3,445 であり、完了時刻は 3,517 である。よって実行時間は 72 である。
  開始命令は 1,721 であり、完了命令は 1,793 である。よって実行命令数は 72 である。

  一回目から三回目まで実行した三回分の平均実行時間は 109(32.72%削除) である。
  一回目から三回目まで実行した三回分の平均実行命令数は 72(40.98%削除) である。

(表 3.17 に示す)

- sig_sem
  ロックコードが不要となる場合 (OPT)：
  一回目:(SEM_A により、sig_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 7,536 であり、完了時刻は 7,657 である。実行時間は 121 である。
  開始命令は 4,325 であり、完了命令は 4,376 である。よって実行命令数は 51 である。
  二回目:(SEM_B により、sig_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 7,789 であり、完了時刻は 7,840 である。よって実行時間は 51 である。
  開始命令は 4,498 であり、完了命令は 4,549 である。よって実行命令数は 51 である。
  三回目:(SEM_C により、sig_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 8,900 であり、完了時刻は 8,951 である。よって実行時間は 51 である。
  開始命令は 5,365 であり、完了命令は 5,416 である。よって実行命令数は 51 である。

  一回目から三回目まで実行した三回分の平均実行時間は 74(38.84%削除) である。
  一回目から三回目まで実行した三回分の平均実行命令数は 51(47.96%削除) である。
  (表 3.17 に示す)

- wai_sem
  ロックコードが不要となる場合 (OPT)：
  一回目:(SEM_A により、wai_sem の処理が始まるから、終わるまで実行時間と実行命令数の値である)
  開始時刻は 7,155 であり、完了時刻は 7277 である。よって実行時間は 122 である。
  開始命令は 4,084 であり、完了命令は 4,136 である。よって実行命令数は 52 である。
  二回目:(SEM_B により、wai_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 7,617 であり、完了時刻は 7,669 である。よって実行時間は 52 である。
  開始命令は 4,376 であり、完了命令は 4,428 である。よって実行命令数は 52 である。
  三回目:(SEM_C により、wai_sem の処理が始まってから、終わるまでの実行時間と実行命令数の値である)
  開始時刻は 8,482 であり、完了時刻は 8,544 である。よって実行時間は 62 である。
  開始命令は 5,007 であり、完了命令は 5,059 である。よって実行命令数は 52 である。

  一回目から三回目まで実行した三回分の実行した三回の平均実行時間は 79(40.15%削

表 3.16: セマフォ機能についての実験 6FULL

| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
|---|---|---|---|---|---|---|---|
| cre_sem | 1 回目 | 3,167 | 3,399 | 232 | 1,563 | 1,685 | 122 |
| | 2 回目 | 3,146 | 3,548 | 132 | 1,692 | 1,814 | 122 |
| | 3 回目 | 3,565 | 3,687 | 122 | 1,821 | 1,943 | 122 |
| | 3 回分の平均実行時間:162 | | | | 3 回分の平均実行命令数:122 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| sig_sem | 1 回目 | 7,546 | 7,714 | 168 | 4,325 | 4,423 | 98 |
| | 2 回目 | 7,846 | 7,944 | 98 | 4,545 | 4,643 | 98 |
| | 3 回目 | 8,994 | 9,092 | 98 | 5,459 | 5,557 | 98 |
| | 3 回分の平均実行時間:121 | | | | 3 回分の平均実行命令数:98 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| wai_sem | 1 回目 | 7,185 | 7,367 | 182 | 4,084 | 4,186 | 102 |
| | 2 回目 | 7,707 | 7,809 | 102 | 4,426 | 4,528 | 102 |
| | 3 回目 | 8,602 | 8,714 | 112 | 5,107 | 5,209 | 102 |
| | 3 回分の平均実行時間:132 | | | | 3 回分の平均実行命令数:102 | | |

除) である。
一回目から三回目まで実行した三回分の平均実行命令数は 52(49.02%削除) である。
(表 3.17 に示す)

表 3.17: セマフォ機能についての実験 6OPT

| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
|---|---|---|---|---|---|---|---|
| cre_sem | 1 回目 | 3,157 | 3,329 | 172 | 1,563 | 1,635 | 72 |
| | 2 回目 | 3,346 | 3,428 | 82 | 1,642 | 1,714 | 72 |
| | 3 回目 | 3,445 | 3,517 | 72 | 1,721 | 1,793 | 72 |
| | 3 回分の平均実行時間:109 | | | | 3 回分の平均実行命令数:72 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| sig_sem | 1 回目 | 7,536 | 7,657 | 121 | 4,325 | 4,376 | 51 |
| | 2 回目 | 7,789 | 7,840 | 51 | 4,498 | 4,549 | 51 |
| | 3 回目 | 8,900 | 8,951 | 51 | 5,365 | 5,416 | 51 |
| | 3 回分の平均実行時間:74 | | | | 3 回分の平均実行命令数:51 | | |
| | 回数 | 開始時刻 | 完了時刻 | 実行時間 | 開始命令 | 完了命令 | 実行命令数 |
| wai_sem | 1 回目 | 7,155 | 7,277 | 122 | 4,084 | 4,136 | 52 |
| | 2 回目 | 7,617 | 7,669 | 52 | 4,376 | 4,428 | 52 |
| | 3 回目 | 8,482 | 8,544 | 62 | 5,007 | 5,059 | 52 |
| | 3 回分の平均実行時間:79 | | | | 3 回分の平均実行命令数:52 | | |

# 第4章　まとめ

本研究では、アプリケーションコードの解析、およびシステムコール内の不必要なコードの除去を自動的に行う手法を提案した。本手法を利用することにより、実行オーバヘッドとバイナリサイズが削除されることを評価により示した。アプリケーションタスク群のソースコードを解析することにより、利用するシステムコール内の不必要なエラーコードとロックコードを検出する方法を検討し、具体的な検出方式を提案した。エラーチェックは主にシステムコールのオペランドと記号定数の解析により必要性を決定する。また、各 RTOS 機能の利用がタスク間で競合するかどうかを解析し、ロック機能の必要性を決定する。実装するシステムはアプリケーションのソースコードを入力し、不必要なコードを除去したシステムコール関数を出力するものである。実装したシステムに作成したタスクセットを適用し、アプリケーションの実行性能を評価した。評価にはクロックサイクルベールのＣＰＵシミュレータを使用した。実験によって対象システムコール内の各エラーチェックコードの削除によってバイナリサイズを平均 16.88%削減した。実行時間は平均 10.46%削減した。実行命令数は平均 9.46%削減された。対象システムコール内のロックコードの削除によってバイナリコードのサイズを平均 6.84%削減した。実行時間を平均 28.51%削減した。実行命令数を平均 35.21%削減した。

# 謝辞

本研究を進めるにあたり、日頃より丁寧な指導を賜りました田中　清史准教授に心より感謝申し上げます。また、日頃より有益な助言を頂いた田中研究室と金子研究室の学生の皆様にご礼申しあげます。その他、あたたかく応援を下さった両親と多くの友人たちに深く感謝申しあげます。

# 参考文献


[1] $\mu$ ITRON4.0 仕様, （社）トロン協会 ITRON 仕様検討グループ

[2] The SPARC Architecture Manual Version 8.SPARC International Inc., Prentice Hall ,1992

[3] 請園智玲　ハードウェア解析システムによる実行コードの動的最適化に関する研究 北陸先端科学技術大学院大学　博士論文　Mar.2010

[4] 宮内哲夫　FPGA 用ソフトプロセッサのための自動最適化コンフィギュレータの構築 北陸先端科学技術大学院大学　修士論文　Mar.2015

[5] Y. Eustache, J. P. Diguet and M. Elkhodary, “RTOS extensions for dynamic hardware / software monitoring and configuration management”, Proceedings 20th IEEE International Parallel & Distributed Processing Symposium, 2006

[6] C. Boke, M. Gotz, T. Heimfarth, D. El Kebbe, F. J. Rammig and S. Rips, “(Re-) configurable real-time operating systems and their applications”, Object-Oriented Real-Time Dependable Systems, 2003. (WORDS 2003). Proceedings of the Eighth International Workshop on, 2003

[7] M. Sindhwani and T. Srikanthan, “Framework for Automated Application-Specific Optimization of Embedded Real-Time Operating Systems”, 2005 5th International Conference on Information Communications & Signal Processing, Bangkok, 2005

[8] F. J. Rammig, M. Gotz, T. Heimfarth, P. Janacik and S. Oberthur, “Real-time operating systems for self-coordinating embedded systems”, Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06), Gyeongju, 2006