社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004－OS－97 (4)
2004／8／1

# 動的リンクライブラリの実行中入替えを可能にする基本機構

吉原 隼人† 谷口 秀夫††

†岡山大学大学院自然科学研究科
††岡山大学工学部

24時間無停止でサービスを提供するシステムの実現のためには，動作中のソフトウェアを停止させることなく変更できる必要がある．そこで，我々は，実行中のプログラムの一部分を入替える制御法を提案した．また，本制御法に対して動的リンク機能を利用した状態把握メカニズムを適用することにより，状態把握を意識した応用プログラムのソースコードやオブジェクトファイルや実行可能ファイルの作成が必要なくなることを示した．本入替え法に対し，入替える対象となるプログラム部分を動的リンクライブラリとし，入替え処理に動的リンク機能を適用することにより，入替え内容の作成における利便性が向上する．本稿では，動的リンク機能を利用した動的リンクライブラリの入替え法について述べる．

# Basic Mechanism for Exchanging Dynamic Link Libraries during Excution

Hayato YOSHIHARA † Hideo TANIGUCHI ††

†The Graduate School of Natural Sience and Technology,Okayama University
††Faculty of Enginering,Okayama University

To allow continuous services of computer, it is essential to exchange parts of the program without down time. So, we have purposed the mechanism for exchanging parts of program. And, we have applied the method of grasping the execution states of program using dynamic linking to our purposed mechanism. By using this mechanism, it is not need to change application codes, object files, and executable files. If exchanged targets are dynamic link library, we can use dynamic linker for exchanging. And it makes possible to exchange parts of program easily. In this paper, we suggest a mechanism of exchanging executed dynamic link library.

## 1 はじめに

近年，計算機の普及が進み，さまざまなサービスに計算機が利用されるようになった．これに伴い，計算機によるサービスを24時間無停止で提供することが強く望まれている．この場合，計算機に対して，より高い信頼性が必要となる．しかし，多くの場合，計算機によるサービスには，ハードウェア面あるいはソフトウェア面でさまざまな障害が起こり得る．このため，不具合の修正によって，サービスを一時停止させるような場面も多々ある．もし，動作中のソフトウェアを停止させることなく，その内容を変更することができれば，高い信頼性を有する計算機を提供することができる．

このような背景から，我々は，プロセスとして走行している応用プログラム（以降，APと略す）を停止することなく，その一部分を変更する方法を提案した [1]．この入替え法は，次の二つの大きな特徴を持つ．一つは，サービスプログラムが，入替えの契機を意識しなくてもよい点である．もう一つは，入替え対象のプログラム部分が，入替え対象でない別プログラム部分を呼び出している場合にも，プログラム入替えを可能にしている点である．さらに，複数のプロセス間で共有されたプログラム部分の入替え法も示している [2]．

プログラム部分の入替えに際しては，入替えを行う段階で入替え対象となるプログラム部分を実行し

ているプロセスが存在しない必要がある．このため，プログラム部分の実行状態を把握する必要がある．文献 [1] [2] では，プログラム部分の呼出と復帰を検出するためのシステムコール（以降，状態把握用システムコールと呼ぶ）を AP のソースコードに埋め込む方式を採用している．一方，文献 [3] では，動的リンク機能を利用してプログラム部分の実行状態の把握を行う方式を提案している．この方式は，状態把握用システムコールを AP のソースコードに埋め込む必要がないという特徴を持つ．さらに，動的リンク機能を利用した入替えを可能にすることにより，入替え内容に関して，高級言語で記述，入替えの前後で入替えるプログラム部分の大きさを一致させることが不要，という二つの利点があることを示唆している．しかし，具体的な入替え方式については明らかにされていない．

そこで，本稿では，プログラム部分の単位として動的リンクライブラリに着目し，実行中の AP の一部分として動的リンクライブラリを入替える方式について述べる．

## 2 入替えの基本機構

### 2.1 プログラム実行状態の分類

図 1 に示すように，プログラム部分に対するプロセスの実行状態は三つの状態に分類できる [1]．

(1) 未使用：実行する前または実行を終えた状態
(2) 走行中：実行中の状態
(3) 呼出中：他のプログラム部分を呼び出している状態

このうち，走行中状態のみは，内部変数値などが過度的な状態であるため，入替えを行うことはできない．

### 2.2 入替え可能条件

入替えの前後で整合性を保つため，入替えられるプログラム部分は，その実行状態に応じて，以下に述べる条件を必要とする [1]．

(条件 1) 呼出と返却値のインターフェースの一致
(条件 2) 処理の矛盾の回避

上記 2 条件は，入替え内容の必須条件である．未使用状態で入替えを実行する場合には，この 2 条件以外に必要な条件はない．しかし，呼出中状態でも入替えを実行する場合には，この 2 条件に加えて，下記 4 条件を加える必要がある．

(条件 3) 戻り先のアドレスを変更しないこと
(条件 4) 静的リンクの場合，メモリ上の外部変数の参照アドレスが同じこと
(条件 5) 外部変数の値が入替えの前後で同じであることの保証が必要か否か

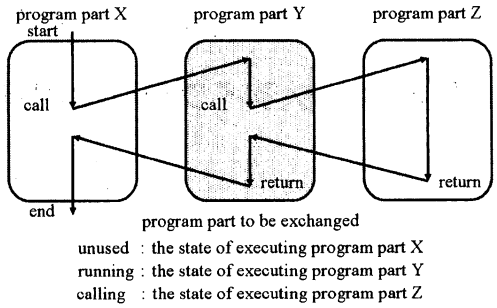


図 1 実行状態と入替え対象プログラム部分の関係

(条件 6) アドレス渡しによる外部変数や内部変数の参照や更新がどのように行なわれているか

本稿では，入替え内容の大きさに制限がない入替えを目指している．この場合，(条件 3) の制約があることは望ましくない．なぜならば，入替え内容が大きくなることにより，入替えるプログラム部分をロードするアドレスが，入替えの前後で異なる可能性がある．つまり，明らかに戻り先のアドレスは異なる場合である．したがって，本稿では未使用状態での入替えのみを対象とすることにする．これにより，入替えに必要な条件は，(条件 1) と (条件 2) の 2 条件のみとなる．

### 2.3 プログラム実行状態の把握

プログラム部分の実行状態を把握する手順は，以下の二つのステップからなる [1]．

(1) 状態遷移の検出：プログラム部分間の呼出と復帰の検出
(2) 実行状態の遷移：プログラム部分の実行状態の遷移

図 1 に示したように，プログラム部分の実行状態は，別プログラム部分から当該プログラム部分の呼出と復帰時，及び当該プログラム部分から別プログラム部分の呼出と復帰時に遷移する．したがって，第 1 ステップとして，プログラム部分間の呼出と復帰を検出する必要がある．そして，第 2 ステップでは，第 1 ステップで検出したプログラム部分間の呼出と復帰を契機として，当該プログラム部分の実行状態を遷移させる．

各ステップの処理は次のように分担している．状態遷移の検出処理については，オペレーティングシステム（以降，OS と略す）外で行っている．これは，OS 内でプログラム部分の実行位置を把握することは，命令トレースの処理を必要とし，処理負荷が大きいためである．一方，実行状態の遷移処理に

ついては，OS 内で行っている．これは，実際に入替え処理を行う OS 内で実行状態を管理し，入替え可否を判別することにより，入替え可能状態になった後，直ちに入替え処理を行うことができる．したがって，プログラム部分の呼出と復帰の検出処理において状態把握用のシステムコールを発行することにより，OS 外から OS 内へ実行状態の遷移契機を通知し，システムコールの処理内で実行状態の遷移処理を行い，実行状態を把握する．

### 2.4 入替えの制御法

入替え対象となるプログラム部分を入替える処理は，以下の三つの処理からなる [1]．

(1) 入替え可能状態への移行処理
(2) 入替え可能状態の保持処理
(3) 入替え処理

入替え可能状態への移行処理とは，入替え不可能状態にある各プロセスを入替え可能状態へ遷移させる処理である．この処理は，プログラムの実行そのものであり，特別な処理は行わない．入替え可能状態の保持処理とは，入替え可能状態にある各プロセスの状態を保持させる処理である．具体的には，入替え可能状態から入替え不可能状態へ遷移しようとするプロセスの走行を遷移の直前で停止させる．

入替え可能状態の保持処理を行うことにより，有限時間内で入替え対象となるプログラム部分の実行状態を入替え可能状態へ遷移させることを保証できる．ただし，入替え不可能状態であるプログラム部分内で一時停止中のプロセスが有限時間内にプログラム実行を再開する保証がないときは例外的な場合であり，入替えを行うことはできない．なお，この処理は，サービスへ与える影響が大きいため，その対策として，プロセスの停止に対して制御時間（以降，タイムアウト時間とする）を設ける [2]．タイムアウト時間による制御の様子を図 2 に示す．入替え可能状態の保持処理を行うことにより，入替え要求後に入替え対象となるプログラム部分へと突入しようとするプロセスは，突入の直前で走行を停止させられる．しかし，当該プロセスの停止時間がタイムアウト時間と等しくなると，当該プロセスは再び走行を開始し，入替え対象となるプログラム部分への突入を許可する．

入替え処理とは，実際に入替え対象となるプログラム部分を入替える処理である．

## 3 動的リンク機能の利用

動的リンク機能を利用する場合，一つの AP は複数のロードセグメント（以降，LS と略す）から構成される．各 LS はプログラム実行時に動的ローダによってロードされ，各 LS 間のシンボルによる参照は実行時の動的リンカによって解決される．

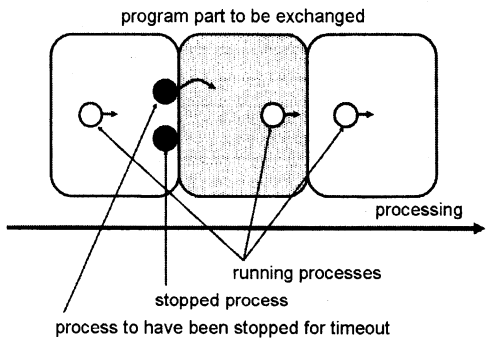


図 2 タイムアウト時間による制御

### 3.1 状態遷移の検出

文献 [3] では，動的リンカにおいて各 LS 間の呼出と復帰を検出する方法を示している．この方法について，LS 間で関数シンボル function() の参照が行われたとする場合を例にとって説明する（文献 [3] の図 4，図 5 を参照）．なお，function() の参照を行った LS は，function() のアドレスを解決するための情報（以降，refobj-symbol 情報と呼ぶ）を持っている．

動的リンカ内には，シンボルの参照と定義の対応を管理する表（以降，refobj-symbol defobj-location 表と呼ぶ）を用意してあり，この表から function() に対応するエントリを探索し，function() のアドレスが登録されているか否かを調べる．既に，function() のアドレスが登録済みの場合，このエントリから function() のアドレスを解決する．また，function() のアドレスが未登録の場合，refobj-symbol 情報をもとに，ロードされている全 LS を探索してその定義を見つけ出し，アドレス解決を行う．また，refobj-symbol defobj-location 表に解決したアドレスを登録する．function() のアドレス解決後，call 命令により function() へと制御を移す．その後，function() から制御が戻ると，動的リンカを経由して，function() の呼出元のアドレスへと制御を移す．

以上の処理手順により，動的リンカにおいて各 LS 間の呼出と復帰を検出できる．すなわち，それを契機として LS の呼出と復帰を通知する状態把握用システムコールを発行することにより，OS において各プロセスによる当該 LS の実行状態を遷移させることができる．

表 1 入替え処理に必要なフラグ

| 通番 | 名前 | 管理単位 | 意味 | OFF → ON の契機 | ON → OFF の契機 |
|---|---|---|---|---|---|
| (1) | rflag | LS | 入替え要求の有無 | 入替え要求発生直後 | LS 複写処理完了直後 |
| (2) | dflag | LS | LS 複写処理中であるか否か | LS 複写処理開始直前 | LS 複写処理完了直後 |
| (3) | mflag | 当該 LS をロードした各プロセス | LS 情報初期化処理を行う必要があるか否か | rflag が OFF になった直後 | LS 情報初期化処理直前 |

### 3.2 入替え処理

LS を入替える処理は，以下の二つの処理によって構成する．

(1) 外部記憶装置（以降，DK と略す）上での LS 複写処理

(2) 各プロセス空間上での LS 情報初期化処理

DK 上での LS 複写処理（以降，LS 複写処理と略す）を行う前準備として，入替え対象である LS（以降，旧 LS と略す）に対する入替え後の内容が記述されている LS（以降，新 LS と略す）を用意する必要がある．LS 複写処理とは，DK 上で旧 LS の内容を新 LS の内容に書き換える処理である．また，LS 複写処理を行う際には，旧 LS を実行しているプロセスが一つも存在していないことが必須となる．これは，LS 複写処理途中に当該 LS 内でオンデマンドページングが発生した場合に，処理の整合性がとれなくなるためである．したがって，LS 複写処理は，入替え要求が発生し，入替え対象となる LS が入替え可能状態になった直後に行う．

各プロセス空間上での LS 情報初期化処理（以降，LS 情報初期化処理と略す）とは，当該 LS をロードした各プロセスの空間上にある当該 LS の情報を初期化する処理である．具体的には，以下の三つの処理からなる．

(1) 旧 LS のアンロード

(2) LS 複写処理によって新 LS の内容に書き換えられた LS のロード

(3) 既に解決済みの旧 LS に関するアドレス情報のクリア

LS 情報初期化処理は，各プロセス空間に対する処理であるため，当該 LS をロードした各プロセス毎に行う必要がある．LS 情報初期化処理を行う契機は，LS 複写処理が完了した後である．ただし，LS 複写処理が完了した直後に行うのではなく，LS 複写処理の完了後に初めて当該プロセスが当該 LS 内へ処理を移すときを契機として行う．これは，今後利用されることのない LS に対して，無駄に LS 情報初期化処理を行わないようにするためである．

LS 複写処理と LS 情報初期化処理による入替え処理を実現するために，表 1 に示す三つのフラグ rflag，dflag，mflag による制御を行う．rflag は，各

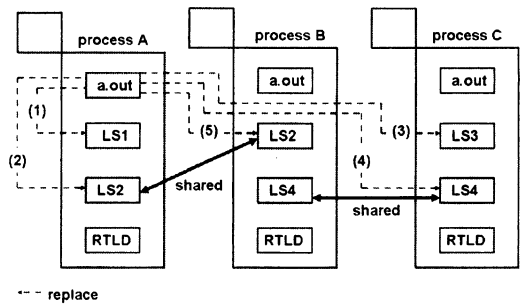


図 3 入替えの種類

LS 毎に管理し，当該 LS に対して入替え要求があるか否かを管理する．なお，LS 複写処理は，rflag が ON であり，かつ入替え可能状態の LS に対して行うことになる．LS 複写処理の完了後，rflag は OFF になる．dflag は，各 LS 毎に管理し，当該 LS に対して，LS 複写処理を行なっているか否かを管理するフラグである．当該 LS に対して LS 複写処理を行っている間のみ，dflag は ON になる．dflag は，タイムアウト時間によって待ちを解除されたプロセスが，LS 複写処理中の LS 内に処理を移さないようにする役割を持つ．mflag は，当該 LS をロードした各プロセス毎に管理し，各プロセスが当該 LS に対して，LS 情報初期化処理を行うべきか否かを管理する．LS 複写処理完了後，当該 LS をロードした全プロセスの mflag は ON になる．その後，LS 情報初期化処理が完了したプロセスは，対応するプロセス管理表の mflag を OFF にする．

## 4 実現方式

### 4.1 想定する入替え

LS の入替えの種類として，図 3 に示す五つが挙げられる．

(1) 自プロセスがロードした LS に対して入替え要求を行う．このとき，当該 LS は非共有である．

(2) 自プロセスがロードした LS に対して入替え要求を行う．このとき，当該 LS はプロセス間で共有されている．

(3) 他プロセスがロードしたLSに対して入替え要求を行う．このとき，当該LSは非共有である．

(4) 他プロセスがロードしたLSに対して入替え要求を行う．このとき，当該LSは自プロセス以外のプロセス間で共有されている．

(5) 他プロセスがロードしたLSに対して入替え要求を行う．このとき，当該LSは自プロセスと共有されている．

なお，(2) と (5) に関しては，入替えの対象となるLSがロードされているプロセス空間が異なることを除けば，全く同じ処理である．また，本稿では，異なるプロセス間のLSを区別しない．そのため，(2) と (5) は特に区別はしない．

本稿では，(3) と (4) の入替えを想定する．すなわち，自プロセスがロードしているLSについては，入替えを行わないものとする．自プロセスがロードしていないLSであれば，複数のプロセス間で共有されていても入替えを行うことができる．

### 4.2 実行状態の管理方式

2.2節で述べたように，LSが入替え可能状態となるのは未使用状態のときのみである．すなわち，入替え可能状態とは，当該LSに対して，関連プロセスが突入した回数と脱出した回数の差分が0の状態である．したがって，この差分を管理する計数フラグを用意し，これにより当該LSの実行状態を管理する．ただし，この計数フラグだけでは，LSがプロセス間で共有されていた場合に，実行状態を正しく把握することができない．このため，各プロセス単位でも当該LSの実行状態を管理する計数フラグを用意する．

### 4.3 プログラム部分の管理法

プログラム状態の管理法は，以下に示す方針に従って実現する．

(1) 複数のLSに対して，同時に入替えを可能にするため，LS単位で実行状態を管理する．

(2) LSがプロセスで共有されていても，正しい実行状態を把握するため，実行状態をプロセス単位で把握する．

上記の方針により，LS，プロセスの単位で管理表を用意し，状態管理を行う．LSに対応する管理表をモジュール管理表，プロセスに対応する管理表をプロセス管理表と呼ぶ．モジュール管理表とプロセス管理表の関係は，図4に示す通りである．モジュール管理表は，管理対象となるLSが複数存在する場合，リスト構造を形成する．これにより，管理対象となるLSの数に制限が生じない．また，モジュール管理表は，対応するLSをロードした全プロセスに対応するプロセス管理表をリスト構造の形式で持つ．このため，異なるモジュール管理表間で同一のプロセスに対応するプロセス管理表が存在することもあり得る．

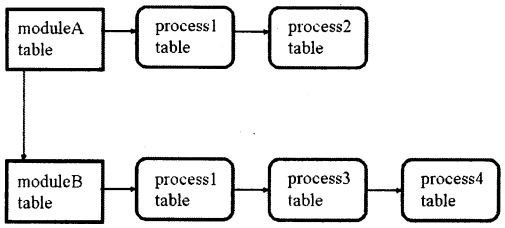


図4 各管理表の関係

表2 モジュール管理表の詳細

| | | | |
|---|---|---|---|
| 機能 | | LS対応で状態把握を行う | |
| 管理単位 | | LS | |
| 作成契機 | | 当該LSの状態把握を開始するとき | |
| 消去契機 | | 当該LSの状態把握を終了するとき | |
| 内容 | 項目 | 役割 | 対応する値 |
| | モジュールID | LSの区別 | ファイル識別子 |
| | フラグ情報 | 入替え要求の管理 | rflag |
| | | LS複写処理中かの管理 | dflag |
| | 利用状況 | 入替え可能状態の検出 | プロセスの突入/脱出回数の差分 |
| | ポインタ情報 | モジュール管理表間でキューを形成 | モジュール管理表の構造体のアドレス |
| | | 当該LSをロードした全プロセスの管理 | プロセス管理表の構造体のアドレス |

表3 プロセス管理表の詳細

| | | | |
|---|---|---|---|
| 機能 | | プロセス対応でLSに対する状態把握を行う | |
| 管理単位 | | プロセス | |
| 作成契機 | | 該当するLSに対する状態把握を開始するとき | |
| 消去契機 | | 該当するLSに対する状態把握を終了するとき | |
| 内容 | 項目 | 役割 | 対応する値 |
| | プロセスID | プロセスの区別 | プロセス識別子 |
| | フラグ情報 | LS情報初期化処理の管理 | mflag |
| | 利用状況 | LSに対する状態の管理 | 当該LSへの突入/脱出回数の差分 |
| | ポインタ情報 | プロセス管理表間でキューを形成 | プロセス管理表の構造体のアドレス |

モジュール管理表とプロセス管理表の内容を表2と表3に示す．モジュール管理表の識別子には，各管理表がLS(ファイル) に対応することから，各ファイルに固有な値（以降，ファイル識別子と呼ぶ）を採用する．また，プロセス管理表の識別子には，各管理表がプロセスに対応することから，各プロセスに固有な値（以降，プロセス識別子と呼ぶ）を採用する．

### 4.4 制御用システムコール

動的リンク機能を利用した入替え処理を実現するために，五つのシステムコールを用意する．各機能

表 4 入替えを制御する機能

| 通番 | システムコール | 形式 | 機能 |
|---|---|---|---|
| (1) | LS 状態把握開始 | syscall_open(file_id) | プロセスが file_id で指す LS の状態把握を開始する. |
| (2) | LS 状態把握終了 | syscall_close(file_id) | プロセスが file_id で指す LS の状態把握を終了する. |
| (3) | 入替え要求 | syscall_request(old_ls_name, new_ls_name) | 旧 LS に対する入替え要求を行う. 旧 LS が入替え可能状態になると, LS 複写処理を行う. |
| (4) | LS 突入 | syscall_enter(file_id) | プロセスが file_id で指す LS に対する状態遷移を OS に通知する. また, LS 情報初期化処理を行う必要がある場合, その旨を戻り値によって発行元に通知する. |
| (5) | LS 脱出 | syscall_leave(file_id) | プロセスが file_id で指す LS に対する状態遷移を OS に通知する. |

を表 4 に示し, 以降に説明する. なお, 入替え要求システムコール, LS 突入システムコール, LS 脱出システムコールについては, 3.2 節で述べたフラグに対する処理を含むため, それぞれ図 5 , 図 6 , 図 7 に処理の流れを示してある.

＜ LS 状態把握開始システムコール＞

本システムコールは, プロセスに対して, 指定する LS の状態把握処理の開始を宣言する. 例えば, AP を実行するためにプロセスがロードした全 LS に対して状態把握を開始したい場合, 当該 AP の処理の開始直前を契機として, 対応する全 LS に対して動的リンカが本システムコールを発行するようにすればよい.

本システムコールでは, まず指定する LS に対応するモジュール管理表が既に作成されているかを調べ, 作成されていない場合には作成する. その後, 当該モジュール管理表に, 当該プロセス管理表を追加する.

＜ LS 状態把握終了システムコール＞

本システムコールは, プロセスに対して, 指定した LS の状態把握処理の終了を宣言する. 例えば, AP を実行するためにプロセスがロードした全 LS に対して状態把握を終了したい場合, 当該 AP の処理の終了直後を契機として, 対応する全 LS に対して動的リンカが本システムコールを発行するようにすればよい.

本システムコールでは, まず指定する LS に対応する当該モジュール管理表から当該プロセス管理表を消去する. 次に, 当該モジュール管理表が所有するプロセス管理表の有無を確認する. プロセス管理表が一つもない場合には, 当該 LS をロードしているプロセスが存在しないということなので, 当該モジュール管理表を消去する.

＜入替え要求システムコール＞

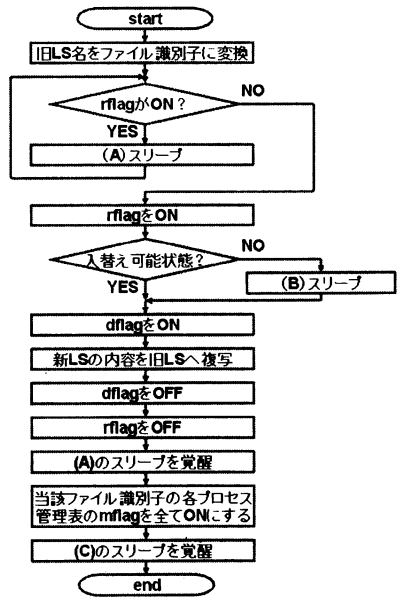


図 5 入替え要求システムコールの処理の流れ

本システムコールでは, 入替え要求を受けた LS に対応するモジュール管理表を判別するために, まず旧 LS 名をファイル識別子に変換する. 次に, 当該 LS に対して既に他の入替え要求が行われているかを調べる. 他の入替え要求が行われている場合には, その入替え要求が解除されるまで (A) として待機する. 当該 LS に対する入替え要求が他に行われていなければ, 入替え要求を受けている LS に対応する rflag を ON にすることで, 当該 LS に対して入替え要求を行う. その後, 当該 LS が入替え可能状態になるまで (B) として待機する. 当該 LS が入替え可能状態になると, (B) の待ちは解除され,

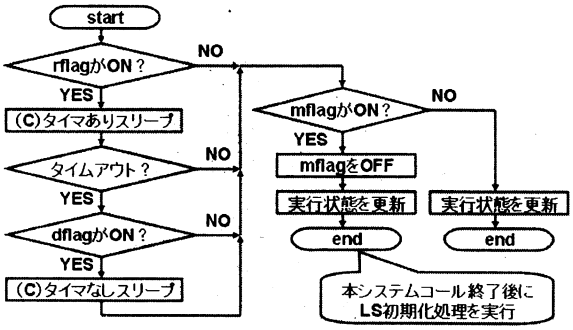


図 6 LS 突入システムコールの処理の流れ

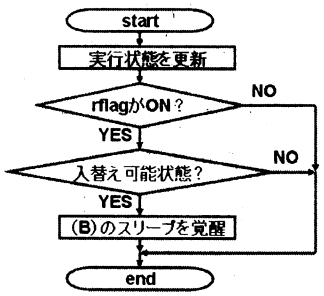


図 7 LS 脱出システムコールの処理の流れ

LS 複写処理を行う．LS 複写処理を行っている間は，当該 LS に対応する dflag を ON にし，当該 LS へプロセスが突入しないようにする．LS 複写処理が完了すると，rflag を OFF にすることで，入替え要求を解除する．このとき，(A) として待機しているプロセスがあれば，その待ちを解除する．入替え要求を解除した後，当該 LS に対応するモジュール管理表が所有する全てのプロセス管理表の mflag を ON にすることにより，当該 LS をロードした全プロセスに対して，LS 情報初期化処理を行うように通知する．また，入替え可能状態の保持処理によって図 6 の (C) として待機しているプロセスが存在する場合にはその待ちを解除する．

＜ LS 突入システムコール＞

本システムコールは，3.1 節で述べた方式によって LS 間の呼出を検出し，それを契機として動的リンカが発行する．本システムコールは，まず当該 LS に対して入替え要求があるかを確認する．入替え要求がある場合，入替え可能状態の保持処理により (C) として待機する．この待機については，タイムアウト時間によって解除されることがある．このとき，LS 複写処理の途中で待ちが解除される可能性がある．そこで，タイムアウトした場合には，dflag をチェックし，当該 LS が LS 複写処理中である場合には，LS 複写処理が完了するまで待機する．また，本システムコールは mflag の確認を行う．mflag が ON である場合には，mflag を OFF にした後に実行状態の更新を行い，本システムコールを終了する．このとき，戻り値によって当該 LS に対して LS 情報初期化処理を行うべきであることを本システムコールの発行元に通知する．また，mflag が OFF の場合には，実行状態の更新を行った後，本システムコールを終了する．

＜ LS 脱出システムコール＞

本システムコールは，3.1 節の方式によって LS 間の復帰を検出し，それを契機として動的リンカが発行する．本システムコールは，まず当該 LS の状態更新を行う．その後，当該 LS に対して，入替え要求があり，かつ入替え可能状態である場合には，入替え要求を行っているプロセスを覚醒させ，本システムコールを終了する．そうでない場合は，そのまま本システムコールを終了する．

### 4.5 LS 突入時の処理

4.4 節＜ LS 突入システムコール＞で述べたように，LS 情報初期化処理を行うべきか否かは，LS 突入システムコールの戻り値によって判定する．このため，LS 情報初期化処理は，LS 突入時の処理の中で行うことになる．3.1 節でも述べたように LS 突入時の処理の流れは，LS の遷移の際に参照された関数シンボルのアドレスが解決済みが否かによって異なる．以降では，これら二つの場合に分けて，LS 情報初期化処理を行う場合の LS 突入時の処理の流れを示す．

#### 4.5.1 関数シンボルのアドレスが解決済みの場合

関数シンボルのアドレスが既に解決済みの場合の処理の流れを図 8 に示す．3.1 節で述べた refobj-symbol defobj-location 表には，当該関数シンボルが定義されている LS の情報も記録してある．したがって，関数シンボルのアドレスが既に解決済みであれば，この表を参照することにより，当該関数シンボルの定義されているアドレスだけでなく，LS に関する情報を取得することができる．これにより，当該 LS を指定して LS 突入システムコールを発行することができる．LS 突入システムコールの処理が終了した後，その戻り値によっては当該 LS に対して，LS 情報初期化処理を行うことになる．また，LS 情報初期化処理を行った後には，当該関数シンボルのアドレスを再解決し，refobj-symbol defobj-location 表から得たアドレスを修正する必要がある．

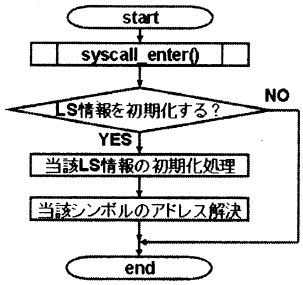


図 8 アドレス解決済み時の LS 突入処理の流れ

#### 4.5.2 関数シンボルのアドレスが未解決の場合

関数シンボルのアドレスが未解決の場合の処理の流れを図 9 に示す．この場合には，当該関数シンボルの定義を当該 AP がロードした全 LS から探索し，そのアドレスを解決する必要がある．このとき，当該 LS に対して LS 複写処理が行われていた場合には，その内容が書き換わっているため，当該 LS を探索するときに，不具合が発生する可能性がある．このため，各 LS において当該関数シンボルの定義を探索する前に LS 突入システムコールを発行し，当該 LS に対して LS 情報初期化処理を行うべきか否かを判定する．そして，LS 情報初期化処理を行うべきであれば，ただちに LS 情報初期化処理を行う．その後，当該関数シンボルの定義を探索し，そのアドレス解決を試みる．アドレスが解決しなかった場合，LS 脱出システムコールを発行してから，同様の処理を当該 AP がロードした全 LS に対して繰り返す．当該関数シンボルのアドレスが解決した時点で，本処理を終了する．

ここで示した方式では，当該関数シンボルのアドレスが解決するまでの間，LS 突入，脱出システムコールを繰り返し発行することになる．これにより，シンボルのアドレス解決時のオーバヘッドが大きくなってしまうという問題がある．しかし，アドレス解決処理は，3.1 節で述べた refobj-symbol defobj-location 表における当該エントリに，解決されたアドレスが登録されていないときにのみ行われる処理である．すなわち，アドレス解決処理は，当該関数シンボルが初めて参照されたとき，あるいは LS 情報初期化処理を行ってから初めて当該関数シンボルが参照されたとき以外では行われることがない．したがって，アドレス解決処理処理によるオーバヘッドは，アドレス解決時の処理のみから考えると大きいものであるが，AP の処理全体からは，それほど大きいものではないと考えることができる．

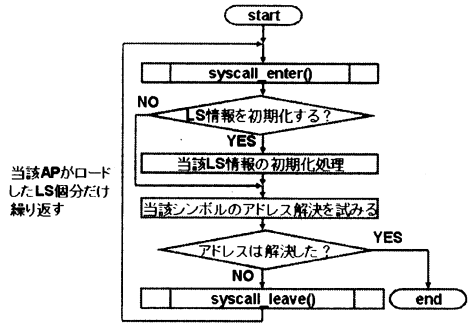


図 9 アドレス未解決時の LS 突入時の処理の流れ

## 5 おわりに

実行中プログラムの一部分を変更する方法として，動的リンクライブラリを入替える方式について述べた．具体的には，DK 上で入替え対象となる動的リンクライブラリの内容を書き換え，当該動的リンクライブラリにおける情報を初期化する方法について述べた．また，動的リンクライブラリの入替えを行うためのプログラム部分の管理方式についても述べた．本方式を利用することで，入替え内容の作成を柔軟に行うことができる．

一方，本方式では，入替え可能状態が未使用状態のみとなることから，入替え要求が発生してから，LS 複写処理が実行されるまでの時間が長くなってしまう問題がある．この問題に関しては，文献 [4] や 文献 [5] で提案した方式の適用が考えられる．

残された課題として，本入替え法の評価がある．

#### 参考文献

[1] 谷口秀夫，伊藤健一，牛島和夫，"プロセス走行時におけるプログラム部分の入替え法，" 信学論 (D-I)，vol.J78-D-I，no.5，pp.492-499，May 1995.

[2] 谷口秀夫，後藤真孝，"走行中のプロセス間で共有されたプログラムの部分入替え法，" 信学論 (D-I)，vol.J80-D-I，no.6，pp.495-504，June 1997.

[3] 山本淳，谷口秀夫，"動的リンクを利用した実行中プログラムの部分入替え法における状態把握法，" 情処学 OS 研報，June 2002，vol.J85-D-I，no.8，pp.713-724，June 2002.

[4] 中島雷太，谷口秀夫，"実行中プログラムの部分入替え法における入替え処理時間の短縮，" 情処学論，vol.41，no.6，pp.1734-pp1744，June 2000.

[5] 山本淳，谷口秀夫，"実行中プログラムの部分入替え法における入替え条件の緩和，" 信学論 (D-I)，vol.J85-D-I，no.8，pp.713-724，Aug 2002.