2005－OS －98（12）
2005／2／23

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の評価

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を実現するためには，サービスを提供しているプログラムを動作させたまま，その内容を変更できる必要がある．そこで，我々は，入替え対象となるプログラム部分として動的リンクライブラリに着目し，プロセス走行中に動的リンクライブラリに対して入替えを行う方法について提案した．動的リンクされたプログラムは，一つの実行可能ファイルと複数の動的リンクライブラリによって構成される．また，動的リンクライブラリは，プログラム実行時に動的ローダによってロードされる．このロード機能を入替えに利用することにより，入替え内容の作成において高い利便性を提供することができる．本稿では，提案した入替え方式に対する評価を行い，その結果を報告する．

Evaluation of Basic Mechanism for Exchanging Dynamic Link Libraries during Excution

Hayato YOSHIHARA † and Hideo TANIGUCHI ††

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

To implement computer which provides continuous services, it is essential to exchange parts of the program without down time. So, we have purposed the mechanism for exchanging parts of dynamic linking program. The program which linked dynamically is consists of an excutable file and some dynamic link libraries. And these file is loaded by dynamic linker at run-time. By using this function, we can create program parts to replace conveniently. In this paper, we evaluate the mechanism of exchanging executed dynamic link libraries which we have purposed.

1　はじめに

現代社会において，計算機は高度化し，計算機によるサービスはさまざまな場面で提供されるようになった．また，それらのサービスの実現にはソフトウェアが深く関与している．計算機によるサービスでは，その保守作業において，その機能を一時停止させる必要がある．しかし，24 時間連続でサービスを提供する場合，このような一時停止は望ましくない．このため，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方法が必要となる．

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，その評価を行っている [2,3] ．

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ことが必須である．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 [4] 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まなくてよ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．そこで， 文献 [5] では，動的リンク機能を利用した入替え方式について提案した．

本稿では，文献 [5] で提案した動的リンク機能を利用した実行中プログラムの部分入替え法を実現し，その評価を行った結果を報告する．

2　入替え法

文献 [5] で提案した動的リンク機能を利用した入替え方式について，簡単に説明する．

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) 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，走行中状態は，当該プログラム部分を実行している状態なので，入替えを行うことはできない．

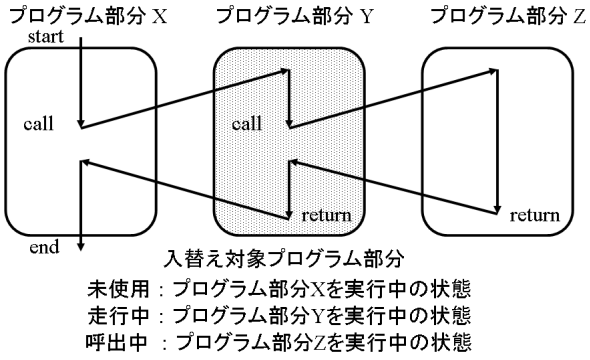

図 1　実行状態と入替え対象プログラム部分の関係

2.2　入替え条件

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 ．

(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，入替えにおける必須条件であり，未使用状態で入替えを実行する場合には，他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，上記 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

(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

ここでは，入替え前後で入替えられるプログラム部分の大きさが異なる入替えについても考慮する．この場合，(条件 3) を満足できない．なぜならば，入替えられたプログラム部分の大きさが入替え前と異なることにより，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ためである．この場合，明らかに戻り先のアドレスは異なる．したがって，ここ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) である．

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により構成される．

(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，およ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，動的リンクライブラリ内のシンボルの参照を契機として，動的リンカ内で行っている．これにより，入替え対象となる AP 内に，状態遷移の検出を行うための処理を記述する必要がなくなる．一方，実行状態の遷移処理については，オペレーティングシステム（以降，OS と略す）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えるようにするためである．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，動的リンカ内から 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) 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．

入替え処理とは，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．これについては，次節で説明する．

2.5 入替え処理

動的リンクライブラリを入替える処理は，以下の二つの処理からなる．なお，以降では，動的リンクライブラリを LS と呼ぶことにする．

(1) LS 複写処理
(2) LS 情報初期化処理

LS 複写処理とは，外部記憶装置上で入替え対象である LS の内容を入替え後の内容に書き換える処理である．LS 複写処理を行う際には，入替え対象である LS 内に対し，全プロセスが未使用状態でなくてはならない．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，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，入替え要求後，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．

LS 情報初期化処理とは，当該 LS をロードした各プロセス単位で管理している当該 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) 入替え対象 LS のアンロード処理
(2) LS 複写処理によって内容を書き換えられた入替え対象 LS のロード処理
(3) 入替え対象 LS の解決していたシンボル情報の初期化処理

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．このため，プロセスは，今後利用することのない LS に対して LS 情報初期化処理を行うことはない．

表 1　入替え制御用システムコール

通番	システムコール	形式	主な機能
(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)	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．
(4)	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．
(5)	入替え要求	syscall_request(old_ls_name, new_ls_name)	入替え対象 LS に対する入替え要求を行う．当該 LS が入替え可能状態になると，LS 複写処理を行う．

3　実装

3.1　入替え制御用システムコール

提案した入替え法を実現するため，表 1 に示す五つのシステムコールを FreeBSD4.3-Release 上に実装した．以下に簡単に説明する．

LS 状態把握開始システムコールは，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．LS 状態把握終了システムコールは，プロセスに対して，指定する LS の状態把握処理の終了を宣言する．LS 突入システムコールは，プロセスに対して，指定した LS の状態遷移を OS に通知する．また，当該 LS に対して入替え要求があれば，入替え可能状態の保持処理を行う．さらに，当該 LS に対して LS 情報初期化処理を行う必要のある場合，戻り値によって本システムコール発行元にその旨を通知する．LS 脱出システムコールは，プロセスに対して，指定した LS の状態遷移を OS に通知する．また，当該 LS に対して入替え要求があり，かつ当該 LS を入替え可能であれば，当該 LS に対して入替え要求を行っているプロセスの待ちを解除する．

入替え処理そのものを行う入替え要求システムコールについては，図 2 に処理の流れを示す．図 2 において，rflag は入替え要求の有無を管理し，dflag は LS 複写処理中であるか否かを管理し，mflag は LS 情報初期化処理を行う必要性の有無を管理する．入替え要求システムコールは，まず旧 LS 名からファイル識別子を

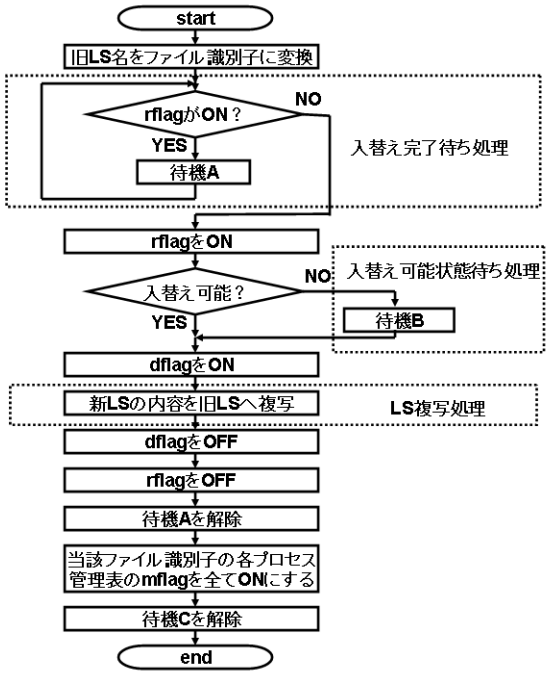

図 2　入替え要求システムコールの処理の流れ

得る．次に，当該 LS に対して既に入替え要求処理を行っている場合，その入替え要求処理の終了を待つ（以降では，入替え完了待ち処理と略す）．入替え完了待ち処理後，rflag を ON にする．そして，当該 LS に対して，入替え可能になるまで待機する（以降では，入替え可能状態待ち処理と略す）．入替え可能状態待ち処理後，dflag を ON にし，LS 複写処理を行う．LS 複写処理完了後，dflag と rflag を OFF にし，入替え完了待ち処理を行っているプロセスがあれ

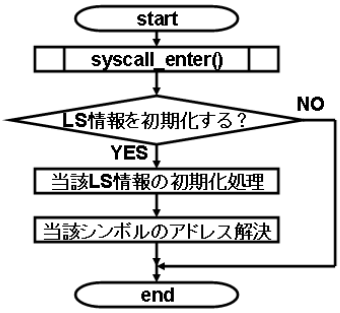

図 3 アドレス解決済み時の LS 突入時の処理の流れ

ば，その待ちを解除する．そして，当該 LS に関連する mflag を ON にし，当該 LS に対して入替え可能状態保持処理中であるプロセスの待ちを解除する．

3.2 LS 突入時の処理

2.5 節に示したように，LS 情報初期化処理は LS 突入時に行う．LS 突入時に行う処理は，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か否かによって異なる．

アドレス解決済みである場合の処理の流れを図 3 に示す．この場合，当該シンボルのアドレス情報は，動的リンカによって管理されている [4]．このため，LS 情報初期化処理後，動的リンカにより再度アドレス解決が行われる．

アドレスの解決済みでない場合の処理の流れを図 4 に示す．この場合，当該 AP のロードした全 LS を探索し，参照したシンボルのアドレス解決処理を行う．アドレス解決処理の際，探索先の LS に対して LS 複写処理を行っていた場合には，その内容が書き換わってしまっているため，不具合が発生する．したがって，LS 突入システムコールを発行することにより，これから探索を行う LS に対して LS 情報初期化処理を行うべきかを確認する．LS 情報初期化処理を行う必要のある場合，LS 情報初期化処理後にアドレス解決処理を行う．

4 評価と考察

4.1 測定項目

入替え制御システムコールに要する処理時間と入替え処理にかかる処理時間を測定し，考察する．

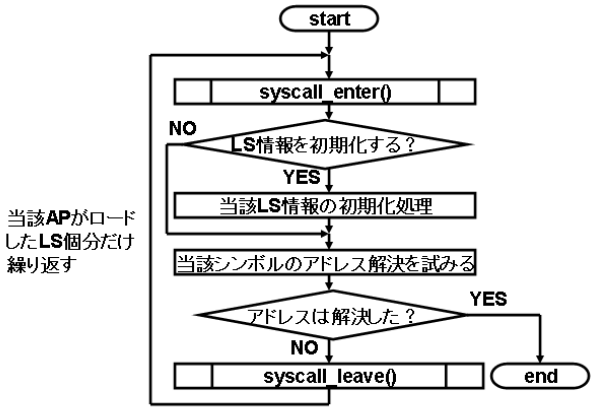

図 4 アドレス未解決時の LS 突入時の処理の流れ

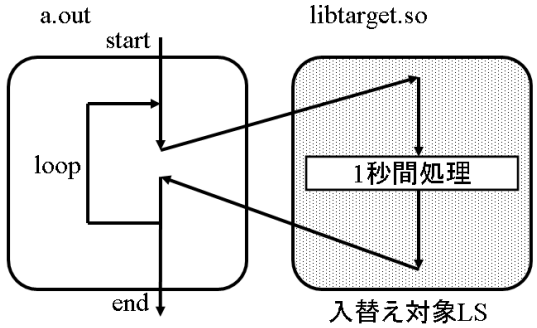

図 5 入替え対象プログラムの処理の流れ

入替え要求システムコールの測定は，以下の四項目に対して行った．

(1) 入替え要求システムコール全体
(2) 入替え完了待ち処理
(3) 入替え可能状態待ち処理
(4) LS 複写処理

LS 情報初期化処理の測定は，以下の四項目に対して行った．

(1) LS 情報初期化処理全体の処理
(2) アンロード処理
(3) ロード処理
(4) シンボル情報初期化処理

4.2 測定条件

入替え対象であるプログラムの処理の流れを図 5 に示す．このプログラムは，一つの実行可能ファイル（a.out）と一つの動的リンクライブラリ（libtarget.so）からなる．libtarget.so は，1 秒間処理を行ってから終了する関数シンボル

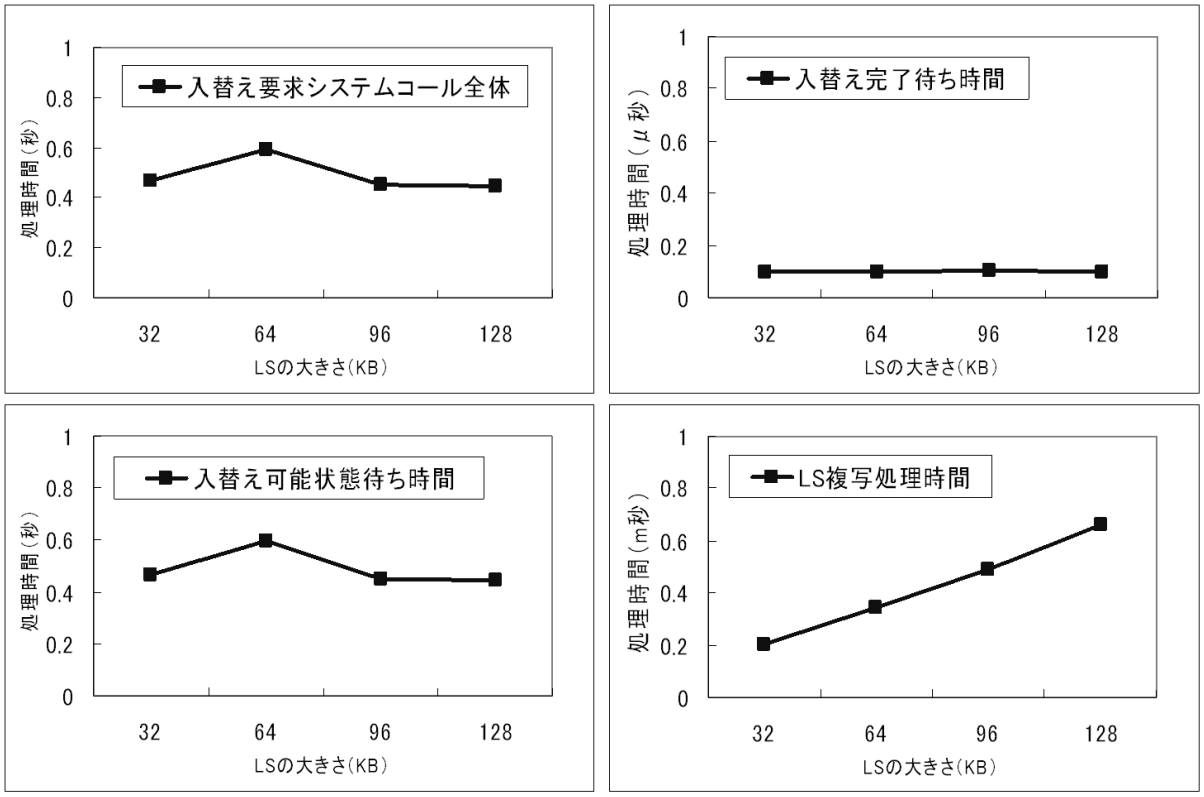

図 6　入替え要求システムコールの処理時間

func() のみである．入替え対象であるプログラムの処理は，a.out から libtarget.so で定義してある関数シンボル func() を繰り返し参照するものである．

入替え要求は，libtarget.so に対して行い，擬似乱数を用いることによりその要求契機は不定期とした．測定は 10 回連続して行い，平均値を測定結果とした．また，実 DK 入出力を発生させないようにして測定を行った．

測定は，Pentium III 550MHz の計算機で行い，ハードウェアクロックカウンタを利用した．なお，入替えの前後で libtarget.so の大きさは同一である．

4.3　入替え制御用システムコール

表 1 の通番 (1) から (4) のシステムコールについて，処理時間を表 2 に示す．表 2 から分かるように，いずれのシステムコールの処理時間も約 1 μ 秒と短い．

表 2　入替え制御システムコールの処理時間

通番	システムコール	処理時間（ μ 秒）
(1)	LS 状態把握開始	1.25
(2)	LS 状態把握終了	1.19
(3)	LS 突入	1.16
(4)	LS 脱出	1.16

4.4　入替え要求システムコール

入替え要求システムコールの処理時間を図 6 に示し，以下に考察する．

(1) 入替え要求システムコール全体の処理時間は，約 0.5 秒である．これに対し，入替え完了待ち時間は約 0.1 μ 秒，入替え可能状態待ち時間は約 0.5 秒，LS 複写処理は約 0.6m 秒以下である．したがって，入替え要求システムコール全体の処理時間の大半は入替え可能状態待ち時間であることが分かる．これは，入替え可能状態待ち時間の期待値が，入替え対象である LS の処理時間の半分であることに起

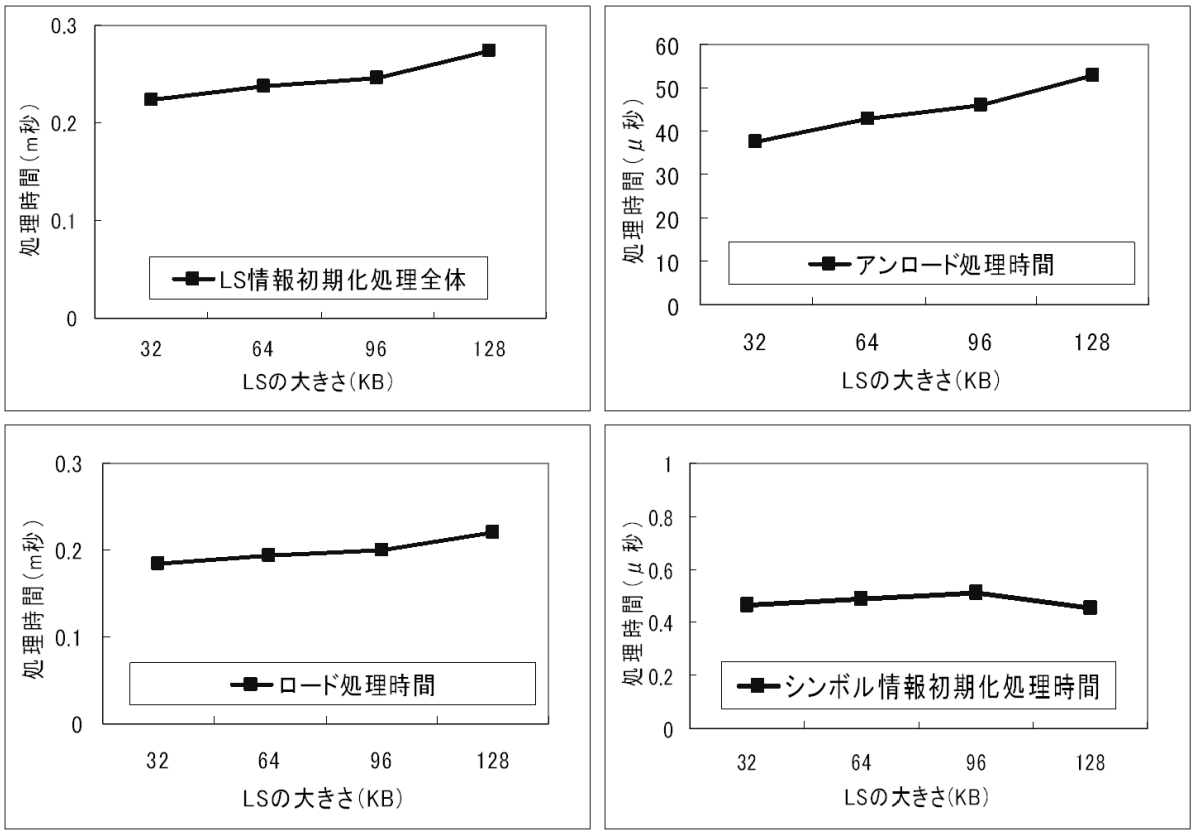

図 7　LS 情報初期化処理の処理時間

因する [3]．本測定では，libtarget.so の処理時間と比較し，入替え完了待ち時間や LS 複写処理時間はかなり短い．

(2) 入替え完了待ち時間は短く，また LS の大きさに関係なくほぼ一定である．これは，入替え完了待ち時間は，同一 LS に対して同時に複数の入替え要求のある場合に変化することに起因する．本測定では，libtarget.so に対し，複数の入替え要求を行っていない．したがって，入替え完了待ち処理は，入替え要求があるか否かの条件判定を一回のみであり，処理時間は変化しない．

(3) LS 複写処理時間は，LS が大きくなるにつれて増加していることが分かる．これは，入替え対象である LS に対する書き込み量が増加するためである．32KB あたりの増加量の平均は，約 0.15m 秒である．本測定では，実 DK 入出力が発生しない環境で測定しているため，この時間は短くなっている．

4.5　LS 情報初期化処理

LS 情報初期化処理時間を図 7 に示し，以下に考察する．

(1) LS 情報初期化処理時間は約 0.25m 秒であり，その約 8 割はロード処理時間である．

(2) シンボル情報初期化処理時間は，LS の大きさに関係なくほぼ一定である．これは，シンボル情報初期化処理が，LS の大きさではなく，当該 LS の利用しているシンボル数に依存する処理であることに起因する．本測定では，LS の大きさに関係なく，シンボル数は一つである．したがって，シンボル表初期化処理時間が変化することはない．

(3) ロード処理時間とアンロード処理時間は，LS の大きさが大きくなるにつれて増加し

ていることが分かる．これは，プロセスアドレス空間に対して実行する物理メモリの割り当て/解放処理の量が増加するためである．ロード時間における 32KB あたりの増加量の平均は，約 11.6 μ 秒である．また，アンロード時間における 32KB あたりの増加量の平均は，約 5.1 μ 秒である．この増加量の観点から，LS の増大に対し，LS 情報初期化処理に与える影響は，アンロード処理よりロード処理の方が大きいといえる．

5 おわりに

動的リンク機能を利用した実行中プログラムの部分入替え法を実現し，その評価結果を報告した．

入替え制御用システムコールの処理時間は，いずれも約 1 μ 秒と短いことを示した．また，入替え要求システムコールの処理では，入替え完了待ち時間，入替え可能状態待ち時間，および LS 複写処理時間の影響について明らかにした．具体的には，入替え完了待ち時間は約 0.1 μ 秒，また LS 複写処理は約 0.6m 秒以下（LS の大きさが 128KB 以下の場合）と短かった．さらに，LS 情報初期化処理に対して，ロード処理による影響が大きいことを明らかにした．具体的には，LS 情報初期化処理時間は約 0.25m 秒であり，その約 8 割はロード処理時間であった

今後は，入替えの前後で LS の大きさが異なる場合に入替え処理の受ける影響について明らかにする必要がある．さらに，実 DK 入出力の影響を受けた場合に入替え処理の受ける影響についても明らかにする必要がある．

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の部分入替え法，” 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] 谷口秀夫，後藤真孝，”実行中プログラムの部分入替え法における入替え時間の評価，” 信学論（D-I），vol.J82-D-I，no.8，pp.998-1007，Aug. 1999.

[4] 山本淳，谷口秀夫，”動的リンク機能を利用した実行中プログラムの部分入替えにおけるプログラム状態把握法，” 信学論 (D-I)，掲載予定.

[5] 吉原隼人，谷口秀夫，”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，” 情処研報，OS-97，pp.25-32 (2004)