動的なプログラム入替えに向けたプログラム実行状態の把握法

LB-5 Grasping Program States for Mechanism of Dynamic Program Exchanging

山本 淳† 谷口 秀夫‡
Atsushi Yamamoto Hideo Taniguchi

1．まえがき

我々は，２４時間無停止でのサービス提供を目指して，サービス提供中のプロセスを終了させることなく，そのプログラムの一部分を入れ替える方法を提案している[1].

プログラム部分の入替えに際しては，当該プログラム部分を実行中のプロセスが存在しないことが必須条件である．このため，各プロセスによるプログラム部分の実行状態を把握する必要がある．本入替え法の実用性を考慮した場合，この状態把握を意識した応用プログラム（AP）作成は好ましくない．そこで，我々は，状態把握方式として，静的リンクを利用してプログラム部分の呼出と復帰を通知するシステムコールを AP に埋め込む方式（injection）ではなく，動的リンクを利用して当該システムコールを動的リンカに組み込む方式（built-in）を提案した[2]．しかし，built-in 方式は，injection 方式に比べて，状態把握を意識した AP 作成は必要なくなる反面，プログラム部分間の呼出と復帰に要する処理時間は動的リンカを経由する分だけ長くなる問題がある．

そこで，本稿では，この問題への対処として，built-in 方式において，プログラム部分間の呼出と復帰に要する処理時間を短縮する方法について述べる．

2．プログラム実行状態の把握法

2．1 基本方式

AP の変更を要求しない状態把握方式として，動的リンクを利用して，AP をロードセグメント（LS）単位で部分分割し，動的リンカにおいて各 LS 間の呼出と復帰を検出する built-in 方式を提案した[2]．この処理流れを図１と図２に示し，以下に説明する．

関数シンボル（function）の参照は，LS（refobj）が保持する symbol-location 表の当該エントリは未解決であるため，動的リンカ（rtld）に制御が移る．このとき，引数としては refobj-symbol 情報が渡される．動的リンカは，引数として渡された refobj-symbol 情報をもとに，ロードされている全 LS を検索してその定義を見つけ出しアドレス解決する（resolve()）．通常，動的リンカは，解決したアドレスを LS（refobj）が保持する symbol-location 表の当該エントリに登録する．しかし，この通常方式では，２回目以降の参照時には動的リンカを経由しないため，動的リンカにおいて各 LS 間の呼出を検出できない．そこで，解決したアドレスを LS（refobj）が保持する symbol-location 表に登録するかわりに，動的リンカにおいて関数シンボルの参照と定義の対応を管理することとし，その対応表に登録することにした（図１の（３））．これにより，２回目以降の参照時にも動的リンカを経由するようにでき，かつ２回目以降には解決済アドレスを使用できるため（図２の（３）），アドレス解決に要する処理時間も削減できる．また，通常，動的リンカは，解決したアドレスへ jmp 命令によって制御を移す．しかし，この通常方式では，当該関数からの復帰時には動的リンカを経由しないため，

†九州大学大学院システム情報科学府
‡九州大学大学院システム情報科学研究院
Graduate School of Information Science and Electrical Engineering, Kyushu University

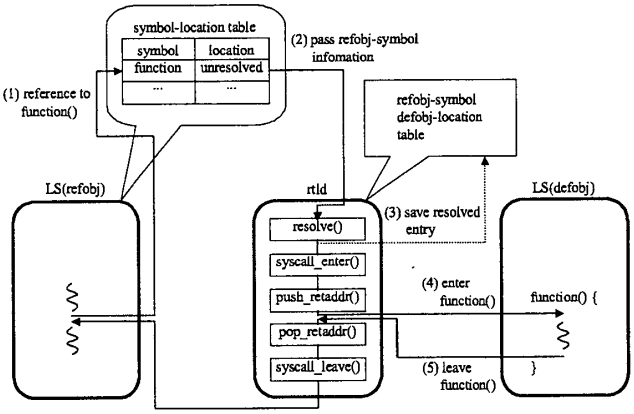

図１ アドレス解決時の処理流れ（１回目）

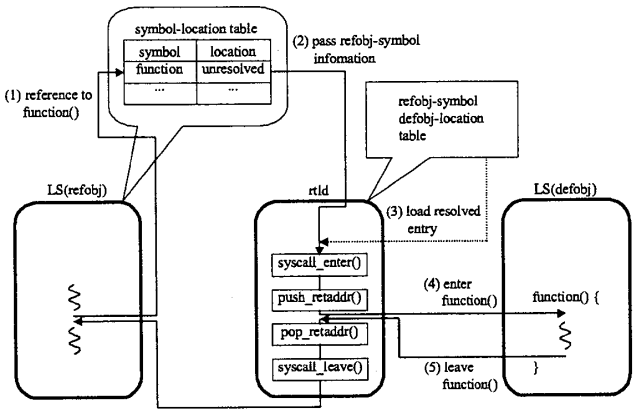

図２ アドレス解決時の処理流れ（２回目以降）

動的リンカにおいて各 LS 間の復帰を検出できない．そこで，解決したアドレスへ jmp 命令ではなく call 命令によって制御を移すようにした．これにより，当該関数からの復帰時にも動的リンカを経由するようにできる．しかし，この場合，LS（refobj）内の真のリターンアドレスが上書きされる．このため，真のリターンアドレスを動的リンカで保存する必要がある．LS（defobj）が LS（refobj）にもなりうることを考慮すると，リターンアドレスはスタックで管理する必要がある．図１と図２において，push_retaddr()と pop_retaddr()がこれにあたる．

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（syscall_enter(), syscall_leave()）を発行することにより，OS において各プロセスによる LS の実行状態を把握でき，当該 LS の入替え可否を判別できる．

2．2 問題点と対処法

built-in 方式は，状態把握用のコードを動的リンカに局所化できるため，状態把握を意識した AP 作成は必要ないという利点がある．一方，injection 方式に比べて，プログラム部分間の呼出と復帰時には動的リンカを経由する必

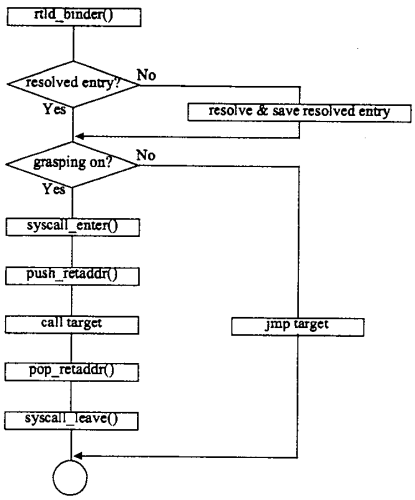

図３　動的リンカの処理手順

要があり，その分だけプログラム部分間の呼出と復帰に要する処理時間は長くなる問題がある．しかし，当然のことながら，状態把握に伴う処理オーバヘッドは小さいことが望ましい．

文献[3]では，injection 方式において，状態把握に伴う処理オーバヘッドを削減するため，状態把握の ON/OFF を宣言するシステムコールを実現し，その有効性を示している．しかし，これは，OS レベルでの状態把握の ON/OFF 機能の実現であり，高価な状態把握用システムコールの発行は回避できない．injection 方式では，AP から OS に直接制御が移るため，OS レベルでの状態把握の ON/OFF 機能しか実現できないが，built-in 方式では，AP から動的リンカを経由して OS に制御が移るため，動的リンカ（ユーザ）レベルでの状態把握の ON/OFF 機能を実現できる．これにより，状態把握 OFF 時に高価な状態把握用システムコールの発行を回避でき，大幅な処理オーバヘッドの削減を期待できる．この改善した動的リンカの処理手順を図３に示す．図３において，二つ目の条件分岐（grasping on?）が状態把握の ON/OFF による分岐であり，この条件分岐が無条件に Yes のものが基本方式である．

３．評価と考察

測定プログラムとして，メインプログラムからライブラリ関数の呼出を１０回繰り返すプログラムを作成した．当該ライブラリ関数は，何も処理を行わず，復帰するのみの関数である．プログラム部分間の呼出と復帰に要する処理時間として，ライブラリ関数の呼出直前と復帰直後のハードウェアクロックを取得し，その差分のクロック数を測定した．

状態把握の実現方式としては，静的リンクの場合には injection 方式，動的リンクの場合には built-in 方式とした．また，状態把握の ON/OFF 機能の実現方式としては，injection 方式と built-in（old）方式は OS レベルで実現する方式，built-in（new）方式は動的リンカ（ユーザ）レベルで実現する方式とし，各方式による状態把握 ON/OFF 時のプログラム部分間の呼出と復帰に要する処理時間を比較した．なお，測定には，静的リンクと動的リンクの両方をサポートしている FreeBSD-4.3 が走行する計算機（PentiumIII 550MHz）を使用した．

測定結果を表１に示す．測定結果は２回目以降の測定値の平均である．これは，動的リンクの場合，１回目の測定値には，アドレス解決に要する処理時間が大きく影響するためである．表１から，built-in（new）方式は，

表１　プログラム部分間の呼出と復帰に要する処理時間

	stat(injection) [μ sec.]	dyn(built-in) [μ sec.]		dyn/stat [ratio]	
		old	new	old	new
default	0.08	0.09		1.13	
grasp_off	2.20	3.47	1.03	1.58	0.47
grasp_on	2.85	4.25	4.93	1.49	1.73

built-in（old）方式に比べて，状態把握 OFF 時の処理オーバヘッドは 1.03 μ 秒と大幅に小さいことがわかる．また，built-in（new）方式は，injection 方式と比較しても，状態把握 OFF 時の処理オーバヘッドは 0.47 倍と小さいことがわかる．一方，状態把握 ON 時の処理オーバヘッドは 1.73 倍と大きいが，プログラム部分の入替え要求時にだけ状態把握を ON にすればよいこと，及びサービスの提供時間に対して保守作業の時間の占める割合はせいぜい数%程度であることを考慮すると，この処理オーバヘッドは問題にならないと考えられる．すなわち，built-in（new）方式は，injection 方式に比べて，状態把握を意識した AP 作成が必要ないだけでなく，状態把握に伴う実効的な処理オーバヘッドも小さくなるといえる．

４．関連研究

AP の変更を要求しない状態把握法については，動的リンクを利用して，任意の LS 間に拡張 LS を挿入可能にすることにより，実行状態を把握する方法が提案されている[4]．しかし，この研究での状態把握は，実行トレースの収集が主な目的であり，プログラム入替えのためのものではない．一方，プログラム入替えのための状態把握法も提案されている[5]．この方法では，Portal と呼ばれる状態把握用のコードをコンパイル時に自動生成し，それにより実行状態をユーザレベルのプロセス毎に保持させている．そして，入替え要求時にだけ OS が各プロセスの実行状態を参照することにより，状態把握に伴う処理オーバヘッドを抑制している．しかし，この方法は，仮想記憶をサポートしてないまたは単一仮想記憶の場合にのみ有効であり，多重仮想記憶の場合，入替え可否の判別に要する処理時間が長くなり，入替え要求から入替え処理を終了するまでの時間が長大化すると考えられる．我々は，多重仮想記憶を想定しているため，各プロセスの実行状態を OS で集中管理する方法を採用している．

５．むすび

実行中プログラムの部分入替え法について，プログラム部分の実行状態を把握する built-in 方式の改善を図った．残された課題として，動的ローダを利用した入替え処理の実現と評価がある．

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の部分入替え法,” 信学論(D-I), vol.J78-D-I, no.5, pp.492-499, May 1995.

[2] 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.

[3] 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.

[4] Albert Serra, Nacho Navarro, Toni Cortes, “DITools: Application-level Support for Dynamic Extension and Flexible Composition,” *Proc. of USENIX Annual Technical Conference*, June 2000.

[5] 小林良岳, 佐藤友隆, 唐野雅樹, 結城理憲, 前川 守, “彩：コンパイル時に自動生成される Portal をもとに動的再構成可能なオペレーティングシステム,” 信学論(D-I), vol.J84-D-I, no.6, pp.605-616, June 2001.