ソフトウェア工学 101−13
（1994. 11. 18）

Ｕｎｉｘ通信ドライバのためのテスト実行環境の実現

西木 健哉† 平田 俊明† 松永 和男‡
(株) 日立製作所 †システム開発研究所 ‡ソフトウェア開発本部

ＵＮＩＸ系通信ドライバのテスト・デバッグ効率の向上を目的として、ドライバプログラムの実機での動作環境を利用者モードのプロセスでシミュレートする統合テスト環境を提案する。各種プロセス動作や入出力割り込み等のＯＳインターフェースの模擬や、コマンド・デーモン等のＡＰプログラムとの連携、回線データの折り返し機能を有する入出力装置の模擬などを特徴とする。本テスト環境を当社ＵＮＩＸサーバ通信管理プログラム開発の連動テスト工程に適用し、不良対策工数比で約４０％削減の効果を確認した。

An Integrated Testing System for UNIX Communication Driver Program

Ken'ya Nishiki † Toshiaki Hirata † Kazuo Matsunaga ‡
Hitachi, Ltd. † Systems Development Laboratory ‡ Software Development Center

In this paper, we propose an integrated testing system for UNIX communication driver program in which target programs are executed using only user-mode processes on the non-target machine. This system has the facilities to simulate OS-driver interface(process control, I/O interrupt, etc.), AP-kernel systemcall interface, and I/O device control. We applied this system to the development of UNIX Server's communication product and confirmed the effect that total debugging cost in program test phase decreased about 40%.

1 はじめに

ＬＡＮの普及、コンピュータのオープン化の進展に伴い、通信系プログラムの重要性がますます高まってきている。また、ＷＳ・ＰＣの普及を背景としたダウンサイジングの浸透により、価格競争力のある製品を短期間で開発することが求められている。ＵＮＩＸ通信ドライバは、他のデバイスドライバと同様にカーネルに組み込まれ、カーネルモードで動作するため通常のアプリケーションの開発環境がそのまま利用できない。そのため、複数のドライバモジュールをＯＳに結合し、コマンド・デーモン等のＡＰプログラム、入出力装置と連動させる必要のある統合テストは実機中心に行われている。これには次のような問題がある。

(1) カーネルモードで実行中の被テストプログラムには他のプロセスから割り込みがかけられないため、任意の時点でプログラムを停止させて内部状態を検査するという通常のデバッグ方法が適用できない。ダンプ解析によるデバッグは効率が悪い。

(2) カーネルモードで動作中の被テストプログラムが不当なマシン命令を実行し、計算機システム全体が停止すること（パニック）がありうる。

(3) カーネルモードで動作中の被テストプログラムは計算機のＣＰＵを占有するため、同じ計算機上でこれらのプログラムのテストを複数同時に行うことができない。

これに対して従来より、ターゲット（実機）によらないテストを行うためのシミュレータ環境が提案されている [1][2]。これらは主に組み込み型マイコンソフトへの適用を考えており、通信系ドライバの開発に適用するには次のような課題があった。

(1) 被テストプログラムを利用者モードとカーネルモードの２つの実行モードを切り替えて模擬実行することが考慮されていない。

(2) 実機では独立したプロセスとして動作する複数のプログラムを１つの利用者プロセスで動作させるため、被テストプログラムをシミュレータ用に修正する必要があったり、プログラムの並行動作が保障されない。

本稿では、まず上記課題を解決するＵＮＩＸ通信ドライバのためのテスト実行環境を提案し、これをＵＮＩＸサーバ通信管理プログラム開発に適用した結果を示す。

2 通信ドライバ統合テスト環境の提案

本章では、通信ドライバを対象にした統合テスト環境を実現するための手段として、デバイスドライバプログラムを利用者モードのプロセスで模擬実行する動作環境シミュレータ及び入出力装置シミュレータを提案する。

2.1 デバイスドライバの基本動作

デバイスドライバを介した入出力処理の流れ（図１）及び本稿で入出力制御の対象とするハードウェア構成例（図２）を示す。

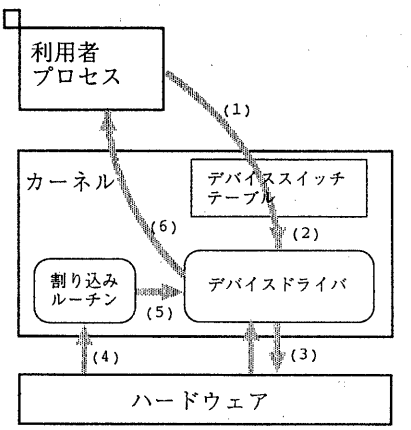

図 1: デバイスドライバの処理の流れ

(1) 利用者プロセスから入出力を要求するシステムコール (ioctl 等) を発行。

(2) 利用者プロセスは利用者モードからカーネルモードに切り替わり、対応するドライバモジュールを呼び出す。ドライバモジュールは利用者プロセスの延長、もしくはカーネルに常駐するシステムプロセスで動作する。

(3) ドライバモジュールは入出力装置を開始させ、入出力要求を渡し、入出力装置からの完了待ち状態に入る（WAIT)。このとき制御は他のプロセスに渡る。入出力装置の制御はすべてメモリ空間に割り当てられたデバイスレジスタを介して行い、データ転送にはＤＭＡ (Direct Memory Access) もしくはＰＩＯ (Programed I/O) が使われる。

(4) 入出力装置において処理が完了するとI/O 終了報告を作成して、カーネルにI/O 割り込み

をかけ、デバイスドライバの割り込みルーチンが呼び出される (ポーリング方式は前提としない)。

(5) 割り込みルーチンは完了待ちプロセスを起こす (POST)。

(6) カーネルモードから利用者モードに切り替わり、入出力結果とともに制御を利用者プロセスに戻す。

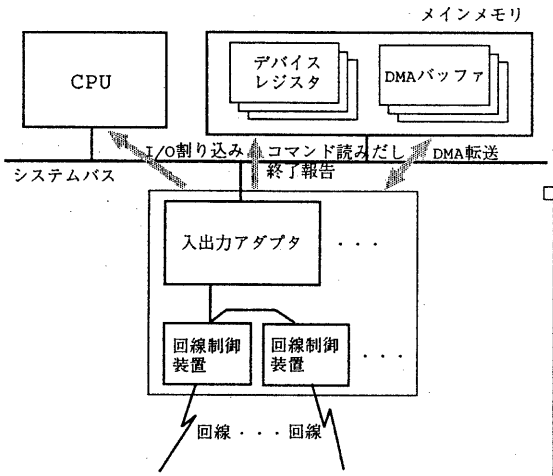

図 2: 入出力ハードウェア構成

2.2 シミュレータの制御構造

シミュレータ全体の制御構造を 図3 に示す。

(1) ユーザモードで動作する部分はそれぞれＡＰプロセスとして実現し、カーネルモードで動作する部分は唯一のＡＰプロセス (疑似カーネルプロセスと呼ぶ) として実現する。カーネルモードで動作するＡＰプロセス、システムプロセス及び割り込みルーチンは各々疑似カーネルプロセス内のスレッドとして実現する。

(2) Ｉ／Ｏアダプタ及びネットワークのシミュレータはユーザの指定した構成の独立したＡＰプロセスとする。

(3) ＡＰ空間とカーネル空間との間のデータ複写、及び主記憶とＩ／ＯアダプタのＤＭＡ転送は専用の共有メモリを介してデータを受け渡すことによりシミュレートする。

(4) シミュレータ環境を構成する各プロセス間の情報の受渡しは、シグナルによる割り込みと共有メモリを用いて行う。

(5) シミュレータの初期化及びユーザコマンド処理を行うプロセスを設け、ＡＰプログラム (被テスト、シミュレータ)、疑似カーネル、疑似Ｉ／Ｏアダプタの各プロセスは子プロセスとして制御する。ユーザコマンドは実機でのテスト同様の操作を可能とし、デバッグは被テストプログラムの起動時または起動中にデバッグモードを指定して汎用のデバッガを起動することにより行う。また、プロセスの分岐を利用して再テスト実行機能を提供する。

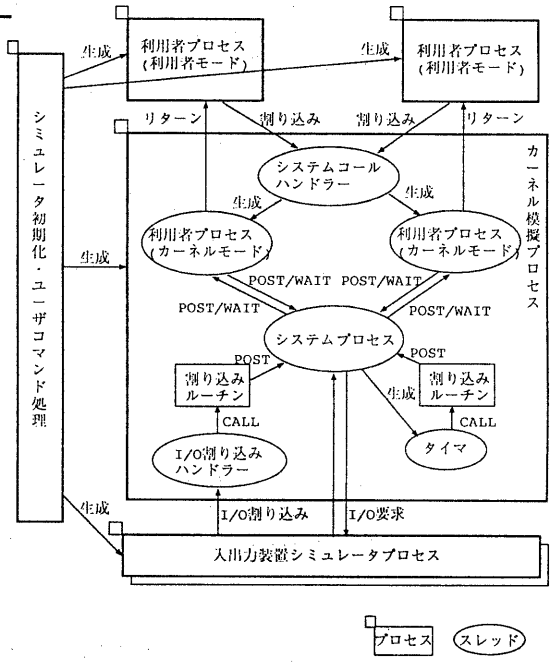

図 3: シミュレータの制御構造

2.3 シミュレート機能

(1) 分離モジュール結合インターフェースのシミュレート

分離モジュール結合インターフェースとは、従来カーネルにバインドしていたドライバ等の機能を別のロードモジュールとして分離し、システム起動時に動的に結合する方法である。以下のサブ機能からなる。

(i) ＯＳ内の各種関数の実行、システム及びカーネル変数の参照／更新。
本シミュレータではドライバに提供されるＯＳインターフェースライブラリをシミュレート版で置き換えることで対応する。

(ii) 分離ドライバの動的ロード・アンロード。
システムのメモリマップ機能を利用して分離ロードモジュールの動的ロード・アンロードを行う。また、分離モジュールの外部参照シンボルのアドレス解決も行う。

(iii) 組み込まれた分離ドライバの初期化。
本シミュレータではシミュレータ起動時に予め登録した分離モジュール結合処理のエントリ関数を順に呼び出すことで分離モジュールの初期化を行う。さらに分離モジュールインターフェーステーブルに設定された内容に基づき以下の処理を行う。

(a) デバイススイッチテーブルの作成。
(b) I/O 用の特殊なメモリ空間の確保 (ハードウェアの設定したアドレスを直接読むための V=R 連続領域等)。
(c) 割り込みルーチンテーブルの作成。
(d) 登録されたシステムプロセスをカーネル模擬プロセス内のスレッドとして起動。

(2) プロセス動作のシミュレート
上記プロセス構成をとることにより、ＡＰプロセス間の並行動作やＣＰＵ及び入出力装置間での並行動作をシミュレートする。

(i) ユーザモードでの動作
独立したＡＰプロセスで動作する。システムコールの発行を契機にカーネルモードに切り替わり、システムコールからリターンを待つ。

(ii) カーネルモードでの動作
疑似カーネルプロセス内のスレッドにマッピングして動作させ、プロセス情報をスレッド単位にテーブルで管理する。カーネルモードではハードウェア割り込みを除きＣＰＵの横取りはなく（ノンプリエンプティブ）、動作中のプロセスがウエイトしない限り他のプロセスに制御が渡らない。これはイベントスケジュールキューを設け、ポスト発行時に要求をキューに登録し、ウエイト発行時にキューからポスト要求を取り出してディスパッチする（当該スレッドにシグナルを送り、自らはシグナル待ちでウェイトする）ことによりシミュレートする。さらに、カーネルモードで動作中は他のＡＰプロセスから割り込まれることはないため、これをセマフォを用いた排他制御機構でシミュレートする。

(3) メモリ空間のシミュレート
上記プロセス構成をとることにより、ＡＰ空間は各プロセス毎にアドレス空間を持ち、カーネルは１枚のアドレス空間を持つことをシミュレートする。また、入出力データ転送を制御するメモリマップトデバイスレジスタ用のＩ／Ｏ空間は、共有メモリの領域をＩ／Ｏモジュール毎に確保して割り当てる。疑似カーネル空間におけるメモリ管理はアドレス変換、スワップ、領域のロック、キャッシュ制御等を含めてシステムに任せることにする (実機と同じ動作を保障することが困難なため)。ユーザ空間とシステム空間の間のデータ複写は、シグナル割り込みを契機に予めシミュレータで確保した共有メモリ領域を介して行う。但し、カーネル空間からＡＰ空間へのダイレクトアクセスは被テストプログラムでＯＳ提供のデータ複写関数を用いるよう変更することで対応する。

(4) ファイルシステムのシミュレート
ユーザ作成のシミュレータ構成情報ファイルに記述されたスペシャルファイル情報、及び mknod システムコールにより作成されたスペシャルファイルを管理する。open, close, ioctl 等のＩ／Ｏまわりのシステムコールで使用されるファイル記述子は、システムで実際に割り当てる番号と重複しないように割り当てて管理する。

(5) システムコールインターフェースのシミュレート
システムコールをシミュレータルーチンコールに置き換え、シグナル割り込みによって疑似カーネルプロセスへの通知（トラップ）をシミュレートする。疑似カーネルプロセスはデバイススイッチテーブルを参照して該当するドライバルーチンを呼び出す。

(6) Ｉ／Ｏ割り込み、ハードウェアインターフェースのシミュレート
ハードウェアからの非同期割り込みは、疑似Ｉ／Ｏプロセスから疑似カーネルプロセスに対するシグナル割り込みによってシミュレートする。疑似カーネルプロセスは疑似Ｉ／Ｏプロセスからの引渡し情報を基に割り込みテーブルを参照して該当する割り込みルーチンを呼び出す。割り込み時のプログラムレベルを基に次のように優先制御を行う。

(i) I/O 割り込みレベル ＞ 動作中のプログラムレベルの場合
動作中のプログラムを中断して、割り込みルーチンを実行する。

(ii) I/O 割り込みレベル ≦ 動作中のプログラムレベルの場合
「I/O 割り込みレベル ＞ 動作中のプログラムレベル」の状態になるまで割り込みルーチンの実行を保留する。

さらにI/O 割り込み処理中はＡＰプロセスから割り込まれることはないため、これをセマフォを用いた排他制御機構によりシミュレートする。なお、ユーザスレッドを使用する場合は割り込みの非同期性は保証されず、スレッドのディスパッチされるタイミングに依存する。
主記憶とＩ／Ｏアダプタ間のＤＭＡ転送は、疑似カーネルプロセスと疑似Ｉ／Ｏプロセス間でシグナル割り込みを契機に予めシミュレータで確保した共有メモリ領域を介して行う。

(7) タイマ機構のシミュレート
インターバルタイマの設定要求があった場合、当該要求をタイマーキューに登録すると同時に割り込み用スレッドを生成する。当該スレッドは指定されたインターバルでウェイトし、その後キューの先頭要求を取り出して割り込みルーチンを呼び出す（既にキューから削除されている場合はそのままリターン）。
インターバルタイマの削除要求があった場合、当該要求をタイマーキューから外す。

(8) 入出力装置、ネットワークのシミュレート
デバイスドライバのレジスタアクセスを入出力装置シミュレータへのＩ／Ｏ要求発行に置き換える。入出力装置シミュレータは、レジスタコマンドの読み書き、送受信したプロトコルデータの解析、応答用データの作成、I/O終了報告の作成を行う。さらに指定された通信回線のペアに関し、一方の回線への送信データを他方の回線の受信データとして疑似的に折り返すことにより同一システム上でデータ送受信のテストを行う。テスト項目に合わせた応答モード（全自動、Ｉ／Ｏ完了レベルのパラメータ指定可能、レジスタ制御レベルのパラメータ指定可能）を設定できるようにする。

2.4 シミュレータを利用したテスト実行形態

上記シミュレート機能により、被テストのドライバモジュールを結合したカーネル模擬プロセスとＡＰシミュレータ(通信ライブラリを含む）プロセス、コマンド・デーモンプロセス、入出力装置シミュレータプロセス等を連動してテストすることができる。以下の適用形態が考えられる。

(1) 通信系疎通テスト・ＡＰＩテスト
あるＡＰＩに対する疎通テスト（回線起動～コネクション確立～データ送受信～コネクション解放の正常系シーケンス）及びＡＰＩのテストは、ＡＰシミュレータを対向で起動し、入出力装置シミュレータの回線折り返し機能を使って行うことができる。

(2) レイヤのプロトコルテスト
被テストレイヤに切り口（ＡＰＩ）を持つ場合には、プロトコルシミュレータを対向で載せ、被テストモジュールと折り返し通信によりテストできる。図4にOSI・NL層モジュールのプロトコルテストの例を示す。

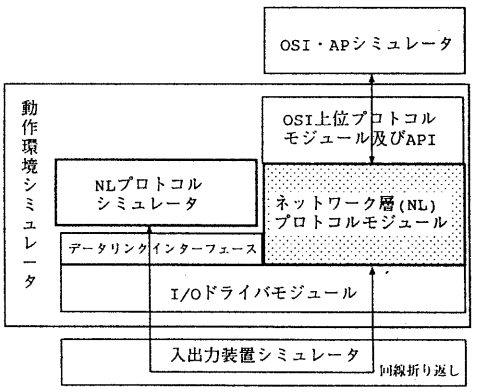

図 4: レイヤのプロトコルテストの例

(3) ハードウェアインターフェースのテスト
入出力装置シミュレータ用のテストパラメータを使用して、アダプタ障害、回線障害等の障害系を含むテストを行える。さらに実機では実施困難な入出力装置の最大構成テストや、ハードウェアすれ違い等のタイミングクリティカルなケースも再現可能である。

3 ＵＮＩＸサーバ通信管理プログラム開発への適用

本章では、実現した統合テスト環境を当社のＵＮＩＸサーバ通信管理プログラムの連動テスト工程に適用した結果に基づき、定量的な評価を述べる。

3.1 適用内容

(1) テスト対象：ＵＮＩＸサーバ通信管理プログラムを構成する１プロトコルモジュール(新規サポート)

(2) テスト構成：前記の通信系疎通テスト・ＡＰＩテスト

(3) 比較対象：過去のＵＮＩＸ通信管理プログラム開発における同一工程の実績。実機もしくは実機に近い環境(カーネルモード)で実施している。プロジェクトの開発経験はほぼ同等のレベルであった。

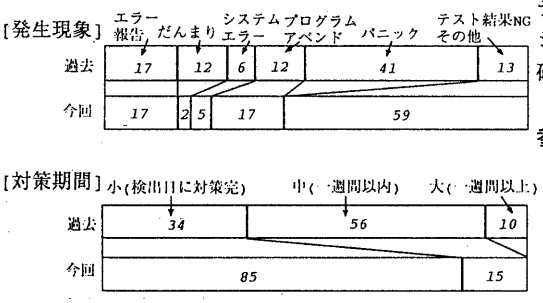

図 5: 連動テスト工程における不良摘出結果の比較（数字は%）

3.2 適用結果

今回適用したプロジェクト及び過去のプロジェクトの連動テストにおける不良摘出実績を発生現象及び対策期間(「小」１日,「中」2.5日,「大」５日換算)で比較した結果が、図５である。発生現象をみるとカーネルモードで頻繁に発生していたシステムパニックがなくなり、通常のテスト結果から不良を検出する割合が増加した。対策期間をみると過去の平均2.25日/件に対して今回の平均1.23日/件と約４０％の効率向上が図れた。なお、ｋｓｔｅｐあたりの不良摘出件数は同等レベルであり、シミュレータ環境での不良摘出能力に問題はないと考えられる。

4 おわりに

ＵＮＩＸ通信ドライバの実機での動作環境を利用者モードで模擬し、コマンド・デーモン等のＡＰプログラムや入出力装置シミュレータと連動させたテスト実行環境の実現について述べた。本テスト環境を実現することにより、

(1) 汎用のシンボリックデバッガを使用してデバッグを行える、

(2) システムパニックを引き起こす不当命令を捕捉して、テストを継続できる、

(3) 同一マシンを使用して異なるテスト構成で並行してテストできる、

(4) 実機では実施困難な障害系や最大構成テスト、再現困難なタイミングクリティカルなテストを行える、

以上により実機テストの前に効率よく統合テスト・デバッグを実施することができる。実際の開発プロジェクトへ適用し、約４０％のデバッグ効率向上を確認した。

参考文献

[1] 栗山 和也、他：シミュレーションによるリアルタイムシステム開発機構：計測と制御, vol.31, No.7, pp.811-815 (1992.7)

[2] 島袋 潤、他：機器組み込み型マイコンソフトの設計・テスト支援方式：情報処理学会（平成５年前期）全国大会, 5-309