

電子情報通信学会技術報告

FIIS-24-600

UNIX第5版におけるプロセススイッチングの可視化

岩島 楓也† 白井 千智† 田村 修†
工藤 信一朗† 石井 充‡ 鷹合 大輔†

† 金沢工業大学

‡ 関東学院大学

2024年6月28日

電子情報通信学会

機能集積情報システム研究会

主催 ディペンダブルコンピューティング研究専門委員会

UNIX 第5版におけるプロセススイッチングの可視化

岩島楓也[†] 白井千智[†] 田村修[†] 工藤信一朗[†] 石井充^{††}
鷹合大輔[†]

† 金沢工業大学 〒 921-8501 石川県野々市市扇が丘 7-1

†† 関東学院大学 〒 236-8503 神奈川県横浜市金沢区六浦東 1-50-1

E-mail: †{c1102668,c1050291,b1817607}@planet.kanazawa-it.ac.jp,
††{o.tamura,takago}@neptune.kanazawa-it.ac.jp, †††ishii@kanto-gakuin.ac.jp

あらまし 本稿では、以前開発した UNIX 第5版（1974年）用のシステムトレーサの改良版について紹介する。新トレーサは旧トレーサと比べて、個々のプロセスの詳細な情報が一覧表示されるようになったことに加え、コマンドの名前やコンテクストスイッチング回数といった、プロセス構造体やユーザ構造体に含まれていない情報も収集して表示するようにした。新トレーサによる解析事例として4つ取り上げ、その有効性を示した。

キーワード オペレーティングシステム, UNIX 第5版, プロセス, カーネル, トレース, プロセス切り替え

Visualization of Process Switching in the UNIX 5th Edition

Fuya IWASHIMA[†], Chisato SHIRAI[†], Osamu TAMURA[†], Shinnichiro KUDO[†], Mitsuru ISHII^{††},
and Daisuke TAKAGO[†]

† Kanazawa Institute of Technology, 7-1 Ohgigaoka, Nonoichi, Ishikawa 921-8501 Japan

†† Kanto Gakuin University, 1-50-1 Mutsuura-Higashi, Kanazawa-ku, Yokohama, Kanagawa 236-8503 Japan
E-mail: †{c1102668,c1050291,b1817607}@planet.kanazawa-it.ac.jp,
††{o.tamura,takago}@neptune.kanazawa-it.ac.jp, †††ishii@kanto-gakuin.ac.jp

Abstract This paper introduces an improved system tracer that visualizes the internal information of UNIX 5th Edition released in 1974. Compared with the previous tracer, the new tracer displays not only a list of detailed information on individual processes, but also information such as the names of commands not included in the process or user structures and the number of context-switching times. Four examples of analysis using the improved tracer are presented to demonstrate the effectiveness of the new tracer.

Key words Operating System, UNIX 5th edition, Process, Kernel, Tracing, Process Switching

1. はじめに

UNIX はベル研究所で開発されたオペレーティングシステムで、第1版は1971年にリリースされた[1]。第5版[2,3]までは実験的なシステムとして主にベル研究所内で使われ、外部にリリースされたのは第6版（1975年）からである。第1版の登場から半世紀を経て、第1版から第4版（1973年）のソースコードやファイルシステムの多くが失われてしまったが、第5版はソースコードとファイルシステムが残っており、現在でも旧式計算機のシミュレータ[4]を用いて稼働させることができる。

これまでに筆者らは初期の UNIX の仕組みを解析・記録することを目的として、第1版と第5版を稼働させたままシステム

状態を確認できるシステムトレーサを開発した[5,6]。しかしながら、これまでのトレーサでは、プロセススイッチやメモリ割当て状況のプロット表示が読み取りにくい上に詳細な情報が示されていないという問題点があった。そこでプロット表示を廃止し、代わりにテキストで詳細な情報をリスト表示する方式に変更した。本稿では第5版におけるプロセス管理の解説と、新トレーサを使ったプロセススイッチングのトレース事例について述べる。

本稿の構成は次のとおりである。まず2.で、本稿を読むにあたって必要な知識として第5版のプロセス管理機構について述べる。続いて、改良したシステムトレーサについて3.で述べ、それを用いた解析事例を4.で述べる。5.でむすびとする。

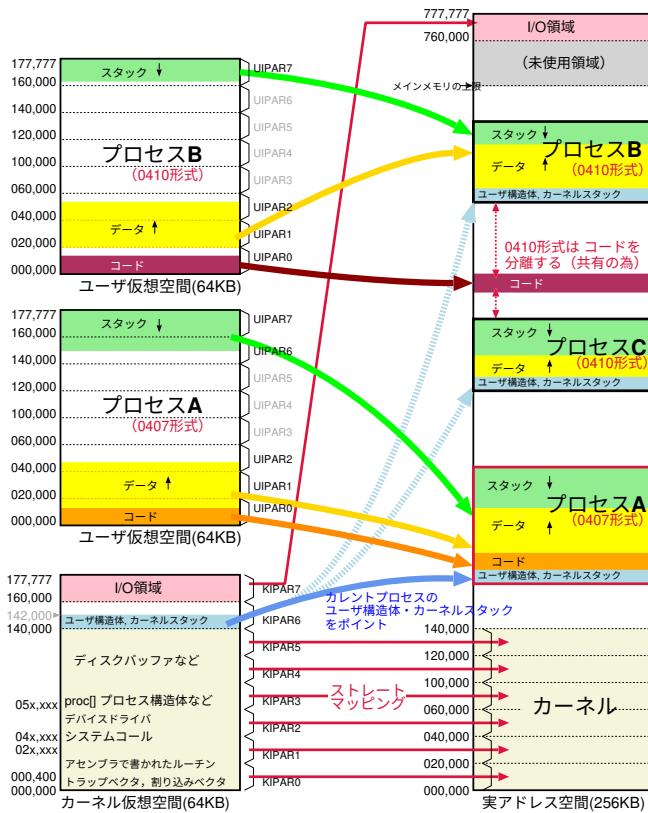


図 1 UNIX 第 5 版のメモリ空間 ([6] より引用)

Fig. 1 Memory spaces for UNIX 5th Ed.

2. UNIX 第 5 版におけるプロセス管理

本章は、次章以降に必要な知識を整理したものである。整理にあたり、第 5 版ソースプログラム [3]、及び第 6 版に関する文献 [7–9] を参考とした。

2.1 メモリ空間

図 1 に第 5 版のメモリ空間を示す [6]。

(1) I/O 領域とメインメモリ領域

実アドレス空間は 256kB の空間で上位 8kB は I/O 領域となっており、下位領域がメインメモリに割り当てられる。

(2) カーネル仮想空間 (64kB)

カーネル仮想空間の上位 8kB は I/O 領域にマッピングされ、下位 48kB は実空間にストレートマッピングされる。カーネル仮想空間の上位から 2 つめのある 1kB 領域 ($014,000_8 \sim 0142,000_8$ 番地) は、カレントプロセスに関するユーザ構造体とカーネルスタック (プロセスがカーネルモードで動作しているときに使うスタック) であり、プロセス切替えの際にマッピング先が切り替わられる。

(3) ユーザ仮想空間 (64kB)

ユーザ仮想空間はスタック領域、データ領域、コード領域で構成される。スタック領域が不足すると 1280 バイト単位でカーネルによってスタック領域が拡張される。

2.2 プロセス構造体 (proc)

プロセス構造体は PID(プロセス ID)、プロセスの実行状態、フラグ、プライオリティ、プロセスに割り当てられている物理メモリ領域、スワップアウト先のディスクブロック番号などを

格納する 20 バイトの構造体である。UNIX 第 4 版以降では 50 個のプロセス構造体がカーネル空間上に確保されており、プロセススケジューラはその情報を元に次に実行するプロセスの選定を行っている。50 個のプロセス構造体のうち、0 番はシステムプロセス sched(PID:0) に割り当てられる。また、1 番は init(PID:1) に割り当てられる。

第 5 版ではプロセスの実行状態として、SSLEEP、SWAIT、SRUN、SIDL、SZOMB の 5 つの状態がある。SSLEEP と SWAIT はウェイクアップされるのを待っている状態 (両者の違いは優先順位)、SIDL はプロセス生成処理中の状態、SZOMB はゾンビ状態 (プログラムの実行が完了した状態) である。SRUN は実行中及び実行可能であることを表す状態で、スケジューラによる CPU 割り当てる際の選択候補となる。

フラグは SLOAD、SSYS、SLOCK、SSWAP の 4 つのフラグがある。SLOAD はプロセスがオンメモリであることを表し、SSYS はシステムプロセスであることを表す (PID:0 の shced)。SLOCK はスワップアウト禁止、SSWAP はスワッpin時に追加処理が必要であることを表す。プロセス構造体のアドレス情報 (メンバ p_addr) は、オンメモリプロセスに対してはメインメモリ上のアドレスとなっているが、スワップアウトするとスワップアウト先のディスクブロック番号となる。

2.3 ユーザ構造体 (user)

ユーザ構造体はプロセスが新たに生成される際に用意され、そのプロセスが終了すると廃棄される構造体である (サイズは 242 バイト)。この構造体にはユーザ仮想空間の情報、参照しているプロセス構造体のアドレス、ユーザモード・カーネルモードでの CPU 使用時間、システムコールを行った際の引数、開いているファイルの情報などが格納される。メインメモリ上では、ユーザ構造体はプロセスの実体 (コード領域・データ領域・スタック領域) と連続する領域に配置される。

プロセススケジューラによって、CPU の使用権がプロセスにディスパッチされる際、そのプロセスのユーザ構造体がカーネル仮想空間の $014,000_8$ 番地にマッピングされるようになっており、それによりカーネルはそのプロセスが持っている情報にアクセスできるようになる。また、プロセスがシステムコールでカーネルモードに切り替わり、カーネル空間上を走行するときにもアクセスできる。

ユーザ仮想空間上にはコード領域・データ領域・スタック領域がマッピングされるが、ユーザ構造体はマッピングされないので、ユーザプログラムからはユーザ構造体にアクセスすることはできない。

2.4 スワップ領域

スワップ領域とは、プロセスが置かれたメインメモリ領域を退避するために設けられたディスク領域である。スワップ領域の用途は次のとおりである。

- (1) メモリの空き領域の確保 (CPU を割り当てていないオンメモリのプロセスをスワップアウト)
- (2) プロセスのスタック領域を拡張する際の作業領域 (一時的にスワップアウトしてスタック領域を拡大)
- (3) プロセスの終了処理の一部 (プロセスがゾンビプロセス)

スとなる時に使っていたユーザ構造体をスワップアウト^(注1))

2.5 プロセスの生成と終了に関するシステムコール

プロセスの生成と終了に関するシステムコールとして fork, exec, exit, wait について簡単に説明する。

(1) fork はプロセスのコピーを行うシステムコールで、コピーされる情報はプロセス構造体、データセグメント（ユーザ構造体とカーネルスタック、データ領域、スタック領域）である。ただし、新しいプロセス構造体の PID には新しい PID が設定され、PPID として親プロセスの PID が設定される。

(2) exec はコード領域とデータ領域を実行可能ファイルからロードして、プロセスイメージを差し替えるためのシステムコールである。exec によってロードされる実行プログラム及び、それらに与えるコマンド引数はスタック領域の最下部部分に格納されるようになっている。コマンドの情報はプロセス構造体やユーザ構造体には格納されないことに注意する。

(3) exit はプロセスが終了される際に呼び出されるシステムコールで、次の処理を行う。

- ユーザ構造体をスワップアウト
- プロセスに割り当てたメモリ領域の開放
- プロセス状態を SZOMB (ゾンビ) にセット
- init プロセスと親プロセスをウェイクアップにセット
- 子プロセスがある場合は、その親プロセスを init(PID:1) にセット（自分が消えるので、親を切り替える）

• swtch() で実行プロセスを切り替える

(4) wait は生成した子プロセスをフォアグラウンドで実行させたいときに親プロセス側が実行するシステムコールで、次の処理を行う。

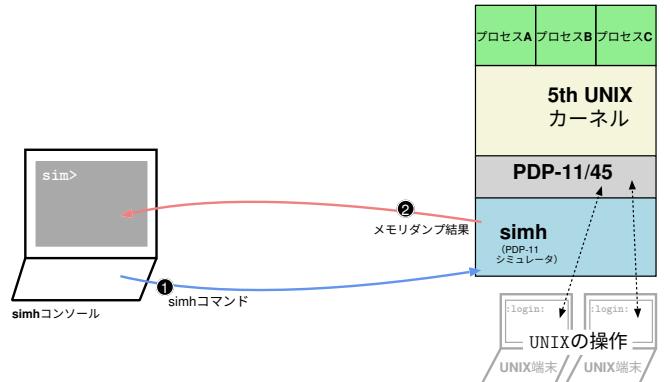
- スリープして、子の終了を待つ（ウェイクアップは、終了処理に入った子プロセスの exit システムコールによって行われる）
- ゾンビプロセスの開放 (proc 構造体の初期化、スワップ領域上のユーザ構造体から CPU 使用時間を取り出した後にスワップ開放)

3. システムトレーサの改良

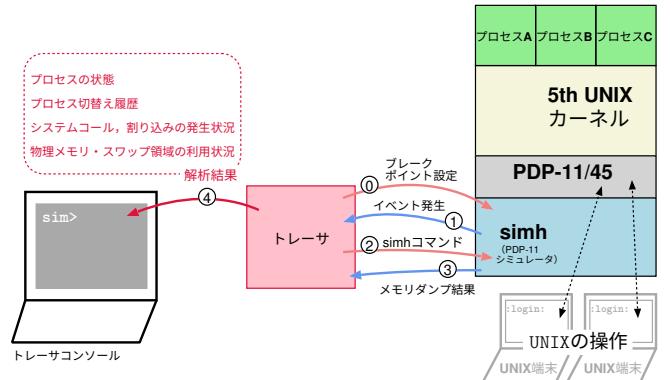
3.1 トレーサの必要性、仕組み

UNIX 第 5 版は DEC 社の PDP-11/40 や PDP-11/45 で動作するように設計されている。PDP-11 シリーズを含む旧式の計算機は現在では simh (History Simulator) [4] と呼ばれるシミュレータで実行することができる。しかしながら、図 2(a) に示すように simh で UNIX を稼働させて、ユーザ仮想空間・カーネル仮想空間・物理メモリ空間をダンプしても、表示される結果が何の情報であるかを特定することは容易ではない。そこで筆者らは図 2(b) のように simh を間接的に制御し、simh から得られた結果を自動的に UNIX カーネルの情報に直して出

(注1) : UNIX 第 5 版ではユーザ構造体 (242 バイト) が含まれる 4kB (8 ブロック) をスワップアウトさせるのに対して、UNIX 第 6 版では 512B (1 ブロック) をスワップアウトさせる変更されている。ユーザ構造体は 1 ブロックに収まるため、不要な領域を転送しなくて済むように変更されたのではないかと思われる。



(a) PDP11 シミュレータの直接操作



(b) トレーサによる UNIX の内部情報の取得

図 2 UNIX 第 5 版のシステムトレーサの仕組み
Fig. 2 Internal processing of the system tracer for UNIX 5th ed.

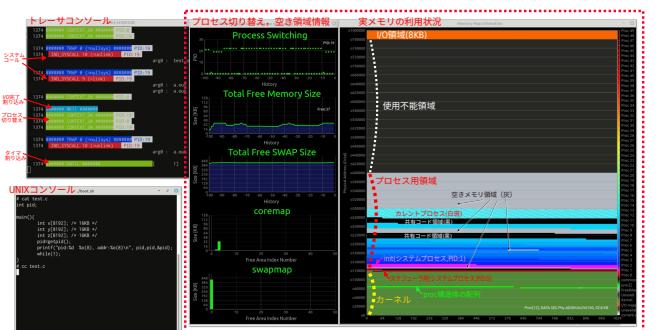


図 3 旧トレーサの画面例 ([6] より引用、編集)

Fig. 3 An example screen of the previous tracer

力するトレーサを以前開発した [6]。トレーサでは予め抜き出しておいたシンボルテーブルをもとに、カーネル仮想空間内のプロセス構造体、スワップ空き領域 (swap_map) とメインメモリ空き領域 (core_map) を取得し、プロット表示するようになっている。以降、以前開発したトレーサを旧トレーサ、今回改良したトレーサを新トレーサと呼称する。新トレーサでは、旧トレーサと比べ、個々のプロセスの詳細な情報表示が可能となっている。

3.2 改良箇所

旧トレーサではコンテキストスイッチ履歴、メモリレイアウト、メインメモリ及びスワップの空き状況をプロット表示していた (図 3)。しかし、(1) プロセスが配置されているメモリア



図 4 新トレーサの画面構成
Fig. 4 An example screen of the latest tracer

ドレスやサイズ、各プロセスの境界の目視が困難であること、(2)どのプロセスから、どのプロセスへスイッチングしたのかわかりにくいくこと、(3)空きメモリ領域と空きスワップ領域の正確な空き容量の確認が困難であることが問題であった(図3の赤点線枠部分)。そこで赤点線枠部分について、新トレーサでは(1)プロセス構造体リスト、(2)コンテクストスイッチ履歴、(3)メモリレイアウト、という3つのテキスト画面(図4)に置き換えた。これらの画面は同時に表示することが可能で、旧トレーサよりも詳細な情報を確認できるようになっている。

(1) プロセス構造体リスト画面

この画面では、プロセスごとの基本情報として、プロセス構造体番号、PID、PPID、コマンド名、コンテクストスイッチ回数、プライオリティ、プロセス状態、フラグ、メインメモリアドレス(プロセスが置かれている領域の先頭アドレスとサイズ)、スワップアウト先ブロック番号をリスト表示する(図4左上)。これらの情報のうち、コマンド名とコンテクストスイッチ回数については、プロセス構造体には元々含まれていない情報であるので、トレーサ側で管理している情報である。フラグは4文字で表示するようにし、SLOAD, SSYS, SLOCK, SSWAPの順でLSIsとなる。コマンド名はユーザ仮想空間の上位部分に配置されるスタック領域から取り出すようにした。これにより、ユーザ仮想空間に割り当てられている現行プロセスのコマンドだけを取り出すように設計することで、シミュレーション停止時間が短くなるようにした。

(2) コンテクストスイッチ履歴画面

この画面では、CPUが新たに割り当てられたプロセスの情報を、プロセス構造体リスト画面から抜き出したものを時系列表示する(図4左下)。最新の情報が画面下部に挿入されるため、古い情報ほど上のほうに表示され、そのうちスクロールアウトする。旧トレーサとは異なり、PIDに加えてプロセスの基本情報を表示するように変更した。これにより、PPIDやプライオリティを同時に確認することができるようになり、利便性が向上した。また、新トレーサではスケジューラプロセス(sched, PID:0)へのスイッチは表示しないように変更した。カーネルはスケジューラプロセスにスイッチさせてから、通常のプロセスにスイッチさせるようになっているが、スケジューラプロセスも履歴に含めてしまうと読みづらいと考えた。

(3) メモリレイアウト画面

この画面では、カーネル領域、オンメモリのプロセス、空きメモリ領域、使用不能領域、I/O領域をアドレス順にソートして表示する(図4右下)。スワップアウトしたプロセスはメモリレイアウト画面に表示されなくなるため、オンメモリプロセスの把握が容易になった。旧トレーサでは、プロセスの区切りや使用番地がわかりにくかったが、新トレーサではテキストで表示することで視認性が向上した。

3.3 制限事項、既知の問題

共有テキスト領域の可視化、空きスワップ領域の可視化については対応していない。既知の問題としてプロセスの状態がSWAITであるにも関わらずSRUNと表示されてしまうこと、トレーサが不意に停止することがある。これらの制限事項・既

図 5 事例 I) ログインとシェルの操作（コンテクストスイッチ履歴）

Fig. 5 Login shell operations (context switch history screen)

知の問題については今後対応する。

4. 新トレーサによるトレース事例

メインメモリが 128kB の DEC PDP-11/45 上で 5 版を稼働させた状態でトレースした 4 つの事例について述べる。

4.1 事例 I) ログインとシェルの操作

ユーザがログインして、最初のコマンド（ここでは echo）の実行を終えたときコンテクストスイッチ履歴画面を図 5 に示す。

UNIX 第5版ではログインシェルは cmd 欄に - (ハイフン) で示される。同図から、先ずログインシェルが起動され、続いてシェルが子プロセスを生成されること (fork)，子プロセスでは実行可能プログラムをディスクから読み出してプロセスイメージを差し替えること (exec)，生成された子プロセスの実行が終わると init プロセスと親プロセスがウェイクアップすることかわかる。

4.2 事例 II) タイマ割込みに伴うプロセススイッチング

UNIX 第 5 版では、KW11 と呼ばれる電源周波数を利用したリアルタイムクロック割り込み回路を用いて、1 秒間に 50 回（または 60 回）のタイマ割り込み処理を行っている。タイマ割り込み回数をカウントし、1 秒計時するごとにプロセスのスケジューリング処理を行っている。

図 6 は、bas (BASIC インタプリタ) で無限ループする処理 (10 goto 10) を 2 プロセス生成し、1 秒ごとにプロセススイッチする様子を示したものである。bas の起動は echo run | bas b.out のように、echo コマンドで bas に run 指示を与えるようにした。同図 (a) から、2 つの bas プロセス (PID17 と 19) がオンメモリであること、echo は終了してゾンビ化していることがわかる。また同図 (b) から、両プロセスが 1 秒ごとにスイッチしている様子がわかる。

4.3 事例 III) メモリ不足時のプロセス再配置

第5版ではメモリが不足すると、オンメモリプロセスをスワップアウトしてから、実行対象のプロセスをオンメモリにして実行する。

図 8(a) に示されている C プログラムは、スタック上に 48kB の配列データを確保するプログラムである。同図より、連続する空きメモリ領域が 46kB しかないため、プログラムが配置できないことがわかる。このプログラムを実行すると、init やシェ

```

# pid kill cmd          rx pri stat flag proc
  0  0  sleep          19 -189 sleep L-- 01101800-012607771, 1.0KB
  1  0  /etc/init       16 40 wait L-- 01234567-013207771, 1.0KB
  2  0  /etc/init       4  40 sleep L-- 01234567-013207771, 0.0KB
  3  8  /etc/init       2  10 wait L-- 01410480-014527771, 2.0KB
  4  8  /etc/init       2  10 wait L-- 01410480-014527771, 2.0KB
  5  9  /etc/init       2  10 wait L-- 01433000-015247771, 0.0KB
  6 10  /etc/init       2  10 wait L-- 01525000-015677771, 0.0KB
  7 10  /etc/init       2  10 wait L-- 01525000-015677771, 0.0KB
  8 12  /etc/init       2  10 wait L-- 01772000-020437771, 2.0KB
  9 13  /etc/init       2  10 wait L-- 01772000-020437771, 2.0KB
 10 13  /etc/init      16 40 sleep L-- 01772000-020437771, 0.0KB
 11 15  /etc/init      64  40 sleep L-- 02235000-024477771, 3.0KB
 12 16  echo run      15 -50 zomb L-- 007674-0077083, 881KB
 13 16  echo run      15 -50 zomb L-- 007674-0077083, 881KB
 14 18  echo run      11 -50 zomb L-- 0087784-0077133, 881KB
 15 19  bas b.out     13 102 run L-- 03310800-035757771, 12.4KB
 16 19  bas b.out     13 102 run L-- 03310800-035757771, 12.4KB
 17 0  none           0  free --- 00000000-00000000, 0.0KB
 18 0  none           0  free --- 00000000-00000000, 0.0KB
 19 0  none           0  free --- 00000000-00000000, 0.0KB
 20 0  none           0  free --- 00000000-00000000, 0.0KB
 21 0  none           0  free --- 00000000-00000000, 0.0KB
 22 0  none           0  free --- 00000000-00000000, 0.0KB
 23 0  none           0  free --- 00000000-00000000, 0.0KB
 24 0  none           0  free --- 00000000-00000000, 0.0KB
 25 0  none           0  free --- 00000000-00000000, 0.0KB
 26 0  none           0  free --- 00000000-00000000, 0.0KB
 27 0  none           0  free --- 00000000-00000000, 0.0KB
 28 0  none           0  free --- 00000000-00000000, 0.0KB
 29 0  none           0  free --- 00000000-00000000, 0.0KB
 30 0  none           0  free --- 00000000-00000000, 0.0KB
 31 0  none           0  free --- 00000000-00000000, 0.0KB
 32 0  none           0  free --- 00000000-00000000, 0.0KB
 33 0  none           0  free --- 00000000-00000000, 0.0KB
 34 0  none           0  free --- 00000000-00000000, 0.0KB
 35 0  none           0  free --- 00000000-00000000, 0.0KB
 36 0  none           0  free --- 00000000-00000000, 0.0KB
 37 0  none           0  free --- 00000000-00000000, 0.0KB

[proc] FREE: 22.5MB, FREE SWAP: 415.00GB, 630k(s), CX: 242

```

(a) プロセス構造体リスト画面

```

14 18 bas - 7 -189 run (-l--0231189-027037771, 1.4KB) 31.725403
14 18 bas - 7 -189 run (-l--0231189-027037771, 1.4KB) 31.727722
14 18 echo run 8 -180 run (-l--0244280-025177771, 2.9KB) 31.383963
14 18 echo run 4 -59 run (-l--0231189-037477771, 4.8KB) 31.322693
14 18 echo run 9 -180 run (-l--0244280-025177771, 2.9KB) 31.382408
14 18 echo run 18 -59 run (-l--0244280-025177771, 2.9KB) 31.376386
14 18 echo run 11 -59 run (-l--0244280-025177771, 2.9KB) 31.400755
14 18 echo run 11 -59 run (-l--0244280-025177771, 2.9KB) 31.487595
14 19 bas b.out 6 -59 run (-l--0331080-03577771, 11.4KB) 31.417472
14 19 bas b.out 6 -59 run (-l--0331080-03577771, 11.4KB) 31.417472
15 19 bas b.out 7 -189 run (-l--0331080-03577771, 11.4KB) 31.433642
13 17 bas b.out 27 -189 run (-l--0267598-031627771, 11.4KB) 31.488678
15 19 bas b.out 9 -180 run (-l--0331080-03577771, 11.4KB) 31.497761
13 17 bas b.out 28 -185 run (-l--0267598-031627771, 11.4KB) 31.519337
15 19 bas b.out 18 -180 run (-l--0331080-03577771, 11.4KB) 31.547689
15 19 bas b.out 11 -180 run (-l--0331080-03577771, 11.4KB) 31.576859
15 19 bas b.out 12 -180 run (-l--0331080-03577771, 11.4KB) 31.684406
15 19 bas b.out 13 -182 run (-l--0331080-03577771, 11.4KB) 31.704406
15 19 bas b.out 14 -183 run (-l--0331080-03577771, 11.4KB) 33.932182
15 19 bas b.out 29 -180 run (-l--0331080-03577771, 11.4KB) 33.932182
15 19 bas b.out 16 1 bas (PID:17, 19) が1秒ごとにスイッチ
15 19 bas b.out 30 -180 run (-l--0331080-03577771, 11.4KB) 33.932182
4 6 1 /etc/update 4 -98 run (-l--0165380-01717771, 2.3KB) 39.430852
4 6 1 /etc/update 5 -190 run (-l--0165380-01717771, 2.3KB) 39.436693
15 19 bas b.out 20 -185 run (-l--0267598-031627771, 11.4KB) 40.469943
15 19 bas b.out 17 185 run (-l--0331080-03577771, 11.4KB) 40.469943
13 17 bas b.out 32 -185 run (-l--0267598-031627771, 11.4KB) 41.628625
15 19 bas b.out 18 -185 run (-l--0267598-031627771, 11.4KB) 41.628625
13 17 bas b.out 33 -185 run (-l--0267598-031627771, 11.4KB) 43.885822
15 19 bas b.out 19 -185 run (-l--0331080-03577771, 11.4KB) 44.980758
15 19 bas b.out 24 -185 run (-l--0331080-03577771, 11.4KB) 45.980758
15 19 bas b.out 28 -185 run (-l--0331080-03577771, 11.4KB) 47.082926
15 19 bas b.out 31 -185 run (-l--0331080-03577771, 11.4KB) 47.082926
15 19 bas b.out 21 -185 run (-l--0331080-03577771, 11.4KB) 49.263397
13 17 15 bas b.out 36 -185 run (-l--0267598-031627771, 11.4KB) 50.274127

```

(b) コンテクストスイッチ履歴画面

図 6 事例 II) bas 実行時の実メモリ状態の変化とプロセス切り替え

Fig. 6 Real memory state changes and process switching associated with the BAS command

ルをスワップアウトして、メモリを確保した状態で実行される。スワップアウトした後のメモリマップを図 8(b) に示す。

また、プログラムの実行前後のプロセス構造体リスト画面を図 8(c)(d) に示す。両図から複数のプロセスがスワップアウトされた様子がわかる。

4.4 事例 IV) ゾンビプロセスの発生、親プロセスによるリソース開放

シェルはバックグラウンドジョブとして子プロセスを起動する場合は wait システムコールを行わず、変わりに read システムコールで端末からの入力を待つ状態に入る。wait システムコールを行わないため、子プロセスが終了してゾンビプロセス化しても、ゾンビプロセスの開放が行われない。つまり、ユーザがシェルでフォアグラウンドで子プロセスを起動するまではゾンビプロセスが残ったままとなる。

図 9(a) は、シェルから 2 つのプログラム (echo と bas) をバックグラウンドジョブとして起動させ、それぞれの実行が終了してゾンビ化している状態を示している。ゾンビプロセスに対しては割り当てられていたメモリ領域が開放されること、またユーザ構造体を含む領域がスワップアウトされた状態となっていることが確認できる。

ここで図 9(b) のように新しいプログラム (cal) をフォアグラウンドで実行すると、シェルが `wait` を行うことでゾンビがスワップ領域からも消され、リソースが完全に開放されたことがわかる。

```

pid pidLnd cmd      cx  pri stat flag mem(Oct)
0  0  0 [sched]    24 -180 sleep LS--- 01161080-01160771, 0.0KB
1  0  0 [free]     *FREE MEMORY** 28 40 wait L--- 0123500-01254771, 0.0KB
2  1  /etc/int    4 10 wait L--- 0132700-01408771, 2.0KB
3  8  /etc/int    2 10 wait L--- 0149100-01452771, 2.0KB
4  6  /etc/int    2 10 wait L--- 0152500-01524771, 2.0KB
5  9  /etc/int    2 10 wait L--- 0152500-01576771, 2.0KB
6  10  /etc/int   2 10 wait L--- 0152500-01576771, 2.0KB
7  11  /etc/int   2 10 wait L--- 0157700-01650771, 2.0KB
8  12  /etc/int   3 10 wait L--- 0172000-01737771, 2.0KB
9  13  /etc/int   2 10 wait L--- 0223400-02343771, 4.0KB
10 14  /etc/int   2 10 wait L--- 0224400-02157771, 2.0KB
11 15  /etc/int   62 180 run L--- 0244200-03777771, 12.0KB
#FREE MEMORY** 16 0 none
#RESERVED#
#1.0KB
#789000-07777771, 8.0KB

```

現在の最大連続空きメモリ領域（46KB）

```

Trying 127.0.0.1...
Connected to localhost.
Escape character is '^}'.

Connected to the PDP-11 simulator CON-TELNET device
login: root
# cc test1.c
# ./a.out

```

(a) test.c の実行前

```

pid pidLnd cmd      cx  pri stat flag mem(Oct)
0  0  0 [sched]    51 -180 sleep LS--- 01161080-01160771, 0.0KB
1  22  15 [mem0]   59 101 run L--- 0244200-03777771, 31.0KB
#FREE MEMORY** 2 0 none
#RESERVED#
#1.0KB
#789000-07777771, 8.0KB

```

システムプロセス(PID:0)と、a.out(PID:22)だけがオンメモリ

```

Trying 127.0.0.1...
Connected to localhost.
Escape character is '^}'.

Connected to the PDP-11 simulator CON-TELNET device
login: root
# cat test1.c
int pid;
main()
{
    int x[8192];
    int y[8192];
    int z[8192];
    pid = getpid();
    printf("pid=%d\n", pid);
    while(1);
}
# cc test1.c
# ./a.out
pid:22 26(8), addr:1726(8)

```

(b) test.c の実行後

図 7 事例 III) プロセスの再配置の様子（メモリマップ画面）

Fig. 7 Process relocation (memory map screen)

```

pid pidLnd cmd      cx  pri stat flag mem(Oct)
0  0  0 [sched]    24 -180 sleep LS--- 01161080-01280871, 0.0KB
1  0  0 [free]     *FREE MEMORY** 28 40 wait L--- 0123500-01254771, 0.0KB
2  1  /etc/int    4 10 wait L--- 0132700-01408771, 2.0KB
3  8  /etc/int    2 10 wait L--- 0149100-01452771, 2.0KB
4  6  /etc/int    2 10 wait L--- 0152500-01524771, 2.0KB
5  9  /etc/int    2 10 wait L--- 0152500-01576771, 2.0KB
6  10  /etc/int   2 10 wait L--- 0152500-01576771, 2.0KB
7  11  /etc/int   2 10 wait L--- 0157700-01650771, 2.0KB
8  12  /etc/int   3 10 wait L--- 0172000-01737771, 2.0KB
9  13  /etc/int   2 10 wait L--- 0172200-02043771, 2.0KB
10 14  /etc/int   2 10 wait L--- 0172200-02043771, 2.0KB
11 15  1 none
12 16  0 none
13 17  0 none
14 18  0 none
15 19  0 none
16 20  0 none
17 21  0 none
18 22  0 none
19 23  0 none
20 24  0 none
21 25  0 none
22 26  0 none
23 27  0 none
24 28  0 none
25 29  0 none
26 30  0 none
27 31  0 none
28 32  0 none
29 33  0 none
30 34  0 none
31 35  0 none
32 36  0 none
33 37  0 none
34 38  0 none
35 39  0 none

```

(a) test.c の実行前

```

pid pidLnd cmd      cx  pri stat flag mem(Oct)
0  0  0 [sched]    51 -180 sleep LS--- 01161080-01280871, 0.0KB
1  0  0 [free]     *FREE MEMORY** 28 40 wait L--- 0180113- 0180291, 0.0KB
2  7  1 /etc/int    4 10 wait L--- 0180211- 0180291, 0.0KB
3  8  1 /etc/int    2 10 wait L--- 0180353- 0180441, 0.0KB
4  6  /etc/update   7 99 wait L--- 0180353- 0180441, 0.0KB
5  9  1 /etc/int    2 10 wait L--- 0180442- 0180471, 0.0KB
6  10  /etc/int   10 100 wait L--- 0180442- 0180471, 0.0KB
7  11  /etc/int   2 10 wait L--- 0180506- 0180631, 0.0KB
8  12  /etc/int   2 10 wait L--- 0180641- 0180711, 0.0KB
9  13  /etc/int   2 10 wait L--- 0180800- 0180851, 0.0KB
10 14  /etc/int   2 10 wait L--- 0180800- 0180851, 0.0KB
11 15  1 none
12 22  15 [/a.out] 64 185 run L--- 0123100-02057571, 51.0KB
13 0  0 none
14 0  0 none
15 0  0 none
16 0  0 none
17 0  0 none
18 0  0 none
19 0  0 none
20 0  0 none
21 0  0 none
22 0  0 none
23 0  0 none
24 0  0 none
25 0  0 none
26 0  0 none
27 0  0 none
28 0  0 none
29 0  0 none
30 0  0 none
31 0  0 none
32 0  0 none
33 0  0 none
34 0  0 none
35 0  0 none

```

(b) test.c の実行後

図 8 事例 III) プロセスの再配置の様子（プロセス構造体リスト画面）

Fig. 8 Process relocation (process structure list screen)

5. むすび

UNIX 第 5 版の解析用に開発したトレーサを改良し、テキス

```

pid pidLnd cmd      cx  pri stat flag mem(Oct)
0  1  0 [etc/int]  10 -180 sleep LS--- 0115200-01200771, 1.0KB
1  2  7 1 /etc/int  4 10 wait L--- 01152700-01400771, 2.0KB
2  3  8 1 /etc/int  2 10 wait L--- 0115300-01171771, 2.0KB
3  4  6 1 /etc/update 3 10 wait L--- 0115300-01171771, 2.0KB
4  5  9 1 /etc/int  2 10 wait L--- 0115300-01524771, 2.0KB
5  6  10 1 /etc/int 2 10 wait L--- 0115300-01576771, 2.0KB
6  7  11 1 /etc/int 2 10 wait L--- 0115300-01576771, 2.0KB
7  8  12 1 /etc/int 2 10 wait L--- 0115700-01717771, 2.0KB
8  9  13 1 /etc/int 2 10 wait L--- 0117200-01773771, 2.0KB
9  10 14 1 /etc/int 2 10 wait L--- 0117200-01773771, 2.0KB
10 15 14 1 /etc/int 58 10 wait L--- 0123300-02443771, 1.0KB
11 16 15 15 echo 12 -50 zone L--- 0123300-02443771, 1.0KB
12 17 15 bas 12 -50 zone L--- 0077666- 0076751, 801KB
13 18 0 none
14 19 0 none
15 20 0 none
16 21 0 none
17 22 0 none
18 23 0 none
19 24 0 none
20 25 0 none
21 26 0 none
22 27 0 none
23 28 0 none
24 29 0 none
25 30 0 none
26 31 0 none
27 32 0 none
28 33 0 none
29 34 0 none
30 35 0 none
31 36 0 none
32 37 0 none
33 38 0 none
34 39 0 none
35 40 0 none

```

- ・シェル(PID:15)は端末入力を待っている状態。
- ・echoとbas (PID:16と17)は終了してゾンビ状態。
- ・シェルはバックグラウンドで起動したプログラムの終了を待たないためゾンビ状態のまくなる。

(a) リソース前

```

pid pidLnd cmd      cx  pri stat flag mem(Oct)
0  1  0 [sched]    19 -180 sleep LS--- 0115100-01200771, 1.0KB
1  1  0 [etc/int]  4 10 wait L--- 01152500-01326771, 2.0KB
2  2  7 1 /etc/int  2 10 wait L--- 0115300-01452771, 2.0KB
3  3  8 1 /etc/int  2 10 wait L--- 0115300-01452771, 2.0KB
4  4  6 1 /etc/update 7 99 wait L--- 0115300-01717771, 2.0KB
5  5  9 1 /etc/int  2 10 wait L--- 01152500-01576771, 2.0KB
6  6  10 1 /etc/int 2 10 wait L--- 01152500-01576771, 2.0KB
7  7  11 1 /etc/int 2 10 wait L--- 0115700-01650771, 2.0KB
8  8  12 1 /etc/int 2 10 wait L--- 0117200-01773771, 2.0KB
9  9  13 1 /etc/int 2 10 wait L--- 0117200-02043771, 2.0KB
10 14 1 /etc/int 14 -50 zone L--- 0123300-02443771, 1.0KB
11 15 15 echo 12 -50 zone L--- 0077666- 0076751, 801KB
12 16 15 bas 12 -50 zone L--- 0077666- 0076751, 801KB
13 17 0 none
14 18 0 none
15 19 0 none
16 20 0 none
17 21 0 none
18 22 0 none
19 23 0 none
20 24 0 none
21 25 0 none
22 26 0 none
23 27 0 none
24 28 0 none
25 29 0 none
26 30 0 none
27 31 0 none
28 32 0 none
29 33 0 none
30 34 0 none
31 35 0 none
32 36 0 none
33 37 0 none
34 38 0 none
35 39 0 none

```

- ・シェルでフォアグラウンドでプログラムを実行させると子プロセス終了待ちを行つた、そこでゾンビを開放

(b) リソース開放後

図 9 事例 IV) 終了プロセスのゾンビ化と、その親プロセスによるリソースの開放の様子（プロセス構造体リスト画面）

Fig. 9 Zombification of exiting process and release of resources by its parent process (process structure list screen)

ト画面による詳細な情報を表示できるようにした。プロセス構造体やユーザ構造体に含まれないコマンド名や、コンテクストスイッチ回数を取得・表示できるようにした。新トレーサにより、コンテクストスイッチの様子や、メモリの使用状況、スワップアウト先などの情報を把握しやすくなつたことを事例によつて示した。トレーサには幾つかの不具合が残つてゐるため、今後対応する予定である。

謝辞 本研究は JSPS 科研費 21K00256 助成を受けた。

文 献

- [1] K. Thompson, D.M. Ritchie, "The UNIX Time-Sharing System", Communications of the ACM, Vol.17, No.7, 1974.
- [2] K. Thompson, D.M. Ritchie, "UNIX PROGRAMMER'S MANUAL Fifth Edition", June. 1974.
- [3] Warren Toomey, "The Unix Heritage Society", <https://www.tuhs.org/>, 参照 Feb.28, 2023.
- [4] simh, "The Computer History Simulation Project", <https://github.com/simh/simh>, 参照 June 4, 2024.
- [5] 工藤信一朗、鷹合大輔、田村修、石井充, "UNIX 第 1 版におけるタスク切り替え機構の解析", 信学技法 (FIIS-22-558), 2022.
- [6] 鷹合大輔、田村修、工藤信一朗、石井充, "初期 UNIX におけるプロセス切り換え機構の動的解析", 信学技法 (FIIS-23-570), 2023.
- [7] John Lions: A commentary on the Sixth Edition UNIX Operating System, 1977.
- [8] 岩本信一訳, John Lions: Lions' Commentary on UNIX, 1998.
- [9] 青柳 隆宏: はじめての OS コードリーディング UNIX V6 で学ぶカーネルのしくみ, 技術評論社, 2013.