Vol.2018-OS-142 No.1
2018/2/27

2プロセス間における入出力対象スワップ機能の提案

鈴木 森羅[1]　乃村 能成[1]　谷口 秀夫[1]

概要：プロセスは，ファイルや通信といった入出力の対象となる資源を操作する．そこで，2 プロセス間で入出力対象を交換 (スワップ) できれば，既に提案している走行中のプロセスを他プロセスから複製する機能と併用することで，複製先のプロセスへ複製元のプロセスとは異なる入力を与えることができる．例えば，これにより，そのプロセスの動作を観察できる．この様な事例として，ソフトウェアのテストのような特定の状態のプロセスに様々な入力を与えたい場面において，有用である．本稿では，2 プロセス間における入出力対象スワップ機能を提案する．

1. はじめに

プロセスは，入出力を行うため，ファイルや通信といった入出力の対象となる資源をオープンする．これに伴い，カーネルは，プロセス毎に入出力対象に対応するファイル記述子 (以降，fd) を割り当てる．そして，プロセスは，この割り当てられた fd を基に入出力対象を管理，操作する．

走行中のプロセスの入出力対象を変更できれば，既に提案している走行中のプロセスを他プロセスから複製する機能 [1][2](以降，プロセス複製機能) と併用することで，複製先のプロセスへ複製元のプロセスとは異なる入力を与えることができる．例えばソフトウェアのテストにおいて有用である．ソフトウェアのテストでは，特定の状態のプロセスに様々な入力を与えたい場合がある．この際，与えたい入力の数だけプロセスの状態を再現する必要がある．そこで，プロセス複製機能を用いて走行中のプロセスを複製し，複製先のプロセスの入出力対象を変更する．これにより，複製先のプロセスへ複製元のプロセスとは異なる入力を与えることができ，様々なテストが可能になる．もう 1 つの例として，既に提案されているゲームの途中状態を複製・共有するシステム [3] へも利用できる．

走行中のプロセスの入出力対象を変更する方法として，FreeBSD では，UNIX ドメインソケットを用いた fd の受け渡しが実現されている [4]．ここで，UNIX ドメインソケットとは，単一のホスト上のプロセス間での通信に用いられるソケットである．fd の受け渡しは，REX[5] や OpenSSH[6] において，特権分離のため，特権を持つプロセスのオープンしたファイルを特権を持たないプロセスへ渡す際に用いられる．ただし，この方法は，受け渡しを行うプロセス間で通信をしなければならない．このため，他プロセスから入出力対象を変更できない．また，SockMi[7] や Migratory TCP[8] のように TCP/IP 接続をプロセス間で受け渡す技術も研究されている．さらに，reptyr[9] や injcode[10] のように走行中のプロセスに対して新たな制御端末を取り付けるツールも開発されている．

本稿では，2 プロセス間における入出力対象スワップ機能を提案する．入出力対象スワップ機能では，2 つのプロセス間で入出力対象を交換 (スワップ) することで，入出力対象を変更する．

[1] 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University

2. 入出力対象スワップ機能

2.1 機能

入出力対象スワップ機能は，2 つのプロセス間で入出力対象を交換する機能である．具体的には，プロセスが利用している各入出力対象を指定する識別子 (fd) を保存したまま，入出力対象を全て交換する．

本機能の様子を**図 1** に示す．プロセス A は，fd の 1 と 3 を使って入出力対象 1 と入出力対象 2 を利用している．プロセス B は，fd の 1 と 3 を使って入出力対象 3 と入出力対象 4 を利用している．この状態において，本機能を利用すると，各プロセスの入出力対象は交換され，プロセス A は fd の 1 と 3 を使って入出力対象 3 と入出力対象 4 を，プロセス B は fd の 1 と 3 を使って入出力対象 1 と入出力対象 2 を利用できる状態になる．

図 1 に示したように，本機能では，両プロセスの全ての fd について交換を行う．したがって，交換する 2 つのプロセスにおいて，利用している fd の数，およびその数値が全く同じである必要がある．

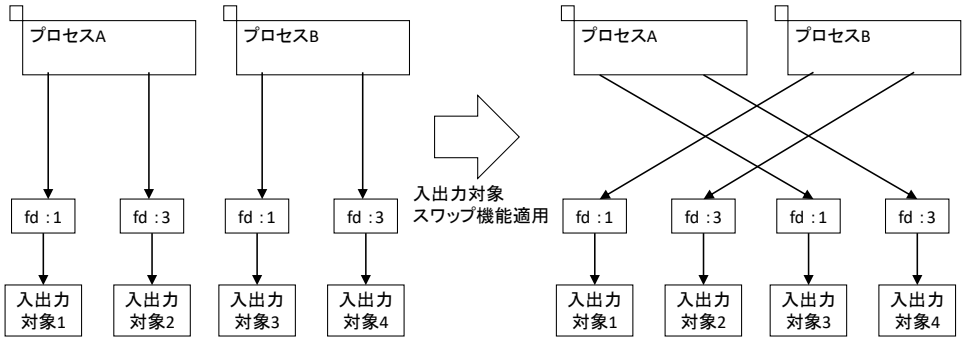

図 1 入出力対象スワップ機能による入出力対象の交換

表 1 交換可能状態の一覧

通番	状態	実行コンテキスト	実行中の処理	交換可能状態か否か
1	RUN	ユーザ空間		可
2		カーネル空間	fd を利用する	不可
3			fd を利用しない	可
4	READY	ユーザ空間		可
5		カーネル空間	fd を利用する	不可
6			fd を利用しない	可
7	WAIT	カーネル空間	fd を利用する	不可
8			fd を利用しない	可

2.2 課題

本機能の実現には，以下のような課題がある．

(課題 1) 交換対象プロセスは入出力対象を交換可能な状態 (以降，交換可能状態) であるか否か判定する方法
プロセスは走行中であり，入出力を行っている可能性がある．入出力中に入出力対象を交換すると，処理の途中で fd に割り当てられた入出力対象が変わってしまい，動作に不具合が発生しかねない．そこで，交換前にプロセスの状態を確認し，交換可能状態か否か判定する必要がある．

(課題 2) 入出力対象の交換を有限時間内で保証する方法
入出力対象を交換するには，両プロセスを判定し，両プロセスは共に交換可能状態でなければならない．ここで，交換対象プロセスの内，一方が交換対象プロセスであり，もう一方は交換可能状態でないとする．特に制御を行わない場合，その後，後者のプロセスが交換可能状態になった際，前者のプロセスが交換可能状態である保証はない．このため，入出力対象の交換を有限時間内で保証できない．そこで，両プロセスは共に交換可能状態であると保証する方法が必要である．

2.3 交換可能状態の判定

2.3.1 交換可能状態

交換可能状態とは，プロセスが入出力対象の管理，操作を行っておらず，交換後も不具合の発生しない状態である．具体的には，プロセスがシステムコールを発行し，入出力対象の管理，操作を行うような fd を利用した処理を実行していない状態である．

したがって，プロセスは，以下の場合に，交換可能状態である．一覧を**表 1** に示す．

(1) RUN 状態であり，(A) 実行コンテキストはユーザ空間にある，または (B) 実行コンテキストはカーネル空間にあり実行中の処理は fd を利用しない
(2) READY 状態であり，(A) 実行コンテキストはユーザ空間にある，または (B) 実行コンテキストはカーネル空間にあり実行中の処理は fd を利用しない
(3) WAIT 状態であり，実行コンテキストはカーネル空間にあり実行中の処理は fd を利用しない

2.3.2 判定の契機

本機能では，入出力対象を交換する前に，2 つの交換対象プロセスが 2.3.1 項で定義した交換可能状態であるか否かを判定する．このため，プロセスの状態に応じた判定を行う必要がある．そこで，交換可能状態の判定方法として，判定を行う契機の異なる以下の 3 つの方法を挙げる．

(方法 A) コンテキストスイッチを契機に判定
コンテキストスイッチの際，コンテキストスイッチ先のプロセスに対して判定を行う．このとき，コンテキストスイッチ先のプロセスは，READY 状態である．このため，コンテキストスイッチ先のプロセスが交換対象プロセスであった場合，実行コンテキストから交換可能状態か否かを判定できる．この結果，実行コンテキストがユーザ空間であれば，コンテキストスイッチ先のプロセスは交換可能状態であると判定できる．

(方法 B) プロセスのシステムコール発行を契機に利用

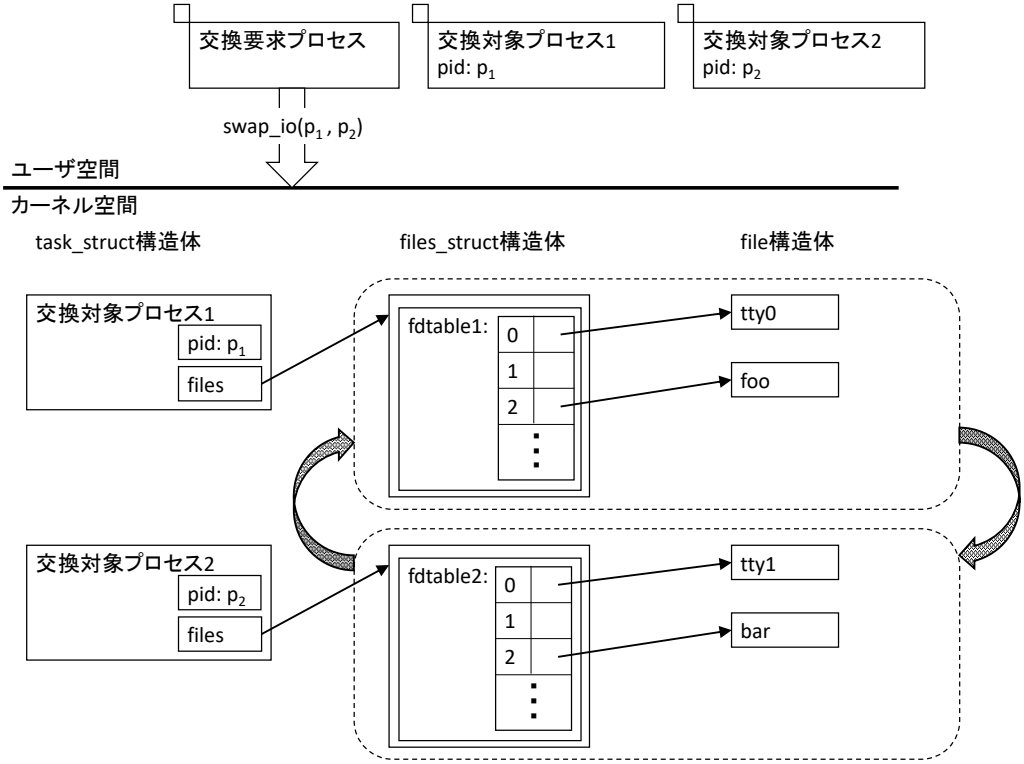

図 2 Linux での入出力対象の管理構成と交換

システムコールを発行したプロセスは，RUN 状態であり，実行コンテキストはカーネル空間にある．ただし，システムコールの発行直後は，ユーザ空間から遷移した直後であり，fd を利用する処理は実行していない．このため，システムコールを発行した直後のプロセスは，交換可能状態である．よって，交換対象プロセスであった場合，判定を行うことなく，システムコールを発行したプロセスは交換可能状態であると分かる．

(方法 C) タイマ割込を契機に判定

タイマ割込の際に割り込まれたプロセスに対して判定を行う．このとき，割り込まれたプロセスは，RUN 状態である．このため，割り込まれたプロセスの実行コンテキストから交換可能状態か否かを判定する．そして，実行コンテキストがユーザ空間の場合，割り込まれたプロセスは交換可能状態であると判定する．

3 つの方法に利用する判定の契機の内，タイマ割込は，他の 2 つの契機に比べ，実行される回数が多い．このため，(方法 C) は，交換可能状態か否かを判定するためのオーバヘッドが大きい．また，(方法 A) と (方法 B) を比べると，(方法 A) はコンテキストスイッチの際に実行コンテキストを確認することで交換可能状態か否かを判定しなければならない．一方，(方法 B) の場合，システムコールを発行したプロセスは，その時点で交換可能状態であり，判定の必要はない．よって，(方法 B) は，(方法 A) に比べ，必要な処理が少ない．

2.4 有限時間内での交換の保証

2.3.2 項で述べた交換可能状態の判定方法は，交換要求後，有限時間内での交換を保証していない．そこで，交換の要求後，有限時間内に交換を実行し，完了できるようにする必要がある．交換可能を保証することは，「2 つの交換対象プロセスが有限時間内で共に交換可能状態になること」を保証することと同じである．

そこで，2 つのプロセスの内，一方のプロセスが交換可能状態であると判定できたとする．このとき，判定できたプロセスの走行を停止させ，交換可能状態のまま待機させる．これにより，もう一方のプロセスが交換可能状態であると判定できた時点で交換可能を保証する．ただし，交換対象プロセスが何らかの事象待ちをしている場合，この事象待ちが有限時間内に解除されないときは交換可能を保証しない．

ここで，交換可能状態の判定方法として，2.3.2 項で述べた交換可能状態の判定方法の中から，(方法 B) を採用する．なぜなら，(方法 B) は，交換可能状態か否かを判定するためのオーバヘッドが小さいためである．また，(方法 B) は，判定のために必要な処理も少ない．したがって，交換対象プロセスがシステムコールを発行した際，交換可能状態であるとする．そして，先に交換可能状態であるとしたプロセスの走行を停止させる．これにより，もう一方のプロセスも交換可能状態となり，入出力対象の交換を待機させる．

表 2 交換要求システムコールの仕様

形式	`int swap_io(pid_t pid1, pid_t pid2)`
引数	`pid_t pid1`: 交換対象プロセスの PID `pid_t pid2`: 交換対象プロセスの PID
返り値	成功:1 失敗:0
機能	`pid1` と `pid2` の PID を持つプロセス間で入出力対象を交換する.

3. 実現

3.1 基本方針

本機能を実現するための基本方針について述べる．まず，Linux における入出力対象の管理構成を**図 2** に示す．プロセスが利用する入出力対象には，カーネルによって fd が割り当てられる．そして，各プロセスは，利用する入出力対象へ割り当てられた fd を管理するテーブルである fd テーブルを持つ．交換対象プロセス 1 は，fdtable1 で自身の利用する fd を管理し，fd の 0 と 2 を使って入出力対象である tty0 と foo を利用している．交換対象プロセス 2 は，fdtable2 で自身の利用する fd を管理し，fd の 0 と 2 を使って入出力対象である tty1 と bar を利用している．このような状態において，本機能を利用した場合，交換対象プロセス 1 は fd の 0 と 2 を使って tty1 と bar を，交換対象プロセス 2 は fd の 0 と 2 を使って tty0 と foo を利用できる状態にする．

そこで，本機能では，図 2 に網掛けの矢印で示したようにプロセス間で fd テーブルを交換する．これにより，交換対象プロセス 1 は fdtable2 で，交換対象プロセス 2 は fdtable1 で自身の利用する fd を管理する．この結果，両プロセスの入出力対象は全て交換される．また，プロセスが利用する入出力対象の数に関係なく，1 度の交換で済む.

加えて，交換の要求は，交換対象ではないプロセスからシステムコールを用いて行う．要求の際には，任意のプロセス 2 つを PID で指定する．これにより，交換要求プロセスの任意のタイミングで交換を要求できる.

3.2 インタフェース

本機能を利用するためのインタフェースである交換要求システムコールの仕様を**表 2** に示す．交換要求システムコールは，2 つの任意なプロセスの PID を引数に渡され，呼び出される．これにより，2 つのプロセスを交換対象プロセスとして指定し，交換を要求する．そして，交換を要求したプロセス (以降，交換要求プロセス) は，交換完了まで走行を停止し，待機する．その後，交換が完了すると，走行が再開され，返り値を返却する.

3.3 交換可能状態の保持

有限時間内での交換を保証するため，交換可能状態を保持する．交換可能状態の保持には，**表 3** に示す 2 つの変数を用いる．2 つの変数は，プロセスの情報を管理する task_struct 構造体のメンバである．ここで，2 つの変数を用いた交換可能状態の保持の処理流れを**図 3** に示し，以下で説明する．図 3 は，交換を要求してから交換対象プロセスが共に交換可能状態になるまでの流れを示す.

表 3 交換可能状態判定のため各プロセスへ追加した変数

形式	説明
`pid_t swap_dest`	交換を要求されている場合，もう一方の交換対象プロセスの PID を保持する．要求されていない場合，0.
`int swap_ready`	プロセスが交換を待機しているか否かを示す．待機している場合，1，していない場合，0 を保持する.

(1) 交換要求プロセスから交換対象プロセス 1 と交換対象プロセス 2 を指定し，交換を要求する．このとき，交換要求システムコールの引数に各交換対象プロセスの PID を引数として渡す.

(2) カーネルは，交換対象プロセス 1 の `swap_dest` に交換対象プロセス 2 の PID である p_2 を，交換対象プロセス 2 の `swap_dest` に交換対象プロセス 1 の PID である p_1 を格納する．これにより，交換対象プロセスは交換を要求されていることになる.

(3) 交換対象プロセス 1 はシステムコールを発行する．このとき，カーネルは，交換対象プロセス 1 の `swap_dest` の値を確認する．これにより，交換対象プロセス 1 が交換を要求されているか否かを判定する．そして，交換を要求されていた場合，カーネルは，交換対象プロセス 2 の `swap_ready` の値を確認する．これにより，交換対象プロセス 2 が既に交換を待機しているか否かを判定する.

(4) 交換対象プロセス 2 はまだ待機していないため，カーネルは，交換対象プロセス 1 を待機させる．ここで，交換のための待機は，入出力対象の交換完了まで走行を再開されてはならない．このため，割込による走行再開はできないことが好ましい．Linux の場合，TASK_UNINTERRUPTIBLE 状態である．また，交換対象プロセス 1 が交換を待機していることを示すため，交換対象プロセス 1 の `swap_ready` の値を 1 にする.

(5) 交換対象プロセス 2 はシステムコールを発行する.

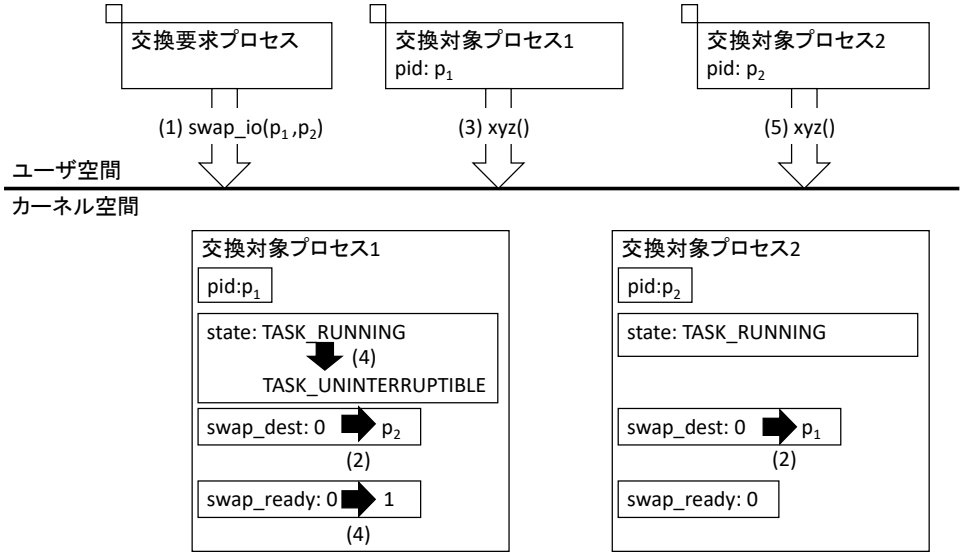

図 3 交換可能状態保持のための処理流れ

(3) の処理と同様に，カーネルは，交換対象プロセス 2 の swap_dest の値から交換対象プロセス 2 が交換を要求されているか否かを，交換対象プロセス 1 の swap_ready の値から交換対象プロセス 1 が既に交換を待機しているか否かを確認する．ここでは，交換対象プロセス 1 の swap_ready は 1 であり，交換対象プロセス 1 は既に待機している．これにより，この時点で両交換対象プロセスは交換可能状態になる．

3.4 処理流れ

本機能実現のため，交換可能状態の判定処理と入出力対象の交換処理を加えたシステムコールの処理流れを**図 4** に示し，以下で説明する．なお，図 4 に示した処理の内，追加した処理は (2) から (8) までの処理である．

(1) プロセスはシステムコールを発行する．
(2) システムコールを発行したプロセスは，交換を要求されているか否かを確認する．交換を要求されている場合，(3) の処理，要求されていない場合，(9) の処理へ遷移する．
(3) システムコールを発行したプロセスは，もう一方の交換対象プロセスが既に交換を待機しているか否かを確認する．待機していない場合，(4) の処理，待機している場合，(5) の処理へ遷移する．
(4) システムコールを発行したプロセスは走行を停止し，交換完了まで待機する．交換完了後，システムコールを発行したプロセスは走行を再開させられ，(9) の処理に遷移する．
(5) 待機している交換先のプロセスと自身の fd テーブルを比較する．比較の結果，利用している fd の数と数値が一致することを確認する．一致すれば，(6) の処理，一致しなければ，(7) の処理へ遷移する．
(6) システムコールを発行したプロセスは，もう一方の交換対象プロセスと自身の fd テーブルを交換し，入出力対象を交換する．
(7) 交換要求プロセスの走行を再開させる．
(8) もう一方の交換対象プロセスの走行を再開させる．
(9) (1) の処理で発行したシステムコールを実行し，返り値を返却する．

4. 評価

4.1 観点と環境

本評価では，入出力対象スワップ機能に関して以下の観点から評価を行う．また，評価環境を**表 4** に示す．

(観点 1) 交換処理時間

交換処理時間とは，図 4 に示した交換処理の処理時間である．具体的には，交換対象プロセス 2 が交換可能状態であると判定されてから交換完了し，待機しているプロセスを再開させるまでの時間である．入出力対象スワップ機能では，プロセス間で fd テーブルを交換することで入出力対象を交換する．このため，プロセスがいくつ入出力対象を持っていても処理内容は変わらない．よって，交換処理時間は一定となる．そこで，この時間を測定し，入出力対象スワップ機能はプロセスの持つ入出力対象の数に影響を受けず入出力対象を交換できできることを評価する．

(観点 2) 入出力対象スワップ機能によるオーバヘッド

入出力対象スワップ機能では，各プロセスの発行した全てのシステムコールの処理中に交換を要求されているか否かを判定する．このため，機能導入前に比べ，全てのシステムコールの処理にオーバヘッドが加わる．ただし，交換対象ではないプロセスのシステムコールに加わる処理は，条件分岐 1 回であるため，小さいと

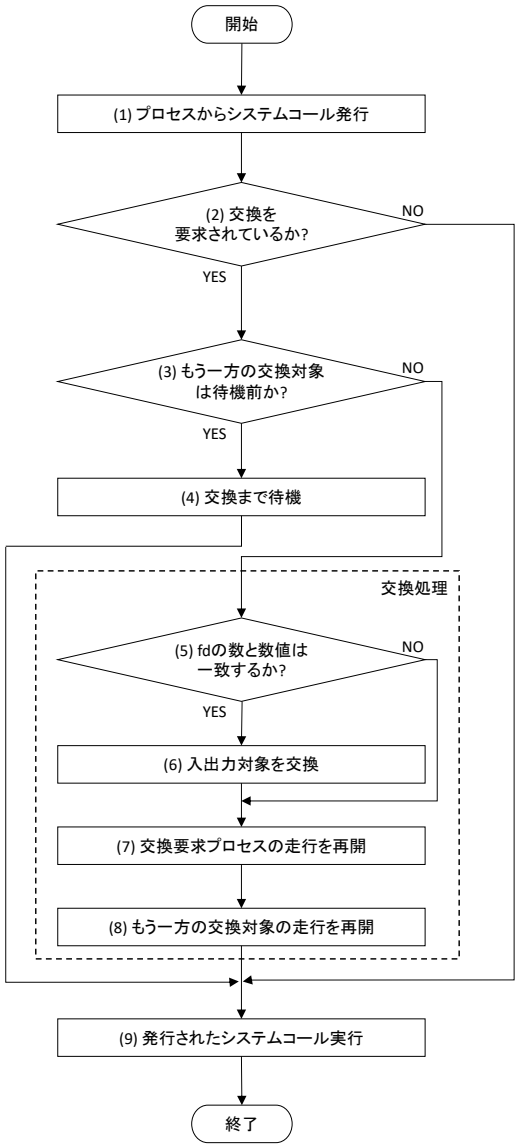

図 4 交換可能状態の判定処理と入出力対象の交換処理を加えたシステムコールの処理流れ

考える．そこで，このオーバヘッドは十分に小さいか否かを評価する．

表 4 評価環境

項目名	環境
OS	Debian 7.10
カーネル	Linux 3.0.8 64bit
CPU	Intel(R) Core(TM) i7/2.93GHz
メモリ	2GB

4.2 交換処理時間

4.2.1 測定方法

交換処理時間を測定し，入出力対象スワップ機能は，プロセスの持つ入出力対象の数に影響されず，入出力対象を交換できることを評価する．測定では，同じ処理を繰り返す 2 つの交換対象プロセスの間で入出力対象を交換した．

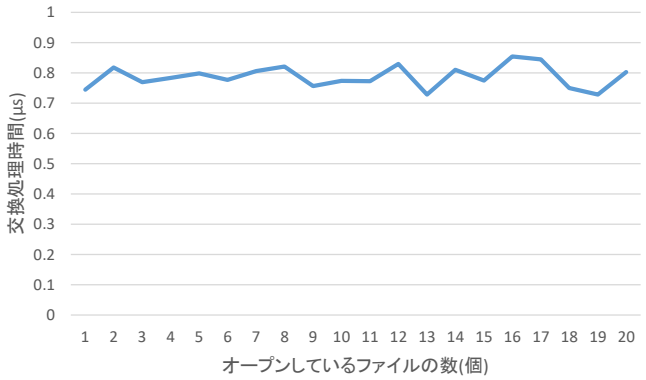

図 5 オープンしているファイル数を変化させた際の交換処理時間

このとき，交換処理時間を測定した．ここで，交換対象プロセスは，100μs のループ処理と `getpid()` システムコールの発行を繰り返す．また，このプロセスは，プロセスの生成時に指定した数のファイルをオープンする．そして，入出力対象の交換を 20 回行い，測定した平均値を測定結果とした．これをオープンするファイルの数を 1 個から 20 個まで変化させて測定した．

4.2.2 結果

4.2.1 項で述べた測定方法を基に測定した結果を**図 5** に示す．図 5 から交換処理時間は，約 0.73μs から約 0.85μs までの約 0.1μs の間に収まっており，ほぼ一定である．このことから，入出力対象スワップ機能は，交換対象プロセスの持つ入出力対象の数に影響を受けず，一定の時間で入出力対象を交換できることが分かる．

4.3 入出力対象スワップ機能によるオーバヘッド

4.3.1 測定方法

入出力対象スワップ機能の導入によりシステムコールに加わるオーバヘッドを評価する．そこで，入出力対象スワップ機能導入前後のカーネルにおけるシステムコールの処理時間を測定する．ここでは，`getpid()` システムコールを測定対象とする．なお，システムコールの発行から返り値の取得までを処理時間とし，20 回測定した平均値を測定結果とした．

4.3.2 結果

4.3.1 項の測定結果を**表 5** に示す．表 5 によると，入出力対象スワップ機能の導入により全てのシステムコールには 0.005μs のオーバヘッドが加わる．これは変更前のカーネルにおけるシステムコールの処理時間に対して，`getpid()` システムコールの場合，約 8 %である．`getpid()` システムコールは最も計算量の小さいシステムコールであるため，システムコールにかかるオーバヘッドは最大約 8 %となる．この結果，システムコールに加わるオーバヘッドは十分に小さいと言える．

表 5 入出力対象スワップ機能導入前後のシステムコールの処理時間の測定結果 (μs)

カーネルの状態	処理時間
実装前	0.060
実装後	0.065
オーバヘッド	0.005

5. おわりに

2 プロセス間における入出力対象スワップ機能を提案した．入出力対象の交換を可能にするため，プロセスが入出力対象を交換可能な状態を明らかにし，交換可能な状態の判定方法を示し，2 つのプロセスが共に交換可能な状態となることを保証する方法を述べた．交換可能な状態か否かは，プロセスの待機の契機，実行コンテキスト，および実行中の処理内容からプロセスが実行中の処理において fd を用いているか否かを区別することで決まる．交換可能な状態の判定は，システムコールの発行を契機に行う．また，2 つの交換対象プロセスのうち，先に交換可能状態であると判定できたプロセスの走行を停止させ，もう一方のプロセスの交換可能状態の判定を待機させる．これにより，2 つの交換対象プロセスは共に交換可能状態になることを保証する．

評価の結果，入出力対象スワップ機能は，交換対象プロセスの持つ入出力対象の数に影響を受けず，一定の時間で入出力対象を交換できる．また，入出力対象スワップ機能導入によりシステムコールに加わるオーバヘッドは，最大約 8 %であり，十分に小さい．

今後は，fd 単位での入出力対象の交換の実現やマルチスレッドプロセス間での入出力の交換方法の検討をする必要がある．

参考文献

[1] 鈴木森羅，乃村能成，谷口秀夫：走行中プロセスを他プロセスから複製する機能の提案，情報処理学会研究報告，Vol. 2017-OS-140, No. 4, pp. 1–7 (2017).
[2] 鈴木森羅，乃村能成，谷口秀夫：プロセス複製機能における被複製プロセスの走行を停止させる方法の提案，平成 29 年度 (第 68 回) 電気・情報関連学会中国支部連合大会講演論文集，pp. 1–2 (2017).
[3] 市川優平，乃村能成：ゲームの途中状態を複製・共有するシステムの提案，情報処理学会第 78 回全国大会講演論文集，Vol. 3, pp. 23–24 (2016).
[4] UNIX-domain protocol family, FreeBSD Online Manual (online), available from ⟨https://www.freebsd.org/cgi/man.cgi?query=unix⟩ (accessed 2017-1-12).
[5] Kaminsky, M., Peterson, E., Giffin, D. B., Fu, K., Mazires, D. and Kaashoek, M. F.: REX: Secure, extensible remote execution, *Proceedings of the General Track: 2004 USENIX Annual Technical Conference*, pp. 199–212 (2004).
[6] Provos, N., Friedl, M. and Honeyman, P.: Preventing privilege escalation, *12th USENIX Security Symposium*, pp. 231–242 (2003).
[7] Bernaschi, M., Casadei, F. and Tassotti, P.: SockMi: a solution for migrating TCP/IP connections, *Parallel, Distributed and Network-Based Processing, 2007. PDP '07. 15th EUROMICRO International Conference on*, pp. 221–228 (2007).
[8] Sultan, F., Srinivasan, K., Iyer, D. and Iftode, L.: Migratory TCP: connection migration for service continuity in the Internet, *Proceedings 22nd International Conference on Distributed Computing Systems*, pp. 469–470 (2002).
[9] Elhage, N.: reptyr, GitHub, Inc. (online), available from ⟨https://github.com/nelhage/reptyr⟩ (accessed 2017-1-24).
[10] Habets, T.: injcode, GitHub, Inc. (online), available from ⟨https://github.com/ThomasHabets/injcode⟩ (accessed 2017-1-24).