Vol.2019-OS-145 No.1
2019/2/28

入出力対象交換機能を利用したストリームデータ処理の実現

鈴木 森羅[1]　乃村 能成[1]　谷口 秀夫[1]

概要：ストリームデータ処理では，IoT，金融取引，および通信などにおいて継続的に発生する時系列データをリアルタイムに処理する．これを実現する際，Apache Storm のようなフレームワークを導入する必要があり，フレームワークに合わせた処理の実装が新たに必要となる．ここで，我々は，2 プロセス間において入出力対象を交換する機能として，入出力対象交換機能を提案した．この入出力対象交換機能を利用することで，例えば，パイプでつながれた 2 つのプロセスの間へ容易に別のプロセスを挟める．これによって，既存のプロセスに意識させず，プロセスの入出力対象を交換しデータの流れを制御したり，データを加工したりできる．このため，簡単に動的なストリームデータ処理を実現できる．

1. はじめに

ビッグデータに注目が集まるとともにこれを処理する技術も活発に研究開発されている．そして，ビッグデータの中でも継続的かつ無制限に生成されるものはストリームデータと呼ばれ，これを対象とした処理技術はストリームデータ処理と呼ばれる．ストリームデータは，例えば，センサーデバイスの計測データ，金融取引の情報，交通情報，および Twitter[1] や Facebook[2] といった SNS の投稿などである．ストリームデータ処理では，継続的に受信する 1 つ，またはいくつかのデータを対象に単純な処理を行い，短い遅延時間で結果を出力する．また，ストリームデータ処理は，簡単な処理を行う複数のプロセスの組み合わせで構成される．これらのプロセスは，プロセス同士でネットワークを構築し，受信したデータを処理し次のプロセスに送信する．ストリームデータ処理は多くの場面で利用されており，例えば，YouTube[3] のような動画配信サービス，交通渋滞の情報提供 [4]，およびシステムの異常予兆の検知 [5] がある．

ストリームデータ処理を実現するために，Apache Storm[6]，Apache Flink[7]，および Amazon Kinesis Streams[8] のような既存のフレームワークを導入することがある．この場合，開発者はフレームワークに合わせた処理を既存のプログラムへ新たに実装しなければならない．その際，フレームワークの使用方法を学習する必要があり，学習コストが高い．

そこで，入出力対象交換機能 [9] を利用したストリームデータ処理の実現について述べる．ここで，入出力対象交換機能は 2 つのプロセス間でそれぞれの利用している入出力対象を交換する機能である．入出力対象とは，ファイルや通信といった入出力の対象となる資源である．そして，カーネルは，プロセスごとに入出力対象に対応するファイル記述子 (fd) を割り当てる．入出力対象交換機能では，この fd に割り当てられた入出力対象を 2 プロセス間で交換する．入出力対象交換機能を利用すれば，例えば，パイプで接続された 2 つのプロセスの間に別のプロセスを追加できる．このようにプロセスの入出力先を変更することで，入出力対象交換機能はストリームデータ処理におけるデータの流れを制御できる．さらに，ストリームデータ処理は追加されたプロセスにおいてデータを加工できる．つまり，入出力対象交換機能はパイプやソケットといった UNIX 既存の通信方法でプロセス同士のネットワークを構築し，ストリームデータ処理を実現できる．また，入出力対象交換機能による入出力対象の交換は，プロセスのプログラムに変更を必要とせず，プロセスに交換を意識させず行われる．したがって，入出力対象交換機能を利用すれば，ストリームデータ処理は UNIX 既存の通信を利用し簡単かつ動的に実現できる．

2. ストリームデータ処理

2.1 ストリームデータ処理とは

ストリームデータは継続的に発生する時系列データである．そして，ストリームデータ処理はストリームデータを短い遅延時間で処理し続けるものである．ここで，ストリームデータ処理の具体的な利用例を**図 1** に示す．図 1 では，温度センサの計測データを，ストリームデータ処理を利

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

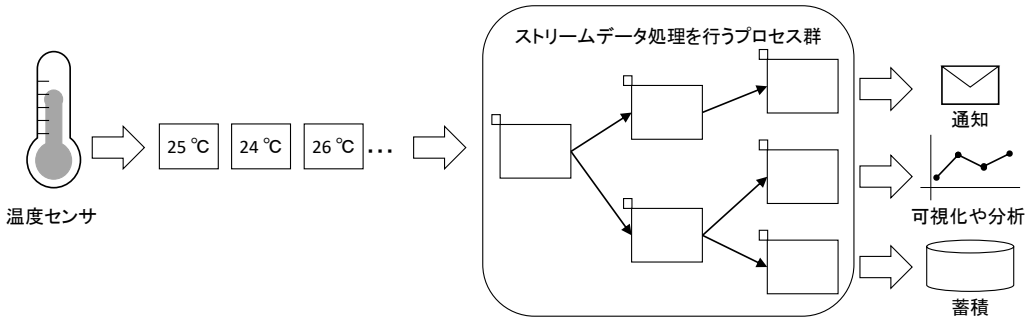

図 1 ストリームデータ処理の利用例

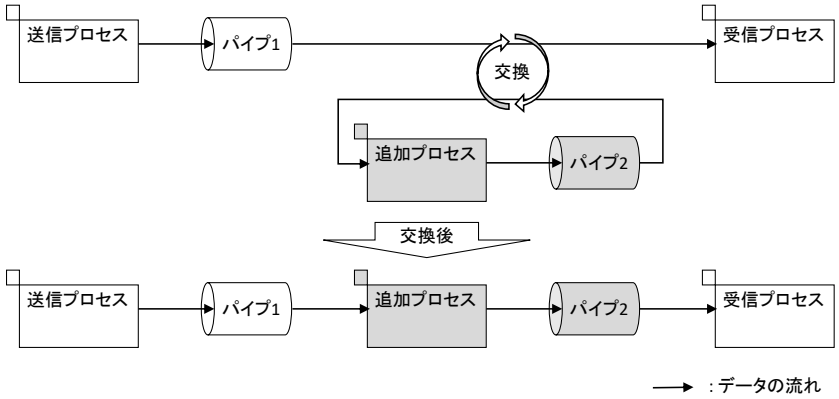

図 2 入出力対象交換機能を利用して 2 つのプロセスの間に別のプロセスを追加する様子

用して処理する流れを示す．温度センサは定期的に温度を計測し，データを出力し続ける．ストリームデータ処理を行うプロセス群は，プロセス間でデータのやり取りを行うネットワークを構築している．また，各プロセスはデータを受信し，加工を行い，次のプロセスへデータを送信する．これらのプロセスによる処理の結果，ストリームデータ処理は温度がしきい値を超えていた際に通知したり，データをグラフにプロットして可視化や分析をしたり，データを蓄積したりする．

2.2 課題と対処

ストリームデータ処理の課題は，ストリームデータ処理を行うシステムを開発する際，Apache Storm[6] のような既存のフレームワークの導入を必要とすることである．このため，開発者はプログラムへフレームワークに合わせた処理を新たに実装したり，開発や運用のためフレームワークのマニュアルを読み利用方法を学習したりする必要がある．

そこで，課題への対処として，これまでに我々の提案している入出力対象交換機能を利用したストリームデータ処理を実現する．ここで，入出力対象交換機能とは，プロセスの利用するファイル記述子 (fd) の参照先をプロセス間で交換することで入出力対象を交換する機能である [9]．入出力対象交換機能を利用すれば，例えば，パイプで接続した 2 つのプロセスの間に別のプロセスを挟み込める．ここで，2 つのプロセスの間に別のプロセスを追加する様子を**図 2** に示す．図 2 では，追加プロセスはパイプ 1 で接続している送信プロセスと受信プロセスの間に追加される．つまり，パイプ 1 で接続された送信プロセスと受信プロセスは，送信プロセスと追加プロセスをパイプ 1 で，追加プロセスと受信プロセスをパイプ 2 で接続している状態になる．各プロセスの入出力対象の利用状況として，送信プロセスはパイプ 1 の書き込み側，受信プロセスはパイプ 1 の読み出し側，追加プロセスはパイプ 2 の書き込み側と読み出し側を利用している．このような状態において，受信プロセスは，入出力対象交換機能により，自身の利用するパイプ 1 の読み出し側と追加プロセスの利用するパイプ 2 の読み出し側を交換される．これにより，受信プロセスはパイプ 2 の読み出し側，追加プロセスはパイプ 1 の読み出し側とパイプ 2 の書き込み側を利用する．このため，送信プロセスによりパイプ 1 に書き込まれたデータは，追加プロセスによりパイプ 1 から読み出され，パイプ 2 へ書き込まれる．その後，パイプ 2 へ書き込まれたデータは受信プロセスによりパイプ 2 から読み出される．図 2 のように入出力対象を交換しデータの流れを変更することで，入出力対象交換機能はストリームデータ処理を行うプロセス群のネットワークを構築できる．入出力対象交換機能は，プログラムへの変更を必要とせず，プロセスへ交換を意識させない．また，入出力対象の交換は UNIX 既存の構造体を利用して行われるため，プロセスはプロセス間の通信におい

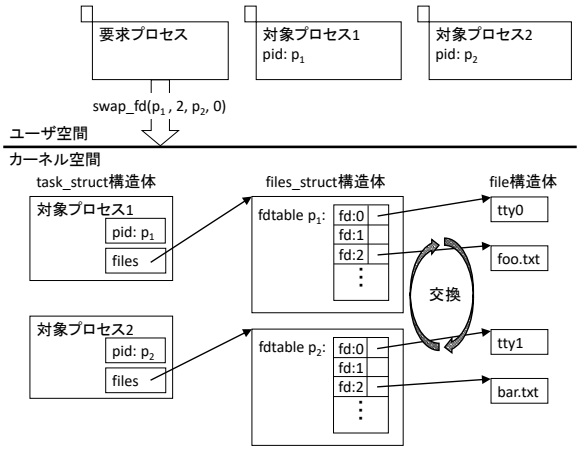

図 3 入出力対象交換機能による入出力対象の交換

て UNIX 既存の通信を利用できる．このため，開発者は新たに知識を学習する必要はない．したがって，入出力対象交換機能を利用すれば，ストリームデータ処理は容易かつ動的に実現できる．

3. 入出力対象交換機能

3.1 機能

入出力対象交換機能は，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，プロセスの利用している fd の参照先を 2 つのプロセス間で交換する．

本機能の交換の様子を**図 3** に示す．図 3 は交換対象プロセス 1 と 2 の間で入出力対象の交換される様子を示す．ここで，図 3 において，走行しているプロセスは入出力対象を交換される交換対象プロセス 1 と 2，および入出力対象の交換をカーネルに要求する交換要求プロセスである．また，交換対象プロセス 1 と 2 の PID はそれぞれ p_1 と p_2 である．さらに，交換対象プロセス 1 は fd の 0 と 2 を使って tty0 と foo.txt を，交換対象プロセス 2 は fd の 0 と 2 を使って tty1 と bar.txt を利用している．このような状態において，交換要求プロセスは交換対象プロセス 1 の fd の 2 と交換対象プロセス 2 の fd の 0 から参照される入出力対象の交換を要求する．入出力対象の交換の要求は交換要求システムコールの発行により行われる．交換の結果，交換対象プロセス 1 は fd の 0 と 2 を使って tty0 と tty1 を，交換対象プロセス 2 は fd の 0 と 2 を使って foo.txt と bar.txt を利用できる状態になる．

3.2 インタフェース

入出力対象交換機能を利用するためのインタフェースは交換要求システムコールであり，この仕様を**表 1** に示す．交換要求システムコールは，2 つの走行中のプロセスの PID とそれぞれのプロセスで利用している fd を 1 つずつ，合計 4 つの引数で，呼び出される．ここで，交換要求システムコールの処理流れを以下で説明する．

表 1 交換要求システムコールの仕様

形式	`int swap_fd(pid_t pid1, int fd1, pid_t pid2, int fd2)`
引数	`pid_t pid1`: 交換対象プロセスの PID `int fd1`: `pid1` で指定したプロセスの交換する入出力対象の fd `pid_t pid2`: 交換対象プロセスの PID `int fd2`: `pid2` で指定したプロセスの交換する入出力対象の fd
返り値	成功:1 失敗:0
機能	`pid1` のプロセスの `fd1` に割り当てられた入出力対象と `pid2` のプロセスの `fd2` に割り当てられた入出力対象を交換する．また，`fd1` と `fd2` の値が共に-1 である場合，`pid1` と `pid2` の 2 つのプロセスの利用している入出力対象をすべて交換する．

(1) 交換要求システムコールは，引数として渡された交換対象プロセスの PID と入出力対象の fd の値をもとに交換を要求する．
(2) 交換要求プロセスは，交換完了まで走行を停止され，待機する．
(3) 交換が完了すると，交換要求プロセスは走行が再開され，交換要求システムコールは交換要求プロセスへ返り値を返却する．

また，引数として渡す fd の値を共に-1 とした場合，入出力対象交換機能は，2 つの交換対象プロセスの利用するすべての入出力対象を交換する．

3.3 交換可能状態の判定

3.3.1 交換可能状態

交換対象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，fd は処理の途中で割り当てられた入出力対象を変えられてしまい，プロセスは動作に不具合を発生しかねない．このため，入出力対象交換機能では，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を交換可能な状態とする．

3.3.2 判定の契機

入出力対象交換機能は交換対象プロセスのシステムコールの発行を契機に利用して，交換可能な状態であるか否かを判定する．システムコールの発行直後は，ユーザ空間からカーネル空間に処理が遷移した直後であり，カーネルは発行されたシステムコールの処理を呼び出すための事前処理を実行している．この処理は，入出力を行わず，fd を利用していない．このため，たとえ fd を利用するシステムコールであっても，システムコール発行直後は fd を利用した処理は実行されておらず入出力対象は交換可能である．

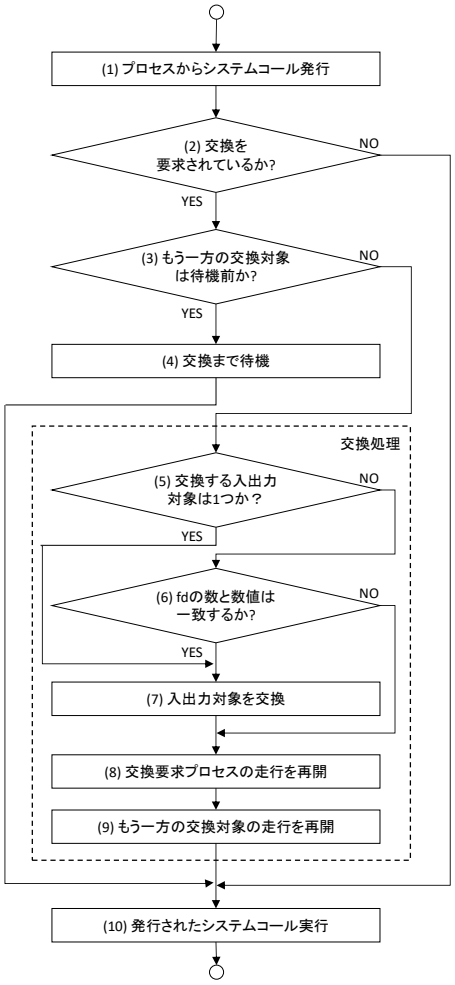

図 4 交換可能状態の判定処理と入出力対象の交換処理を加えたシステムコールの発行から実行までの処理流れ

3.4 有限時間内での交換の保証

入出力対象を交換するには，2 つの交換対象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，一方は交換可能状態であり，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，前者のプロセスは交換可能状態である保証はない．このため，2 つのプロセスはともに交換可能状態にならず，入出力対象交換機能は有限時間内に入出力対象を交換できる保証はない．そこで，2 つの交換対象プロセスのうち，カーネルは先に交換可能状態になっ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．つまり，カーネルは先にシステムコールを発行したプロセスの走行を停止させ，待機させる．これにより，入出力対象交換機能はもう一方のプロセスも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．

3.5 処理流れ

入出力対象交換機能では，プロセスがシステムコールを発行してから発行されたシステムコールの処理を実行するまでの間に，交換可能状態の判定処理と入出力対象の交換処理を追加した．これらの処理を加えたシステムコールの発行から実行までの処理流れを**図 4** に示し，以下で説明する．なお，図 4 に示した処理の内，(2) から (9) までの処理を追加した．

(1) プロセスはシステムコールを発行する．入出力対象交換機能は発行されたすべてのシステムコールを契機であるか否か判定する．このとき，プロセスの処理はカーネル空間に移り，以降の処理はカーネルによりおおなわれる．

(2) システムコールを発行したプロセスへ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れば (3) の処理，要求されていなければ (10) の処理へ移る．

(3) システムコールを発行したプロセスではない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) の処理へ移る．

(4) システムコールを発行したプロセスの走行を停止させ，交換完了まで待機させる．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，(10) の処理に移る．

(5) 交換する入出力対象が 1 つか否かを確認する．1 つであれば (7) の処理，全てであれば (6) の処理へ移る．

(6) 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，(7) の処理，一致しなければ，(8) の処理へ移る．

(7) 2 つの交換対象プロセスの指定された fd の参照先，または fd テーブルを交換し，入出力対象を交換する．

(8) 交換要求プロセスの走行を再開させる．

(9) もう一方の交換対象プロセスの走行を再開させる．

(10) (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．

3.6 ストリームデータ処理への利用

入出力対象交換機能はストリームデータ処理を実現する．図 2 のように入出力対象交換機能により，通信している 2 つのプロセスは，その間に別のプロセスを追加される．入出力対象交換機能は，このようにデータの流れを変更しプロセス群のネットワークを構築することでストリームデータ処理を実現する．

また，開発者は各プロセス間の通信方法と各プロセスの入出力を統一し，入出力対象交換機能を利用することで，ストリームデータ処理を行うプロセス群においてネットワークを容易に構築できる．つまり，ストリームデータ処理を行うシステムを容易に開発できる．例えば，プロセス間の通信方法は図 2 のようにパイプ，プロセスの入出力は標準入力からデータを読み込み標準出力から結果を出力するとする．すると，最初に標準入力からデータを読み込み

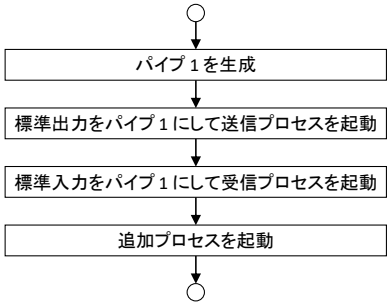

図 5 測定用のプログラムの処理流れ

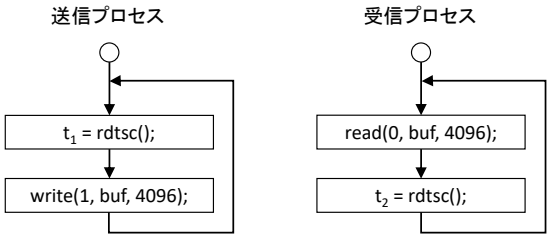

図 6 送信プロセスと受信プロセスの処理流れ

標準出力から結果を出力するというプログラムは各プロセスの処理を記述するテンプレートとなる．開発者はテンプレートを利用し，データの読み込みから結果の出力までの間にそれぞれの処理を記述する．最後に，入出力対象交換機能を利用して，プロセス間のネットワークを構築する．

ただし，入出力対象交換機能は，有限時間内での交換を保証するために先に交換可能状態と判定し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，交換を行う間，プロセスの処理は行われず，プロセス間の通信は影響を受ける可能性がある．

4. 評価

4.1 観点

本評価の評価観点は，入出力対象交換機能を利用したプロセスの追加によるデータ送受信時間の遅延の大きさである．データ送受信時間とは，データを送信するシステムコールの呼び出し直前から受信するシステムコールの返り値の返却直後までの時間である．入出力対象交換機能では，2 つの交換対象プロセスをともに交換可能状態にするため，先に交換可能状態となったプロセスの走行を停止させ待機させる．これにより，ストリームデータ処理において，交換対象プロセスの行う通信は一時的に止まる．したがって，データ送受信時間は延びる．そこで，実際に通信中のプロセスの間にプロセスを追加しデータ送受信時間を測定する．これを基に遅延は十分に小さいことを評価する．

4.2 環境

評価環境を**表 2** に示す．そして，図 2 のようにパイプで接続した 2 つのプロセスの間に入出力対象交換機能を用いて別のプロセスを追加する．また，送信プロセスは受信

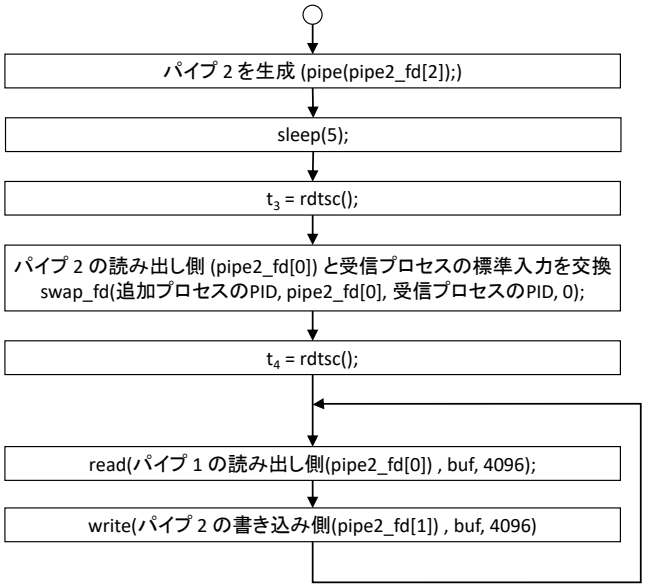

図 7 追加プロセスの処理流れ

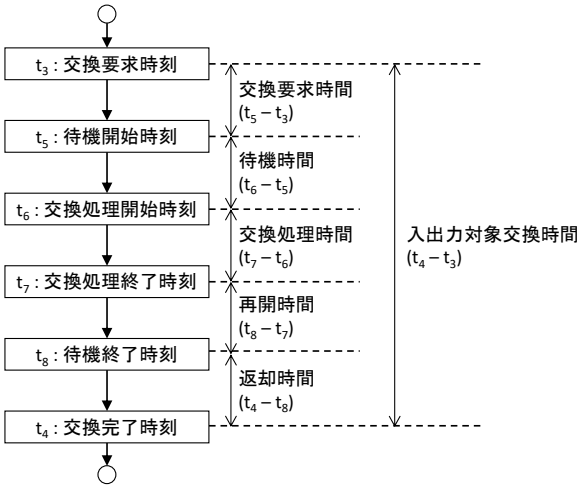

図 8 入出力対象交換機能の処理において記録した時刻と入出力対象交換時間の内訳

表 2 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
パイプのバッファサイズ	4KB

プロセスへ 4KB のデータを 5,000,000 回送信する．このとき，送信プロセスのデータの送信開始時刻と受信プロセスの受信完了時刻を記録する．これらの時間の差をデータ送受信時間とする．また，パイプのバッファサイズは 1 度の送信でパイプが満たされる 4KB とする．このため，送信と受信は必ず交互に行われ，データは 1 つずつ処理される．これにより，データ送受信時間へ他の処理の影響はなくなり，入出力対象交換機能を利用したプロセスの追加の遅延は明確になる．また，各プロセスは，バックグラウン

ドプロセスとして実行した測定用のプログラムにおいて，同じ優先度で起動される．さらに，プロセスの追加は，追加プロセスの起動から 5 秒後に行う．測定用のプログラムの処理流れを**図 5** に示し，以下で説明する．

(1) 送信プロセスと受信プロセスを接続するためにパイプ 1 を生成する．
(2) 標準出力をパイプ 1 の書き込み側にして送信プロセスを起動する．
(3) 標準入力をパイプ 1 の読み出し側にして受信プロセスを起動する．
(4) 追加プロセスを起動する．

また，送信プロセスと受信プロセスの処理流れを**図 6** に，追加プロセスの処理流れを**図 7** に示し以下で説明する．

(送信プロセス) 以下の処理を 5,000,000 回繰り返し，データを受信プロセスへ送信する．
(1) 送信開始時刻 t_1 を記録する．
(2) 標準出力へ 4KB (パイプのバッファサイズ) のデータを書き込み，データを送信する．

(受信プロセス) 以下の処理を 5,000,000 回繰り返し，送信プロセスからのデータを受信する．
(1) 標準入力から 4KB のデータを読み込み，データを受信する．
(2) 受信完了時刻 t_2 を記録する．

(追加プロセス) パイプ 2 の生成後，入出力対象交換機能を利用し自身を送信プロセスと受信プロセスの間に追加する．その後，送信プロセスから受信したデータを受信プロセスへ転送する．
(1) パイプ 2 を生成する．
(2) `sleep()` を使い，5 秒待機する．
(3) 交換要求時刻 t_3 を記録する．
(4) パイプ 2 の読み出し側 (pipe2_fd[0]) と受信プロセスの標準入力を交換する．これにより，追加プロセスは送信プロセスと受信プロセスの間に追加される．この結果，追加プロセスは pipe2_fd[0] を使ってパイプ 1 の読み出し側を利用できる．
(5) 交換完了時刻 t_4 を記録する．
(6) パイプ 1 の読み出し側 (pipe2_fd[0]) から 4KB のデータを読み込み，データを受信する．
(7) パイプ 2 の書き込み側 (pipe2_fd[1]) へ 4KB のデータを書き込み．データを送信する．

追加プロセスは送信プロセスからデータを送信されなくなるまで，(5) と (6) の処理を繰り返す．

送信開始時刻 t_1 と受信完了時刻 t_2 の差はデータ送受信時間 $(t_2 - t_1)$，交換要求時刻 t_3 と交換完了時刻 t_4 の差は入出力対象交換時間 $(t_4 - t_3)$ とする．また，入出力対象交換機能の処理において以下の時刻を記録した．

(待機開始時刻 t_5) 交換要求システムコールにおいて交換の要求後，交換の完了まで待機を開始する直前の時刻
(交換処理開始時刻 t_6) 図 4 の交換処理における (5) の処理の直前の時刻
(交換処理終了時刻 t_7) 図 4 の交換処理における (9) の処理の直後の時刻
(待機終了時刻 t_8) 交換要求システムコールにおいて交換の完了後，待機を終了した直後の時刻

なお，これらの時刻は，交換要求時刻 t_3 から交換完了時刻 t_4 までに含まれる．そこで，入出力対象交換機能の処理において記録した時刻を時系列順に並べ，**図 8** に示す．

表 3 入出力対象交換時間の内の各箇所の処理時間

	処理時間 (マイクロ秒)
交換要求時間 $(t_5 - t_3)$	2.5
待機時間 $(t_6 - t_5)$	2.5
交換処理時間 $(t_7 - t_6)$	0.6
再開時間 $(t_8 - t_7)$	0.8
返却時間 $(t_4 - t_8)$	0.1
入出力対象交換時間 $(t_4 - t_3)$	6.5

4.3 結果と考察

評価の結果，以下のことが分かった．

(1) 入出力対象交換機能によりプロセスを追加すると，追加直後に処理されたデータはデータ送受信時間へ約 3.2 マイクロ秒の遅延が発生する．これは**表 3** の交換要求時間，交換処理時間，および返却時間の合計である．遅延の発生は 1 度だけであり，これ以降にはない．また，プロセス追加前後のスループットは，プロセスの起動当初からパイプで接続していた場合と同じであり，入出力対象交換機能の利用による影響はない．
(2) 交換対象プロセスの待機はストリームデータ処理に影響しない．先に交換可能状態になった交換対象プロセスは，有限時間内の交換を保証するため，もう一方のプロセスも交換可能になるまで待機する．これにより，このプロセスの行う処理は遅延する．ただし，ストリームデータ処理は，この待機の間も他のプロセスにより処理を行われ，待機による影響はない．

以上のことから，入出力対象交換機能は，今回の条件では入れ替え時に約 3.2 マイクロ秒の遅延を許容するシステムにおいて適用可能である．

以下で詳細を述べる．4.2 節で測定したデータ送受信時間をグラフにプロットし**図 9** に示す．図 9 は横軸がデータ送受信列であり，縦軸がデータ送受信にかかった時間である．また，図 9 の (A) はプロセス追加後，追加プロセスの最初に処理したデータである．ここで，プロセス追加による遅延に注目するため，(A) の周辺を拡大したものを**図 10** に示す．図 10 によると，追加プロセスの最初に処理したデータのデータ送受信時間は延びており，約 20.5 マイクロ秒かかっている．また，プロセス追加後のデータ送受信時

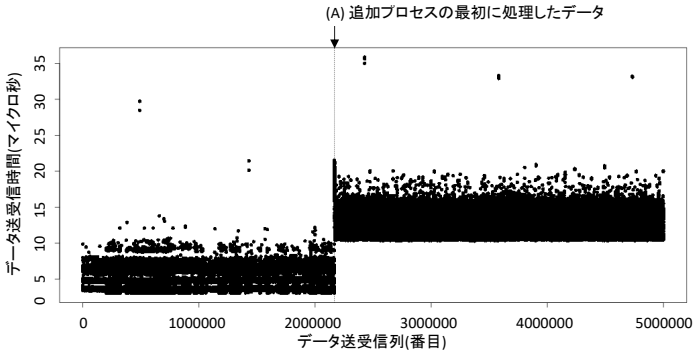

図 9 データ送受信時間

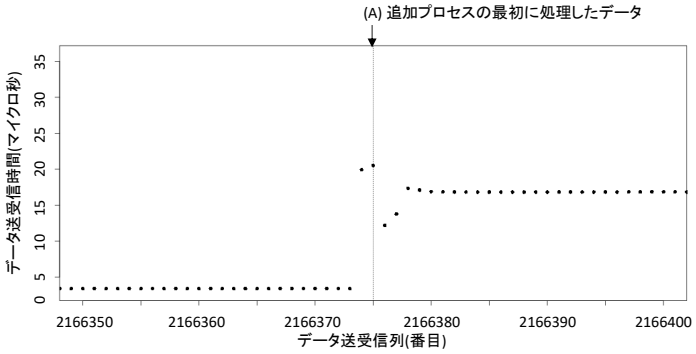

図 10 プロセス追加時点周辺のデータ送受信時間

間はと約 16.8 マイクロ秒である．このため，データ送受信時間の遅延は約 3.7 マイクロ秒である．ここで，入出力対象交換時間 $(t_4 - t_3)$ は約 6.5 マイクロ秒である．この交換時間を図 8 のように分け，以下で説明する．また，この処理時間の内訳を表 3 に示す．

(交換要求時間 $(t_5 - t_3)$**)** 交換要求システムコールを発行してから交換要求プロセスの走行を停止し待機するまでの時間

(待機時間 $(t_6 - t_5)$**)** 交換要求プロセスが待機してから交換対象プロセスがともに交換可能状態になるまでの時間

(交換処理時間 $(t_7 - t_6)$**)** 図 4 の交換処理の処理時間

(再開時間 $(t_8 - t_7)$**)** 交換を完了してから交換要求プロセスの走行を再開するまでの時間

(返却時間 $(t_4 - t_8)$**)** 走行を再開してから交換要求システムコールの返り値を返すまでの時間

これらのうち，交換要求時間，交換処理時間，および返却時間は一定である．一方，待機時間と再開時間は一定ではなく，プロセスの数，プロセスのシステムコールを発行する間隔，および交換を要求するタイミングに依存する．ただし，これらの時間の間，送信プロセスと受信プロセスは実行されているため，これらの時間はデータ送受信時間の遅延ではない．よって，入出力対象交換時間のうち，データ送受信時間の遅延は，交換要求時間，交換処理時間，および返却時間の合計約 3.2 マイクロ秒である．これは，追加プロセスの最初に処理したデータのデータ送受信時間の遅延と同程度である．この結果，入出力対象交換機能を利用してプロセスを追加すると，データ送受信時間はプロセスの追加後最初に処理したデータにおいて約 3.2 マイクロ秒遅延する．よって，入出力対象交換機能はこの遅延を許容するシステムにおいては，適用可能である．

約 3.2 マイクロ秒の遅延は，プロセス追加後のデータ送受信時間である約 16.8 マイクロ秒に対して約 20 %である．本評価の環境では，走行しているプロセスは 3 つ，各プロセスはデータ送受信のみを行い繰り返しシステムコールを発行している．このため，通信している 2 つのプロセスの間に別のプロセスを追加する場合，データ送受信時間は本評価の環境において最も短くなると考える．つまり，入出力対象交換機能による遅延は最大で約 20 %である．

また，約 3.2 マイクロ秒の遅延は，2 つの交換対象プロセスのうち後から交換可能状態になったプロセスにかかる遅延である．先に交換可能状態になったプロセスのデータ送受信時間は，さらに自身が交換可能状態になってからもう一方のプロセスも交換可能状態になるまでの時間も含む．ただし，ストリームデータ処理では，この待機の間も他のプロセスにより処理を行われる．このため，交換対象プロセスの待機はストリームデータ処理に影響しない．

なお，プロセスを追加する直前のデータのデータ送受信時間も延びており，約 19.9 マイクロ秒である．プロセス追加前のデータ送受信時間は約 3.4 マイクロ秒である．このため，プロセス追加直前のデータは約 16.5 マイクロ秒遅延している．ここで，このデータの送信開始時刻 t_1 と交換要

求時刻 t_3 の差は約 13.9 マイクロ秒である．これは，待機していた追加プロセスの走行を再開させるための遅延であると考える．また，受信完了時刻 t_2 は待機開始時刻 t_5 以降であり，データ送受信時間は交換要求時間を含み，約 2.5 マイクロ秒遅延する．

最後に，同様の環境でプロセス間通信のスループットを測定したところ，スループットはプロセス 2 つでの通信の場合約 23.9Gbps，3 つの場合約 7.6Gbps であった．これらは，入出力対象交換機能を利用せず，プロセスの起動当初からパイプで接続した場合と同じである．

5. 関連研究

5.1 ストリームデータ処理

ストリームデータ処理に関する既存技術として，処理を行うプロセス群のネットワークを構築/管理するフレームワークがある．このフレームワークには，Twitter など多くの事例で利用されている Apache Storm[6] がある．Apache Storm では，開発者はネットワークのトポロジを定義し，各プロセスの処理を記述することでネットワークを構築する．ただし，Apache Storm において，ストリームデータ処理は事前に定義されたトポロジで走行し，走行中の動的なトポロジの変更には対応していない．

5.2 入出力対象の変更

入出力対象交換機能と同様に走行中のプロセスの入出力対象を変更する技術について以下で述べる．

(1) プロセス間での fd の受け渡し
Free BSD では，プロセス間での UNIX ドメインソケットを介した fd の受け渡しは実現されている [10]．fd の受け渡しでは，プロセスは，自身の利用している入出力対象を参照している fd の値を他のプロセスに送信する．すると，送信された fd の参照する入出力対象は受信したプロセスの fd に割り当てられ，受信したプロセスでも利用される．

(2) プロセス間での TCP/IP 接続の受け渡し
プロセス間で TCP/IP 接続を受け渡す技術として，SockMi[11] や Migratory TCP[12] がある．具体的には，プロセスはソケットの状態を保存し，この保存したソケットの情報を他のプロセスへ送信する．そして，送信されたプロセスはこの情報を受けとり，ソケットの状態を復元する．

これらの技術はともに入出力対象を変更するプロセスの処理に変更を加える必要がある．

6. おわりに

入出力対象交換機能を利用したストリームデータ処理の実現について述べた．ストリームデータ処理は，プロセス同士でネットワークを構築し，プロセス間でデータをやり取りして順に処理を行う．そこで，入出力対象交換機能を利用することで，データの流れを制御しプロセス同士のネットワークを構築できることを示した．本稿では，データの流れを制御する例としてパイプで接続した 2 つのプロセスの間に別のプロセスを追加する様子について述べた．既存のプロセスに意識させない，UNIX 既存の通信を利用した簡単で動的なストリームデータ処理を実現した．

プロセスの追加によるデータ送受信時間の遅延と入出力対象交換時間を分析し，入出力対象交換機能による遅延を明らかにした．プロセス追加後，最初に処理されたデータのデータ送受信時間は約 3.2 マイクロ秒遅延し，これは交換の要求，入出力対象の交換，および交換要求システムコールの返り値の返却の処理時間の合計であった．この遅延は 1 度のみで，最初に処理されたデータ以降には発生しない．このため，入出力対象交換機能を利用したストリームデータ処理は，今回の条件では入れ替え時に約 3.2 マイクロ秒の遅延を許容するシステムにおいて適用可能である．

参考文献

[1] Twitter, Twitter, Inc. (online), available from ⟨https://twitter.com⟩ (accessed 2019-1-17).
[2] Facebook, Facebook, Inc. (online), available from ⟨https://www.facebook.com⟩ (accessed 2019-1-17).
[3] YouTube, Google (online), available from ⟨https://www.youtube.com⟩ (accessed 2019-1-17).
[4] 喜田弘司，中村暢達，今井照之，藤山健一郎：データストリーム処理機版を用いた交通渋滞情報提供システム，NEC 技法，Vol. 62, No. 3, pp. 110–113 (2009).
[5] 花森利弥，西村利浩：システムの異常予兆を検知するリアルタイム監視ソリューション，*FUJITSU*, Vol. 67, No. 2, pp. 23–28 (2016).
[6] Apache Storm, Apache Software Foundation. (online), available from ⟨storm.apache.org⟩ (accessed 2019-1-17).
[7] Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S. and Tzoumas, K.: Apache Flink: Stream and Batch Processing in a Single Engine, *IEEE Data Eng. Bull.*, Vol. 38, pp. 28–38 (2015).
[8] Amazon Kinesis, Amazon Web Services, Inc. (online), available from ⟨https://aws.amazon.com/kinesis/⟩ (accessed 2019-1-17).
[9] 鈴木森羅，乃村能成，谷口秀夫：2 プロセス間における入出力対象スワップ機能の提案，情報処理学会研究報告，Vol. 2018-OS-142, No. 1, 第 142 回システムソフトウェアとオペレーティング・システム研究会，pp. 1–8 (2018).
[10] UNIX-domain protocol family, FreeBSD Online Manual (online), available from ⟨https://www.freebsd.org/cgi/man.cgi?query=unix⟩ (accessed 2017-1-12).
[11] 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).
[12] 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).