Vol.2022-ARC-248 No.48
Vol.2022-SLDM-198 No.48
Vol.2022-EMB-59 No.48
2022/3/11

分散共有メモリ型通信機構 MBCF/Linux の実装について

松本尚 [1]　上村敬志 [2]　大家彩 [1]

概要：オリジナル並列分散 OS SSS-PC の基幹となる通信同期機構 MBCF の Linux OS 版である MBCF/Linux の開発を行った．高スループットと低レイテンシを両立し，高機能な共有メモリ操作を実現し，ロックフリーの不可分共有メモリ操作が可能な MBCF 通信機構を Linux 上で動かすことに成功した．本稿では，MBCF の特徴となるパケット受信割り込み時におけるユーザメモリ空間直接アクセス方式を始めとする，Linux における MBCF の実装方式について述べる．

キーワード：MBCF，MBCF/Linux，分散共有メモリ，通信同期機構，SSS-PC

Implementations of Memory-Based Communication Facility in the Linux kernel

TAKASHI MATSUMOTO[†1]　KEIJI UEMURA[†2]　AYA OYA [†1]

Abstract: We have developed MBCF/Linux, which is the Linux OS version of MBCF, which is the mandatory communication synchronization mechanism of our original parallel distributed OS SSS-PC. It achieves both high throughput and low latency and realizes high-performance shared memory operations, and also achieves lock-free atomic shared memory operations on Linux. In this paper, we describe the implementation method of MBCF in Linux, including the user memory space direct access method at the time of packet reception interrupt, which is a feature of MBCF implementation.

Keywords: MBCF，MBCF/Linux，Distributed Shared Memory，communication and synchronization mechanism，SSS-PC

1. はじめに

オリジナル並列分散 OS である SSS-PC[3] および SSS-CORE[2] の基盤となる通信同期機構として採用していた Memory Based Communication Facility (**MBCF**)[4][5]の Linux OS 版である MBCF/Linux を開発するとともに, MBCF を 64bit アドレスに対応させた MBCF64 を新たに規定して, 64bit アドレスの Linux OS にも対応可能にした．2021 年 12 月の報告[1]において，MBCF の概要を振り返り，64bit アドレス対応のための拡張方式を解説し，Linux 版の開発基本方針と実装時の注意点を述べた．本稿では，紙面と発表時間の都合で説明できなかった MBCF/Linux の実装の詳細について解説する. なお, 64bit アドレス対応の MBCF/Linux であることを明示したい場合は MBCF64/Linux と表記して, 従来の 32bit アドレス対応の Linux 版 MBCF であることを明示したい場合には MBCF32/Linux と表記する．なお，SSS-PC 版と Linux 版に共通な特徴や機能に言及する場合には，単に MBCF, MBCF32, MBCF64 と表記する．

2. MBCF の概要

MBCF は Gigabit Ethernet のような高速ネットワークで接続された PC クラスタやワークステーションクラスタにおいてマシン境界を越えた共有メモリアクセス環境を実現するための通信同期機構として 1994 年に松本によって考案され，1996 年に発表された[2]ものである． MBCF を実装するに当たっては，特殊なハードウェアを必要としない．通信のための API(Application Program Interface)が共有メモリ操作となっている点が，socket に代表されるこれまでの通信 API とは大幅に異なる点である．OS が提供する通信機構自体が共有メモリ操作を基本とする API を直接的に提供することにより，高機能，低遅延，低オーバヘッド，ロックフリーの通信同期機構の実現が可能になる．

MBCF は非常に多面的な特徴を持つため，概要を説明するだけで多くの紙面が割かれてしまう．より詳細な概要に関しては文献[1]を参照されたい．

3. MBCF/Linux の機種非依存の実装

3.1 Ethernet 周りのインタフェースに関して

SSS-PC/SSS-CORE OS では，Ethernet MAC チップのハードウェアレベルの割り込みをそのまま活かして MBCF の実装がなされている．このため，Ethernet MAC チップはパケット受信時に割込みを発生させるだけではなく，パケット送信完了時にも割込みを発生させる．しかし，Linux の抽象化された Ethernet 層である sk_buff 層のインタフェースを見てみると，パケット受信の割り込み時に呼び出す関数（厳密には，単なる関数ではなくタスクレット関数）は登録できるが，パケット送信完了時に呼び出す関数を定義するインタフェースは存在しない．厳密に言うと，ハードウェアレベルの割り込みは sk_buff 層内で扱われており，こ

1 奈良女子大学
Nara Women's University
2 三菱電機株式会社
Mitsubishi Electric Corp.

Vol.2022-ARC-248 No.48
Vol.2022-SLDM-198 No.48
Vol.2022-EMB-59 No.48
2022/3/11

のsk_buff層を利用してEthernet上のプロトコルはすべて記述されている．既存のLinux用Ethernet NICのデバイスドライバを活用するためには，このsk_buff層のインタフェースに従う他ない．このために，Linuxにおける Ethernet上のMBCFプロトコル記述では，送信完了の割り込み通知を知ることができない．SSS-PC/SSS-CORE OSでは，送信完了の割り込み時に，送信用パケットに使ったバッファ領域を再送に備えてキープするか再利用に回すか判断して，再利用時は削除して空きバッファとして登録している．この仕組みと同様の効果がsk_buffに対して実施できるように，パケット送信のセットアップ時にsk_buffの参照カウント（skbusers）を使って制御する必要がある．送信完了時にsk_buffが回収されないようにするためには，参照カウントを1以上にしておき，回収してもらいたいときは参照カウントを0にしておく．この修正により，sk_buffによるEthernetデバイスの抽象化の恩恵を享受した上で，送信完了時の割り込み通知が利用できないことに対応した．sk_buff層を使用するデバイスドライバとしてMBCF/Linuxを実装することによって，Linux用のデバイスドライバを持つすべてのEthernet NICを活用してMBCF通信機構がLinux上で使用できる．

3.2 特権モードにおけるユーザ権限下のメモリアクセス

MBCFでは，受信割り込みルーチン（Linuxでは受信割り込みで起動されるタスクレット）内で操作対象タスクのメモリに対して直接オペレーションを施すことにより，受信時のオーバヘッドコストとカーネル内のバッファ領域を大幅に削減している．この機能を実現するためには，二つの課題をクリアする必要がある．一つ目は，受信割り込みルーチン内で一時的に操作対象タスクのメモリ空間にコンテクストスイッチを行う方法の確立である．二つ目は，受信割り込みルーチン内は当然特権モードであるが，このルーチン内でMBCFのメモリ操作はユーザ権限で実行される必要がある．このためのメモリアクセス方法の確立である．本節では，二つ目の課題である特権モード内におけるユーザ権限によるメモリアクセス方法について述べる．

特権モード下においてユーザアクセス権限でメモリアクセスを実現する方法は，CPUによって千差万別である．このため，CPU機種依存コードとなることをMBCF/Linux開発開始当初は覚悟していた．しかし，Linuxカーネルの標準関数であるcopy_to_user(), copy_from_user()が正にこの用途に利用されていることに気がついた．システムコールを呼び出して，特権モードにモードを切り替えた後に，copy_from_user()によりユーザ空間のパラメータをユーザ権限で読み込み，copy_to_user()によりシステムコールの結果や情報をユーザ空間にユーザ権限で書き込みを行っている．メモリ保護違反が生じた場合は，命令例外トラップを引き起こすが，この二つの関数に関しては，命令例外トラップからEFAULT返り値を返す関数に復帰するようにコードが作成されている．より詳細には，これらの関数によってメモリ保護違反で命令例外トラップを起こす命令のアドレスがOSに登録されていて，これらのアドレスで起こった命令例外トラップはEFAULT返り値を返す関数に復帰し，例外が発生しなかったように振る舞う．copy_to_user()関数とcopy_from_user()関数はEXPORT_SYMBOL設定がなされているため，デバイスドライバから呼び出すことが可能である．しかし，受信割り込み時に起動される関数（タスクレット）から呼び出し可能かどうかは不明であったため，実際にテストコードを作ってパケット受信時にこれらの関数を呼び出してメモリアクセスを実行したところ，正常にユーザ空間のメモリを操作することが可能であった．このことから，これらの関数を使用することにより，特権モード下におけるユーザ権限によるメモリアクセスの実現には，CPU機種依存コードによって新規実装する必要はないと判断した．

3.3 MBCF_SIGNALの実装

MBCF_SIGNALは非同期のユーザ関数起動とFIFOキューによる要求元からのパラメータの受け渡しを組み合わせたものである．FIFOキュー操作は受信割り込みルーチン内における操作対象アドレス空間への切替とユーザ権限によるメモリアクセスが可能になれば，問題なく実現が可能である．Linuxにおいて実装が困難なのは，受信割り込みルーチンからの非同期のユーザ関数起動である．Unix/Linuxでこの動作に最も近いものはsignalであり，カーネルである条件が成立するとsignalが発生し，要因に応じたユーザ関数の起動が可能である．MBCF_SIGNALでは，ユーザメモリ上のMB_SIGNAL構造体ごとに非同期起動関数を定義可能なため，非同期起動関数の種類数に実質的な上限は存在しない．一方，Unix/Linux signalは全部で64種類しか定義されておらず，下位31種はsignalの発生回数と非同期起動の回数の一致が保障されない．上位33種（SIGRTMINからSIGRTMAX）はsignal発生回数と同数の非同期ユーザ関数の起動が保障される．64種をはるかに超える多種のユーザ関数の非同期起動をカーネルから直接可能にするという課題に対して，Unix/Linux signalは直接的には使えない．一般的に非同期にユーザ関数をカーネルから起動するコードは，CPU機種依存コードとならざるを得ないため，可能であれば，既存のsignal機構を利用して，MBCF_SIGNALの機能をLinuxに実現したい．このため，signal機構を使ってMBCF_SIGNALが起動すべきユーザ関数をディスパッチする関数（ユーザ関数）を呼び出す方式を取ることにした．ユーザプロセス内において関数呼び出しが一段増えることになるが，この程度の実行時のオーバヘッドコストは，機種依存コードを新規に開発する手間と比べて大きなものではないと考え，目をつぶることにした．Unix/Linux signalの最大の弱点は引数をほとんど伴えない点である．

Vol.2022-ARC-248 No.48
Vol.2022-SLDM-198 No.48
Vol.2022-EMB-59 No.48
2022/3/11

```
Nov  3 18:32:55 coop-i5-ubuntu kernel: [   432.602442]  MBCF_ARP_REPLY recvfrom p_id=1 00:1b:21:2f:17:df
Nov  3 18:32:58 coop-i5-ubuntu kernel: [   435.546253] ------------[ cut here ]------------
Nov  3 18:32:58 coop-i5-ubuntu kernel: [   435.546272] WARNING: CPU: 3 PID: 2036 at /build/
linux-4WcAio/linux-4.4.0/kernel/softirq.c:150 __local_bh_enable_ip+0x64/0x80()
Nov  3 18:32:58 coop-i5-ubuntu kernel: [   435.546277] Modules linked in: mbcf32(OE) nvidia_uvm(POE)
intel_rapl x86_pkg_temp_thermal intel_powerclamp coretemp kvm_intel kvm irqbypass eeepc_wmi
snd_hda_codec_hdmi asus_wmi sparse_keymap crct10dif_pclmul snd_hda_codec_realtek crc32_pclmul
snd_hda_codec_generic ghash_clmulni_intel snd_hda_intel aesni_intel aes_86_64 snd_hda_codec lrw
snd_seq_midi snd_hda_core
Nov  3 18:32:58 coop-i5-ubuntu kernel: [   435.546323] mbcf32: mcnt=64 plen=44 cmd=0x14000000 srcNode=1
srcPtask=_RFQKG(0x1234567)
```

リスト 1　syslog の warning

page fault (segmentation fault, SIGSEGV)時には，メモリアクセス違反の発生場所を伝えるために，signal を介してアドレス情報を受け渡す必要がある．そのために，struct siginfo が使用される．send_sig_info()関数を使用することにより，siginfo 構造体にセットした値をユーザの siginfo 構造体に受け渡してユーザ関数の非同期起動が実現されると思われた．しかし，siginfo 構造体のメンバの si_signo に SIGRTMAX (=64)を設定し，si_errno に 0 を設定し，si_code に適当な値を設定して send_sig_info()を呼び出しても，siginfo 構造体の _sifields 内の値がユーザ空間の siginfo に受け渡されなかった．Linux のソースコードを丹念に読むと，si_code に**負の値**を設定しない限り，_sifields 内の値がユーザ空間に受け渡されない仕様になっていた．si_code に負の値を設定し，_sifields 内にユーザ空間に伝えたいパラメータ（最終的に動かしたいユーザ関数のポインタを含む）をセットして，send_sig_info()を呼び出すことにした．非同期に起動されるユーザのディスパッチ関数を経由することにより，MBCF_SIGNAL で起動すべきユーザ関数を何の制限も無しに呼び出すことが可能になった．

Linux カーネル version 4.4 ではカーネル内もユーザ空間も siginfo 構造体は同一の形式であり，siginfo 構造体の _sifields は 128byte の領域があり，多くの情報をカーネル空間からユーザ空間に引き渡すことができた．しかし，Linux カーネル version 5.0 以降では，カーネル内では struct kernel_siginfo という新たな構造体が定義され，カーネル内の siginfo 構造体として使われる変更がなされている．kernel_siginfo 構造体の_sifields に相当する領域は 32byte しかなく，それ以前と同じ感覚で利用すると kernel_siginfo 構造体をはみ出して，カーネルを破壊する恐れがある．MBCF/Linux を Linux カーネル version 5.0 以降に移植する場合には，この点に注意を払う必要がある．

3.4 spin_lock 関数による排他区間の確保

SSS-PC の MBCF 実装コードと同様に MBCF/Linux ドライバの実装には，多数の排他区間（クリティカル区間）が存在する．排他性を保障するために，受信割り込みルーチン内において spin_lock を使用している．これは，受信割り込みルーチンが独立したプロセスではないので，spin_lock の使用はやむを得ないものと考えている．また，プロセスを寝せたり起こしたりするオーバヘッドがないため，コストは mutex 等と比べて低いとも考えていた．spin_lock によって排他区間内にいる時に，CPU が割り込まれた場合には，MBCF の処理が遅延を被る恐れがあるため，spin_lock の獲得は必ず割り込み禁止・禁止解除と組み合せて行っていた．より具体的には，spin_lock_irqsave()でロックを獲得し，spin_unlock_irqrestore()でロックを解放していた．この実装方針でほぼ問題なくコードが動いているように見えたが，一つの問題が解決されずに残っていることに気がついていた．それは，ethernet パケットを飛ばすための関数である dev_queue_xmit()を irqsave の割り込み禁止状態で呼び出すことに対する warning メッセージが syslog に吐かれていた(リスト 1).

MBCF/Linux ではパケットの到着保障と順序管理を行うために，宛先ノード単位の順序情報（通し番号）のコピーをパケット内に持たせて，dev_queue_xmit()が成功すると，ノードの順序情報を更新（パケット番号を+1）する．このため，dev_queue_xmit()の実行途中で順序情報が他のパケット送信や受信時に更新前の値が取得されたり，順序情報が変更されるとまずいため，順序情報に関するロックを取っている．ロック中に割り込まれると大きな遅延の原因となりうるので，当然，割り込みも禁止して排他制御を行っている．このため，安易に，dev_queue_xmit()を割り込み許可状態で呼ぶわけには行かない．SSS-PC では，パケット送信は，NIC HW の送信キューにパケットをセットするだけなので，割込み禁止状態で送信手続きを行うことに何の問題も生じない．どうやら，Linux の dev_queue_xmit()関数は irq 操作や bottom half の操作を伴うため，単純なキューへの登録操作ではなく，割り込み禁止状態で呼び出すことは望ましくないことが判明した．この問題を考えるに当たって，

Vol.2022-ARC-248 No.48
Vol.2022-SLDM-198 No.48
Vol.2022-EMB-59 No.48
2022/3/11

Linux が割り込みを top half と bottom half に分けて考えていることに，初めて注意が向いた．SSS-PC の MBCF の受信割り込みルーチンは純粋に割り込み処理内から直接関数呼び出しで呼ばれる関数であるが，Linux の ethernet の各種受信関数は受信割り込みで起動されるタスクレット（つまり，bottom half）であることに気がついた．これによって，受信割り込みルーチンとの排他処理は，spin_lock_irqsave()と spin_lock_irqrestore()の組み合わせではなく，spin_lock_bh()と spin_unlock_bh()の組み合わせで処理されるべきであると判明した．この変更により，syslog に MBCF/Linux 関連の warning メッセージとして出ていたメッセージを消すことに成功した．

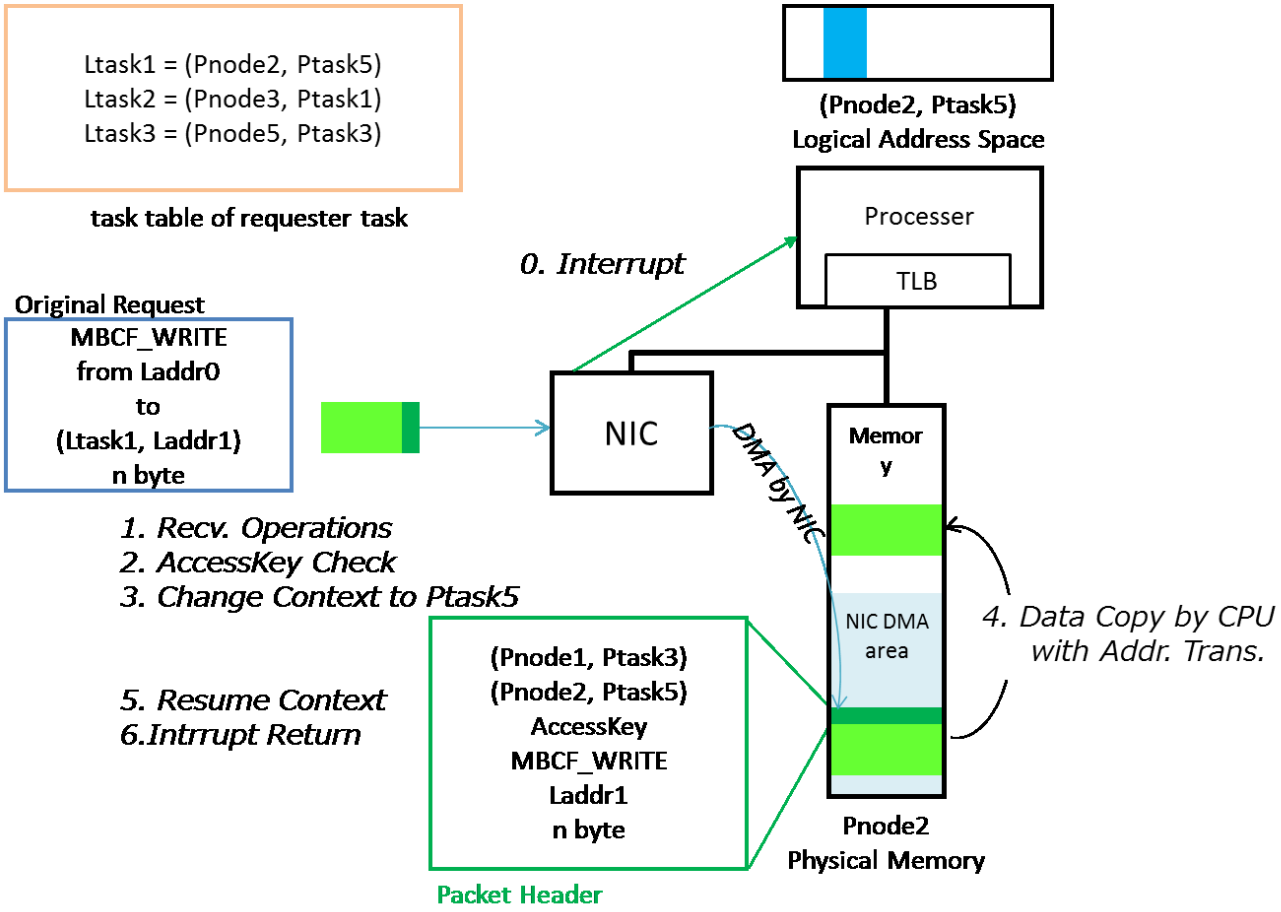

図 1 MBCF パケット受信時の動作（MBCF_WRITE のケース）

4. MBCF/Linux の CPU 機種依存の実装

4.1 パケット受信割り込み時の動作概要

受信割り込みルーチン内における操作対象タスク（プロセス）のアドレス空間への切替処理が CPU 機種依存コードとなる．詳細を説明する前に，MBCF パケットの受信割り込みルーチンの概要を本節では記述する．遠隔書き込みである MBCF_WRITE コマンドを例に挙げて説明する．MBCF 操作要求パケットは要求先ノード(ここでは Pnode2 とする)までネットワークによって運ばれてくる．パケット到着時に，要求先ノードの NIC は DMA によってパケットのデータイメージをリングバッファにコピーし，その後に要求先ノードのプロセッサにパケットが到着したことを知らせるために，割込み信号を発生させる．図 1 は要求先ノードにおけるパケット受信割込みルーチン内の MBCF 関連処理を示す．NIC からの受信割込み発生により，受信先ノードのプロセッサはパケット受信ルーチンに制御が切り替わり，NIC によって要求される低レベルのパケット受信手順を最初に実行する．もしも，受信したパケットが MBCF 要求パケットである場合には，割込みを抜けることなく引き続き MBCF 受信ルーチンを実行する．ただし，Linux ではタスクレットとして MBCF 受信ルーチンがスケジューリングされるが，特権モードの実行であり以下の議論は同じになる．MBCF 受信ルーチンでは，まず物理タスク ID（図中の Ptask5）から要求先タスク（プロセス）を特定する．具体的には，そのタスクのタスク構造体（UNIX/Linux ではプロセス構造体に相当）へのポインタを得る．次に，パケット内の AccessKey が要求先タスクのものと一致しているかどうかチェックして，一致している場合のみ要求先タスクのメモリ空間内のメモリ操作を許可する．一致した場合は，メモリ空間のコンテクストを要求先タスクのものに切り替えて（図中の 3.），パケットで運ばれてきた n バイトのデ

Vol.2022-ARC-248 No.48
Vol.2022-SLDM-198 No.48
Vol.2022-EMB-59 No.48
2022/3/11

ータを目的論理アドレス（Laddr1）から，特権レベルとしてではなくユーザ権限によって書き込む．メモリ操作終了後には，メモリ空間のコンテクストを割込み発生時のものに戻しておく（図中の 5.）．

4.2 Linuxにおけるメモリ空間切替処理

Linux において，プロセスメモリ空間の切替は必ずプロセス切替に付随する処理として行われており，メモリ空間切替のための CPU 依存のない関数は定義されておらず, 当然ながら，デバイスドライバでの利用のために公開されてもいない．このため，MBCF 受信ルーチン内に操作対象タスク（プロセス）のメモリ空間に切り替える CPU 依存コードを用意する必要がある．

4.3 TLB 全フラッシュを回避するアドレス空間 ID

1994 年に SSS-CORE OS において MBCF を最初に実装した CPU である SuperSPARC[6]はアドレス空間ごとにコンテクスト ID と呼ばれるアドレス空間 ID を持っており，現在のコンテクスト ID を設定するレジスタに操作したいアドレス空間の ID を設定することにより，アドレス空間切替時の TLB 全フラッシュを回避することが可能であった. つまり，TLB には論理アドレスのページ番号の他にコンテクスト ID もキャッシュされており，現在のコンテクスト ID とアクセスしようとする論理アドレスがキャッシュされたコンテクスト ID と論理アドレスページ番号の両方と一致した場合のみ，TLB ヒットとして扱っていた．このアドレス空間 ID のサポートは MBCF 受信ルーチンの高効率な実装に大きな役割を果たす．メモリ空間切替処理を MBCF 受信ルーチン内にコーディングするのであれば，アドレス空間 ID をサポートすることが望ましい. それどころか, Linux がアドレス空間 ID に基づくプロセスメモリ空間切替しかサポートしていなければ，MBCF 受信ルーチンでもアドレス空間 ID を必ずサポートする必要がある．

4.4 CPU ごとのアドレス空間 ID のサポート状況

MBCF/Linux が対象とする CPU は ARM の AArch32[7]と AArch64[8]，そして intel の x86_64（intel64[9]）である．他の CPU の上で動く Linux にも対応可能であるが, 現状ではこの 3 種の CPU アーキテクチャをターゲットとしている．

(1). intel x86_64 (intel64)

x86_64 CPU は IA-32[9]から拡張されてリリースされた当初は，メモリ管理ユニットにアドレス空間 ID に相当するものを持っていなかった．これは，IA-32 のメモリ管理ユニットの設計が古いことが大きな理由だと思われる．この ID の不存在はアドレス空間切替が必ず TLB 全フラッシュを伴うことを意味して, MBCF 実装上は非常に大きな欠点であると考えていた．しかし，アドレス空間切替は MBCF 以外でも頻繁に必要になるケースがあるため, 2010 年には x86_64 CPU にも 12-bit の process-context identifiers (PCIDs)として導入された．これにより intel x86_64 アーキテクチャへの MBCF 高速実装の大きな障害が取り除かれた．

(2). ARM AArch32 と AArch64

ARM CPU は ARMv6 アーキテクチャ以降のメモリ管理ユニット（MMU）において，Address Space ID (ASID)と呼ばれる 8bit のアドレス空間 ID をサポートしている．

4.5 CPU 機種依存のメモリ空間切替の実装

(1). intel x86_64 (intel64)

ubuntu 16.04 のカーネルである Linux カーネル version 4.4 のソースコードをプロセス切替と関連して追いかけると, switch_mm()という関数でアドレス空間を切り替えていることが判明した．x86_64 CPU 用の switch_mm()関数の中を読んで見ると，アドレス空間切替に関わっているのは x86_64 の CR3 レジスタの書き換えのみであることが判り，SSS-PC における IA32 CPU のアドレス空間切替手法と何ら差がないことが判明した．あるプロセスのアドレス空間に切り替えるためには，そのプロセスのプロセス構造体である task_struct の mm 構造体の中の pgd エントリを CR3 に設定すればよいことが判った．ただし，CR3 は物理アドレス（厳密にはリニアアドレス）で値を設定する必要があるのだが，Linux は全物理メモリを論理アドレス空間にマップするという「悪しき」伝統があるため，pgd には論理アドレスで値が格納されている．受信割り込みルーチンを抜ける前には，もちろん元通りのアドレス空間に CR3 のエントリを戻しておく. Linux カーネル version 4.4 は PCID に対応しておらず，32bit の IA32 と同様に，CR3 を完全に書き換えることで TLB を全消去してアドレス空間を切り替えている．Ubuntu 18.04 以降のカーネルでは，PCID をサポートしている．しかし，intel CPU のコンパチビリティの高さのおかげからか，ここで述べた PCID を無視する実装であっても無事に動作することを確認した．

なお，次に述べるように ARM CPU に対してはアドレス空間 ID に相当する ASID を Linux カーネル 4.1 でさえもサポートしている. x86_64 CPU と ARM CPU に対する実装の差異は, 想定する最大プロセス数によるものだと思われる．PCID は 12bit の bit 幅を持っているため，個々のプロセスに割り当てると，最大プロセス数が 4096（厳密には 4095）に限定されることになる．大規模なサーバマシンに採用されることが多い，x86_64 用の Linux にはこの制限が許容できないようである．Ubuntu 18.04 以降の PCID サポートは個々のプロセスに PCID を個別に対応させるのではなく，短い時間内に再スケジューリングされたプロセスにのみ PCID を対応させる実装が行われている．つまり，久しぶりにスケジュールされたプロセスは従来通り CR3 を書き換えることにより TLB 全フラッシュを伴ってアドレス空間が切り替わる．

(2). ARM AArch32

ターゲット評価ボードに対応した Linux カーネル 4.1 のソースコードの switch_mm()周辺の記述を読むと，AArch32

Vol.2022-ARC-248 No.48
Vol.2022-SLDM-198 No.48
Vol.2022-EMB-59 No.48
2022/3/11

に対しては，ASID を利用したアドレス空間の切替を行っていることが判明した．AArch32 の ASID は 8bit であるため，AArch32 用の Linux は同時には 256 個のアドレス空間しか効率良く保持できないことが判明した．リスト 2 に，Linux カーネル version 4.1 におけるアドレス切替コードを掲載する．このコードから，task_struct 構造体の中の mm_struct 構造体の context.id.counter に ASID が格納されており，ページテーブルの root pointer の上位 16bit にその ASID をセットして，root pointer として設定していることが判る．同じ働きをするコードを MBCF32/Linux デバイスドライバに用意して，AArch32 CPU のアドレス空間切替関数とした．もちろん，受信割り込みルーチン内において，MBCF のメモリ操作が終了した直後には，ページテーブルの root pointer（ASID を含む）は切替前の元の状態に戻しておく．

```
/* cpu_v7_switch_mm(pgd_phys, tsk)
 * Set the translation table base pointer to be pgd_phys (physical address of the new TTB).
 */
ENTRY(cpu_v7_switch_mm) !switch_mm(pgd,mm) pgdl:r0, pgdh:r1, mm:r2
        ldr     r2, [r2, #MM_CONTEXT_ID]
        and     r2, r2, #255
        orr     rpgdh, rpgdh, r2, lsl #(48 - 32)        @ upper 32-bits of pgd
        mcrr    p15, 0, rpgdl, rpgdh, c2                @ set TTB 0
        isb
        ret     lr
ENDPROC(cpu_v7_switch_mm)

./kernel/asm-offsets.c:   DEFINE(MM_CONTEXT_ID, offsetof(struct mm_struct, context.id.counter));
```

リスト 2　Linux version 4.1 の Aarch32 用のアドレス空間切替コード（一部編集済み）

(3). ARM AArch64

AArch32 と同様に ASID を利用したアドレス空間の切替を行っていることが判明し，同じ働きをするコードを MBCF64/Linux デバイスドライバに用意して, AArch64 CPU のアドレス空間切替関数とした.

4.6 task_struct 構造体経由の mm_struct 参照の問題点

簡単なテストプログラム程度では発生しないが，本格的に MBCF/Linux を使用するアプリケーションプログラム実行中に Linux ごとマシンがハングする現象が発生した．デバイスドライバのコードを書き換えながらデバッグを続けるうちに, このハングが, task_struct 構造体の中の mm_struct 構造体へのポインタが NULL になっていて，そのポインタを辿ろうとして page fault を受信割り込み内で発生させることによって起こっていることが判明した．mm_struct 構造体は task_struct 構造体内に埋め込まれる形で実装されているわけではなく, task_struct 構造体には mm_struct 構造体へのポインタが mm として格納されており，それを辿ることにより mm_struct 構造体内のメモリ管理情報に到達できる．ページテーブルの pgd(root pointer)や ASID といった基本となるメモリ管理情報はプロセスが生成されてから消滅するまで変化しないと考えられるため，mm_struct 構造体が途中で入れ替えられるケースを筆者らは想定しておらず, mm_struct 構造体へのポインタ mm は不変であると思い込んでいた．しかし，実際に並列アプリケーションプログラムを動かした経験から，このポインタ mm が NULL になる時間帯が存在することがあり，その時間帯に MBCF 操作要求パケットを受信するとポインタ mm を辿ることができなくてマシンがハングすることが判明した.

4.7 改良されたアドレス空間切替法

前節に述べた内容から task_struct 構造体内の mm_struct 構造体へのポインタである mm を辿ってメモリ管理情報に受信割り込みルーチン内でアクセスするという方式の実装には致命的な問題があることが明らかになった．Linux のメモリ管理の詳細を完全に把握できているわけではないが, 追加のメモリ領域や二次記憶への退避・復旧，mlock 情報なども mm_struct 構造体に格納されているものと予想される．そのため，プロセスが実行中に mm_struct 構造体を更新する必要があり，更新時にはポインタ mm を一旦 NULL にして古い構造体を破棄（free）して，更新後の値を持つ構造体を指すポインタと置き換えるという操作を行っているものと思われる．もしくは更新中はポインタを辿ってもらいたくないため，mm を NULL にしておいて mm_struct 構造体内の値が更新された後に，ポインタを再設定しているものと思われる．ただし，MBCF/Linux の受信割込みルーチンにおいて参照したいメモリ管理データはページテーブルの root pointer と ASID といったごく基本的なものだけである．これらの値がプロセス実行中に変化するとは考えにくいため，MBCF デバイスを open した時点で，これらの

Vol.2022-ARC-248 No.48
Vol.2022-SLDM-198 No.48
Vol.2022-EMB-59 No.48
2022/3/11

データのコピーを MBCF/Linux のためのタスク構造体（struct L_taskinfo）の一部として格納しておくことにした．この改良により，受信割り込みルーチン内で task_struct 構造体の mm ポインタを辿る必要がなくなり，マシンがハングすることもなくなった．逆に，L_taskinfo 構造体へのポインタが受信割り込みルーチンでは task_struct 構造体へのポインタよりも先に得られるため，そこから task_struct 構造体を辿り，さらに mm_struct 構造体を辿るというオーバヘッドが無くなったため，ほんの僅かではあるがオーバヘッドが減っているはずである．この改良の結果，並列アプリケーションプログラムを安定して動かすことが可能となった．ページテーブルの root pointer と ASID のコピーを作成したことによる不具合は今のところ発生していない．

5. おわりに

オリジナル並列分散 OS SSS-PC の基幹となる通信同期機構 MBCF の Linux OS 版である MBCF/Linux の開発を行った. 64bit の Linux OS に対応するために, MBCF 機構も 64bit 論理アドレスへの対応を行った．なお，サポートした CPU としては x86_64(intel64) のみではなく，ARM の AArch32 と AArch64 にも対応した．本稿では，MBCF の特徴となるパケット受信割り込み時におけるユーザメモリ空間直接アクセス方式を始めとする，Linux における MBCF の実装方式の詳細について述べた．MBCF/Linux の性能測定の第一報については，別稿[10]で改めて報告する．開発時に使用した Linux カーネル version 4.4 が x86_64 のアドレス空間 ID である PCID に対応していなかったため，現状の MBCF/Linux は PCID に対応できておらず，x86_64 では受信割り込みルーチンによってアドレス空間切替が起こるたびに，TLB 全フラッシュを余儀なくされている．PCID をサポートしたカーネルを対象に PCID をサポートした MBCF64/Linux デバイスドライバを開発する予定である．

謝辞 ARM AArch32 を対象にした MBCF32/Linux の開発は奈良女子大学と三菱電機株式会社先端技術総合研究所との共同研究「組込み SoC クラスタ化技術に関する研究」の下で実施された．また，その共同研究が MBCF/Linux の本格的開発の端緒となった．関係者のみなさまに深く感謝申し上げます．日頃の研究活動と研究室運営を支えてくれている松本研究室の所属メンバに心より感謝いたします．

参考文献

[1] 松本 尚: Linux 版の MBCF 通信機構について．コンピュータシステムシンポジウム論文集, 2021，情報処理学会, pp.27--36 (December 2021).
[2] 松本 尚，他: 汎用超並列オペレーティングシステム: SSS−CORE --- ワークステーションクラスタにおける実現 ---．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報告 No.73-20，情報処理学会, pp.115--120 (August 1996).
[3] 松本 尚: 次世代オペレーティングシステム SSS-PC の開発 --- IA32 用カーネルアーキテクチャ ---. ITX2002 Summer 論文集（CDROM），情報処理振興事業協会, (June 2002)
[4] Matsumoto, T. et al.: MBCF: A Protected and Virtualized High-Speed User-Level Memory-Based Communication Facility. Proc. of the 1998 ACM Int. Conf. on Supercomputing, pp.259--266 (July 1998).
[5] Matsumoto, T.: A Study on Memory-Based Communications and Synchronization in Distributed-Memory Systems. Dissertation Thesis, Graduate School of Science, Univ. of Tokyo (February 2001).
[6] SPARC International, Inc.:The SPARC Architecture Manual Version 8 Architecture Manual. SPARC International, Inc. 1991.
[7] Arm Limited: ARM Architecture Reference Manual Supplement - ARMv8, for the ARMv8-R AArch32 architecture profile. https://developer.arm.com/documentation/ddi0568/latest (2022 年 1 月 9 日確認).
[8] Arm Limited: Arm Architecture Reference Manual Supplement - Armv8, for Armv8-R AArch64 architecture profile. https://developer.arm.com/documentation/ddi0600/ac (2022 年 1 月 9 日確認).
[9] intel: Intel® 64 and IA-32 Architectures Software Developer Manuals.https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html (2022 年 1 月 9 日確認).
[10] 大家彩，猪俣雪乃，上村敬志，松本尚: 分散共有メモリ型通信機構 MBCF/Linux の基本性能について. ETNET2022 発表予定 (March 2022).