Solaris 8 のソフトウェア開発 (追補)

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
U.S.A. 650–960–1300

Part Number 806-7115-10
2001 年 2 月

原典: *Solaris 8 Software Developer Supplement*

Part No: 806-6612-10

Revision A

Adobe PostScript

目次

はじめに

『*Solaris 8* のソフトウェア開発 (追補)』では、Solaris™ 8 Update リリースにおいて追加および変更された機能について説明します。ここでの説明は、すでにリリースされている Solaris 8 のマニュアルセットの内容を補足または変更するものです。Solaris のマニュアルは、Solaris のこのリリースの DOCUMENTATION CD に含まれています。

注 - Solaris オペレーティング環境は、2 種類のハードウェア (プラットフォーム) 上で動作します。つまり、SPARC™ と IA (Intel アーキテクチャ) です。Solaris オペレーティング環境は、64 ビットと 32 ビットの両方のアドレス空間で動作し、IA では 32 ビットのアドレス空間でのみ動作します。このマニュアルで説明する情報は、章、節、注、箇条書き、図、表、例、またはコード例において特に明記しない限り、両方のプラットフォームおよびアドレス空間に該当します。

Sun のマニュアルの注文方法

専門書を扱うインターネットの書店 Fatbrain.com から、米国 Sun Microsystems™, Inc. (以降、Sun™ とします) のマニュアルをご注文いただけます。

マニュアルのリストと注文方法については、`http://www1.fatbrain.com/documentation/sun` の Sun Documentation Center をご覧ください。

Sun のオンラインマニュアル

http://docs.sun.com では、Sun が提供しているオンラインマニュアルを参照することができます。マニュアルのタイトルや特定の主題などをキーワードとして、検索を行うこともできます。

表記上の規則

このマニュアルでは、次のような字体や記号を特別な意味を持つものとして使用します。

表 P–1 表記上の規則

字体または記号	意味	例
AaBbCc123	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コード例を示します。	.login ファイルを編集します。 ls -a を使用してすべてのファイルを表示します。 system%
AaBbCc123	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して示します。	system% **su** password:
AaBbCc123	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには、rm *filename* と入力します。
『 』	参照する書名を示します。	『コードマネージャ・ユーザーズガイド』を参照してください。

表 P–1 表記上の規則 続く

字体または記号	意味	例
「 」	参照する章、節、ボタンやメニュー名、強調する単語を示します。	第 5 章「衝突の回避」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合に、継続を示します。	`sun% grep '^#define \` `XV_VERSION_STRING'`

ただし AnswerBook2™ では、ユーザーが入力する文字と画面上のコンピュータ出力は区別して表示されません。

コード例は次のように表示されます。

- C シェル

```
machine_name% command y|n [filename]
```

- C シェルのスーパーユーザー

```
machine_name# command y|n [filename]
```

- Bourne シェルおよび Korn シェル

```
$ command y|n [filename]
```

- Bourne シェルおよび Korn シェルのスーパーユーザー

```
# command y|n [filename]
```

[ ] は省略可能な項目を示します。上記の例は、*filename* は省略してもよいことを示しています。

| は区切り文字 (セパレータ) です。この文字で分割されている引数のうち 1 つだけを指定します。

キーボードのキー名は英文で、頭文字を大文字で示します (例: Shift キーを押します)。ただし、キーボードによっては Enter キーが Return キーの動作をします。

ダッシュ (-) は 2 つのキーを同時に押すことを示します。たとえば、Ctrl-D は Control キーを押したまま D キーを押すことを意味します。

一般規則

- このマニュアルでは、「IA」という用語は、Intel 32 ビットのプロセッサアーキテクチャを意味します。これには、Pentium、Pentium Pro、Pentium II、Pentium II Xeon、Celeron、Pentium III、Pentium III Xeon の各プロセッサ、および AMD、Cyrix が提供する互換マイクロプロセッサチップが含まれます。

第 1 章

新規機能の概要

この章では、Solaris 8 Update リリースに追加された新機能について説明します。

注 - 最新のマニュアルページを参照するには、man コマンドを使用してください。Solaris 8 Update リリースのマニュアルページには、「*Solaris 8 Reference Manual Collection*」には記載されていない新しい情報も提供されています。

表 1–1 Solaris 8 の新規機能

機能	サポート開始リリース
ドライバ	
Generic LAN ドライバ (GLD) を使用すると、Solaris ネットワークドライバに必要な STREAMS および Data Link Provider Interface (DLPI) 機能の大部分を実装できます。Solaris 8 10/00 より前のリリースでは、GLD モジュールを利用できるのは、Solaris の Intel 版ネットワークドライバに限定されていました。Solaris 8 10/00 からは、Solaris の SPARC 版ネットワークドライバでも GLD を利用できます。 詳細は、第 2 章を参照してください。	10/00
第 3 章では、ドライバの強化およびサービス利用可能性の確保によって高可用性をサポートするドライバの設計方法について、詳しく説明しています。この情報は、Solaris 8 の『*Writing Device Drivers*』の内容を補足するものです。 詳細は、第 3 章を参照してください。	10/00
その他のソフトウェア開発情報	

表 1–1 Solaris 8 の新規機能 続く

機能	サポート開始リリース
『マルチスレッドのプログラミング』が更新され、バグ ID 4308968、4356675、4356690 が修正されました。	1/01
ヨーロッパ言語版 Solaris ソフトウェア用に追加された部分ロケールには、ロシア語およびポーランド語対応の Universal Transformation Format-8 (UTF-8) ロケールの追加とともに、カタロニア語対応の 2 つの新しいロケールが含まれています。 詳細は、55ページの「ヨーロッパ言語版 Solaris ソフトウェア用に追加された部分ロケール」を参照してください。	10/00
リンカーとライブラリ用に多数の新機能が追加されました。 詳細は、65ページの「『リンカーとライブラリ』の更新」を参照してください。	1/01 および 10/00
Java	
Solaris 8 1/01 リリースには、前回のリリース以後のバグ修正によって改良された、JDK™ 1.2.2_06 および JDK 1.1.8_12 が組み込まれています。	1/01
Solaris 8 10/00 リリース以降には、JDK 1.2.2_05a の次の新機能が含まれます。 ■ スケーラビリティの向上 (20 以上の CPU に対応) ■ JIT (just-in time) コンパイラの最適化の向上 ■ テキストの描画性能の向上 ■ `poller` クラスのデモパッケージ ■ Swing の向上 詳細は、表 5–1 を参照してください。	10/00
32 ビット: `mod_jserv` モジュールおよび関連ファイルの追加によって、Apache Web サーバーで Java™ Servlet がサポートされるようになりました。 詳細は、62ページの「Apache Web サーバーにおける Java Servlet のサポート」を参照してください。	10/00
アーリーアクセス	
このリリースでは、アーリーアクセス (EA) ディレクトリにアーリーアクセスソフトウェアが含まれています。詳細は、Solaris 8 のリリースの SOFTWARE 2 of 2 CD に含まれる各アーリーアクセスソフトウェアの `README` を参照してください。	

第 2 章

ネットワークデバイス用のドライバ

注 - 最新のマニュアルページを参照するには、man コマンドを使用してください。Solaris 8 Update リリースのマニュアルページには、「*Solaris 8 Reference Manual Collection*」には記載されていない新しい情報も提供されています。

Solaris 8 10/00 リリースで、次の新しい機能が更新されました。

Generic LAN ドライバ (GLD) により、Solaris ネットワークドライバに STREAMS および Data Link Provider Interface (DLPI) 機能の大部分が実装されます。

Solaris 8 10/00 より前のリリースでは、GLD モジュールを利用できるのは、Solaris Intel 版のネットワークドライバに限定されていました。Solaris 8 10/00 からは、Solaris SPARC 版のネットワークドライバでも GLD を利用できます。

詳細は、次のマニュアルページを参照してください。`gld(7D)`、 `dlpi(7P)`、`gld(9E)`、 `gld(9F)`、 `gld_mac_info(9S)`、 `gld_stats(9S)`

Generic LAN ドライバ (GLD) の概要

GLD は、Solaris ローカルエリアネットワークデバイスドライバをサポートする、クローン化可能でロード可能なマルチスレッド化カーネルモジュールです。Solaris のローカルエリアネットワーク (LAN) デバイスドライバは、DLPI を使用してネットワークプロトコルスタックと通信する、STREAMS ベースのドライバです。これらのプロトコルスタックは、ネットワークドライバを使用して、ローカルエリア

ネットワーク上でパケットを送受信します。ネットワークデバイスドライバは、DDI/DKI 仕様、STREAMS 仕様、DLPI 仕様、およびデバイスそのもののプログラム式インタフェースの要件を満たすように実装する必要があります。

GLD には、Solaris LAN ドライバに必要な STREAMS および DLPI 機能の大部分が実装されています。いくつかの Solaris ネットワークドライバは GLD を使用して実装されています。

GLD を使用して実装された Solaris ネットワークドライバは、2 種類の部分で構成されます。STREAMS および DLPI インタフェースを扱う汎用コンポーネント、および特定のハードウェアデバイスを扱うデバイス固有のコンポーネントです。デバイス固有のモジュールは、GLD モジュール (`/kernel/misc/gld` に存在する) への依存を意味し、ドライバの `attach(9E)` 関数内から GLD に自身を登録します。正常に読み込まれると、ドライバは DLPI に準拠することになります。ドライバのデバイス固有の部分は、データを受信するとき、または GLD のサービスが必要になったときに、`gld(9F)` 関数を呼び出します。GLD は、デバイス固有のドライバが GLD に登録するときに提供したポインタを介して、デバイス固有のドライバの `gld(9E)` エントリポイントを呼び出します。`gld_mac_info(9S)` 構造体が GLD とデバイス固有のドライバ間のメインデータインタフェースです。

GLD 機能が現在サポートしているデバイスのタイプは、`DL_ETHER`、`DL_TPR`、および `DL_FDDI` です。GLD ドライバは、フルフォームの MAC レイヤーパケットを処理することが想定されており、論理リンク制御 (LLC) の処理は想定されていません。

状況によっては、GLD 機能を使用しないで 、DLPI に完全に準拠したドライバを実装する場合もあるでしょう。これには、ISO 8802 スタイル (IEEE 802) ではない LAN デバイスの場合、または GLD ではサポートされないデバイスタイプまたは DLPI サービスが必要な場合、などがあります。

タイプ `DL_ETHER`：Ethernet V2 および ISO 8802-3 (IEEE 802.3)

タイプとして `DL_ETHER` が指定されたデバイスについては、GLD は Ethernet V2 と ISO 8802-3 (IEEE 802.3) の両方のパケット処理をサポートします。Ethernet V2 によって、データリンクサービスのユーザーは、プロバイダのプロトコルを特に知らなくても、さまざまな適合するデータリンクサービスプロバイダにアクセスして利用することができます。サービスアクセスポイント (SAP) は、ユーザーがサービスプロバイダと通信するときに通過するポイントです。

0 ～ 255 の範囲の SAP 値と結合された Stream は、ユーザーが 8802-3 モードの使用を求めていることを示します。DL_BIND_REQ の SAP フィールド値がこの範囲内にある場合、GLD はその Stream の各後続の DL_UNITDATA_REQ メッセージ長 (14 バイトのメディアアクセス制御 (MAC) ヘッダーは含まない) を計算し、MAC フレームのヘッダーの type フィールドにその長さの 8802-3 フレームを送信します。この長さは 1500 を超えてはなりません。

0 ～ 1500 の範囲の type フィールドを持つメディアから受信されたフレームはすべて、8802-3 フレームとみなされ、8802-3 モードのオープンなすべての Stream (0 ～ 255 の範囲の SAP 値に結合された Stream) にルーティングされます。8802-3 モードの Stream が複数ある場合、着信フレームがコピーされて、該当するそれぞれの Stream にルーティングされます。

1500 より大きい SAP 値 (Ethernet V2 モード) と結合された Stream は、Ethernet MAC ヘッダーの type 値が、Stream と結合された SAP 値と正確に一致する着信パケットを受信します。

タイプ DL_TPR および DL_FDDI：SNAP 処理

メディアタイプ DL_TPR および DL_FDDI については、GLD は 255 より大きい SAP 値と結合された Stream に、最小限の SNAP (Sub-Net Access Protocol) 処理を実装します。0 ～ 255 の範囲の SAP 値は LLC SAP 値であり、メディアパケットフォーマットによって通常どおりに伝送されます。255 より大きい SAP 値には、LLC ヘッダーに従属し、16 ビット Ethernet V2 スタイルの SAP 値を伝送する SNAP ヘッダーが必要です。

SNAP ヘッダーは、宛先 SAP 0xAA が指定された LLC ヘッダーとして伝送されます。SAP 値が 255 より大きい発信パケットには、GLD は LLC+SNAP ヘッダーを作成します。このヘッダは、必ず次のような形式になります。

```
AA AA 03 00 00 00 XX XX
```

XX XX は、Ethernet V2 スタイルの type に対応する 16 ビットの SAP を表します。これがサポートされる唯一の SNAP ヘッダーのクラスです。ゼロ以外の OUI フィールドおよび 03 以外の LLC 制御フィールドは、SAP 0xAA を持つ LLC パケットとみなされます。これ以外の SNAP フォーマットを使用するクライアントは、LLC を使用し、SAP 0xAA に結合する必要があります。

着信パケットは上記のフォーマットに準拠しているかどうかが検証されます。このフォーマットに準拠するパケットは、そのパケットの 16 ビット SNAP タイプに結

合された Stream と照合されるとともに、LLC SNAP SAP 0xAA と一致するとみなされます。

LLC SAP として受信されたパケットは、メディアタイプ DL_ETHER の箇所で説明したように、LLC SAP と結合されているすべての Stream に渡されます。

タイプ DL_TPR：ソースルーティング

タイプ DL_TPR デバイスについては、GLD は最小限のソースルーティングサポートを実装します。ソースルーティングにより、ブリッジングされたメディア上でパケットを送信するステーションは、ネットワーク上でパケットがたどるルートを決定づけるルーティング情報を (パケットの MAC ヘッダーに) 指定できます。

GLD が提供するソースルーティングは、ルートを学習し、可能性のある複数のルートについて情報を要求するように指示し、要求に応答し、使用できるルートの中から選択する機能があります。さらに、発信パケットの MAC ヘッダーに Routing Information Fields を追加したり、着信パケット内のこのようなフィールドを認識したりします。

GLD のソースルーティングは、ISO 8802-2 (IEEE 802.2) の Section 9 で指定されている Route Determination Entity (RDE) を完全に実装するわけではありません。しかし、同じ (または縮約された) ネットワークに存在し得るソースルーティング実装方式と相互運用を行うように設計されています。

Style 1 および Style 2 の DLPI プロバイダ

GLD は、Style 1 および Style 2 の DLPI プロバイダを両方とも実装します。Physical Point of Attachment (PPA) は、システムが物理通信メディアと接続するポイントです。その物理メディア上の通信はすべて、PPA を通過します。Style 1 のプロバイダは、オープンされているメジャー/マイナーデバイスに基づいて、特定の PPA に Stream を接続します。Style 2 のプロバイダは、DLS ユーザーが DL_ATTACH_REQ を使用して必要な PPA を明示的に特定するよう要求します。その場合、open(9E) がユーザーと GLD 間の Stream を作成し、その後、DL_ATTACH_REQ が所定の PPA とその Stream を対応付けます。Style 2 はゼロのマイナー番号で示されます。マイナー番号がゼロ以外のデバイスノードがオープンしている場合、Style 1 が示され、対応する PPA はマイナー番号から 1 を差し引いたものになります。Style 1 と Style 2 が両方とも open の場合、デバイスはクローン化されます。

実装される DLPI プリミティブ

GLD は、いくつかの DLPI プリミティブを実装します。DL_INFO_REQ プリミティブは、DLPI Stream に関する情報を要求します。メッセージは 1 つの M_PROTO メッセージブロックで構成されます。GLD はこの要求に対する DL_INFO_ACK 応答で、gld_register() に渡された gldm_mac_info(9S) 構造体に GLD ベースのドライバが指定した情報に基づいて、デバイスに依存する値を返します。ただし、GLD は、すべての GLD ベースのドライバに代わって次の値を返します。

- バージョンは DL_VERSION_2 です。
- サービスモードは DL_CLDLS です。GLD はコネクションレスモードのサービスを実装します。
- プロバイダスタイルは、Stream がどのようにオープンされたかにより、DL_STYLE1 または DL_STYLE2 になります。
- オプションの Quality Of Service (QOS) サポートはありません。QOS フィールドはゼロです。

注 - DLPI 仕様と異なり、GLD は、Stream が PPA に接続される前であっても、DL_INFO_ACK でデバイスの正確なアドレス長およびブロードキャストアドレスを返します。

PPA と Stream を対応付けるために、DL_ATTACH_REQ プリミティブが使用されます。この要求は、Style 2 の DLS プロバイダが通信を行う物理メディアを特定するために必要です。完了すると、状態が DL_UNATTACHED から DL_UNBOUND に変わります。メッセージは 1 つの M_PROTO メッセージブロックで構成されます。ドライバを Style 1 モードで使用している場合は、この要求は許可されません。Style 1 を使用してオープンされた Stream は、オープンの完了時には PPA にすでに接続されているからです。

DL_DETACH_REQ プリミティブは、Stream から PPA を切り離すことを要求します。これが認められるのは、Stream が Style 2 を使用してオープンされた場合だけです。

DL_BIND_REQ プリミティブおよび DL_UNBIND_REQ プリミティブは、DLSAP と Stream の結合および結合解除を行います。1 つの Stream と対応付けられた PPA は、Stream の DL_BIND_REQ の処理が完了する前に初期化を完了します。複数の Stream を同じ SAP に結合できますが、このような Stream はそれぞれ、その SAP で受信したパケットのコピーを受け取ります。

DL_ENABMULTI_REQ プリミティブおよび DL_DISABMULTI_REQ プリミティブは、個々のマルチキャストグループアドレスの受け付けを可能または不可能にしま

す。アプリケーションまたは他の DLS ユーザーは、これらのプリミティブを繰り返し使用して 1 組のマルチキャストアドレスを Stream 単位で作成または変更することが可能です。これらのプリミティブが受け入れられるように、Stream を PPA に接続する必要があります。

DL_PROMISCON_REQ プリミティブおよび DL_PROMISCOFF_REQ プリミティブは、物理レベルまたは SAP レベルのどちらかで、プロミスキュアス (promiscuous) モードを Stream 単位で有効または無効にします。DL Provider は、DL_DETACH_REQ または DL_PROMISCOFF_REQ を受信するまで、または Stream がクローズされるまで、そのメディアで受信したすべてのメッセージを DLS ユーザーにルーティングします。物理レベルのプロミスキュアスの受信を、そのメディア上の全パケットに指定することも、マルチキャストパケットに限定して指定することもできます。

注 - これらのプロミスキュアスモードのプリミティブが受け入れられるように、Stream を PPA に接続する必要があります。

DL_UNITDATA_REQ プリミティブは、コネクションレス型転送でデータを送信する場合に使用します。これは未承認のサービスなので、配信の保証はありません。メッセージは 1 つの M_PROTO メッセージブロックとそれに続く 1 つ以上の M_DATA ブロック (1 バイト以上のデータを含む) で構成されます。

DL_UNITDATA_IND タイプは、パケットを受信してアップストリームに渡す場合に使用します。パケットは、プリミティブを DL_UNITDATA_IND に設定した M_PROTO メッセージに格納されます。

DL_PHYS_ADDR_REQ プリミティブは、Stream に接続された PPA にその時点で対応付けられている MAC アドレスを要求します。このアドレスは、DL_PHYS_ADDR_ACK プリミティブによって返されます。Style 2 を使用している場合、このプリミティブが有効なのは DL_ATTACH_REQ が成功した場合に限られます。

DL_SET_PHYS_ADDR_REQ プリミティブは、Stream に接続された PPA にその時点で対応付けられている MAC アドレスを変更します。このプリミティブは、現在および将来にわたりこのデバイスに接続された他のすべての Stream に作用します。いったん変更すると、現在オープンしている、または今後オープンされてこのデバイスに接続されるすべての Stream が、この新しい物理アドレスを取得します。新しい物理アドレスは、このプリミティブを使用して再び物理アドレスを変更するまで、またはドライバがリロードされるまで有効です。

注 - スーパーユーザーは、他の Stream が同じ PPA に結合されている際には PPA の物理アドレスを変更することができます。

DL_GET_STATISTICS_REQ プリミティブは、Stream に接続された PPA に対応する統計情報を収めた DL_GET_STATISTICS_ACK 応答を要求します。DL_ATTACH_REQ を使用して Style 2 の Stream を特定の PPA に接続しておかないと、このプリミティブは成功しません。

実装される ioctl 関数

GLD は、以下で説明する ioctl *ioc_cmd* 関数を実装します。認識できない ioctl コマンドを受信した GLD は、gld(9E) に記述されているように、デバイス固有のドライバの gldm_ioctl() ルーチンにそのコマンドを渡します。

DLIOCRAW ioctl 関数は、snoop(1M) コマンドをはじめ、一部の DLPI アプリケーションで使用されます。DLIOCRAW コマンドは、Stream を raw モードにし、それによってドライバが、着信パケットの報告に通常使用される DL_UNITDATA_IND 形式に変換するのではなく、M_DATA メッセージで MAC レベルの着信パケット全体をアップストリーム送信するようにします。パケット SAP のフィルタリングは、raw モードの Stream にも実行されます。Stream のユーザーがすべての着信パケットの受け取りを希望する場合は、適切なプロミスキュアスモード (複数可) も選択しなければなりません。raw モードを正しく選択しているアプリケーションは、フルフォーマットのパケットを伝送する M_DATA メッセージとしてドライバに送ることもできます。DLIOCRAW は引数を使用しません。raw モードが有効になった Stream は、クローズされるまでそのモードのままです。

GLD ドライバの要件

GLD ベースのドライバには、ヘッダーファイル <sys/gld.h> が組み込まれていなければなりません。

また、GLD ベースのドライバには次の宣言が含まれていることも必要です。

```
char _depends_on[] = "misc/gld";
```

GLD はデバイス固有のドライバのために、open(9E) および close(9E) 関数の他に、要求される STREAMS の put(9E) および srv(9E) 関数を実装します。さらに、ドライバ用に getinfo(9E) 関数も実装します。

module_info(9S) 構造体の mi_idname 要素は、ドライバ名を指定する文字列です。ファイルシステムに存在するドライバモジュールの名前と正確に一致しなければなりません。

読み取り側の qinit(9S) 構造体では、次の要素を指定する必要があります。

```
qi_putp             NULL

qi_srvp             gld_rsrv

qi_qopen            gld_open

qi_qclose           gld_close
```

書き込み側の qinit(9S) 構造体では、次の要素を指定する必要があります。

```
qi_putp             gld_wput

qi_srvp             gld_wsrv

qi_qopen            NULL

qi_qclose           NULL
```

dev_ops(9S) 構造体の devo_getinfo 要素では、getinfo(9E) ルーチンとして gld_getinfo を指定する必要があります。

ドライバの attach(9E) 関数は、ハードウェア固有のデバイスドライバと GLD 機能を対応付け、デバイスとドライバを使用できるように準備を整える一切の作業を行います。

attach(9E) 関数は、gld_mac_alloc() を使用して gld_mac_info(9S) (macinfo) 構造体を割り当てます。ドライバは通常、1 つのデバイスについて、macinfo 構造体で定義されているより多くの情報を保存しなければならないので、必要な追加分のデータ構造体を割り当て、gld_mac_info(9S) 構造体の gldm_private メンバーにそのデータ構造体へのポインタを保存する必要があります。

attach(9E) ルーチンは、gld_mac_info(9S) に記述されているように macinfo 構造体を初期化し、その後 gld_register() を呼び出して、ドライバと GLD モジュールを結びつけなければなりません。ドライバは必要に応じてレジスタをマップし、完全に初期化して、gld_register() を呼び出す前に割り込みを受け付けるように準備する必要があります。attach(9E) 関数が割り込みを追加することはあっても、デバイスに割り込みを発生させてはなりません。ドライバは、ハー

ドウェアが確実に静止しているように、gld_register() を呼び出す前に、ハードウェアをリセットする必要があります。gld_register() の呼び出し前に、デバイスが起動されたり、割り込みが発生するような状態になったりしてはなりません。これは、gld(9E) に記載されているように、後に GLD がドライバの gldm_start() エントリポイントを呼び出した時点で行います。gld_register() が正常に完了すると、GLD はいつでも gld(9E) エントリポイントを呼び出すことができます。

attach(9E) ルーチンは、gld_register() が正常に完了した場合は、DDI_SUCCESS を返します。gld_register() がエラーになった場合は、DDI_FAILURE を返し、gld_register() を呼び出す前に割り当てられたあらゆるリソースを、attach(9E)ルーチンで割り当て解除する必要があります。そして、DDI_FAILURE を返します。どのような状況でも、エラーの起きた macinfo 構造体を再利用することがあってはなりません。gld_mac_free() を使用して、割り当てを解除する必要があります。

detach(9E) 関数は、GLD からドライバの登録を解除しようとします。これは、gld(9F) に記載されている gld_unregister() を呼び出すことによって行います。detach(9E) ルーチンは、ddi_get_driver_private(9F) を使用することによって、デバイスの専用データから必要な gld_mac_info(9S) 構造体に対するポインタを取得できます。gld_unregister() は、一定の条件をチェックして、ドライバを切り離せないかどうかを調べます。このチェックに失敗すると、gld_unregister() は DDI_FAILURE を返します。その場合、ドライバの detach(9E) ルーチンはデバイスを動作状態にしたまま、DDI_FAILURE を返さなければなりません。

チェックが成功すると、gld_unregister() は、(必要に応じてドライバの gldm_stop() ルーチンを呼び出して) デバイスの割り込み中止を確認し、ドライバを GLD フレームワークから切り離し、DDI_SUCCESS を返します。この場合、detach(9E) ルーチンは割り込みを削除し、attach(9E) ルーチンに割り当てられていたすべてのデータ構造を (macinfo 構造体の割り当てを解除する gld_mac_free() を使用して) 割り当て解除したうえで、DDI_SUCCESS を返さなければなりません。gld_mac_free() を呼び出す前に割り込みを削除することが重要です。

ネットワーク統計

Solaris ネットワークドライバは統計変数を実装しなければなりません。一部のネットワーク統計については、GLD 本体で記録されますが、他のネットワーク統計は

GLD ベースのドライバごとにカウントする必要があります。GLD は、GLD ベースのドライバによるネットワークドライバ統計の標準セットのレポートをサポートします。統計は、kstat(7D) および kstat(9S) メカニズムを使用して、GLD が報告します。DLPI コマンドの DL_GET_STATISTICS_REQ を使用して、現在の統計カウンタを検索することもできます。統計は特に指定が無い限り、いずれも符号無しで維持され、32 ビットです。

GLD が維持および報告する統計は、次のとおりです。

rbytes64	インタフェース上で正常に受信したトータルのバイト数 (64 ビット)
rbytes	インタフェース上で正常に受信したトータルのバイト数
obytes64	インタフェース上で送信を要求したトータルのバイト数 (64 ビット)
obytes	インタフェース上で送信を要求したトータルのバイト数
ipackets64	インタフェース上で正常に受信したトータルのパケット数 (64ビット)
ipackets	インタフェース上で正常に受信したトータルのパケット数
opackets64	インタフェース上で送信を要求したトータルのパケット数 (64 ビット)
opackets	インタフェース上で送信を要求したトータルのパケット数
multircv	グループおよび機能アドレスを含め、正常に受信したマルチキャストパケット (long)
multixmt	グループおよび機能アドレスを含め、送信を要求したマルチキャストパケット (long)
brdcstrcv	正常に受信したブロードキャストパケット (long)

brdcstxmt	送信を要求したブロードキャストパケット (long)
unknowns	どの Stream も受け付けなかった有効な受信パケット (long)
noxmtbuf	送信バッファが使用中だった、または送信バッファを割り当てることができなかったために、出力で廃棄されたパケット (long)
blocked	キューがフロー制御にされていたために、受信パケットを Stream に格納できなかった回数 (long)
xmtretry	リソース不足のために送信が遅延された後で、送信が再試行された回数 (long)
promisc	インタフェースの現在の「プロミスキュアス」状態 (文字列)

デバイス依存型のドライバは、次の統計情報をカウントし、専用のインスタンス別構造体でその記録を保管します。統計情報を報告するように求められた GLD は、gld(9E) に記述されているように、ドライバの gldm_get_stats() エントリポイントを呼び出して、gld_stats(9S) 構造体のデバイス固有の統計情報を更新します。GLD は次に、以下に示す名前付き統計変数を使用して、更新された統計情報を報告します。

ifspeed	ビット/秒で表したインタフェースの現在の帯域幅の見積り (64ビット)
media	デバイスが現在使用しているメディアタイプ (文字列)
intr	割り込みハンドラが呼び出され、割り込みが要求された回数 (long)
norcvbuf	受信バッファを割り当てることができなかったために、有効な着信パケットが廃棄されたことが判明している回数 (long)
ierrors	エラーが含まれていたために処理できなかったトータルの受信パケット (long)

oerrors	エラーが原因で正常に送信されなかったトータルのパケット (long)
missed	受信時にハードウェアによってドロップされたことが判明しているパケット (long)
uflo	送信時の FIFO アンダーフロー回数 (long)
oflo	受信時の受信側オーバーフロー回数 (long)

次の統計グループは、DL_ETHER タイプのネットワークに適用されます。これらの統計は、前述のように、そのタイプのデバイス固有のドライバによって維持されます。

align_errors	フレーム指示エラーの起きた受信パケット (オクテットの整数ではない) (long)
fcs_errors	CRC エラーの起きた受信パケット (long)
duplex	インタフェースの現在の二重モード (文字列)
carrier_errors	送信試行時にキャリアが失われたか、または全く検出されなかった回数 (long)
collisions	送信時の Ethernet の衝突 (long)
ex_collisions	送信時に発生した衝突が多すぎて、送信エラーとなったフレーム (long)
tx_late_collisions	遅れて (512 ビットタイム) 発生した送信衝突の回数 (long)
defer_xmts	衝突は発生しなかったが、メディアがビジー状態だったために最初の送信試行が遅延されたパケット (long)
first_collisions	1 回の衝突だけで正常に送信されたパケット
multi_collisions	複数の衝突が起きたが正常に送信されたパケット
sqe_errors	SQE テストエラーが報告された回数
macxmt_errors	キャリアおよび衝突エラー以外の送信 MAC エラーが検出されたパケット

macrcv_errors	align_errors、fcs_errors、toolong_errors 以外の、MAC エラーが起きた受信パケット
toolong_errors	許容される最大長を超えていた受信パケット
runt_errors	許容される最小長に満たなかった受信パケット (long)

次の統計グループは、DL_TPR タイプのネットワークに適用されます。これらの統計は、前述のように、そのタイプのデバイス固有のドライバによって維持されます。

line_errors	非データビットまたは FCS エラーのあった受信パケット
burst_errors	ハーフビットタイマー 5 回の間に変位が発生しなかったことが検出された回数
signal_losses	リングでシグナル損失状態が検出された回数
ace_errors	AMP フレームの介入の無い、別の同様の SMP フレームが続く、A、C が共に 0 の AMP または SMP フレームの数
internal_errors	ステーションが内部エラーを認識した回数
lost_frame_errors	送信中に TRR タイマが期間満了した回数
frame_copied_errors	このステーション宛てのフレームが、FS フィールドの「A」ビットを 1 に設定して受信された回数
token_errors	アクティブモニターとして動作しているステーションが、トークンの送信を必要とするエラー状態を認識した回数
freq_errors	着信信号の周波数が予期された周波数と異なっていた回数

次の統計グループは、DL_FDDI タイプのネットワークに適用されます。これらの統計は、前述のように、そのタイプのデバイス固有のドライバによって維持されます。

mac_errors	別の MAC ではエラーが検出されず、この MAC ではエラーが検出されたフレーム

mac_lost_errors	フレームが取り除かれるなどのフォーマットエラーが起きた受信フレーム数
mac_tokens	受信トークン数 (制限無しと制限付きの合計)
mac_tvx_expired	TVX が期間満了になった回数
mac_late	この MAC がリセットされて以後、またはトークンの受信以後、TRT が期間満了になった回数
mac_ring_ops	リングが「Ring Not Operational」状態から「Ring_Operational」状態になった回数

宣言とデータ構造

gld_mac_info 構造体

GLD の MAC 情報 (gld_mac_info) 構造体は、デバイス固有のドライバと GLD 間のメインデータインタフェースです。この構造体には、GLD が必要とするデータ、およびオプションの追加のドライバ固有情報の構造体に対するポインタが含まれます。

gld_mac_info 構造体は、gld_mac_alloc() を使用して割り当て、gld_mac_free() を使用して割り当てを解除します。ドライバ側でこの構造の長さを想定してはなりません。長さは Solaris の各リリース、各 GLD、またはその両方によって異なる可能性があるためです。このマニュアルには記載されていない GLD 専用の構造体のメンバーをデバイス固有のドライバで設定したり読み取ったりしてはなりません。

gld_mac_info(9S) 構造体には、次のフィールドがあります。

```
caddr_t             gldm_private;             /* Driver private data */
int                 (*gldm_reset)();          /* Reset device */
int                 (*gldm_start)();          /* Start device */
int                 (*gldm_stop)();           /* Stop device */
int                 (*gldm_set_mac_addr)();   /* Set device phys addr */
int                 (*gldm_set_multicast)();  /* Set/delete multicast addr */
int                 (*gldm_set_promiscuous)(); /* Set/reset promiscuous mode */
int                 (*gldm_send)();           /* Transmit routine */
uint_t              (*gldm_intr)();           /* Interrupt handler */
int                 (*gldm_get_stats)();      /* Get device statistics */
int                 (*gldm_ioctl)();          /* Driver-specific ioctls */
char                *gldm_ident;              /* Driver identity string */
uint32_t            gldm_type;                /* Device type */
uint32_t            gldm_minpkt;              /* Minimum packet size */
```

```
                                                    /* accepted by driver */
uint32_t             gldm_maxpkt;                 /* Maximum packet size */
                                                    /* accepted by driver */
uint32_t             gldm_addrlen;                /* Physical address length */
int32_t              gldm_saplen;                 /* SAP length for DL_INFO_ACK */
unsigned char        *gldm_broadcast_addr;        /* Physical broadcast addr */
unsigned char        *gldm_vendor_addr;           /* Factory MAC address */
t_uscalar_t          gldm_ppa;                    /* Physical Point of */
                                                    /* Attachment (PPA) number */
dev_info_t           *gldm_devinfo;               /* Pointer to device's */
                                                    /* dev_info node */
ddi_iblock_cookie_t  gldm_cookie;                 /* Device's interrupt */
                                                    /* block cookie */
```

デバイスドライバは、gld_mac_info 構造体のメンバーを認識できます。

gldm_private　この構造体のメンバーは、デバイス固有のドライバ専用であり、GLD が使用したり変更したりすることはありません。これは、従来専用データに対するポインタとして使用されたもので、ドライバが定義し、またドライバが割り当てたインスタンス別データ構造体を指し示します。

次の構造体メンバーのグループは、gld_register() を呼び出す前にドライバで設定しなければなりません。以後は、ドライバ側で変更してはなりません。gld_register() がこれらの構造体のメンバーの値を使用またはキャッシュすることがあるので、gld_register() を呼び出した後でドライバが変更を行うと、予期せぬ結果を招く可能性があります。

gldm_reset　ドライバのエントリポイントに対するポインタ。gld(9E) を参照

gldm_start　ドライバのエントリポイントに対するポインタ。gld(9E) を参照

gldm_stop　ドライバのエントリポイントに対するポインタ。gld(9E) を参照

gldm_set_mac_addr　ドライバのエントリポイントに対するポインタ。gld(9E) を参照

gldm_set_multicast　ドライバのエントリポイントに対するポインタ。gld(9E) を参照

gldm_set_promiscuous	ドライバのエントリポイントに対するポインタ。gld(9E) を参照
gldm_send	ドライバのエントリポイントに対するポインタ。gld(9E) を参照
gldm_intr	ドライバのエントリポイントに対するポインタ。gld(9E) を参照
gldm_get_stats	ドライバのエントリポイントに対するポインタ。gld(9E) を参照
gldm_ioctl	ドライバのエントリポイントに対するポインタ。NULL にすることができる。gld(9E) を参照
gldm_ident	デバイスに関する短い記述を含む文字列に対するポインタ。システムメッセージ内のデバイスを識別する場合に使用します。
gldm_type	ドライバが処理するデバイスのタイプ。GLD で現在サポートしている値は、DL_ETHER (ISO 8802-3 [IEEE 802.3] および Ethernet Bus)、DL_TPR (IEEE 802.5 Token Passing Ring)、および DL_FDDI (ISO 9314-2 Fibre Distributed Data Interface) です。GLD を正しく動作させるには、この構造体のメンバーを適切に設定する必要があります。
gldm_minpkt	最小の Service Data Unit サイズ - デバイスが送信する最小パケットサイズです (MAC ヘッダーは含まない)。デバイス固有のドライバが必要なすべてのパディングを処理する場合は、このサイズをゼロにすることができます。
gldm_maxpkt	最大の *Service Data Unit* サイズ - デバイスが送信できる最大パケットサイズです (MAC ヘッダーは含まない)。Ethernet の場合、この値は 1500 です。
gldm_addrlen	デバイスが処理する物理アドレスの長さ (バイト数)。Ethernet、Token Ring、および FDDI の場

	合、この構造体のメンバーの値は 6 でなければなりません。
gldm_saplen	ドライバが使用する SAP アドレスの長さ (バイト数)。GLD ベースのドライバでは、これは常に -2 に設定します。この値は、2 バイトの SAP 値がサポートされること、および DLSAP アドレスで物理アドレスの後に SAP が来ることを意味します。詳細は、DLPI 仕様の「Message DL_INFO_ACK」を参照してください。
gldm_broadcast_addr	送信に使用されるブロードキャストアドレスが格納されるバイト配列の長さ gldm_addrlen に対するポインタ。ドライバはブロードキャストアドレスを収めるスペースを提供し、そのスペースに適切な値を入れ、その値を指すように gldm_broadcast_addr を設定する必要があります。Ethernet、Token Ring、および FDDI の場合、ブロードキャストアドレスは通常、0xFF-FF-FF-FF-FF-FF です。
gldm_vendor_addr	ベンダーが提供したデバイスのネットワーク物理アドレスが格納される、バイト配列の長さ gldm_addrlen に対するポインタ。ドライバはこのアドレスを収めるスペースを提供し、そのスペースにデバイスから読み取った情報を入れ、その情報を指すように gldm_vendor_addr を設定する必要があります。
gldm_ppa	デバイスのこのインスタンスに対応する PPA 番号。通常、ddi_get_instance(9F) から返されたインスタンス番号に設定します。
gldm_devinfo	このデバイスの dev_info ノードに対するポインタ
gldm_cookie	ddi_get_iblock_cookie(9F)、ddi_add_intr(9F)、ddi_get_soft_iblock_cookie(9F)、または ddi_add_softintr(9F) から返された割り込

みブロック cookie。これは、gld_recv() の呼び出し元となる、デバイスの受信割り込みに対応しなければなりません。

gld_stats 構造体

GLD 統計 (gld_stats) 構造体は、gld(9E) および gld(7D) で記述されているように、ドライバの gldm_get_stats() ルーチンから戻るときに、GLD ベースのドライバから GLD に統計情報および状態情報を伝えるために使用します。この構造体のメンバーは、GLD ベースのドライバによってセットされ、GLD が統計情報を報告するときに使用されます。後出の表では、GLD が報告する統計変数の名前がコメントで示されています。個々の統計情報の詳細については、gld(7D) のマニュアルページを参照してください。

ドライバはこの構造体の長さに関して仮定することができません。Solaris、GLD、またはその両方のリリースごとに異なる可能性があります。このマニュアルに記載されていない GLD 専用の構造体のメンバーは、デバイス固有のドライバによって設定したり読み取ったりしてはなりません。

次の構造体のメンバーは、すべてのメディアタイプに対して定義されます。

```
uint64_t        glds_speed;                     /* ifspeed */
uint32_t        glds_media;                     /* media */
uint32_t        glds_intr;                      /* intr */
uint32_t        glds_norcvbuf;                  /* norcvbuf */
uint32_t        glds_errrcv;                    /* ierrors */
uint32_t        glds_errxmt;                    /* oerrors */
uint32_t        glds_missed;                    /* missed */
uint32_t        glds_underflow;                 /* uflo */
uint32_t        glds_overflow;                  /* oflo */
```

次の構造体のメンバーは、メディアタイプ DL_ETHER に対して定義されます。

```
uint32_t        glds_frame;                     /* align_errors */
uint32_t        glds_crc;                       /* fcs_errors */
uint32_t        glds_duplex;                    /* duplex */
uint32_t        glds_nocarrier;                 /* carrier_errors */
uint32_t        glds_collisions;                /* collisions */
uint32_t        glds_excoll;                    /* ex_collisions */
uint32_t        glds_xmtlatecoll;               /* tx_late_collisions */
uint32_t        glds_defer;                     /* defer_xmts */
uint32_t        glds_dot3_first_coll;           /* first_collisions */
uint32_t        glds_dot3_multi_coll;           /* multi_collisions */
uint32_t        glds_dot3_sqe_error;            /* sqe_errors */
uint32_t        glds_dot3_mac_xmt_error;        /* macxmt_errors */
uint32_t        glds_dot3_mac_rcv_error;        /* macrcv_errors */
uint32_t        glds_dot3_frame_too_long;       /* toolong_errors */
uint32_t        glds_short;                     /* runt_errors */
```

次の構造体のメンバーは、メディアタイプ DL_TPR に対して定義されます。

```
uint32_t        glds_dot5_line_error              /* line_errors */
uint32_t        glds_dot5_burst_error             /* burst_errors */
uint32_t        glds_dot5_signal_loss             /* signal_losses */
uint32_t        glds_dot5_ace_error               /* ace_errors */
uint32_t        glds_dot5_internal_error          /* internal_errors */
uint32_t        glds_dot5_lost_frame_error        /* lost_frame_errors */
uint32_t        glds_dot5_frame_copied_error      /* frame_copied_errors */
uint32_t        glds_dot5_token_error             /* token_errors */
uint32_t        glds_dot5_freq_error              /* freq_errors */
```

次の構造体のメンバーは、メディアタイプ DL_FDDI に対して定義されます。

```
uint32_t        glds_fddi_mac_error;              /* mac_errors */
uint32_t        glds_fddi_mac_lost;               /* mac_lost_errors */
uint32_t        glds_fddi_mac_token;              /* mac_tokens */
uint32_t        glds_fddi_mac_tvx_expired;        /* mac_tvx_expired */
uint32_t        glds_fddi_mac_late;               /* mac_late */
uint32_t        glds_fddi_mac_ring_op;            /* mac_ring_ops */
```

上記の統計変数のほとんどは、特定のイベントが検出された回数を示すカウンタです。例外は次のとおりです。

glds_speed
: インタフェースの現在の帯域幅の概算値 (ビット/秒）。帯域幅の変動のないインタフェース、または正確な推定ができないインタフェースの場合、このオブジェクトには公称の帯域幅が入ります。

glds_media
: ハードウェアで使用されているメディア (配線) またはコネクタのタイプ。現在サポートされているメディア名には、 GLDM_AUI、GLDM_BNC、GLDM_TP、GLDM_10BT、 GLDM_100BT、GLDM_100BTX、 GLDM_100BT4、GLDM_RING4、 GLDM_RING16、 GLDM_FIBER、および GLDM_PHYMII が含まれます。GLDM_UNKNOWN を指定することもできます。

glds_duplex
: インタフェースの現在の二重状態。サポートされる値は、GLD_DUPLEX_HALF および GLD_DUPLEX_FULL です。GLD_DUPLEX_UNKNOWN を指定することもできます。

エントリポイントおよびサービスルーチン

GLD ルーチンで使用される引数

macinfo	gld_mac_info(9S) 構造体に対するポインタ。
macaddr	有効な MAC アドレスが格納されたキャラクタ配列の先頭に対するポインタ。この配列は、gld_mac_info(9S) 構造体の gldm_addrlen 要素で、ドライバによって指定された長さになります。
multicastaddr	マルチキャスト、グループ、または機能アドレスが格納されたキャラクタ配列の先頭に対するポインタ。この配列は、gld_mac_info(9S) 構造体の gldm_addrlen 要素で、ドライバによって指定された長さになります。
multiflag	マルチキャストアドレスの受け付けを可能にするか不可能にするかを示すフラグ。この引数は、GLD_MULTI_ENABLE または GLD_MULTI_DISABLE として指定します。
promiscflag	有効になっているプロミスキュアスモードのタイプを示すフラグ。この引数は、GLD_MAC_PROMISC_PHYS、GLD_MAC_PROMISC_MULTI、または GLD_MAC_PROMISC_NONE として指定します。
mp	gld_ioctl() は、実行する ioctl が格納されている STREAMS メッセージブロックへのポインタとして *mp* を使用します。gld_send() は、送信するパケットが格納されている STREAMS メッセージブロックへのポインタとして *mp* を使用します。gld_recv() は、受信パケットが格納されているメッセージブロックへのポインタとして *mp* を使用します。
stats	統計カウンタの現在値が入る、gld_stats(9S) 構造体に対するポインタ。
q	ioctl への応答で使用される、queue(9S) 構造体に対するポインタ。

dip	デバイスの dev_info 構造体に対するポインタ。
name	デバイスのインタフェース名

エントリポイント

これらのエントリポイントは、GLD とのインタフェースとして設計されたデバイス固有のネットワークドライバによって実装される必要があります。

gld(7D) で記述されているように、デバイス固有のドライバと GLD モジュール間の通信に関するメインデータ構造は、gld_mac_info(9S) 構造体です。この構造体の一部の要素は、ここで説明するエントリポイントへの関数ポインタです。デバイス固有のドライバは、attach(9E) ルーチンで、これらの関数ポインタを初期化してから gld_register() を呼び出さなければなりません。

```
int prefix_reset(gld_mac_info_t * macinfo);
```

gldm_reset() は、ハードウェアを初期状態にリセットします。

```
int prefix_start(gld_mac_info_t * macinfo);
```

gldm_start() により、デバイスは割り込みを発生させ、受信データパケットを GLD に配信する目的で、ドライバが gld_recv() を呼び出せるようにします。

```
int prefix_stop(gld_mac_info_t * macinfo);
```

gldm_stop() は、デバイスが割り込みを発生させることを不可にし、データパケットを GLD に配信する目的でドライバが gld_recv() を呼び出すことを止めさせます。GLD は gldm_stop() ルーチンによって、デバイスがこれ以上割り込みをかけなくなることを確実に保証しなければなりません。この関数は常に、GLD_SUCCESS を返します。

```
int prefix_set_mac_addr(gld_mac_info_t * macinfo, unsigned char * macaddr);
```

gldm_set_mac_addr() は、ハードウェアがデータの受信に使用する物理アドレスを設定します。この関数は、デバイスを渡された MAC アドレス *macaddr* にプログラムしなければなりません。現在利用できるリソースが不足していて要求を満たすことができない場合は、GLD_NORESOURCES を返します。要求された関数がサポートされていない場合は、GLD_NOTSUPPORTED を返します。

```
int prefix_set_multicast(gld_mac_info_t * macinfo, unsigned char * multicastaddr,
    int multiflag);
```

gldm_set_multicast() は、特定のマルチキャストアドレスのデバイスレベルでの受け付けを可能または不可能にします。3 番目の引数 *multiflag* が GLD_MULTI_ENABLE に設定されている場合、この関数は 2 番目の引数によって示されたマルチキャストアドレスを持つパケットを受信するよう、インタフェースを設定します。*multiflag* が GLD_MULTI_DISABLE に設定されている場合、ドライバは指定されたマルチキャストアドレスの受け付けを不可にすることが許可されます。

この関数は、GLD がマルチキャスト、グループ、または機能アドレスの受け付けを可能または不可能に設定するたびに呼び出されます。GLD は、デバイスがどのような方法でマルチキャストをサポートするのか、どのような方法でこの関数を呼び出して特定のマルチキャストアドレスを有効または無効にするのかについて、何も想定を行いません。デバイスによっては、ハッシュアルゴリズムとビットマスクを使用して、マルチキャストアドレスの集合を有効にするものもあります。この手順は認められており、GLD が余分なパケットをフィルタリングして除外します。1 つのアドレスを無効にするとデバイスレベルで複数のアドレスが無効になる可能性がある場合、GLD が有効にしているアドレスを無効にしてしまうことがないように、必要な情報を保存するのはデバイスドライバ側の役目です。

gldm_set_multicast() は、すでに有効になっている特定のマルチキャストアドレスを有効にするために呼び出されることはなく、また現在有効になっていないアドレスを無効にするために呼び出されることもありません。GLD は同じマルチキャストアドレスに対する複数の要求を追跡し、特定のマルチキャストアドレスを有効にすることを求める最初の要求、または無効にすることを求める最後の要求に限り、ドライバのエントリポイントを呼び出します。そのとき、利用できるリソースが不足していて要求を満たせない場合は、GLD は GLD_NORESOURCES を返します。要求された関数がサポートされていない場合は、GLD_NOTSUPPORTED を返します。

```
int prefix_set_promiscuous(gld_mac_info_t * macinfo, int promiscflag);
```

gldm_set_promiscuous() は、プロミスキュアスモードを有効または無効にします。この関数は、GLD がメディア上のすべてのパケットの受信、またはメディア上のすべてのマルチキャストパケットの受信を可能または不可能に設定するたびに呼び出されます。2 番目の引数 *promiscflag* が GLD_PROMISC_PHYS の値に設定されている場合は、この関数は物理レベルのプロミスキュアスモードを有効にし、その結果、メディア上のすべてのパケットが受信されます。*promiscflag* が GLD_PROMISC_MULTI に設定されている場合は、すべてのマルチキャストパケットの受信が可能になります。*promiscflag* が GLD_PROMISC_NONE に設定されている場合は、プロミスキュアスモードが不可になります。

プロミスキュアスマルチキャストモードの要求の場合、マルチキャスト専用プロミスキュアスモードを備えていないデバイスのドライバは、デバイスを物理プロミスキュアスモードに設定して、すべてのマルチキャストパケットが受信されるようにしなければなりません。その場合、ルーチンは GLD_SUCCESS を返さねばなりません。GLD ソフトウェアが余分なパケットをフィルタリングして除外します。そのとき、利用できるリソースが不足していて要求を満たせない場合、GLD_NORESOURCES を返します。要求された関数がサポートされていない場合は、GLD_NOTSUPPORTED を返します。

上位互換性を維持するために、gldm_set_promiscuous() ルーチンは、*promiscflag* として認識できないすべての値を GLD_PROMISC_PHYS のように扱う必要があります。

```
int prefix_send(gld_mac_info_t * macinfo, mblk_t * mp);
```

gldm_send() は、デバイスへのパケットを送信待ちのキューに入れます。このルーチンには、送信されるべきパケットが入った STREAMS メッセージが渡されます。メッセージには複数のメッセージブロックが含まれることがあり、送信されるパケット全体にアクセスするために、送信ルーチンはそのメッセージ内のすべてのメッセージブロックを通らなければなりません。連結内の長さがゼロのメッセージ継続ブロックを認識してスキップするように、ドライバを準備する必要があります。ドライバでは、パケットが許容される最大パケットサイズを超えていないかどうかをチェックし、必要であれば、許容される最小パケットサイズになるまでパケットにパディングをしなければなりません。送信ルーチンがパケットを正常に送信した場合、またはキューに格納した場合は、GLD_SUCCESS を返します。

送信ルーチンは、送信パケットをすぐに受け付けることができない場合、GLD_NORESOURCES を返します。その場合、GLD は後で再試行を行います。gldm_send() が GLD_NORESOURCES を返した場合、ドライバはリソースが利

用できるようになった時点で、gld_sched() を呼び出さなければなりません。この gld_sched() への呼び出しは、ドライバが以前送信用のキューに入れることができなかったパケットを再試行するように、GLD に通知します (ドライバの gldm_stop() ルーチンが呼び出されても、後に gldm_send() ルーチンから再び GLD_NORESOURCES を返すまで、ドライバはこの義務を免除されます。ただし、gld_sched() を余分に呼び出しても、誤った動作になることはありません)。

ドライバの送信ルーチンが GLD_SUCCESS を返し、ドライバとハードウェアでそのメッセージがもう不要になったときに、メッセージを解放するのはドライバの役目です。送信ルーチンがデバイスにメッセージをコピーした場合、または専用バッファにコピーした場合は、コピー後に送信ルーチンでメッセージを解放できます。ハードウェアが DMA を使用して、メッセージデータブロックから直接データを読み取る場合は、ハードウェアによるデータの読み取りが完了するまで、ドライバはメッセージを解放してはなりません。この場合、ドライバは割り込みルーチンでメッセージを解放するか、または将来の送信動作の開始時にバッファ再請求動作でメッセージを解放するのが一般的です。送信ルーチンが GLD_SUCCESS 以外のものを返した場合、ドライバはメッセージを解放してはなりません。ネットワークまたはリンクパートナーとの間に物理接続が無いときに gldm_send() が呼び出された場合は、GLD_NOLINK を返します。

```
int prefix_intr(gld_mac_info_t * macinfo);
```

gldm_intr() は、デバイスに割り込みがかけられている可能性がある場合に呼び出されます。割り込みを他のデバイスと共有する可能性があるので、ドライバはデバイスの状態を調べ、実際に割り込みが発生したかどうかを判別しなければなりません。ドライバが制御しているデバイスで割り込みが起きなかった場合、このルーチンは DDI_INTR_UNCLAIMED を返さなければなりません。それ以外の場合は、割り込みに対処し、DDI_INTR_CLAIMED を返さなければなりません。パケットの正常な受信によって割り込みが発生した場合、このルーチンは受信パケットを M_DATA タイプの STREAMS メッセージに格納し、メッセージを gld_recv() に渡す必要があります。

gld_recv() は、着信パケットをアップストリーム方向に、ネットワークプロトコルスタックの該当する次のレイヤーに渡します。gld_recv() を呼び出す前に、STREAMS メッセージの b_rptr および b_wptr メンバーを正しく設定しておくことが重要です。

ドライバは、gld_recv() の呼び出し時に、相互排他 (mutex) ロックまたは他のロックを保持していないようにしなければなりません。特に、送信スレッドが使用

する可能性のあるロックは、gld_recv() の呼び出し時に保持されていてはなりません。場合によっては、gld_recv() を呼び出す割り込みスレッドが、発信パケットの送信を含めた処理を実行してしまい、その結果、ドライバの gldm_send() ルーチンが呼び出されてしまうことがあるためです。gldm_intr() ルーチンが gld_recv() を呼び出すときに保持していた相互排他ロックを、gldm_send() ルーチンが取得しようとした場合、相互排他エントリが繰り返されることになり、パニックに陥る可能性があります。gld_recv() を呼び合いドライバが保持する相互排他ロックを他のドライバのエントリポイントが取得しようとした場合、デッドロックに陥る可能性があります。

割り込みコードは、あらゆるエラーに関する統計カウンタを増分しなければなりません。このエラーには、受信データ用バッファの割り当てエラーをはじめ、CRC エラーやフレーミングエラーなどのハードウェア固有のエラーが含まれます。

```
int prefix_get_stats(gld_mac_info_t * macinfo, struct gld_stats * stats);
```

gldm_get_stats() は、ハードウェア、ドライバ専用カウンタ、またはその両方から統計情報を収集し、*stats* で指し示された gld_stats(9S) 構造体を更新します。このルーチンは、統計要求を受けた時に GLD によって呼び出され、GLD が統計要求に対する応答を作成する前に、ドライバからデバイスに依存する統計情報を得るためのメカニズムを提供します。定義されている統計カウンタの詳細は、gld_stats(9S) および gld(7D) のマニュアルページを参照してください。

```
int prefix_ioctl(gld_mac_info_t * macinfo, queue_t * q, mblk_t * mp);
```

gldm_ioctl() は、すべてのデバイス固有の ioctl コマンドを実装します。ドライバが ioctl 関数を全く実装しない場合、この要素は NULL として指定できます。メッセージブロックを ioctl 応答メッセージに変換し、GLD_SUCCESS を呼び出す前に greply(9F) 関数を呼び出すのは、ドライバの役目です。この関数は常に GLD_SUCCESS を返すべきです。ドライバに報告させる必要のあるエラーは、greply(9F) に渡すメッセージで返させる必要があります。gldm_ioctl 要素が NULL として指定されている場合、GLD は M_IOCNAK タイプのメッセージを EINVAL というエラーとともに返します。

戻り値

これまでに説明した戻り値および制限事項のほかに、一部の GLD エントリポイント関数は次の値を返すことができます。

GLD_BADARG	関数が不適切な引数、たとえば、不良マルチキャストアドレス、不良 MAC アドレス、不良パケット、不良パケット長などを検出した場合
GLD_FAILURE	ハードウェア障害の場合
GLD_SUCCESS	成功した場合

サービスルーチン

```
gld_mac_info_t * gld_mac_alloc(dev_info_t * dip);
```

gld_mac_alloc() は、新しい gld_mac_info(9S) 構造体を割り当て、その構造体に対するポインタを返します。この構造体の GLD 専用の要素のなかには、gld_mac_alloc() を返す前に初期化されるものがありますが、他の要素はすべてゼロに初期化されます。デバイスドライバは、mac_info 構造体へのポインタを gld_register() へ渡す前に、gld_mac_info(9S) に記述されているように、一部の構造体メンバーを初期化する必要があります。

```
void gld_mac_free(gld_mac_info_t * macinfo);
```

gld_mac_free() は、以前に gld_mac_alloc() によって割り当てられていた gld_mac_info(9S) 構造体を解放します。

```
int gld_register(dev_info_t * dip, char * name, gld_mac_info_t * macinfo);
```

gld_register() は、デバイスドライバの attach(9E) ルーチンから呼び出され、GLD ベースのデバイスドライバと GLD フレームワークを結びつけるために使用されます。デバイスドライバの attach(9E) ルーチンは、gld_register() を呼び出す前に、gld_mac_alloc() を最初に使用して gld_mac_info(9S) 構造体

を割り当て、その構造体の要素のいくつかを初期化しなければなりません。詳細は、gld_mac_info(9S) のマニュアルページを参照してください。gld_register() が正常に呼び出されると、次の動作が発生します。

- デバイス固有のドライバを GLD システムに接続する
- デバイス固有のドライバの専用データポインタが (ddi_set_driver_private(9F) を使用して) macinfo 構造体を指し示すように設定する
- マイナーデバイスノードを作成する
- DDI_SUCCESS を返す

gld_register() に渡すデバイスインタフェース名は、ファイルシステムに存在しているドライバモジュール名と完全に一致しなければなりません。

ドライバの attach(9E) ルーチンは、gld_register() が正常に完了した場合に DDI_SUCCESS を返す必要があります。gld_register() が DDI_SUCCESS を返さなかった場合、attach(9E) ルーチンは gld_register() を呼び出す前に割り当てたすべてのリソースの割り当てを解除し、その後 DDI_FAILURE を返さなければなりません。

```
int gld_unregister(gld_mac_info_t * macinfo);
```

gld_unregister() は、デバイスドライバの detach(9E) 関数によって呼び出され、成功した場合は次の作業を実行します。

- デバイスの割り込みが中止されたことを確認し、必要に応じてドライバの gldm_stop() ルーチンを呼び出す
- マイナーデバイスノードを削除する
- GLD システムとデバイス固有のドライバ間のリンクを切断する
- DDI_SUCCESS を返す

gld_unregister() が DDI_SUCCESS を返した場合、detach(9E) ルーチンは attach(9E) ルーチンで割り当てられたすべてのデータ構造体を割り当て解除し、gld_mac_free() を使用して、macinfo 構造を割り当て解除し、DDI_SUCCESS を返します。gld_unregister() が DDI_SUCCESS を返さなかった場合、ドライバの detach(9E) ルーチンはデバイスを動作状態にしたまま、DDI_FAILURE を返さなければなりません。

```
void gld_recv(gld_mac_info_t * macinfo, mblk_t * mp);
```

gld_recv() は、ドライバの割り込みハンドラによって呼び出され、受信したパケットをアップストリームに渡します。ドライバは raw パケットを格納した STREAMS M_DATA メッセージを作成して渡さなければなりません。gld_recv() がパケットのコピーを受け取るべき STREAMS キュー (あれば) を判別し、必要に応じてコピーします。さらに、必要であれば DL_UNITDATA_IND メッセージをフォーマットして、データを該当するすべての Stream に渡します。

ドライバは、gld_recv() の呼び出し時に、相互排他 (mutex) ロックまたは他のロックを保持していないようにしなければなりません。特に、送信スレッドが使用するロックは、gld_recv() の呼び出し時には保持できません。場合によっては、gld_recv() を呼び出す割り込みスレッドが、発信パケットの送信を含めた処理を実行してしまい、その結果、ドライバの gldm_send() ルーチンが呼び出されることがあるためです。gldm_intr() ルーチンが gld_recv() を呼び出すときに保持していた相互排他ロックを、gldm_send() ルーチンが取得しようとした場合、相互排他エントリが繰り返されることになり、パニックになる可能性があります。gld_recv() を呼び合いドライバが保持する相互排他ロックを他のドライバのエントリポイントが取得しようとした場合、デッドロックに陥る可能性があります。

```
void gld_sched(gld_mac_info_t * macinfo);
```

gld_sched() は、据え置かれていた発信パケットを再スケジューリングするために、デバイスドライバによって呼び出されます。ドライバの gldm_send() ルーチンが GLD_NORESOURCES を返すたびに、ドライバは後で gld_sched() を呼び出して、以前送信できなかったパケットについて再試行するように、GLD フレームワークに通知しなければなりません。gld_sched() は、リソースが利用可能になった時点で、できるかぎり迅速に呼び出され、GLD がドライバの gldm_send() ルーチンに対する発信パケットの受け渡しを、タイミングよく再開できるようにしなければなりません (ドライバの gldm_stop() ルーチンが呼び出されても、gldm_send() ルーチンから再び GLD_NORESOURCES が返されるまで、ドライバはこの義務を免除されます。ただし、gld_sched() を余分に呼び出しても、誤った動作になることはありません)。

```
uint_t gld_intr(caddr_t);
```

gld_intr() は、GLD のメインの割り込みハンドラです。通常、デバイスドライバの ddi_add_intr(9F) 呼び出しで、割り込みルーチンとして指定します。割り込みハンドラに対する引数 (ddi_add_intr(9F) 呼び出しに *int_handler_arg* として指定) は、gld_mac_info(9S) 構造体へのポインタでなければなりません。gld_intr() は、該当する場合、デバイスドライバの gldm_intr() 関数を呼び出し、そのポインタを gld_mac_info(9S) 構造体に渡します。しかし、ドライバが上位レベルの割り込みを使用する場合は、独自の上位割り込みハンドラを提供し、その中からソフト割り込みを起動しなければなりません。この場合、gld_intr() は通常、ddi_add_softintr() 呼び出しにソフト割り込みハンドラとして指定されます。gld_intr() は、割り込みハンドラに適した値を返します。

高可用性ドライバ

注 - 最新のマニュアルページを参照するには、man コマンドを使用してください。Solaris 8 Update リリースのマニュアルページには、「*Solaris 8 Reference Manual Collection*」には記載されていない新しい情報も提供されています。

この機能は、Solaris 8 10/00 リリースで追加されました。

可用性 (availability) は、障害の発生率と修復速度の両方により決定されます。多くの場合、個々のデバイスで障害が発生しても、トータルなシステム障害に至るわけではありません。冗長構成のハードウェアコンポーネントを高可用性をサポートする設計のドライバと組み合わせることで、個々のコンポーネントで障害が発生しても、システム動作を継続することができます。このようなドライバを使用している場合は一般に、サービスを継続したまま、システムを修復できます。

デバイス障害がもたらすドライバ障害をプログラミングによって排除することを、ドライバの強化 (driver hardening) といいます。強化されたドライバは、故障したデバイスからシステムの他の部分にエラーが拡がらないように、エラーに耐え、他を守ることができます。

障害の特定および回復/修理所要時間の短縮に役立つ、ドライバ内部の機能により、システムのサービス利用可能性が向上します。その結果、修復所要時間が短縮され、可用性が向上します。

Solaris デバイスドライバの作成方法に関する補足情報については、『*Writing Device Drivers*』を参照してください。

ドライバの強化

強化 (hardening) とは、ドライバが制御する入出力デバイスで障害が発生しても、またはシステムコアの外部に起因するその他の障害が発生しても、そのドライバが正常に動作するようにするプロセスです。強化されたドライバは、パニックに陥ったり、システムをハングさせたり、またこのような障害の結果として壊れたデータを無秩序に拡散させたりしてはなりません。

ドライバの開発者は、次の役割を担います。

- DDI 関数を正しく使用する
- デバイス入出力の破壊を検出して報告する
- 異質な割り込みロジックのデバイスを処理する

すべての Solaris ドライバは強化されるべきです。強化されたドライバは、次の規則を守ります。

- ハードウェアの各部分をそれぞれ別個のデバイスドライバインスタンスで制御する必要があります。
- プログラムされた入出力 (PIO) は、必ず、適切なデータアクセスハンドルを使用して、DDI アクセス関数からのみ実行されなければなりません。
- デバイスドライバは、デバイスから受け取ったデータが壊れている可能性があることを想定しなければなりません。ドライバは、データを使用する前に、データの整合性をチェックしなければなりません。
- ドライバは、検出した障害の影響を制御しなければなりません。既知の不良データをシステムの他の部分に流してはなりません。
- ドライバは、デバイスによる DMA バッファ (`DDI_DMA_READ`) へのすべての書き込みが、ドライバが全面的に制御するメモリーページ内に収まるようにしなければなりません。これにより、DMA の障害が原因でシステムメインメモリーの部分が破壊される事態を防止します。
- デバイスドライバは、デバイスがロックアップした場合に、システムリソースを無制限に割り当ててはなりません。デバイスが連続使用を要求した場合には、ドライバはタイムアウトすべきです。ドライバはさらに、有害な stuck 割り込み要求を検出し、適切な処置を取らなければなりません。
- ドライバは、障害発生後にリソースを解放しなければなりません。たとえば、ハードウェア障害の後でも、システムはすべてのマイナーデバイスをクローズし、ドライバインスタンスを切り離すことができなければなりません。

デバイスドライバのインスタンス

Solaris カーネルは、複数のドライバのインスタンスの使用を認めます。インスタンスごとに専用のデータ領域を持ちますが、テキストおよび一部のグローバルデータは、他のインスタンスとの共有になります。デバイスはインスタンス単位で管理されます。強化されたドライバは、ドライバがフェイルオーバーを内部的に処理する設計になっていない限り、ハードウェアの部分ごとに別々のインスタンスを使用すべきです。たとえば、多機能カードなど、スロットごとに複数のドライバインスタンスが存在することになりますが、これは Solaris デバイスドライバの標準の動作です。

DDI アクセスハンドルの排他的使用

強化されたドライバによるすべてのプログラムされた入出力 (PIO) アクセスでは、必ず、`ddi_get`*X*、`ddi_put`*X*、`ddi_rep_get`*X*、および `ddi_rep_put`*X* ルーチンファミリーに含まれる Solaris DDI アクセス関数を使用しなければなりません。ドライバが、`ddi_regs_map_setup(9F)` から返されたアドレスを使用して、マップされたレジスタに直接アクセスしてはなりません。アクセスハンドルを使用することによって、入出力エラーが確実に制御され、その影響が戻り値に限定されるので、マシン状態の他の部分を壊す可能性が減少します (アクセスハンドルを使用しないため、`ddi_peek(9F)` および `ddi_poke(9F)` ルーチンは避けてください)。

DDI アクセスメカニズムは、カーネルにデータを読み込む方法を制御する機会を提供するので重要です。DDI アクセスルーチンは、バスタイムアウトトラップの影響を限定することによって、保護能力を提供します。

破壊されたデータの検出

以下の節では、どこでデータ破壊が発生する可能性があるか、およびそれらを検出するにはどのような手順が必要かについて説明します。

デバイス管理および管理データの破壊

ドライバは、PIO によるか DMA によるかを問わず、デバイスから取得したデータは壊れている可能性があるということを想定しなければなりません。特に、デバイスから得たデータから読み取った、または計算されたポインタ、メモリーオフセット、配列インデックスは、慎重に扱ってください。この種の値は有害な場合があり、参照に利用される場合、カーネルパニックを引き起こす可能性があります。この種の値はすべて、範囲および整列 (必要な場合) をチェックしてから使用してください。

ポインタが有害でなくても、誤った結果を導く可能性もあります。たとえば、オブジェクトの有効なインスタンスを指し示していても、該当する正しいインスタンスではないことがあります。可能であれば、ドライバはポインタと指示先オブジェクトをクロスチェックするか、またはポインタによって取得したデータの妥当性を検証すべきです。

他のタイプのデータ (パケット長、ステータスワード、チャネル ID など) も誤った結果を導く可能性があります。データの各タイプを可能な範囲でチェックすべきです。たとえば、パケット長の場合は、範囲をチェックして、負ではないか、収容バッファより大きくないかどうかを確認できます。また、ステータスワードの場合は、「不可能」ビットの有無をチェックできます。チャネル ID は有効 ID のリストと突き合わせます。

値を使用して Stream を識別する場合、ドライバは Stream が実際に存在しているかどうかを確認しなければなりません。STREAMS 処理は非同期の性質があるので、デバイスの割り込みがまだ処理されないうちに、Stream が取り除かれてしまうことがあります。

ドライバはデバイスからデータを再読み込みしてはなりません。データは 1 回だけ読み取って、妥当性を検証し、ドライバのローカル状態で格納します。こうすることによって、最初に読み取って検証したときには正しかったが、後で再読み取りしたときには正しくないというデータによる危険性を回避することができます。

さらに、連続して `BUSY` 状態を返すデバイス、または別のバッファ処理を要求するデバイスがシステム全体をロックアップすることがないように、ドライバで、すべてのループがバインドされていることも確認すべきです。

受信したデータの破壊

デバイスエラーによって、受信バッファに壊れたデータが格納されてしまうことがあります。このような破壊を、デバイスドメインの外 (たとえば、ネットワーク内など) で発生した破壊と区別することはできません。通常は、プロトコルスタックのトランスポートレイヤーやデバイスを使用するアプリケーション内部での整合性チェックなど、この種の破壊に対処する既存のソフトウェアが既に配備されています。

ディスクドライバなどのように、受信したデータの整合性を上位レイヤーでチェックしない場合は、ドライバ内部で整合性をチェックすることができます。受信したデータの破壊を検出する方法は、一般にデバイス固有です (チェックサム、CRC など)。

障害の検出

従来のデバイスドライバでは、障害を検出した場合に、デバイスまでのデータパスを無効にできます。PIO アクセスが無効にされている場合にそのデバイスから読み取りを行おうとすると、undefined 値が返り、書き込みが無視されます。DMA アクセスが無効にされている場合は、デバイスはメモリーへのアクセスが禁止される可能性があり、または読み取り時に未定義データを受け取り、書き込みが廃棄される可能性があります。

デバイスドライバは次の DDI ルーチンを使用することによって、データパスが無効にされているかどうかを検出することができます。

- ddi_check_acc_handle(9F)
- ddi_check_dma_handle(9F)

各関数は、指定のハンドルによって示されたデータパスに影響を与えるような障害が検出されているかどうかを調べます。いずれかの関数が DDI_FAILURE を返した場合、そのデータパスに障害があります。ドライバは ddi_dev_report_fault(9F) を使用してその障害を報告し、必要なクリーンアップを実行し、さらに可能であれば、呼出側に適切なエラーを返す必要があります。

障害の封じ込め

システムの整合性を維持するには、システム状態が変更されないうちに障害を検出する必要があります。したがって、ドライバは、システムが使用する予定のデータがデバイスから返されるたびに、障害の有無をテストしなければなりません。

- 上位レイヤーにデータブロックを渡す直前など、重要な接合部分で、ddi_check_acc_handle(9F) および ddi_check_dma_handle(9F) を呼び出す必要があります。
- デバイスで障害が発生した場合は、ドライバから外部へデータを転送してはなりません。
- ドライバでは、障害がシステムの整合性に与える影響が他に無いかどうかをチェックしなければなりません。ドライバは、データを転送できない場合には、メモリーなどのカーネルリソースが永久に失われていないかどうかを確認しなければなりません。発生することのない信号を待機して、スレッドがブロックのままになっていてはなりません。

- ドライバは、障害状態時の処理を制限する必要があります (たとえば、wput ルーチンでメッセージを解放する、障害のあるボードからの割り込みを永久的に無効にするようにする、など)。

DMA の切り離し

障害のあるデバイスは、バス上で不適切な DMA 転送を開始する可能性があります。このデータ転送によって、配信済みの正常データが破壊されることがあります。障害のあるデバイスは、そのデバイスのドライバに属さないメモリーにまで悪影響をおよぼすような破壊されたアドレスを生成しかねません。

IOMMU が備わったシステムでは、デバイスは DMA 用に書き込み可能としてマップされたページに限って書き込むことができます。したがって、DMA 書き込みのターゲットとなるページは、1 つのドライバインスタンスが単独で所有し、他のカーネル構造体と共有することはありません。該当するページが DMA 用に書き込み可能としてマップされている場合、ドライバはそのページのデータを疑ってみるべきです。ページをドライバの外部に渡す前に、またはデータを検証する前に、ページと IOMMU のマッピングを解除しなければなりません。

ddi_umem_alloc(9F) を使用して、整合ページ全体が割り当てられる、または複数のページを割り当てて最初のページ境界より下のメモリーを無視することができます。ddi_ptob(9F) を使用すると、IOMMU ページのサイズを調べることができます。

あるいは、ドライバは、メモリーの安全な部分にデータをコピーしてから、そのデータを処理することもできます。この場合、最初に ddi_dma_sync(9F) を使用してデータを同期させる必要があります。

ddi_dma_sync(9F) を呼び出すときには、DMA を使用してデバイスにデータを転送する前に SYNC_FOR_DEV を指定し、デバイスからメモリーへ DMA を使用してデータを転送した後で SYNC_FOR_CPU を指定する必要があります。

IOMMU が備わった一部の PCI ベースのシステムでは、デバイスは PCI デュアルアドレスサイクル (64 ビットアドレス) を使用して、IOMMU をバイパスすることができます。その結果、デバイスにおいてメインメモリーの領域が破壊されてしまう潜在的可能性が生じます。強化されたデバイスドライバでは、このようなモードを使用しようとしてはならず、使用不可にしておくべきです。

stuck 割り込みの処理

割り込みが絶えないと、システムパフォーマンスが大幅に低下し、シングルプロセッサマシンの場合はほぼ確実に作業が進まなくなってしまうので、ドライバは stuck 割り込みを識別する必要があります。

時にはドライバにとって、特定の割り込みを偽として識別するのが困難な場合があります。ネットワークドライバの場合、受信した割り込みが指示されても、新しいバッファが利用できなければ作業は不要です。これが単独で発生した場合は問題ありません。実際の作業は別のルーチン (たとえば読み取りサービスなど) によってすでに完了している可能性があるからです。

一方、ドライバが処理する作業を伴わない割り込みが連続した場合、stuck 割り込みラインを示していることがあります。そのため、防御手段を講じる前に、すべてのプラットフォームで多数の明らかに偽の割り込みの発生を許してしまうことになります。

処理すべき作業がありそうなのにハングしてしまったデバイスは、対応するバッファ記述子を更新できなかった可能性があります。ドライバは、このような繰り返しの要求に対して防御しなければなりません。

場合によっては、プラットフォーム固有のバスドライバに請求によらない持続的な割り込みを識別する能力があり、攻撃しているデバイスを使用禁止にできます。しかし、これは有効な割り込みを識別して適切な値を返すことができるという、ドライバの能力に依存します。ドライバはしたがって、デバイスが正当な割り込みをかけた (すなわちデバイスがドライバになんらかの有用な作業を行うことを実際に要求している) ことを検出した場合以外は、`DDI_INTR_UNCLAIMED` を返さなければなりません。

偶発性の高い他の割り込みの正当性を確認することは、さらに困難です。この目的のためには、割り込みが有効かどうかを評価する手段として、割り込み想定フラグが役立ちます。デバイスの記述子が全部すでに割り当てられている場合に生成できる、descriptor free などの割り込みを例として仮定してください。ドライバはカードの最後の記述子を使用してしまったことを検出すると、割り込み想定フラグを設定できます。対応する割り込みが伝達された時にこのフラグが設定されていないと、疑わしいといえます。

メディアが切断された、またはフレーム同期が失われたといったことを伝える、情報通知目的の割り込みには、予測できないものがあります。このような割り込みが stuck かどうかを検出する最も簡単な方法は、最初のイベントでその送信元を次のポーリングサイクルまでマスクすることです。

禁止されている間に割り込みが再び発生した場合は、偽の割り込みとみなすべきです。デバイスによっては、マスクレジスタが対応する送信元を無効にし、割り込みを発生させない場合でも読み取ることのできる、割り込み状態ビットがあります。ドライバの設計者は、それぞれのデバイスに合わせて、より適切なアルゴリズムを工夫することができます。

割り込み状態ビットが無限ループに陥らないようにしてください。パスの最初に設定された状態ビットセットがいずれも実際の作業を必要としない場合は、このようなループを切断してください。

ドライバの強化に関するその他の考慮事項

すでに説明した要件のほかに、ドライバの開発者は、次の問題も考慮する必要があります。

- スレッドのインタラクション
- トップダウン式要求における危険
- 代替の対応策

スレッドのインタラクション

デバイスドライバのカーネルパニックは、デバイス障害の発生後に生じた予想外のカーネルスレッドのインタラクションによって引き起こされることがよくあります。デバイスで障害が発生すると、設計者が予想しなかった形でスレッドのインタラクションが起きることがあります。

たとえば、処理ルーチンが正常に完了しないうちに打ち切られた場合、条件変数を待機している他のスレッドに知らせることができないことがあります。他のモジュールに障害を知らせようとしたり、または予想外のコールバックを処理しようとしたりすると、望ましくないスレッドのインタラクションが発生する可能性があります。デバイス障害時に発生する可能性のある、相互排他ロックの取得と放棄の順序を検証してください。

アップストリームの STREAMS モジュールを起点とするスレッドは、予想外にそのモジュールをコールバックするために使用された場合、望ましくないパラドックスに陥る可能性があります。代替スレッドを使用して、例外メッセージを処理してください。たとえば、wput 手順では、読み取り側の putnext で直接通信するのではなく、読み取り側のサービスルーチンを使用して M_ERROR と通信することができます。

障害の発生したSTREAMSデバイスが、クローズ時に(障害が原因で)静止できなかった場合、Streamを取り除いた後に割り込みが発生する可能性があります。割り込みハンドラは、古いStreamポインタを使用して、メッセージを処理しようとしてはなりません。

トップダウン式要求における危険

ドライバの設計者は、ハードウェアの故障からシステムを保護する一方で、ドライバの誤用に対しても防御する必要があります。ドライバは、カーネル基盤は常に正しい(信頼できるコア)ということを前提にできますが、ドライバに渡されるユーザー要求は潜在的な破壊性を有している可能性があります。

たとえば、ユーザーが提供したデータブロック(M_IOCTL)に対してアクションを実行することをユーザーが要求し、そのデータブロックがメッセージの制御部で指示されたサイズより小さいという場合があります。ドライバはユーザーアプリケーションを信用してはなりません。

設計時には、害を引き起こす可能性があるという観点から、受け取ることのできる各タイプのioctlの構造を検討すべきです。ドライバは、異常なioctlを処理しないように、チェックしなければなりません。

代替の対応策

ドライバは障害の起きたハードウェアを引き続き使用してサービスを提供し、代替のデバイスアクセス方法を使用することによって、特定された問題への対処を試みることができます。ハードウェアの故障が予測不能で、設計が複雑になるリスクを伴うことを考えると、代替の対応策をとることが常に賢明な選択というわけではありません。せいぜい、周期的な割り込みポーリングと再試行に限定すべきです。周期的にデバイスを再試行することによって、ドライバはデバイスが復旧したかどうかを知ることができます。周期的ポーリングを使用すると、ドライバが割り込みを強制的に禁止された後でも、割り込みメカニズムを制御することができます。

重要なシステムサービスを提供できるように、システムが常に代替デバイスを備えているのが理想です。カーネルまたはユーザースペースのサービスマルチプレクサは、デバイス障害時にシステムのサービスを維持する最良の手段です。しかし、この章ではこの種の方式については記載していません。

サービス利用可能性

サービス利用可能性を確保するには、ドライバが下記を実行できるようにしなければなりません。

- 障害デバイスを検出して障害を報告する
- デバイスを取り除く (Solaris のホットプラグモデルでサポートされる)
- 新しいデバイスを追加する (Solaris のホットプラグモデルでサポートされる)
- 潜在的な障害を検出できるように定期的な健全性チェックを実行する

現在のデバイス状態のチェック

ドライバはリソースの無駄なコミットを避けるために、適切なポイントでデバイス状態をチェックしなければなりません。`ddi_get_devstate(9F)` 関数を使用すると、ドライバでフレームワークによって維持されているデバイスの現在の状態を判別できます。

```
ddi_devstate_t ddi_get_devstate(dev_info_t *dip);
```

ドライバは通常、`OFFLINE` のデバイスを扱うときには呼び出されません。デバイス状態は一般に、再構成アクティビティによって変更されたというような、以前のデバイス障害レポートを反映します。

デバイス障害時の適切な動作

システムは、デバイスのサービス利用可能性に対する影響という観点から、障害を報告しなければなりません。一般に次の場合に、サービス損失が予期されます。

- PIO または DMA エラーが検出された。
- データ破壊が検出された。
- デバイスがロックされている、またはハングしている (たとえば、コマンドが完了しない場合など)。
- ドライバの設計時にあり得ないと見なされたために、ドライバが対処しないことになっている状態が発生した。

ddi_get_decstate(9F) から返されたデバイス状態が、デバイスが使用不能であることを示している場合、ドライバは新規または未処理の入出力要求をすべて拒否し、(可能であれば) 適切なエラーコード (EIO など) を返す必要があります。STREAMS ドライバでは、M_ERROR または M_HANGUP をアップストリームに送り、ドライバが使用不能であることを示さなければなりません。

各メジャーエントリポイントで、またオプションで操作のためにリソースをコミットする前、および障害の報告後に、デバイスの状態をチェックする必要があります。いずれかの段階で、デバイスが使用不能であることが判明した場合は、ドライバで必要なクリーンアップ処理 (リソースの解放など) を実行し、タイミングよく戻る必要があります。再試行または回復処理を試行してはなりません。障害の報告も不要です。状態は障害ではなく、状態はすでにフレームワークと管理エージェントに認識されています。状態は、現在の要求および他の未処理すなわちキュー内の要求に完了マークを付け、可能であれば再びエラー標識を出します。

ioctl() エントリポイントは、デバイスが使用不能で、他 (エラー状態の回復など) が引き続き動作中である場合、所定の ioctl 操作における、デバイスに対する入出力 (たとえば、ディスクのフォーマットなど) 障害を意味する問題を提示します。したがって、状態チェックはコマンド単位で行う必要があります。または、別のエントリポイントまたはマイナーデバイスモードを使用して、あらゆる状態でこれらの操作を実装できますが、この方法は既存アプリケーションとの互換性の点から制約を受ける可能性があります。

close() は、デバイスが使用不能な場合であっても、常に正常に完了しなければなりません。デバイスが使用不能な場合は、割り込みハンドラがその後のすべての割り込みに DDI_INTR_UNCLAIMED を返す必要があります。割り込みが引き続き発生する場合は、最終的に割り込みが禁止されることがあります。

障害の報告

次の関数は、ドライバがデバイス障害を検出したことをシステムに通知します。

```
void ddi_dev_report_fault(dev_info_t *dip, ddi_fault_impact_t impact,
            ddi_fault_location_t location, const char *message);
```

impact パラメータは、デバイスの通常のサービス利用可能性に障害が与える影響を示します。このパラメータは、障害に対する適切な処置を決定するために、システムの障害管理コンポーネントが使用します。この処置によってデバイス状態が変化することがあります。サービスが損われる障害が発生すると、デバイス状態が

DOWN に変わります。サービスが低下する障害の場合は、デバイス状態が DEGRADED に変わります。

デバイスは次の場合に、障害として報告されなければなりません。

- PIO エラーが検出された。
- 破壊されたデータが検出された。
- デバイスがロックアップされた。

できるだけ、ドライバが同じ障害を繰り返し報告しないようにします。特に、デバイスがすでに使用不能状態になっている場合に、ドライバがエラーを報告しても無駄です (望ましくありません) (ddi_get_devstate(9F) のマニュアルページを参照)。

接続プロセスでハードウェア障害が検出された場合、ドライバは ddi_dev_report_fault(9F) のを使用して障害を報告するとともに、DDI_FAILURE を返さなければなりません。

定期的な健全性チェック

潜在的な障害とは、他のアクションが発生するまで明らかにならない障害のことです。たとえば、コールドスタンバイモードのデバイスで発生したハードウェア障害は、マスターデバイスで障害が発生するまで検出されない可能性があります。マスターデバイスで障害が発生した時点で、システムに 2 つの障害デバイスがあり、動作を継続できない可能性があることが判明します。

一般論として、未検出のままの潜在的障害は、最終的にシステム障害につながります。潜在的な障害を検査しなかった場合は、冗長型システム全体の可用性が損なわれる危険があります。これを回避するには、デバイスドライバで潜在的な障害を検出し、他の障害と同様に報告しなければなりません。

ドライバには、デバイスに対する定期的な健全性チェックを行うメカニズムが必要です。デバイスを第 2 デバイスすなわちフェイルオーバーデバイスにできる、障害耐久 (フォールトトレラント) の状況では、故障した第 2 デバイスを早い段階で検出することが重要です。そうすれば、第 1 デバイスで障害が発生しないうちに修理または交換することができます。

定期的な健全性チェックには、次のものがあります。

- ボードに対してクイックアクセスチェック (書き込み、読み取り) を実行し、さらに ddi_check_acc_handle(9F) ルーチンを使用してデバイスをチェックする。
- 前回のポーリング以後、値がドライバの予期値と決定的に異なっているデバイスについて、レジスタまたはメモリーロケーションをチェックする。

一般に決定的な動作を示すデバイス機能には、ハートビートセマフォ、デバイスタイマー (たとえば、ダウンロードで使用されるローカル `lbolt` など)、イベントカウンタがあります。デバイスから更新済みの予想可能値を読み取ると、進行が順調であるかどうか、ある程度確かなことがわかります。

- ドライバによる発行時に、発信要求 (送信ブロックまたはコマンド) にタイムスタンプを設定する。

  定期的な健全性チェックで、期限を過ぎていながら完了していない要求を検索できます。

- 予定された次回のチェックまでに完了すべき、デバイスに対するアクションを開始する。

  このアクションを割り込みにすると、デバイスの回路が割り込みのデリバリにまだ対応できるかどうかを確認するうえで理想的です。

第 4 章

その他のソフトウェア開発情報

この章では、一部の新しいロケールについて説明します。

注 - 最新のマニュアルページを参照するには、man コマンドを使用してください。Solaris 8 Update リリースのマニュアルページには、「*Solaris 8 Reference Manual Collection*」には記載されていない新しい情報も提供されています。

ヨーロッパ言語版 Solaris ソフトウェア用に追加された部分ロケール

この機能は、Solaris 8 10/00 リリースで追加されました。

新規機能として、ロシア語およびポーランド語対応の UTF-8 ロケール、さらにカタロニア語対応の新しい 2 つのロケールが追加されました。ロケール名は、次のとおりです。

- ru_RU.UTF-8
- pl_PL.UTF-8
- ca_ES.ISO8859–1
- ca_ES.ISO8859–15

追加されたロケールは、言語サポート (メッセージの翻訳および GUI) がないので、部分的ロケールです。

Solaris 製品のローカライゼーション

中央ヨーロッパ

表 4–1 中央ヨーロッパ

ロケール	ユーザーインタフェース	地域	コードセット	言語サポート
cs_CZ.ISO8859-2	英語	チェコ	ISO8859-2	チェコ語 (チェコ)
de_AT.ISO8859-1	ドイツ語	オーストリア	ISO8859-1	ドイツ語 (オーストリア)
de_AT.ISO8859-15	ドイツ語	オーストリア	ISO8859-15	ドイツ語 (オーストリア、ISO8859-15 - Euro)
de_CH.ISO8859-1	ドイツ語	スイス	ISO8859-1	ドイツ語 (スイス)
de_DE.UTF-8	ドイツ語	ドイツ	UTF-8	ドイツ語 (ドイツ、Unicode 3.0)
de_DE.ISO8859-1	ドイツ語	ドイツ	ISO8859-1	ドイツ語 (ドイツ)
de_DE.ISO8859-15	ドイツ語	ドイツ	ISO8859-15	ドイツ語 (ドイツ、ISO8859-15 - Euro)
fr_CH.ISO8859-1	フランス語	スイス	ISO8859-1	フランス語 (スイス)
hu_HU.ISO8859-2	英語	ハンガリー	ISO8859-2	ハンガリー語 (ハンガリー)
pl_PL.ISO8859-2	英語	ポーランド	ISO8859-2	ポーランド語 (ポーランド)
pl_PL.UTF-8	英語	ポーランド	UTF-8	ポーランド語 (ポーランド、Unicode 3.0)
sk_SK.ISO8859-2	英語	スロバキア	ISO8859-2	スロバキア語 (スロバキア)

東ヨーロッパ

表 4–2 東ヨーロッパ

ロケール	ユーザーインタフェース	地域	コードセット	言語サポート
bg_BG.ISO8859-5	英語	ブルガリア	ISO8859-5	ブルガリア語 (ブルガリア)
et_EE.ISO8859-15	英語	エストニア	ISO8859-15	エストニア語 (エストニア)
hr_HR.ISO8859-2	英語	クロアチア	ISO8859-2	クロアチア語 (クロアチア)
lt_LT.ISO8859-13	英語	リトアニア	ISO8859-13	リトアニア語 (リトアニア)
lv_LV.ISO8859-13	英語	ラトビア	ISO8859-13	ラトビア語 (ラトビア)
mk_MK.ISO8859-5	英語	マケドニア	ISO8859-5	マケドニア語 (マケドニア)
ro_RO.ISO8859-2	英語	ルーマニア	ISO8859-2	ルーマニア語 (ルーマニア)
ru_RU.KOI8-R	英語	ロシア	KOI8-R	ロシア語 (ロシア、KOI8-R)
ru_RU.ANSI1251	英語	ロシア	ansi-1251	ロシア語 (ロシア、ANSI 1251)
ru_RU.ISO8859-5	英語	ロシア	ISO8859-5	ロシア語 (ロシア)
ru_RU.UTF-8	英語	ロシア	UTF-8	ロシア語 (ロシア、Unicode 3.0)
sh_BA.ISO8859-2@bosnia	英語	ボスニア	ISO8859-2	ボスニア語 (ボスニア)
sl_SI.ISO8859-2	英語	スロベニア	ISO8859-2	スロベニア語 (スロベニア)
sq_AL.ISO8859-2	英語	アルバニア	ISO8859-2	アルバニア語 (アルバニア)
sr_YU.ISO8859-5	英語	セルビア	ISO8859-5	セルビア語 (セルビア)
tr_TR.ISO8859-9	英語	トルコ	ISO8859-9	トルコ語 (トルコ)

南ヨーロッパ

表 4–3 南ヨーロッパ

ロケール	ユーザーインタフェース	地域	コードセット	言語サポート
ca_ES.ISO8859-1	英語	スペイン	ISO8859-1	カタロニア語 (スペイン)
ca_ES.ISO8859-15	英語	スペイン	ISO8859-15	カタロニア語 (スペイン、ISO8859-15 - Euro)
el_GR.ISO8859-7	英語	ギリシャ	ISO8859-7	ギリシャ語 (ギリシャ)
es_ES.ISO8859-1	スペイン語	スペイン	ISO8859-1	スペイン語 (スペイン)
es_ES.ISO8859-15	スペイン語	スペイン	ISO8859-15	スペイン語 (スペイン、ISO8859-15 - Euro)
es_ES.UTF-8	スペイン語	スペイン	UTF-8	スペイン語 (スペイン、Unicode 3.0)
it_IT.ISO8859-1	イタリア語	イタリア	ISO8859-1	イタリア語 (イタリア)
it_IT.ISO8859-15	イタリア語	イタリア	ISO8859-15	イタリア語 (イタリア、ISO8859-15 - Euro)
it_IT.UTF-8	イタリア語	イタリア	UTF-8	イタリア語 (イタリア、Unicode 3.0)
pt_PT.ISO8859-1	英語	ポルトガル	ISO8859-1	ポルトガル語 (ポルトガル)
pt_PT.ISO8859-15	英語	ポルトガル	ISO8859-15	ポルトガル語 (ポルトガル、ISO8859-15 - Euro)

ヨーロッパのローカライゼーション

Solaris 8 ソフトウェアは、ユーロ通貨をサポートします。下位互換性を維持するために、各国のこれまでの通貨記号も引き続き使用できます。

表 4–4 ユーロ通貨をサポートするユーザーロケール

地域	ロケール名	**ISO** コードセット
オーストリア	de_AT.ISO8859-15	8859-15
ベルギー (フランス語)	fr_BE.ISO8859-15	8859-15
ベルギー (オランダ語)	nl_BE.ISO8859-15	8859-15
デンマーク	da_DK.ISO8859-15	8859-15
フィンランド	fi_FI.ISO8859-15	8859-15
フランス	fr_FR.ISO8859-15	8859-15
ドイツ	de_DE.ISO8859-15	8859-15
アイルランド	en_IE.ISO8859-15	8859-15
イタリア	it_IT.ISO8859-15	8859-15
オランダ	nl_NL.ISO8859-15	8859-15
ポルトガル	pt_PT.ISO8859-15	8859-15
スペイン	ca_ES.ISO8859-15	8859–15
スペイン	es_ES.ISO8859-15	8859-15
スウェーデン	sv_SE.ISO8859-15	8859-15
英国	en_GB.ISO8859-15	8859-15
米国	en_US.ISO8859-15	8859-15

第 5 章

開発のための Java 関連情報

この章では、新しい Java 機能について説明します。

注 - 最新のマニュアルページを参照するには、man コマンドを使用してください。Solaris 8 Update リリースのマニュアルページには、「*Solaris 8 Reference Manual Collection*」には記載されていない新しい情報も提供されています。

表 5–1　各 Update リリースにおける JDK のバージョン

Solaris 8 1/01 リリースには、前回のリリース以後のバグ修正によって改良された、JDK 1.2.2_06 および JDK 1.1.8_12 が組み込まれています。

Solaris 8 10/00 リリースには、バグ修正によって改良された JDK 1.1.8_10 が組み込まれています。

Solaris 8 10/00 リリースには、Java 2 Standard Edition v. 1.2.2_05a が組み込まれています。これは、Java 2 Standard Edition v. 1.2.2_05 (末尾に「a」が無い) のバグ修正版であり、次の新機能および拡張機能が組み込まれています。

- スケーラビリティの向上 (20 以上の CPU に対応)

  プリミティブおよびスレッドの並行処理能力の向上により、マルチスレッド化プログラムの性能が向上し、多数のスレッドを使用するプログラムにおいてガーベッジコレクションによる休止時間が大幅に削減されました。
- JIT コンパイラの最適化の向上

  JIT コンパイラが実行する新しい最適化機能は、仮想および非仮想方式のインライン化、拡張基本ブロック内の CSE、配列結合の検査を不要にするループ分析、およびより速いタイプチェックです。
- テキストの描画性能の向上

  Direct Graphics Access (DGA) サポートが組み込まれていない Solaris ソフトウェアプラットフォーム上の Java 2 Standard Edition では、いくつかのグラフィック最適化機能によって、テキストの描画性能が大幅に向上しました。これらのプラットフォームには、Ultra 5、Ultra 10、Solaris (Intel 版) オペレーティング環境、およびすべてのリモートディスプレイシステムが含まれます。
- `poller` クラスのデモパッケージ

  Java アプリケーションから C の `poll(2)` ルーチンの関数に、効率的にアクセスできるようになりました。これは、サンプルサーバーを組み込んだデモパッケージとして提供されます。
- Swing の向上

  Swing クラスに関して、品質と性能の両面で大幅な改善がなされました。詳細については、次の URL にアクセスしてください。
  - `http://Java.sun.com/products/jdk/1.2/changes.html`
  - `http://java.sun.com/products/jdk/1.2/fixedbugs/index.html`

Apache Web サーバーにおける Java Servlet のサポート

`mod_jserv` モジュールおよび関連ファイルの追加により、Apache Web サーバーソフトウェアで Java Servlet がサポートされるようになりました。現在、`/etc/apache` に次の構成ファイルが保存されています。

- `zone.properties`
- `jserv.properties`
- `jserv.conf`

`mod_jserv` モジュールは、他の Apache ソフトウェアと同様、オープンなソースコードであり、Sun 以外のグループによって保守されています。このグループは、旧リリースの Apache および `mod_jserv` との互換性の維持に努めています。

第 6 章

Solaris 8 マニュアルの変更点の概要

Solaris 8 のマニュアルのうち何冊かが Solaris 8 Update リリースで更新されました。この章は、これらのマニュアルにおける更新事項を記述します。

注 - 最新のマニュアルページを参照するには、man コマンドを使用してください。Solaris 8 Update リリースのマニュアルページには、「*Solaris 8 Reference Manual Collection*」には記載されていない新しい情報も提供されています。

『システムインタフェース』の更新

『システムインタフェース』が Solaris 8 6/00 リリースで更新され、旧版でのバグが修正されました。テキストおよびサンプルソースコードのタイプミスが修正されています。『システムインタフェース』を参照してください。

『リンカーとライブラリ』の更新

このマニュアルは、Solaris 8 1/01 リリースおよび Solaris 8 10/00 リリースで更新されました。

表 6–1 『リンカーとライブラリ』の更新事項

『リンカーとライブラリ』の Solaris 8 1/01 リリースでの更新事項	■ dladdr(3DL) から入手できるシンボリック情報が、dladdr1() の導入により拡張されました。 ■ 動的オブジェクトの $ORIGIN が dlinfo(3DL) から入手できます。 ■ crle(1) で作成された実行時構成ファイルの管理が、構成ファイルの作成に使用されたコマンドラインオプションの表示によって簡単になりました。また、拡張機能も利用できます (–u オプションを参照)。 ■ 実行時リンカーおよびデバッガインタフェースが拡張され、プロシージャリンクテーブルエントリの解決を検出できるようになりました。この拡張は、新しいバージョンナンバーで識別することができます。「エージェント操作」の rd_init() を参照してください。この更新により rd_plt_info_t 構造体が機能拡張されます。「プロシージャのリンクテーブルのスキップ」の rd_plt_resolution() を参照してください。 ■ 新しい mapfile セグメント記述子 STACK を使用してアプリケーションスタックを非実行可能ファイルに定義することができます。「セグメントの宣言」を参照してください。
『リンカーとライブラリ』の Solaris 8 10/00 リリースでの更新事項	■ 実行時リンカーが環境変数 LD_BREADTH を無視します。「初期設定および終了ルーチン」の節を参照してください。 ■ 実行時リンカーおよびそのデバッガインタフェースが拡張され、実行時分析とコアファイル分析の性能が向上しました。この更新は、新しいバージョン番号で識別されます。「エージェント操作」の節の rd_init() 関数を参照してください。この更新により、rd_loadobj_t 構造体の rl_flags、rl_bend、および rl_dynamic フィールドが拡張されました。「読み込み可能オブジェクトの走査」の節を参照してください。 ■ ディスプレイスメント再配置されたデータがコピー再配置で使用されるか、使用される可能性があることを検査する機能が提供されるようになりました。「ディスプレイスメント再配置」の節を参照してください。 ■ 64 ビットフィルタが、リンカーの -64 オプションを使用してマップファイルから単独で構築できるようになりました。「標準フィルタの生成」の節を参照してください。 ■ $ORIGIN 動的文字列トークンの拡張がなぜセキュアアプリケーション内に限定されるのかの説明が追加されました。「セキュリティ」の節を参照してください。 ■ 動的オブジェクトの依存関係の検索に使用される検索パスを、dlinfo(3DL) を使用して調べることができるようになりました。 ■ dlsym(3DL) と dlinfo(3DL) 検索の方法が新しいハンドル RTLD_SELF によって拡張されました。 ■ 動的オブジェクトの再配置に使用される実行時シンボル検索メカニズムは、各動的オブジェクト内に直接結合情報を確立することによって、大幅に削減されるようになりました。「外部結合」と「直接結合」の節を参照してください。

『*Solaris* モジューラデバッガ』の更新

このマニュアルは、Solaris 8 10/00 ソフトウェアリリースで更新されました。

『*Solaris* モジューラデバッガ』では、次の情報が更新されています。

- 第 3 章の「演算機能の拡張」の節が更新され、単項演算が含まれるようになりました。
- 技術的なマイナーエラーが修正されました。

『マルチスレッドのプログラミング』の更新

このマニュアルは、Solaris 8 1/01 ソフトウェアリリースで更新されました。

『マルチスレッドのプログラミング』が更新され、バグ ID 4308968、4356675、4356690 が修正されました。