(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第3992479号
(P3992479)
(45)発行日　平成19年10月17日(2007.10.17)　(24)登録日　平成19年8月3日(2007.8.3)

(51)Int.Cl.		F I		
G06F 21/24	(2006.01)	G06F	12/14	520A
G06F 3/06	(2006.01)	G06F	3/06	301A
G06F 12/16	(2006.01)	G06F	12/16	320L

請求項の数 5　(全 24 頁)

(21)出願番号　特願2001-344970(P2001-344970)
(22)出願日　平成13年11月9日(2001.11.9)
(65)公開番号　特開2002-202914(P2002-202914A)
(43)公開日　平成14年7月19日(2002.7.19)
審査請求日　平成16年10月27日(2004.10.27)
(31)優先権主張番号　09/726,852
(32)優先日　平成12年11月30日(2000.11.30)
(33)優先権主張国　米国(US)

(73)特許権者　398038580
ヒューレット・パッカード・カンパニー
HEWLETT-PACKARD COMPANY
アメリカ合衆国カリフォルニア州パロアルト　ハノーバー・ストリート　3000
(74)代理人　100081721
弁理士　岡田　次生
(74)代理人　100105393
弁理士　伏見　直哉
(74)代理人　100111969
弁理士　平野　ゆかり

最終頁に続く

(54)【発明の名称】大容量記憶装置により提供される、制御装置ＬＵＮを介したＬＵＮへのアクセスを安全にするシステム

(57)【特許請求の範囲】

【請求項１】

大容量記憶装置により提供される論理ユニットにリモート・エンティティからアクセスすることを許可する許可システムであって、リモート・エンティティにより生成される動作の実行を求める要求を検出する要求検出コンポーネント、特定の論理ユニットにアクセスする、特定のリモート・エンティティの許可をそれぞれ表すエントリを含むアクセス・テーブル、特定の論理ユニットにアクセスする、特定の制御装置論理ユニットの許可をそれぞれ表すエントリを含む追加アクセス・テーブル、および前記指定された制御装置論理ユニットにアクセスする、前記リモート・エンティティの許可を表すエントリが前記アクセス・テーブル内に存在し、前記１つまたは複数の他の指定された論理ユニットのそれぞれに、前記他の指定された論理ユニットにアクセスする、前記指定された制御装置論理ユニットの許可を表すエントリが追加アクセス・テーブル内に存在する場合に限り、リモート・エンティティにより要求され、前記要求検出コンポーネントにより検出され、指定された制御装置論理ユニットを対象とし、１つまたは複数の他の指定された論理ユニットを含む要求を許可する制御論理を備えるシステム。

【請求項２】

前記大容量記憶装置が、リモート・エンティティからの要求を受け取るポートを含み、前記制御論理が前記大容量記憶装置内にある、請求項１に記載のシステム。

【請求項３】

論理ユニットまたは制御装置論理ユニットの標示、ポートの標示、およびリモート・エ

ンティティの標示をそれぞれ含むエントリを前記アクセス・テーブルが含む、請求項２に記載のシステム。

【請求項４】

制御装置論理ユニットの標示、および論理ユニットの標示をそれぞれ含むエントリを前記追加アクセス・テーブルが含む、請求項２に記載のシステム。

【請求項５】

前記大容量記憶装置がディスク・アレイであり、リモート・エンティティが、通信媒体を介して前記ディスク・アレイと相互接続されたリモート・コンピュータである、請求項２に記載のシステム。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、大容量記憶装置内に用いられるセキュリティ機構に関し、特に制御装置論理ユニットを介してリモート・コンピュータから論理ユニットに間接アクセスする間に、大容量記憶装置によってリモート・コンピュータに提供された論理ユニットへのアクセスを確保する方法およびシステムに関する。

【０００２】

本発明は、リモート・コンピュータがアクセス特権を有する大容量記憶装置内に格納されたデータに限って、リモート・コンピュータがアクセスを得ることができるように保証することに関する。本発明を、いくつかのリモート・コンピュータからのＩ／Ｏ要求に対して機能するディスク・アレイ・コントローラ内に含まれる一実施形態を参照しながら述べ、例示する。しかし、他の多数のタイプの記憶装置のコントローラおよび一般的電子サーバ・アプリケーションにおいて、本発明の代替実施形態を用いることができる。

【０００３】

【従来の技術】

図１は標準ディスク・ドライブのブロック図である。ディスク・ドライブ１０１は、コンピュータ・バス、ファイバ・チャネル、または他の類似の電子通信媒体などの通信媒体１０２を介して、リモート・コンピュータからＩ／Ｏ要求を受け取る。図１に示すディスク・ドライブ１０１を含む多数のタイプの記憶装置にとって、大部分のＩ／Ｏ要求はＲＥＡＤまたはＷＲＩＴＥ要求のいずれかである。ＲＥＡＤ要求は、記憶装置内に格納された電子データのうち、いくらかの要求された量を記憶装置が要求元リモート・コンピュータに返すことを要求する。ＷＲＩＴＥ要求は、リモート・コンピュータから供給された電子データを記憶装置が格納することを要求する。したがって、記憶装置により実行されたＲＥＡＤ動作の結果として、データは通信媒体１０２を介してリモート・コンピュータに返され、ＷＲＩＴＥ動作の結果として、データは、通信媒体１０２を介してリモート・コンピュータから記憶装置に受け取られ、記憶装置内に格納される。

【０００４】

図１に示されるディスク・ドライブ記憶装置は、電子メモリ、１つまたは複数のプロセッサまたは処理回路、およびコントローラ・ファームウェアを含むコントローラ・ハードウェアおよび論理１０３を含み、電子データを格納するために磁気媒体が塗布されたいくつかのディスク・プラッタ１０４もまた含む。ディスク・ドライブは、読取り／書込みヘッド、高速電子モータ、ドライブ・シャフト、ならびに他の電子的、機械的、および電子機械的コンポーネントを含む、図１に示されていない他の多数のコンポーネントを含む。ディスク・ドライブ内のメモリは、リモート・コンピュータから受け取ったＩ／Ｏ要求を格納する要求／応答バッファ１０５、および要求／応答バッファ１０５内に格納されたＩ／Ｏ要求に対応する内部Ｉ／Ｏコマンドを格納するＩ／Ｏキュー１０６を含む。中でも、リモート・コンピュータとディスク・ドライブの間の通信、Ｉ／Ｏ要求から内部Ｉ／Ｏコマンドへの変換、そしてＩ／Ｏキューの管理は、ディスク・ドライブＩ／Ｏコントローラにより、ディスク・ドライブＩ／Ｏコントローラ・ファームウェア１０７によって指定されたとおりに実行される。内部Ｉ／Ｏコマンドを、データがディスク・プラッタ１０４上に

格納またはそこから取得される電子機械的ディスク動作に変換することは、ディスク・ドライブＩ／Ｏコントローラにより、ディスク媒体読取り／書込み管理ファームウェア１０８によって指定されたとおりに実行される。したがってディスク・ドライブＩ／Ｏ制御ファームウェア１０７およびディスク媒体読取り／書込み管理ファームウェア１０８、ならびにファームウェアの実行を可能にするプロセッサおよびメモリが、ディスク・ドライブ・コントローラを構成する。

【０００５】

２つのリモート・コンピュータにより使用されるデュアル・ポート・ディスク・ドライブ、およびファイバ・チャネルなどの通信媒体を介して多数のリモート・コンピュータからアクセスできるマルチ・ポート・ディスク・ドライブを提供することが一般的であったが、図１に示されるディスク・ドライブなどの個々のディスク・ドライブは通常、単一のリモート・コンピュータに接続され、かつそれに使用される。しかし単一ディスク・ドライブ内に格納できる電子データ量は限られている。多数のリモート・コンピュータから効率的にアクセスできるはるかに大容量の電子データ記憶装置を提供するために、ディスク製造業者は通常、図１に示されるディスク・ドライブなどの多数の異なる個別ディスク・ドライブをディスク・アレイ装置に組合わせ、記憶容量を増加させ、ならびにディスク・アレイ内に含まれる複数のディスク・ドライブの同時動作により機能する並列Ｉ／Ｏ要求に対する容量を増加させる。

【０００６】

図２は、ディスク・アレイの単純なブロック図である。ディスク・アレイ２０２は、いくつかのディスク・ドライブ装置２０３、２０４、２０５を含む。図２では、例示を単純化するために、ディスク・アレイの中に３つの個別ディスク・ドライブしか示されていないが、ディスク・アレイは、数十または数百の個別ディスク・ドライブを含むことができる。ディスク・アレイは、ディスク・アレイ・コントローラ２０６およびキャッシュ・メモリ２０７を含む。一般に、はるかに遅い電子機械的ディスク・ドライブからではなく迅速にアクセス可能なキャッシュ・メモリからデータを読取ることにより、同じデータに対するその後の要求をより迅速に満足させることができるように、ＲＥＡＤ要求に応答してディスク・ドライブから取り出されたデータを、キャッシュ・メモリ２０７内に格納することができる。その後合理的な時間内に再要求される可能性が非常に大きいデータをキャッシュ・メモリ２０７内に維持するために、様々な精巧な機構が用いられる。データがその後ＲＥＡＤ要求を介して要求される可能性がある場合、または物理記憶媒体への遅いデータ書込みを先送りするために、ディスク・アレイ・コントローラ２０６は、ＷＲＩＴＥ要求を介してリモート・コンピュータから受け取ったデータをキャッシュ・メモリ２０７内に格納することもまた選択することができる。

【０００７】

電子データは、ディスク・アレイ内で具体的なアドレス可能な位置に格納される。ディスク・アレイは、多数の異なる個別ディスク・ドライブを含む可能性があるので、ディスク・アレイにより表されるアドレス・スペースは巨大であり、一般に数千ギガバイトである。全アドレス・スペースは通常、論理ユニット（「LUN」）と呼ばれるいくつかの抽象的データ記憶リソースに分割される。ＬＵＮは、ディスク・アレイ内の１つまたは複数のディスク・ドライブのデータ記憶スペースにマッピングされた、所定量の電子データ記憶スペースを含み、アクセス特権、バックアップ頻度、および１つまたは複数のＬＵＮとのミラーリング調整を含む様々な論理パラメータに関連付けることができる。ＬＵＮは、ランダム・アクセス・メモリ（「ＲＡＭ」）、ハード・ディスク以外の大容量記憶装置、あるいはメモリ、ハード・ディスク、および／または他のタイプの大容量記憶装置の組合わせに基づくこともまたできる。リモート・コンピュータは一般に、内部ディスク・ドライブ２０３～２０５およびディスク・アレイ・コントローラ２０６を介してディスク・アレイにより提供される多数の抽象的ＬＵＮ２０８～２１５の１つを介してディスク・アレイ内のデータにアクセスする。したがってリモート・コンピュータは、ディスク・アレイに対応するバス通信媒体アドレスを用いて、バイト、ワード、ブロックなど特定の単位データ

量、通常は６４ビットの整数であるＬＵＮ識別子およびＬＵＮを指定する３２ビット、６４ビット、または１２８ビットのデータ・アドレス、ならびにＬＵＮに割り振られた論理データ・アドレス・パーティション内のデータ・アドレスを指定することができる。ディスク・アレイ・コントローラは、そのようなデータの指定をディスク・アレイ内の特定のディスク・ドライブおよびディスク・ドライブ内の論理データ・アドレスの標示に変換する。ディスク・ドライブ内のディスク・ドライブ・コントローラは最後に、論理アドレスを物理媒体アドレスに変換する。通常、電子データは、連続した３２ビットまたは６４ビットのコンピュータ・ワードの１つまたは複数のブロックとして読取り／書込みされ、アクセスの細分性(granularity)の正確な詳細は、ディスク・アレイおよび個別ディスク・ドライブ内のハードウェアおよびファームウェア機能ならびにＩ／Ｏ要求を生成するリモート・コンピュータのオペレーティング・システムおよびディスク・アレイとリモート・コンピュータを相互接続する通信媒体の特性に依存する。

【０００８】

ディスク・アレイ内のディスク・アレイ・コントローラは、ディスク・アレイ・コントローラ内に実施されたインタフェースを介してリモート・コンピュータと対話する。このインタフェースは、コンピュータ・ネットワーキング・システムの高級プロトコルと、あるいはオペレーティング・システムからアプリケーション・プログラムおよび人間のユーザに提供される関数呼出しインタフェースと似ている。ディスク・アレイ・コントローラにより提供されるインタフェースは、通信媒体を介してディスク・アレイにアクセスするリモート・コンピュータから要求することができる１組の動作を論理的に含む。動作を要求するためにリモート・コンピュータは、リモート・コンピュータの通信媒体アドレス、リモート・コンピュータから要求されている動作タイプの標示、データ・アドレスなど要求された動作に特有のパラメータ、およびディスク・アレイ・コントローラがその動作を実行すべきターゲットＬＵＮを通常含むいくつかの異なるタイプの情報を提供する。したがって例えば、ディスク書込み動作を実行するために、リモート・コンピュータは、書込むデータの位置、データが書込まれるアドレス、およびデータを書込むべきアドレスを含むアドレス・スペースを提供するＬＵＮの識別子を指定する必要がある。

【０００９】

しかし、リモート・コンピュータがディスク・アレイ・コントローラに要求できる動作にはいくつかの異なるタイプがあり、非ＬＵＮベースの動作でも複数のＬＵＮにまたがる動作でもある。一例として、リモート・コンピュータは、ディスク・アレイ・コントローラが第１ＬＵＮを第２ＬＵＮにミラーリングするように要求することができる。第１ＬＵＮが第２ＬＵＮにミラーリングされるとき、第２ＬＵＮが第１ＬＵＮの忠実なミラー・コピーとなるように、ディスク・アレイ・コントローラは、第１ＬＵＮを対象とするすべての書込みを第１および第２ＬＵＮの両方に自動的に実行する。したがってリモート・コンピュータは、ディスク・アレイ・コントローラが第１ＬＵＮを第２ＬＵＮにミラーリングするように要求するために、ミラーリング動作の実行を要求する際に両方のＬＵＮを指定する必要がある。最初のミラーリング接続が確立された後で、リモート・コンピュータは、第１のプライマリＬＵＮに単純に書込み、データがセカンダリのミラーＬＵＮに内部的にコピーされるものと保証される。他の例として、リモート・コンピュータは、１組のＬＵＮを指定された時間間隔で指定されたバックアップ装置に自動的にバックアップするように、ディスク・アレイ・コントローラに命じることを望むことができる。

【００１０】

ディスク・アレイ・コントローラから要求元リモート・コンピュータに提供されるいくつかの動作が、複数のＬＵＮを含むことがあるという事実と、一般に、現在の多数のディスク・アレイ・コントローラ・インタフェースを介して任意の特定の動作を呼び出す際に、リモート・コンピュータが単一のターゲットＬＵＮを指定しなければならないという事実を調和させるために、制御装置ＬＵＮ（「ＣＤＬＵＮ」）として知られているある種の仮想ＬＵＮが、リモート・コンピュータが動作を呼び出すインタフェースの一部としてディスク・アレイ・コントローラにより提供される。ＣＤＬＵＮは本質的には、ディスク・ア

レイ・コントローラにより提供され実行される様々な動作に対するアクセスのポイントである。したがって第１ＬＵＮが第２ＬＵＮにミラーリングされるべきであると指定するために、リモート・コンピュータは、ミラーリング動作を呼び出し、動作のターゲットとして特定のＣＤＬＵＮを指定する。ＣＤＬＵＮは、アレイ内のＬＵＮ対の制御動作に対する間接メモリ・マッピングされたアクセスを提供する。ＣＤＬＵＮ内の具体的論理アドレス・オフセットを対象とする制御動作は、定義により、そのオフセットに関連付けられたアレイ内のＬＵＮを対象とする。

【００１１】

ディスク・アレイ・コントローラはさらに、リモート・コンピュータに提供されるインタフェースの一部として、特定のリモート・コンピュータまたはリモート・コンピュータのグループが、１つまたは複数のＬＵＮに対する独占的アクセスを獲得かつ維持できるようにする様々なセキュリティ機構を提供することができる。そうすることにより、リモート・コンピュータまたはリモート・コンピュータのグループは、許可されないエンティティによるアクセスおよび破損の可能性からプライベート・データを保護することができる。ディスク・アレイは最初、単一の組織内の使用のために開発され、ディスク・アレイ内に格納されたデータに対するセキュリティは、コンピュータ・ルーム内のディスク・アレイの物理的分離および組織内の信用あるコンピュータにしかディスク・アレイを接続しないことにより一般に得られていた。しかしディスク・アレイのサイズおよび複雑性が増し、ファイバ・チャネルなどの通信媒体の機能が非常に向上したことに伴い、ファイバ・チャネルなどの通信媒体を介してディスク・アレイ内に格納されたデータにアクセスするいくつかの異なる遠隔組織間でディスク・アレイが分割されることがますます一般的になっている。

【００１２】

第１レベルのセキュリティとして、ディスク・アレイは通常、ディスク・アレイ・コントローラにより管理される中央に格納されたアクセス・テーブルを介してＬＵＮに対するアクセスを分割する。アクセス・テーブルは通常、それぞれがＬＵＮ標示、ポート標示、およびリモート・コンピュータの一意な識別子を含むエントリを格納する。ディスク・アレイ・コントローラが、特定の通信ポートを介して指定されたターゲットＬＵＮを有する動作を実行する要求をリモート・コンピュータから受け取るとき、ディスク・アレイ・コントローラは、ターゲットＬＵＮ、要求を受け取ったポート、要求を受け取ったリモート・コンピュータの一意な識別子に一致する識別子の標示を含むエントリを求めてアクセス・テーブル内を見る。そのようなエントリが見つかれば、ディスク・アレイ・コントローラはその動作が進行するのを許可する。他方、そのようなエントリが見つからなければ、ディスク・アレイ・コントローラは、そのような記憶ＬＵＮが存在しないという標示を要求元リモート・コンピュータに返す。アクセス・テーブルは通常、システム管理者またはネットワーク管理者によるコンソール・インタフェースを介したエントリで取り込まれる。したがってシステム管理者またはネットワーク管理者は、動作の実行を求めて入って来るそれぞれの要求をディスク・アレイ・コントローラによりチェックして適切に許可するために、ディスク・アレイ・コントローラにより使用されるディスク・アレイ内のアクセス・テーブルをセットアップしメンテナンスすることにより、リモート・コンピュータ間のＬＵＮに対するアクセスを分割する。

【００１３】

残念ながら、複数のＬＵＮにまたがり、上述のようにターゲットＣＤＬＵＮの指定を必要とする動作クラスは現在、上述のアクセス・テーブル法により充分に安全にはされない。現在、ディスク・アレイ・コントローラは、アクセス・テーブル内のエントリが、要求元リモート・コンピュータの一意な識別子、ＣＤＬＵＮ標示、および要求を受け取ったポート標示を含むかどうかをチェックすることにより、ターゲットＣＤＬＵＮを指定する要求された動作をチェックする。しかしディスク・アレイ・コントローラはその後、要求元リモート・コンピュータが、その動作の実行を求める要求の一部として指定された他のＬＵＮのいずれかにアクセスすることを許可されているかどうかをチェックしない。例えばリ

モート・コンピュータが特定のターゲットＣＤＬＵＮを介して第１ＬＵＮが第２ＬＵＮにミラーリングされるように要求する場合、ディスク・アレイ・コントローラは、リモート・コンピュータがそのターゲットＣＤＬＵＮにアクセスすることを許可されているかどうかを見るためにチェックするだけであり、その後リモート・コンピュータが、さらに指定された第１および第２ＬＵＮにアクセスすることを許可されているかどうかを見るためにはチェックしない。

【００１４】

ＣＤＬＵＮを介して間接的にアクセスされ、ターゲットＣＤＬＵＮに対する動作の実行を求める要求の一部としてさらに指定されるＬＵＮに対する、ディスク・アレイ・コントローラによる許可チェックがないことは、ディスク・アレイ大容量記憶装置内の、ディスク・アレイのデータを格納し取り出すリモート・コンピュータに対するかなり大きなセキュリティ侵害の可能性を意味する。第１の組織に属するリモート・コンピュータが、第１の組織により要求されたミラーリング動作の一部として、第２の組織に属するＬＵＮを誤ってまたは悪意をもって指定する可能性がある。同様に、第１の組織のリモート・コンピュータが、自動バックアップを求める要求の一部として指定された１組のＬＵＮに、第２の組織に属するＬＵＮを誤ってまたは悪意をもって組込む可能性がある。そのような場合、第２の組織に属するディスク・アレイ内に格納されたデータの破損または第２の組織に属するデータの不正コピーのいずれかをもたらす動作を求める要求を、第１の組織が誤ってまたは悪意をもってディスク・アレイに命ずる可能性がある。

【００１５】

【発明が解決しようとする課題】

ディスク・アレイ・コントローラは通常ファームウェアと共に論理回路内に実施される。これらの実施例は、ソフトウェア・プログラムの実施例のようには容易に変更されない。さらに、ハードウェアおよび内部メモリ制約のため、汎用オペレーティング・システムおよびネットワーク・プロトコル内に通常ある精巧なセキュリティ方法論およびプロトコルは法外に高価であり、ファームウェア・ディスク・アレイ・コントローラの実施例の一部として実施するには困難であろう。これらの理由により、ディスク・アレイの設計者、製造者、およびユーザは、ターゲットＣＤＬＵＮに対して実行される動作を介して許可されないリモート・コンピュータがＬＵＮにアクセスすることを防止する比較的容易に実施される他のセキュリティ機構に対する必要性を認識して来た。

【００１６】

【課題を解決するための手段】

本発明の一実施形態においてディスク・アレイ・コントローラは、他のＬＵＮの指定を含むターゲットＣＤＬＵＮを対象とする、リモート・コンピュータから要求された動作の許可をチェックするために、２つのアクセス・テーブルを使用する。最初にディスク・アレイ・コントローラは、その要求の指定されたターゲットＣＤＬＵＮ、その要求を受け取ったポート、その要求を受け取ったリモート・コンピュータの一意な識別子、に等しいＬＵＮ、ポート、およびリモート・コンピュータ識別子の標示、を有するエントリが第１アクセス・テーブル内にあるかどうかを判定する。第１アクセス・テーブル内にそのようなエントリが存在するとき、ディスク・アレイ・コントローラは、要求元リモート・コンピュータがそのターゲットＣＤＬＵＮにアクセスすることを許可されていると仮定する。次にディスク・アレイ・コントローラは、第２の、追加アクセス・テーブルをチェックして、その動作の実行を求める要求の一部として指定された他のそれぞれのＬＵＮに対してその動作のために指定されたターゲットＣＤＬＵＮの標示と対になった他のＬＵＮの標示を含むエントリが存在するかどうかを判定する。ディスク・アレイ・コントローラが、その動作の実行を求める要求内で指定された他の各ＬＵＮのためのそのようなエントリを追加アクセス・テーブル内に見つけるときに限って、ディスク・アレイ・コントローラはその動作の実行を許可する。

【００１７】

一般的な読取りおよび書込み動作など、多数の非ＣＤＬＵＮ介在動作にとって、ディスク

・アレイ・コントローラによる許可チェックは変わらない。そのような動作が進行するためには、ディスク・アレイ・コントローラがアクセス・テーブル内に対応するエントリを見つけなければならない。ディスク・アレイ・コントローラは、ＬＵＮミラーリングを求める要求など、ターゲットＣＤＬＵＮを指定する要求であって、他のＬＵＮの指定を含む動作を求める要求のためだけに、２つのレベルのアクセス・テーブルおよび追加アクセス・テーブル許可チェックを用いる。

【００１８】

したがって、ターゲットＣＤＬＵＮを指定し他のＬＵＮの指定を含む動作を正常に要求するために、要求元リモート・コンピュータは、指定されたターゲットＣＤＬＵＮにアクセスすることを許可されなければならず、ターゲットＣＤＬＵＮは、さらに指定されたそれぞれのＬＵＮにアクセスすることを許可されなければならない。アクセス・テーブルおよび追加アクセス・テーブルの両方は、コンソールを介して、または安全な相互接続を介してリモートでディスク・アレイ・コントローラと対話するシステム管理者またはネットワーク管理者により取り込まれ、編成、維持される。ターゲットＣＤＬＵＮではなくターゲットＬＵＮを指定する多数の動作の許可チェックが変わらないので、かつ追加アクセス・テーブルおよび追加アクセス・テーブルを用いる許可チェックの実施が、アクセス・テーブルおよびアクセス・テーブルを用いる許可チェックの現在の実施とほとんど同一であるので、本発明に述べた実施形態は、ディスク・アレイ・コントローラのファームウェア実施例の一部として、経済的かつ比較的容易に実施される。

【００１９】

【発明の実施の形態】

本発明は、要求された動作を求める要求元リモート・コンピュータの許可をチェックする、ディスク・アレイ・コントローラが引受けるセキュリティ対策に関する。前述のように、ディスク・アレイ・コントローラの現在の特定のファームウェア実施例において、ディスク・アレイ・コントローラは、要求の一部として指定されたターゲットＬＵＮの標示、要求を受け取ったポートの標示、および要求元リモート・コンピュータの一意な識別子を含むエントリを求めてアクセス・テーブルを検索する。アクセス・テーブル内にそのようなエントリが見つかると、ディスク・アレイ・コントローラは、要求された動作の実行を続けるが、そのようなエントリが見つからないとき、ディスク・アレイ・コントローラは、そのような記憶ＬＵＮが存在しないという標示を要求元リモート・コンピュータに返す。前記議論のように、この許可方法は、ディスク・アレイ・コントローラにより提供される動作クラスであって、指定されたターゲットＣＤＬＵＮを対象とし他の指定されたＬＵＮを含む動作クラスには不充分である。前記議論のように、第１ＬＵＮを第２ＬＵＮに、または第１グループの諸ＬＵＮを第２グループの諸ＬＵＮにミラーリングすることを要求する動作、あるいは１グループのＬＵＮを指定されたバックアップ装置に自動的にバックアップすることを要求する動作は、ターゲットＣＤＬＵＮならびに他のＬＵＮの指定を含む動作の例である。現在このクラスの動作は、要求元リモート・コンピュータが追加指定されたＬＵＮにアクセスすることを許可されているかどうかに関わらず、要求元リモート・コンピュータがターゲットＣＤＬＵＮにアクセスすることを許可されているときにディスク・アレイ・コントローラにより許可される。誤ったまたは悪意のあるリモート・コンピュータがアクセス特権を有するターゲットＣＤＬＵＮを介して、その誤ったまたは悪意のあるリモート・コンピュータがアクセス特権を有しないＬＵＮに、その誤ったまたは悪意のあるリモート・コンピュータが容易にアクセスを得ることができる。

【００２０】

現在のディスク・アレイ・コントローラ実施例内のこの潜在的に大きなセキュリティ・ホールを閉じるために、本発明の一実施形態は、他のＬＵＮとともにターゲットＣＤＬＵＮを指定することにより要求されたそれらの動作のために第２の、追加アクセス・テーブルを用いる。追加アクセス・テーブルは、それぞれがＣＤＬＵＮの標示および非ＣＤＬＵＮの標示を含むエントリを有する。追加アクセス・テーブル内のＣＤＬＵＮ／ＬＵＮ対の存在は、そのＣＤＬＵＮがＬＵＮにアクセスすることができることを示す。リモート・コン

ピュータが、他のＬＵＮとともにターゲットＣＤＬＵＮを指定することにより動作を要求することを許可されるために、リモート・コンピュータは、アクセス・テーブル内のエントリを介してターゲットＣＤＬＵＮにアクセスすることを許可されなければならず、ターゲットＣＤＬＵＮは、追加アクセス・テーブル内のエントリを介してその要求中に指定された他のそれぞれのＬＵＮにアクセスすることを許可されなければならない。

【００２１】

以下に詳細に述べるように、本発明に述べた実施形態は、ディスク・アレイ・コントローラのファームウェア実施例内に経済的かつ比較的容易に実施される。最初に、ディスク・アレイ・コントローラにより提供される動作の大半である、ターゲットＣＤＬＵＮではなくてターゲットＬＵＮを含む動作に対する許可チェック方法は変わらない。許可チェックのための現在の方法に対する変更は、指定されたターゲットＣＤＬＵＮを対象とし追加指定されたＬＵＮを含むクラスの動作にしか関連しない。そのクラスの動作に対して、２つの部分からなる許可、すなわち、現在の許可方法と同一の第１部分、および現在の許可方法に非常に類似の第２部分の実施、が使用される。

【００２２】

本発明の一実施形態を、Ｃ＋＋類似の疑似コードの実施例を参照しながら以下に詳細に述べる。Ｃ＋＋類似の疑似コードの実施例は、明確化および例示の目的で選ばれている。一般にディスク・アレイ・コントローラは、ファームウェア内、または論理回路内に直接実施される。ファームウェア実施例ならびに論理回路実施例は、現代の技術により、高級プログラム類似の仕様のコンパイルによって設計、実施されるので、疑似コードによって本発明の実施例を示すことは、極めて適切なことである。

【００２３】

次のＣ＋＋類似の疑似コード実施例は、特定のディスク・アレイ・コントローラ内で使用される現在のセキュリティ機構ならびに本発明の一実施形態を示す。最初に、いくつかの定数およびＬＵＮ、ポート、およびサーバのための識別子を表す３つのクラス宣言が以下に提供され、この疑似コード実施例中のサーバは、より一般的なリモート・コンピュータの概念と等価である。

【００２４】

```
1  const int WWW_name_length = 20;
2  const int TableLength = 1000;

3  class LUN
4  {
5  private:
6        int  name;
7  public:
8        int  getVal()                  {return name;};
9        LUN& operator=(int l)          {name = l; return *this;};
10       LUN& operator=(LUN& l)             {name = l.getVal(); return *this;};
11       bool     operator==(LUN& l)    {return (l.getVal() == name);};
12       bool     operator<(LUN& l)     {return (name < l.getVal());};
13       bool     operator>(LUN& l)     {return (name > l.getVal());};
14       LUN(LUN& l)                    {name = l.getVal();};
15       LUN (int l)                    {name = l;};
16       LUN();
17 };

18 class port
19 {
20 private:
21       int  name;
22 public:
23       int  getVal()                  {return name;};
24       port&    operator=(int p)      {name = p; return *this;};
```

```
25      port&   operator=(port& p)    {name = p.getVal(); return *this;};
26      bool    operator==(port& p)  {return (p.getVal() == name);};
27      bool    operator<(port& p)   {return (name < p.getVal());};
28      bool    operator>(port& p)   {return (name > p.getVal());};
29      port(port& p)                {name = p.getVal();};
30      port (int p)                 {name = p;};
31      port();
32  };

33  class server
34  {
35  private:
36      char        name[WWW_name_length];
37  public:
38      char*       getVal() {return name;};
39      server&     operator=(char* s);
40      server&     operator=(server& s);
41      bool        operator==(server& s);
42      bool        operator<(server& s);
43      bool        operator>(server& s);
44      server(server& s);
45      server (char* s);
46      server();
47  };
```

【００２５】

上記行１上で宣言された定数「WWW_name_length」は、サーバまたはリモート・コンピュータのための文字列識別子のために任意に定義された長さである。上記行２上で宣言された定数「TableLength」は、アクセス・テーブルの任意の最大長である。実際の実施例において、これらの定数に任意に割り当てられた値は、全く異なるか、あるいはサーバ名およびアクセス・テーブルの長さのいずれかまたは両方が可変の長さを有する可能性があることに留意されたい。

【００２６】

上記行３～１７上で宣言されたクラス「LUN」は、特定のＬＵＮの標示または識別子を表す。疑似コード実施例においてＬＵＮ識別子は、基本的に整数であり、クラス「LUN」は対応して、上記行６上で宣言された整数データ・メンバ「name」を含む。上記行８上で宣言されたメンバ関数「getVal」は、データ・メンバ「name」の整数値を返す。上記行９～１３上で宣言された他の関数メンバは、１つのＬＵＮが他のＬＵＮに割り当てられること、ＬＵＮに整数値が割り当てられること、および１つのＬＵＮが他のＬＵＮと比較されることを可能にする、割り当ておよび関係演算子を提供する。最後にクラス「LUN」は、ＬＵＮが異なるＬＵＮの値を有するように構築されること、コンストラクタ（ｃｏｎｓｔｒｕｃｔｏｒ）に対する引数として供給された整数値を有することを可能にし、あるいはＬＵＮが、データ・メンバ「name」の整数値を指定することなく構築されることを可能にする、上記行１４～１６上で宣言された３つのコンストラクタを含む。上記行１８～３２および３３～４７上で宣言されたクラス「port」および「server」はそれぞれ、同様にポート識別子およびサーバまたはリモート・コンピュータ識別子をそれぞれ表す。ポートはＬＵＮ同様、整数により識別され、サーバまたはリモート・コンピュータは、長さ「WWW_name_length」の文字列により識別される。

【００２７】

次にアクセス・テーブルを定義する２つのクラス宣言が提供される。

【００２８】

```
1  class AccessEntry
2  {
3  private:
4       LUN ln;
5       port pt;
6       server sv;
7  public:
8       LUN&        getLUN()              {return ln;};
9       void        setLUN(LUN& l)        {ln = l;};
10      port&       getPort()             {return pt;};
11      void        setPort(port& p)      {pt = p;};
12      server& getServer()               {return sv;};
13      void        setServer(server& s)  {sv = s;};
14      AccessEntry& operator=(AccessEntry& ae);
15      bool        operator==(AccessEntry& ae);
16      bool        operator<(AccessEntry& ae);
17      bool        operator>(AccessEntry& ae);
18      AccessEntry(LUN& l, port& p, server& s);
19      AccessEntry();
20 };

21 class AccessTable
22 {
23 private:
24      AccessEntry table[TableLength];
25      int size;
26 public:
27      int   addEntry(LUN& l, port& p, server& s);
28      int   deleteEntry(LUN& l, port& p, server& s);
29      int   findEntry(LUN& l, port& p, server& s);
30      bool      retrieveEntry(int index, LUN& l, port& p, server& s);
31      AccessTable();
32 };
```

【００２９】

上記行１～２０上で宣言されたクラス「AccessEntry」は、アクセス・テーブル内の単一エントリを表す。上記議論のように、アクセス・テーブル内の各エントリは、特定のＬＵＮ、ポート、およびサーバまたはリモート・コンピュータの標示を含む。それらの標示は、クラス「AccessEntry」において、上記行４～６上で宣言された３つのデータ・メンバ「In」、「pt」および「sv」により表される。クラス「AccessEntry」は、上記行８～１３上で宣言された３つのデータ・メンバ内の値の取り出しおよび格納のためのメンバ関数、上記行１４および行１５～１７上でそれぞれ宣言された割り当て演算子および様々な関係演算子、ならびに上記行１８～１９上で宣言されたいくつかのコンストラクタを含む。Ｃ＋＋の規則にしたがい、データ・メンバの値を取り出す関数は、プレフィクス「get」を含む名前を有し、データ・メンバに値を格納する関数は、プレフィクス「set」を含む名前を有することに留意されたい。上記行１４上で宣言された割り当て演算子は、クラス「AccessEntry」の１つのインスタンスに、クラス「AccessEntry」の他のインスタンスの値を割り当てることを可能にする。上記行１５～１７上で宣言された関係演算子は、クラス「AccessEntry」の１つのインスタンスにより表される値と、クラス「AccessEntry」の他のインスタンス内に格納された値を比較することを可能にする。

【００３０】

上記行２１～３２上で宣言されたクラス「AccessTable」は、特定のサーバによる特定のポートを介した特定のＬＵＮおよびＣＤＬＵＮに対するアクセスの許可を格納するために、ディスク・アレイ・コントローラの実施例により使用されるアクセス・テーブルを表す。クラス「AccessTable」のインスタンスは、クラス「AccessEntry」のインスタンスのアレイを含み、本質的にＬＵＮ／ポート／サーバの三つ組みを表す。クラス「AccessEntry」のインスタンスのアレイは、「table」と呼ばれるが、上記行４上で宣言され、行５上では、クラス「AccessTable」のインスタンスにより表されるアクセス・テーブル内に現在格納される有効なエントリの数を含む整数データ・メンバ「size」が宣言される。現在の実施例において、クラス「AccessEntry」のインスタンスは、インデックス「０」を有するテーブル内の第１スロットで始まるテーブル内に順次格納される。したがって２つの

有効なエントリを有するテーブルは、インデックス０および１を有するテーブル・スロット内に、クラス「AccessEntry」のインスタンスを含み、データ・メンバ「size」は値２を有するであろう。したがってsizeは、テーブル内の有効なエントリの数、およびＬＵＮ／ポート／サーバの三つ組みを割り当てることのできるテーブル内の次のフリー・スロットのインデックスの両方を表す。クラス「AccessTable」は、次のメンバ関数を含む。（１）「addEntry」、ＬＵＮ／ポート／サーバの三つ組みをクラス「AccessTable」のインスタンスにより表されるアクセス・テーブルに追加する、上記行２７上で宣言されたメンバ関数、（２）「deleteEntry」、アクセス・テーブルから特定のＬＵＮ／ポート／サーバの三つ組みを削除する、行２８上で宣言されたメンバ関数、（３）「findEntry」、アクセス・テーブル内に格納された特定のＬＵＮ／ポート／サーバの三つ組みのインデックスを返す、行２９上で宣言されたメンバ関数、（４）「retrieveEntry」、参照引数「l」、「p」および「s」として供給されたクラス「LUN」、「port」および「server」のインスタンス内に、引数「index」内に供給されたインデックスを有するテーブル・エントリからＬＵＮ／ポート／サーバの三つ組みを取り出す、行３０上で宣言されたメンバ関数、ならびに（５）上記行３１上で宣言されたコンストラクタ。

【００３１】

次に、クラス「LUN」、「port」および「server」の様々なメンバ関数の実施例が提供される。これらの実施例は、多くの場合最適に効率的とはほど遠いが、非常に単純明快である。制御論理およびプログラム・コードを開発する際には、実行時効率と設計および実施効率、実行時効率とコードの複雑性、ならびに実行時効率対メンテナンスおよびその後の改善の経済性の間の多数のトレードオフが通常存在する。他の多数のトレードオフおよび対応する実施例を選択することができ、それらは本発明の範囲内ではあるが、例示の目的で、実行時の効率よりも単純さを支持し、必ずしも最適に効率的なアルゴリズムではなく単純なアルゴリズムを選択した。このセクションおよび次のセクション内の多数のメンバ関数の実施例は、単純明快であり、詳細には議論しない。

【００３２】

```
1  LUN::LUN()
2  {
3  }

4  port::port()
5
6  }

7  server::server()
8  {
9  }
```

```
10 server& server::operator=(char* s)
11 {
12     char* k = name;
13     for (int i = 0; i < WWW_name_length; i++) *k++ = *s++;
14     return *this;
15 }

16 server& server::operator=(server& s)
17 {
18     char* s1 = s.getVal();
19     char* k = name;
20     for (int i = 0; i < WWW_name_length; i++) *k++ = *s1++;
21     return *this;
22 }

23 bool server::operator==(server& s)
24 {
25     char* s1 = s.getVal();
26     char* k = name;
27     for (int i = 0; i < WWW_name_length; i++)
28     {
29         if (*k++ != *s1++) return false;
30     }
31     return true;
32 }

33 bool server::operator<(server& s)
34 {
35     char* s1 = s.getVal();
36     char* k = name;
37     for (int i = 0; i < WWW_name_length; i++)
38     {
39         if (*k < *s1) return true;
40         else if (*k++ > *s1++) return false;
41     }
42     return false;
43 }

44 bool server::operator>(server& s)
45 {
46     char* s1 = s.getVal();
47     char* k = name;
48     for (int i = 0; i < WWW_name_length; i++)
49     {
50         if (*k > *s1) return true;
51         else if (*k++ < *s1++) return false;
52     }
53     return false;
54 }

55 server::server(server& s)
56 {

57     char* s1 = s.getVal();
58     char* k = name;
59     for (int i = 0; i < WWW_name_length; i++) *k++ = *s1++;
60 }

61 server::server (char* s)
62 {
63     char* k = name;
64     for (int i = 0; i < WWW_name_length; i++) *k++ = *s++;
65 }
```

【００３３】

上記行１０～１５上に示されているサーバ割り当て演算子の実施例は、割り当て演算子の実施の典型となる。この場合、データ・メンバ「name」内に格納された値は、この割り当て関数に引数「s」として渡される文字列により表される値を有するように割り当てられる。行１３のforループにおいて引数「s」により参照される文字列は、ローカル変数「k」により参照されるデータ・メンバ「name」にコピーされる。最後に、行１４上でこの割り当て関数は、クラス「server」の現在のインスタンスに参照を返す。

【００３４】
上記行３３～４３上に示されるサーバの「<」関係演算子の実施例は、関係演算子の実施の典型となる。この演算子は、クラス「server」の現在のインスタンスの値と参照引数「s」により参照される異なるサーバ・インスタンスの値を比較する。クラス「server」の現在のインスタンスの値が、引数「s」により参照されるサーバ・インスタンスの値より小さい場合、「<」演算子は、真のブール値を返し、そうでない場合、偽のブール値を返す。行３７～４１のforループにおいて、現在のインスタンスのデータ・メンバ「name」の値が、引数「s」により参照されるサーバ・インスタンスの値と比較される。実施された比較は、見慣れた電話帳内の名前の順序付けを生成する古典的な辞書的文字列比較と等価である。

【００３５】
次に、クラス「AccessEntry」の様々なメンバ関数の実施例が提供される。

【００３６】

```
1  AccessEntry& AccessEntry::operator=(AccessEntry& ae)
2  {
3      ln = ae.getLUN();
4      pt = ae.getPort();
5      sv = ae.getServer();
```

```
6       return *this;
7   }

8   bool AccessEntry::operator==(AccessEntry& ae)
9   {
10      return (ae.getLUN() == ln &&
11              ae.getPort() == pt &&
12              ae.getServer() == sv);
13  }

14  bool AccessEntry::operator<(AccessEntry& ae)
15  {
16      if (ln < ae.getLUN()) return true;
17      else if (ln == ae.getLUN())
18      {
19          if (pt < ae.getPort()) return true;
20          else if (pt == ae.getPort())
21          {
22              if (sv < ae.getServer()) return true;
23              else return false;
24          }
25          else return false;
26      }
27      else return false;
28  }

29  bool AccessEntry::operator>(AccessEntry& ae)
30  {
31      if (ln > ae.getLUN()) return true;
32      else if (ln == ae.getLUN())
33      {
34          if (pt > ae.getPort()) return true;
35          else if (pt == ae.getPort())
36          {
37              if (sv > ae.getServer()) return true;
38              else return false;
39          }
40          else return false;
41      }
42      else return false;
43  }

44  AccessEntry::AccessEntry(LUN& l, port& p, server& s)
45  {
46      ln = l;
47      pt = p;
48      sv = s;
49  }

50  AccessEntry::AccessEntry()
51  {
52  }
```

【００３７】

次に、クラス「AccessTable」の様々なメンバ関数の実施例が以下に提供される。

【００３８】

```
1  int  AccessTable::addEntry(LUN& l, port& p, server& s)
2  {
3       int i = 0;
4       int j;
5       AccessEntry ae(l, p, s);

6       if (size == TableLength) return -1;
7       while (i < size && table[i] < ae) i++;
8       if (i < size && table[i] == ae) return -2;
9       else
10      {
11           for (j = size; j > i; j--) table[j] = table[j-1];
12           table[i] = ae;
13           size++;
14      }
15      return size;
16 }

17 int AccessTable::deleteEntry(LUN& l, port& p, server& s)
18 {
19      int i, j;

20      i = findEntry(l, p, s);
21      if (i >= 0)
22      {
23           j = i+1;
24           while (j < size) table[i++] = table[j++];
25           size--;
26           return size;
27      }
28      return -1;
29 }

30 int AccessTable::findEntry(LUN& l, port& p, server& s)
31 {
32      int i = 0;
33      AccessEntry ae(l, p, s);

34      while (i < size && table[i] < ae) i++;
35      if (i < size && table[i] == ae) return i;
36      else return -1;
37 }

38 bool AccessTable::retrieveEntry(int index, LUN& l, port& p, server& s)
39 {

40      if (index >= 0 && index < size)
41      {
42           l = table[index].getLUN();
43           p = table[index].getPort();
44           s = table[index].getServer();
45      return true;
46 }
47 else return false;
48 }

49 AccessTable::AccessTable()
50 {
51      size = 0;
52 }
```

【００３９】

再び、AccessTableメンバ関数の実施例は単純明快であり、AccessTableメンバ関数「addEntry」の実施例のみを代表例として議論する。メンバ関数「addEntry」は、クラスAccessTableのインスタンスにより表されるアクセス・テーブルに、三つ組みとして追加されるＬＵＮ、ポート、およびサーバを参照する参照引数を受け取る。行５上で、ローカル変数「ae」が、参照引数により指定されたＬＵＮ／ポート／サーバの三つ組みを含むように構築される。行６上でaddEntryにより検出されるように、アクセス・テーブルが一杯であれば、エントリは追加されないで、負の値が返される。行７のwhileループにおいて、AccessEntry関係演算子「<」により定義されたように、addEntryは、ローカル変数「ae」の値以上の値を有するクラス「AccessEntry」のインスタンスを求めてテーブル内の有効なエ

ントリを走査する。whileループの終わりにおいてローカル変数「i」は、ローカル変数「ae」の値以上の有効な最初のエントリのオフセット、またはテーブル内で次に利用可能なエントリのオフセットのいずれかである。行８上でaddEntryにより検出されるように、テーブル内にエントリが存在しローカル変数「ae」により表されるのと同じ三つ組みを表す場合、アクセス・テーブルに第２の等価な三つ組みを追加する点がないので、addEntryは負の値を返す。そうでない場合、行１２上のforループにおいて、addEntryは、追加されるエントリに続くすべてのエントリを下方に１つ分動かして、新しいエントリのためのスペースを作り、行１３上で、新しいエントリを追加する。新しいエントリの追加に続いて、addEntryは、データ・メンバ「size」内の値を増分してアクセス・テーブルの新しいサイズを反映する。

【００４０】

上記に提供された宣言および実施例とともに、現在利用可能な特定のディスク・アレイ・コントローラにおいて用いられる現在の許可技術を示すために、次の関数「currentAuthorization」が提供される。

【００４１】

```
1  bool      currentAuthorization (LUN& CDLUN, port& p, server& s,
2                                   AccessTable& at)
3  {
4       if (at.findEntry(CDLUN, p, s) >= 0)
5            return true;
6       else return false;
7  }
```

【００４２】

関数「currentAuthorization」は、要求された動作、この場合、前に議論したミラーリングまたはバックアップ要求など、他のＬＵＮの指定を含むターゲットＣＤＬＵＮに対する動作、が許可されているかどうかをチェックするために、ディスク・アレイ・コントローラ内のコードにより呼び出すことができる。関数「currentAuthorization」は、ＬＵＮ、ポート、サーバ、およびアクセス・テーブルを参照する参照引数、それぞれ「CDLUN」、「p」、「s」、および「at」を受け取る。関数currentAuthorizationは、行４上で、ＬＵＮ／ポート／サーバの三つ組みCDLUN／p／sを表すエントリをアクセス・テーブルが現在含むかどうかを判定するために、アクセス・テーブル・メンバ関数「findEntry」を呼び出す。そうであれば、関数「currentAuthorization」は行５上で真を返し、そうでない場合、行６上で偽を返す。したがって前に議論したように、現在の許可技術は、ターゲットＣＤＬＵＮ、要求を受け取ったポート、および要求元サーバまたはリモート・コンピュータの一意な識別子が許可テーブル内に三つ組みとして存在するかどうかをチェックすることを含み、そうであれば、要求された動作が許可されていると見なす。しかし上述のように、許可された動作は、要求元サーバがアクセスする許可を有しないＬＵＮにアクセスすることを含みデータ破損の可能性につながるか、または要求元サーバの組織の外部組織に属するデータを読取る可能性があるのでこの許可技術は非常に不充分である。

【００４３】

本発明の一実施形態は、以下に提供される２つの他のクラスとともに、ＬＵＮ、ポート、サーバ、accessEntry、およびaccessTableクラスの上記宣言および実施例を用いる。

【００４４】

```
1  class SupplementalAccessEntry
2  {
3  private:
4       LUN ln;
5       LUN cd;
6  public:
7       LUN&    getLUN()              {return ln;};
8       void    setLUN(LUN& l)        {ln = l;};
9       LUN&    getCDLUN()            {return cd;};
10      void    setCDLUN(LUN& l)      {cd = l;};
11      SupplementalAccessEntry& operator=(SupplementalAccessEntry& ae);
12      bool    operator==(SupplementalAccessEntry& ae);
13      bool    operator<(SupplementalAccessEntry& ae);
14      bool    operator>(SupplementalAccessEntry& ae);
15      SupplementalAccessEntry(LUN& l, LUN& c);
16      SupplementalAccessEntry();
17 };

18 class SupplementalAccessTable
19 {
20 private:
21      SupplementalAccessEntry table[TableLength];
22      int size;
23 public:
24      int     addEntry(LUN& l, LUN& c);
25      int     deleteEntry(LUN& l, LUN& c);
26      int     findEntry(LUN& l, LUN& c);
27      bool    retrieveEntry(int index, LUN& l, LUN& c);
28      SupplementalAccessTable();
29 };
```

【００４５】

クラス「SupplementalAccessEntry」は、前述のクラス「AccessEntry」に類似し、クラス「SupplementalAccessTable」は、前述のクラス「AccessTable」に類似する。SupplementalAccessEntryは、ＣＤＬＵＮ／ＬＵＮ対を提示し、SupplementalAccessTableは、いくつかのＣＤＬＵＮ／ＬＵＮ対を含む。前記議論のように、特定のＣＤＬＵＮ／ＬＵＮ対の存在は、そのＣＤＬＵＮがターゲットＣＤＬＵＮである動作の一部として、ＣＤＬＵＮがＬＵＮにアクセスできることを示す。この２つの新しいクラスは、以前に宣言されたクラス「AccessEntry」および「AccessTable」と非常に類似しており、したがってさらに述べることはない。SupplementalAccessEntryおよびSupplementalAccessTableメンバ関数の実施例が以下に提供される。

【００４６】

```
1  SupplementalAccessEntry&
```

```
SupplementalAccessEntry::operator=(SupplementalAccessEntry& ae)
{
    ln = ae.getLUN();
    cd = ae.getCDLUN();
    return *this;
}

bool SupplementalAccessEntry::operator==(SupplementalAccessEntry& ae)
{
    return (ln == ae.getLUN() && cd == ae.getCDLUN());
}

bool SupplementalAccessEntry::operator<(SupplementalAccessEntry& ae)
{
    if (ln < ae.getLUN()) return true;
    else if (ln == ae.getLUN())
    {
        if (cd < ae.getCDLUN()) return true;
        else return false;
    }
    else return false;
}

bool SupplementalAccessEntry::operator>(SupplementalAccessEntry& ae)
{
    if (ln > ae.getLUN()) return true;
    else if (ln == ae.getLUN())
    {
        if (cd > ae.getCDLUN()) return true;
        else return false;
    }
    else return false;
}

SupplementalAccessEntry::SupplementalAccessEntry(LUN& l, LUN& c)
{
    ln = l;
    cd = c;
}


SupplementalAccessEntry::SupplementalAccessEntry()
{
}


int  SupplementalAccessTable::addEntry(LUN& l, LUN& c)
{
    int i = 0;
    int j;
    SupplementalAccessEntry ae(l, c);
```

```
45      if (size == TableLength) return -1;
46      while (i < size && table[i] < ae) i++;
47      if (i < size && table[i] == ae) return -2;
48      else
49      {
50          for (j = size; j > i; j--) table[j] = table[j-1];
51          table[i] = ae;
52          size++;
53      }
54      return size;
55  }

56  int SupplementalAccessTable::deleteEntry(LUN& l, LUN& c)
57  {
58      int i, j;

59      i = findEntry(l, c);
60      if (i >= 0)
61      {
62          j = i+1;
63          while (j < size) table[i++] = table[j++];
64          size--;
65          return size;
66      }
67      else return -1;
68  }

69  int SupplementalAccessTable::findEntry(LUN& l, LUN& c)
70  {
71      int i = 0;
72      SupplementalAccessEntry ae(l, c);

73      while (i < size && table[i] < ae) i++;
74      if (i < size && table[i] == ae) return i;
75      else return -1;
76  }

77  bool SupplementalAccessTable::retrieveEntry(int index, LUN& l, LUN& c)
78  {
79      if (index >= 0 && index < size)
80      {
81          l = table[index].getLUN();
82          c = table[index].getCDLUN();
83          return true;
84      }
85      else return false;
86  }

87  SupplementalAccessTable::SupplementalAccessTable()
88  {
89      size = 0;
90  }
```

【００４７】

クラス「SupplementalAccessEntry」および「SupplementalAccessTable」の疑似コード宣言および実施例を考慮して、本発明の一実施形態を表す許可関数「newAuthorization」を次に提供することができる。

【００４８】

```
1  bool     newAuthorization (LUN& CDLUN, LUN* LUNlist, int listSize,
2                             port& p, server& s, AccessTable& at,
3                             SupplementalAccessTable& st)
4  {
5      if (at.findEntry(CDLUN, p, s) >= 0)
6      {
7          while (listSize > 0)
8          {
9              if (st.findEntry(CDLUN, *LUNlist) < 0) return false;
10             listSize--;
11             LUNlist++;
12         }
13         return true;
14     }
15     else return false;
16 }
```

【００４９】
この新しい許可関数は、次の引数を受け取る。（１）「CDLUN」、動作のターゲットＣＤＬＵＮを表すＬＵＮに対する参照、（２）「LUNlist」、ミラーリング動作においてミラーリングされるＬＵＮなど、その動作においてさらに含まれるＬＵＮのリストに対するポインタ、（３）「listSize」、リスト「LUNlist」内のＬＵＮの数を指定する整数、（４）「p」、ディスク・アレイ・コントローラが動作を求める要求を受け取ったポート、（５）「s」、そこから要求を受け取ったサーバまたはリモート・コンピュータ、（６）「at」、アクセス・テーブルに対する参照、および（７）「st」、追加アクセス・テーブルに対する参照。最初に、行５上でnewAuthorizationは、上述の関数「currentAuthorization」において実施された前の行４の許可技術とちょうど同様に、ＣＤＬＵＮ／ｐ／ｓの三つ組みがアクセス・テーブル内に現在存在するかどうかを判定する。存在しない場合、要求元サーバが、参照引数「CDLUN」により指定されたターゲットＣＤＬＵＮに対して、動作を要求する許可を有しないので、newAuthorizationは、行１５上で偽のブール値を返す。そうでない場合、行７～１２のwhileループにおいて、newAuthorizationは、許可を求めてリスト「LUNlist」内の各ＬＵＮをチェックする。行９上でnewAuthorizationは、追加アクセス・テーブル「st」が、LUNlistから選択されたＣＤＬＵＮ／ＬＵＮの対を含むかどうかを判定する。含まない場合、動作のターゲットＣＤＬＵＮは、その動作を求めて指定されたＬＵＮの１つにアクセスすることを許可されておらず、newAuthorizationは、行９上で偽のブール値を返す。リスト「LUNlist」内の各ＬＵＮに対するＣＤＬＵＮ／ＬＵＮ対が、行７～１２のwhileループ内で見つかれば、その動作は許可されており、行１３上でnewAuthorizationは、真のブール値を返す。

【００５０】
したがって本発明の上述の実施形態は、ディスク・アレイ・コントローラのファームウェアの実施例に、第２のアクセス・テーブルである追加アクセス・テーブルを加え、他のＬＵＮの指定を含むターゲットＣＤＬＵＮに対して動作を求めるリモート・コンピュータからの要求に対するより完全な許可チェックを行う能力を、ディスク・アレイ・コントローラに提供する。新しい許可技術は、指定されたターゲットＬＵＮに対して動作を要求する要求元リモート・コンピュータの許可のチェック、ならびに指定されたターゲットＣＤＬＵＮが、その要求の追加指定されたＬＵＮにアクセスすることを許可されていることのチェックを含む。二段階の許可機構を使用することにより、本発明の本明細書に述べた実施形態は、ディスク・アレイ・コントローラ実施例および多数の他のタイプの大容量記憶装置のコントローラの実施例において以前存在した重大なセキュリティ・ホールを閉じる。上述の実施形態では、許可されたエンティティはリモート・コンピュータであるが、代替実施形態では、リモート・プロセスまたはユーザなど他のエンティティが許可されてよい。

【００５１】
本発明を特定の実施形態の見地から述べて来たが、本発明がこの実施形態に限定されることを意図するものではない。当業者には、本発明の趣旨を超えない修正が明らかであろう

。例えば本発明は、ファイバ・チャネルなどの高速通信媒体によりリモートでアクセスされる様々な大容量記憶装置のためのコントローラの実施例において用いることができる。上記のように本発明は、本発明が用いられる大容量記憶装置のコントローラの実施例の性格次第で、ハードウェア回路、ファームウェアまたはソフトウェア内に実施することができる。任意のソフトウェアの実施例のとき、本発明は、ほとんど無制限の数の様々な方法で実施することができる。様々な制御構造およびモジュール構成を用いることができ、様々な基本的動作を有する様々なテーブル・フォーマットを用いることができ、本発明は、任意の数の様々なプログラムまたは仕様言語において実施することができる。本明細書に述べた実施形態は、ディスク・アレイ・コントローラ内で実施されるが、代替実施形態は、大容量記憶装置にアクセスするリモート・コンピュータ上で稼働する許可ルーチン内、許可サーバ内、または他のタイプの装置およびシステム内に実施することができる。

【００５２】

前記記述は説明の目的で、本発明の完全な理解を提供するために、具体的用語を使用した。しかし本発明を実施するために具体的詳細は必要でないことが、当業者には明らかであろう。本発明の具体的実施形態の前記記述は、例示および説明の目的で提示されている。それらの記述は、網羅的であること、または開示された正確な形態に本発明を限定することを意図するものではない。明らかに、上記教示を考慮して多数の修正および変形形態が可能である。諸実施形態は、本発明の原理およびその実際の応用を最もよく説明するために示し述べられ、それによって本発明および意図した特定の使用法に適する様々な修正を含む様々な実施形態を、他の当業者が最もよく利用できるようにする。本発明の範囲は、特許請求の範囲およびそれらの均等物により定義されるものである。

【００５３】

本発明の態様を以下に例示する。

【００５４】

１．大容量記憶装置（２０２）により提供される論理ユニット（２０８～２１５）にリモート・エンティティからアクセスすることを許可する方法であって、特定の論理ユニットにアクセスする、特定のリモート・エンティティの許可をそれぞれ表すエントリを含むアクセス・テーブルを提供すること、特定の論理ユニットにアクセスする、特定の制御装置論理ユニットの許可をそれぞれ表すエントリを含む追加アクセス・テーブルを提供すること、および指定された制御装置論理ユニットを対象として１つまたは複数の他の指定された論理ユニットを含む動作の実行をリモート・エンティティが要求するときに、前記指定された制御装置論理ユニットにアクセスする、前記リモート・エンティティの許可を表すエントリが前記アクセス・テーブル内に現在存在し、前記１つまたは複数の他の指定された論理ユニットのそれぞれに、前記他の指定された論理ユニットにアクセスする、前記指定された制御装置論理ユニットの許可を表すエントリが追加アクセス・テーブル内に存在する場合に限って、前記動作の実行を求める要求を許可することを含む方法。

【００５５】

２．前記大容量記憶装置（２０２）が、リモート・エンティティからの要求を受け取るポートを含み、実行を求める要求の許可が、前記大容量記憶装置内のコントローラ（２０６）により実行される、上記１に記載の方法。

【００５６】

３．論理ユニットまたは制御装置論理ユニットの標示、ポートの標示、およびリモート・エンティティの標示をそれぞれ含むエントリを前記アクセス・テーブルが含む、上記２に記載の方法。

【００５７】

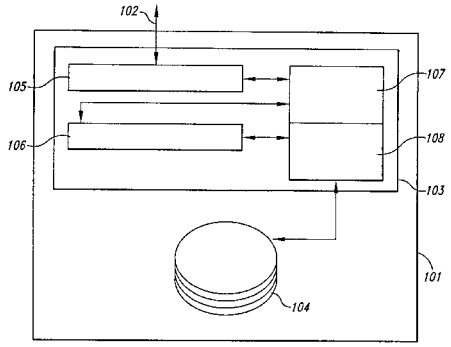
４．制御装置論理ユニットの標示、および論理ユニットの標示をそれぞれ含むエントリを前記追加アクセス・テーブルが含む上記２に記載の方法。

【００５８】

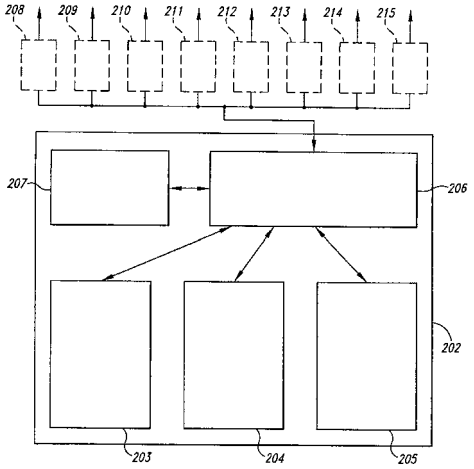
５．前記大容量記憶装置（２０２）がディスク・アレイであり、リモート・エンティティが、通信媒体を介して前記ディスク・アレイと相互接続されたリモート・コンピュータで

ある、上記２に記載の方法。
【００５９】
６．大容量記憶装置（２０２）により提供される論理ユニット（２０８～２１５）にリモート・エンティティからアクセスすることを許可する許可システムであって、リモート・エンティティにより生成される動作の実行を求める要求を検出する要求検出コンポーネント、特定の論理ユニットにアクセスする、特定のリモート・エンティティの許可をそれぞれ表すエントリを含むアクセス・テーブル、特定の論理ユニットにアクセスする、特定の制御装置論理ユニットの許可をそれぞれ表すエントリを含む追加アクセス・テーブル、および前記指定された制御装置論理ユニットにアクセスする、前記リモート・エンティティの許可を表すエントリが前記アクセス・テーブル内に存在し、前記１つまたは複数の他の指定された論理ユニットのそれぞれに、前記他の指定された論理ユニットにアクセスする、前記指定された制御装置論理ユニットの許可を表すエントリが追加アクセス・テーブル内に存在する場合に限り、リモート・エンティティにより要求され、前記要求検出コンポーネントにより検出され、指定された制御装置論理ユニットを対象とし、１つまたは複数の他の指定された論理ユニットを含む要求を許可する制御論理（２０６）を備えるシステム。
【００６０】
７．前記大容量記憶装置（２０２）が、リモート・エンティティからの要求を受け取るポートを含み、前記制御論理（２０６）が前記大容量記憶装置内にある、上記６に記載のシステム。
【００６１】
８．論理ユニットまたは制御装置論理ユニットの標示、ポートの標示、およびリモート・エンティティの標示をそれぞれ含むエントリを前記アクセス・テーブルが含む、上記７に記載のシステム。
【００６２】
９．制御装置論理ユニットの標示、および論理ユニットの標示をそれぞれ含むエントリを前記追加アクセス・テーブルが含む、上記７に記載のシステム。
【００６３】
１０．前記大容量記憶装置（２０２）がディスク・アレイであり、リモート・エンティティが、通信媒体を介して前記ディスク・アレイと相互接続されたリモート・コンピュータである、上記７に記載のシステム。
【図面の簡単な説明】
【図１】標準ディスク・ドライブのブロック図である。
【図２】ディスク・アレイの単純なブロック図である。
【符号の説明】
２０２　ディスク・アレイ
２０３、２０４、２０５　ディスク・ドライブ装置
２０６　ディスク・アレイ・コントローラ
２０７　キャッシュ・メモリ
２０８、２０９、２１０、２１１、２１２、２１３、２１４、２１５　抽象的ＬＵＮ

【図１】


102
105
106
107
108
103
101
104

【図２】


208
209
210
211
212
213
214
215
207
206
202
203
204
205

フロントページの続き

(72)発明者　ロバート・エー・コクラン
アメリカ合衆国９５７６５カリフォルニア州ロックリン、ヴァイン・サークル　１６０８
(72)発明者　グレゴリー・ディー・ドルカス
アメリカ合衆国９５６０２カリフォルニア州オーバーン、リバーウッズ・ドライブ　４１１０

審査官　平井　誠

(56)参考文献　特開平１０－３３３８３９（ＪＰ，Ａ）
特開２００１－２６５６５５（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F　21/24
G06F　 3/06
G06F　12/16