第7回 保護とセキュリティ

コンピュータシステムに対する脅威

- 災害:天災(地震・洪水)、人災(テロ・戦争)
- 故障:停電、ハート・ウェア故障、ソフトウェア故障(バグ等)
- 操作ミス(偶発的な人為災害)
- 故意の人為災害(特に、これが厄介)
 - 不法侵入、盗聴
 - 不法アクセス
 - 不正プログラム、不正トラフィック
- 保護(protection): 誤り(内部的な脅威)への対処
 - 信頼性、可用性などOSの機能が重要な役割
- セキュリティ(security)(注):故意による安全性への脅威への対処
 - 運用方法やアプリケーションを含めた対応が必要
- 注:機密保護とも訳されるが、意味が狭いため、近年は、カタカナで呼ばれる。 用法に注意(「セキュリティ対策」とは言わない)

p.166,167

• 信頼性(reliability):システムが故障せずに動く

- ハードウェア故障、ソフトウェア故障=バグによる誤った動作

安全性に関する特性

- 信頼性向上:ハードウェア+OSを含めたソフトウェア
- 可用性(availability):システムが機能を維持し続ける
 - 冗長系による故障時の切り替え
 - 無停電電源装置(UPS)による停電時の動作維持
- セキュリティ(security):システムの安全を確保する(保安)
 - 侵入、不正使用、情報の窃盗・改ざん、破壊などへの対処
- 完全性(integrity):システムに欠陥が無く、意図どおりに機能する
 - セキュリティの問題を生じる安全性の欠陥:セキュリティホール

保護の目的と割り込み

- 多重プログラミング:複数の利用者によるコンピュータの共用
 - コンピュータの使用率が改善される反面、
 - 1つのプログラムのバグが多くのプログラムに影響
 - 他人のカード(データ)を自分のものとして入力
 - ・他のプログラムのデータを変更、OSのデータさえも変更
- 不正なプログラムから、OS自身、他のプログラム、データを保護
 - 他のプログラムの誤動作を引き起こさないことを保障
- プログラムの誤りの多くは、
 - ハードウェアによって検出され、 割り込みが重要な役割
- - OSによって処理される
- 外部割込み:ハードウェアに起因する割り込み
- ハードウェアの故障・エラー、入出力完了、<u>時計(タイマ)</u>
- 内部割込み:プログラムの実行が原因で発生する割り込み - 不正命令、メモリ保護違反、システムコール

基礎OSのスライト

オペレーティング・システム(OS)の起動

- OS (正確にはカーネル)はイベント駆動(注)プログラムである
 - 何かが起こるのを待つ
 - マウスのクリック、文字の入力、印刷の終了、メモリのエ
- ハート・ウェアがイヘントを検出し、割込みによりOS(カーネル)を起動
 - そのとき実行していたプログラムを中断し、OS(カーネル)の特定番地 にジャンプ
 - このように、OS(カーネル)は、割込みによってのみ起動される
- 割り込みの種類
 - 外部割込み(external interrupt)
 - ハードウェア割り込み(hardware interrupt)とも呼ばれる
 - 内部割込み (internal interrupt)
 - ソフトウェア割り込み(software interrupt)とも呼ばれる
 - 外部割込みを「割込み」、内部割込みを「割り出し(trap)」と言う人 もいる

外部割り込み

- 直接的にはプログラムの実行に起因しない割り込み (ハードウェアに起因する割り込み)
- 外部割込みの種類
 - マシンチェック割り込み
 - ハードウェアの故障
 - メモリの読み出しエラー(パリティチェックエラー)など
 - 入出力割り込み
 - 入出力装置からの動作終了通知
 - 異常通知など
 - 時計(タイマ)割り込み

基礎OSのスライド

内部割り込み

基礎OSのスライド

- プログラムの実行が原因で発生する割り込み
- 内部割込みの種類
 - プログラムの誤り
 - 演算例外(ゼロ除算、オーバフロー)
 - 不正命令違反(定義されていない命令コードの使用)
 - メモリ保護違反(許可されていないメモリ領域へのアクセス)
 - 仮想記憶におけるページフォールトもこの一種
 - システムによっては、ページフォールト専用の割込み番号を 持つもの有り
 - _ システムコール
 - その他(デバッグ用、エラー解析用など)

実行モード

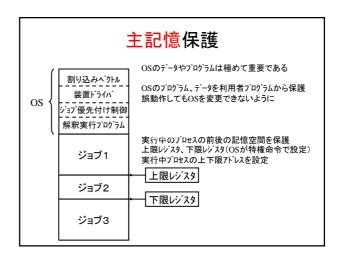
- 非特権モード(ユーザモード)
- 一般のAPを実行するモード。特権命令は実行不可
- 特権モード(カーネルモード)
- OSのカーネルを実行するモード。特権命令も実行可能
- 特権命令
 - 入出力命令 (入出力保護に使用)
 - 記憶管理レジスタ変更命令 (記憶保護に使用)
 - タイマ起動、停止、タイマ値変更命令 (CPU保護に使用)
 - プログラムの実行制御のために用意された特別な命令割込み禁止命令、割込み禁止解除命令
 - 実行モード切替命令
 - HALT命令(CPUを停止させる命令)
- CPU保護:特定のAPによるCPUの独占(無限ループ)に対する保護
 - APを実行中とする時にタイマを設定(タイマ値>量子時間)。
 - CPUバーストが続けば、無限ループと判断。

入出力保護

- 入出力保護:入出力誤りに対する保護
 - 入出力誤りの例
 - 他人のデータを自分のものとして入力
 - 割り当てエリアをオーバした出力
 - 保護の方法
 - 装置ドライバをOSが準備し、入出力要求の正当性を検査
- APのプログラマに対し,強制的に装置ドライバを使用させる必要がある
 - 入出力命令を<mark>特権命令と</mark>する(APは入出力命令を使用不可)
 - APが入出力命令を実行すると不正命令の内部割込み
 - (APは非特権モードで実行するため)
 - 入出力要求のシステムコールを提供する
 OS(装置トライバ)が起動され、特権モードで入出力命令実行

重要:入出力保護の実現(1) レディ (非特権モート・) 実行中 入出カ P1 (非特権モード) 内部割込み 割込 エラー 処理 スケシューラ kill P2開始 →分析 (特権モード) 入力装置 システムコール ● 特権命令 APのプロセス(P1)が特権命令を直接実行するとエラーとなる 内部割込みによりOSに制御が渡る 割込み分析の結果(不正命令)、OSは利用者プログラムを終了させ、エラー処理 (その後、他のプログラムを実行させる)

重要:入出力保護の実現(2) 再開 実行中 停止 レディ (非特権モート*) 実行中 入出力要求 待機 レディ 内部割込み割込 割认 入出力トライハ・スケジューラ 分析 ↓ 分析 (特権モード) スケシ・ューラ 外部割込み 入出力動作 入出力装置 -● システムコール 完了 ● 特権命令 APのプロセス(P1)が入出力要求のシステムコールを発行 内部割込みによりOSに制御が渡る 割込み分析の結果(システムコール)、OSはプロセスP1を待機中にし、特権命令を実行 (OSの実行は特権モー、で行われるので、エラーとはならない) 入力が終わると、外部割込み(入出力完了)によりOSIに制御が渡る 割込み分析の結果、プロセスP1をレディ状態にする(P2が終わればP1が実行)



資源へのアクセス制御機構

領域(アクセス主体) オブジェクト(資源) プリンタ 操作 利用者 ネットワーク プロセス ファイル ホスト ディレクトリ 読み、書き、実行 追加、削除

保護の論理的なモデル: アクセス制御行列(実現には用いない) 行列:主体×オブジェクト。各要素は、オプジェクトに対し、主体が許されている操作。

実現法1:アクセス制御リスト

オブジェクト毎に、く主体、操作>の組からなるリストを持たせる Unix:主体(所有者、グループ、その他)、保護モード(rwx)の組 Windows:主体と許される操作、許されない操作のリスト(ACL)

実現法2:資格(Capability)リスト

主体に、<オブジェクト、操作>の組からなるリストを持たせる Unix:ファイルをオープンする際にプロセスにファイル識別子を渡す

モデル(アクセス制御行列)と実現法

モデル:アクセス制御行列

オブジェクト	ファイル1	ファイル2	ファイル3	カート・リータ・	プリンタ
領域	(F1)	(F2)	(F3)	(CR)	(P)
一般ユーサ [*] A	読み		読み		
一般ユーサ [*] B				読み	書き(印刷)
グループC		読み	実行		
所有者D	読み・書き		読み・書き		

疎行列(sparse matrix:中身が殆ど空)なので、効率が悪い

実現法1:アクセス制御リスト F1:<A, r>, <D, rw> F2:<C r>

F3: <A, r>, <C, x>, <D, rw> CR: <B, r>

実現法2:資格リスト A:<F1,r>, <F3,r> B:<CR, r>, <P, w> C:<F1, w>, <F3,x> D:<F1, rw>, <F3, rw>

P: <B, w>

Unixのファイル保護(重要)

アクセス制御リスト方式:ファイル(オプジェクト)毎に、主体に許されるアクセス許可 モート・(操作)を管理(所有者、グループ、他者に分け、操作の可否を管理)

所有者 グループ 他の利用者

d r w x r w x r w x

- r:読出し(read) 該当ビットが
- w:書込み(write) 1:許可 x:実行(execute) 0:不許可
- d: ディレクトリ識別 1: ディレクトリ

0:ファイル

 $\left. \frac{d\mathbf{r} \mathbf{w} \mathbf{x}^{------}}{1111000000} \right\} \hat{r}^{-}$ (ループ、他者は全て不可) におは全て不可)

-rw-rw-rw-0110110110 計 でも、読み書き可。実行不可

5

8進	rwx	意味
0		すべて不可
1	x	実行のみ可
2	-w-	書きのみ可
3	-wx	書き・実行可
4	r	読みのみ可
5	r-x	読み・実行可
6	rw-	読み・書き可
7	rwx	すべて許可

実現法の比較

	アクセス制御リスト	資格リスト
要求条件の記述	0	×
情報の局在化	×	0
効率	×	0

UNIXでは、領域は、利用者に対応する。 大規模システムでは、領域(利用者)数が多い。

保護の方法も重要だが、ここで最重要なのは、幾つかの方式を比較し、どれを採用するかの決定方法(上の表) 知識も必要だが(7ウセス制御リスト、資格リスト、情報の局在化、効率・・というの

は知ってますが、どちらの方式の方が良いかは分かりませんでは困る)、知

識を組み合わせて実務に適用する方法論の方が大事。 (情報卒は、ソフトを使ったシステムに従事する人も多いだろうから・・・・)

実現例(アクセス制御リストと資格の併用) 利用者名(領域)はOSが知っている ファイル名、操作 ①open ファイル 3[index] ディレクトリ プロセス 4[index] (2) アクセス制御リスト ファイル制御表 探索 index(資格) これに時刻情報などを連結 (古い資格でアクセスできない ようにする) 登録 ファイル 操作 領域、アクセス法 ⑤close時に抹消 予めファイル作成時に定義 Unixの場合、このindex値をファイル記述子 Unixの場合、iノードリストに相当 (file descripter:fd)という

ファイル入出力プログラムの例

ファイルへの書込み

int fd;

fd = open("foo", O_WRONLY|O_CREAT|O_TRUNC); write(fd, "hello\u00ean", 6); close(fd);

ファイルの読出し

int n. fd: char buf[10]; fd = open("foo", O_RDONLY); n = read(fd, buf, 10);printf("n=%d, buf=|%s|\n", n, buf); close(fd);

簡単のため、エラー処理や文字列長の指定を省略