

第8回 保護とセキュリティ(1)

入出力保護, アクセス制御

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、ハードウェア故障、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)

安全性に関する特性

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
 - 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
- **外部割り込み: 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、I/O完了、時計(タイマ)
- **内部割り込み: 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

オペレーティングシステム(OS)の起動

基礎OSのスライド

- 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、メモリのエラー、...
- ハードウェアがイベントを検出し、**割り込み**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、**割り込みによってのみ起動される**
- 割り込みの種類
 - **外部割り込み**(external interrupt)
 - ハードウェア割り込み(hardware interrupt)とも呼ばれる
 - **内部割り込み**(internal interrupt)
 - ソフトウェア割り込み(software interrupt)とも呼ばれる
 - 外部割り込みを「割り込み」、内部割り込みを「**割り出し**(trap)」と言う人もいる
- 注: イベント(event: 事象)=出来事

外部割り込み

基礎OSのスライド

- 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)
- 外部割り込みの種類
 - 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
 - I/O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のスライト

- ・ **プログラムの実行が原因**で発生する割り込み
- ・ **内部割り込みの種類**
 - － **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り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）

- **非特権モード(ユーザーモード)**
 - 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**
 - I/O命令 (入出力保護に関連)
 - 記憶管理レジスタ変更命令 (記憶保護に関連)
 - タイマ起動、停止、タイマ値変更命令 (CPU保護に関連)
 - プログラムの実行制御のために用意された特別な命令
 - 割り込み禁止命令、割り込み禁止解除命令
 - 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API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ーした出力
 - 保護の方法
 - **装置ドライバ**をOSが準備し、I/O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
 - I/O命令を**特権命令**とする(APはI/O命令を使用不可)
 - APがI/O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)
 - I/O要求のシステムコールを提供する
 - OS(装置ドライバ)が起動され、特権モードでI/O命令実行

The diagram illustrates the process flow from P1 to P2. It consists of three main horizontal timelines: P1 (Non-privileged mode), OS (Privileged mode), and Input Device (入力装置). Above these, a green bar represents the execution time of P2, which is divided into two phases: 'Waiting' (レディ) and 'Executing' (実行中).

- P1 Timeline:** Starts with '実行中' (Executing). It then issues an 'I/O命令' (I/O command), leading to '内部割込み' (Internal interrupt). This is followed by '終了' (End) and 'エラー分析' (Error analysis).
- OS Timeline:** Receives the 'エラー分析' (Error analysis) from P1. It then performs 'エラー処理' (Error processing), which leads to 'スケジューラ' (Scheduler). The scheduler then 'kill' P1 and 'P2開始' (Start P2).
- Input Device Timeline:** Shows '入力装置' (Input device) receiving a 'システムコール' (System call) and executing a '特権命令' (Privileged command).

The diagram also includes a legend indicating that green dots represent 'システムコール' (System call) and red dots represent '特権命令' (Privileged command).

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)

The diagram illustrates the execution flow of a process (P1) in a multi-processor system. The diagram shows the interaction between P2 (Non-privileged mode), P1 (Non-privileged mode), OS (Privileged mode), and I/O devices. P1 issues an I/O request, which is handled by the OS. The OS then schedules the process to wait for I/O completion. Once the I/O device completes its operation, the OS schedules the process to resume execution. The diagram also shows the process being preempted by P2 and resuming execution later.

Legend:

- システムコール (System Call)
- 特権命令 (Privileged Command)

APのプロセス(P1)がI/O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を待機状態にし、特権命令を実行(OSの実行は特権モードで行われるので、ユーザとはならない)

I/O動作が終わると、外部割込み(I/O完了)によりOS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)

OSのデータやプログラムは極めて重要である

OSのプログラム、データを利用者プログラムから保護
誤動作してもOSを変更できないように

実行中のプロセスの前後の記憶空間を保護
上限レジスタ、下限レジスタ(OSが特権命令で設定)
実行中プロセスの上下限アドレスを設定

ジョブ1

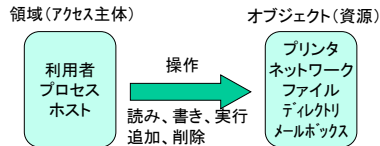
ジョブ2

ジョブ3

上限レジスタ

下限レジスタ

資源へのアクセス制御機構



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、オブジェクトに対し、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、<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

領域 \ オブジェクト	ファイル1 (F1)	ファイル2 (F2)	ファイル3 (F3)	カードリーダー (CR)	プリンタ (P)
一般ユーザA	読み		読み		
一般ユーザ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>
P: <B, w>

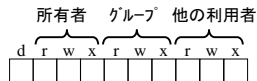
実現法2: 資格リスト

A: <F1, r>, <F3, r>
B: <CR, r>, <P, w>
C: <F1, w>, <F3, x>
D: <F1, rw>, <F3, rw>

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 (オブジェクト) 毎に、主体に許されるアクセス許可モード (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)

r: 読出し (read) 該当ビットが 1: 許可
w: 書込み (write) 0: 不許可
x: 実行 (execute)



d: ディレクトリ識別 1: ディレクトリ
0: ファイル

例

drwx----- ディレクトリ、所有者は全て可 (読み書き実行可)
1111000000 グループ、他者は全て不可
7 0 0

-rwxr-xr-x ファイル、
0110110110 誰でも、読み書き可。実行不可
6 6 6

drwxr-xr-x ディレクトリ、所有者は全て可。
111101001 グループは読み実行可、他者は実行のみ可
7 5 1

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	rx-	読み・実行可
6	rwx	読み・書き可
7	rwX	すべて許可

実現法の比較

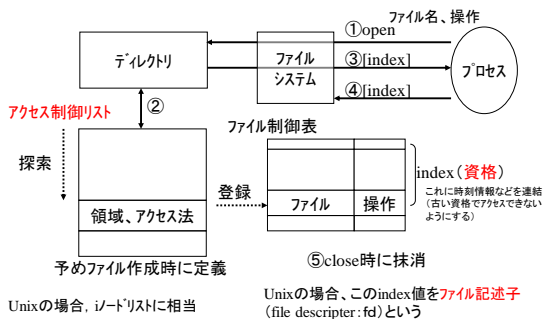
	アクセス制御リスト	資格リスト
要求条件の記述	○	×
情報の局在化	×	○
効率	×	○

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、ここで最も重要なのは、幾つかの方式を比較し、どれを採用するか決定方法 (上の表)
知識も必要だが (アクセス制御リスト、資格リスト、情報の局在化、効率... というのは知っていますが、どちらの方式の方が良いかは分かりませんでは困る)、知識を組み合わせる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



Unixの場合、iノードリストに相当

Unixの場合、このindex値を**ファイル記述子** (file descriptor: fd) という

ファイル入出力プログラムの例

ファイルへの書き込み

```
int fd;
fd = open("foo", O_WRONLY|O_CREAT|O_TRUNC);
write(fd, "hello\n", 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