オペレーティングシステム(OS)

柴山　潔

10. プロセス管理 (4)

- プロセススケジューリング
- スレッド

プロセススケジューリングとプロセスの性質 (1)

➢ プロセススケジューリングは，ユーザプロセスが共用/共有するハードウェア資源（プロセッサ，メインメモリ，入出力装置）の時間的/空間的有効利用のため

■ ハードウェア資源の使用要求に関するプロセスの性質

- プロセッサバウンド（CPUバウンド(CPU bound)）：プロセッサ(CPU)だけの使用を主に要求，入出力装置の使用要求はなしor少
  → 実行中/実行可能状態プロセスが多
  → プロセッサを一定期間占有する可能性が高
- I/Oバウンド(I/O bound)：入出力装置(I/O)の使用を主にor多く要求
  → 待ち状態プロセスが多
  → プロセッサは遊びや待ちが多

（プロセスの性質の利用例）

- 「プロセッサバウンドのプロセスとI/Oバウンドのプロセスとは，見かけ上同時実行しても，使用資源の競合はほとんどなし」という経験則をプロセススケジューリングに適用
  → プロセッサと入出力装置との時間的かつ空間的な有効利用を図る！

プロセススケジューリングとプロセスの性質 (2)

■ プロセス(の性質)の判定(定量的)指標(例)

(1) プロセッサ時間:そのプロセスの実行中状態時間(←普通は既知ではない)の総和

→「プロセッサ時間が長」はプロセッサバウンド

(2) 入出力時間:そのプロセスが入出力(関係)処理に要する時間の総和

→「入出力時間が長」はI/Oバウンド

(3) 入出力命令数:そのプロセスが実行する入出力関係命令の総数

→「入出力命令数が多」はI/Oバウンド

(4) ブロック要因:そのプロセスの待ち状態への状態遷移(ブロック)が入出力をOSに依頼するSVC命令(=ブロック割り込み)の実行によるか否か?

→「ブロック要因」はI/Oバウンド

プロセスのスケジューリングアルゴリズム

■ プロセススイッチそのものへのOSのかかわり方による分類

(A) 横取りなし（ノンプリエンプティブ(non-preemptive)，横取り不可能）

- いったん実行中状態になったプロセスは，自身が待ち状態へ遷移/消去になるまでプロセッサを未解放
- OSは「実行中状態のプロセスが自律的に待ち状態に遷移/消去」を待つ
  - このアルゴリズムによるプロセススイッチに対しては，OS（プロセススケジューラ）は受動的に動作

(B) 横取りあり（プリエンプティブ(preemptive)，横取り可能）

- 「OSが，強制的（能動的）に，実行中状態プロセスを実行可能状態へ遷移させる（＝横取り，プリエンプション(preemption)）」を許す
  - 横取りの場合，スケジューリングアルゴリズムにしたがって，実行可能状態プロセスのうち（実行可能キュー）から1個を選定，実行中状態へ遷移させる

スケジューリングアルゴリズムとプロセス状態遷移 (1)

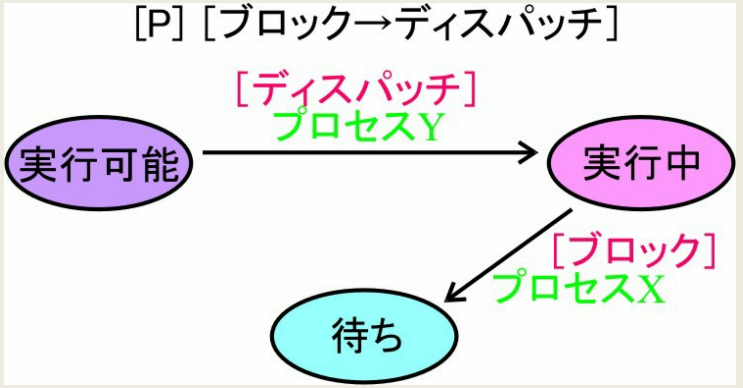
■ 割り込みに伴うプロセス管理手順
1. 割り込み処理
2. プロセススケジューリング
3. プロセスディスパッチ

■ スケジューリングアルゴリズム[手順2]とプロセス(状態)管理[手順1～3]との関係
([プロセスX → プロセスY]プロセススイッチ)

(a) ブロック割り込みの場合

◆ 横取りなし/横取りありのどちらでも↓([P])

- 実行可能状態プロセスのうちからスケジューリングアルゴリズムによって選定するプロセスYがディスパッチによって実行可能→実行中状態遷移
- プロセスX自身は，自分が発したブロック割り込みによって，(2のプロセススケジューリングの前に)実行中→待ち状態遷移



スケジューリングアルゴリズムとプロセス状態遷移 (2)

(b) ブロック割り込み以外の場合

◆ 横取りなし↓ (**[Q]**)

- 実行中状態のまま中断しているプロセスXが再開

◆ 横取りあり → 次のどちらかをスケジューリングアルゴリズムが選定

↓

- 実行中状態のまま中断しているプロセスXが再開(**[Q]**)
- 実行中状態のまま中断しているプロセスXが横取りによって実行中→実行可能状態遷移して，かつ，実行可能状態プロセスのうちからスケジューリングアルゴリズムによって選定するプロセスYがディスパッチによって実行可能→実行中状態遷移(**[R]**)

スケジューリングアルゴリズムの選定指標 (1)

［定義2.12］ **プロセッサ(CPU)利用率**

- コンピュータシステムの総稼働時間に対してプロセッサの有効動作時間（＝何らかのプロセスを実行している時間＝何らかの仕事をしている＝使用時間＝アイドル(idle; 不使用)時間以外の時間，＝全プロセスの有効時間の総和）が占める割合

➢ 実際には，各プロセスの有効動作時間を定量的に測るのは困難

↓

➢ アイドルプロセス(＝有効な仕事を何もしないで時間だけを消費するプロセス，最低優先度，他に実行可能状態プロセスがないときだけ実行中状態に)を 常時稼働しておき，「アイドルプロセスが実行中状態にある時間以外＝プロセッサの有効動作時間」に

スケジューリングアルゴリズムの選定指標 (2)

■ プロセッサの時間的有効利用度を測る個別指標（プロセッサ利用率以外）

- スケジューリングアルゴリズムが評価（＝スケジューリング）で利用
- 「適用しているスケジューリングアルゴリズムが適切か」の判定に使用

(a) スループット(throughput)：プロセッサが単位時間あたりに行う仕事量（＝完了できるプロセス数）
[スケジューリング目標] 最大に！

(b) ターンアラウンド時間(turnaround time)：あるプロセスの実行を要求（＝プロセスを生成）してから完了（＝消去）するまでの経過時間
＝ 実行可能状態時間＋実行中状態時間＋待ち状態時間
＝ プロセスの寿命（＝ライフタイム）
[スケジューリング目標] 最小に！

(c) 待ち時間(waiting time)：あるプロセスが生成～消去（＝寿命，ライフタイム）に実行可能キューで費やす時間
＝ 実行可能状態時間＋待ち状態時間
＝ ターンアラウンド時間（＝プロセス寿命）－実行中状態時間（＝プロセッサ時間）
[スケジューリング目標] 最小に！

(d) 応答時間(response time)：「あるプロセスの実行を要求してから最初の応答を得る（＝応答を開始する）」までに要する時間
[スケジューリング目標] 最小に！

スケジューリングアルゴリズムの選定指標 (3)

- 実際の「スケジューリング」や「スケジューリングアルゴリズムの選定」では，
  - いずれの指標を優先するのか？
  - 各指標において，最大値/平均値/最小値のどれを重視するのか？

  の詳細を具体的に設定

■ マルチタスキングの多重度：生きている（生成～消去）プロセスの総数

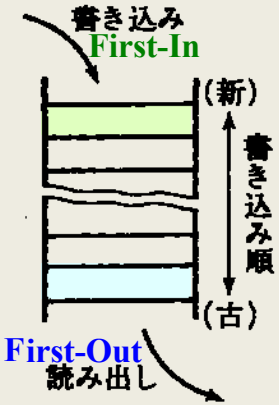
［スケジューリング目標］安定（一定）に！

- (a)～(d)の定量的な指標によるプロセススケジューリングの定性的な目標

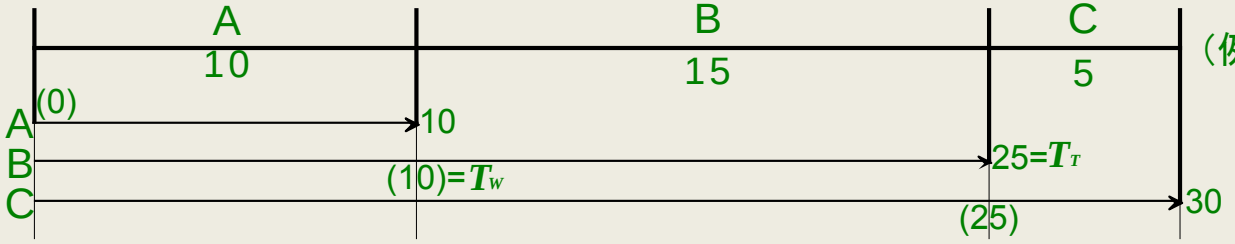
横取りなしスケジューリングアルゴリズム（例）(1)

(N-1) **FCFS**（First Come First Service; 到着順）：実行可能状態に遷移した順に実行中状態に

- ○ 実現する機構は単純 ← 実行可能キューが実行可能状態になったプロセスをFIFO (First-In First-Out)順に保持
- × プロセス特性の活用が不可能
- × プロセッサ時間（実行中状態時間）が長いプロセスの実行順が先だと，（平均ターンアラウンド時間(T_T)＋平均待ち時間(T_W)）が大（全体効率が悪）← 横取り不可能

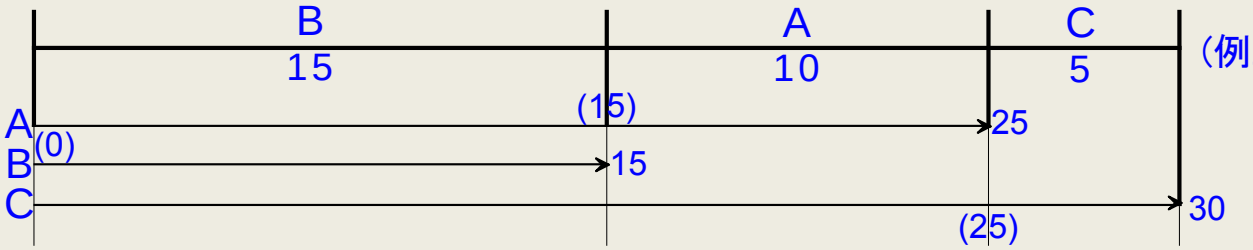


(2) キュー



（例1）
T_T = 22
(T_W = 12)

（例）プロセス	処理時間
A	10
B	15
C	5



（例2：最悪の場合）
T_T = 23
(T_W = 13)

（参考2.6）

（再掲）

キュー(queue)

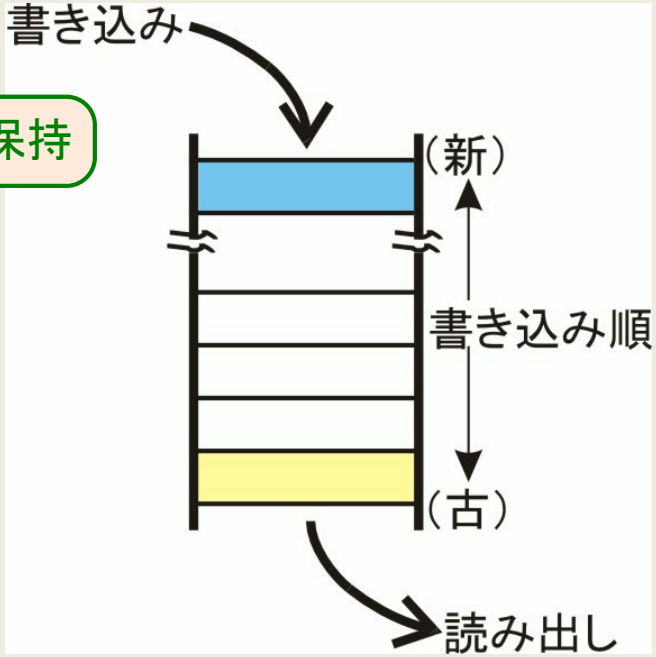
- 書き込みと読み出しの独立したアクセスポートを両端に備える1次元メモリ（構造）
  - FIFO (**F**irst **I**n **F**irst **O**ut):“一番最初に格納したものを一番最初に読み出す”順でアクセス
  - “書き込み順序を保持して，その書き込み順（正順）で読み出す”機能を自然に装備

［**OS**での適用例］

(a) 実行可能/待ち状態プロセスのPCBリスト

(b) FCFSのスケジューリングアルゴリズムでのプロセス実行順序の保持

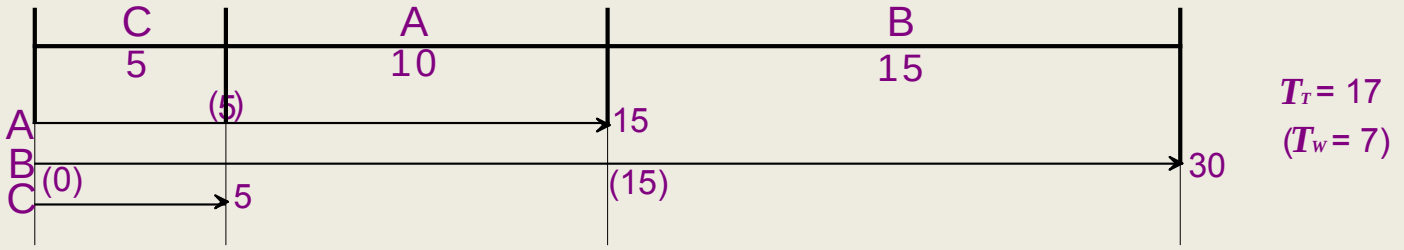
◆ メインメモリを流用&ソフトウェア機能で実現



横取りなしスケジューリングアルゴリズム（例）(2)

(N-2) **SJF**（Shortest Job First; 最短要求時間順）：最も短いプロセッサ時間（＝実行中状態時間）を要求するプロセスを最優先して実行中状態に

- ○ 実現するための機構は簡単
- ○ 最小（最良）の（平均ターンアラウンド時間＋平均待ち時間）を保証
- × プロセッサ時間（＝実行中状態時間）の既知が要件 → 非現実的



C→A→B

（例）プロセス	処理時間
A	10
B	15
C	5

横取りありスケジューリングアルゴリズム（例）(1)

(P-1) 優先度順：基準［例：(1)制限時間（＝デッドライン(deadline)）；(2)メインメモリの使用量；(3)要求プロセッサ時間；(4)入出力処理時間；(5)OS外の要因による緊急性］にしたがって各プロセスに優先度を付与，各時点での最高優先度のプロセスを実行中状態に

- タイマ（割り込み）でプロセッサ時間を計測（通知）

○プロセスの特性を活用したスケジューリングが可能

×「低優先度プロセスを強制的に実行中状態に（実行する）」というスケジューリングは原則なし

→ 無限のブロック（定義2.13）問題が発生する場合に対処が必要

［定義2.13］ 無限のブロック (infinite blocking)

- 「いったん付与した優先度は固定・変更不可」の場合，低優先度プロセスがいつまでたっても実行中状態にならない（＝実行されない）現象

＝ 無限の延期，飢餓

横取りありスケジューリングアルゴリズム（例）(2)

■ 優先度順(P-1)において，高い優先度を付与する指標

(a) 最短実行時間：最短ジョブ(SJ, Shortest Job)を優先　　（＝ SJF法）
- 最短ジョブがあらかじめ分かっている必要があるのでバッチ処理向き

(b) 最小残余時間：要求時間のうち未使用時間が最小のプロセスを優先

(c) I/Oバウンド型プロセスを優先（プロセッサバウンド型プロセスよりも）

(d) 同一優先度どうしでは**FCFS**で

(e) エージング(aging)：長時間システムに滞在しているプロセスの優先度を時間経過につれて徐々に高く
- 無限のブロック問題を解決

(f) あらかじめ資源利用率を割り当て，割り当てた時間は制限時間として保証

横取りありスケジューリングアルゴリズム（例）(3)

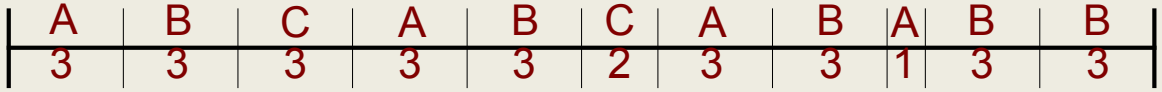
(P-2) ラウンドロビン(round robin)：原始的な時分割制御(TSS)，一定時間（＝タイムスライス）ごとにプロセスを固定順で切り替え

- タイマ（割り込み）でプロセッサ時間を計測･管理
- 性能はタイムスライス長に依存 → タイムスライスが極端に長いとFCFSと同じに，短いとプロセスコンテクスト切り替え時間がオーバヘッドとして顕在化

○ 実現が簡単

× プロセス特性を不利用

A→B→C



[タイムスライス＝3]

（例）プロセス	処理時間
A	10
B	15
C	5

（上級）

多重レベルスケジューリング

［定義2.14］ **多重レベルスケジューリング (multi-level scheduling)**

- プロセスを特性別に分け，それぞれを複数の独立する実行可能キューで管理し，それぞれの特性に適した個別のスケジューリングアルゴリズムを適用するスケジューリング

（例）会話型OS → ラウンドロビン；リアルタイムOS → 優先度順；

- 「どの実行可能キューを優先するか」の決定にスケジューリングアルゴリズムを適用することも

［定義2.15］ **多重レベルフィードバックスケジューリング (multi-level feedback scheduling)**

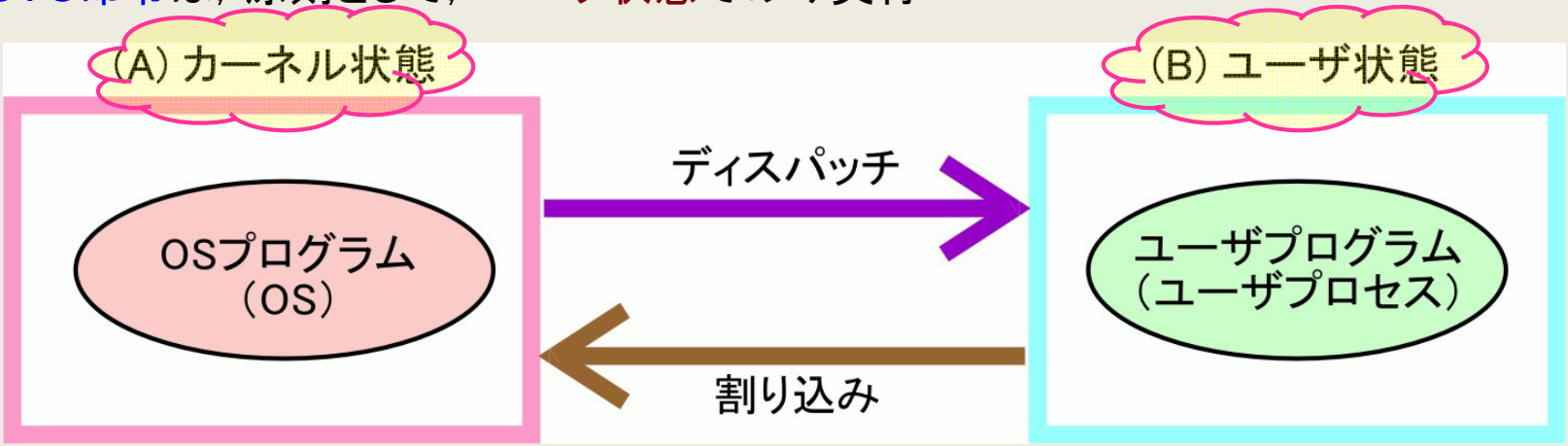
- プロセスの実行可能キュー間移動を許す多重レベルスケジューリング

（例）プロセッサバウンドのプロセスを低優先度キューへ，低優先度キューで待ちが長いプロセスを高優先度キューへ

- 現代のプロセススケジューリングで最も一般的 ← カスタマイズ(customize)や変更が容易

プロセッサ状態
＝ 実行モード，プロセッサモード

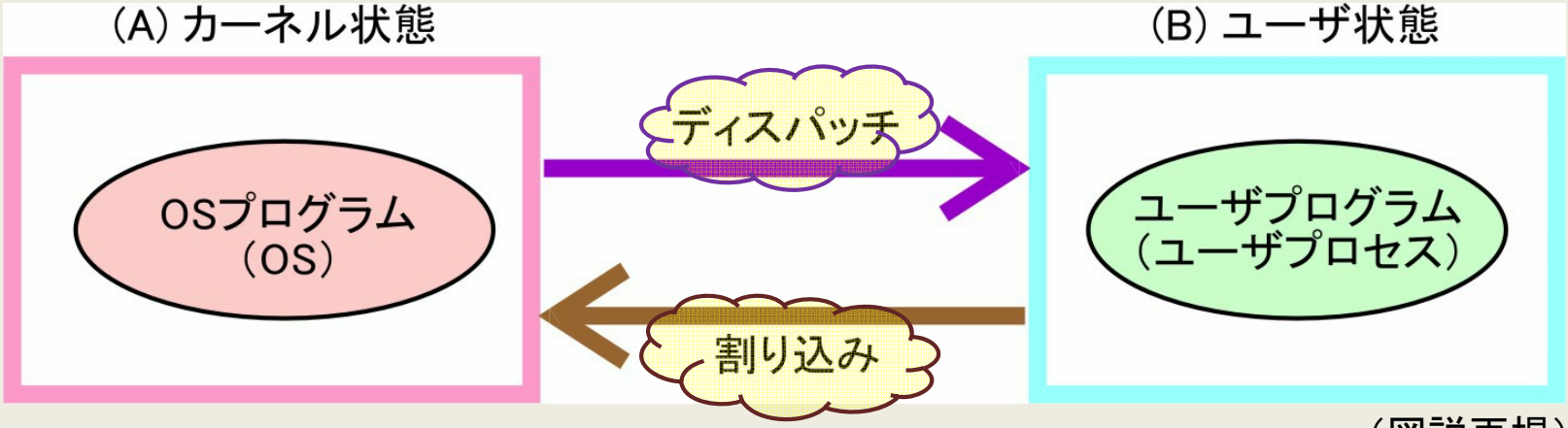
- プロセッサがユーザプログラム（ユーザプロセス）/OSプログラム(OS)のどちらを実行しているか？
  - (A) カーネル状態：OSそのものを実行しているときのプロセッサ状態
    ＝ カーネルモード，特権モード，システムモード，スーパバイザ状態
    - ➢ OSしか実行が許されていない命令（＝ 特権命令，例：不可分操作など）も実行可能
  - (B) ユーザ状態：OS以外のユーザプログラム（＝ユーザプロセス）を実行しているときのプロセッサ状態
    ＝ ユーザモード
    - ➢ SVC命令は，原則として，ユーザ状態でのみ実行



プロセッサ状態の遷移例（図）

- 割り込みによって，ユーザ状態→カーネル状態
- ディスパッチによって，カーネル状態→ユーザ状態

> ➢ プロセッサ状態は，PSW（プロセッサ状態ワード）の構成項目，プロセススイッチやスレッドスイッチ時にOSが切り替え



（図説再掲）

【まとめ】割り込みとカーネル（再考）－OSカーネル機能の観点－

■ **OSカーネル**：カーネル状態で動作する**OSの中核**機能

- (1) 割り込み処理；(2) プロセス管理（特に，プロセス状態やその遷移の管理）；(3) プロセススイッチ（主として，プロセスディスパッチ）；などの**OSの基本的な機能**を実現
- 各種の割り込みによって起動，割り込み処理をきっかけに(1)～(3)を遂行

【まとめ】割り込みとカーネル（再考）
－OSとハードウェアとのトレードオフの観点－

■ 割り込みが**OS**（ソフトウェア）とハードウェア機構に与える実際的効果

(A) ユーザプログラム（実際には，マシン命令列）の実行を強制的かつ動的に中断，**OS**プログラムの実行へ分岐（変更）

(B) プロセススイッチの発生

(C) ブロック割り込みは「実行中状態のユーザプロセスを待ち状態へ遷移させる」割り込み要因（＝ブロック要因）そのもの

(D) ウェイクアップ割り込みは「待ち状態のユーザプロセスを実行可能状態へ遷移させる」ウェイクアップ事象

(E) プロセッサ状態が（ユーザ状態から）カーネル状態へ移行 & **OS**カーネルが起動

割り込み：「ハードウェア機構やユーザプログラム（ソフトウェア）による**OS**（ソフトウェア）の呼び出し or **OS**（ソフトウェア）への依頼」機能

スレッド

［定義2.16］ **スレッド (thread)**

- プロセスから生成できる「マシン命令の実行/制御の流れ（＝制御フロー）」
- プロセッサにおける「動的な実行単位」

＝軽量プロセス

  - 現に実行している（＝実行中状態の）プロセスが動的に生成

- **スレッド**の実際例
  - (a) OS自身が生成するスレッド：ユーザプログラムに「スレッド生成」指令（実際には，マシン命令列）を埋め込んでおく
  - (b) ループ（loop; くり返し）：コンパイラで検出可能
  - (c) 基本ブロック：ハードウェア機構で検出

↓

(a)～(c)それぞれを構成するマシン命令列の実行 ＝ **スレッド**

スレッドの特徴 —プロセスとの比較の観点—

- 有効時間/生きている(生成～消滅)時間(寿命，ライフタイム)がプロセスよりも短
- より局所的な実行環境(＝スレッドコンテクスト(thread context))
- スレッドコンテクストは，プロセスに割り付けたハードウェア資源 (＝物理的なプロセスコンテクスト)に比べて，空間サイズは小，占有時間は短

➢ スレッドの切り替え(＝スレッドスイッチ)はプロセススイッチよりも軽快で速

■ スレッド：実行主体としてハードウェア資源(特に，プロセッサ)へ割り当てる最小単位
＝プロセッサにおける動的な実行単位

↓ ↑

■ プロセス：実行主体としてハードウェア資源(特に，メインメモリ)へ割り付ける最小単位
＝メインメモリへの静的な割り付け単位

スレッドコンテクスト（例）—プロセスコンテクストとの比較—

- **スレッドコンテクスト**
  - (a) プロセッサ状態
  - (b) コンディション（条件）
  - (c) プログラムカウンタ(PC)
  - (d) 汎用レジスタ
  - (e) メモリイメージ：ヒープ，スタックフレームのみ（プロセス領域の一部）

- **プロセスコンテクスト**
  - (a) プロセッサ状態
  - (b) コンディション（条件）
  - (c) プログラムカウンタ(PC)
  - (d) 汎用レジスタ
  - (e) メモリイメージ：コード，データ，ヒープ，スタックフレームのすべて（プロセス領域の全部）
  - (f) メインメモリについての管理情報
  - (g) 割り込みの優先度

➢ スレッドの切り替え（＝スレッドスイッチ）はプロセススイッチよりも軽快で速い

スレッド管理

(A) スレッド生成：プロセスからスレッドを生成

(B) スレッド割り当て：スレッドをプロセッサ（時間）に割り当て

(C) スレッドスイッチ：実行スレッドを別のスレッドに切り替え

- 「(B)スレッド割り当てと(C)スレッドスイッチとを併せた機能として実現する」場合もあり

■ スレッドを導入する効果

- スレッド管理は「スレッド状態&その状態遷移の管理」や「スレッドスケジューリング」
  ← プロセス管理を全面的にスレッド管理に置き換え

（上級）

マルチスレッド

- ある1個のプロセスは，「並列に（並行して，同時に）実行できる複数個のスレッド」（＝マルチスレッド）を自然に生成

■ 同時マルチスレッドアーキテクチャ（Simultaneous Multi-Thread architecture; SMTアーキテクチャ）：**OS**がプロセスから生成する複数個スレッドを並列に実行/処理可能なアーキテクチャ

- 同時マルチスレッドアーキテクチャを採るコンピュータの**OS**
  - 「並列実行可能なハードウェア資源（特に，プロセッサ）に並列実行可能な複数個スレッドを適切に割り当てる」機能が必須