5 割込み

5–1 割込みとは何か? —割込みと例外—

計算機を稼働している際には、ハードウェアの誤動作, 電源異常, ユーザプログラムのエラー, 並行して独立に動作させている入出力機器からのメッセージ到着、といった事象 (event) が起こり得る。

これらの事象は緊急の処置を要するものが多いので、通常はこれらの事象に対して、CPU の使用主体を OS(の然るべきルーチン) に切替えて、

(1) 計算機がその時点で実行している処理を一旦中断し、
(2) 起こった事象に対する処置を完了した上で、
(3) 元々実行していた処理を再開する、

といったことを OS に行ってもらう。
これらの切替え機能を一般に割込み (interrupt) と言う。
また、割込みを引き起こす原因を割込み要因、
個々の割込み要因に対して行われる処置を割込み処理と言う。

補足： 吉沢 (2000) や岩波情報科学辞典では 0 による除算なども割込み事象の一種 (内部割込み) と見ているが、谷口 (2000) は対処の仕方/目的の違いを理由に割込みと区別し、0 による除算を始めとしたソフトウェア異常に対処する機能を例外と呼んでいる。

同様の見方の違いは実際のコンピュータシステムにおいても見受けられる。

割込みが必要な理由：

計算機の生産性の低下を防ぎ、またシステムを安全に保つために、いつ起こるか分からない割込み要因に対して即座に CPU の使用主体を OS(の然るべきルーチン) に切替えて、OS が緊急の処置を施す必要がある。

割込みレベル：

割込み処理中に別の割込みが発生する可能性もある。 そのため、各々の割込みには割込みレベルが設定され、割込み処理の間の実行の優先度が決められている。

5–2 割込みの種別とOSの処理の概要

外部割込み：

実行中のプログラムとは関係なく発生するもの

(1) ハードウェアの誤り検出

命令フェッチ時のパリティエラー、瞬時の電源異常など。

⇨ 保守のためにログとして記録する。要因に固有の対応処置も。

(2) 入出力処理終了の報告 {⟹ 5.4節}

入出力コントローラ訂正から「入出力完了」の通知

⇨ (必要なら) 入出力コントローラに対する次の制御メッセージ。
入出力完了のプロセスを待ち状態から実行可能状態に移す。

補足：

CPUの処理効率を上げるには、低速な入出力装置をどう制御するかも重要な課題となる。

⇨ 通常、入出力装置をCPUと独立に動作させる。

(3) タイマ

監視時計から「指定した時刻を過ぎた」という通知

⇨ それに対応する処置

タイマの使用例 ：

端末からの入力が5分以内に完了しない時は会話を打ち切る。

(4) 外部信号割込み

外部の測定器、操作ボタンからの情報

⇨ 外部からの情報を収集し、
それを担当するプロセスに処置させる。

内部割込み (例外) :

プログラム実行の結果として発生するもので、トラップ, 割出しとも言う。

(5) ソフトウェアの誤り検出

Segmentation Faults, Bus Error, ゼロ除算など。

⇨ 異常処理ルーチンが用意されていればそれを実行する。
用意されてなければ当該プロセスを異常終了させる。

(6) ページフォールト/アドレス変換例外

仮想記憶方式において、プログラムがアクセスした番地のブロックが実メモリ上にない。

⇨ ページインする。すなわち、該当ページまたはセグメントに対して、実メモリを割り当てる。 ($\Longrightarrow$ **9.8**節を参照。)

(7) システムコール/カーネルコール

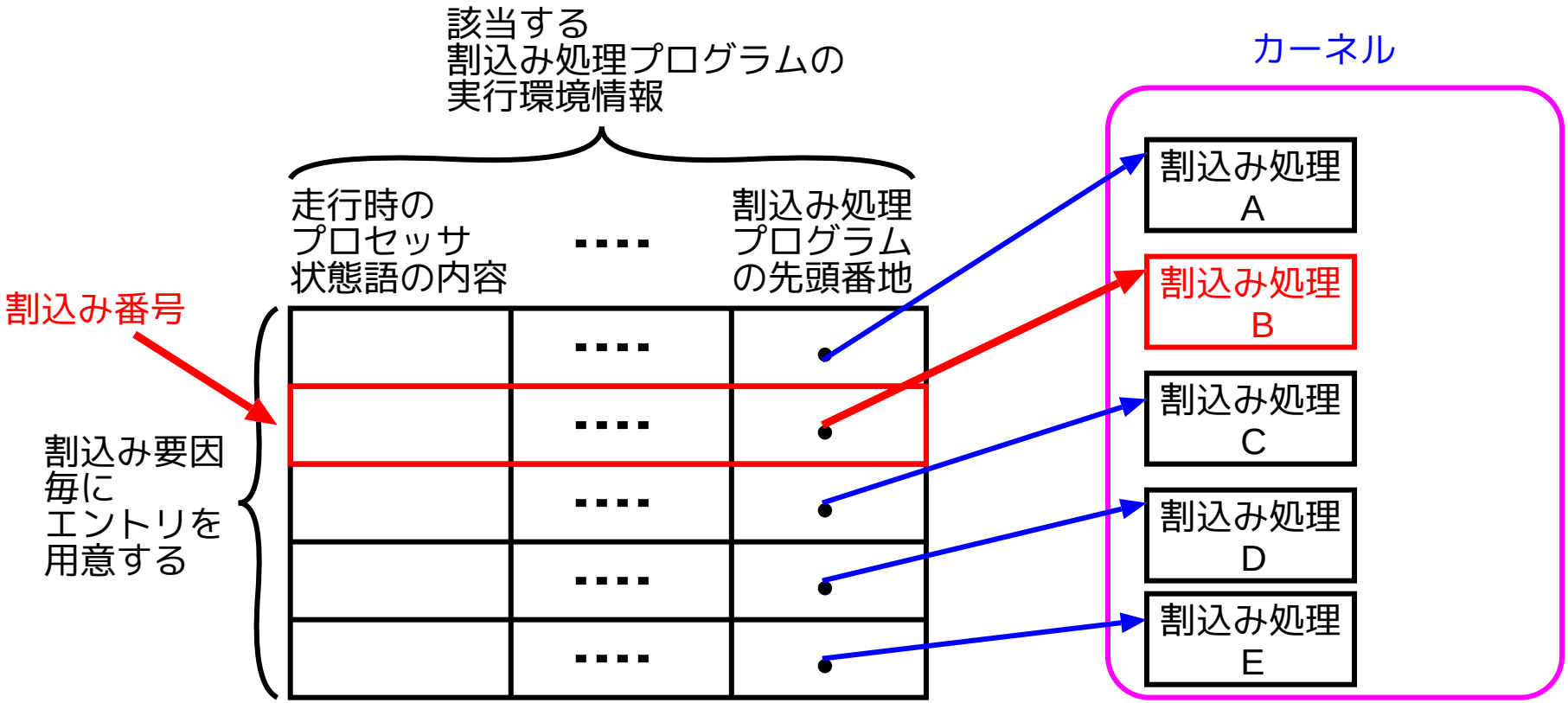
OSの機能呼出しがプログラム内に書かれている。

⇨ **CPU**のモードを権限の強いモードに切替えて、
指定された機能を実行する。

5–3 割込み処理呼び出しの機構

割込み処理ルーチンのアドレスをどうやって得るか：

- 各々の割込み種別/番号から対応する割込み処理ルーチンのアドレスを引くための表(割込みベクタ表と言う) を用意。



- その表へのポインタをレジスタに記憶する。 表を引く際はIndexing

割込み事象が発生すると...：

次のような手順で処理が切替えられていく。

割込み要求をすぐに受け入れるかどうかの判断：

(1) 生起した割込み事象の割込みレベルと現在の走行レベルを比較し、

(1.1) もし現在の走行レベルの方が(優先度が)高ければ、割込み要求は待たせて現在の実行を継続する。

(1.2) もし生起した割込みのレベルの方が高ければ、次のステップ(2)に移る。

前処理：　　...(「受け入れ」の判断の場合)

(2) レジスタ群の内容を然るべき領域に退避する。

(3) 利用するスタックを切替える。

個別処理：

(4) 走行モードをスーパバイザモードに、走行レベルを生起した割込みのレベルに変更する。

(5) 実行要求のあった割込み処理を起動する。例えば、

(ゼロ除算の場合は)

ゼロ除算に対処するルーチンが用意されていればそのルーチンを実行する。

用意されてなければゼロ除算を起こしたプロセスを強制終了させ、ステップ(6')に移る。 ディスパッチャの呼出し

(カーネルコール呼出しの場合は)

指定された機能を実行する。 ···(e.g. 入出力装置への指示)

入出力要求などの場合には、現在実行しているプロセスを実行中からイベント待ち状態に変更し、ステップ(6')に移る。

ディスパッチャの呼出し

(割当てられたCPU時間を使い切った場合は)

現在実行しているプロセスを実行中から実行可能状態に変更し、ステップ(6')に移る。

ディスパッチャの呼出し

(入出力処理終了の報告の場合は)

その装置に対する入出力要求が他にあれば、次の要求に応えるために装置に次の制御メッセージを送る。

また、終了報告のあった入出力を行ったプロセスを待ち状態から実行可能状態に変更する。
そして、次にステップ(6) または ステップ(6')に移る。

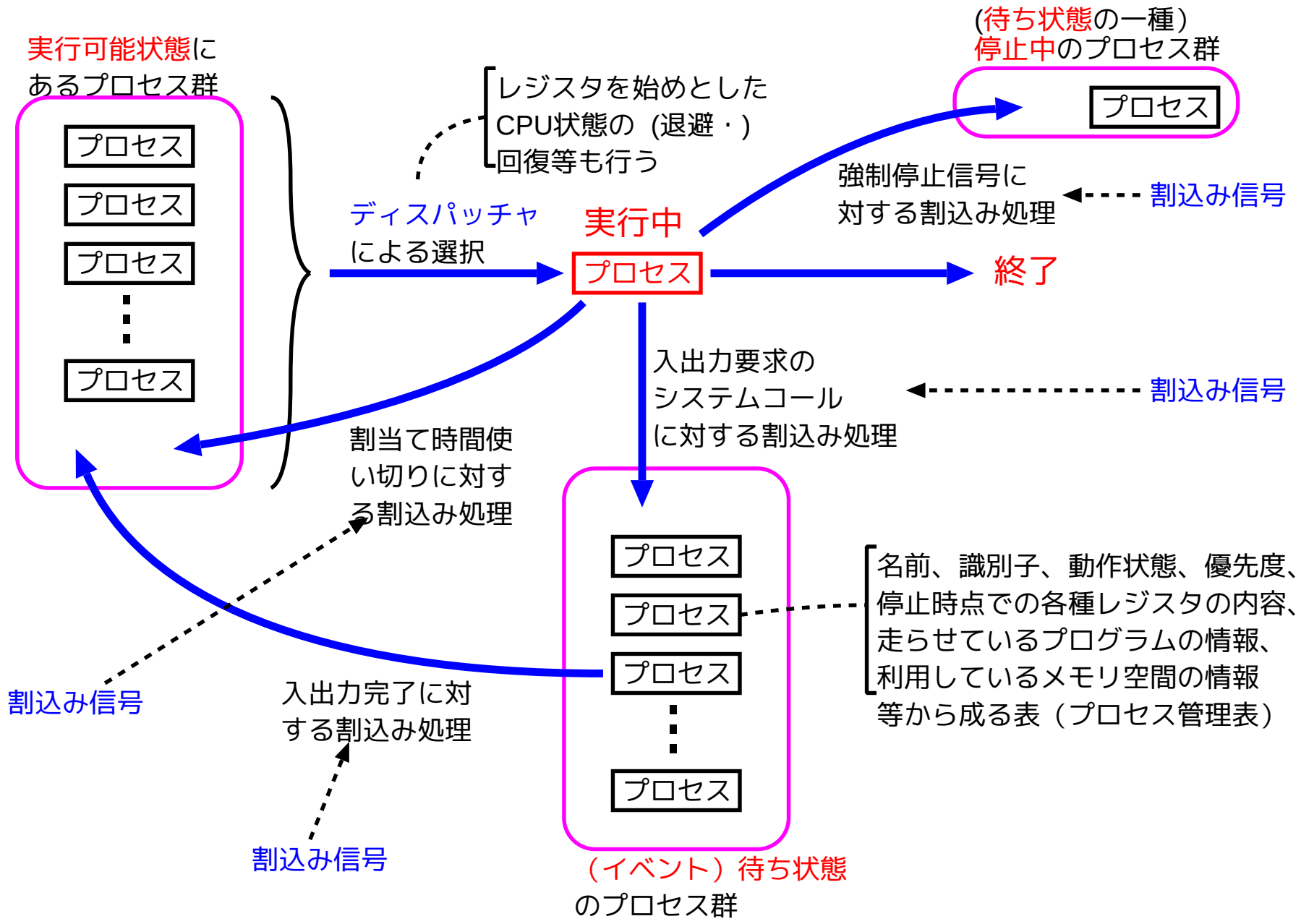
ステップ(6)と(6')のどちらに移るか：

non-preemptive スケジューリングを採用している場合は、次にステップ(6)に移る。

プリエンプティブ・スケジューリングを採用している場合は、次にステップ(6')に移る。

ディスパッチャ

参考：　・・・(8.1節からの引用)


実行可能状態に
あるプロセス群
プロセス
プロセス
プロセス
プロセス
レジスタを始めとした
CPU状態の（退避・）
回復等も行う
ディスパッチャ
による選択
実行中
プロセス
(待ち状態の一種)
停止中のプロセス群
プロセス
強制停止信号に
対する割込み処理
割込み信号
終了
入出力要求の
システムコール
に対する割込み処理
割込み信号
割当て時間使
い切りに対す
る割込み処理
プロセス
プロセス
プロセス
プロセス
名前、識別子、動作状態、優先度、
停止時点での各種レジスタの内容、
走らせているプログラムの情報、
利用しているメモリ空間の情報
等から成る表（プロセス管理表）
割込み信号
入出力完了に対
する割込み処理
割込み信号
（イベント）待ち状態
のプロセス群

後処理：

(6) レジスタをステップ (2) の前の状態に回復する。

(7) 利用するスタックを切替える。

(8) 走行モード、走行レベルの回復。

(9) 割込み事象発生時に走行していたプロセスを継続して実行する。

(6') 「ディスパッチャ」を呼び出して、その時点で最も優先度の高いプロセスを走らせる。 ···(レジスタ回復等も)

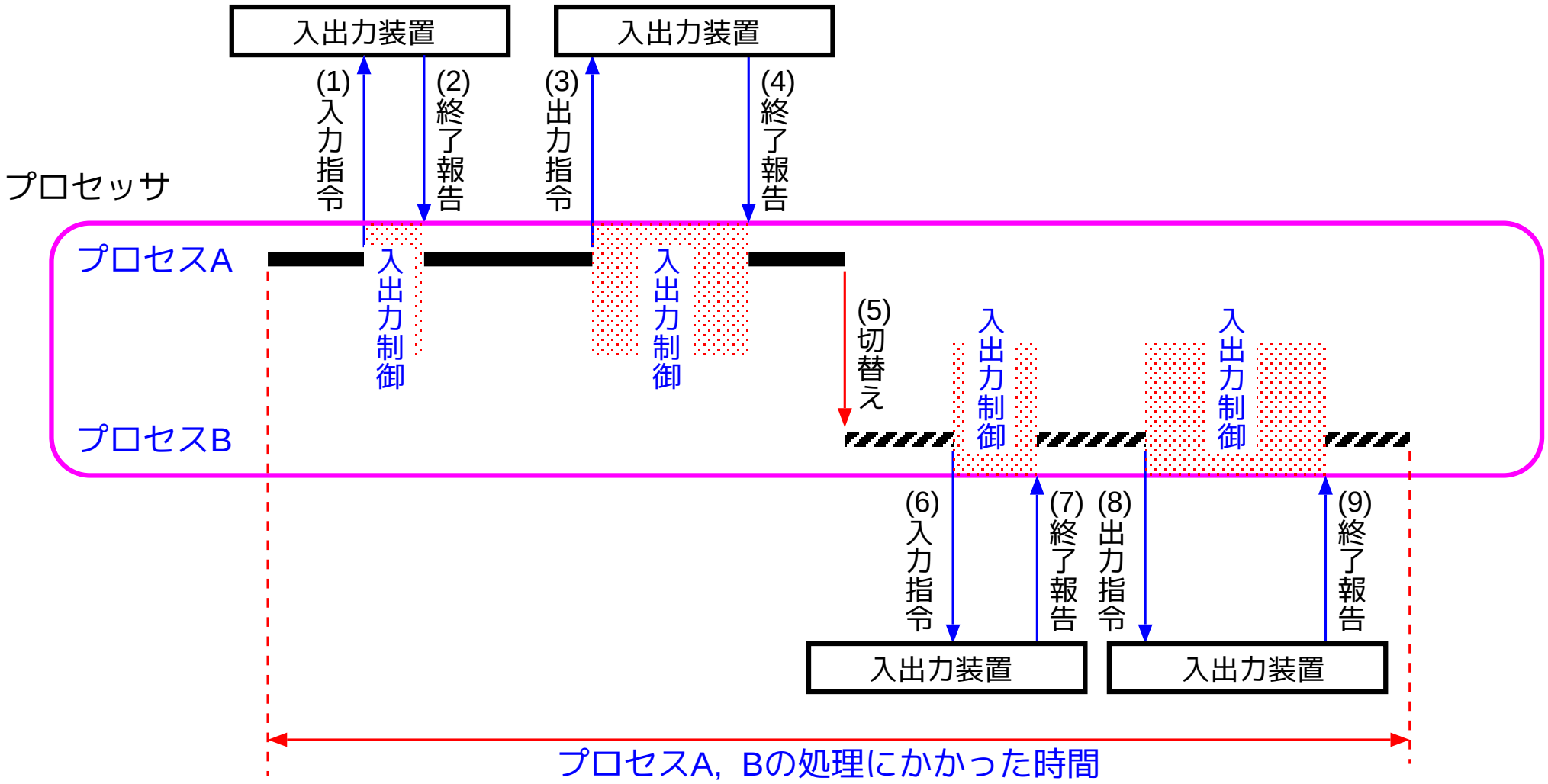
補足：

実行可能状態にある (複数の) プロセスに対して実際にどういう順に CPU を割り付けるかを決める作業をディスパッチ、その作業を行うモジュールをディスパッチャあるいは (CPU の) スケジューラと言う。

⇨ 8.1節

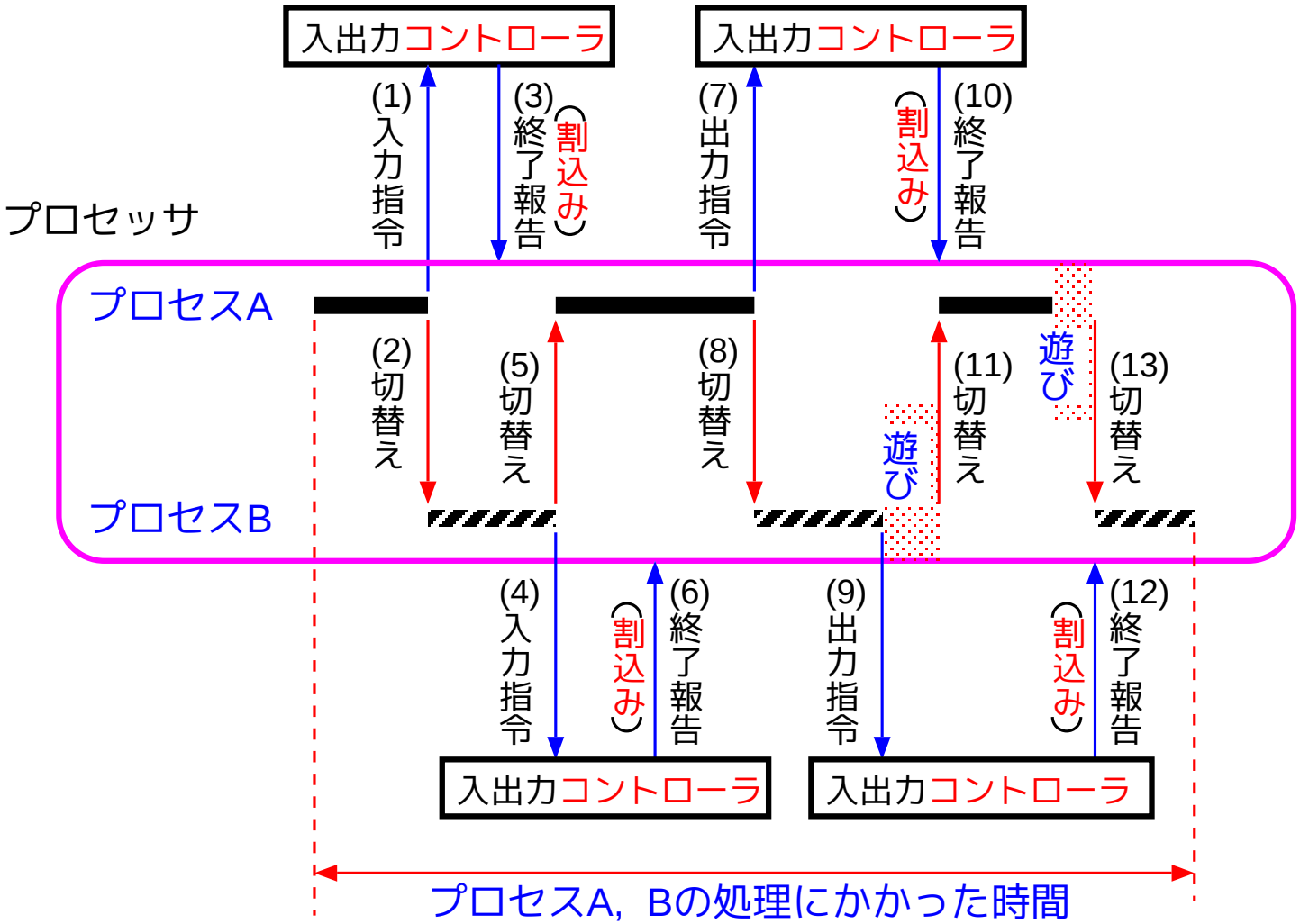
5–4 多重プログラミングを容易にする割込み機構

連続バッチ処理の様に逐次的に１つずつジョブを実行したのでは、入出力動作中はCPUが遊んでしまう。

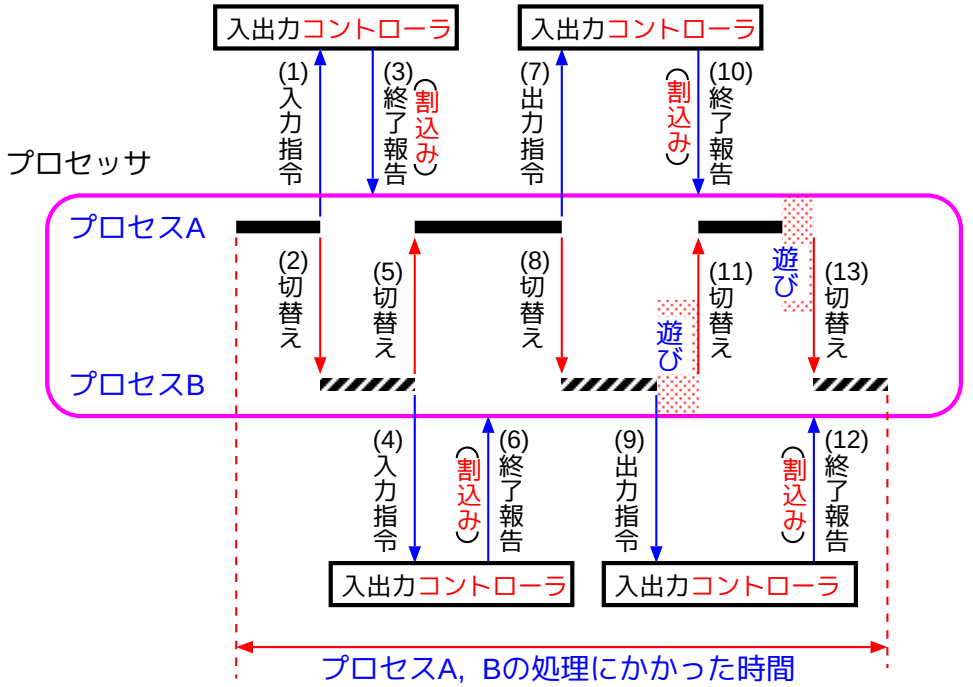


⇨ 多重プログラミング

- 複数のプロセスを並行して走らせる。
- 入出力装置の制御は別の装置に任せる。



⇨ 多重プログラミング



- 一般には入出力コントローラ、特にメインフレームの場合は入出力チャネル

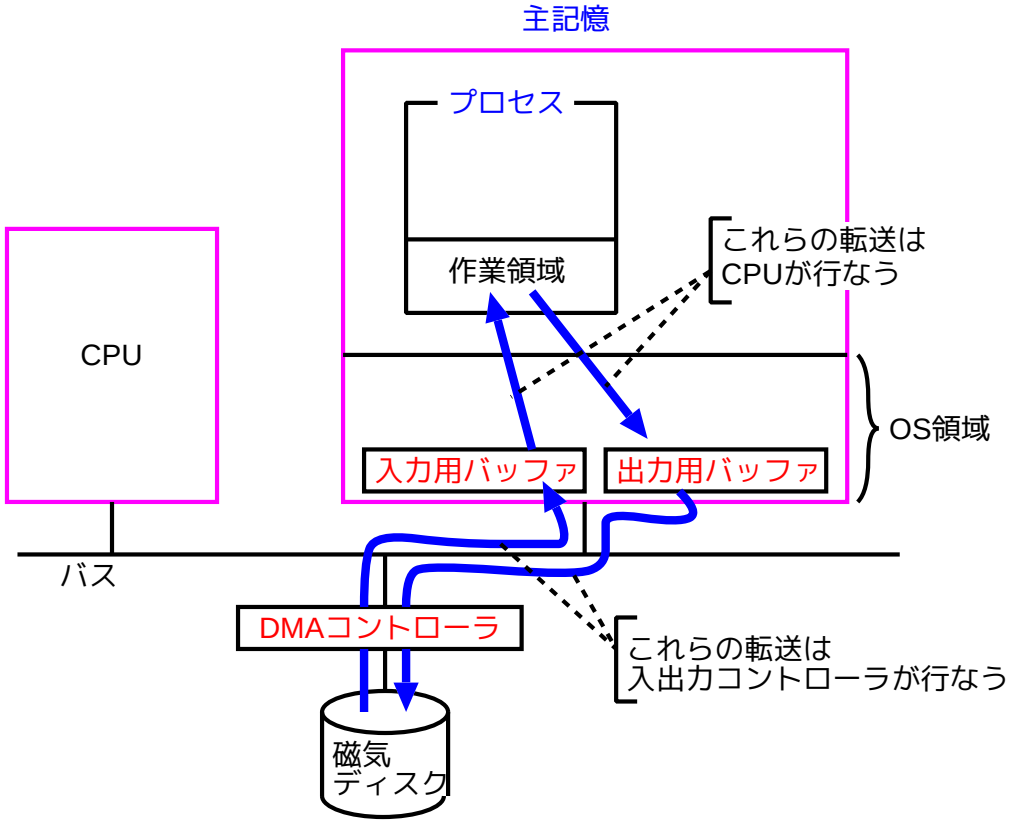
- 実行中のプロセスの次の動作が入出力なら 、
  ① その入出力動作を行うように入出力コントローラに指令を出す、
  ② そのプロセスは一時的に待ち状態にして、
  別の実行可能状態プロセスを（並行して）走らせる。

- 入出力コントローラは入出力動作が完了した時点で割り込み信号を(CPUに対して）発生。

- 入出力終了の割込み信号を受けたら 、その入出力動作を行ったプロセスを待ち状態から実行可能状態に変更する。

補足： 入出力コントローラのことをデバイスコントローラ, 入出力制御装置,入出力制御部と呼ぶこともある。

DMAについて： 主記憶のOS領域内にバッファを設け、CPUを介さずに入出力コントローラが直接そのバッファと入出力装置間でデータ転送を行なうという方式が考えられた。この機構を直接メモリアクセス(DMA) と言い、DMA方式でデータ転送を行なう入出力コントローラをDMAコントローラと言う。



5–5 OSの機能呼び出し

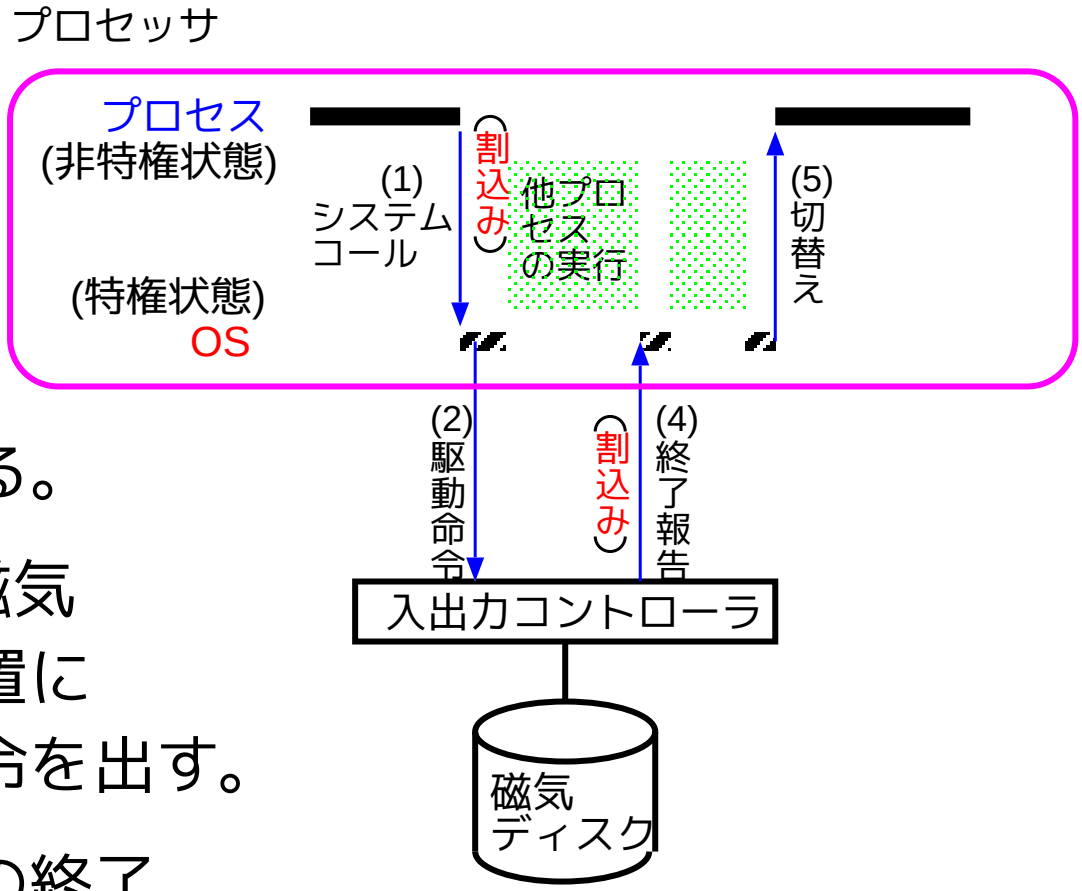
p.15からの引用：
入出力機器は機器毎に制御の仕方が違う。
⇨ 入出力などの共通機能を容易に実行できる環境をハードウェアに付随して提供すれば、プログラマの生産性向上につながる。

- 個々の入出力機器の細かな制御を個々のシステムプログラマに任せてしまうというのでは、プログラマの負担が大きくなる。
- 複数のユーザが共用する資源 (e.g.磁気ディスク,...)を一般のユーザプログラムが独立に操作しては、システムが混乱してしまう。

⇨
- 共用資源に対する操作は、全て依頼の上OSに実行してもらう。
- 共用資源に対する操作は、CPUが特権状態にある場合にのみ実行できるようにする。

一般のプロセスは特権状態になれないので、特権命令は実行できない。

例5. 1 (共用資源に対する操作がどう実行されるか)　一般のプロセスからOSにファイル読み込み要求を出す場合は、次のように処理が進行する。



(1) プロセスはファイル読み込み操作をOSに行ってもらうために割込み(システムコールと言う)を発生させ、待ち状態に入る。

(2) 割込み信号を拾ったOSは磁気ディスクを制御している装置に「ファイル読み込み」の指令を出す。

(3) 「ファイル読み込み」操作の終了。

(4) OSは「ファイル読み込み」の通知(割込み)を受け取ると、プロセスを実行可能状態にする。

(5) CPUが空いていれば実行を再開する。