はじめに

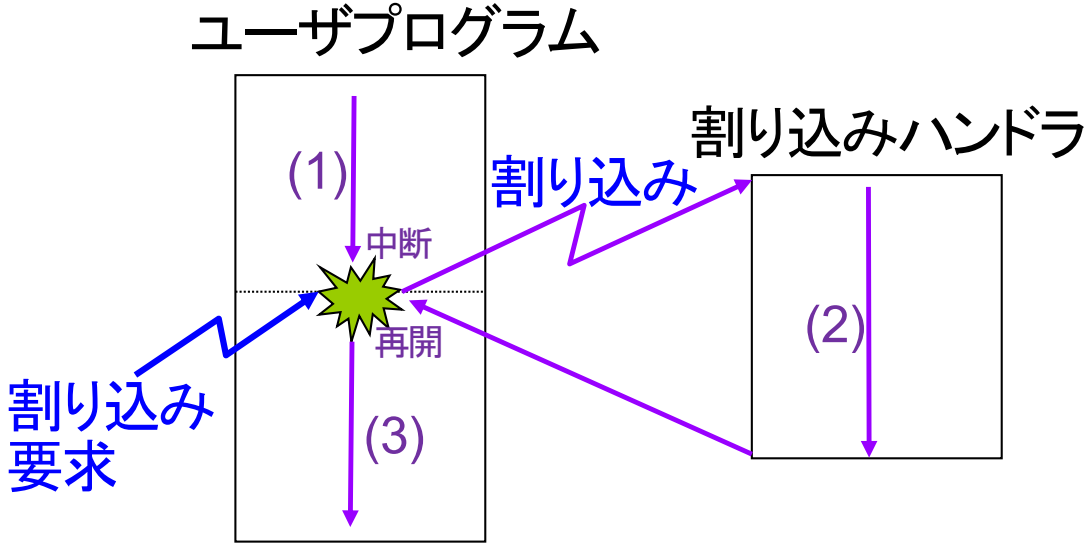
- 今回は、コンピュータにとってとても重要な機能の1つである「割り込み」について講義します。

- 教科書の対応範囲は、以下の通りです。
  - 5.3節（p.149～162）
    - この講義資料では、5.3節のすべての内容について解説しませんが、解説しない部分についても、教科書は読んで理解しておいてください。
  - 5.2.3項(i)（p.147～148）

割り込みとは？

- 割り込みとは？
  - ハードウェアによる算術演算エラーの検出や外部装置（CPU以外）からのイベント通知にリアルタイムに応答する目的で、
    プログラム実行の流れを強制的かつ動的に変更すること、およびその機構（仕組み）を「割り込み（interrupt）」といいます。

– あるプログラムの実行中に割り込みが発生する（割り込み要求が発生してその要求をそれを受け付ける）と、そのプログラムの実行を中断して、それとは全く別（独立）のプログラムの実行を開始します。
  - 割り込みによって実行が開始されるルーチン（プログラム）を、割り込み処理ルーチンまたは割り込みハンドラといいます。
    – 以下の(1)～(3)の順に実行します。



- 割り込まれるプログラムが原因で割り込みが発生する場合もあれば、割り込まれるプログラムとはまったく無関係な原因で割り込みが発生する場合もあります。割り込みが発生したことで、割り込まれたプログラムの実行結果が変わっては困ります。したがって、割り込まれるプログラムの実行内容に変化が生じないように割り込みを制御する必要があります。
- ユーザプログラムだけでなく、原則として、オペレーティングシステム（OS）の実行中でも割り込みを受け付けます。しかし、OSの走行中は、どうしても「今割り込まれては困る」というときがあります。そのために、割り込みを禁止する機能や、受け付ける割り込みの種類を制限する機能をハードウェアで用意し、それを用いてOSは割り込みの受け付けを制御します。もちろん、このような制御ができるのはOSのみです。ユーザプログラムは、常に「割り込み可」の状態で実行されます。

割り込みの必要性

- 割り込みは、ノイマン型コンピュータにとってなくてはならない機能です。
  - その理由は、教科書の5.3.1（a）に列挙（(1)～(6)）されています。
  - それらの理由の源は、以下の3つの事項に起因します。
    - いつ起こるかわからない。→(1)(3)(5)(6)
    - 滅多に起こらない。→(1)(2)
    - 流用。→(4)

    以降で、これらについて順に見ていきます。

- “いつ起こるかわからない”
  - いつ起こるかわからないことにソフトウェア（プログラム）だけで対処しようとすると、ひどく非効率になってしまいます。
  - 例えば、皆さんはゲームをしますね。あなたがゲームコントローラのAボタンをいつ押すか、ゲーム機（コンピュータ）にはわかりません。プログラムであなたがAボタンを押すまで待っていたら、その間、敵が攻撃してくる処理を進めることができません（敵も攻撃してこずに止まったままでいるでしょう）。これではゲームになりませんね。敵の攻撃処理を進めながら、あなたがAボタンを押した時点でそのことを知らせてくれる機能が必要です。これを実現するのが「割り込み」なのです。

- “滅多に起こらない”
  - 例えば、C言語のプログラムで以下のような文を書いたとします。

```
a = b / c;
```

  - しかし、c=0のときは困りますね。
    - 一応、コンピュータ内の除算器はデタラメな値でも出力はします。
  - ちゃんと書くなら、以下のようにしなければなりません。

```
if( c != 0 ) {a = b / c; }
else         {エラー処理}
```

- c=0の可能性がそれなりにあるのなら、やはりこのように丁寧にc=0かどうかを検査すべきです。しかし、c=0になる可能性が低い、あるいは、c=0にはなり得ないことがわかっているなら、
  - このように書くのは面倒で、冗長です。
  - また、if文の実行にも時間がかかりますので、このような記述が多いと、プログラム全体の実行時間も長くなってしまします。
- したがって、ソフトウェアでの検査(if文)はせずに、実際に0で割ろうとした時にだけ、ハードウェアで検出して教えてくれると助かります。

- 流用
  - 入出力の際の入出力コントローラの制御はOSが行います。したがって、ユーザプログラムで入出力を行いたい時は、それをOSに依頼しなければならず、これを行うためのインタフェースを設けなければなりません。
  - ところが、特別にそのインタフェースを設けなくても、割り込みの処理機構がそのまま流用できます。
    - 割り込みハンドラはOSの一部です。
      - 逆に、OSは、その外界とのインタフェースをすべて、割り込みハンドラに統一できます。
    - したがって、あとは、「自分で故意に割り込みを起こす」命令があれば十分です。

「割り込み」を表す術語

- 割り込み要因のニュアンスによって、以下のような術語が用いられる場合があります。
  - 割り込み(interrupt)
    - 「割り込み」を指す一般的な術語です。
  - 例外(exception)
    - 割り込み要因が何らかのエラーを表す場合に用いられる術語です。
  - フォールト(fault)
    - エラーではないけれど、(たまたま)上手くいかなかった、というような事象を表す場合に用いられる術語です。やり直せば、次はうまくいくかもしれません。
  - トラップ(trap)
    - 「そこを踏んだら割り込みハンドラに飛ばされる」というような“罠”のニュアンスを含みます。

割り込み要因の例

- 代表的な割り込み要因について、以下に簡単に説明します。
  - リセット
    - 一応、割り込み要因として挙げてはいますが、特殊です。コンピュータのリセットボタンを押すことによって発生しますが、ハードウェアのレベルで初期化を開始します。割り込みハンドラが呼ばれるわけではありません。
  - 命令実行例外
    - マシン命令の実行に関して、エラーが生じた場合の割り込み要因で、以下のような場合があります。

- 未定義命令使用：　命令語をデコードする段階で、OPコードが定義されていなかった場合に検出します。
- 特権命令例外：　マシン命令には、OSのみが実行できる特権命令と、ユーザプログラムとOSのどちらもが実行できる非特権命令とが設けられています。ユーザプログラムで特権命令を実行しようとした際に発生する割り込みです。
  これを検出・制御するために、ハードウェアで、ユーザモード（非特権モード）とスーパバイザモード（特権モード）の2種類以上の実行モードを設けています（プロセッサ内の制御レジスタのビットでそのどちらかを記憶しています）。OSを実行中の実行モードがスーパバイザモードです。ユーザモードで特権命令をデコードした時に、特権命令例外として検出します。

- アドレス境界違反： コンピュータによっては、データのサイズと格納番地との関係に制約を設けているものがあります。例えば、int型の4バイトのデータは、メモリ番地が4の倍数のアドレスから格納する、など。アドレッシングモードにしたがって求めた実効アドレスがその制約を満たさなかった場合に発生します。
- 演算例外： 演算結果が正しくないことを示すための割り込み要因です。以下のような要因の割り込みを起こします。
  - ゼロ除算： 0で割った場合。
  - オーバフロー： 演算結果が、表現できる範囲を超えて大きくなってしまった場合。
  - アンダーフロー： 浮動小数点数の演算において、演算結果の絶対値が表現できる範囲を外れて小さくなり過ぎ、0とせざるを得なくなった場合。

- ページフォールト（page fault）：　アクセスしようとしたデータがメモリに存在しなかった場合に発生する割り込みです。

  実行中のプログラムは、命令語もデータもメモリに置いておくのが原則ですが、現在では、プログラム中の使用する部分のみをメモリに置き、それ以外の部分はファイル装置（HDDやSSDなど）中の専用領域においています。これを仮想メモリ（仮想記憶）といいます。

  アクセスしようとしたデータがたまたまメモリに存在しなかった場合は、このページフォールトの割り込みによってOSがアクセス対象のデータをファイル装置からメモリにコピーします。割り込みを発生させたプログラムは、実行を再開後に再び同じ番地にメモリアクセスを行いますが、今度はメモリに置かれているはずなので失敗はしません。

仮想メモリについては、次回の講義で詳しく解説します。

- アクセス保護違反： ユーザプログラムが、アクセスしてはならないとされるメモリ番地にアクセスしようとした場合に発生する割り込みです。
  プログラム実行中に、OSや他のプログラムの中身を読み出したり、破壊したりするのを防ぐために設けられた機能です。
  なお、OSには、メモリのどこにでもアクセスすることが許されています。
  - 皆さんは、プログラムを作成する時に、ポインタ変数にNULLを入れたままでその先のデータにアクセスしようとして、プログラムが強制終了されられてしまった、という経験はありますか？ これは、アクセス保護違反の割り込みによって、OSが強制終了させた結果です。一般に、メモリの0番地はOSのみがアクセスできる場所となっています。
    - » 強制終了させられると腹が立ったり悲しくなるかもしれませんが、むしろエラーが見つかったと喜ぶべきでしょう。OSの場合ならアクセス保護違反にはなりませんので、そのようなバグは容易には見つかりません。

- SVC（SuperVisor Call）命令： ユーザプログラムからOSの機能を呼び出すときの割り込みです。このような割り込みを起こすマシン命令を、SVC命令、システムコール命令、トラップ命令、などと呼びます（ISAによって呼び方は異なります）。
- ブレークポイント（トレース）命令： ユーザプログラムでこの命令を実行すると、ブレークポイント割り込みを発生します。
  - プログラムのデバッグの際に、ブレークポイントを設定しますよね？ デバッガはあなたのプログラムを書き換えて、あなたが設定した場所にブレークポイント命令を埋め込んで実行を再開（開始）させます。ブレークポイント命令を実行した段階で割り込みが発生し、あなたのプログラムは中断させられます。制御がOSを経由してデバッガに渡った際に、デバッガは、あなたのプログラムをブレークポイント命令を埋め込む前の状態に戻して、あなたの次の入力を受け付けます。

- ハードウェア異常：　電源異常や温度異常、メモリのパリティエラーなど、ハードウェアの障害や故障に関するセンサからの通知です。
- 入出力割り込み：　入出力動作の完了など、入出力コントローラからの通知です。
  - プロセッサが各入出力コントローラに対して、順番に、入出力動作が完了したかを尋ねる（確認する）方式を、ポーリング（polling）といいます。まだ完了していなかった場合は確認作業自体が無駄だったとも考えられますし、完了していても確認されるまでの間の時間は入出力コントローラを遊ばせておくことになるのでよろしくありません。また、確認して回るための処理を行う間、プロセッサはユーザプログラムを実行することはできません。

- これに対して、割り込みを用いると、効率の良い入出力が可能になります。プロセッサは入出力のための準備をした後、入出力コントローラにデータ転送を指示します。入出力コントローラは、メモリと入出力装置/ファイル装置との間でテータ転送を行い、完了すると、プロセッサに入出力完了の割り込みをかけます。この間、プロセッサは他のプログラムを実行することができますし、また、他に入出力待ちの処理があれば、1つの入出力が完了した時点で即座に次の入出力を開始できるようになります。

– タイマ割り込み： タイマ装置（interval timer）から一定時間が経過したことを知らせるための割り込みです。
  - コンピュータの「時計」は、このインターバルタイマからの通知を契機に、経過時間を累積することによって「時刻」を進めます。

プロセスコンテクスト

- 割り込みが発生すると、割り込まれるプログラムの実行途中の状態を保存しなければなりません。その状態のことを、プロセスコンテクスト（process context）といいます。
  - OSでは、実行中のプログラムをプロセスやタスク（task）などと呼び、ファイルとして格納されている状態のプログラムとは区別して扱います。
  - 割り込みは、マシン命令の切れ目で受け付けます。したがって、プロセスコンテクストも、どのマシン命令までを実行終了した状態かで考えます。

- プロセスコンテクストは、プログラム実行中のプロセッサやメモリの状態で、以下のものを含みます。
  1. 実行する命令アドレス（PCの内容）
  2. データレジスタの内容： 汎用（整数）レジスタや浮動小数点レジスタの内容すべて
  3. 制御レジスタの内容： 条件コード、実行モード（特権/非特権モードの別）など
  4. メモリ上の命令コードおよびデータのすべて
  – 割り込みハンドラは、その実行中にこれらの情報を破壊して（書き換えて）はいけません。4.については、メモリの別の領域を使用すればよいのですが、1.～3.はプロセッサ内に記憶する情報で代替がないので、使用する前にどこかにコピーして保存します。

- 最終的には、メモリに保存します。「プロセスコンテクストを“退避する（save）”」という言い方をします。
  逆に、メモリに退避したコンテクストを戻すときは、“回復する（restore）”と言います。
- データレジスタは数～数十本ありますので、その退避・回復にはそれなりの時間もかかります。
- 昔のIBM社の大型コンピュータ（メインフレームコンピュータ）では、1.と3.をまとめてPSW（Program Status Word）と呼んでいました。現在のコンピュータで正式にPSWを定義して用いているものはありませんが、1.と3.の情報をまとめて今でもPSWと呼ぶことがあります。

典型的な割り込み処理の手順

- 割り込みが発生した時点から、誰（どれ）が何を行うかを、以下に時系列順で示します。
  1） 割り込みの検出・受付（・応答） [*hardware*]
    - ハードウェアで割り込み信号を検出します。
    - 異なる割り込み要因で複数の割り込みが同時に発生する場合があります。割り込み要因ごとに優先度を付けておき、発生した割り込み要因の中で最も優先度の高いものを選びます。
    - 割り込み不可状態でなければ、割り込みを受け付けます。割り込み不可状態の場合や、優先度の低い割り込みは、すぐには受け付けずに待たせておきます。

2a） 割り込み不可能状態への移行 [*hardware*]

2b） PSWの退避 [*hardware*]

- プロセッサ内に設けた退避用の専用レジスタにコピーします。

2c） 割り込み要因の識別 [*hardware*]

- 呼び出す割り込みハンドラの先頭アドレスを求めます。
- 2a）～2c）は、ハードウェアで同時に行います。
- 1）および2c）のハードウェア機構は、教科書の5.3.2項（p.160～162）に詳しく書かれています。

3a） スーパバイザモードへの移行 [*hardware*]

3b） 割り込みハンドラへの分岐 [*hardware*]

- 割り込みハンドラの先頭アドレスをPCにセットします。あとは、普通に割り込みハンドラの命令を順に実行していきます。

4） コンテクストのメモリへの退避 [*software*]

- 割り込みハンドラのプログラムで、まず、汎用レジスタ、浮動小数点レジスタ、一時退避されたPSW、のすべての内容をメモリに退避します。
- 退避先は、OSがプロセスの情報を集約して管理するデータ構造PCB（Process Control Block）またはプロセステーブルです。
- コンテクストの退避を終えると、（可能な限り早期に）割り込み可能状態に移行します。

5） 割り込み処理 [*software*]

- 割り込みハンドラが割り込み要因に対応する処理を行います。必要に応じて、OS内の他のルーチンを呼び出します。

6） プロセススケジューラの実行 [*software*]

- プロセススケジューラはOS内のルーチンで、次に実行（再開）するプロセスを選びます。

7） コンテクストの回復 [*software*]

- 4）で退避したプロセスコンテクスト（ただし、PSWを除く）を回復します。
- この回復作業を担当するOS内のルーチンをディスパッチャ（dispatcher）と呼びます。

8） PSWの回復 [*software & hardware*]

- ディスパッチャの最後で、PSWを回復するための専用命令を実行します。
- 制御レジスタを回復することで、実行モードが元に戻ります。
- PCを回復することで、割り込みによって中断していたプログラムの実行を再開します。あとは、実行を再開したプログラムの命令を順に実行していきます。

割り込みの使用例

- 割り込み要因の解説において、その直接的な使用目的も含めて説明しましたが、ここでは、もっと上のレベルで「割り込み」の機能をどのように活用しているのかを見ていきます。

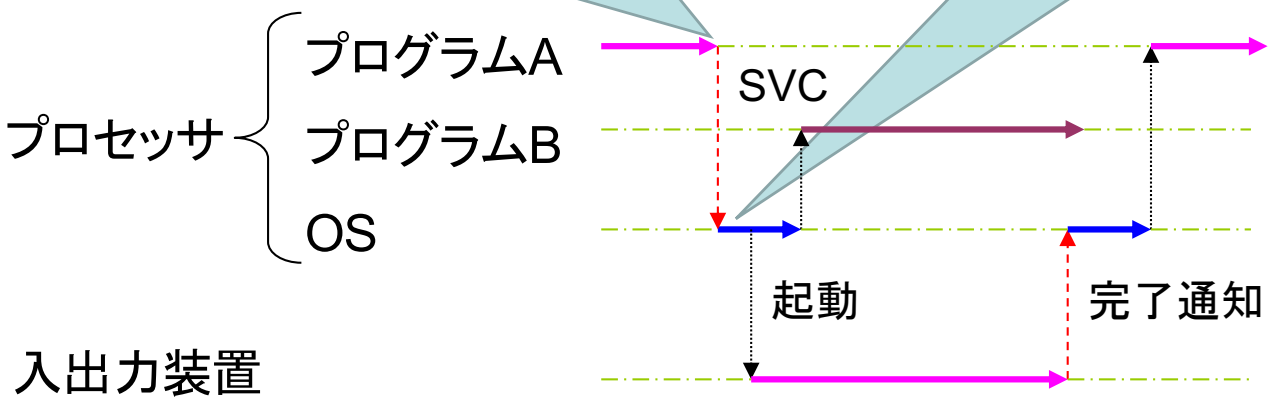
- 例として、「マルチプログラミング」を取り上げます。
  - マルチプログラミング（multi-programming）とは、あるプログラムの入出力動作中にプロセッサが遊んでしまう（有用な仕事をしない）ことを防ぐために、他のプログラムを実行することによって、見かけ上、複数のプログラムを同時に実行しているかのように見せかける方式をいいます。

マルチプログラミング

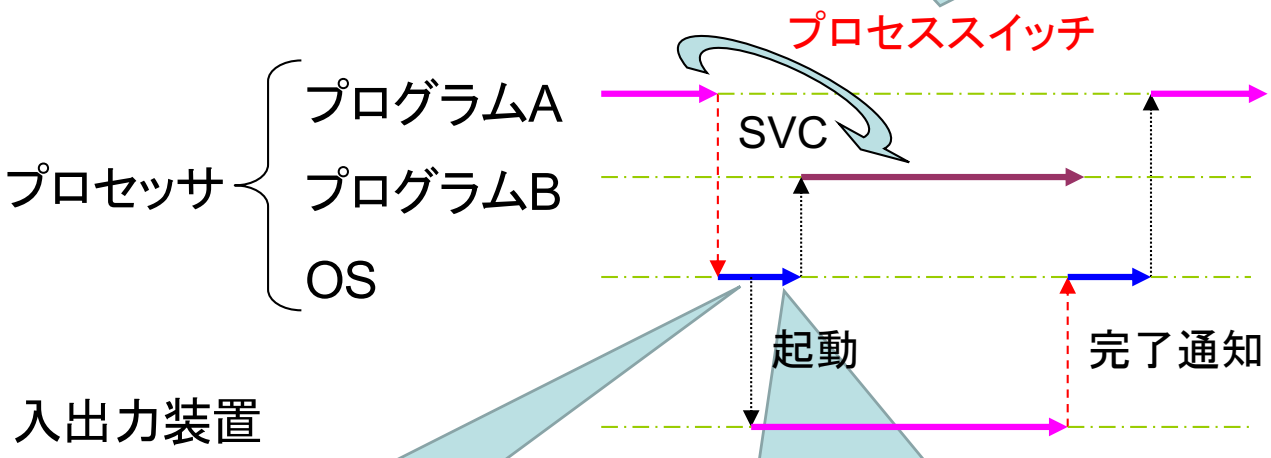
- 典型的な例として、次のようなシナリオを考えます。

プログラムAをプロセッサで実行しているとします。ここで、プログラムAが入出力を行うとき、SVC命令を実行して、OSに入出力処理を依頼します。

SVC命令によって割り込みが発生し、OS（割り込みハンドラ）に制御が渡ります。



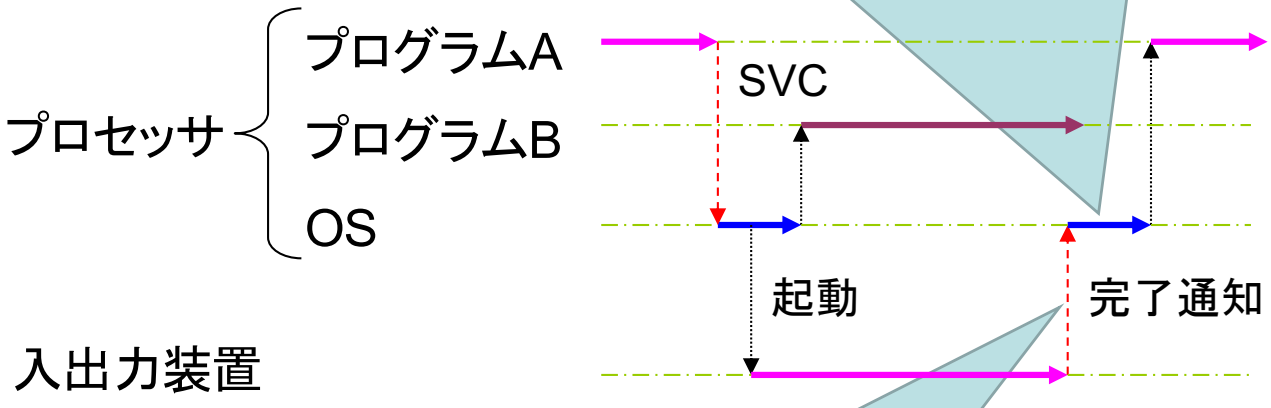
途中でOSの実行を挟みますが、実行するプロセスを切り替えることを、プロセススイッチと言います。



割り込みハンドラは入出力装置を起動します。具体的には、入出力コントローラにデータ転送の指示を出します。

プロセススケジューラで次に実行するプロセスを選択します。プログラムBが選ばれたとすると、プログラムBのプロセスのコンテクストを回復して、プログラムBの実行を再開（開始）します。

プロセススケジューラで次に実行するプロセスを選択します。プログラムAが選ばれたとすると（ただし、プログラムBが再び選ばれる可能性もあります）、そのプロセスのコンテクストを回復して、入出力で中断していたプログラムAの実行を再開します。



データ転送が完了すると、入出力コントローラは入出力完了の割り込みをかけます。これにより、プログラムBの実行が中断され、OSに制御が渡ります。

- 以上のように、プロセスを切り替えることで、プロセッサの遊び時間を減らし、稼働率を上げることができます。

- 現在では当たり前に使われている技術ですが、かなり昔（バッチ処理の時代）に考え出されたため、今となっては「マルチプログラミング」という呼び方も少し変です。
  - 「プログラミング」はプログラムを作成することであって、プログラムを実行することではありませんから。
- 呼び方としては、「マルチタスキング（multi-tasking）」の方がふさわしいのかもしれませんが、これは複数の仕事を見かけ上同時に行うことを意味するものの、プロセッサの稼働率を上げるという目的とは異なる文脈で用いられることも多く、ぴったりの表現とも言えません。
- ということで、ニュアンスの違いを気にしない人はマルチタスキングと呼び、気にする人は、（変だとは思いながらも）仕方なく今でもマルチプログラミングと呼ぶようです。