オペレーティングシステム論
(Theory of Operating Systems)
第四回

ソフトウェア情報学部
基盤ソフトウェア講座
澤本　潤

プロセス：スケジューリング（２）

並行プロセスの定義と生成

- 並行プロセス
  - 同時に並行して実行可能なプロセスを並行プロセス（コンカレントプロセス）という。
  - ここで、同時に実行可能とは、「真に同時に実行した結果」と「任意に逐次順序で実行した結果」が同じである場合をいう。
  - マルチタスキング機能によって実行する複数プロセスは理論上の並行プロセスといえる
  - 並行に実行できないプロセスを逐次プロセスという

並行プロセスの定義と生成

- 並行プロセスの生成
  - UNIXのfork&join
  - Concurrent Pascal の cobegin&coend

プロセスの同期と相互排除

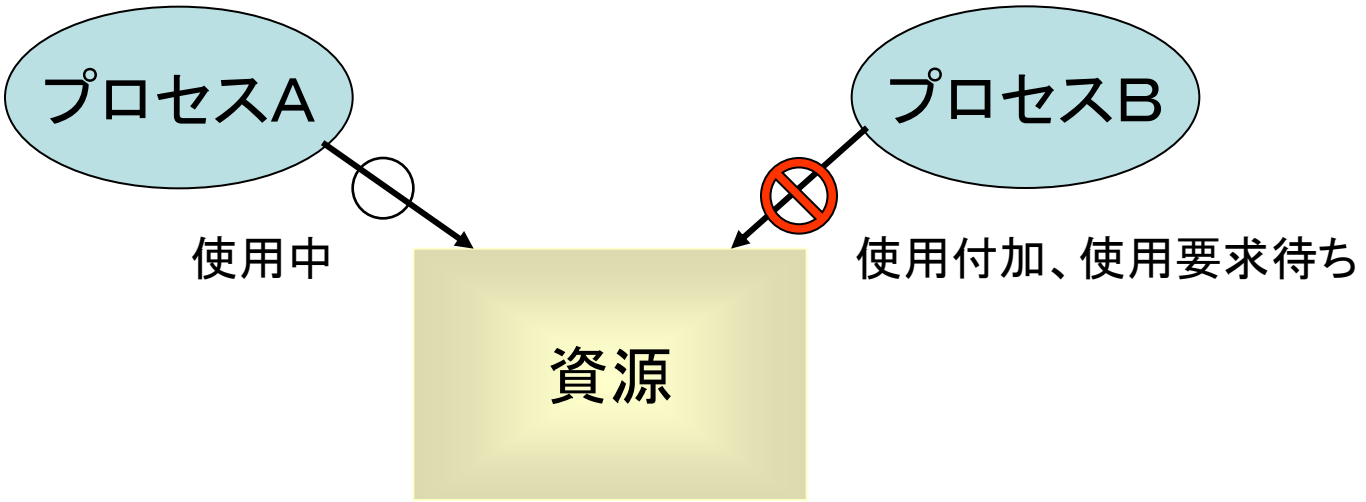
- 同期の必要性
  - 並行プロセスでは、共有もしくは共用する資源に対する使用要求が、「競合」する
    - ハードウェア：メインメモリ、入出力装置
    - ソフトウェア：プログラム、データ　など
  - 相互に情報を送受信する「プロセス間通信」を行う

相互排除の条件

- 相互排除は次の三つの条件を満たす
  - ① 資源がどのプロセスにも利用されていない場合、資源を利用しようとしたプロセスは直ちに資源が利用できる。
  - ② 複数のプロセスが資源を利用しようとしている際でも、資源利用ができるプロセスの選択が無期限に延期されない（デッドロック防止）。
  - ③ どのプロセスも資源を利用できる機会が公平に与えられている（公平性）。

プロセスの同期と相互排除

- 排他制御
  - プロセス実行において、ある1個の共有資源を複数プロセスが同時に使用しないように制御する同期機能



競合状態ともいう

プロセスの同期と相互排除

- 排他制御は以下の機能を組み合わせて構成
  - 排除：共有資源のユーザ数を制限する。ユーザ数を「１」とする場合相互排除。
  - 協同：プロセス間通信により実現

クリティカルセクション

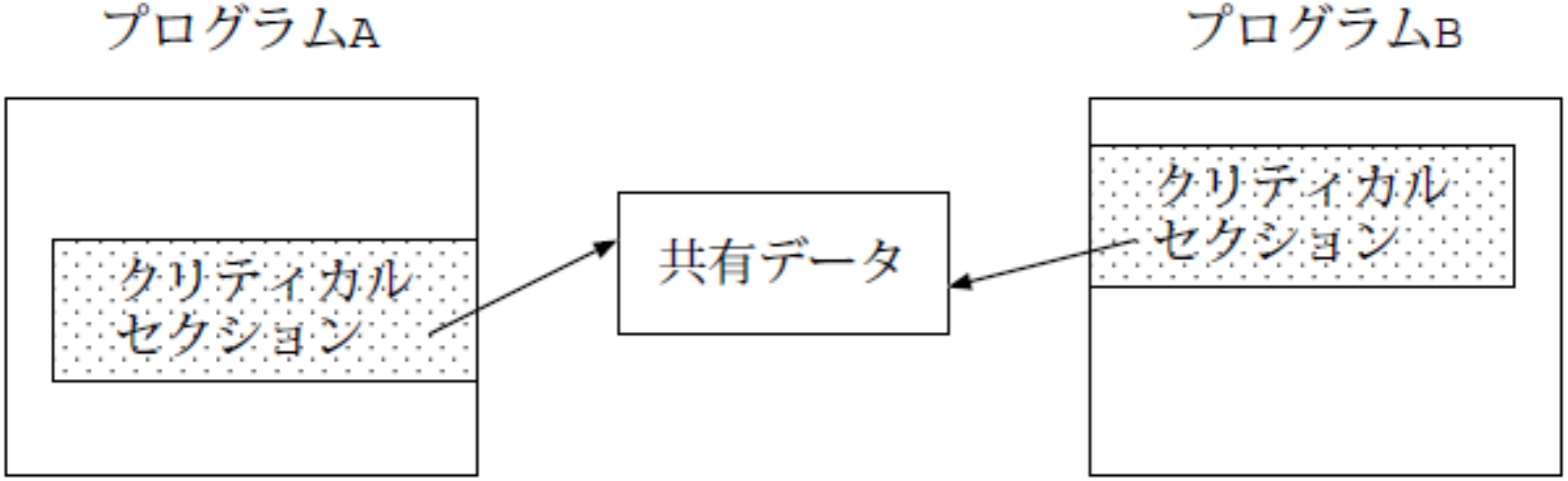
- 同時には1個のプロセスだけが使用できる共有資源を使用するプロセス部分
- 臨界領域、きわどい部分
- 割り込みなどで邪魔されたくない部分

クリティカルセクションと競合状態の回避

同時に2つのプロセスが、

クリティカルセクションに入らないように制御できれば、

競合状態を回避できる。



クリティカルセクションと競合状態の回避

共有データを使用して並列プロセスを正しく、

しかも効率良く協調させるための条件

1. 2つのプロセスがクリティカルセクションに同時に存在してはならない。
2. 速度やCPUの個数に仮定を設けてはならない。
3. クリティカルセクション以外で実行しているプロセスから、別のプロセスをブロックしてはならない。
4. クリティカルセクションに入るのを永遠に待つプロセスは、あってはならない。

- 競合状態とは、
  - ある共有データの読み書きを2つ以上のプロセスが行っており、実行結果がどのプロセスがどのタイミングで起動されるかに依存する状態

競合状態を含むプログラムは、予期しない動作をする。

↑

デバッグは容易ではない。

… 競合状態を回避するには、どうしたらよいだろうか？

競合状態を回避するには

２個以上のプロセスが、
同時に共有データを読み書きしなければよい。

排他制御 (mutual exclusion)

あるプロセスが共有変数を使っているとき、
他のプロセスは同じことを絶対にできない
ように排除する。

排他制御の方法

以下で、排他制御を行うための
いくつかの提案された方法を見てみる。

1. 割り込みの無効化
2. ロック変数
3. 厳密な交互実行
4. ~~Petersonの解決案~~
5. TSL命令 （参考書書（柴山潔（著））ページ128のTS命令と同じ）

割り込みの無効化

最も簡単な解決方法

プロセスごとに、クリティカルセクションに入った直後に、すべての割り込みを無効化する。

（クリティカルセクションを出るときに有効にする。）

↓

プロセス切り替えが起きない。

割り込みの無効化 – 問題点

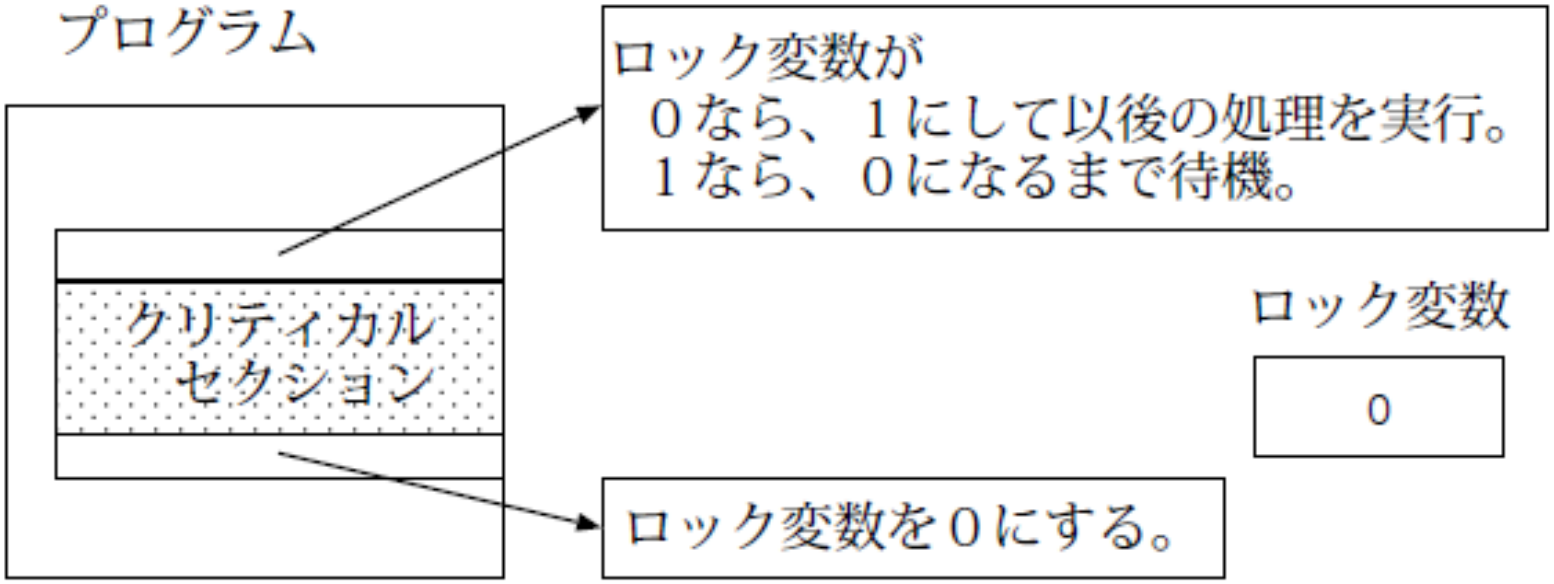
- ユーザプロセスに、大きな権限を与えることになる。
  もしも、ユーザプロセスの一つが、割り込み禁止をしたまま有効にしなかったら .... The End
- CPUが複数あった場合、
  禁止の命令を実行したCPUだけになる。
  他のCPUからの共有メモリのアクセスは禁止できない。

↓

カーネルが使うのには便利だが、ユーザプロセスには不適切。
ユーザプロセス向けの方法は ....

ロック変数

共有変数（ロック変数という）を使う。



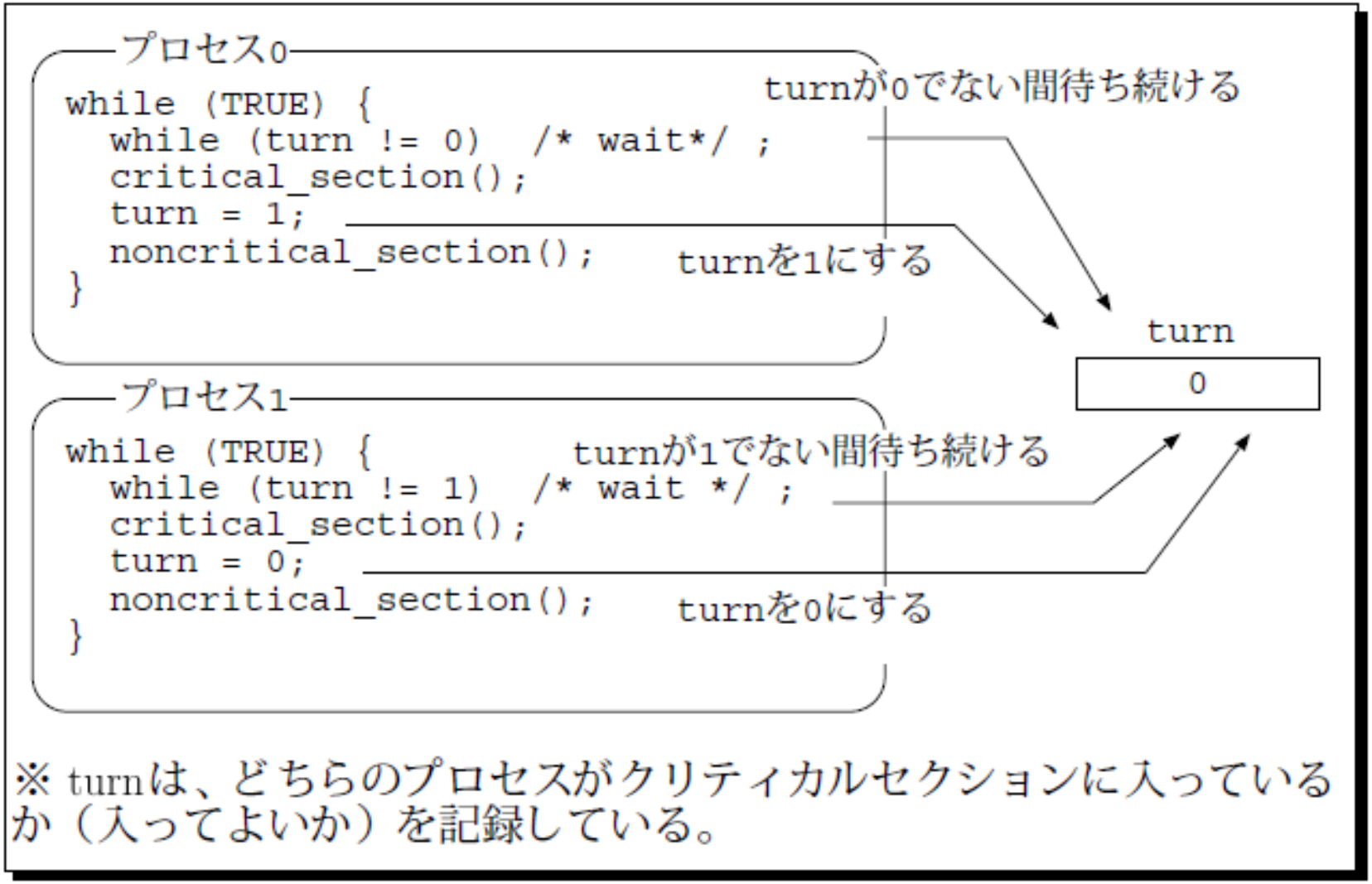
- ロック (lock): 最初に入った人が鍵をかける
- アンロック (unlock): 部屋を出るときに鍵をあけておく

これで、本当に うまくいくだろうか？

ロック変数の不具合

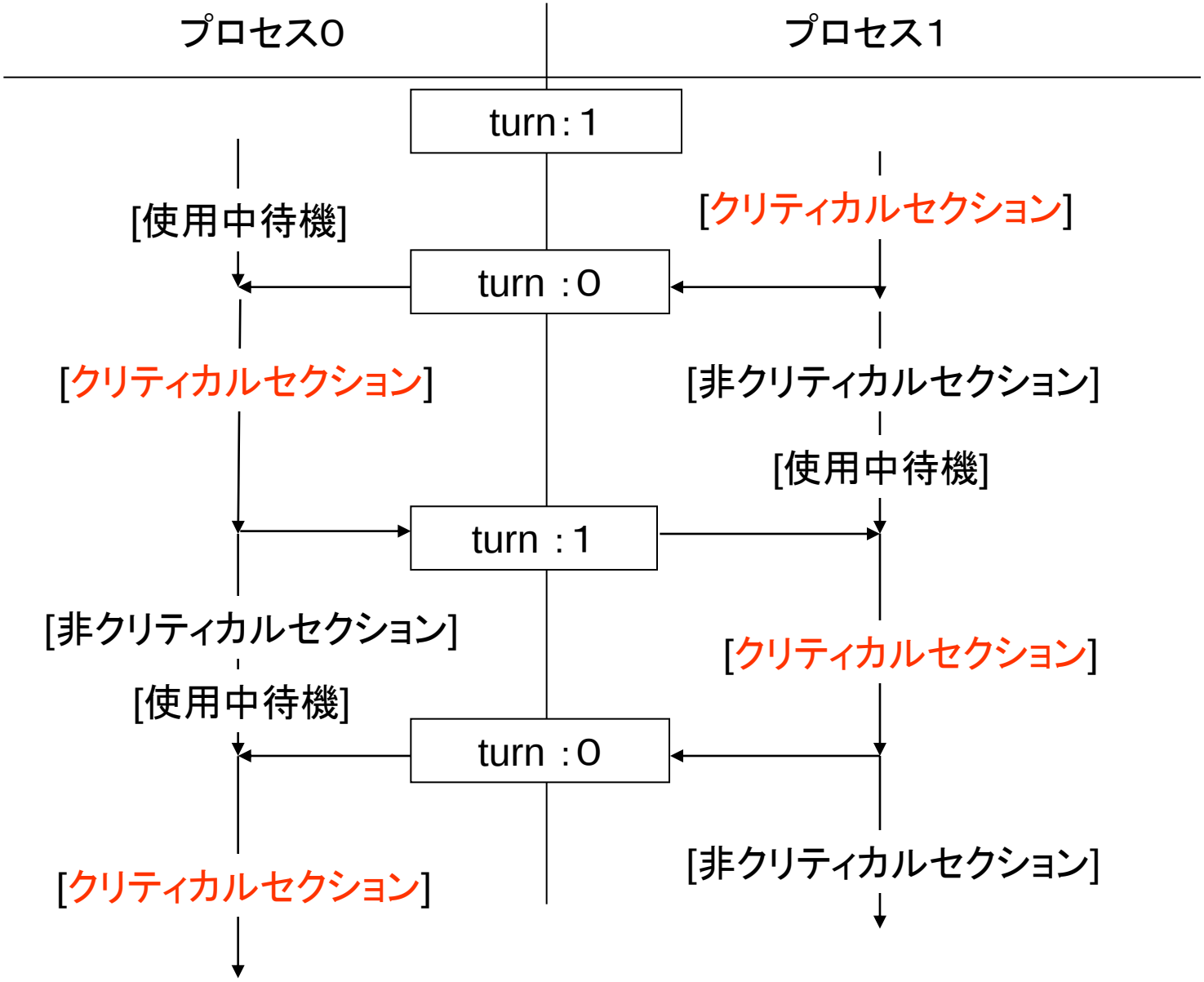
プロセスA	プロセスB	ロック変数
ロック変数を読む		0
<<プロセス切り替え>>→		
	ロック変数を読む	0
	ロック変数を1にセット	1
	クリティカルセクションに入る	
←<<プロセス切り替え>>		
ロック変数を1にする		1
クリティカルセクションに入る		

厳密な交互実行



※ turnは、どちらのプロセスがクリティカルセクションに入っているか（入ってよいか）を記録している。

厳密な交互実行の流れ（例の図示）



ビジーウェイト (busy wait) – 使用中待機

ある変数が、ある値になるまでテストしながら待つこと。

↑

待ち時間が短いことが**確実**でなければ、
使用しない。(「待つ」間もCPUを消費してしまう。)

厳密な交互実行の問題点

- ビジーウェイトを使用して、CPUを消費している。
- 一方のプロセスが片方よりも非常に遅い場合には、うまくいかない。（前述の競合状態回避の条件3に反する。）

  ※プロセス1の 非クリティカルセクションの処理が非常に遅い場合を考えてみよ。

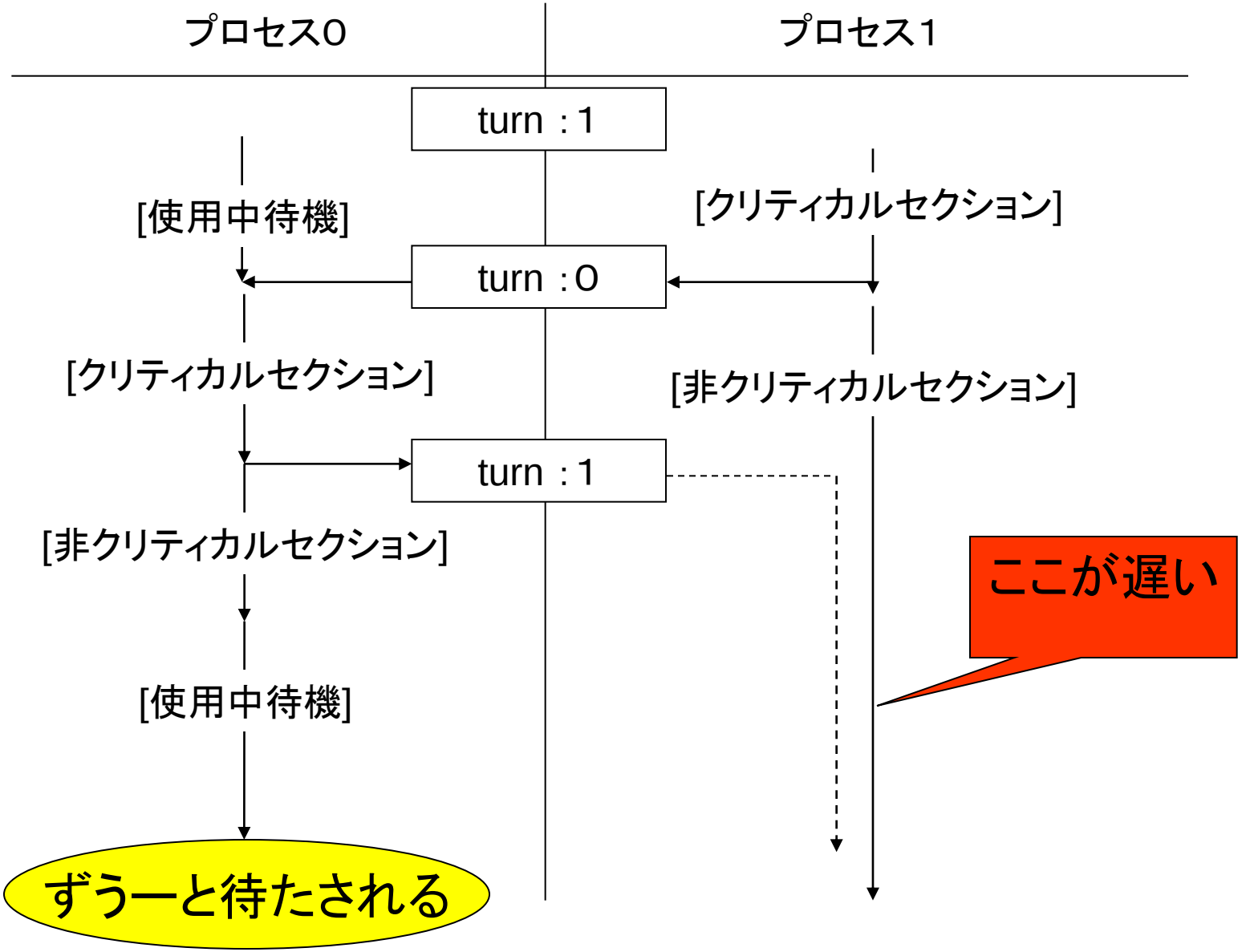
- 片方が連続実行したくても、必ず一方に制御を移さねばならない。

  ※プロセスのスケジュールが公平とは保証されていないので、うまくいかない。

このアルゴリズムは、競合状態は回避できるものの、効率上好ましくない。

条件３：クリティカルセクション以外で実行しているプロセスから、別のプロセスをブロックしてはならない。

厳密な交互実行の問題点（例の図示）



Test and **S**et **L**ock 命令(TSL命令)

多くのコンピュータ、

とくにマルチプロセッサを念頭に置いて設計されたコンピュータには、

次の動作をするTSL命令が用意されている。

1. あるアドレスのメモリのワードの内容を
   1つのレジスタに読み込み、
2. 非ゼロの値を そのアドレスのメモリに格納する。

読み込みと格納の操作は分割されないことが保証される。

つまり、この命令が終るまで、

そのワードをアクセスするプロセッサは他に無い。

TSL命令を使ったロック

架空の（だが典型的な）アセンブリ言語による方法

```
enter_region:
   tsl register,flag    | flagの内容をregisterへ読み取り、flagを1にセット
   cmp register,#0      | flagが0か判定する。
   jnz enter_region     | 0でなければ、ロックされているのでループする。
   ret                  | 呼び出し元に戻り、クリティカルセクションに入る。

 leave_region:
   mov flag,#0          | flagに0を格納。
   ret                  | 呼び出し元に戻る。
```

自由課題）上のアセンブリプログラムで、うまくいくことを確かめよ。

ビジーウェイト（busy-wait）を行うので、プロセッサ時間を消費し、それがオーバヘッドとなる。

不可分操作

- それ以上分割できない一連の操作機能を不可分操作またはアトミックトランザクション（Atomic Transaction）という
  - TS命令（TSL命令）
  - セマフォー（Semaphore）

プロセス間通信(**I**nter**P**rocess **C**ommunication)

もう一度、前の例を見てみよう。

cat *chapter*1 *chapter*2 *chapter*3 | grep *tree*

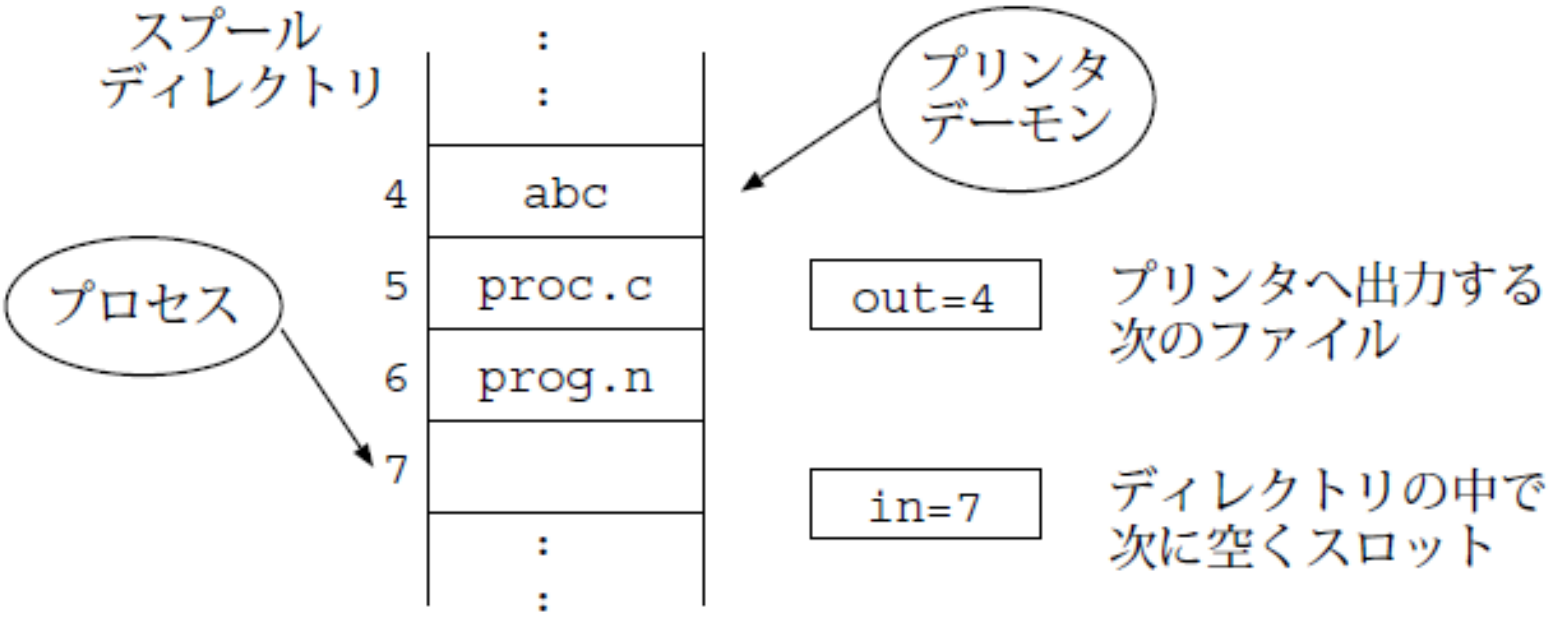
上は、プロセスからプロセスへデータを渡している。

うまくプロセス間で通信するには ...

以下で、プロセス間通信とそれに関係する問題について考える。

プロセス間通信の単純な例 – 出力スプーラ

プロセスが、ファイルを出力する。
スプール・ディレクトリ (spooler directory) を介して、
プリンタ・デーモン (printer daemon) にファイル名を知らせる。



複数のプロセスが、同時にディレクトリに書こうとしたら ....

出力スプーラにおける競合状態へのシナリオ

プロセスA	プロセスB
inから値を読み込み、次の空きスロットが7だと記録する。	
《割り込みが発生し、プロセスBに切り替わる》	
	inの値から、空きスロット7にファイル名を書き込む。 in を8に更新して、作業完了。
《プロセスAの実行が再開させられる》	
空きスロットは7のはずだと思い、スロット7にファイル名を書き込む。inを8に更新する。	→（プロセスBが、さっき書いたファイル名は消される!）

結局、プロセスBの要求したファイル出力は、行われない。

↑

どのプロセスがどのタイミングで起動されるか に依存する。

これが 競合状態(race condition)

セマフォによる同期操作

- セマフォ：腕木信号機
- P操作（DOWN操作ともいう）、V操作（UP操作ともいう）からなる
- P操作：資源を「使用」する不可分操作
  - セマフォ変数をデクリメント
- V操作：資源を「解放」する不可分操作
  - セマフォ変数をインクリメント

演習クイズ（第４回）

- 競合状態とは何か？

演習クイズ（第４回）

- 共有データを使用して並列プロセスを正しく、しかも効率よく協調させるための条件を４つ挙げよ

演習クイズ（第４回）

- 厳密な交互実行の問題点の例を図示せよ
  - 一方のプロセスが片方よりも非常に遅い場合には、うまくいかない。（前述の競合状態回避の条件に反する。）

    ※ プロセス１の非クリティカルセクションの処理が非常に遅い場合を考えてみよ。