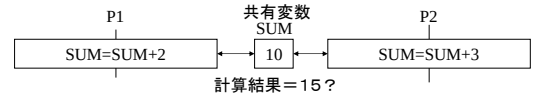
第3回　並行プロセス（2）

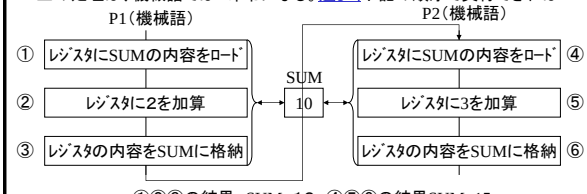
排他制御の原理
プロセス間通信

排他制御の必要性

P1とP2が変数SUMを共有し、それぞれ下記の命令を実行。SUMの値は？



上の処理は、機械語では3命令になる。運よく下記の順序で実行できれば

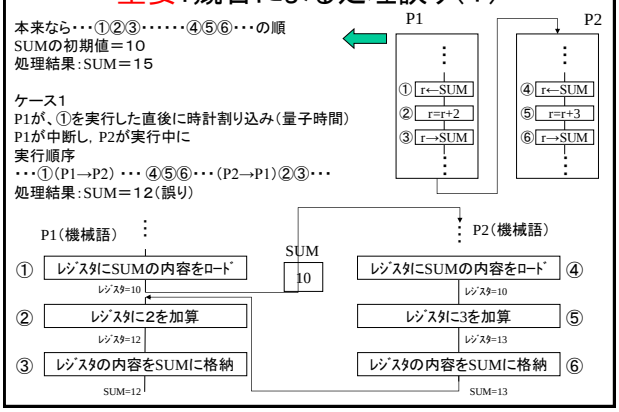


①②③の結果、SUM=12。④⑤⑥の結果SUM=15

重要：競合による処理誤り（1）

本来なら･･･①②③･･･････④⑤⑥･･･の順
SUMの初期値＝10
処理結果：SUM＝15

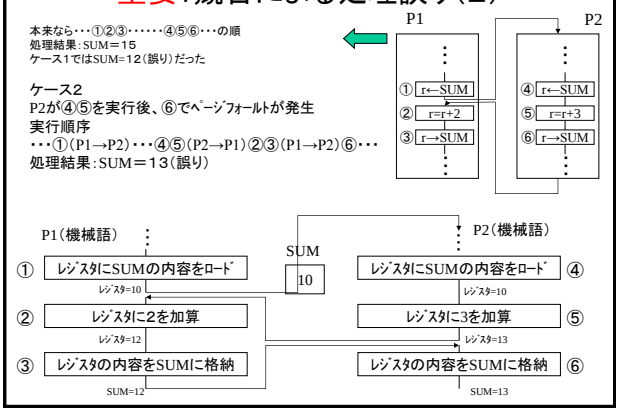
ケース1
P1が、①を実行した直後に時計割り込み（量子時間）
P1が中断し，P2が実行中に
実行順序
･･･①（P1→P2）･･･ ④⑤⑥･･･（P2→P1）②③･･･
処理結果：SUM＝12（誤り）



重要：競合による処理誤り（2）

本来なら･･･①②③･･･････④⑤⑥･･･の順
処理結果：SUM＝15
ケース1ではSUM=12（誤り）だった

ケース2
P2が④⑤を実行後、⑥でﾍﾟｰｼﾞﾌｫｰﾙﾄが発生
実行順序
･･･①（P1→P2）･･･④⑤（P2→P1）②③（P1→P2）⑥･･･
処理結果：SUM＝13（誤り）



実行結果

ケース1

	レジスタ	SUM
① r←SUM	10	10
割り込み		
レジスタ退避（10）、P2に切り替え		
･･･		
④ r←SUM	10	10
⑤ r=r+3	13	10
⑥ r→SUM	13	13
･･･ P2が終了		
P1が再度実行中になる		
（レジスタを復元）	10	13
② r=r+2	12	13
③ r→SUM	12	12
･･･ P1が終了		

レジスタの値はPCBに退避される

ケース2

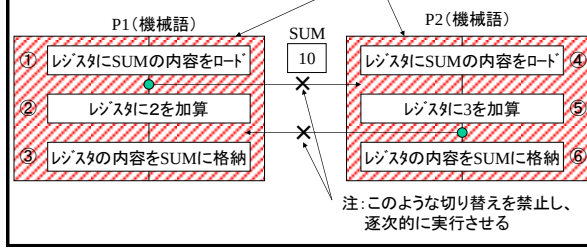
	レジスタ	SUM
① r←SUM	10	10
割り込み		
レジスタ退避（10）、P2に切り替え		
･･･		
④ r←SUM	10	10
⑤ r=r+3	13	10
割り込み		
レジスタ退避（13）、P1に切り替え		
P1が再度実行中になる		
（レジスタを復元）	10	10
② r=r+2	12	10
③ r→SUM	12	12
･･･ P1が終了		
P2が再度実行中になる		
（レジスタを復元）	13	12
⑥ r→SUM	13	13
･･･ P2が終了		

切り替えの順番や初期値などを変えて出題
SUMの値が何になるか、理解しておくこと

クリティカルセクション（危険区域）

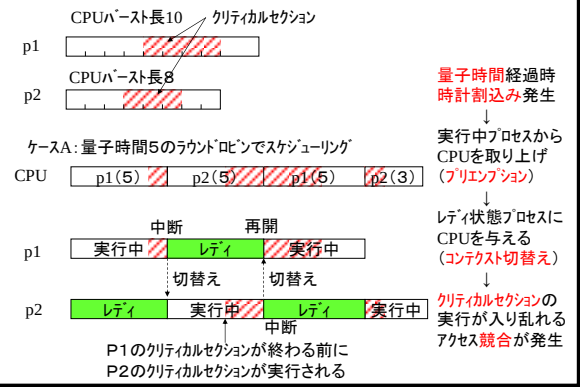
共有変数SUMへのアクセスが競合すると不正な結果となる
↑
（P1、P2の両方が使用する資源）

これを防止するために、①～③と④～⑥を排他的に実行する(注)必要がある
このように、排他的実行が必要な部分をクリティカルセクションという

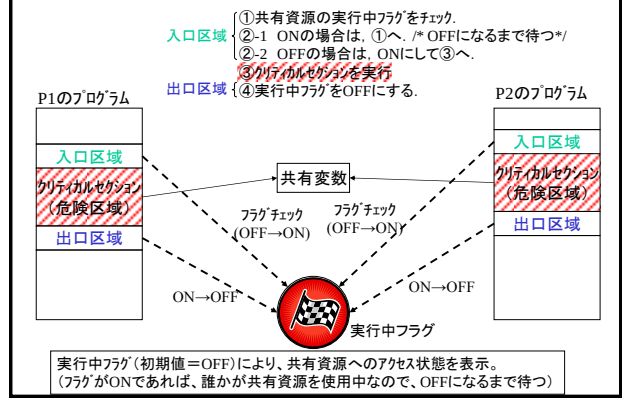


注：このような切り替えを禁止し、
逐次的に実行させる

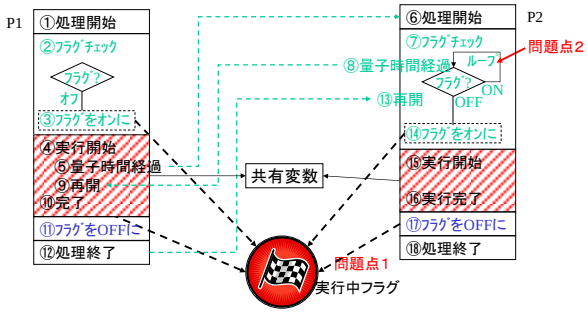
CPUスケジューリングによる競合



排他制御の基本的な考え方

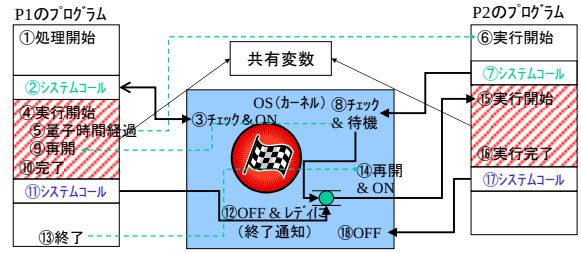


排他制御の基本的な考え方の問題点



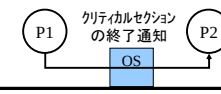
問題点1：フラグも共有変数（フラグON・OFF処理自体の排他制御が必要）
問題点2：⑤でP1が処理を中断し，P2が⑦のフラグチェックを繰り返しながらOFFになるのを待つ．
（P1が再開してクリティカルセクションの処理が終わらない限り，フラグはOFFにならない）
これを繁忙待機（busy wait：CPUを浪費しながら事象発生を待つ）と言う．

排他制御の実現方法とプロセス間通信

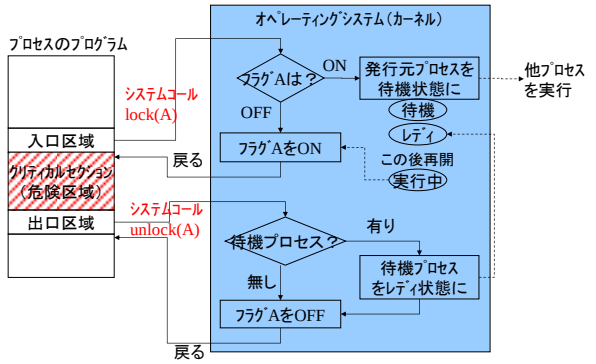


問題点1の解決法：システムコールにより，OSのみがフラグの操作をする②⑦⑪⑰
なお，OSがフラグを操作中は，割込み禁止（またはフラグのチェックと操作を一度に行う命令を使う）

問題点2の解決法⑧⑫
フラグONの場合，プロセス（P2）を待機状態にし，
P1のクリティカルセクション終了後にレディ状態にする
（右図のようなプロセス間通信とみなせる）



lock, unlockによる排他制御の原理



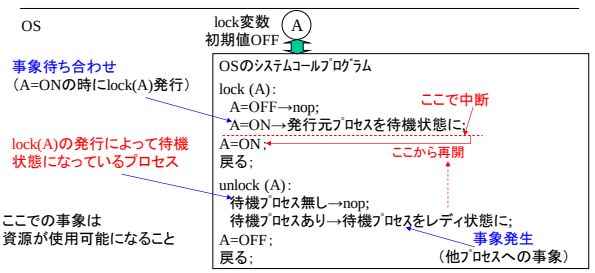
lock, unlockのプログラムと使用方法

P1のプログラム

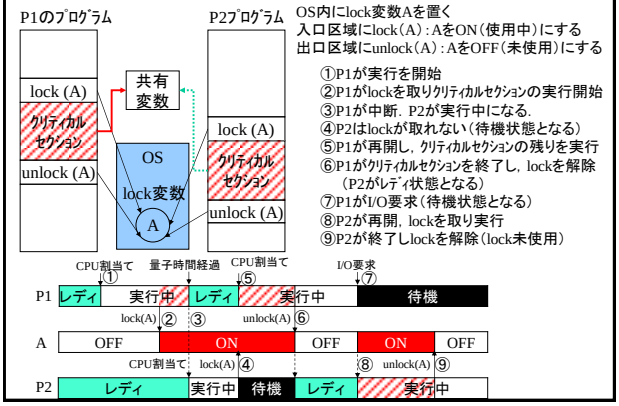
```
:
lock(A)          /*入口区域*/
SUM=SUM+2        /*クリティカルセクション*/
unlock(A)        /*出口区域*/
:
```

P2のプログラム

```
:
lock(A)          /*入口区域*/
SUM=SUM+3        /*クリティカルセクション*/
unlock(A)        /*出口区域*/
:
```



lock/unlockによる排他制御の動作例



OS内にlock変数Aを置く
入口区域にlock（A）：AをON（使用中）にする
出口区域にunlock（A）：AをOFF（未使用）にする

①P1が実行を開始
②P1がlockを取りｸﾘﾃｨｶﾙｾｸｼｮﾝの実行開始
③P1が中断．P2が実行中になる．
④P2はlockが取れない（待機状態となる）
⑤P1が再開し，ｸﾘﾃｨｶﾙｾｸｼｮﾝの残りを実行
⑥P1がｸﾘﾃｨｶﾙｾｸｼｮﾝを終了し，lockを解除
（P2がﾚﾃﾞｨ状態となる）
⑦P1がI/O要求（待機状態となる）
⑧P2が再開．lockを取り実行
⑨P2が終了しlockを解除（lock未使用）

セマフォ（semaphore）

ｼｽﾃﾑｺｰﾙ　P（S）、V（S）
ｾﾏﾌｫ変数Sの値＝使用可能な資源の数

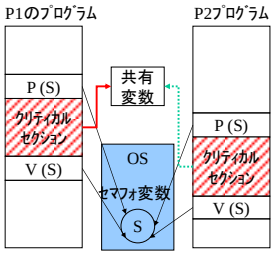
OSのｼｽﾃﾑｺｰﾙ（ｾﾏﾌｫ処理）ﾌﾟﾛｸﾞﾗﾑ

```
P(S)：
Sの値を1減らす；      ここで中断
 S≧0→nop；            /* 資源を1個使用
 S＜0→発行元ﾌﾟﾛｾｽを待機に；  /* 資源ありの場合
戻る；                 /* 資源無しの場合
                ここから再開
V(S)：
Sの値を1増やす；          /* 資源を1個返却
 S＞0→nop；               /* 待機ﾌﾟﾛｾｽ無し
 S≦0→待機ﾌﾟﾛｾｽをﾚﾃﾞｨに；  /* 待機ﾌﾟﾛｾｽ有り
戻る；
```

注：
S<0の場合，P(S)を発行したプロセスは，「戻る」前に実行が中断．相手プロセスがV(S)を発行し（レディ状態），さらにCPUが割り当てられて実行中になった後に戻ってくる．
（「戻る」から実行が再開され，発行元のプロセスに戻る）

Semaphore
鉄道などの腕木式信号機
（信号装置、または、手旗信号という意味）

セマフォによる排他制御



OS内にｾﾏﾌｫ変数S初期値nを置く

入口区域でP（S）を発行
出口区域でV（S）を発行
ｼｽﾃﾑｺｰﾙ

```
P(S)：
Sの値を1減らす；          ここで中断
 S≧0→nop；
 S＜0→発行元ﾌﾟﾛｾｽを待機に；(注1)
戻る；                    ここから再開

V(S)：
Sの値を1増やす；
 S＞0→nop；
 S≦0→待機ﾌﾟﾛｾｽをﾚﾃﾞｨに；(注2)
戻る；
```

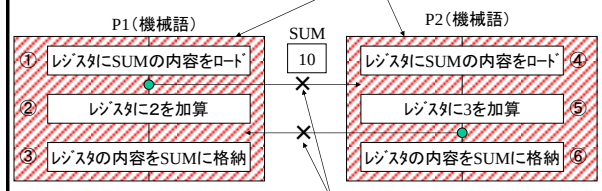
注1：S<0は、資源が無いことを示す。OSはｼｽﾃﾑｺｰﾙ発行元のﾌﾟﾛｾｽを待機状態にし，ｾﾏﾌｫ専用の待ち行列に並ばせる．
（状態遷移上は，事象待ち合わせ）
注2：S≦0は、待機状態ﾌﾟﾛｾｽが存在することを示す。OSは、そのﾌﾟﾛｾｽを再開させるために、ﾚﾃﾞｨ状態とし、実行待ち行列に並ばせる。（状態遷移上は，事象発生）

第2回のスライド

クリティカルセクション（危険区域）

共有変数SUMへのアクセスが競合すると不正な結果となる
↑
（P1、P2の両方が使用する資源）

これを防止するために、①～③と④～⑥を排他的に実行する(注)必要がある
このように、排他的実行が必要な部分をクリティカルセクションという



注：このような切り替えを禁止し、逐次的に実行させる

使用方法（1）

呼出し側でセマフォの変数名と初期値を決める
セマフォ変数 A　初期値=1 （関数呼び出しの実引数と同様に，変数名はSでなくても良い）
この例では，P(A), V(A)は，それぞれlock(A), unlock(A)と同じ働きをする

```
P1のプログラム
      ：実引数
P(A)          /*入口区域*/
SUM=SUM+2     /*ｸﾘﾃｨｶﾙｾｸｼｮﾝ*/
V(A)          /*出口区域*/
      ：
```

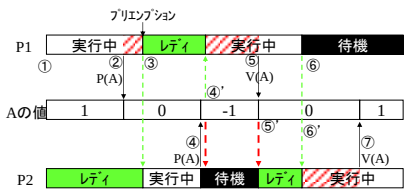
```
P2のプログラム
      ：
P(A)          /*入口区域*/
SUM=SUM+3     /*ｸﾘﾃｨｶﾙｾｸｼｮﾝ*/
V(A)          /*出口区域*/
      ：
```

A=1　⇔lock変数がOFF.
A≦0 ⇔lock変数がON.
（A=-1⇔P1またはP2の一方が待機状態）．

A　初期値1

```
OSのｼｽﾃﾑｺｰﾙﾌﾟﾛｸﾞﾗﾑ
P(S)：                          仮引数
   Sの値を1減らす;          V(S)：
     S≧0→nop;                 Sの値を1増やす;
     S＜0→発行元ﾌﾟﾛｾｽを待機状態に;    S＞0→nop;
   戻る;                        S≦0→待機ﾌﾟﾛｾｽをﾚﾃﾞｨに;
                                戻る;
```

CPUｽｹｼﾞｭｰﾘﾝｸﾞから見たｾﾏﾌｫ




```
P(S)：
Sの値を1減らす;
 S≧0→nop;
 S＜0→発行元ﾌﾟﾛｾｽを
      待機状態に;
戻る;
V(S)：
Sの値を1増やす;
 S＞0→nop;
 S≦0→待機ﾌﾟﾛｾｽを
      ﾚﾃﾞｨ状態に;
戻る;
```

①CPU割当てにより，P1が実行中に（P2はレディ状態）
②P1は入口区域でP(A)を発行．A=0となる．A≧0なのでP1に戻り，ｸﾘﾃｨｶﾙｾｸｼｮﾝ開始
③量子時間経過によりﾌﾟﾘｴﾝﾌﾟｼｮﾝが発生．OSはP1をﾚﾃﾞｨ状態にし、P2を実行中に
④P2は入口区域でP(A)を発行．A=-1となる．A<0なのでOSは発行元のP2を待機状態にする．
④' また，OSはP1にCPUを割当て実行中にする．P1はｸﾘﾃｨｶﾙｾｸｼｮﾝの残りを実行する．
⑤P1がｸﾘﾃｨｶﾙｾｸｼｮﾝを終了．出口区域でV(A)を発行し，A=0となる．
⑤' A≦0 なのでOSは、待機状態のP2をﾚﾃﾞｨ状態にし，発行元のP1に戻る．
⑥P1がI/O要求（入出力のｼｽﾃﾑｺｰﾙ発行）．CPUが空く．
⑥' OSはP2にCPUを割当て実行中にする．P2は再開して（P(S)から戻る）ｸﾘﾃｨｶﾙｾｸｼｮﾝ開始．
⑦P2がｸﾘﾃｨｶﾙｾｸｼｮﾝを終了し，出口区域でV(A)を発行．A=1となる．

事象の発生，状態遷移とｾﾏﾌｫ

④～⑥の考え方

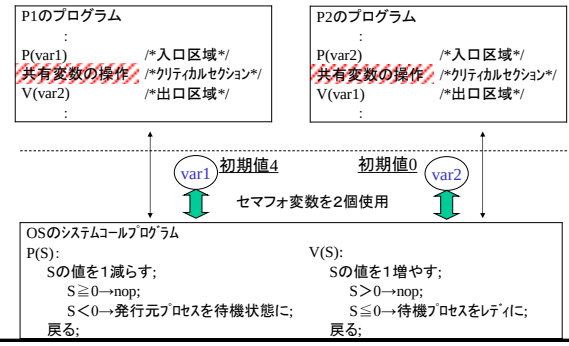
P1がｸﾘﾃｨｶﾙｾｸｼｮﾝの処理途中のため待ち合わせが必要この制御にｾﾏﾌｫSを使用

時刻	事象の発生（及びOSの処理）	P1	A	P2
①	P1にCPU割当て	実行中	1	レディ
②	P（A）発行（P1に戻る）	〃	0	〃
③	ﾌﾟﾘｴﾝﾌﾟｼｮﾝ，CPU割当て	レディ	0	実行中
④	P（A）発行(注1)（P2を待機状態に），CPU割当て	実行中	-1	待機
⑤	V（A）発行(注2)（P2をレディに，発行元に戻る）	〃	0	レディ
⑥	I/O要求，CPU割当て	待機	0	実行中
⑦	V（A）発行（P2に戻る）	〃	1	〃
	:			

注1：状態遷移上は，事象待ち合わせ　注2：状態遷移上は，事象発生

使用方法（2）

セマフォ変数 var1, var2　初期値var1=4, var2=0
（複数のｾﾏﾌｫ変数を使うことで，色々な制御が可能となる）



OSのｼｽﾃﾑｺｰﾙﾌﾟﾛｸﾞﾗﾑ

P(S):
　Sの値を1減らす;
　　S≧0→nop;
　　S＜0→発行元ﾌﾟﾛｾｽを待機状態に;
　戻る;

V(S):
　Sの値を1増やす;
　　S＞0→nop;
　　S≦0→待機ﾌﾟﾛｾｽをﾚﾃﾞｨに;
　戻る;