第4回　並行プロセス（3）

排他制御機構
同期問題

第3回のスライド

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　/* 資源を1個使用
　S≧0→nop；　/* 資源ありの場合
　S＜0→発行元ﾌﾟﾛｾｽを待機に；　/* 資源無しの場合
戻る；　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）

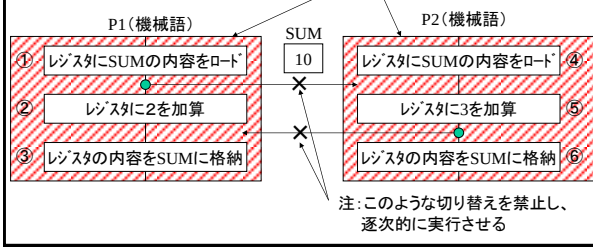
Semaphore
鉄道などの腕木式信号機
（信号装置、または、手旗信号という意味）

第3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

第3回のスライド

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
：
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）.

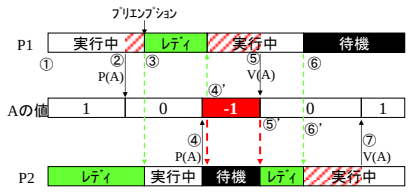
A　初期値1

OSのｼｽﾃﾑｺｰﾙﾌﾟﾛｸﾞﾗﾑ

P(S)：
　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;

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

CPUスケジューリングとセマフォ

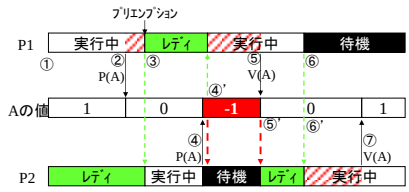


P(S)：
Sの値を1減らす;
　S≧0→nop;
　S＜0→発行元ﾌﾟﾛｾｽを待機状態に;
戻る;

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



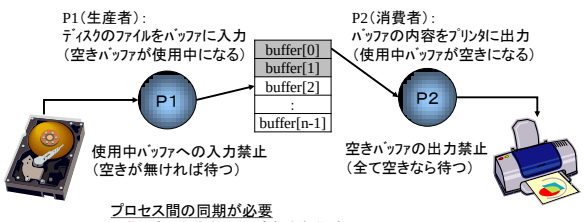
④～⑥の考え方
P2はこのように実行したいが
④　⑥
待ち
P1がｸﾘﾃｨｶﾙｾｸｼｮﾝの処理途中のため待ち合わせが必要
この制御にｾﾏﾌｫSを使用

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	[P1]P(A)（戻る；）	〃	0	〃
③	[P1]ﾌﾟﾘｴﾝﾌﾟｼｮﾝ，[P2]CPU割当て	レディ	〃	実行中
④	[P2]P(A)=事象待ち合わせ，[P1]CPU割当て	実行中	-1	待機
⑤	[P1]V(A)=[P2に対して]事象発生	〃	0	レディ
⑥	[P1]I/O要求，[P2]CPU割当て	待機	〃	実行中
⑦	[P2]V(A)（戻る；）	〃	1	〃
	：			

第2回のスライド

プロセス間の協調と同期（制約バッファ問題）

ディスクからファイルを読み込み，プリンタに出力．これを効率的に処理したい．
2つのプロセス（P1, P2）が複数のバッファを共有（共有資源）．これを用いた並行処理で実現．
バッファの数は有限（そのため，制約バッファ問題という）



プロセス間の同期が必要
同期：時間的な処理の流れを制御すること

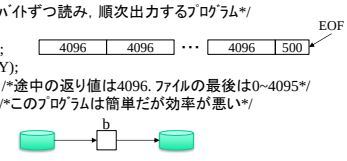
同期に必要な機能
・排他制御：共有資源を一度に1つのプロセスのみが使用するようにする
・プロセス間通信：プロセス間で情報の受け渡しをする

参考：Linuxの入出力システムコール

open(*pathname, flags): ファイルまたはデバイスの使用を開始し，ファイル識別子(fd)を返す．
*pathname：ファイルのpathname
flags：読み出し専用（O_RDONLY），書き込み専用（O_WRONLY）などの指定
close(fd): fdで指定されたファイルの使用を終了し，割り当てられたfd値を解放する．
（Unix系のOSは，プリンタやネットワークなどのデバイスもファイルとして扱う）

read(fd, *buf, count): fdで指定されたファイルから最大countバイトを変数bufに読み込む．
write(fd, *buf, count):変数bufからfdで指定されたファイルに最大countバイトを書き出す．
（どちらのシステムコールも返り値は，実際に読み書きされたバイト数）

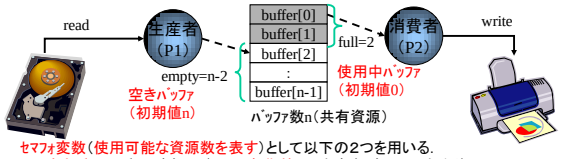
```
int fdi, fdo;          /*ファイルを4096バイトずつ読み，順次出力するプログラム*/
char b[4096];
fdi = open("in_file", O_RDONLY);
fdo = open("out_file", O_WRONLY);
while (4096 < read(fdi, b, 4096)) { /*途中の返り値は4096. ファイルの最後は0~4095*/
    write(fdo, b, 4096);          /*このプログラムは簡単だが効率が悪い*/
}
  close(fdi);
  close(fdo);
```



制約バッファ問題のプログラムは，複数バッファを使用し，並行プロセスにより，読み込みと書き込みを同時に行うので効率が良い．そのかわり，排他制御が必要になる．

制約バッファ問題（セマフォ変数）

生産者（P1）はディスクからデータを読み込み、バッファに格納する
消費者（P2）はバッファからデータを取り出し、プリンタに出力する

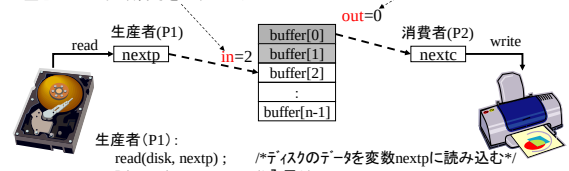


セマフォ変数（使用可能な資源数を表す）として以下の2つを用いる．
（1）空きバッファ（データが空のバッファ）（初期値n）：生産者が利用可能な資源
（2）使用中バッファ（データ入力済バッファ）（初期値0）：消費者が使用可能な資源

```
生産者（P1）：
    P(empty); /*入口*/   空きバッファ数を1つ減らす（無ければ自分を待機状態に）
    クリティカルセクション  空きバッファにデータを入れ，使用中バッファにする
    V(full);  /*出口*/   使用中バッファ数を1つ増やす（待機消費者をレディ状態に）
消費者（P2）：
    P(full);  /*入口*/   使用中バッファ数を1つ減らす（無ければ自分を待機状態に）
    クリティカルセクション  使用中バッファからデータを取り出し，空きバッファにする
    V(empty); /*出口*/   空きバッファ数を1つ増やす（待機生産者をレディ状態に）
```

制約バッファ問題（その他の変数）

ポインタ変数（in、out：初期値はどちらも0）
空きバッファの先頭番号を示すポインタ　　使用中バッファの先頭番号を示すポインタ




```
生産者（P1）：
    read(disk, nextp) ;     /*ディスクのデータを変数nextpに読み込む*/
    P(empty);               /*入口*/
    buffer[in]=nextp;       /*buffer[in]にnextpの内容を入れる*/
    in=(in+1)%n;            /*ポインタを次に更新*/
    V(full);                /*出口*/
消費者（P1）：
    P(full);                /*入口*/
    nextc=buffer[out];      /*buffer[out] のデータを変数nextcに取り出す*/
    out=(out+1)%n;          /*ポインタを次に更新*/
    V(empty);               /*出口*/
    write(printer, nextc);  /*変数nextcのデータをプリンタに印刷*/
```

%n：剰余
(in=n→0)

制約バッファ問題の実現プログラム

バッファ数n　buffer[0]～buffer[n-1]
先頭の空きバッファの番号　in=0　　先頭の使用中バッファの番号　out=0
セマフォ変数の初期値（使用可能な資源数）
使用中バッファ数　full=0　　空きバッファ数　empty=n
（注：セマフォ変数名は何でも良い．
　期末試験ではfull, emptyではなく，異なる変数名を使って出題．）

```
生産者：（以下を繰り返す）
(1)read(disk, nextp);
(2)P(empty);
(3)buffer[in]=nextp;
(4)in=(in+1) % n;
(5)V(full);
```

ディスクのデータを変数nextpに読み込む

```
消費者（以下を繰り返す）
(6)P(full);
(7)nextc = buffer[out];
(8)out=(out+1) % n;
(9)V(empty);
(10)write(printer, nextc);
```

変数nextcのデータをプリンタに印刷

```
P(S)：
  Sの値を1減らす;
  S≧0→nop;
  S＜0→発行元プ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→待機プロセスをレディ状態に;
  戻る;
```

各プロセス単独の動作

生産者プロセス：ディスクのデータを読み（readシステムコール），バッファに入れる処理を繰り返す．
ほとんどの時間はread中で待機状態．

(1)read(disk, nextp);
1回目の実行はreadのみ

```
(2)P(empty);
(3)buffer[in]=nextp;
(4)in=(in+1) % n;
(5)V(full);
(1)read(disk, nextp);
```

消費者プロセス：バッファのデータをプリンタに出力する（writeシステムコール）処理を繰り返す．
ほとんどの時間はwrite中で待機状態．

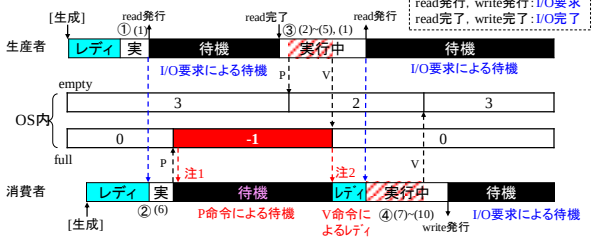

```
(6)P(full);
(7)nextc = buffer[out];
(8)out=(out+1) % n;
(9)V(empty);
(10)write (printer, nextc);
```

動作例（n=3）：生成直後

①生産者が実行．(1) readにより待機状態に
②消費者が実行．(6) P命令により待機状態に
③生産者が実行．(2)P命令によりempty=2
　(5)V命令によりfull=0．消費者をレディ状態に．
　(1)readにより待機状態に
④消費者が実行．(7)V命令によりempty=3
　(10)writeにより待機状態に

生産者	消費者
(1)read(disk, nextp)；	(6)P(full);
(2)P(empty);	(7)nextc = buffer[out];
(3)buffer[in]=nextp;	(8)out=(out+1) % n;
(4)in=(in+1) % n;	(9)V(empty);
(5)V(full)；	(10)write (printer, nextc);

read発行，write発行：I/O要求
read完了，write完了：I/O完了



注1：消費者自身が発行したP命令によりfull<0となったので、待機状態にされる
注2：生産者が発行したV命令によりfull=0となったので、消費者がレディ状態にされる

動作例（n=3）：生成直後の状態遷移

バッファ数n=3，セマフォ変数　初期値empty=3（空きバッファ数），full=0（使用中バッファ数）

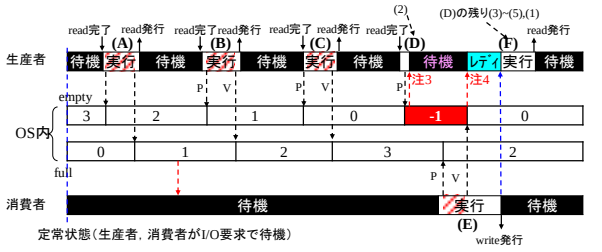
①生産者：CPU割当て（レディ→実行中），read命令=I/O要求（実行中→待機）
②消費者：CPU割当て（レディ→実行中），P命令でfull=-1<0（実行中→待機）
③生産者：read完了=I/O完了，CPU割当て（待機→レディ→実行中），P命令でempty=2，
　V命令でfull=0（消費者が待機→レディ），read命令=I/O要求（実行中→待機）
④消費者：CPU割当て（レディ→実行中），V命令でempty=3，write命令=I/O要求（実行中→待機）

実行サイクル	事象の発生	生産者	empty	full	消費者
	[生成]	レディ	3	0	レディ
①生産者	CPU割当て	実行中	〃	〃	〃
	read(disk, nextp)=I/O要求，	待機	〃	〃	〃
②消費者	CPU割当て	〃	〃	〃	実行中
	P(full)=事象待ち合わせ	〃	〃	-1	待機
③生産者	read完了=I/O完了，CPU割当て	実行中	〃	〃	〃
	P(empty)	〃	2	〃	〃
	V(full)=（消費者に対して）事象発生	〃	〃	0	レディ
	read(disk, nextp)=I/O要求，	待機	〃	〃	〃
④消費者	CPU割当て	〃	〃	〃	実行中
	V(empty)	〃	3	〃	〃
	write(printer, nextc)=I/O要求	〃	〃	〃	待機

動作例（n=3）：定常状態以後

前スライドに示した実行によって，生産者，消費者は待機状態，セマフォ変数は，empty=3, full=0.
この後，(A)生産者，(B)生産者，(C)生産者，(D)生産者，(E)消費者，(F)生産者の順で実行．

生産者	消費者
(1)read(disk, nextp)；	(6)P(full);
(2)P(empty);	(7)nextc = buffer[out];
(3)buffer[in]=nextp;	(8)out=(out+1) % n;
(4)in=(in+1) % n;	(9)V(empty);
(5)V(full)；	(10)write (printer, nextc);

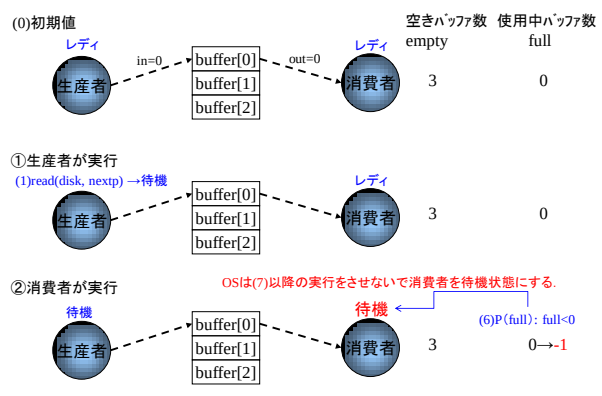


注3：生産者自身が発行したP命令によりempty<0となったので、待機状態にされる
注4：消費者が発行したV命令によりempty=0となったので、生産者がレディ状態にされる

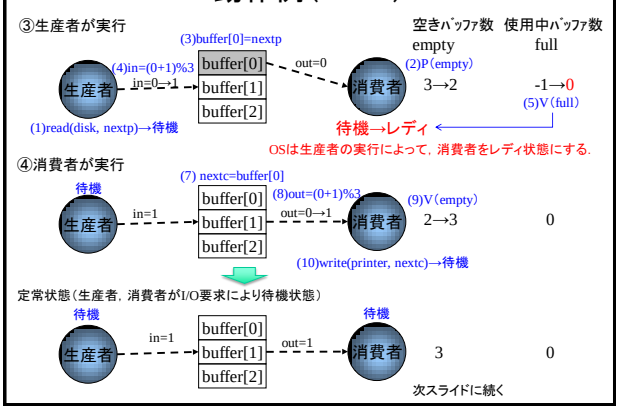
動作例（n=3）：定常状態以後の状態遷移

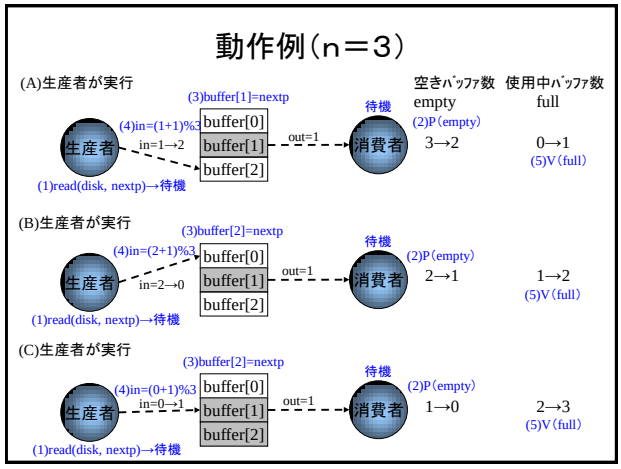
実行サイクル	事象の発生	生産者	empty	full	消費者
	定常状態：前スライドの続き	待機	3	0	待機
(A)生産者	read完了=I/O完了，CPU割当て	実行中	〃	〃	〃
	P(empty)	〃	2	〃	〃
	V(full)	〃	〃	1	〃
	read(disk, nextp)=I/O要求	待機	〃	〃	〃
(B)生産者	read完了=I/O完了，CPU割当て	実行中	〃	〃	〃
	P(empty)	〃	1	〃	〃
	V(full)	〃	〃	2	〃
	read(disk, nextp)=I/O要求	待機	〃	〃	〃
(C)生産者	read完了=I/O完了，CPU割当て	実行中	〃	〃	〃
	P(empty)	〃	0	〃	〃
	V(full)	〃	〃	3	〃
	read(disk, nextp)=I/O要求	待機	〃	〃	〃
(D)生産者	read完了=I/O完了，CPU割当て	実行中	〃	〃	〃
	P(empty)=事象待ち合わせ	待機	-1	〃	〃
(E)消費者	write完了=I/O完了，CPU割当て	〃	〃	〃	実行中
	P(full)	〃	〃	2	〃
	V(empty)=（生産者に対して）事象発生	レディ	0	〃	〃
	write(printer, nextc)=I/O要求，	〃	〃	〃	待機
(H)生産者	CPU割当て	実行中	〃	〃	〃
(F)の残り	V(full)	〃	〃	3	〃
	read(disk, nextp)=I/O要求	待機	〃	〃	〃

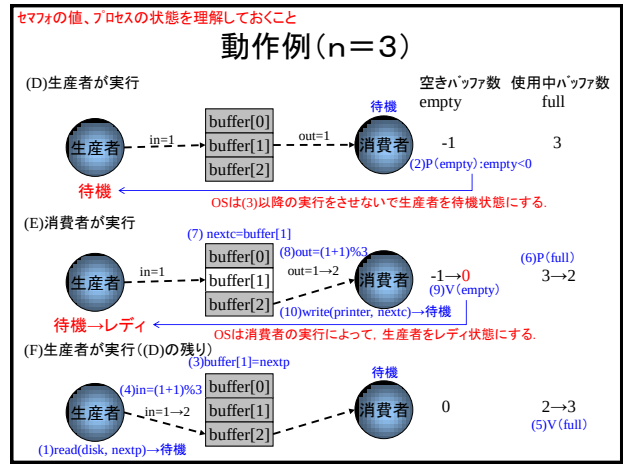
動作例（n＝3）



動作例（n＝3）




動作例（n＝3）
(A)生産者が実行
空きﾊﾞｯﾌｧ数 empty
使用中ﾊﾞｯﾌｧ数 full
(3)buffer[1]=nextp
(4)in=(1+1)%3
in=1→2
buffer[0]
buffer[1]
buffer[2]
out=1
待機
(2)P（empty）
3→2
0→1
(5)V（full）
(1)read(disk, nextp)→待機
(B)生産者が実行
(3)buffer[2]=nextp
(4)in=(2+1)%3
in=2→0
(2)P（empty）
2→1
1→2
(5)V（full）
(1)read(disk, nextp)→待機
(C)生産者が実行
(3)buffer[2]=nextp
(4)in=(0+1)%3
in=0→1
(2)P（empty）
1→0
2→3
(5)V（full）
(1)read(disk, nextp)→待機


ｾﾏﾌｫの値、ﾌﾟﾛｾｽの状態を理解しておくこと
動作例（n＝3）
(D)生産者が実行
空きﾊﾞｯﾌｧ数 empty
使用中ﾊﾞｯﾌｧ数 full
in=1
buffer[0]
buffer[1]
buffer[2]
out=1
待機
-1
3
(2)P（empty）:empty<0
待機
OSは(3)以降の実行をさせないで生産者を待機状態にする.
(E)消費者が実行
(7) nextc=buffer[1]
in=1
(8)out=(1+1)%3
out=1→2
-1→0
(9)V（empty）
(6)P（full）
3→2
(10)write(printer, nextc)→待機
待機→レディ
OSは消費者の実行によって，生産者をレディ状態にする.
(F)生産者が実行（(D)の残り）
(3)buffer[1]=nextp
(4)in=(1+1)%3
in=1→2
0
2→3
(5)V（full）
(1)read(disk, nextp)→待機