Operating Systems

第9回 プロセスの同期とプロセス間通信

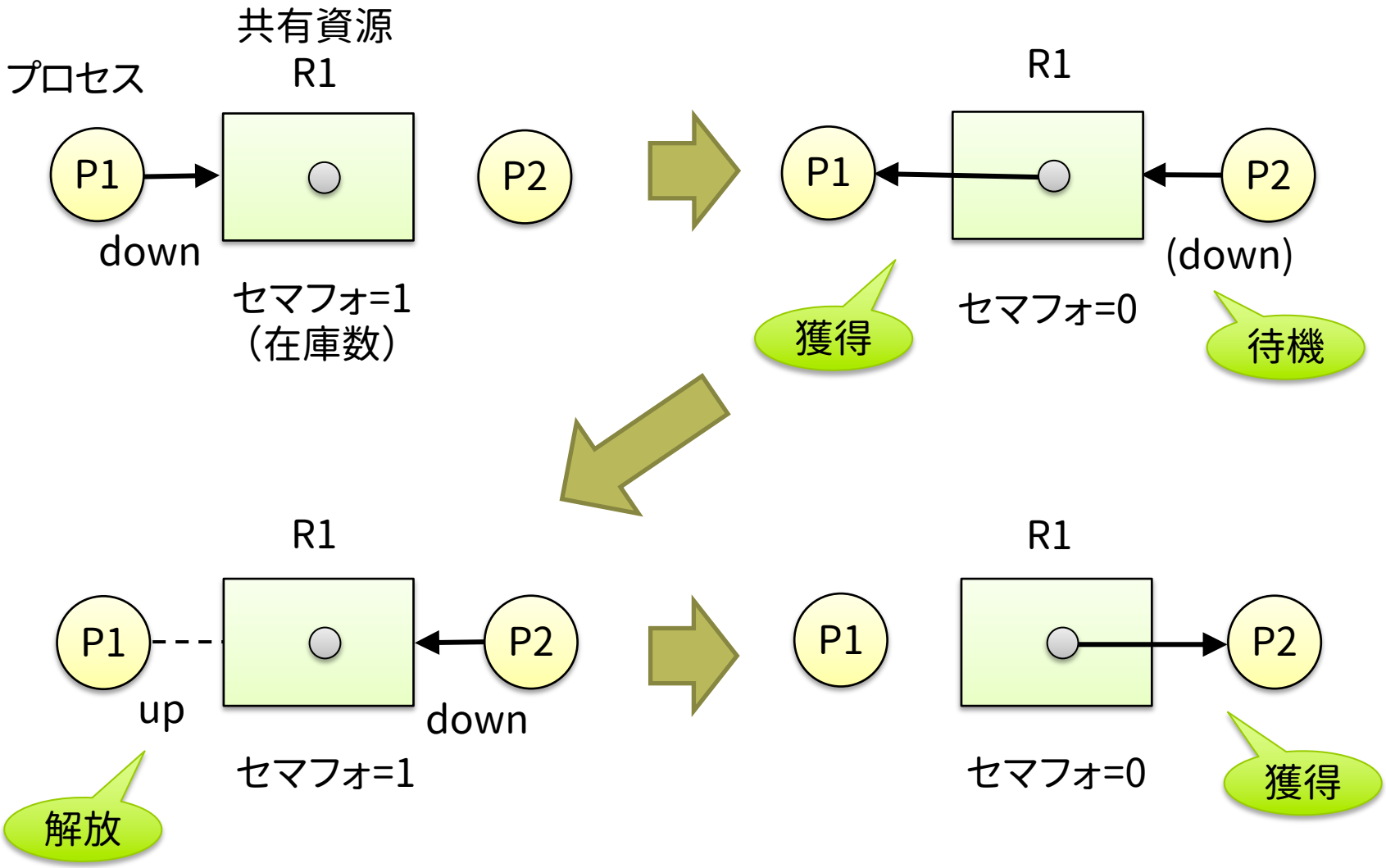
プロセスの同期

- 同期(synchronization)とは?
  - プロセス同士が実行の"タイミングを調整"すること
  - 同期を実現するためには,プロセス同士が連絡するしくみが必要
- 同期が必要な場面
  - 複数のプロセスが,同じ資源にアクセスする場合(排他制御)
  - プロセス同士が連携していて,お互いの進行を確認・待機する場合
  - あるプロセスから別のプロセスへ,何かを伝える必要がある場合
- 同期機能の分類
  - 排他制御 ← 前回からの続き
  - 事象(イベント)の連絡・待ち合わせ
  - プロセス間通信 ← 通信を利用して同期にも使える

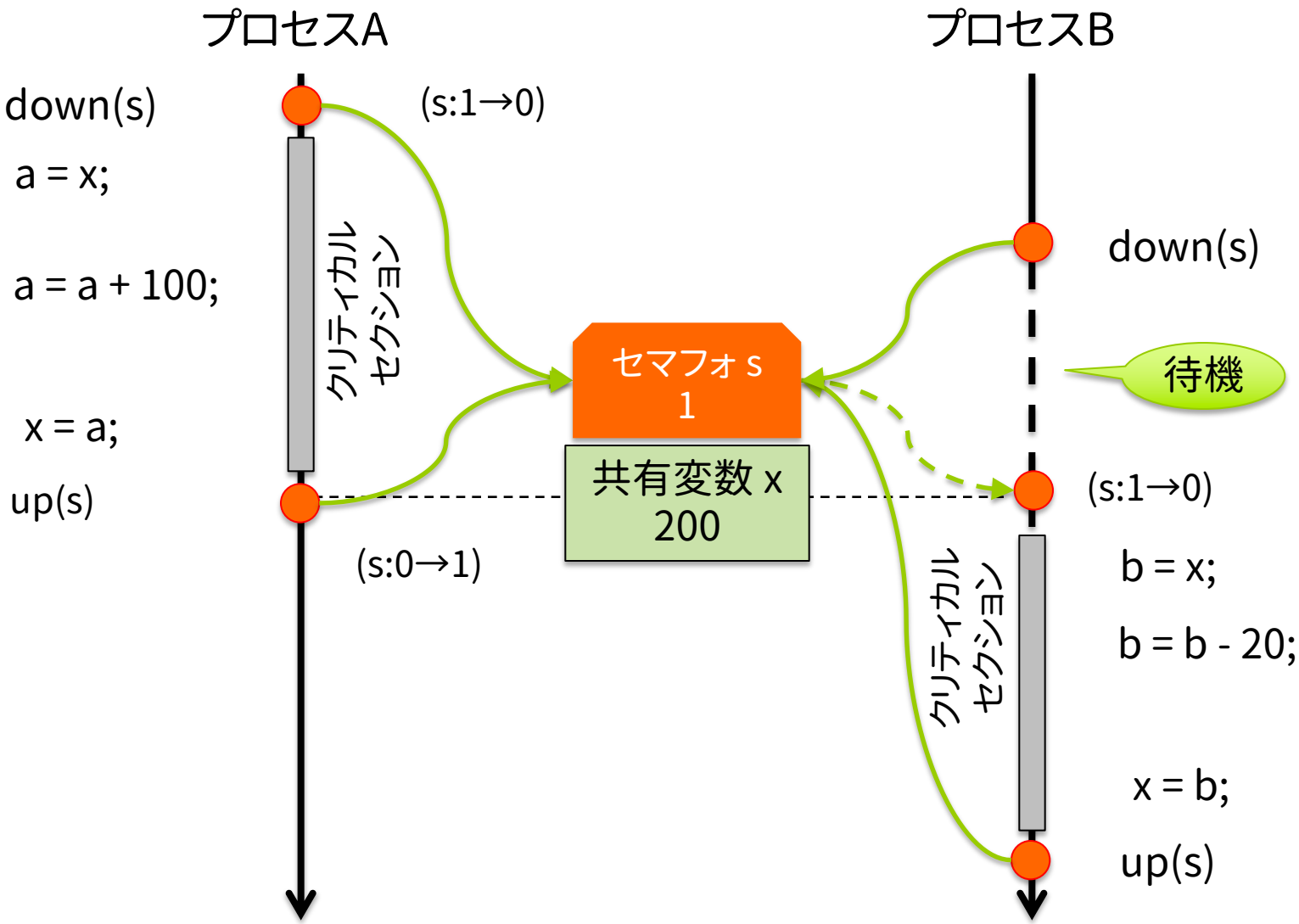
セマフォによる排他制御

- セマフォ(semaphore)
  - 一種の整数カウンタ(言葉の意味は昔の鉄道の腕木信号)
  - 資源の残り数(在庫数)をカウンタの値にする
  - down操作(信号↓)とup操作(信号↑)を必ず対応させる
- 資源の獲得=down操作(別名:P操作,wait操作)
  - プロセスは資源を使う前に,セマフォ(在庫数)の値を1減らす
  - セマフォが0だったら,資源に残りがないのでそこで待機(休眠)する(OSが自動的にプロセスを休眠状態にする)
- 資源の解放=up操作(別名:V操作,signal操作)
  - プロセスは資源を使い終わったら,セマフォ(在庫数)の値を1増やす
  - セマフォがプラスになったら,OSは待機中のプロセスを目覚めさせる

セマフォの概念


共有資源
R1
プロセス
P1
down
P2
セマフォ=1
（在庫数）
R1
P1
P2
(down)
獲得
セマフォ=0
待機
R1
P1
up
P2
down
セマフォ=1
解放
R1
P1
P2
セマフォ=0
獲得

セマフォの例


プロセスA
プロセスB
down(s)
a = x;
a = a + 100;
x = a;
up(s)
(s:1→0)
クリティカル
セクション
セマフォ s
1
共有変数 x
200
(s:0→1)
down(s)
待機
(s:1→0)
b = x;
b = b - 20;
x = b;
up(s)
クリティカル
セクション

セマフォの特徴

- セマフォによる排他制御の利点
  - 資源(リソース)の空きを待っているプロセスは休眠状態になるので,CPUで無意味に実行されるプロセスがなく,効率的である
  - 資源(リソース)に対応するセマフォを作っておけば,あらゆる資源の排他制御ができるので,汎用的である

- セマフォでは解決できない問題
  - 1つのプロセスが複数のセマフォを同時に待つことはできない
    ⇒「どれでもいいから空いた資源から使う」ということはできない
  - プログラマーが適切にdownとupを書かなければならない
    ⇒ 対応ミスを犯しやすい(エラー・例外処理でのupを忘れやすい)

その他の排他制御方法

- ミューテックス
  - Mutex = mutual exclusion（相互排除）
  - 基本的には，最大値1（0 or 1）のセマフォ（バイナリセマフォ）
  - 先着順ではなく，プロセスの優先順位を考慮するものが多い
- モニタ
  - プログラミング言語が排他制御機能を備える（オブジェクト指向的）
  - 内部にセマフォ等を持ち，メソッドやブロック単位で排他制御する
  - 例）Java の synchronized

```
void method() {
    synchronized(共有資源) {
        // クリティカルセクション
    }
}
```

```
synchronized void method() {

    // thisを共有資源とする
    // クリティカルセクション
}
```

事象の連絡・待ち合わせ

- 事象（イベント）の連絡
  - あるプロセスから他のプロセスへ，何か起きたことを知らせること
  - 共有資源に関係しない同期処理にも用いられる
- ポーリングによる方法
  - 知らされる側が，一定の時間間隔で状態（共有メモリ等）を調べる
- セマフォによる方法
  - 事象が発生したかどうか（事象の発生回数）を，セマフォの値にする
  - 事象待ち＝知らされる側が，down操作（カウンタダウン）をして待つ
  - 事象発生＝知らせる側が，up操作（カウンタアップ）をする
- プロセス間通信による方法
  - 通信（メッセージ送信）によって連絡する

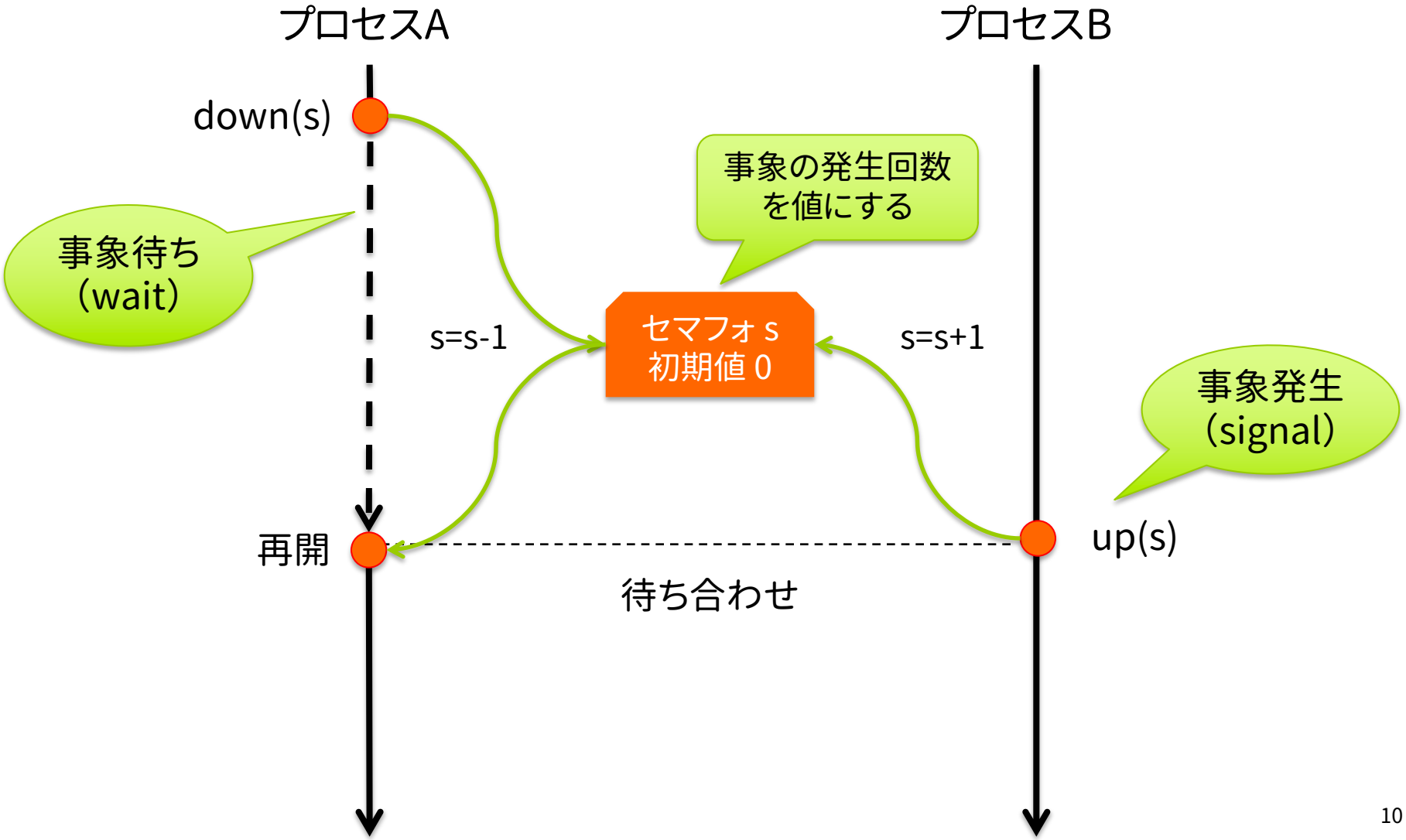
ポーリング

- ポーリングとは
  - 資源等に対して，定期的に状態を問い合わせる（チェックする）方法
  - 基本的な考えはスピンロックと同様だが，一定の時間間隔をあける
  - 主に，高速な応答が必要でないデバイス等の処理に用いられる

```
/* 単純なポーリングの例 */
while (1)  {
    if (資源Aの獲得が成功か?) {
        資源Aを利用した処理;
        資源Aを解放する;
    }
    if (資源Bの獲得が成功か?) {
        資源Bを利用した処理;
        資源Bを解放する;
    }
    一定時間待つ;
}
```

```
/* ポーリングによる事象の連絡 */
while (1) {
    if (事象が発生したか?) {

        事象に関する処理;

        事象をリセット;
    }

    しばらく休憩する;

}
```

セマフォによる事象の連絡



プロセス間通信

- プロセス間通信
  - 並行プロセス間の通信機能
  - 送信 send　⇔　受信 receive

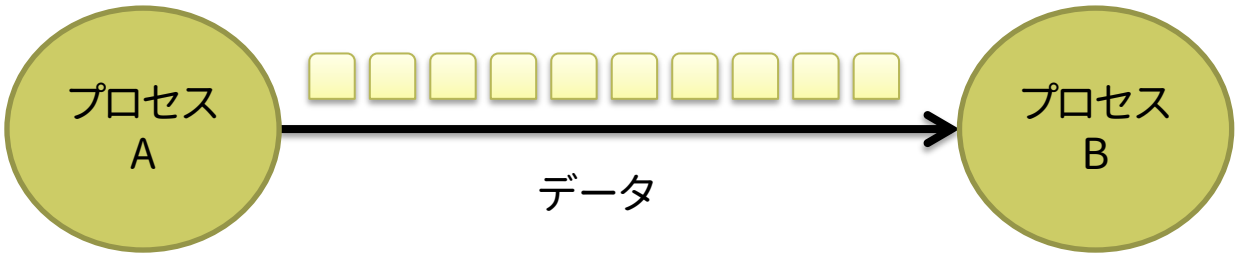
- 同期型（ブロッキング型）
  - 送受信の両者が，通信路を通して直接メッセージを交換する
  - 受信側がメッセージを受け取るまで，送信側は次の処理に進めない
  - 例）待ち合わせ方式，電話方式，（インターネットTCP）

- 非同期型（ノンブロッキング型）
  - 送信側は，メッセージをOSに預け，そのまま処理を続行する
  - 受信側は，好きなときにOSからメッセージを取り出す
  - 例）手紙方式，電子メール方式，（インターネットUDP）

代表的な通信方式

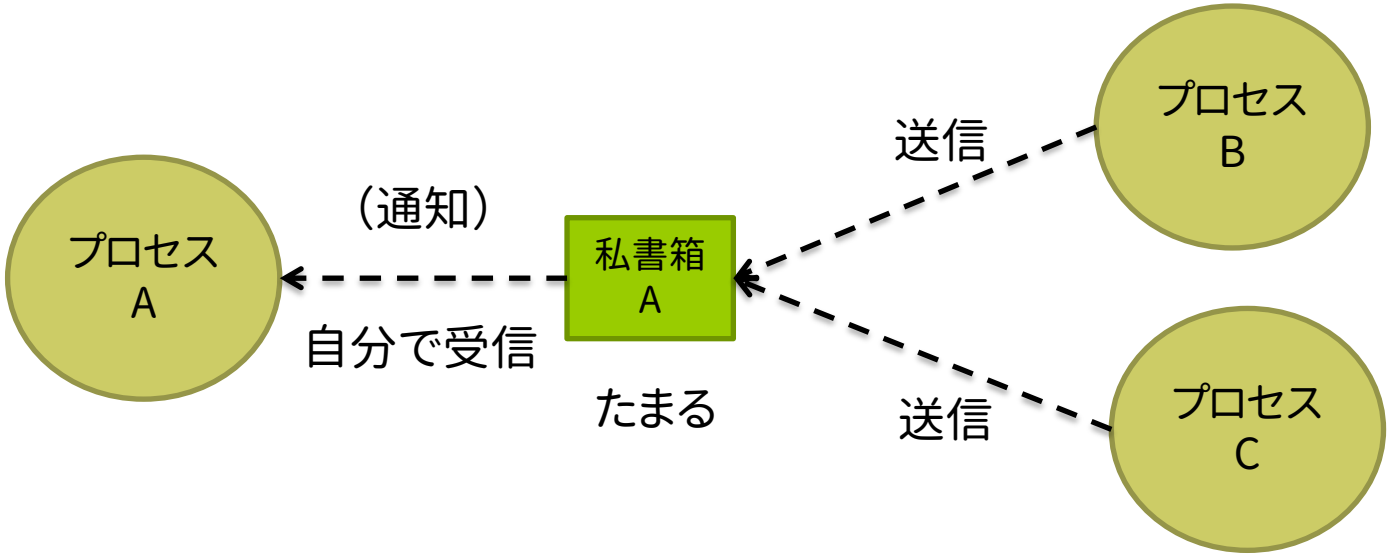
- 共有メモリ(共有変数)
  - メモリ領域を複数のプロセスが共有する
  - アクセス管理には,セマフォなどOSの排他制御機能を利用する

- パイプライン / データキュー
  - 2つのプロセスの入出力を直接的に接続する(同期型)
  - 区切りのないバイト列が,一方から他方に直接流れる



代表的な通信方式

- メールボックス / メッセージバッファ
  - “名前つきの箱”に届いたメッセージが順序どおりに並ぶ（非同期型）
  - OSが送受信を管理するので，各プロセスでは排他制御が要らない



- 遠隔手続き呼び出し（RPC）
  - 他のプロセスが持つ関数（手続き）を実行させて，結果を受信する
  - プログラム上では，通常の関数呼び出しとほぼ同様に記述できる

通信相手の指定方式

- 直接指名方式
  - プロセスIDなどで，通信相手のプロセスを直接指定する
  - 特定のプロセスに対して送信するので，通信経路には名前は不要
  - 例）RPC（remote procedure call 遠隔手続き呼び出し）など
- 大域名方式
  - 受信プロセスが，“名前つきの通信経路”（窓口/私書箱）を作成する
  - 名前を公開することにより，全プロセスからの通信を受け付けられる
  - 例）メールボックス，名前つきパイプ，ソケットなど
- （無名方式）
  - 親子のプロセス間または子プロセス同士なら，子プロセスの生成時に通信路を確保しておけば，通信時に相手を指定する必要がない
  - 例）パイプ，（ある種の）共有メモリなど

演習課題

- 課題 9a HOSにおけるセマフォを使った排他制御
  - この課題の狙いは，排他制御の必要性と排他制御のためのセマフォの使い方を，実際のプログラミングを通して学ぶことである。
  - HOSでは，セマフォのdown操作は wai_sem(セマフォID)，up操作は sig_sem(セマフォID) のサービスコールを用いる。
- 手順
  - win-semaphore のプログラムをコンパイル・実行して画面表示を示し，ソースコードを読んで動作について考察せよ。
  - 次に共有資源 shared にアクセスしているクリティカルセクションを wai_sem(SEMID) と sig_sem(SEMID) ではさむことで，適切な排他制御を実現せよ（ソースコードの変更点を明示せよ）。
  - 表示結果を確認し，セマフォの動作と役割について考察せよ。
  - どこからどこまでがクリティカルセクションになるか（いつからいつまで資源を独占する必要があるか），よく意味を考えること。

演習課題

- 課題 9b HOSにおけるデッドロックと排他制御
  - この課題の狙いは，デッドロックという現象を実際のプログラムで実感した上で，それが起きないようにする対策を学ぶことである。
- 手順
  - win-deadlock のプログラムは共有資源を2つ使用している。これを実行すると，最後まで完了せずに途中で停止してしまうことがある。
  - なぜそのようなことが起きるのか原因をなるべく詳しく考察せよ。
  - このようなことを防止するため，第8回のデッドロックの防止策1～3のどれかに対応する対策を組み込み，実行結果を示して考察せよ。
  - 【注意】資源に関する処理の全体を wai_sem と sig_sem ではさみ，原則として pol_sem や twai_sem は使わないこと。これらを単独で使うと排他制御をせずにクリティカルセクションに入ってしまう。
  - 【注意】delay()は作業（遊び）の時間をシミュレーションしているものなので，無効にしたり，移動したり，変更したりしないこと。

デッドロックの防止

- 防止策1「待ち条件の防止」
  - 例）各プロセスは，処理に必要な全資源を一度に要求するようにする
  - つまり，ある資源を確保したまま他の資源を待たないようにする
- 防止策2「横取り不可能条件の防止」
  - 例）プロセスが資源の獲得に失敗した場合は，そこで処理を中断して保持している資源をすべて手放す（いったん他のプロセスに譲る）
  - つまり，資源を使い続けたくても，途中で使用を止めて解放する
- 防止策3「循環待ち条件の防止」
  - 例）資源に順番（番号）をつけ，確保するときはその順番を守る
- 以上は，デッドロックの発生条件2・3・4にそれぞれ対応し，どれか1つでも実現できればデッドロックを防止できる

HOS（μITRON）の同期機能

サービスコール	意味	説明
cre_sem	create semaphore	セマフォの生成
del_sem	delete semaphore	セマフォの削除
sig_sem	signal semaphore	セマフォのup操作
wai_sem	wait samaphore	セマフォのdown操作
pol_sem	poll semaphore	同上（獲得失敗で待機しない）
twai_sem		同上（待ち時間の制限つき）
ref_sem	refer semaphore	セマフォの状態参照
cre_flg	create event flag	イベントフラグの生成
del_flg	delete event flag	イベントフラグの削除
set_flg	set event flag	イベントフラグのセット
clr_flg	clear event flag	インベントフラグのクリア
wai_flg	wait event flag	イベントフラグ待ち

HOS（μITRON）の通信機能

サービスコール	意味	説明
cre_dtq	create data queue	データキューの生成
del_dtq	delete data queue	データキューの削除
snd_dtq	send data queue	データキューへの送信
psnd_dtq		同上（獲得失敗で待機しない）
tsnd_dtq		同上（待ち時間の制限つき）
rcv_dtq	receive data queue	データキューからの受信
cre_mbf	create message buffer	メッセージバッファの生成
del_mbf	delete message buffer	メッセージバッファの削除
snd_mbf	send message buffer	メッセージバッファへの送信
psnd_mbf		同上（獲得失敗で待機しない）
tsnd_mbf		同上（待ち時間の制限つき）
rcv_mbf	receive message buffer	メッセージバッファからの受信