# 15 UNIXプロセス間通信(2)
## —System V系のIPC—

1983年にAT&Tがリリースした System V には、
プロセス間通信のために次の3つの機構が用意された。

- 共有メモリ,
- セマフォア,
- メッセージキュー

これらをまとめてSystem V(系)のIPCと言う。

Inter Process Communication

- 3つのIPCは、どれも次の様な手順で利用する。

  ① 使用の宣言：
  通信相手のプロセス間で約束した値をkeyとして与えて、IPC機構の利用を申し出る。

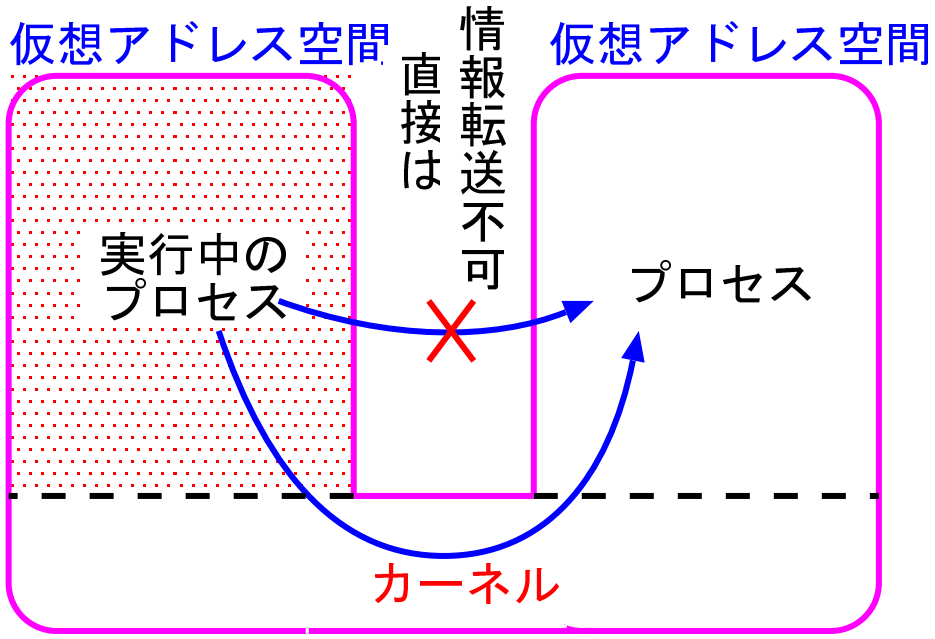
  ② 識別子を獲得：
  ステップ①のシステムコールの返り値として識別子を獲得する。

  ③ 各種操作：
  ステップ②で獲得した識別子を使って色々な操作をする。

  - IPC操作 ··· プロセス固有のアドレス空間への共有メモリセグメントの組み込みとその解除、セマフォア操作、メッセージの送受信。
  - 制御操作 ··· 共有メモリセグメントに関する情報、セマフォア値、あるいはメッセージキューの属性、の調査と変更。

- System Vの IPCを実装するためのセグメントは、パイプと同じ様にカーネル内で管理される。

# 15–1 共有メモリ




- 通常、仮想アドレス空間はプロセス毎に用意される。
- 2つのプロセス間で情報の転送を行おうとすると、両プロセスへのアクセス権限があるカーネルに依頼せざるを得ない。

しかし、

◇ カーネルに依頼するとシステムコール割り込みを伴い、CPUの負担も増える。

◇ 同じ情報をシステム内に2重に保持するという無駄も生じる。

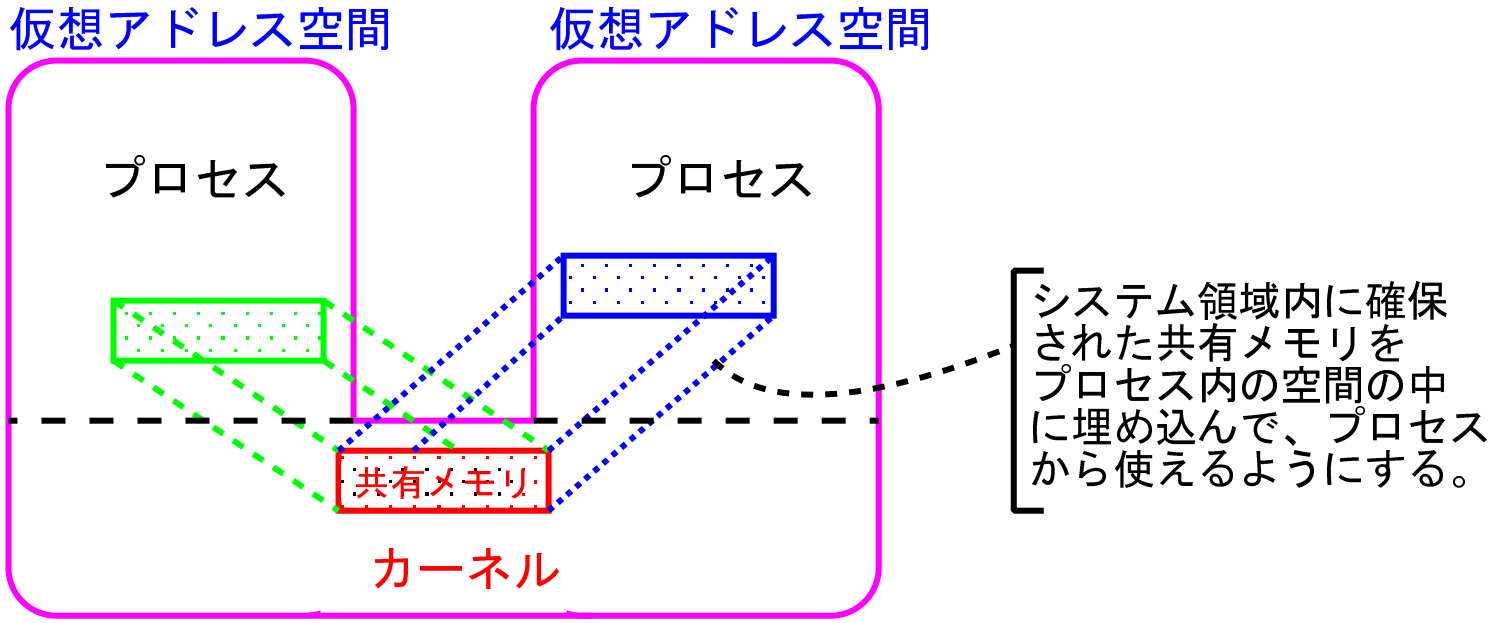
⇨ 複数のプロセス間で1つのメモリ領域を共有する機構が考案された。

⇨ 複数のプロセス間で1つのメモリ領域を共有する機構が考案された。

共有メモリ：

- 共有するメモリとしてはカーネルが管理する領域が使われる。
- 仮想記憶の技術を使って、共有するメモリ領域は各々のプロセスのアドレス空間内に組み込まれる。

ページングの場合、　ページのマップ先をカーネルが管理する領域に変えるだけ。



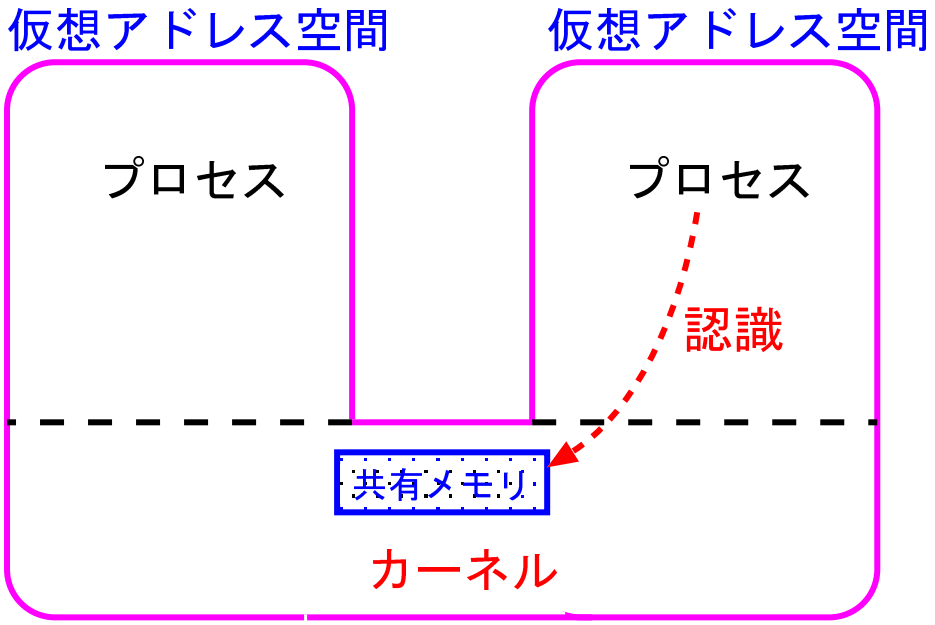

- 応用例　：　銀行オンラインシステム、座席予約システム、...

# 共有メモリセグメントのIDを取得するためのシステムコール

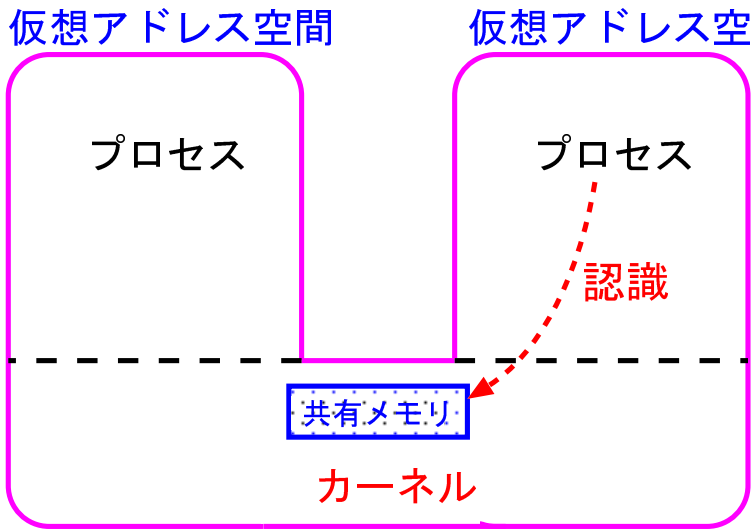
`int shmget(key_t key, int size, int flags);`

- ヘッダファイル<sys/types.h>, <sys/ipc.h> と <sys/shm.h> を必要とする。
- 引数の指定に従ってカーネルの領域内に共有メモリセグメントが生成されるか、指定に合った既存の共有メモリセグメントが探し出される。

成功すると 共有メモリID(正整数) が返され、失敗すると $-1$ が返される。



```
int shmget(key_t key, int size, int flags);
```




- 関数引数の key には、メモリを共有しようとするプロセス間で共通の数字名を指定する。

  IPC_PRIVATE を指定すると、...

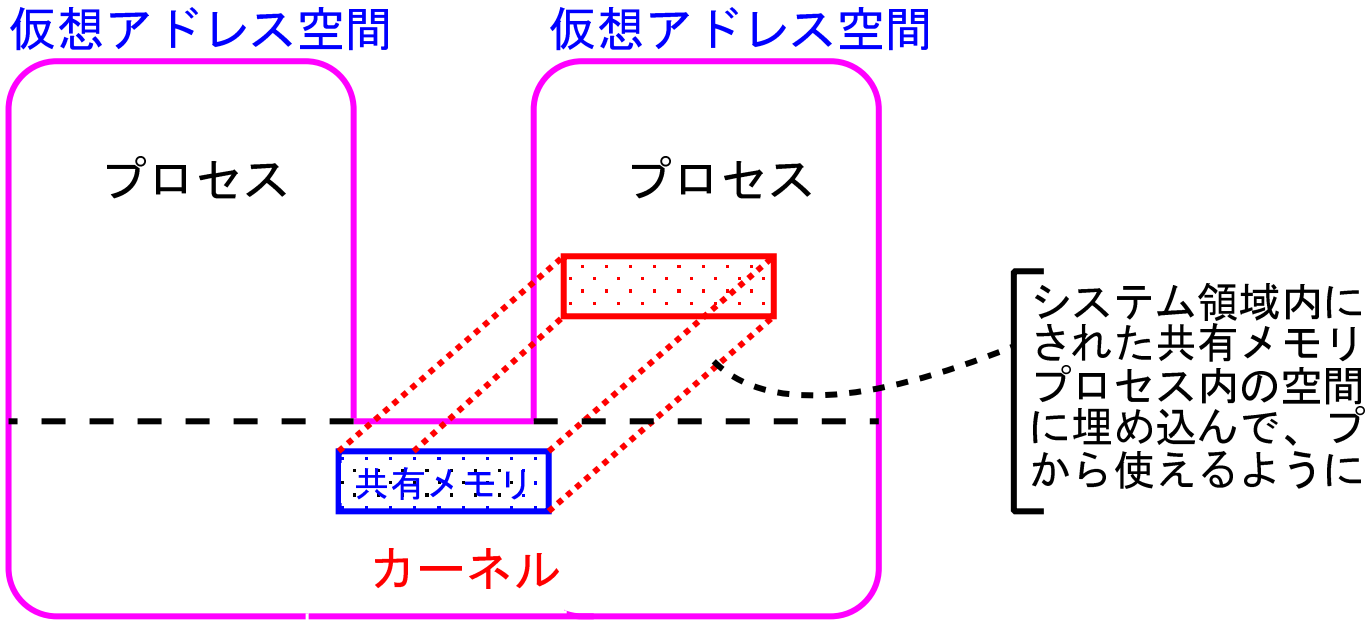
- 関数引数の size は、...
- 関数引数の flags はこのセグメントに対するアクセス許可を指定するために使われる。

  ファイルの保護モードと同様に8進表示で指定することが出来る。

# 共有メモリセグメントを自アドレス空間に組み込むシステムコール

## char *shmat(int shmid, char *shmaddr, int shmflg) :

- ヘッダファイル<sys/types.h>, <sys/ipc.h> と <sys/shm.h> を必要とする。
- 引数の指定に従って共有メモリセグメントが自プロセスのアドレス空間の中に組み込まれる (**attach** される)。




- 成功すると（仮想アドレス空間内での）共有メモリセグメントの先頭番地 が返され、失敗すると（char *）-1 が返される。

- 共有メモリセグメントを自アドレス空間に組み込んだ後、その領域をどう使うかはユーザに任されていて、
  その中の変数領域を使う前に、
  それらの番地を記憶した変数を変数領域毎に用意する
  などの準備が必要である。

---

```
char *shmat(int shmid, char *shmaddr, int shmflg);
```

- 関数引数の shmid には、...
- 関数引数の shmaddr には
  組み込み先の希望のアドレスを指定することも出来るが、
  一般にはNULLを指定してシステムに組み込み先を決めてもらう。
- 関数引数の shmflg は、共有メモリセグメントを読み出し専用にするかどうかのフラグを表す。

  SHM_RDONLY を指定すれば読み出し専用になり、それ以外 (普通は0) を指定すれば読み出しと書き込みの両方が可能である。

# 共有メモリセグメントを自アドレス空間から切り離す システムコール　int shmdt(char *shmaddr) :

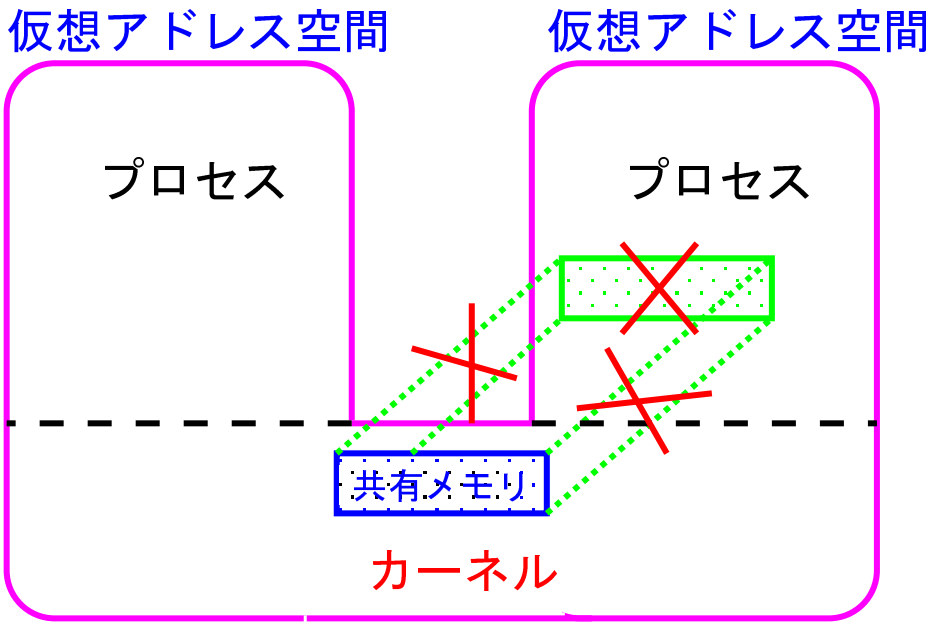
- <sys/types.h>, <sys/ipc.h> と <sys/shm.h> を必要とする。

- 引数で指定された番地にある共有メモリセグメントが自プロセスのアドレス空間から切り離される(**detach** される)。

  成功すると 0 が返され、失敗すると $-1$ が返される。

- この関数を実行した後も、共有メモリセグメントはカーネル内に残ったままになっている。

  共有メモリセグメントを削除するには、shmctl() システムコールまたはipcrmコマンドを用いる。




注意 :
カーネル内に確保されている共有メモリの一覧を表示するには　ipcs -m　とする。

# 共有メモリセグメントの状態を制御するシステムコール

int shmctl(int shmid, int cmd, struct shmid_ds *buf) :

- <sys/types.h>, <sys/ipc.h> と <sys/shm.h> を必要とする。
- 関数引数の shmid には、...
- 関数引数の cmd は制御内容を表す定数で、IPC_STAT, IPC_SET, または IPC_RMID のいずれかを指定する。
- 引数の buf の指す構造体型 struct shmid_ds は ...
- shmctl(shmid,IPC_STAT,buf) が呼ばれると、
  IDがshmidの共有メモリセグメントの情報(e.g.大きさ, セグメントを作成したプロセスのID, .....) がbufの指す構造体に格納される。
- shmctl(shmid,IPC_SET,buf) が呼ばれると、
  IDがshmidの共有メモリセグメントの情報がbufの指す構造体に記録された通りに変更される。
- shmctl(shmid,IPC_RMID,buf) が呼ばれると、
  IDがshmidの共有メモリセグメントが除去される。
- どの場合も、成功すると 0 が返され、...

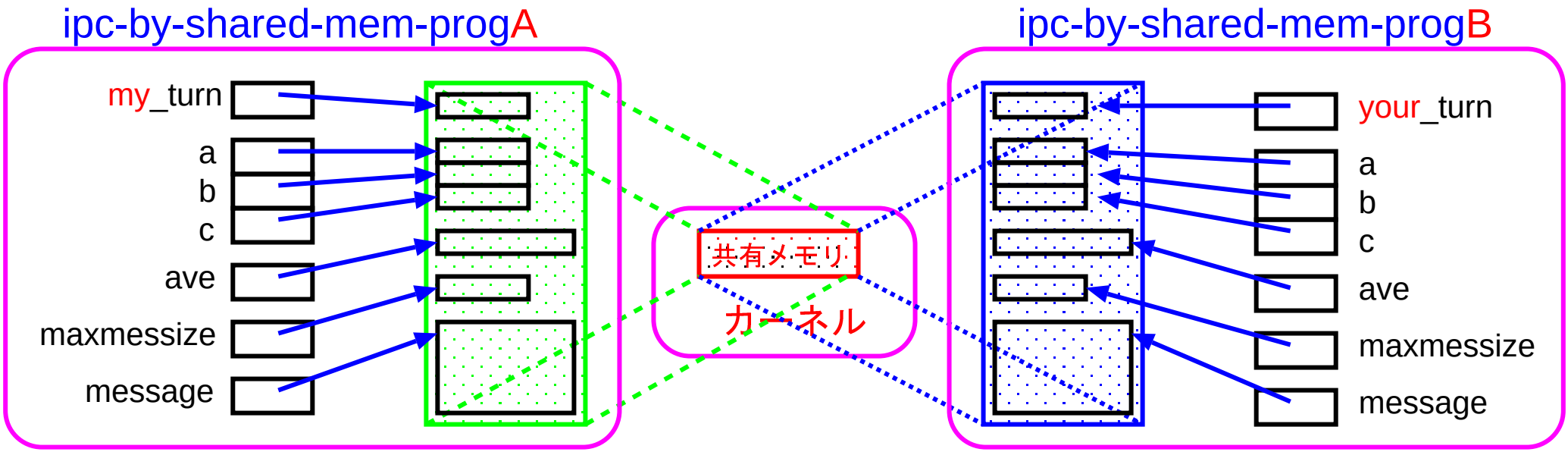
## 例15. 1 (共有メモリを使って血縁のないプロセス間で会話をする)

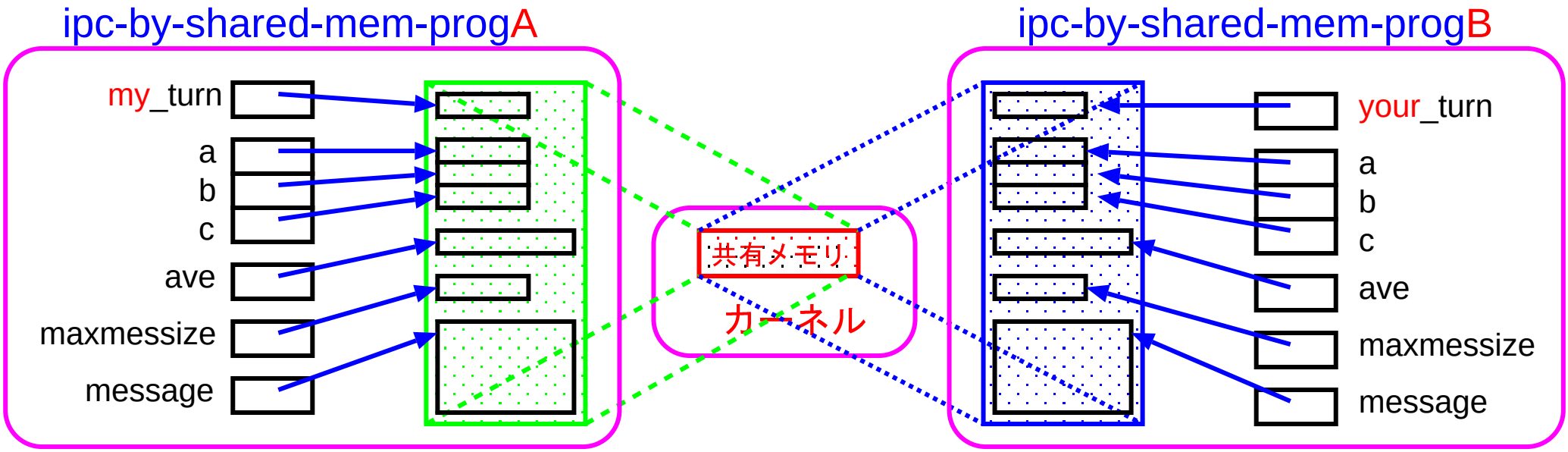
① 一方のプログラム(ipc-by-shared-mem-progA) が共有メモリ内のint型領域(*a，*b，*cの3つ)に値をセットし、

② 別のプログラム(ipc-by-shared-mem-progB)がそれらの値の平均を計算して共有メモリ内のdouble型領域(*ave)に値をセットし共有メモリ内のメッセージ用領域にも書き込む、

③ この時点で最初のプログラムの実行が再開し書き込まれたデータを出力する、

といった処理を行うプログラムのペアを次に示す。






補足説明：

2つのプログラムの実行を交互に行うために
共有メモリ内に1つのフラグ用の領域を確保した。

このフラグ用領域は、単一の領域だが

ipc-by-shared-mem-progA 内では *my_turn,

ipc-by-shared-mem-progB 内では *your_turn

という名前で参照する。

```
[motoki@x205a]$  pwd
/home/motoki/Operating-Systems2011/C-Programs
[motoki@x205a]$  nl ipc-by-shared-mem-progA.c
 1 /*************************************************
 2 /* Operating-Systems/C-Programs/ipc-by-shared-mem-progA.c
 3 /*-------------------------------------------------
 4 /* 共有メモリを使って2つのプログラム間で
 5 /* コミュニケーションする際のMaster側のプログラム
 6 /*************************************************
 7 #include <stdio.h>
 8 #include <stdlib.h>     /* for exit() library function */
 9 #include <sys/types.h>  /* for system calls of shared memor
10 #include <sys/ipc.h>    /* for system calls of shared memor
11 #include <sys/shm.h>    /* for system calls of shared memor

12 #define  TRUE   1
13 #define  FALSE  0
```

```
14 typedef  int  Boolean;

15 int main(void)
16 {
17   key_t key=1234;
18   int shmid, size;
19   char *shmaddress;
20   /*---------pointers to shared memory----------*/
21   Boolean *my_turn;
22   int *a, *b, *c;
23   double *ave;
24   int *maxmessize;
25   char *message;      /*array of size 256*/
26   /*---------pointers to shared memory----------*/

27   /* memory allocation */
```

```
28    size=sizeof(Boolean)+4*sizeof(int)
29          +sizeof(double)+256*sizeof(char)+4;
30                  /* "+4"は語の境界の制約がある場合のための保険 *
31    if ((shmid=shmget(key,size, IPC_CREAT|0666)) < 0) {
32      perror("shmget");
33      exit(EXIT_FAILURE);
34    }
35    if ((shmaddress=shmat(shmid,NULL,0)) == (char *)-1) {
36      perror("shmat");
37      exit(EXIT_FAILURE);
38    }
39    my_turn=(Boolean *)shmaddress;
40    a       =(int *)(my_turn+1);
41    b       =(int *)(a+1);
42    c       =(int *)(b+1);
43    ave     =(double *)(c+1);
```

共有メモリセグメントを作成

attach

共有メモリ内の変数へのポインタの設定

my_turnの分

```
44    maxmessize=(int *)(ave+1);
45    message=(char *)(maxmessize+1);
46    printf("address(my_turn)=%p\n"          設定したアドレスを表示
47           "address(a)      =%p\n"
48           "address(b)      =%p\n"
49           "address(c)      =%p\n"
50           "address(ave)    =%p\n"
51           "address(maxmessize)=%p\n"
52           "address(message)=%p\n",
53           my_turn, a, b, c, ave, maxmessize, message);

54    /* initialize */
55    *my_turn=TRUE;
56    *a = 1;
57    *b = 2;
58    *c = 4;
59    *maxmessize=256;
```

```
60    *ave = 0.0;

61    *my_turn=FALSE;
62    /*wait until the other process changes (*my_turn) to TRUE
63    while (!(*my_turn)) {
64      printf("<progA> I am waiting. "
                "(*my_turn)=%d (*ave)=%.2f\n",
65              *my_turn, *ave);
66      sleep(3);
67    }

68    /* print a message from the other process */
69    printf("\n<progA> It's my turn. "
            "(*my_turn)=%d (*ave)=%.2f\n"
70          "(message from the other process) %s\n",
71          *my_turn, *ave, message);
```

```
72    /* detach and release a shared memory */
73    if (shmdt(shmaddress) == -1) {
74      perror("shmdt");
75      exit(EXIT_FAILURE);
76    }else if (shmctl(shmid,IPC_RMID,NULL) == -1) {
77      perror("shmctl");
78      exit(EXIT_FAILURE);
79    }

80    return 0;
81  }
[motoki@x205a]$  nl ipc-by-shared-mem-progB.c
 1  /**********************************************************
 2  /* Operating-Systems/C-Programs/ipc-by-shared-mem-progB.c
 3  /*--------------------------------------------------------
 4  /* 共有メモリを使って2つのプログラム間で
 5  /* コミュニケーションする際のSlave側のプログラム          */
```

```
 6 /*************************************************************
 7 #include <stdio.h>
 8 #include <stdlib.h>     /* for exit() library function */
 9 #include <string.h>     /* for strncpy() library function *
10 #include <sys/types.h>  /* for system calls of shared memor
11 #include <sys/ipc.h>    /* for system calls of shared memor
12 #include <sys/shm.h>    /* for system calls of shared memor

13 #define  TRUE   1
14 #define  FALSE  0

15 typedef  int  Boolean;

16 int main(void)
17 {
18   key_t key=1234;
19   int shmid, size;
```

```
20    char *shmaddress;
21    /*--------pointers to shared memory---------*/
22    Boolean *your_turn;
23    int *a, *b, *c;
24    double *ave;
25    int *maxmessize;
26    char *message;      /*array of size 256*/
27    /*--------pointers to shared memory---------*/

28    /* memory allocation */
29    size=sizeof(Boolean)+4*sizeof(int)
30         +sizeof(double)+256*sizeof(char)+4;
31                  /* "+4"は語の境界の制約がある場合のための保険 *
32    if ((shmid=shmget(key,size,0666)) < 0) {
33      perror("shmget");   既存の共有メモリセグメントを探す
34      exit(EXIT_FAILURE);
35    }
```

```
36    if ((shmaddress=shmat(shmid,NULL,0)) == (char *)-1) {
37      perror("shmat");
38      exit(EXIT_FAILURE);
39    }
40    your_turn=(Boolean *)shmaddress;
41    a       =(int *)(your_turn+1);
42    b       =(int *)(a+1);
43    c       =(int *)(b+1);
44    ave     =(double *)(c+1);
45    maxmessize=(int *)(ave+1);
46    message=(char *)(maxmessize+1);
47    printf("address(your_turn)=%p\n"
48           "address(a)         =%p\n"
49           "address(b)         =%p\n"
50           "address(c)         =%p\n"
51           "address(ave)       =%p\n"
```

共有メモリ内の変数へのポインタの設定

```
52          "address(maxmessize)=%p\n"
53          "address(message)  =%p\n",
54          your_turn, a, b, c, ave, maxmessize, message);

55    /*wait until the other process changes (*your_turn) to FA
56    while (*your_turn) {
57      printf("(progB) I am waiting. "
              "(*your_turn)=%d (*ave)=%.2f\n",
58            *your_turn, *ave);
59      sleep(1);
60    }

61    /* processing */
62    *ave = (double)(*a+*b+*c)/3.0;
63    strncpy(message,"It's your turn.", *maxmessize-1);
64    if (15 < *maxmessize)
65      message[15]='\0';
```

```
66    else
67      message[*maxmessize-1]='\0';
68    *your_turn = TRUE;

69    /* detach a shared memory and exit */
70    if (shmdt(shmaddress) == -1) {
71      perror("shmdt");
72      exit(EXIT_FAILURE);
73    }

74    return 0;
75  }
[motoki@x205a]$
    gcc -o ipc-by-shared-mem-progA ipc-by-shared-mem-progA.c
[motoki@x205a]$
    gcc -o ipc-by-shared-mem-progB ipc-by-shared-mem-progB.c
[motoki@x205a]$  ./ipc-by-shared-mem-progA
```

```
address(my_turn)=0xb773c000
address(a)       =0xb773c004
address(b)       =0xb773c008
address(c)       =0xb773c00c
address(ave)     =0xb773c010
address(maxmessize)=0xb773c018
address(message)=0xb773c01c
<progA> I am waiting. (*my_turn)=0 (*ave)=0.00
<progA> I am waiting. (*my_turn)=0 (*ave)=0.00
<progA> I am waiting. (*my_turn)=0 (*ave)=0.00
<progA> I am waiting. (*my_turn)=0 (*ave)=0.00
   ..............................
```

別のプロセスが共有メモリ内の変数 *my_turn の内容を**TRUE(0以外)** に変更するのを待つ状態に入る。

ビジーウェイト(&sleep)状態になるので、ここで、
別の仮想端末上でもう一方のプログラムを起動すると、
2つの仮想端末上で実行が並行して進む。

```
[motoki@x205a]$  pwd
/home/motoki/Operating-Systems2011/C-Programs
[motoki@x205a]$  ./ipc-by-shared-mem-progB
address(your_turn)=0xb77b4000
address(a) =0xb77b4004
address(b) =0xb77b4008
address(c) =0xb77b400c
address(ave) =0xb77b4010
address(maxmessize)=0xb77b4018
address(message) =0xb77b401c
[motoki@x205a]$
```

プロセスが「ビジーウェイト(&sleep)状態」を脱出する。

```
<progA> It's my turn. (*my_turn)=1 (*ave)=2.33
(message from the other process) It's your turn.
[motoki@x205a]$
```

# 15–2 セマフォア

## System Vのセマフォア：

- 非負整数を値域とするセマフォアが使える。
- 複数の資源の排他制御を行うために、複数のセマフォアを同時に要求し確保することが出来る。

注意：
単に、1つのシステムコールの中に複数の資源要求をまとめて書けるだけでなく、必要な資源は同時に確保・解放される。

# セマフォのIDを取得するためのシステムコール

`int semget(key_t key, int nsems, int semflg);`

- <sys/types.h>, <sys/ipc.h> と <sys/sem.h> を必要とする。
- 引数の指定に従ってカーネルの領域内にセマフォア(の組)が生成されるか、指定に合った既存のセマフォア(の組)が探される。

  成功すると セマフォアID(正整数) が返され、...
- 指定されたIDを持つセマフォアは(一般には) 複数出来る。この中の個々のセマフォアには識別のため0,1,2,... という番号が付けられる。
- 関数引数の key には、セマフォアを共有しようとするプロセス間で共通の数字名を指定する。

  IPC_PRIVATEを指定すると、...
- 関数引数の nsems は、セマフォアの個数を表す。
- 関数引数の semflg はこのセマフォア(の組)に対するアクセス許可を指定するために使われる。...

## セマフォア操作のシステムコール

`int semop(int semid, struct sembuf *sops, unsigned nsops) ;`

- <sys/types.h>, <sys/ipc.h> と <sys/sem.h> を必要とする。
- IDがsemidのセマフォア群をカーネル領域から探し出し、その中のセマフォアに対して指定された操作

  `sops[0], sops[1], ..., sops[nsops-1]`

  が同時並行的に処理されてゆく。

  成功すると 0 が返され、失敗すると $-1$ が返される。
- セマフォアに対する各操作 `sops[k]` は

```
struct sembuf {
  short sem_num;   /*セマフォア番号*/
  short sem_op;    /*操作          */
  short sem_flg;   /*操作制御用のフラグ*/
}
```

  という型の構造体で表される。 ⇨ 具体的には、次ページ

個々のsops[k]は次の様に解釈・実行される。

(場合1：sops[k].sem_op<0)

P操作を実行する。 すなわち、

① sops[k].sem_num番目のセマフォア値+sops[k].sem_op $\geq 0$ となるまでプロセスを待ち状態にする。

② sops[k].sem_num番目のセマフォア値

$\longleftarrow$ sops[k].sem_num番目のセマフォア値+sops[k].sem_op;

(場合2：sops[k].sem_op>0)

V操作を実行する。 すなわち、

sops[k].sem_num番目のセマフォア値

$\longleftarrow$ sops[k].sem_num番目のセマフォア値+sops[k].sem_op;

(場合3：sops[k].sem_op=0)

**sops[k].sem_num番目のセマフォア値=0** となるまでプロセスを待ち状態にする。

注意：

semop()はアトミックオペレーションで、sops[$k$]操作は全て実行されるか、全く実行されないかのいずれかである。

注意：

sops[k].sem_flgにIPC_NOWAITフラグを立てた時は、場合1, 場合3の条件を満たさない場合は直ちにエラーコードが返される。

# セマフォアの状態を制御するシステムコール

## int semctl(int semid, int semnum, int cmd, union semun arg) :

- <sys/types.h>, <sys/ipc.h> と <sys/sem.h> を必要とする。
- 関数引数の semid には、セマフォア群のID を...
- 関数引数の cmd は制御内容を表す定数で、IPC_STAT, IPC_SET, IPC_RMID, GETVAL, SETVAL, GETPID, GETNCNT, GETZCNT, GETALL, または SETALL のいずれかを指定する。
- 関数引数の arg はcmdで指定される様々な制御を補助するパラメータで、次の共用体型を持つ。

```
union semun {
  int val;                    /*value for SETVAL*/
  struct semid_ds *buf;       /*IPC_STAT's buffer*/
  unsigned short int *array; /*GETALL and SETALL's buffer
}
```

arg引数にNULL以外のものを指定する場合は、この共用体定義と同等のものがプログラム内に必要。

- semctl(semid,semnum,IPC_STAT,arg) が呼ばれると、
  IDがsemidのセマフォア群の情報(e.g.セマフォアの個数, セマフォア群を作成したプロセスのID, .....) がarg.bufに格納される。

- semctl(semid,semnum,IPC_SET,arg) が呼ばれると、
  IDがsemidのセマフォア群の情報がarg.bufに記録された通りに変更される。

- semctl(semid,semnum,IPC_RMID,arg) が呼ばれると、
  IDがsemidのセマフォア群が除去される。

- semctl(semid,semnum,GETVAL,arg) が呼ばれると、
  semnum番目のセマフォアの値が返される。

- semctl(semid,semnum,SETVAL,arg) が呼ばれると、
  semnum番目のセマフォアの値がarg.valに変更される。

- semctl(semid,semnum,GETPID,arg) が呼ばれると、
  semnum番目のセマフォアに対して最後に操作を行ったプロセスのIDが返される。

- semctl(semid,semnum,GETNCNT,arg) が呼ばれると、
  semnum番目のセマフォアの値が現在値より大きくなるのを待っているプロセスの個数が返される。

- semctl(semid,semnum,GETZCNT,arg) が呼ばれると、
  semnum番目のセマフォアの値が0になるのを待っているプロセスの個数が返される。

- semctl(semid,semnum,GETALL,arg) が呼ばれると、
  IDがsemidのセマフォア群中の全てのセマフォアの値がarg.arrayの指す配列に格納される。

- semctl(semid,semnum,SETALL,arg) が呼ばれると、
  IDがsemidのセマフォア群の中の個々のセマフォアの値が配列arg.array に記録された通りに変更される。

## 例15. 2 (食事する哲学者達)

例14.4のプログラムと同等のことをSystem V のセマフォア機構を用いて行うプログラムを次に示す。

[motoki@x205a]$ nl dining-phil-deadlock-systemV-sem.c

```
 1 /**************************************************
 2 /* Operating-Systems/C-Programs/dining-phil-deadlock-Syste
 3 /*-------------------------------------------------
 4 /* SystemV semaphoreを用いて排他制御を行うことにして、      *
 5 /*      不用意に排他制御を行うとDeadlockの状態に陥ること     *
 6 /* を有名な「Dining Philosophersの問題」で確かめる。       *
 7 /* A.ケリー&I.ポール「CのABC(下)」アジソンウェスレイ       *
 8 /*                                    ジャパン,12.5節      
 9 /**************************************************
10 #include <stdio.h>
11 #include <stdlib.h>     /* for exit() library function */
12 #include <unistd.h>     /* for fork() system call */
13 #include <sys/types.h>  /* for wait() and SystemV semapho
```

```
14 #include <sys/ipc.h>     /* for SystemV semaphore system ca
15 #include <sys/sem.h>     /* for SystemV semaphore system ca
16 #include <sys/wait.h>    /* for wait() system call */

17 union semun {   semctl()を使うので
18   int val;                        /* value for SETVAL */
19   struct semid_ds *buf;           /* IPC_STAT's buffer */
20   unsigned short int *array; /* GETALL and SETALL's buffer
21 };

22 #define  NUM_OF_PHIL  3     /*哲学者の人数*/
23 #define  Left_chopstick(x)   (x)
24 #define  Right_chopstick(x) (((x)+1) % NUM_OF_PHIL)

25 int Semid;

26 void simulate_behaviour_of_philosopher(int k);
```

```
27 void Print_an_event(int k, char *event);
28 void initialize_semaphore();
29 void P(int semnum);
30 void V(int semnum);

31 int main(void)
32 {
33   int k, status;

34   initialize_semaphore();

35   for (k=0; k<NUM_OF_PHIL; k++)
36     printf("Philosopher%2d          ", k);
37   printf("\n");
38   for (k=0; k<NUM_OF_PHIL; k++)
39     printf("--------------------  ");
40   printf("\n");
```

例14.4とほぼ同じ

セマフォア(箸)を用意

```
41
42    for (k=0; k<NUM_OF_PHIL; k++) {
43      if (fork()==0) {  /*子プロセスは哲学者*/
44        simulate_behaviour_of_philosopher(k);
45        exit(EXIT_SUCCESS);
46      }
47    }
48                              /*親プロセス*/
49    for (k=0; k<NUM_OF_PHIL; k++)
50      wait(&status);
51    if (semctl(Semid,0,IPC_RMID,NULL) < 0) {
52      perror("semctl");   セマフォア群を削除
53      exit(EXIT_FAILURE);
54    }

55    return 0;
56 }
```

```
57 /* k番目の哲学者の動作をシミュレートする */
58 void simulate_behaviour_of_philosopher(int k)
59 {
60   int i;

61   for (i=0; i<5; i++) {
62     P(Left_chopstick(k));
63     Print_an_event(k, "pick up left stick");
64     Print_an_event(k, "      ***thinking***");
65     sleep(1);
66     P(Right_chopstick(k));
67     Print_an_event(k, "pick up right stick");
68     Print_an_event(k, "***eating***");
69     sleep(1);
70     V(Left_chopstick(k));
71     Print_an_event(k, "put down left stick");
```

パイプを用いて実装した例14.4と全く同じ

```
72      V(Right_chopstick(k));
73      Print_an_event(k, "put down right stick");
74      Print_an_event(k, "      ***thinking***");
75      sleep(1);
76    }
77 }

78 void Print_an_event(int k, char *event)
79 {
80    int i, indentsize;

81    indentsize=22*k;
82    for (i=0; i<indentsize; i++)
83      putchar(' ');
84    printf("%s\n", event);
85 }
```

例 14.4 と同じ

SystemVのセマフォアを使った実装

```
86 /* セマフォアの初期設定, PV操作 */
87 void initialize_semaphore()
88 {
89   int i;
90   union semun arg;
91   if ((Semid=semget(IPC_PRIVATE, NUM_OF_PHIL,
                        IPC_CREAT|0666)) < 0) {
92     perror("semget");
93     exit(EXIT_FAILURE);
94   }
95
96   arg.val = 1;
97   for (i=0; i<NUM_OF_PHIL; i++) {
98     if(semctl(Semid,i,SETVAL,arg) < 0) {
99       perror("semctl");
```

セマフォアの個数

i番目のセマフォアに値1をセット

```
100         exit(EXIT_FAILURE);
101       }
102     }
103 }

104 void P(int semnum)
105 {
106     struct sembuf sem_ops[1];

107     sem_ops[0].sem_num=semnum;
108     sem_ops[0].sem_op =-1;
109     sem_ops[0].sem_flg=0;
110     if (semop(Semid,sem_ops,1) < 0) {
111       perror("semop(P-operation)");
112       exit(EXIT_FAILURE);
113     }
114 }
```

資源番号

資源の獲得

```
115 void V(int semnum)
116 {
117   struct sembuf sem_ops[1];

118   sem_ops[0].sem_num=semnum;
119   sem_ops[0].sem_op =1;
120   sem_ops[0].sem_flg=0;
121   if (semop(Semid,sem_ops,1) < 0) {
122     perror("semop(V-operation)");
123     exit(EXIT_FAILURE);
124   }
125 }
[motoki@x205a]$  gcc dining-phil-deadlock-systemV-sem.c
[motoki@x205a]$  ./a.out
Philosopher 0          Philosopher 1          Philosopher 2
--------------------   --------------------   --------------------
```

資源番号

資源の解放

```
pick up left stick
      ***thinking***
                        pick up left stick
                              ***thinking***
                                                  pick up left stick
                                                        ***thinking***
```

**Ctrl-Cで強制終了**

```
[motoki@x205a]$
```

補足：

カーネルの領域内にセマフォア群が生成され、削除の前にデッドロック状態になってしまったので、生成されたセマフォア群はカーネル領域内に残ったままになっている。 これはipcsコマンドにより確認できる。次の通り。

```
[motoki@x205a]$  ipcs -s
------ セマフォ配列 --------
キー         semid      所有者     権限       nsems
0x00000000 0          motoki     666        3
[motoki@x205a]$
```

カーネル領域内に残ったままのセマフォア群を削除するには

次の様にipcrmコマンドを用いればよい。

```
[motoki@x205a]$  ipcrm sem 0   <--semid=0
リソースを削除しました
[motoki@x205a]$  ipcs -s
------ セマフォ配列 --------
キー         semid      所有者     権限       nsems
[motoki@x205a]$
```

補足(重要):

同時に使えるid番号の個数にはコンピュータ毎に上限があるので、例えば

カーネル領域内に使ったセマフォア群を残したままにしてしまう人が多いと、

(プログラムに間違いが無くても)セマフォアを利用できないことがある。

⇨ **SystemVのIPC機構を使った後は必ず後始末をすること。**

## 演習(食事する哲学者達)

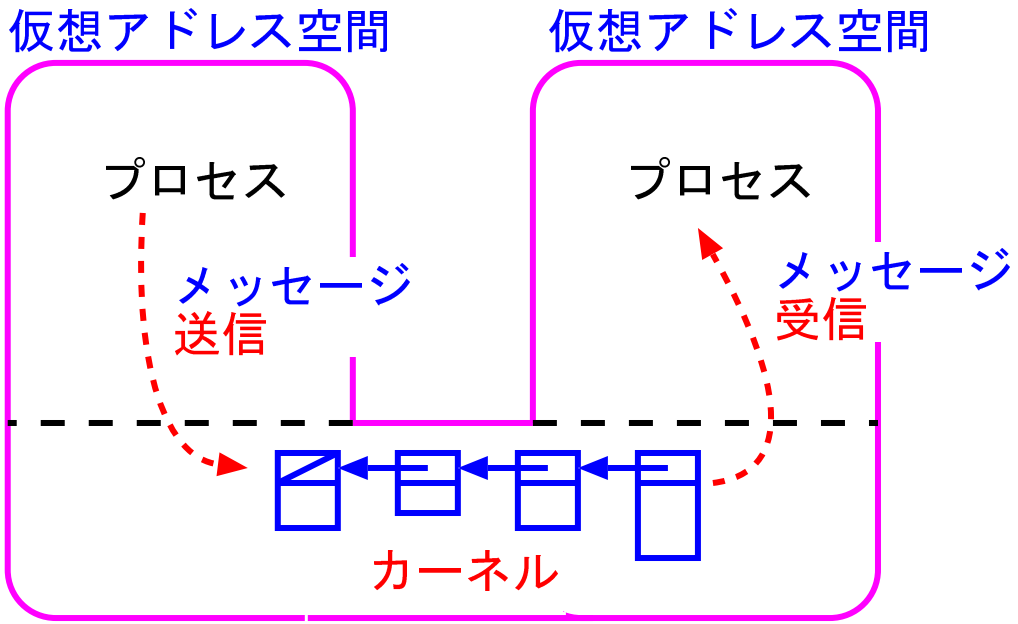
≈レポート課題

上の例15.2のプログラムを修正して、デッドロックが起こらないようにせよ。

# 15–3 メッセージキュー

## メッセージキューとは：

- 名前付きパイプと同じ様に、血縁関係にないプロセス同士でもメッセージのやり取りを行えるための機構が用意されている。
- 名前付きパイプと違って、メッセージ毎に1つのデータ構造が構成され、それらがポインタで線形リスト状に繋がれて待ち行列が構成される。
- 名前付きパイプと違って、メッセージの待ち行列はカーネルの領域内に構成される。




名前付きパイプの場合は、
メッセージ間の区切りはなく、送ったメッセージは前のメッセージの最後尾に追加される。

名前付きパイプの場合は、
ファイルシステム内に作られる。

# メッセージキューのIDを取得するためのシステムコール

`int msgget(key_t key, int msgflg);`

- <sys/types.h>, <sys/ipc.h> と <sys/msg.h> を必要とする。
- 引数の指定に従ってカーネルの領域内にメッセージキューが生成されるか、指定に合った既存のメッセージキューが探し出される。

  成功するとキューID(正整数) が返され、失敗すると $-1$ が返される。
- 関数引数の key には、メッセージキューを共有しようとするプロセス間で共通の数字名を指定する。

  IPC_PRIVATE を指定すると、...
- 関数引数の msgflg はこのメッセージキューに対するアクセス許可を指定するために使われる。...

## メッセージキューにメッセージを送信するシステムコール

`int msgsnd(int msqid, void *msgp, int msgsz, int msgflg) :`

- <sys/types.h>, <sys/ipc.h> と <sys/msg.h> を必要とする。

- IDがmsqidのメッセージキューをカーネル領域から探し出し、そこにポインタmsgpの指す長さmsgszのメッセージを送り出す。

  成功すると 0 が返され、失敗すると $-1$ が返される。

- 関数引数の msgflg は普通は0 と設定してメッセージキューが満杯の時は空きが出来るまでプロセスが待ち状態になる様にするが、

  IPC_NOWAIT フラグを立てた場合は、メッセージキューが満杯の時は直ちにエラーコードが返される。

- ヘッダファイル<sys/msg.h>の中では、メッセージを入れる構造体のひな型が次の様に定義されている。

```
struct msgbuf {
  long mtype;     /*メッセージのタイプ番号(>0)*/
  char mtext[1]; /*メッセージを構成する文字列*/
};
```

⇨ プログラムを実際に作る際は

配列mtext を十分大きく取った構造体を定義する必要がある。

注意： 構造体 struct msgbuf はヘッダファイル<sys/msg.h>の中で既に定義されているので、同じ名前で次の様に書くとエラーになる。

```
struct msgbuf {
  long mtype;       /*メッセージのタイプ番号(>0)*/
  char mtext[256]; /*メッセージを構成する文字列*/
};
```

⟹ 独自の名前の構造体を定義する必要がある。

---

関数msgsnd()の第2引数のデータ型は(void *)であるので、任意のポインタに適合する。

## メッセージキューからメッセージを受け取るシステムコール

```
int msgrcv(int msqid, void *msgp, int msgsz,
                                long mstyp, int msgflg) ;
```

- <sys/types.h>, <sys/ipc.h> と <sys/msg.h> を必要とする。
- IDがmsqidのメッセージキューをカーネル領域から探し出し、そこからメッセージを1つ受け取り、ポインタmsgpの指す構造体領域に書き込む。

  成功すると 0 が返され、失敗すると $-1$ が返される。
- 受け取るメッセージは第4引数mstypの値によって異なる。

  (場合1 : mstyp>0) タイプ番号がmstypのメッセージの内、最初のものを受け取る。

  (場合2 : mstyp=0) メッセージキューの中の最初のメッセージを受け取る。

  (場合3 : mstyp<0) タイプ番号が |mstyp| 以下のメッセージの内、最初のものを受け取る。

- ポインタmsgpの指す構造体はmsgsnd( )の送ったメッセージと同じで、次の形をしている。

```
struct msgbuf {
  long mtype;          /*メッセージのタイプ番号(>0)*/
  char mtext[msgsz]; /*メッセージを構成する文字列*/
}
```

- 関数引数の msgsz にはメッセージを格納する領域の長さを指定する。

  受け取ったメッセージがmsgszより長い場合は、
  msgflgにMSG_NOERRORフラグが立っていればmsgszまで切り詰め、
  そうでなければ直ちにエラーコードが返される。

- 関数引数の msgflg は普通は0 と設定して該当するメッセージが無い時は届くまでプロセスが待ち状態になる様にするが、

  IPC_NOWAITフラグを立てた場合は、該当するメッセージが無い時は直ちにエラーコードが返される。

# メッセージキューの状態を制御するシステムコール

int msgctl(int msqid, int cmd, struct msqid_ds *buf) :

- <sys/types.h>, <sys/ipc.h> と <sys/msg.h> を必要とする。
- 関数引数の msqid には、...
- 関数引数の cmd は制御内容を表す定数で、IPC_STAT, IPC_SET, IPC_RMID のいずれかを指定する。
- 引数の buf の指す構造体型 struct msqid_ds は...
- msgctl(msqid,IPC_STAT,buf) が呼ばれると、
  IDがmsqidのメッセージキューの情報 (e.g. メッセージの個数, 最初のメッセージへのポインタ, .....) がbufの指す構造体に格納される。
- msgctl(msqid,IPC_SET,buf) が呼ばれると、
  IDがsemidのメッセージキューの情報がbufの指す構造体に記録された通りに変更される。
- msgctl(msqid,IPC_RMID,buf) が呼ばれると、
  IDがsemidのメッセージキューが除去される。
- どの場合も、成功すると 0 が返され、失敗すると $-1$ が返される。

## 例15. 3 (メッセージキューを使って親子のプロセス間で会話をする)

例14.3のプログラムとほぼ同等のことをメッセージキューの機構を用いて行うプログラムを次に示す。

```
[motoki@x205a]$  nl ipc-by-message-queue.c
 1 /*************************************************************/
 2 /* Operating-Systems/C-Programs/ipc-by-message-queue.c  */
 3 /*-----------------------------------------------------------*/
 4 /* メッセージキューを使って                                  */
 5 /* 親プロセスと子プロセスの間でコミュニケーションする例 */
 6 /*************************************************************/
 7 #include <stdio.h>
 8 #include <stdlib.h>     /* for exit() library function */
 9 #include <string.h>     /* for strlen() library function */
10 #include <unistd.h>     /* for fork() system call */
11 #include <sys/types.h>  /* for wait() and message queue sys
12 #include <sys/ipc.h>    /* for message queue system calls *
13 #include <sys/msg.h>    /* for message queue system calls *
```

```
14 #include <sys/wait.h>   /* for wait() system call */

15 #define MSGSIZE  256
16 #define PARENT2CHILD  1
17 #define CHILD2PARENT  2

18 typedef struct {
19   long mtype;
20   char mtext[MSGSIZE];
21 }Messagebuf;

22 int main(void)
23 {
24   int msqid, k, status;
25   pid_t childpid;
26   Messagebuf message;
27
```

メッセージタイプ

メッセージを入れる構造体

メッセージを入れる場所

```
28   if ((msqid=msgget(IPC_PRIVATE,IPC_CREAT|0666)) < 0) {
29     perror("msgget");
30     exit(EXIT_FAILURE);
31   }

32   if ((childpid=fork())==-1) {
33     perror("can't fork");
34     exit(EXIT_FAILURE);
35   }else if (childpid==0) {  /* 子プロセス */
```

```
35    }else if (childpid==0) {  /* 子プロセス */
36      for (k=0; k<6; k++) {
37        if (msgrcv(msqid, &message,
                      MSGSIZE, PARENT2CHILD,0) < 0) {
38          perror("child: msgrcv");
39          exit(EXIT_FAILURE);
40        }
41        printf("(child) It's my %d-th turn to process.\n",
                k);
42        sleep(3-k%3);   /* 子プロセスの処理の代わり */
43        message.mtype = CHILD2PARENT;   送るメッセージを構成
44        sprintf(message.mtext,"It's your turn to process.");
45        if (msgsnd(msqid, &message,
                      strlen(message.mtext)+1, 0) < 0) {
46          perror("child: msgsnd");
47          exit(EXIT_FAILURE);
48        }
```

```
49      }
50      exit(EXIT_SUCCESS);
51    }else {                          /* 親プロセス */
52      for (k=0; k<6; k++) {
53        printf("<PARENT> It's my %d-th turn to process.\n",
                k);
54        sleep(k%2+1);    /* 親プロセスの処理の代わり */
55        message.mtype = PARENT2CHILD;
56        sprintf(message.mtext,"It's your turn to process.");
57        if (msgsnd(msqid, &message,
                    strlen(message.mtext)+1, 0) < 0) {
58          perror("parent: msgsnd");
59          exit(EXIT_FAILURE);
60        }
61        if (msgrcv(msqid, &message,
                    MSGSIZE, CHILD2PARENT,0) < 0) {
62          perror("parent: msgrcv");
```

```
63          exit(EXIT_FAILURE);
64        }
65      }
66      wait(&status);    メッセージキューの削除
67      if (msgctl(msqid,IPC_RMID,NULL) < 0) {
68        perror("msgctl(msqid,IPC_RMID,NULL)");
69        exit(EXIT_FAILURE);
70      }
71    }

71    return 0;
73 }
[motoki@x205a]$  gcc ipc-by-message-queue.c
[motoki@x205a]$  ./a.out
<PARENT> It's my 0-th turn to process.
(child) It's my 0-th turn to process.
<PARENT> It's my 1-th turn to process.
```

```
(child) It's my 1-th turn to process.
<PARENT> It's my 2-th turn to process.
(child) It's my 2-th turn to process.
<PARENT> It's my 3-th turn to process.
(child) It's my 3-th turn to process.
<PARENT> It's my 4-th turn to process.
(child) It's my 4-th turn to process.
<PARENT> It's my 5-th turn to process.
(child) It's my 5-th turn to process.
[motoki@x205a]$
```