パイプとプロセス間通信

Step 1：シェル機能を利用して、「標準入力」を「標準出力」に手渡す。

まず、以下のようなプログラムを入力します。名称は、foo.c にします。
（foo とか、bar は、「意味がない」ファイルにつける名称です。Java のサンプルで hoge とか moge を使うのと同じです。決して、「意味がある」ファイルにこれらの名称は使わないで下さい。）

```
# vi foo.c
```

```
#include   <stdio.h>

main( int argc, char **argv )
{
  char c;
  FILE *outfp;

  outfp = fopen( "foo.txt", "w" );

  while( (c=getc(stdin))!=EOF )
  {
    fprintf( stdout, "%c", c );
    fprintf( outfp,  "%c", c );
  }
  fclose( outfp );
}
```

何をやっているプログラムか、と言えば、stdin つまり、標準入力から、EOF（End Of File 入力終了；ファイルの終わり）まで、一文字ずつ読み込みます。読み込んだ文字は、foo.txt というファイルと、「標準出力」に書き出します。

このファイルをコンパイルして、foo という実行プログラムを作ります。

```
# cc -o foo foo.c
```

次に、データファイルとして、Fruits.txt というファイルを作成します。

```
# vi Fruits.txt
```

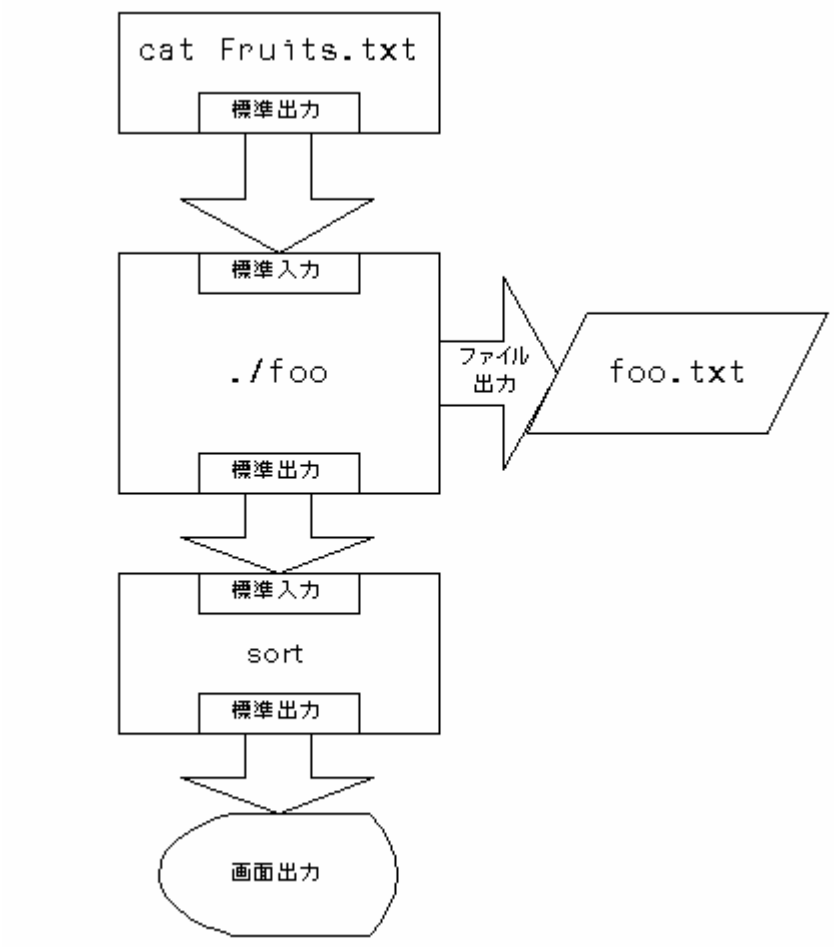
中身は、以下のようにフルーツの名称を入れます。（サンプルです。Sports.txt でも、何で

も構いません。自分で考えて作ってみてください。）

```
Grape
Orange
Apple
Lemon
Banana
Strawberry
```

ここで、以下のコマンドを入力してみてください。

```
# cat Fruits.txt | ./foo | sort
```



cat で標準出力に出力された内容は、|のパイプでそのまま./foo のプログラムに渡されます。Foo では、入力した内容をそのままファイル「foo.txt」に書き込み、同じものを標準出力に渡します。その結果を、今度は sort で受け取り、処理した結果を画面に出力します。

```
# cat foo.txt
```

で、foo.txt の内容を表示し、sort の結果と比べてみてください。sort で何を行ったか考察してみてください。

次に、

```
# sort Fruits.txt | ./foo | sort -r
```

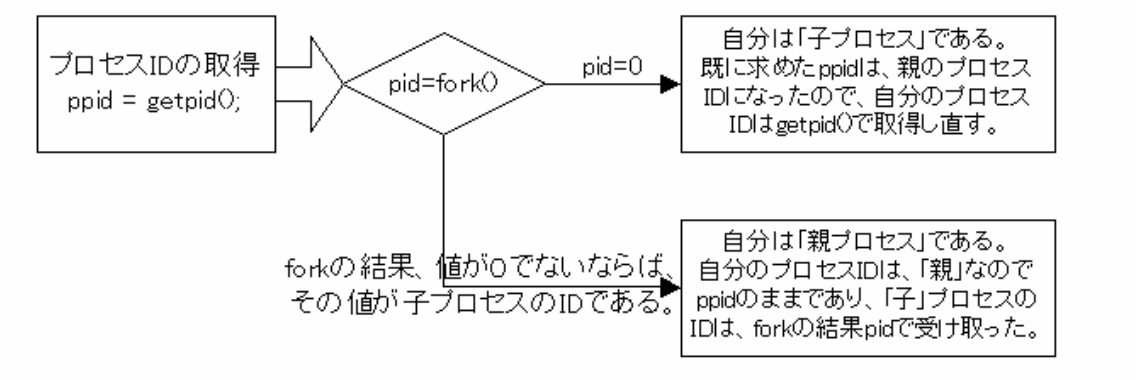
を実行し、同様に foo.txt の内容を画面表示の内容と比べてみてください。

Step 2 : プロセスの fork

次に、以下のプログラムを入力します。

```
# vi  fork.c
```

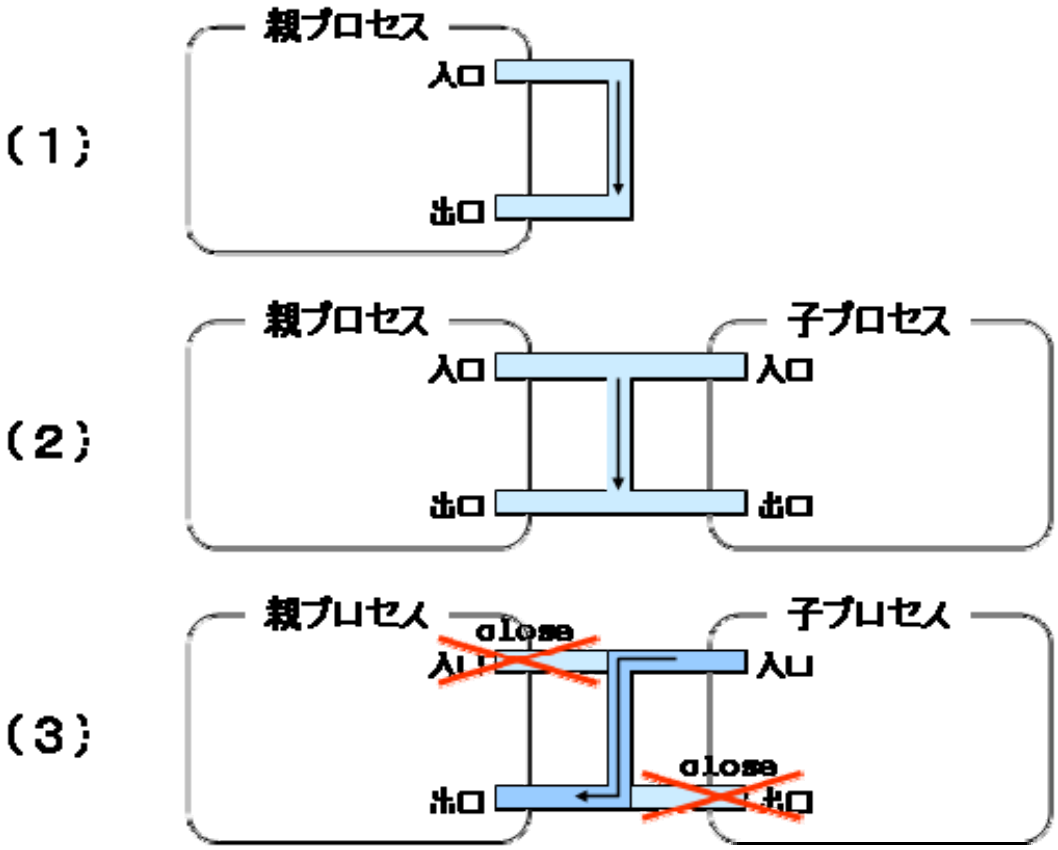
```
#include   <stdio.h>
#include   <sys/types.h>
#include   <unistd.h>

main( int argc, char **argv)
{
  int pid;   /* Process ID */
  int ppid;  /* Parent Process ID */

  ppid = getpid();

  if( (pid=fork())==0 ){
    printf( "I am a child. " );
    printf( "My process ID = %d.¥n", getpid() );
    printf( "And my parent ID=%d¥n", ppid );
  }
  else {
    printf( "I am a parent. " );
    printf( "Child ID=%d, ", pid );
    printf( "and my process ID=%d¥n", ppid );
  }
}
```



画面表示を確認してください。
If の中身と、else の中身の両方が並んで画面に表示されていることがわかります。
つまり、fork()を実行した後で、二つのプロセスが起動していることがわかります。
一旦 fork した後は二つのプロセスは独立ですので、定義されている変数を共有するなどはできません。データの受け渡しを行うには、パイプを作成する必要があります。

Step 3：プロセス間の通信（pipe）

このプロセス間のパイプの作成方法は、以下のようになります。



1. pipe システムコールを呼ぶと，入力と出力それぞれの口（ファイルディスクリプタ）を持つパイプが 1 本作られる．
2. この状態で fork を呼ぶと，パイプ（入出力の口）が両方とも共有された状態になる．
3. 一方のプロセスが入力を，もう一方が出力の口を閉じると，プロセスからプロセスへデータを送れる（プロセス間通信）状態になる．

（引用元：http://www.coins.tsukuba.ac.jp/~syspro/2005/No3.html）

以下のプログラムを入力します。

```
# vi  pipe.c

    #include   <stdio.h>
    #include   <sys/types.h>
    #include   <unistd.h>

    int pipe_fd[2];
```

```
int pid;  /* Process ID */
int ppid;  /* Parent Process ID */

do_child( void )
{
   char *p = "Hello, Dad!¥n";

   pid = getpid();
   printf( "I am child.(%d)¥n", pid );

   close( pipe_fd[0] );
   while( *p )
   {
       write(pipe_fd[1], p++, 1 );
   }
}

do_parent( void )
{
    char c;
    int  count;

    printf( "I am parent.(%d)¥n", ppid );
    close( pipe_fd[1] );

    while( (count=read(pipe_fd[0], &c,1))> 0 )
    {
       putchar( c );
    }
}

main( int argc, char **argv)
{
  ppid = getpid();
  pipe( pipe_fd );

  if( (pid=fork())==0 )
       do_child();
  else do_parent();
}
```

コンパイルします。

```
# cc  -o pipe pipe.c
```

実行します。

```
# ./pipe
```

何気なく画面に表示が出るだけですが・・・、中でどんな動作をしているかわかりますか？
それが課題です。

本日の課題は、プログラムを入力するのが面倒くさい人は、この pdf 教材からプログラムをコピーして、pipe.c　fork.c などのファイルを作成し、tar フォーマットに圧縮します。
MINIX を起動する前に、圧縮した tar ファイルを vfloppya.img にコピーします。（または、vfloppya.img を vfloppya.img.org にリネームしてから、名前を vfloppya.img に変更します。）
こうして MINIX を起動し

```
# tar xvf /dev/fd0
```

と入力すると、仮想フロッピーディスクからプログラムを読み込むことができます。

括弧（）の数や、対応、;（セミコロンの位置）、などに注意して入力してください。