//pipeはコマンドとコマンドをつなぐ接着剤のようなもの
//リダイレクションはdupかdup2、パイプはpipeというシステムコールから実現。
//execveでリダイレクションするプログラム
// ./a.out filename command string という引数で実行する

```
#include <libc.h>
int main(int argc, char *argv[])
{
    int fd;
    if (argc < 3)
    {
        fprintf(stderr, "%s filename command [arg...] \n", argv [0]);
        exit (1);
    }
    fd = open (argv [1], O_RDONLY);
    dup2(fd, STDIN_FILENO);
    execvp(argv[2], &argv[2]);
    return (0);
}
```

//pipeとは、プロセス間で通信、つまりデータ交換するための仕組みのひとつ。
//pipeに書いたデータを自分で読むプログラム

```
#include <libc.h>
int main(void)
{
    int fd[2];
    int nbytes;
    char buf[1024];
    if (pipe(fd) == -1)
    {
        perror("pipe");
        exit (1);
    }
    write(fd[1], "Hello \n", 6);
    nbytes = read(fd[0], buf, sizeof(buf));
    write(STDOUT_FILENO, buf, nbytes);
    close(fd[0]);
    close(fd[1]);
    return (0);
}
```

//pipeに自分で書いて自分で読めるのは、pipeにバッファがあるから。
//pipeのバッファより大きなデータを書き込むと、実行が進まなくなる。
//forkとは、呼び出し元プロセスを複製して、新たに子プロセスを作るコマンド。
//pipeを使って親プロセスから子プロセスにメッセージを送るプログラム

```
#include <libc.h>
#include <unistd.h>
int main(void)
{
    int n;
    int fd[2];
    char buf[1024];
    pid_t pid;
    pipe(fd);
    if ((pid = fork()) == 0)
    {
            // 子プロセス
        close(fd[1]);
        n = read(fd[0], buf, sizeof(buf));
        write(1, buf, n);
        close(fd[0]);
    }
    else
    {
            // 親プロセス
        close(fd[0]);
        write(fd[1], "Hello \n", 6);
        close(fd[1]);
        wait(NULL);
    }
    return (0);
}
```

//以下はshellのpipeを作るプログラム
// ./a.out ls wc など引数2つで実行する。

```
#include <libc.h>
int main(int argc, char *argv[])
{
    int fd[2];
    pid_t pid;
    if (argc < 3)
    {
        fprintf(stderr, "%s command1 command2 \n", argv[0]);
    }
    pipe(fd);
    if ((pid = fork()) == 0)
    {
        //子プロセス
        dup2(fd[1], 1);
        close(fd[0]);
        close(fd[1]);
        execlp(argv[1], argv[1], (char *)NULL);
    }
    //親プロセス
    dup2(fd[0], 0);
    close(fd[0]);
    close(fd[1]);
    execlp(argv[2], argv[2], (char *)NULL);
    return (0);
}
```