ソケットプログラミング(2)
名前付きパイプ

九州工業大学
数理・DS・AI教育推進室

説明内容

I. C言語プログラミング
  - ファイル入出力：ライブラリ関数
  - システムコール

II. プロセス間通信
  - 親子プロセス間　～　パイプの利用
  - 任意プロセス間
    - 名前付きパイプ
    - セマフォ、共有メモリ（※ 省略）
    - ソケット（異なるマシン上のプロセス間通信の実現）

III. ソケット通信

前回課題の解説
(3. システムコールによるファイル入出力)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>  // strlen()
#include <fcntl.h>   // open(), creat()
#include <unistd.h>  // read(), write(), close()

int main(){
  char buf[13] = “\0”;                          // 1.
  int fdin, fdout;

  fdin = open("input2.dat", O_RDONLY);
  fdout = creat("output2.dat", 0644);
  //  ※ファイルが存在すれば以下open()でもOK
  //    fdout = open("output2.dat", O_WRONLY);
  while(read(fdin, buf, sizeof(buf)) > 0){     // 4.
    write(fdout, buf, strlen(buf));             // 2.
    memset(buf, '\0', sizeof(buf));             // 3.
  }
  close(fdin); close(fdout);
}
```

- open()の注意
  - 読み込みファイルの指定
    ～ 存在しなければエラー
    → 必ずエラー処理を実行
  - 書き出しファイルの指定
    ～ 存在しなくてもエラーではないが、作成されないことがある！
    → creat() を使う方が無難

  （※ ”0644”はファイルのアクセス権限）

- 繰り返しread()/write()の注意
  - 読み書き単位は「バイト」
    ～ 行末/文末の区別がない
  - 一般的な手順
    1. 適当なサイズの文字型配列を宣言
    2. 行末/文末区別なく配列読み書き
    3. その都度memset()で初期化
    4. 読み込み文字数=0で繰り返し終了

前回課題の解説
(5. パイプによる親子プロセス間通信)

```
int main(){     //  ※関係するincludeは省略
  int fd[2];
  pid_t pid, pwait;
  int status;

  pipe(fd);       //  要素数2の整数配列を引数にpipe()実行
  pid = fork();

  if(pid == 0){  // 子プロセス ls -l の実行
    close(fd[0]);  close(1);      // 不要記述子の削除
    dup(fd[1]);  close(fd[1]);  // パイプ出力のリダイレクト
    execl("/bin/ls","ls","-l", NULL);
  }
  else{         // 親プロセス grep put の実行
    pwait = wait(&status);
    printf("child  %d is finished\n", pwait);
    close(fd[1]);  close(0);      // 不要記述子の削除
    dup(fd[0]);   close(fd[0]); // パイプ入力のリダイレクト
    execl("/bin/grep","grep", "put", NULL);
  }
}
```

- [net]% ls -l | grep put の実現
  ※ ls は、「カレントディレクトリ」のファイル情報を表示、
  grep は、後続文字列を含む行を抜粋

- pipe(), fork()の実行
  1. 同じ入出力パイプを有する2プロセスが存在
  2. fork() の戻り値で、「親」「子」を判断
  3. 各プロセスで不要なファイル記述子を削除

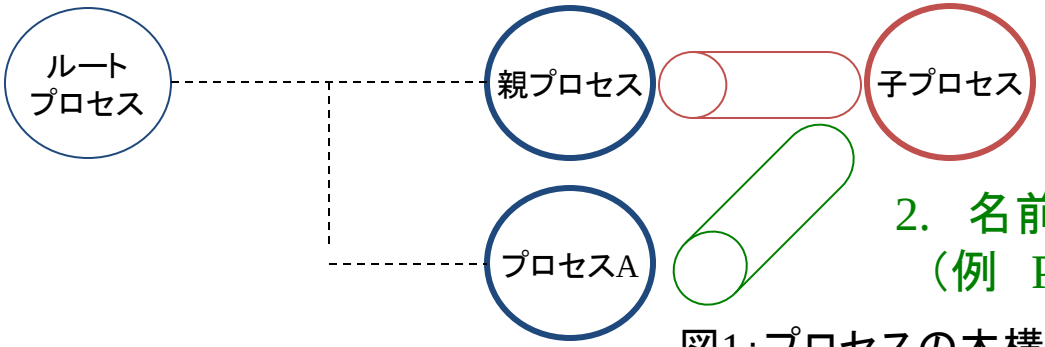
- 子プロセス
  - パイプ出力を「標準出力」にリダイレクト
  - その後、ls –l の実行

- 親プロセス
  - パイプ入力を「標準入力」にリダイレクト
  - その後、grep put の実行

マシン内のプロセス間通信

- Linux OSにおけるプロセス生成
  - あるプロセス（systemd）をルートとした「親子」関係の継承
  - 親プロセスで「分身」「変身」 → 子プロセス生成

1. 親子プロセス間通信：（名前なし）パイプ
   - パイプ（メモリの一部）を作成（pipe()）した後に分身（fork()）
   - 利点：2プロセスの挙動を1プログラムで作成可能
2. 任意プロセス間通信
   - **名前付きパイプ** ～ 任意のプロセスからアクセス可能なパイプ
   - セマフォ ～ 通信する実体への排他制御、同期の仕組みを提供
   - 共有メモリ ～ 通信する実体として主記憶を共有



1. （名前なし）パイプ pipe(), fork()

2. 名前付きパイプ
（例 PIPE）

図1：プロセスの木構造とプロセス間通信

名前付きパイプ

- 名前付きパイプ：特殊なファイル（実体はメモリ）を介したデータ送受信
- コマンド　mkfifo　～　名前付きパイプの作成

```
[net]% mkfifo –m 666 /tmp/mypipe
    //　ディレクトリ　/tmp　に　mypipe という名のパイプを作成
[net]% ls -la /tmp/mypipe
prw-rw-rw- 1 user user 0  6月 28日 17:27 /tmp/mypipe|
    //　 -m 666 で全ユーザが読み書き可能なパイプ
```

- 各プロセス：名前付きパイプを open() して読み書き
  → プロセスごとに独立したプログラムにより作成可能

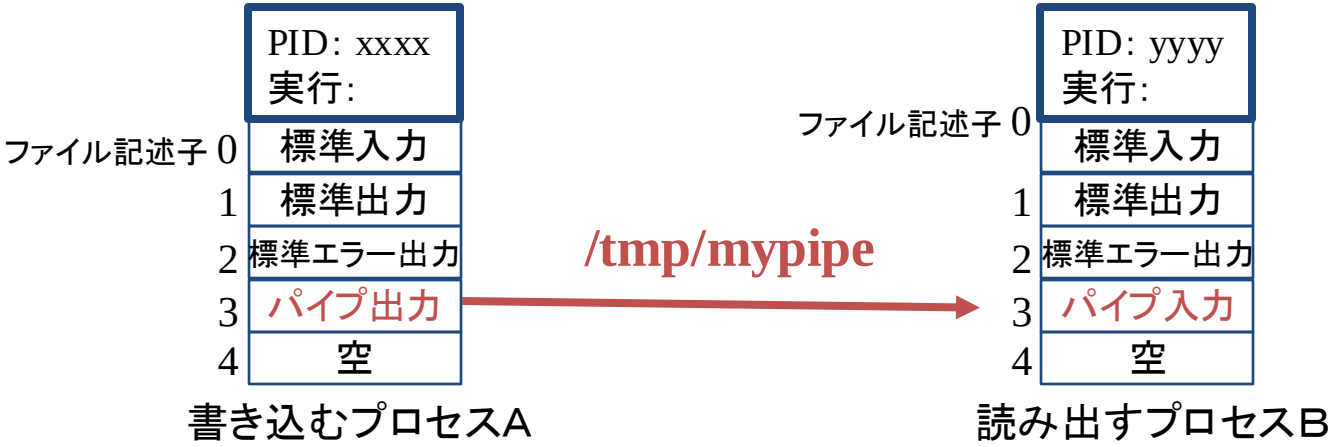


図2：名前付きパイプを介した任意のプロセス間通信

名前付きパイプ出力:プログラム例

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>  // strlen()
#include <fcntl.h>   // open(), creat()
#include <unistd.h>  // read(), write(), close()

#define PIPE "/tmp/mypipe"          // 1.
                          // 名前付きパイプの指定
int main(){
  char buf[80] = "From Server via named PIPE";
  int fd;

  fd = open(PIPE, O_WRONLY);     // 2.
  if (fd == -1){
    fprintf(stderr, "PIPE does not exist! \n");
    exit(1);
  }
  write(fd, buf, strlen(buf));              // 3.
  close(fd);                                // 4.
}
```

◆ パイプ出力の実現

～ 左プログラム(server.cとする)

1. 外部変数 PIPE の定義
   - 事前作成の/tmp/mypipe 指定

   ※ システムコール mkfifo() でもOK
2. パイプのオープン
   - ファイルオープンと同様の操作
3. パイプ出力
   - システムコールwrite() の利用
4. パイプのクローズ

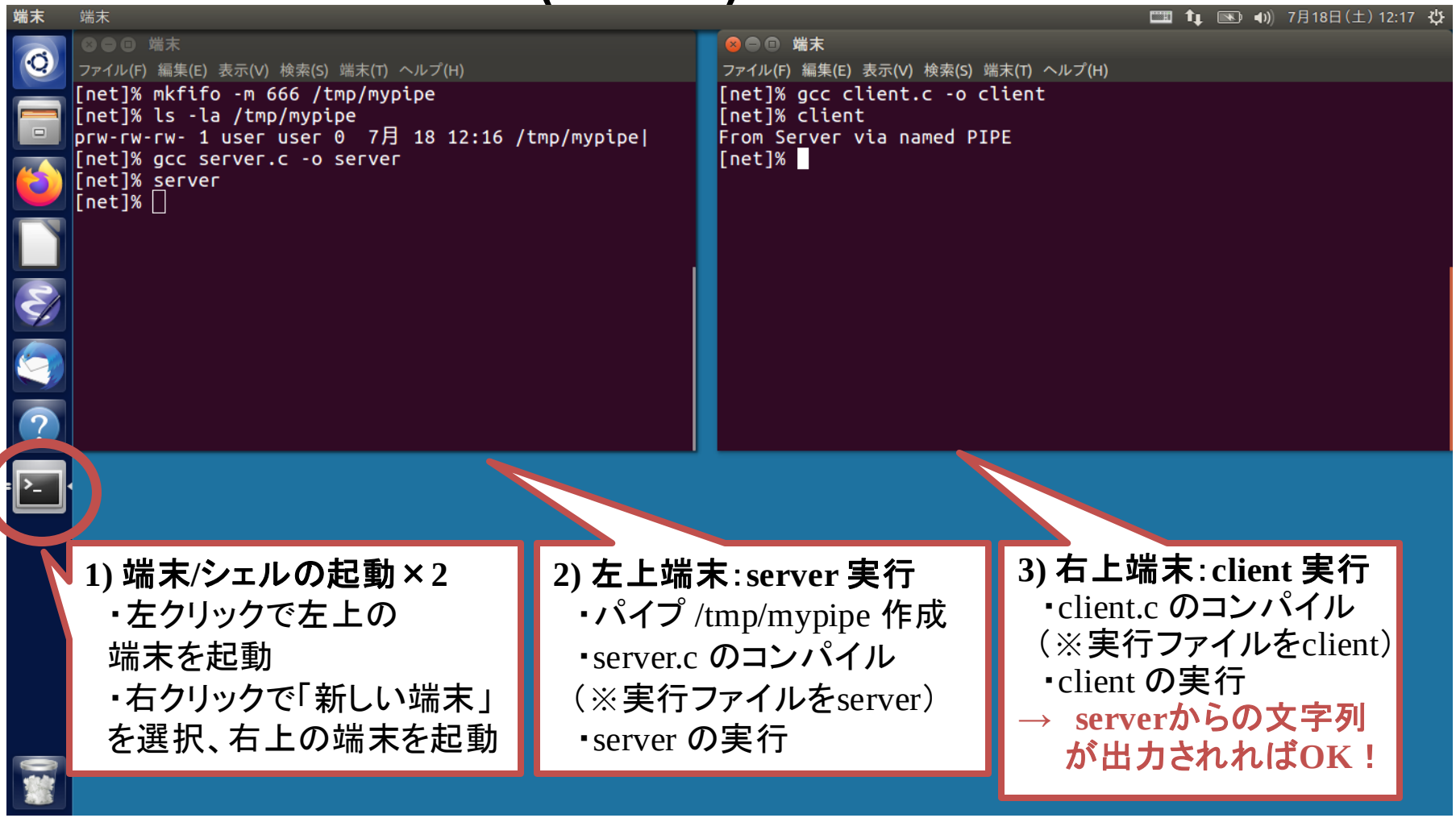
◆ パイプ入力プログラム(client.c)

- 上記手順3.を「入力」に変更
- memset() によるリセット追加

→ 演習として各自作成

※ 次スライドに実行/確認を例示

名前付きパイプ：実行例
（Ubuntu(Linux) on VirtualBox）



国立大学法人
九州工業大学

本教材は九州工業大学が作成しました
2024年度

本コンテンツについてのご質問は
数理・DS・AI教育推進室　お問い合わせ窓口
gak-kyoshien@jimu.kyutech.ac.jp
にメールでお寄せください