システムプログラミング

芝浦工業大学 情報工学科
菅谷みどり

授業計画

9/27　イントロダクション, 歴史，シェルスクリプト
10/4　C言語のライブラリ
10/11　文字列とファイル
10/18　プロセス
10/25　スレッド
11/8　プロセス2, デッドロック
11/15　研究紹介(講演会)
11/22　時刻, 割込み, シグナル
11/29　ネットワークプログラミングI クライアントプログラム
12/1　ネットワークプログラミングII　サーバプログラム
12/13　ネットワークプログラミングIII 高度な通信
12/20　グループワーク
1/10　中間報告
1/17　グループワーク
1/24　発表会，提出

文字列操作

文字、文字列のデータ表現

- **文字コード**
  - コンピュータは２進数しか扱えない
    - 文字も数として表す必要がある
  - ある数値がどの文字にあたるかの対応の決まり
- 代表的な文字コード

文字コード	説明
ASCII	アメリカで作られたアルファベット、数字、稀号を表す文字コード
ISO-2022-JP	JIS 規格により規定されている日本語文字コード
Shift-JIS	Microsoft によって作られた日本語文字コード(SJIS)
EUC-JP	AT&T によって作られた各国用のコードのうち、日本語の文字コード
UTF-8	国際的に統一的に使えるようにと策定された文字コードフォーマット

ASCII コード

- **ASCII (American Standard Code of Information Interchange)**
  - UNIX で標準的に使われてきたコード
  - 7bit で表現され、ローマ字、数字、記号、制御コードからなる

- **制御符号**

•**NUL** ヌル(空文字)	•**DC1** 制御装置1
•**SOH** ヘディング開始	•**DC2** 制御装置2
•**STX** テキスト開始	•**DC3** 制御装置3
•**ETX** テキスト終了	•**DC4** 制御装置4
•**EOT** 伝送終了	•**NAC** 否定応答
•**ENQ** 問い合わせ	•**SYN** 同期文字
•**ACK** 肯定応答	•**ETB** 伝送ブロック終了
•**BEL** ベル	•**CAN** 取消
•**BS** バックスペース	•**EM** 媒体終端
•**HT** 水平タブ	•**SUB**
•**LF/NL** 復帰/改行	•**ESC** (制御コード)拡張
•**VT** 垂直タブ	•**FS** ファイルセパレータ
•**FF** 改ページ	•**GS** グループセパレータ
•**CR** 復帰	•**RS** レコードセパレータ
•**SO** シフトアウト	•**US** ユニットセパレータ
•**SI** シフトイン	•**SP** (半角)スペース
•**DLE** データリンクでの拡張	•**DEL** 削除

EUC-JP

- EUC-JP (Extended UNIX Code)
  - EUC-JPはUNIXでは広く使われている日本語文字コードである.
    - 基本的には漢字1文字を2バイトで表すが，3バイトで表される補助漢字もある.

4ビット→↓ 上位4ビット	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A0		亜	唖	娃	阿	哀	愛	挨	姶	逢	葵	茜	穐	悪	握	渥
B0	旭	葦	芦	鯵	梓	圧	斡	扱	宛	姐	虻	飴	絢	綾	鮎	或
C0	粟	袷	安	庵	按	暗	案	闇	鞍	杏	以	伊	位	依	偉	囲
D0	夷	委	威	尉	惟	意	慰	易	椅	為	畏	異	移	維	緯	胃
E0	萎	衣	謂	違	遺	医	井	亥	域	育	郁	磯	一	壱	溢	逸
F0	稲	茨	芋	鰯	允	印	咽	員	因	姻	引	飲	淫	胤	蔭	

C言語における文字と文字列

- **システムプログラムでは、ASCII コードのみを扱う**
  - 日本語文字コードは煩雑で難しい

- **文字の取り扱い**
  - 文字：シングルクォーテーション ‘ で囲まれる ➔ ‘A’
  - 文字列：ダブルクォーテーション “ で囲まれる ➔ “A”
- **文字**
  - ‘A’ 文字は、char 型の定数
    - 値は ASCII コードにおける文字に対応した値
    - 下記の (1), (2) は同じ値を変数 c に代入している

```
char c;
c = ‘A’;    /* (1) */
c = 0x41; /* (2) */
```


文字の扱い

- **文字は数値として扱われる**
  - 演算や比較の対象となる
  - プログラム例）
    - char 型の変数 c を ++ でインクリメント(7行目）したり、<= で文字定数と比較(6行目）している

```
1 #include <stdio.h>
2
3   main()
4   {
5         char c = 'a';
6         while (c <= 'z')
7                 putchar(c++);
8         putchar('¥n');
9 }
```

文字列

- **ルール（復習）**
  - 文字の並びである文字列は、配列として表される
  - 文字列の終端を表すために、文字列の最後には 0 が置かれる

H	e	l	l	o	¥0
[0]	[1]	[2]	[3]	[4]	[5]

- **確認プログラム1**

```
1 #include <stdio.h>
2
3 char s[] = {'H', 'e', 'l', 'l', 'o', 0};
4
5 int main(void)
6 {
7     int i = 0;
8     printf("%s¥n", s);
9
10    while (s[i]) {
11            printf("[%d] = %c¥n", i, s[i]);
12            i++;
13    }
14 }
```


文字列2

```
1 #include <stdio.h>
2
3  char *ps =“Hello”;
4
5 int main(void)
6 {
7     int i = 0;
8     printf("%s¥n", ps);
9
10    while (*ps) {
11          printf("[%d] = %c¥n", i, *ps);
12          i++;
13          ps++;
13    }
14 }
```

- **ポインタの利用**
  - string1.c は、s 自体は同じメモリ位置をさし、変更できない
  - string2.c では、ps は文字列定数を指すように初期化されたポインタとなる ➔ ポインタが他の場所を指すように変更できる

標準入出力

- **C言語**
  - 標準入出力を用いることで、基本的な入出力を行うことができる
- **標準入力/出力**
  - 標準入力：キーボード
  - 標準出力：端末画面（ウィンドウ）
    - UNIX のシェルは、標準入出力をリダイレクションやパイプによって、ファイルや他のプログラムに切り替えることができる
    - この機能により、ファイルアクセスなしに様々な入出力が可能に
- **標準入力から、文字、行、書式付きの入力を行うライブラリ関数**
  - int getchar(void);
  - char * gets(char *s);
  - int scanf(const char *format, ...);
- **標準出力から、文字、行、書式付きの入力を行うライブラリ関数**
  - int putchar(int c);
  - int puts(const char *s);
  - int printf(const char *format, ...);

標準入出力

- **標準入出力を扱うライブラリ関数**
  - int fgetc(FILE *stream);
  - char * fgets(char *s, int size, FILE *stream);
  - int fscanf(FILE *stream, const char *format, ...);
  - int fputc(int c, FILE *stream);
  - int fputs(const char *s, FILE *stream);
  - int fprintf(FILE *stream, const char *format, ...);
- **標準入出力 (stream 部分の書き換え)**
  - stdin: 標準入力.. fgetc, fgets
  - stdout: 標準出力…fputc, fputs, fprintf
- **標準エラー出力**
  - stderr: 標準エラー出力
    - エラーメッセージ、警告など、例外的な処理に関するメッセージの出力のために使用される

標準入出力を扱うプログラム

- **Sample プログラム**

```
#include <stdio.h>
int main (void)
{
        int c;
        while (( c = getchar()) != EOF)
                putchar(c);
}
```

- **fgetc, fputc を使用した場合**

```
#include <stdio.h>
main()
{
       int     c;
        while ((c = fgetc(stdin)) != EOF)
                fputc(c, stdout);
}
```


標準入出力（行ごと）

```
 1  #include <stdio.h>
 2
 3  #define LINE_LEN        80
 4
 5  main()
 6  {
 7          char    line_buf[LINE_LEN];
 8
 9          while (fgets(line_buf, LINE_LEN, stdin) != NULL)
10                  puts(line_buf);
11  }
```

- **問題点**
  - リターン後に、１行空白ができてしまう
    - fgets は改行文字もバッファに読み込む
    - fputs も改行も出力する仕様
      - １行空白が発生してしまう

課題1

- **問題点を改善せよ**

文字と文字列操作ライブラリ

- **大文字または小文字に変換する関数**

```
int toupper (int c);   /* 大文字へ変換 */
int tolower (int c);   /* 小文字へ変換 */
```

- **文字の種類の判別**

```
int isalnum (int c);   /* 英字又は数字？ */
int isalpha (int c);   /* アルファベット？ */
int isascii (int c);   /* アスキー文字？ */
int isblank (int c);   /* 空白文字（スペース又はタブ）？ */
int iscntrl (int c);   /* 制御文字？ */
int isdigit (int c);   /* 数字？ */
int isgraph (int c);   /* 表示可能？（スペースは含まれない） */
int islower (int c);   /* 小文字？ */
int isprint (int c);   /* 表示可能？（スペースを含む） */
int ispunct (int c);   /* 表示可能？（スペースと英数字を除く） */
int isspace (int c);   /* 空白文字？（スペース，タブ，改行文字など） */
int isupper (int c);  /* 大文字？ */
int isxdigit (int c);  /* 16進数での数字？（0～9, a～f, A～F） */
```

小文字と大文字の変換

```
 1  #include <stdio.h>
 2  #include <ctype.h>
 3
 4  main()
 5  {
 6          int     c;
 7
 8          while ((c = getchar()) != EOF) {
 9                  if (islower(c))
10                          c = toupper(c);
11                  else if (isupper(c))
12                          c = tolower(c);
13                  putchar(c);
14          }
15  }
```


文字列操作

- **string (3) に文字列操作のためのライブラリ関数一覧がある**
- **man 3 string**
  - 関数のリストに含まれて

  #include <strings.h>
  #include <string.h>
  が表示される
  - これらの関数を利用する時には、このヘッダファイルをインクルードしなさい、という意味
- **文字列の長さを調べるライブラリ関数**
  - size_t strlen (const char *s);
    - 戻り値は、文字列の長さ。終端文字は含まれないため、strlen(“abc”) は 3 を返す

課題2：大文字小文字変換プログラムを作成する

- **満たすべき要求**
  - １行ごとに標準入力を読み込む
  - 大文字だったら小文字へ変換(ライブラリ参照)
  - 小文字だったら大文字へ変換(ライブラリ参照)
  - 文字は先頭からみてゆく
    - 先頭から文字をみてゆくためには、文字列の長さが必要になるため、strlen を利用する

文字列の比較

- **比較のためのライブラリ関数**

```
int    strcmp(const char *s1, const char *s2);
int    strncmp(const char *s1, const char *s2, size_t n);
int    strcasecmp(const char *s1, const char *s2);
int    strncasecmp(const char *s1, const char *s2, size_t n);
```

- **二つの文字列s1, s2 を比較し、下記の結果を返す**

条件	戻り値
s1 < s2	0 より小さい値
s1 == s2	0
s1 > s2	0 より大きい値

  - 文字列の大小関係は ASCII コードである
- **違い**
  - strncmp, strncasecmp は, s1の先頭 n 文字について比較を行う
  - Strcasecmp, strncasecmp は大文字、小文字を区別しない

文字や文字列の比較

- **文字列の比較のためのライブラリ関数**

```
char * strchr(const char *s, int c);
char * strrchr(const char *s, int c);
char * index(const char *s, int c);
char * rindex(const char *s, int c);
char * strstr(const char *haystack, const char *needle);
```

- **違い**
  - strchr, index は、文字列 s を先頭から探して、最初に c の文字が現れたところに、ポインタを返す
  - strchr, rindex は、文字列 s を最後尾から探して、最初にc の文字が現れたところへのポインタを返す
  - strstr は, 文字列 haystack を先頭から探して、最初に needle が見つかったところへ、ポインタを返す
  - どれも見つからなかった場合は、NULL が戻り値になる

ファイルパスの構成要素を返す

```
 1  #include <stdio.h>
 2  #include <string.h>
 3  #include <sys/param.h>
 4
 5  main()
 6  {
 7          int     i;
 8          char    line_buf[MAXPATHLEN];
 9          char    *p, *np;
10
11          while (fgets(line_buf, MAXPATHLEN, stdin) != NULL) {
12                  i = 0;
13                  p = line_buf;
14                  while ((np = index(p, '/')) != NULL) {
15                          *np = '¥0';
16                          printf("%d: %s¥n", i++, p);
17                          p = np + 1;
18                  }
19                  printf("%d: %s¥n", i, p);
20          }
21  }
```


プログラムの実行結果

- **プログラムの注意**
  - 8行目の MAXPATHLEN はシステムで定義されているパス名の最大長、sys/param.h で定義されている
- **プログラムの実行結果**

```
% ./filepath
home/doly/file
0: home
1: doly
2: file
/home/doly/file/
0:
1: home
2: doly
3: file
4:
```


文字列のコピーと連結

- **コピーのためのライブラリ関数**

```
char * strcpy(char *dest, const char *src);
char * strncpy(char *dest, const char *src, size_t n);
```

- **違い**
  - stcpy は, src の文字列を終端文字 0 も含めて dest にコピーする, strncpy は最大 n 文字コピーする
  - strncpy はコピーした n 文字に、終端文字 0 が含まれるかどうかはチェックしない、しかし src の文字列が n 文字よりも短かった場合、dest の残りの部分は 0 である

- **連結のためのライブラリ関数**

```
char * strcat(char *dest, const char *src);
char * strncat(char *dest, const char *src, size_t n);
```

- **違い**
  - strcat は、src の文字列を dest の文字列の後に (dest の終端文字列のところから）コピーし、最後に終端文字列 0 を追加する
  - strncat は、src の文字列を n 文字だけコピーするところが異なる

その他の文字列操作関数

```
char * strdup(const char *s);        /* 文字列の複製 */
char * strfry(char *string);         /* 文字列のランダム化 */
char * strsep(char **stringp, const char *delim);
/* トークンの切り出し */
char * strtok(char *s, const char *delim);
/* トークンへの分解 */
size_t strcspn(const char *s, const char *reject);
/* 文字セットに含まれない文字数 */
char * strpbrk(const char *s, const char *accept);
/* 文字セットに含まれる文字の検索 */
size_t strspn(const char *s, const char *accept);
/* 文字セットに含まれる文字数 */
int    strcoll(const char *s1, const char *s2);
/* ロケールに基づく文字列比較 */
size_t strxfrm(char *dest, const char *src, size_t n);
/* ロケールに基づいた文字列変換 */
```


文字列と数値の変換

- **文字列を数値に変換するライブラリ関数**

```
long int      strtol(const char *nptr, char **endptr, int base);
unsigned long int strtoul(const char *nptr, char **endptr, int base);
double     strtod(const char *nptr, char **endptr);
long          atol(const char *nptr);
int           atoi(const char *nptr);
double      atof(const char *nptr);
int           sscanf(const char *str, const char *format, ...);
```

- **数値を文字列に変換するライブラリ関数**

```
int sprintf(char *str, const char *format, ...);
int snprintf(char *str, size_t size, const  char  *format, ...);
```


ファイル操作

UNIXの基本

- **全てのデータはファイルとして入出力で扱われる**

ファイルアクセス

- **ファイルアクセスをする方法には２つある**
  - ライブラリ関数を用いる方法
    - fopen, fclode, fread, fcloseetc.. データをFILE構造体（ストリーム）として扱う
  - システムコールを用いる方法
    - Open,close, read/write, .. データを低レベルなメモリとして扱う
- **ファイル操作に共通する手順**
  - ファイルを開く (open)
  - 読み書きを行う (read, write)
  - ファイルを閉じる (close)
- **課題は文字列操作の参考資料を参考にしてすすめる**

ライブラリ関数：ファイルとの入出力

- **ファイルとの入出力を行うライブラリ関数**

```
FILE * fopen(const char *path, const char *mode);
int    fclose(FILE *stream);
size_t fread(void  *ptr,  size_t size, size_t nmemb, FILE *stream);
size_t fwrite(const void *ptr, size_t size, size_t  nmemb, FILE
*stream);
```

- **ファイル操作**
  - ファイルの開閉: fopen/fclose
  - 読み書き：先頭に f がついた関数
  - FILE *stream を stdin, stdout の代わりに fopen の戻り値として得られれば良い
    - ファイルからの入出力に便利なように, fread, fwrite という関数もある

FILE 構造体

- **stdio.h に定義されている**
  - Linux では /usr/include 以下
- **ファイルに関する情報を持つ**
  - 現在読み書きしている位置（=ファイルポインタ）
  - ファイルに対して許される操作
    - 読み込み、書き込み、または両方
  - 現在アクセスしているファイル
  - 発生しているエラー
- **ファイルポインタ**
  - ファイル構造体へのポインタ
  - 入出力にあたり、ファイルポインタを指定して読み書きできるようになっている

FILE 構造体

- **Stdio.h で定義されている（構成は処理系により異なる）**
- typedef struct {

```
    char mode;        // アクセスモード(r, w, r/w)
    char *ptr;        //読み書きしている位置（先頭からのバイト数)
    int rcount;       //ptrの示す読み位置からファイル終端までのバイト数
    int wcount;
    char *base;       //メモリ上に確保したファイルアクセス用バッファの先頭位置
    unsigned bufsiz;  //ファイルアクセス用バッファのサイズ
    int fd;           //ファイルディスクリプタ
    char smallbuf[1];
} FILE;
```

標準入出力

- **標準入出力を扱うライブラリ関数**
  - int　fgetc(FILE *stream);
  - char * fgets(char *s, int size, FILE *stream);
  - int　fscanf(FILE *stream, const char *format, ...);
  - int　fputc(int c, FILE *stream);
  - int　fputs(const char *s, FILE *stream);
  - int　fprintf(FILE *stream, const char *format, ...);
- **標準入出力 (stream 部分の書き換え)**
  - stdin: 標準入力.. fgetc, fgets
  - stdout: 標準出力…fputc, fputs, fprintf
- **標準エラー出力**
  - stderr: 標準エラー出力
    - エラーメッセージ、警告など、例外的な処理に関するメッセージの出力のために使用される

課題1

- **ファイルデータの調査コマンドを作成しよう**
  - コマンド引数で渡された、
    - ファイルの行数、
    - 文字数(バイト),
    - 単語数
  - を数えるコマンドを作成せよ
- **仕様**
  - 単語は、アルファベットで始まる１文字の欧文単語のみ数える
    - 例) #include <stdio.h> の場合には、include, stdio, hの３単語
  - エラー処理も行う

ヒント

- **基本的な処理内容に分割して組み立てる**
  - 引数をファイル名で渡す
    - エラー処理
  - 渡されたファイルを開く
    - fopen().. どのような関数か？
      - FILE *fp = **fopen**(filename, "r");
    - 閉じるも一緒に書いておく fclode()
  - ファイルポインタから、文字を読み込む
    - 読み込む場合には、ファイルポインタからの標準入力
      - for (int ch = **fgetc**(fp); ch != EOF; ch = fgetc(fp)) {}
  - 読み込んだ文字に対して何をすれば良いか？
    - 文字数のカウント（文字の個数）
    - 行数のカウント（改行の個数）
    - 単語のカウント
      - ヒント：isalpha(ch)

課題2

- **ファイルコピーコマンドを作成しよう**
  - コマンド引数で２つのファイル名を渡し、１つ目のファイルから、２つ目のファイルにその内容をコピーする
    - 例）./copy [file1] [file2]
    - File1からfile2 に内容をコピーする
  - 引数が少ない場合、コピーできなかった場合にはエラーを返す

main 関数への引数

- **ファイルコピープログラムの問題点**
  - ファイル名を変更するたびに、コンパイルしなくてはならない
- **改善方法**
  - プログラムの実行時に、ファイル名をコマンド行の引数として渡す
    - Main 関数への引数として渡す
- **コマンド行の引数を出力するプログラム**

```
1 #include <stdio.h>
2  int main(int argc, char *argv[])
3  {
4       int i;
5       for (i = 0; i < argc; i++)
6                 puts(argv[i]);
7   }
```

- **コンパイル&実行結果**

```
% ./a.out src dst
./a.out
src
dst
```


ヒント

- **基本的な処理内容に分割して組み立てる**
  - 引数をファイル名で渡す
    - エラー処理
  - 渡された2つのファイルを開く
    - fopen()
    - FILE *src = fopen(filename, “r”);
    - FILE *dst = fopen(filename, “w+”);
    - エラーの場合には、閉じる
      - 一緒に書いておく fclode()
  - ファイルポインタから、文字を読み込み、ファイルポインタに書き込む

```
while ((c = fgetc(src)) != EOF)
      fputc(c, dst);
```


システムコールを用いたファイルとの入出力

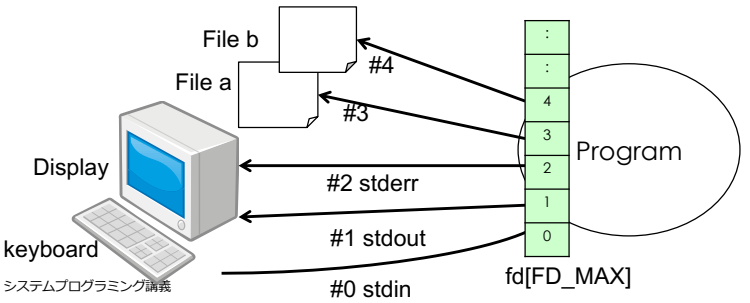
- **システムコールを用いたファイルの開閉**
  - 開閉：open, close
  - 読み書き：read, write

```
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
int close(int fd);
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
```

- ライブラリ関数との違い
  - オープンに成功すると
    - fopen → FILE 構造体へのポインタを返す.
      - その後はオープンしたファイルへのアクセスは、ポインタを介して行う
    - open → ファイルディスクリプタ（fd）を返す
      - オープンしたファイルのアクセスは、ファイルディスクリプタ (int fd) を介して行う、アクセスが終了したら、ファイルディスクリプタを引数に close を呼ぶことで、ファイルをクローズする

ファイルディスクリプタ

- **ファイル記述子 (File Descriptor)**
  - オープン中ファイルの詳細を記録しているカーネル内データ構造（配列）へのインデックス
    - システムコール経由でわたし、カーネルはそのキーに対応するファイルにアクセスする
    - POSIX では, 0 = 標準入力(stdin), 1=標準出力、2=標準エラー出力(stderr) とされており、通常のファイルは 3以降を利用する
  - 整数 (C言語の int 型)
  - 各プロセスは、それぞれ自分のファイル記述子のテーブルを持つ
    - 最大数は制限されている。現在は /proc/sys/fs/file-max (1024)



課題3

- **ファイルコピーコマンド(システムコール版）を作成しよう**
  - システムコール
    - open, read, write, close
  - コマンド引数で２つのファイル名を渡し、１つ目のファイルから、２つ目のファイルにその内容をコピーする
    - 例）./copy_syscall [file1] [file2]
    - File1からfile2 に内容をコピーする
    - 引数が少ない場合、コピーできなかった場合にはエラーを返す

興味がある人

- **Open system call の hack**
  - **Linuxのファイルディスクリプタをハックする**
  - (@tajima_tatsuo)

プログラムの補足

- **ファイルをオープンするモード**
  - O_RDONLY: 読み込みのみ
  - O_WRONLY : 書き込みのみ
  - O_RDWR : 読み書き
  - O_CREAT : ファイルが存在しなければ作成、ファイルが作られた場合のパーミッションの指定が、３つ目にくる
  - O_APPED : ファイルに追加する
  - O_TRUNK : ファイルが既に存在した場合は、ファイルの長さはゼロになる
- **read/write**
  - rcount = read(src, buf, max),
    - max: 読み込める最大バイト数 src から buf へ読み込んで、読んだバイト数だけ rcount に返す
  - wcount = write(dst, buf, rcount)
    - 読み込んだバイト数を buf から dst に書き出して、書き出したバイト数を wcount に返す
  - ※最も効率の良い値は入出力先のデバイス，デバイスを制御するコントローラ，メモリの量などに依存する

課題4

1. **システムコールとライブラリコールのコピープログラムを二つ作成し、100M以上のファイルにてコピーの性能(時間）を比較せよ**
   - 同じ100MBのファイルを作成し，それをコピーする
     - ※ 大きなサイズのファイルの作成方法
2. **システムコールを用いてファイルコピーを行うプログラムについては，バッファサイズを変更して、それぞれの実行時間の結果を記載せよ**

- 計測方法
  - time コマンドを用いて行う
    - ユーザモード、カーネルモード、合計それぞれの実行時間が表示される
- **例）ファイルの時間の計測**

```
time ./sycallbig  src dst
1.380u 19.885s 0:21.49 98.9%    0+0k 0+0io 0pf+0w
```

ユーザ1.38秒, カーネル 19.9秒, 合計 21.45秒

大きなサイズのファイルの作成方法

- **$ echo “1234567890” >> srcbig.txt　で小さいファイルを作成**
- **$ sh (シェルを起動する)**
  - while 文をまわして，ファイルを大きくする（下記の構文）
- **$ while [1]; do cat srcbig.txt >> srcbig2.txt ; ls -lh srcbig2.txt; sleep 1 ; done**

課題の提出

- **文字列　課題1,2**
- **ファイル 課題1,2,3,4 を Exec04フォルダに提出する**

ライブラリとシステムコールの混在

- **同一ファイルのアクセス**
  - ライブラリとシステムコールで混ぜて行うことは避けるべき
    - 理由：ライブラリ関数の入出力は、一文字単位の入出力も効率よく行えるように、入出力データを一時的にバッファリングすることで，システムコールの回数を減らす仕様となっているため