MICS 実験第一 J4 課題レポート

学籍番号 2210342, 鈴木謙太郎

2024 年 7 月 22 日

1 課題 1

まず,write システムコールを直接用いるコード 1 のような mycp 関数を作成した.

Code 1: mycp 関数のソースコード

```c
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define BUFFER_SIZE 128

void mycp(const char *src, const char *dst, size_t buffer_size) {
  int src_fd = open(src, O_RDONLY);
  if (src_fd == -1) {
    perror("Error opening source file");
    exit(EXIT_FAILURE);
  }

  int dst_fd = open(dst, O_WRONLY | O_CREAT | O_TRUNC, 0644);
  if (dst_fd == -1) {
    perror("Error opening destination file");
    close(src_fd);
    exit(EXIT_FAILURE);
  }

  char *buffer = (char *)malloc(buffer_size);
  if (buffer == NULL) {
    perror("Error allocating buffer");
    close(src_fd);
    close(dst_fd);
    exit(EXIT_FAILURE);
```

```
28    }
29
30    ssize_t bytes_read;
31    while ((bytes_read = read(src_fd, buffer, buffer_size)) > 0) {
32      if (write(dst_fd, buffer, bytes_read) != bytes_read) {
33        perror("Error writing to destination file");
34        free(buffer);
35        close(src_fd);
36        close(dst_fd);
37        exit(EXIT_FAILURE);
38      }
39    }
40
41    if (bytes_read == -1) {
42      perror("Error reading from source file");
43    }
44
45    free(buffer);
46    close(src_fd);
47    close(dst_fd);
48  }
49
50  int main(int argc, char const *argv[]) {
51    if (argc != 3) {
52      fprintf(stderr, "Usage: %s <source> <destination>\n", argv[0]);
53      return 1;
54    }
55    char const *source = argv[1];
56    char const *destination = argv[2];
57    mycp(source, destination, BUFFER_SIZE);
58    return 0;
59  }
```

また, この mycp 関数の性能を評価する際の対照として, コード 2 のような標準ライブラリ関数を用いたコピー関数を作成した.

Code 2: 比較用のコピー関数のソースコード

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void copy_file_stdio(const char *src, const char *dst) {
5    FILE *src_file = fopen(src, "rb");
```

```
6   if (src_file == NULL) {
7     perror("Error opening source file");
8     exit(EXIT_FAILURE);
9   }
10
11  FILE *dst_file = fopen(dst, "wb");
12  if (dst_file == NULL) {
13    perror("Error opening destination file");
14    fclose(src_file);
15    exit(EXIT_FAILURE);
16  }
17
18  int ch;
19  while ((ch = fgetc(src_file)) != EOF) {
20    if (fputc(ch, dst_file) == EOF) {
21      perror("Error writing to destination file");
22      fclose(src_file);
23      fclose(dst_file);
24      exit(EXIT_FAILURE);
25    }
26  }
27
28  fclose(src_file);
29  fclose(dst_file);
30 }
31
32 int main(int argc, char const *argv[]) {
33   if (argc != 3) {
34     fprintf(stderr, "Usage: %s <source> <destination>\n", argv[0]);
35     return 1;
36   }
37   char const *source = argv[1];
38   char const *destination = argv[2];
39   copy_file_stdio(source, destination);
40   return 0;
41 }
```

これらの関数を用いて, コピー対象とするファイルのサイズを $16^1 \sim 16^9$ バイトの間で変えながら, それぞれの関数の性能を評価した. 性能評価には,GNU time コマンドを用い, 実行時間を 10^{-2} 秒単位で測定した.

まず,mycp 関数の測定結果は図 1 のようになった.

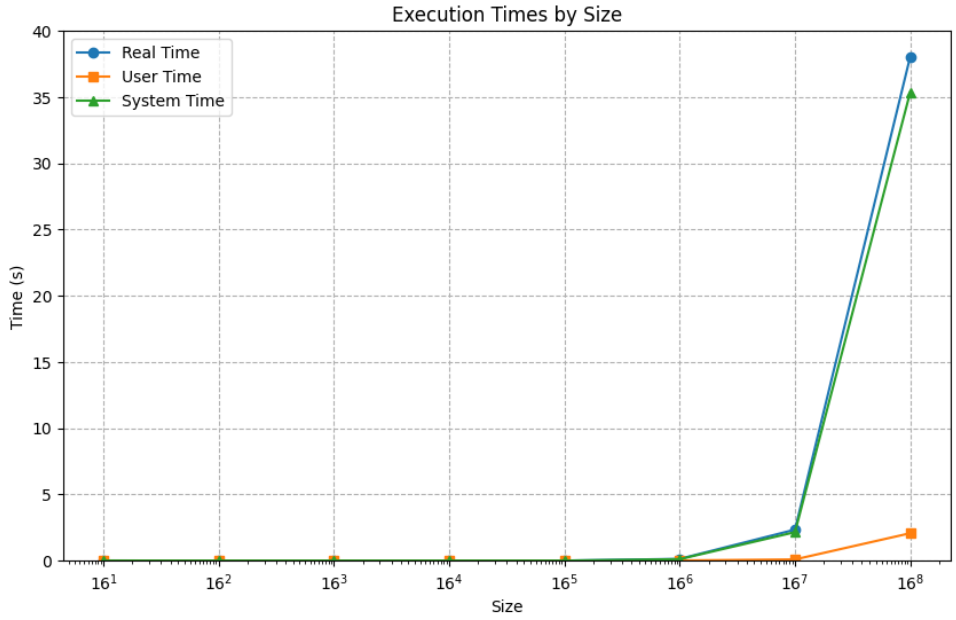

図 1: mycp 関数の性能測定結果

次に, 標準ライブラリ関数を用いたコピー関数の測定結果は図 2 のようになった.

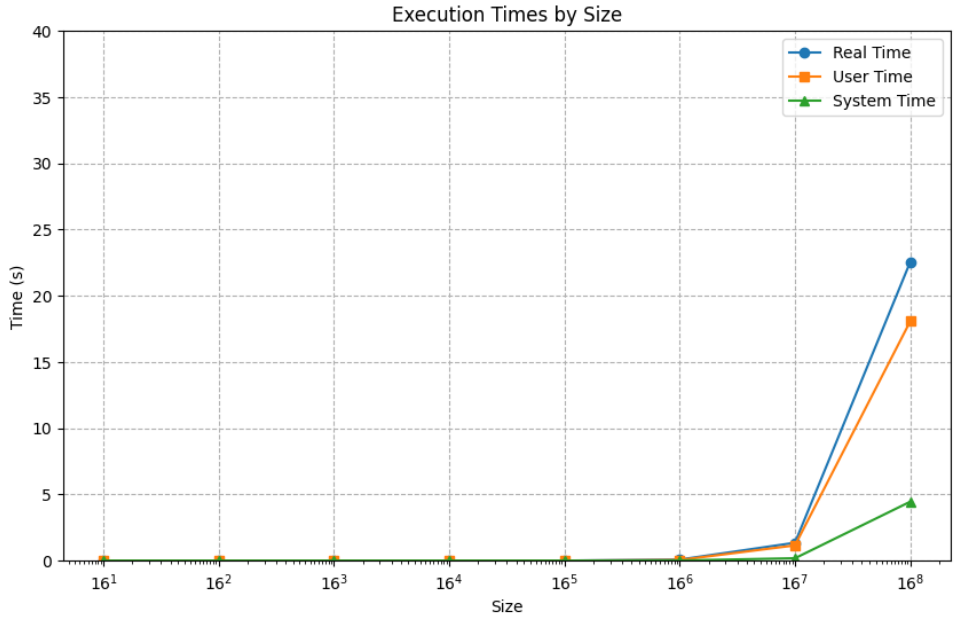

図 2: 標準ライブラリ関数を用いたコピー関数の性能測定結果

2 つの関数の測定結果で共通している点は, ファイルサイズが 16^6 バイト (1 メガバイト) 程度までは 10^{-2} 秒単位での実行時間がほぼ一定であることである. また, ファイルサイズが 16^7 バイト (10 メガバイト) 程度を超えると, どちらの関数も実行時間が time コマンドで測定できる程度になることがわかる.

一方で, 実行時間が増えてきてからの傾向については,mycp 関数と標準ライブラリ関数を用いたコピー関数で異なる点がみられる. 独自に作成した mycp 関数の実行時間は system time が占める割合が大きいが, 標準ライブラリ関数を用いたコピー関数の実行時間は user time が占める割合が大きい.

この違いとしては, システムコールの回数の違いが考えられる. mycp 関数は,read と write のシステムコールを用いてファイルのコピーを行っているため, システムコールの回数が多くなる. 一方で, 標準ライブラリ関数を用いたコピー関数は, ファイルの読み書きを行う際にバッファリングを行っているため, システムコールの回数が少なくなる. fgetc/fputc は 1 バイトずつ読み取りと書き込みを行っているが, 内部的にはある程度大きなバッファで読み書きを行っているため, システムコールの回数が少なくなっていると推測される.

このような結果, システムコールの回数が異なっているので,2 つの関数において実行時間の内訳が異なると考えられる.

2 課題 2

可変長引数としてホスト名を複数渡すと, それらに対して ping を実行する関数を作成する.

まず, 指定されたホストに対して並行で ping を実行する関数をコード 3 のように作成した.

Code 3: ping を並行して実行する関数

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
  if (argc < 2) {
    fprintf(stderr, "Usage: %s host1 host2 ...\n", argv[0]);
    return 1;
  }

  for (int i = 1; i < argc; ++i) {
    pid_t pid = fork();

    if (pid == -1) {
      // フォーク失敗
      perror("fork failed");
      return 1;
    } else if (pid == 0) {
      // 子プロセス
      execlp("ping", "ping", "-c", "3", argv[i], (char *)NULL);
      perror("execlp failed");
      exit(1);
    } else {
      // 親プロセス
```

```
26        // 何もしない
27      }
28    }
29
30    // 親プロセスは全ての子プロセスが終了するのを待つ
31    for (int i = 1; i < argc; ++i) {
32      wait(NULL);
33    }
34
35    return 0;
36  }
```

同様に, 指定されたホストに対して直列に ping を実行する関数をコード 4 のように作成した.

Code 4: ping を並行して実行する関数

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/wait.h>
4  #include <unistd.h>
5
6  int main(int argc, char const* argv[]) {
7    if (argc < 2) {
8      fprintf(stderr, "Usage: %s host1 host2 ...\n", argv[0]);
9      return 1;
10   }
11
12   for (int i = 1; i < argc; i++) {
13     pid_t pid = fork();
14
15     if (pid == -1) {
16       // フォーク失敗
17       perror("fork");
18       return 1;
19     } else if (pid == 0) {
20       // 子プロセス
21       execlp("ping", "ping", "-c", "3", argv[i], (char*)NULL);
22       perror("execvp");
23       exit(1);
24     } else {
25       // 親プロセス
26       int status;
27       waitpid(pid, &status, 0);  // 子プロセスの終了を待つ
```

```
28      }
29    }
30
31    return 0;
32  }
```

これらの関数に対して,9 つのホスト名を指定し, 実行にかかった時間を GNU time コマンドで測定した結果は図 3 のようになった. 並行に実行した結果が parallel, 直列に実行した結果が serial である.

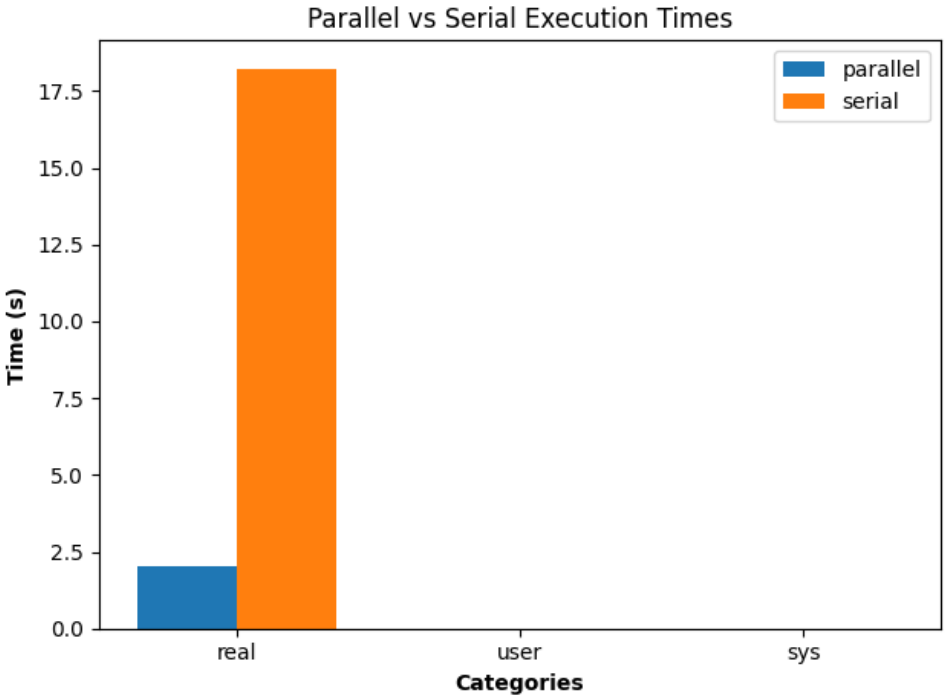

図 3: ping を並行して実行する関数と直列に実行する関数の性能測定結果

実行時間のほぼすべてが user time で占められているのは,ping コマンドを実行して外部のレスポンスを待っていることが大半だからである. また, 並行に実行した場合の方が直列に実行した場合よりも実行時間が短いことがわかる.

今回は 9 つのホスト名を指定してそれぞれの関数を実行したが, 並列に実行した場合が直列に実行した場合より 9 倍速いわけではなかった. 並列実行した場合, 最もレスポンスが返ってくるのに時間を要したホストへの ping コマンドの実行時間が全体の実行時間に大きく影響する. このため, 並列実行した場合の実行時間が直列実行した場合の 1 ホストあたりの実行時間よりも短いとは限らないと考えられる.

3 課題 3

課題で与えられた, コマンドを実行し標準入力に与えられたデータと同じものをログファイルに控えるプログラムをコード 5 のように作成した.

Code 5: コマンドを実行し標準入力に与えられたデータと同じものをログファイルに控えるプログラム

```c
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
  if (argc < 3) {
    fprintf(stderr, "Usage: %s logfile command [args...]\n", argv[0]);
    return 1;
  }

  char *logfile = argv[1];
  char *command = argv[2];
  char **args = &argv[3];

  int pipefd[2];
  if (pipe(pipefd) == -1) {
    perror("pipe");
    return 1;
  }

  pid_t pid = fork();

  if (pid == -1) {
    // フォーク失敗
    perror("fork");
    exit(1);
  } else if (pid == 0) {
    // 子プロセス
    close(pipefd[1]);
    dup2(pipefd[0], STDIN_FILENO);  // リダイレクトする
    close(pipefd[0]);
    execvp(command, args);
    perror("execvp");
    exit(1);
  } else {
    // 親プロセス
    close(pipefd[0]);
```

```
41      int logfd = open(logfile, O_WRONLY | O_CREAT | O_TRUNC, 0644);
42      if (logfd == -1) {
43        perror("open");
44        exit(1);
45      }
46
47      char buffer[1024];
48      ssize_t bytes;
49      while ((bytes = read(STDIN_FILENO, buffer, sizeof(buffer))) > 0) {
50        if (write(pipefd[1], buffer, bytes) == -1) {
51          perror("write to pipe");
52          exit(1);
53        }
54        if (write(logfd, buffer, bytes) == -1) {
55          perror("write to logfile");
56          exit(1);
57        }
58      }
59
60      if (bytes == -1) {
61        perror("read");
62      }
63
64      close(pipefd[1]);  // Close write end of pipe
65      close(logfd);      // Close logfile
66
67      // Wait for child process to finish
68      wait(NULL);
69    }
70
71    return 0;
72  }
```

このコードをコンパイルして, 実験資料で示された./a.out stdin_log more -10 < /usr/include/X11/Xlib.h のようなコマンドを実行した.

動作結果としては,more コマンドで表示した内容をどこまで見るかによって, ログファイルに書き込まれる量も異なるというようになった. このような結果になる理由として,pipe() の動作の仕組みが関係していると推測した.

pipe() は, 親プロセスと子プロセス間でデータをやり取りするためのパイプを作成するシステムコールである. 子プロセスはパイプの出力を標準入力にリダイレクトし, 親プロセスは標準入力から受け取った内容をパイプの入力とログファイルに書き込んでいる. このため,more コマンドで標準入力を読み込んだ分だけパイプに入力され, ログファイルに書き込まれるということになる.

実際に,more コマンドで表示する行数を変えて実行した結果, すべての行を表示した場合はログファイルにもすべての行が書き込まれ, 元ファイルと diff コマンドで比較しても同じであることが確認できた. 一方で途中の行までしか表示しなかった場合はログファイルにも途中までの行しか書き込まれなかった.

このような結果から,UNIX が提供しているパイプはそのとき必要な分だけ IO アクセスを行うことで, 高速にプログラムを動作させる工夫がなされていると考えられる.