ネットワークプログラミング 4

- 前回は、UDPを使用したINADDR_ANYで受信していた。
    - 複数からの受信を行うことが可能だった。
- TCPでの受信を行う場合、同様なことが可能だろうか？

マルチスレッドプログラミングを理解しよう

- 一般的にマルチプロセスとマルチスレッドの違いはなんですか？
    - 簡単にまとめてください。
        - それぞれのメリット・デメリット
        - 陥りやすい失敗など
        - 実際に使用されているそれぞれ例と特徴

マルチスレッドの例

- 今回は、仮想マシンであることを考慮し、マルチスレッドでの理解を深めよう。

【ex00.c】

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

void *ThreadA() {
    printf("thread A start\n");
    for (int i = 0; i < 100; ++i) {
        printf("A-%d\n", i);
    }
    printf("thread A end\n");
}

void *ThreadB() {
    printf("thread B start\n");
    for (int i = 0; i < 100; ++i) {
        printf("B-%d\n", i);
    }
```

```
    printf("thread B end\n");
}

int main(int argc, char const *argv[]) {
    pthread_t th_a;
    pthread_create(&th_a, NULL, ThreadA, NULL);

    pthread_t th_b;
    pthread_create(&th_b, NULL, ThreadB, NULL);

    pthread_join(th_a, NULL);
    pthread_join(th_b, NULL);

    return 0;
}
```

- 基本的なスレッドを用いた例です
    - 初見のヘッダファイル・関数について調査を行ってください。
        - 使い方、引数の意味、動作などについて

うまく動作しない例

【ex01.c】

```
// https://qiita.com/nsnonsugar/items/be8a066c6627ab5b052a
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

unsigned int count_;

void *ThreadA() {
    for (int i = 0; i < 100000; ++i) {
        ++count_;
    }
}

void *ThreadB() {
    for (int i = 0; i < 100000; ++i) {
        ++count_;
    }
}

int main(int argc, char const *argv[]) {
    count_ = 0;
```

```
    pthread_t th_a;
    pthread_create(&th_a, NULL, ThreadA, NULL);

    pthread_t th_b;
    pthread_create(&th_b, NULL, ThreadB, NULL);

    pthread_join(th_a, NULL);
    pthread_join(th_b, NULL);

    printf("count_ : %d\n", count_);

    return 0;
}
```

- なぜ正常に動作しないのでしょうか？

  > https://qiita.com/nsnonsugar/items/be8a066c6627ab5b052a

  この記事ではC++を使用した例なので、このままでは動作しません。
  上記 `ex01.c` は、C用に書き換えていますので、動作します。
- どうすれば、正常に動作させることができますか？
  - 以下について理解してください。
    - セマフォ
    - ミューテックス
- 上記の `ex01.c` をmutexを使用して正常に動作するようにしてください。
  - 以下は記事中のmutexを使用した例です（c++）

```
#include <cstdint>
#include <iostream>
#include <mutex>
#include <thread>

std::mutex mtx_; // 排他制御用ミューテックス
uint32_t count_;

void add_count()
{
    // count_を加算する前にミューテックスを取得する
    std::lock_guard<std::mutex> lock(mtx_);
    ++count_;
}

void ThreadA()
```

```
{
    for(int i=0; i<100000; ++i){
        add_count();
    }
}

void ThreadB()
{
    for (int i = 0; i<100000; ++i) {
        add_count();
    }
}

int main()
{
    count_ = 0;

    std::thread th_a(ThreadA);
    std::thread th_b(ThreadB);

    th_a.join();
    th_b.join();

    std::cout << "count_ : " << count_ << std::endl;

    return 0;
}
```

処理をうまく分けることができる場合

- 処理によっては、MutexやSemophoreを使用しなくても、正常に動作させることが可能な場合があります。
    - もちろん、少しの工夫と、扱う内容によります。

基本のプログラム

- データを初期化するだけのプログラム

【ex02.c】

```
#include <stdio.h>

#define DATA_SIZE 100000

long data[DATA_SIZE];
```

```
void init_function(void) {
    for (int i = 0; i < DATA_SIZE; i++) {
        data[i] = i;
    }
    return;
}

int main(int argc, char const *argv[]) {
    init_function();
    return 0;
}
```

- 内容は、10万個の配列を初期化するプログラム
    - data[0] = 0 , data[1] = 1, data[2] = 2, ... ,data[99999] = 99999

- 上記のプログラムは、少々速すぎるのでwaitをかけて遅くなるようにする
    - `usleep()` を使用します。
        - `sleep()` は秒単位で遅すぎるため、 `usleep()` を使う
        - `usleep()` の働き、引数などを調べてください。
    - さらに、実行時間を測定するため、 `clock_gettime()` を使う
        - 通常 `time()` を使用するが、これも1秒単位なので計測には不向き
        - `clock_gettime()` の働き、引数など調べてください。

処理時間測定版

【ex02_nothread.c】

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <unistd.h>

#define DATA_SIZE 100000

long data[DATA_SIZE];

void init_function(void) {
```

```
    for (int i = 0; i < DATA_SIZE; i++) {
        data[i] = i;
        usleep(100);
    }
    return;
}

int main(int argc, char const *argv[]) {
    struct timespec start_time, end_time;
    unsigned int sec;
    int nsec;
    double d_sec;

    clock_gettime(CLOCK_REALTIME, &start_time);

    init_function();

    clock_gettime(CLOCK_REALTIME, &end_time);

    // 処理中の経過時間を計算
    sec = end_time.tv_sec - start_time.tv_sec;
    nsec = end_time.tv_nsec - start_time.tv_nsec;

    d_sec = (double)sec + (double)nsec / (1000 * 1000 * 1000);

    // 計測時間の表示
    printf("time:%f\n", d_sec);
    return 0;
}
```

※実行結果　参考

```
$ time ./ex02_nothread
time:15.278656

real    0m15.282s
user    0m0.062s
sys     0m0.129s
```

実行環境

```
OS: Ubuntu 22.04.4 LTS
CPU : Ryzen 9 - 5900x
Memory : 64GB
```

スレッド版に書き換え

- 上記、`ex02_nothread.c`を2スレッドで動作するように変更してみる
    - 処理を分割し、同一の要素・変数を変更する
    - 配列100000個を、半分ずつ初期化するようにし、別々のスレッドで初期化する
    - `ex02_thread2.c`として作成し、実行時間を確認してください。

※ 実行結果　参考

```
$ time ./ex02_thread2
time:7.684227

real    0m7.688s
user    0m0.002s
sys     0m0.225s
```

Nスレッド版を作成する

- 先のプログラム`ex02_thread2.c`を、更に改良し、Nスレッドで動作するように書き換える。
    - `pthread_create(&th_a, NULL, ThreadA, NULL);`の最後の`NULL`部分に引数を渡すと処理しやすいかもしれない
    - ファイル名は`ex02_threadN.c`とする。

※ 実行結果　参考

N=4 での場合

```
$ time ./ex02_threadN
time:3.832136

real    0m3.835s
user    0m0.206s
sys     0m0.001s
```

N=8 での場合

```
$ time ./ex02_thread8
time:1.917582

real    0m1.920s
user    0m0.002s
sys     0m0.216s
```

N=16 での場合

```
$ time ./ex02_thread16
time:0.958275

real    0m0.961s
user    0m0.001s
sys     0m0.215s
```

N=24 での場合

```
$ time ./ex02_thread24
time:0.641370

real    0m0.645s
user    0m0.002s
sys     0m0.238s
```

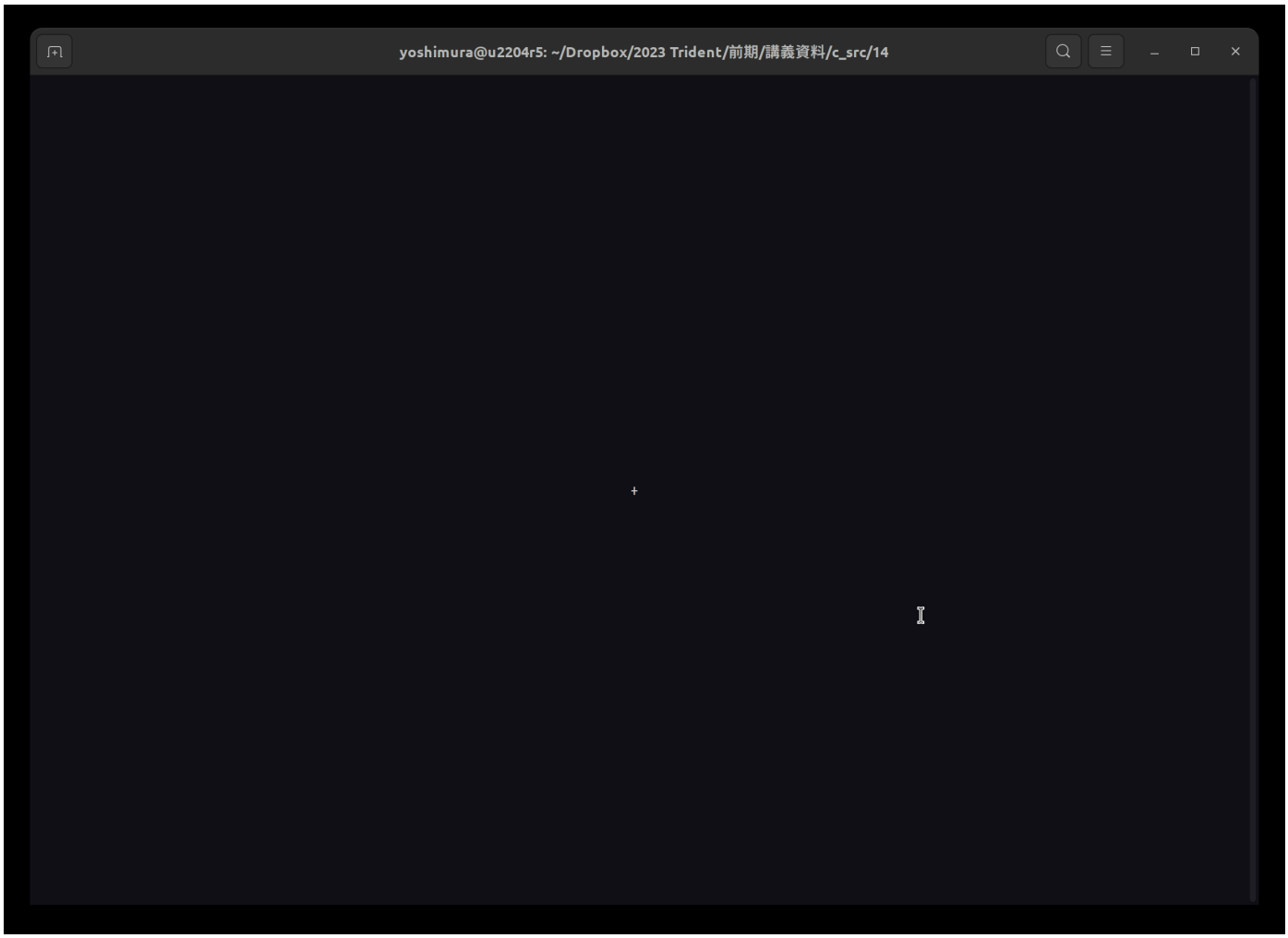
練習 （追記）

以下のプログラムをスレッドを利用して作成してください。

- 異なるタイマーを用いて、3種類のリズムを刻む（表示上）
    - 60BPM（Beats Per Minute）とする（1分間に60回の速さ）
        - 関数A：60BPM
        - 関数B：120BPM
        - 関数C：240BPM
    - 画面上に異なる記号で表示する（吉村の例では、A/B/C → =/+/- とした）
        - 画面上の表示位置は、ほぼ中央とする。（ncursesを使用すると楽に設定できるはず）
        - 表示し続けると拍がわからないので、およそ半分の時間表示し、半分の時間は消去するものとする
            - 関数Aの場合、60BPM→1拍/1秒間 となるので 0.5秒表示し0.5秒消去する
    - 全体で約30秒間（理論上）動作し、終了する
        - スレッド処理なので、正確に時間を測定しても、切り替えの時間が余分にかかるため、理論（計算）通りにはならない

【実行例1】

```
$ ./rhythm_thread
```



使用する関数

- `usleep()`
- ncursesライブラリの各種関数
- pthreadライブラリ