システムソフトウェア中間課題

21B30362　佐久川泰輔

2023 年 12 月 9 日

1　問 1_1

1.1　コードの説明

以下は、スレッドライブラリを実装している uthreads. の一部である。

Listing 1　uthreads.c の冒頭

```
#define STACK_DEPTH 512
#define MAX_THREADS 4

#define UNUSED 0
#define RUNNABLE 1
#define RUNNING 2

struct uthread {
    int tid;
    int state;
    uint64 stack[STACK_DEPTH];
    struct context my_context;
    void (*fun)();
};

struct uthread uthreads[MAX_THREADS];
struct uthread *current_thread;
struct context start_context;
```

構造体 uthread は、スレッドを構成するための構造体である。
スレッドの作成を行う関数は以下である。

Listing 2　make_uthread

```
int make_uthread(void (*fun)()) {
    int empty = -1;
    for (int i = 0; i < MAX_THREADS; i++) {
        if (uthreads[i].state == UNUSED) {
            empty = i;
            break;
```

```
7         }
8     }
9
10    if(empty == -1) {
11        return -1;
12    }
13
14    struct uthread *thread = malloc(sizeof(struct uthread));
15    thread->tid = empty;
16    thread->state = RUNNABLE;
17    thread->fun = fun;
18    thread->my_context.ra = (uint64) fun;
19    thread->my_context.sp = (uint64)(thread->stack + STACK_DEPTH);
20    uthreads[empty] = *thread;
21
22    return thread->tid;
23 }
```

まず、2-8 行目で、uthreads 配列のうち、空いている場所を特定する。空いている場所が無い場合、スレッドの作成を行わず-1 を返す。

次に、uthread 構造体を 1 つ作り、スレッドの初期化を行い、そのスレッド番号を返す。

以下は、スレッドの開始を行う関数である。

Listing 3　start_uthread

```
1 void start_uthreads() {
2     current_thread = &uthreads[0];
3     current_thread->state = RUNNING;
4
5     start_context.ra = (uint64) start_uthreads;
6     start_context.sp = (uint64)(current_thread->stack + STACK_DEPTH);
7     swtch(&start_context, &current_thread->my_context);
8 }
```

配列の 0 番目を current_thread とした後にそれを実行状態にし、現在の関数の情報を start_conext に記録する。

その後、swtch() により current_thread に処理を切り替える。 以下は、スレッドを切り替える関数 yield() である。

Listing 4　yield

```
1 void scheduler() {
2     struct uthread *t = 0;
3     struct uthread *prev = current_thread;
4     int i;
5
6     for (i = current_thread->tid + 1; i <= current_thread->tid + MAX_THREADS;
          i++) {
```

```
7         if (i >= MAX_THREADS) i -= MAX_THREADS;
8         if (uthreads[i].state == RUNNABLE) {
9             t = &uthreads[i];
10            break;
11        }
12
13        if (i == current_thread->tid) {
14            swtch(&prev->my_context, &start_context);
15            return;
16        }
17    }
18
19    t->state = RUNNING;
20    if (t != current_thread) {
21        current_thread = t;
22        swtch(&prev->my_context, &current_thread->my_context);
23    }
24 }
25
26 void yield() {
27    current_thread->state = RUNNABLE;
28    scheduler();
29 }
```

関数 yield() のうち、実際に処理を切り替える部分は別の関数 scheduler() に抽出してある。これは、後の実装でこの部分を使いまわす事が可能になるからである。

まず、current_thread を実行可能状態にする。

その後、uthreads から実行可能なスレッドを一つ選択する。選択できなかった場合、そこで処理を中断する。

選択できた場合、そのスレッドを実行状態にし、current_thread とした後で、swtch() により処理を切り替える。

以下は、現在実行中のスレッド ID を返す関数である。

Listing 5　mytid

```
1 int mytid() {
2     return current_thread->tid;
3 }
```

current_thread のスレッド番号を返す。

1.2　テストコードの実行

以下は、uthreads.c のテストコード uttest1_1.c である。ただし、include 文等は省略してある。

Listing 6　uttest1_1.c の一部

```
1 static int hoge = 0;
2
```

```
3  void foo() {
4      for (;;) {
5          printf("foo (tid=%d): %d\n", mytid(), hoge);
6          hoge++;
7          yield();
8      }
9  }
10
11 void bar() {
12     int c = 0;
13     for (;;) {
14         printf("bar (tid=%d): %d\n", mytid(), hoge);
15         yield();
16         c++;
17         hoge += c;
18     }
19 }
20
21 int main() {
22     make_uthread(foo);
23     make_uthread(bar);
24     start_uthreads();
25     printf("main (tid=%d): end\n", mytid());
26     exit(0);
27 }
```

これを実行した結果は以下である。

図 1　uttest1_1.c の実行結果

2　問 1_2

2.1　コードの説明

以下は、スレッドを終了させる関数 uthread_exit() である。

Listing 7　uthread_exit

```
1 void uthread_exit() {
2     current_thread->state = UNUSED;
3     free(current_thread);
4     scheduler();
5 }
```

まず、現在のスレッドの位置を未使用状態にする。

その後、スレッドに使用していたメモリを解放する。

最後に、yield() の時と同様に別のスレッドに実行を移す。

その際に、全てのスレッドの実行が終了した場合、start_context と current_thread が swtch() されるため、再度 start_uthreads() から戻る。

2.2 テストコードの実行

以下は、uthreads.c のテストコード uttest2.c である。ただし、include 文等は省略してある。

Listing 8　uttest2.c の一部

```
1  void foo() {
2      int c = 0;
3      for (;;) {
4          printf("foo (tid=%d): %d\n", mytid(), c);
5          c += 1;
6          yield();
7          if (c > 6)uthread_exit();
8      }
9  }
10
11 void bar() {
12     int c = 0;
13     for (;;) {
14         printf("bar (tid=%d): %d\n", mytid(), c);
15         yield();
16         c += 2;
17         if (c > 8)uthread_exit();
18     }
19 }
20
21 void baz_sub(int *cp) {
22     printf("baz (tid=%d): %d\n", mytid(), *cp);
23     yield();
24     *cp += 3;
25 }
26
27 void baz() {
28     int c = 0;
29     for (;;) {
30         baz_sub(&c);
```

```
31          baz_sub(&c);
32          if (c > 10) uthread_exit();
33      }
34  }
35
36  int main() {
37      make_uthread(foo);
38      make_uthread(foo);
39      make_uthread(bar);
40      make_uthread(baz);
41      start_uthreads();
42      make_uthread(bar);
43      make_uthread(baz);
44      start_uthreads();
45      exit(0);
46  }
```

これを実行した結果は以下である。

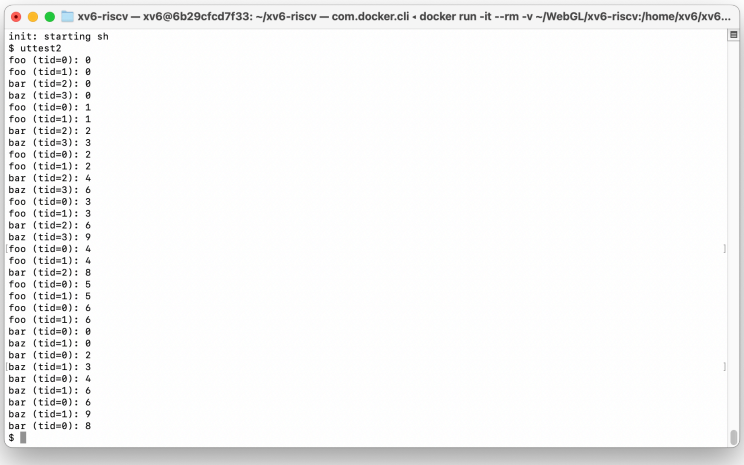

図 2　uttest2.c の実行結果