

ĐẠI HỌC QUỐC GIA TP HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA



THIẾT KẾ VÀ PHÁT TRIỂN ỨNG DỤNG IOT
BÀI TẬP VỀ NHÀ: MULTITHREAD VÀ PROTOTHREAD

LỚP L01 --- HK232

NGÀY NỘP: 09/02/2024

Giảng viên hướng dẫn: Thầy Võ Quế Sơn

Sinh viên thực hiện	Mã số sinh viên
Châu Ngọc Tình	2110588

Thành phố Hồ Chí Minh – 2024

1. Nội dung

1.1. Protothread

Protothread là một cách để cấu trúc code cho phép hệ thống chạy các hoạt động khác khi code đang chờ điều gì đó xảy ra.

Protothread là một hàm C thông thường, được bắt đầu và kết thúc bằng hai macro `PT_BEGIN()` và `PT_END()`. Giữa hai macro có thể sử dụng các protothread khác đã được khai báo.

Preprocessor C của các hoạt động chính của protothread:

```
1 struct pt { lc_t lc };
2 #define PT_WAITING 0
3 #define PT_EXITED 1
4 #define PT_ENDED 2
5 #define PT_INIT(pt) LC_INIT(pt->lc)
6 #define PT_BEGIN(pt) LC_RESUME(pt->lc)
7 #define PT_END(pt) LC_END(pt->lc); \
8 return PT_ENDED
9 #define PT_WAIT_UNTIL (pt, c) LC_SET(pt->lc); \
10 if(!(c)) \
11 return PT_WAITING
12 #define PT_EXIT(pt) return PT_EXITED
```


Khai báo local continuations với cấu trúc switch trong C:

```
1 typedef unsigned short lc_t;
2 #define LC_INIT(c) c = 0
3 #define LC_RESUME(c) switch(c) { case 0:
4 #define LC_SET(c) c = __LINE__; case __LINE__:
5 #define LC_END(c) }
```

struct pt { lc_t lc}: bao gồm một biến `lc` kiểu `unsigned short`, viết tắt của `local continuation`. Vậy mỗi protothread chiếm 2 byte dung lượng.

Protothread sử dụng macro thay vì dùng hàm là vì phù hợp với mục đích làm thay đổi luồng điều khiển. Nếu dùng hàm thì khi biên dịch cần thêm các đoạn hợp ngữ khi gọi hàm, vì vậy dùng macro để thay đổi luồng điều khiển bằng các cấu trúc C tiêu chuẩn sẽ tối ưu hơn.


Ví dụ: (giả sử `__LINE__` đang ở dòng thứ 12 của code):



```

1 static PT_THREAD(example(struct pt *pt))
2 {
3     PT_BEGIN(pt);
4
5     while(1) {
6         PT_WAIT_UNTIL(pt,
7             counter == 1000);
8         printf("Threshold reached\n");
9         counter = 0;
10    }
11
12    PT_END(pt);
13 }

```



```

1 static char example(struct pt *pt)
2 {
3     switch(pt->lc) { case 0:
4
5         while(1) {
6             pt->lc = 12; case 12:
7                 if(!(counter == 1000)) return 0;
8                 printf("Threshold reached\n");
9                 counter = 0;
10            }
11
12        } pt->lc = 0; return 2;
13 }


```

1.2. Protothread in Process

Process trong Contiki OS được viết dựa trên protothread của Dunkel (<http://dunkels.com/adam/pt/>). Vì vậy Contiki có phiên bản protothread riêng của nó, cho phép chờ đợi các sự kiện đến.

Process trong Contiki OS có những đặc điểm sau:

- Các process hoạt động theo dạng `cooperative multithreading`, các source code cho biết chính xác điểm nào sẽ được thực thi. Không phải ở hình thức `preemptive multithreading` nên các process sẽ không chuyển đổi tự động giữa các thread, mà là thực hiện xoay vòng tuần tự.
- Các process cùng sử dụng chung stack, vì vậy không có biến thread-local. Thay vào đó dùng các static variable để bảo toàn các giá trị qua nhiều lệnh gọi process.



```

1 PROCESS_BEGIN();           // Declares the beginning of a process' protothread.
2 PROCESS_END();             // Declares the end of a process' protothread.
3 PROCESS_EXIT();            // Exit the process.
4 PROCESS_WAIT_EVENT();      // Wait for any event.
5 PROCESS_WAIT_EVENT_UNTIL(); // Wait for an event, but with a condition.
6 PROCESS_YIELD();           // Wait for any event, equivalent to PROCESS_WAIT_EVENT().
7 PROCESS_WAIT_UNTIL();      // Wait for a given condition; may not yield the process.
8 PROCESS_PAUSE();           // Temporarily yield the process.

```

Ví dụ:



```
1  PROCESS_THREAD(hello_world_process, ev, data)
2  {
3      static int i;
4      PROCESS_BEGIN();
5      for (i=0;i<10;i++)
6      {
7          printf("%d from 1st process\n",i);
8          process_poll(&hello_world_process2);
9          PROCESS_YIELD();
10     }
11     PROCESS_END();
12 }
13
14 PROCESS_THREAD(hello_world_process2, ev, data)
15 {
16     static int i;
17     PROCESS_BEGIN();
18     for (i=0;i<10;i++)
19     {
20         printf("%d from IIund process\n",i);
21         process_poll(&hello_world_process);
22         PROCESS_YIELD();
23     }
24     PROCESS_END();
25 }
```

2. Bài làm

2.1. Multithread

2.1.1. Đoạn code sử dụng Multithread

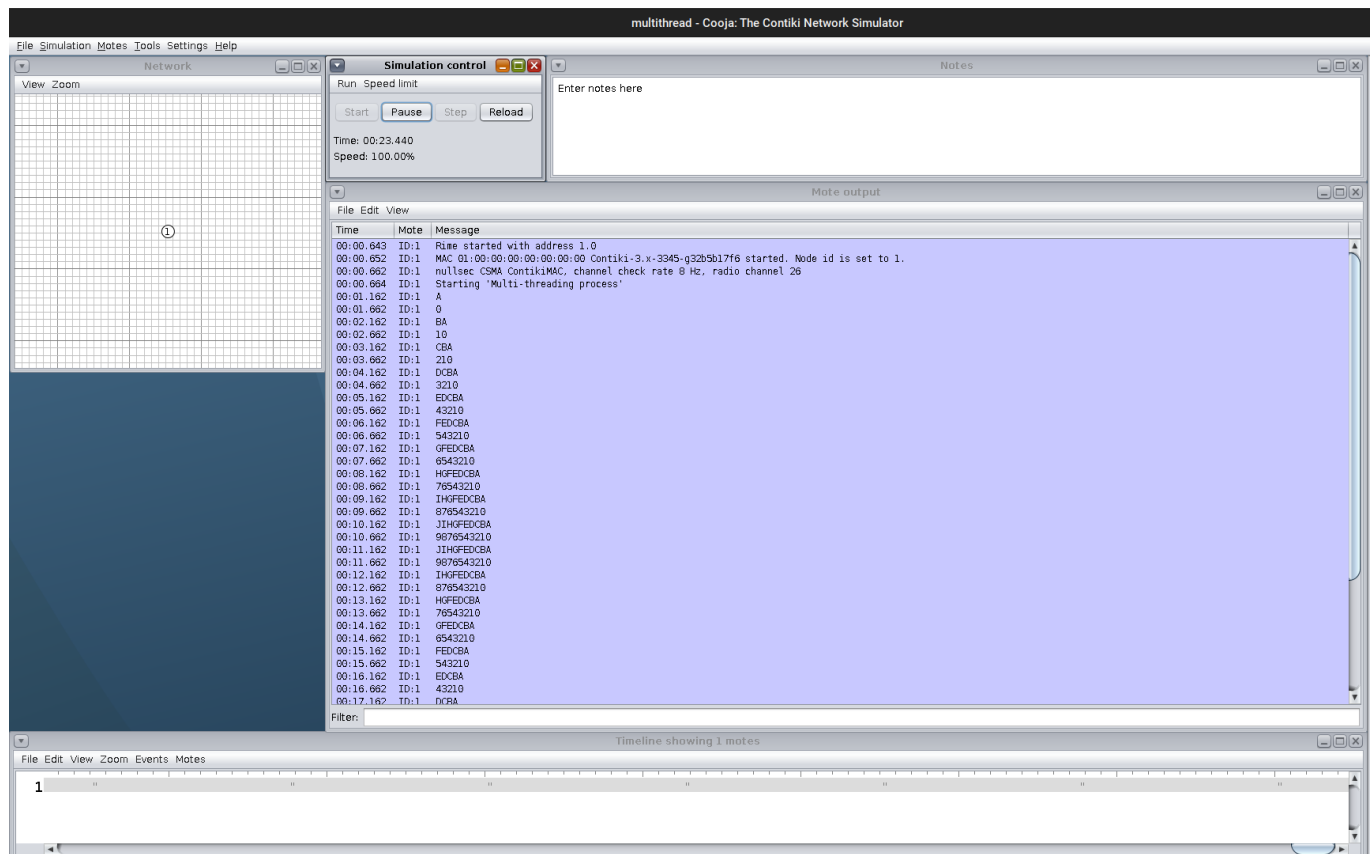
```
1  #include "contiki.h"
2  #include "sys/mt.h"
3  #include <stdio.h>
4
5  static char *ptr;
6
7  PROCESS(multi_threading_process, "Multi-threading process");
8  AUTOSTART_PROCESSES(&multi_threading_process);
9
10 /*-----*/
11 static void thread_func(char *str, int len)
12 {
13     ptr = str + len;
14     mt_yield();
15
16     if(len != 0)
17     {
18         thread_func(str, len - 1);
19         mt_yield();
20     }
21
22     ptr = str + len;
23 }
24
25 /*-----*/
26 static void thread_main(void *data)
27 {
28     while(1)
29     {
30         thread_func((char *)data, 9);
31     }
32     mt_exit();
33 }
```

```

1  /*-----*/
2  PROCESS_THREAD(multi_threading_process, ev, data)
3  {
4      static struct mt_thread alpha_thread;
5      static struct mt_thread count_thread;
6      static struct etimer timer;
7      static int toggle = 1;
8
9      PROCESS_BEGIN();
10
11     mt_init();
12     mt_start(&alpha_thread, thread_main, "JIHGFEDCBA");
13     mt_start(&count_thread, thread_main, "9876543210");
14
15     etimer_set(&timer, CLOCK_SECOND / 2);
16     while(1)
17     {
18         PROCESS_WAIT_EVENT();
19         if(ev == PROCESS_EVENT_TIMER)
20         {
21             if(toggle)
22             {
23                 mt_exec(&alpha_thread);
24                 toggle--;
25             } else
26             {
27                 mt_exec(&count_thread);
28                 toggle++;
29             }
30             puts(ptr);
31
32             etimer_set(&timer, CLOCK_SECOND / 2);
33         }
34     }
35     mt_stop(&alpha_thread);
36     mt_stop(&count_thread);
37     mt_remove();
38
39     PROCESS_END();
40 }

```

2.1.2. Mô phỏng bằng cooja




2.2. Protothread

2.2.1. Đoạn code sử dụng Protothread của Contiki OS



```
1  #include "contiki.h"
2  #include "sys/pt.h"
3  #include "clock.h"
4  #include <stdio.h>
5
6  #define ALPHA      0
7  #define COUNT      1
8  #define UP         0
9  #define DOWN       1
10 #define FIRST_MESS ALPHA
11 #define FIRST_MODE  DOWN
12
13 PROCESS(protothread, "Protothread process");
14 AUTOSTART_PROCESSES(&protothread);
15
16 char alpha[] = "JIHGFEDCBA";
17 char count[] = "9876543210";
18 uint8_t len = 9;
19
```

```

1  PROCESS_THREAD(protothread, event, data)
2  {
3      static struct etimer etim;
4      static uint8_t toogle = FIRST_MESS;
5      static uint8_t mode = FIRST_MODE;
6      PROCESS_BEGIN();
7      etimer_set(&etim, CLOCK_SECOND / 2);
8      while(1)
9      {
10         PROCESS_WAIT_UNTIL(etimer_expired(&etim));
11         etimer_set(&etim, CLOCK_SECOND / 2);
12         if (toogle == ALPHA)
13         {
14             char *message = alpha + len;
15             puts(message);
16             toogle++;
17         } else
18         {
19             char *message = count + len;
20             puts(message);
21             toogle--;
22         }
23         if (toogle == FIRST_MESS)
24         {
25             if (mode == DOWN)
26             {
27                 len--;
28
29                 if (len == 0) mode = UP;
30             } else
31             {
32                 len++;
33
34                 if (len == 9) mode = DOWN;
35             }
36         }
37         etimer_set(&etim, CLOCK_SECOND / 2);
38     }
39     PROCESS_END();
40 }
41

```

2.2.2. Mô phỏng trên cooja

