CS314-001 Operating Systems
Programming Project #2 Description, Summer 2023

Project Due: 11:59:59 p.m. on July 27th (Tuesday)

I. Objectives

The objective in this programming project is to have practical experience for thread programming in UNIX operating systems, using "pthread". Pthread is a UNIX user-mode thread library and it is one of the most popular thread implementations for UNIX platforms. In this project, we implement the bather and boiler-man controls in the same way we did for Project #1, with the following two exceptions: (1) we use threads, instead of processes for implementing three bathers and two boiler-men, and (2) the safeguard is implemented by an interrupt handler (the interrupt handler is the safeguard, no thread for the safeguard).

II. Requirements

(1) Each submission should meet both “the output requirements (see the appendix for the required outputs)” and “the program structure requirements”.

(2) Figure 1 shows the required program structure for Project #2 (structural requirement #1).

(3) The main (the parent) thread should create five child threads after it is started (functional requirement #1).

Figure 1 - the required program structure for Project #2
(4) Project #2 requires the following structure for each bather thread (Figure 2) and each boiler-man thread (Figure 3) primarily for testing purposes after your submission (this program structure is posted to the course home (“required_p2_base.c”)) (structural requirement #2 & #3).

```c
void bather_thread (void * arg)
{
    int myid = *(int *) arg;   // child thread number (not ID)
    int random_num;            // the random number
    int doit;                  // "head or tail" flag

    /* declare the start of this child thread ----------- */
    printf("Bather %d started ...
", myid);

    /* infinite loop for a bather thread --------------- */
    while (1)
    {
        /* flip the coin ---------------------------------- */
        random_num = rand();
        random_num = random_num % 100;
        if (random_num < SLEEP_PROBABILITY)
            { doit = 1; }
        else
            { doit = 0; }

        /* if the coin is "head", this thread sleeps ------ */
        if (doit == 1)
        {
            Time (&T);  // get the current time
            t.tv_sec = T + BATHER_SLEEP_OUT;
        }

        /* semaphore control for a bather to enter the critical section */
        /* the critical section for a bather starts here -- */
        printf("     AT-%d is entering the swimming pool ...
", myid);

        /* if the coin is "head", this thread sleeps ------ */
        if (doit == 1)
        {
            Time (&T);  // get the current time
            t.tv_sec = T + BATHER_SLEEP_IN;
        }

        printf("          AT-%d is leaving the swimming pool ...
", myid);

        /* semaphore control for a bather to leave the critical section */
        /* the critical section for a bather ends here -------- */
    }
}
```

Figure 2 – the required structure for each bather thread (required_p2_base.c)
void boilerman_thread (void * arg) 
{
    int myid = *(int *) arg;  // child thread number (not ID)

    /* declare the start of this child thread -------- */
    printf ("Boilerman %d started ...
", myid);

    /* infinite loop for a boilerman thread ---------- */
    while (1)
    {
        /* thread sleep (out) - always for each boilerman */
        time(&T);  // get the current time
        t.tv_sec = T + BOILERMAN_SLEEP_OUT;

        /* semaphore control for a boilerman to enter the critical section */
        /* the critical section for a boilerman starts here ------ */
        printf ("BM-%d is entering the swimming pool ..
", myid);

        /* thread sleep (in) - always for each boilerman */
        time(&T);  // get the current time
        t.tv_sec = T + BOILERMAN_SLEEP_IN;

        printf ("     BM-%d is leaving the swimming pool..
", myid);

        /* semaphore control for a boilerman to leave the critical section */
        /* the critical section for a boilerman ends here --------- */
    }
}

void clock_interrupt_handler(void)
{
    /* safeguard tries to enter the critical section ----- */

    /* the critical section starts here ----------------*/
    printf ("************************************************
");
    printf ("safeguard is entering the swimming pool (%d) ..
", safeguard_count);
    printf ("************************************************
");

    /* the critical section ends here ------------------ */

    /* increase the safeguard counter ------------------ */
    safeguard_count = safeguard_count + 1;

    /* scheduler wakes up again one second later -------- */
}
(6) To control the activity timing of the bathers and the boiler-men threads (the timing parameters will be changed when each submission is tested) the following five labels should be declared at the top of your source code and be used in the critical section as specified below (including the following labels in your C source code is required) (structural requirement #5).

- SAFEGUARD_INTERVAL 10 // the safeguard duty interval (in seconds)
- BATHER_SLEEP_OUT 2 // '2' for '1' second
- BATHER_SLEEP_IN 2 // '2' for '1' second
- BOILERMAN_SLEEP_OUT 2 // '2' for '1' second
- BOILERMAN_SLEEP_IN 3 // '3' for '2' seconds

Note: The above timing parameters as “labels” (should be in “micro-second” order) are posted to the CS314 course home (“required_test_parameters_P2.c”).

(7) In addition to the five timing parameters above, the following parameter should appear at the beginning of each submitted *.c source code file (structural requirement #6):

SLEEP_PROBABILITY 40 // sleep probability (40%) for each bather

(8) Each submitted source code file will be tested after the timing parameters are changed. Multiple different child thread timing parameter sets will be tested.

(9) Other requirements

(a) No spin-wait by the parent thread, all five child threads, and the interrupt handler (functional requirement #2).

(b) No boiler-man starvation (functional requirement #3).

(c) Multiple (concurrent) accesses to the critical section by 2, or 3 bather threads should happen (functional requirement #4).

(d) Exclusive access to the critical section by the two boiler-man threads and by the safeguard (functional requirement #5).

(e) The interrupt interval should be declared by a label, “#define SAFEGUARD_INTERVAL”, which specifies the interrupt interval in seconds (e.g., “#define SAFEGUARD_INTERVAL 10” means that the interrupt handler will be activated once in each ten second) (structural requirement #7).

(f) The outputs from all threads (including those from the interrupt handler) should follow the sample outputs posted to the course home. No output should be made except by
the required outputs (i.e., the required `printf` in the required structure). Any deviation from this requirement may be considered as cheating (structural requirement #8).

(g) When the interrupt handler (i.e., the safeguard) wakes up, it should display (functional requirement #6):

```
safeguard is entering the swimming pool (0) ..
```

“safeguard is entering the swimming pool (d)”, where ‘d’ represents how many times the interrupt handler was activated so far (since the beginning of the program run). Please see the sample outputs posted to the course home.

(h) You can add (use) your own global variable(s). You can use as many pthread condition variables as you like.

![Sample program outputs](image)

Figure 5 – Sample program outputs
III. Testing

Each submission will be tested by changing the timing parameters:

IV. Required Submission

Program source code file (named as “p2_nnn.c” where ‘nnn’ represents the last three digits of your SIUE 800- ID number (some of you are expected to specify four digits, instead of three)) should be emailed to hfujino@siue.edu by the deadlines.

(a) Extra-credit early submission deadline: 11:59:59 p.m., on July 22nd

(b) Free-feedback early submission deadline: 11:59:59 p.m., on July 25th

(c) Final submission deadline: 11:59:59 p.m., on July 27th (Tuesday)

(d) Late submissions within 48 hours past the final submission due: -10% for each 12 hours after the final submission deadline.

(e) Late submissions over 48 hours past the final submission due: will not be accepted

V. Other Requirements

• To be posted, if any.

VI. Questions about this project

• To encourage early starts, no new question will be answered within 48 hours before the final submission due.
APPENDIX (the required outputs):

While the program (the five child threads and the interrupt handler) is in progress (functional requirement #7):

```
******************************************************************************
safeguard is entering the swimming pool (12) ..
******************************************************************************
AT-2 is entering the swimming pool ..
AT-2 is leaving the swimming pool ..
AT-1 is entering the swimming pool ..
AT-1 is leaving the swimming pool ..
AT-3 is entering the swimming pool ..
AT-3 is leaving the swimming pool ..
EM-2 is entering the swimming pool ..
EM-2 is leaving the swimming pool ..
AT-2 is entering the swimming pool ..
AT-2 is leaving the swimming pool ..
EM-1 is entering the swimming pool ..
EM-1 is leaving the swimming pool ..
AT-1 is entering the swimming pool ..
AT-3 is entering the swimming pool ..
AT-2 is entering the swimming pool ..
AT-2 is leaving the swimming pool ..
AT-1 is leaving the swimming pool ..
AT-3 is leaving the swimming pool ..
EM-2 is entering the swimming pool ..
EM-2 is leaving the swimming pool ..
EM-1 is entering the swimming pool ..
EM-1 is leaving the swimming pool ..
AT-2 is entering the swimming pool ..
AT-1 is entering the swimming pool ..
AT-1 is leaving the swimming pool ..
AT-1 is leaving the swimming pool ..
AT-3 is entering the swimming pool ..
AT-3 is leaving the swimming pool ..
AT-2 is leaving the swimming pool ..
```