



Міністерство освіти і науки України
НТУУ «Київський політехнічний інститут»
Фізико-технічний інститут

ОПЕРАЦІЙНІ СИСТЕМИ
Комп'ютерний практикум №8
Варіант №11

Виконав:

Студент 2 курсу ФТІ
Групи ФІ-92
Поночевний Назар Юрійович

Перевірив:

Ільїн Костянтин Іванович

Київ – 2021

Робота №8. Засоби синхронізації потоків

Варіант №11

Мета: Оволодіння практичними навичками розроблення багатопотокових програм з підтримкою засобів синхронізації.

Завдання для самостійної підготовки

1. Ознайомитись з документацією і прикладами використання засобів синхронізації такими як семафори, м'ютекси, умовні змінні:

- man pages;
- книги з числа рекомендованих, зокрема [1, розд. 2.3], [5, розд. 5];
- [11, с. 103-126];
- [12, розд. 7, 8];
- корисна стаття [13] (у цій роботі нас цікавлять лише семафори і м'ютекси, але ми до цієї статті ще повернемося);
- великі книги з програмування в Linux, що орієнтовані на кодерів, містять приклади коду, перекладені російською мовою, тому комусь можуть бути цікавими, зрозумілими, і взагалі дуже корисними [14, 15, 16] (для цієї роботи див. розділи про семафори, м'ютекси, умовні змінні, тощо).

2. Якщо не робили попередні роботи, то перевірити, чи встановлений у вашій системі Linux компілятор C/C++ (g++). Якщо ні, встановіть за допомогою менеджера пакетів.

Завдання до виконання

1. Розминка. Стандартна задача виробник-споживач. Задача була розглянута на лекції. Також детально розглянута в рекомендованих книжках [1, 5]. Розробіть програму, що демонструє рішення цієї задачі за допомогою семафорів. Для цього напишіть:

- функції виробника і споживача (наприклад, як на лекції, або як у Шеховцові, але так, щоби працювало);

- функції створення і споживання об'єктів (рекомендується “UNIX-подібні”, є багатозадачнимстворювати” рядки тексту шляхом зчитування їх з файлу, хоча можливі й інші варіанти за вибором викладача або за вашою фантазією, наприклад розрахунки геш-функцій sha2 з рядків рандомних символів, а “UNIX-подібні”, є багатозадачнимиспоживати” їх шляхом роздрукування на екрані з додатковою інформацією такою як ідентифікатор потоку і мітка часу, причому і там, і там для моделювання складного характеру реального життя виробників і споживачів можна додавати рандомні затримки);
- функцію main(), що створює потоки-виробники і потоки-споживачі, при цьому треба передбачити введення з клавіатури або як параметри командного рядка кількості записів у буфері, кількості виробників і кількості споживачів для досліджень їх роботи;
- обов'язково передбачити коректне завершення усього цього господарства. Продемонструвати викладачеві як воно працює (не менше двох виробників і двох споживачів) і код, що ви написали.

```

1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <time.h>
4  #include <pthread.h>
5  #include <semaphore.h>
6  #include <unistd.h>
7
8  typedef unsigned long int buffer_item;
9  #define BUFFER_SIZE 5
10 #define RAND_DIVISOR 100000000
11
12 sem_t lock, full, empty;
13 buffer_item buffer[BUFFER_SIZE];
14 int counter;
15
16 void *producer(void *param);
17 void *consumer(void *param);
18
19 void initializeData() {
20     sem_init(&lock, 0, 1);
21     sem_init(&full, 0, 0);
22     sem_init(&empty, 0, BUFFER_SIZE);
23     counter = 0;
24     srand(time(NULL));
25 }
26
27 int insert_item(buffer_item item) {
28     if (counter < BUFFER_SIZE) {
29         buffer[counter] = item;
30         counter++;
31         return 0;
32     }
33     return -1;
34 }
35
36 int remove_item(buffer_item *item) {
37     if (counter > 0) {
38         *item = buffer[(counter-1)];
39         counter--;
40         return 0;
41     }
42     return -1;
43 }
44
45 void *producer(void *param) {
46     buffer_item item;

```

```

45 void *producer(void *param) {
46     buffer_item item;
47     pthread_t tid = pthread_self();
48
49     while (1) {
50         int rNum = rand() / RAND_DIVISOR;
51         usleep(rNum);
52
53         item = rand() % 100000 + 1000;
54
55         sem_wait(&empty);
56         sem_wait(&lock);
57
58         if (insert_item(item))
59             printf("(%d) Producer report error condition\n", tid);
60         else
61             printf("(%d) producer produced %d\n", tid, item);
62
63         sem_post(&lock);
64         sem_post(&full);
65     }
66 }
67
68 void *consumer(void *param) {
69     buffer_item item;
70     pthread_t tid = pthread_self();
71
72     while (1) {
73         int rNum = rand() / RAND_DIVISOR;
74         usleep(rNum);
75
76         sem_wait(&full);
77         sem_wait(&lock);
78
79         if (remove_item(&item))
80             printf("(%d) Consumer report error condition\n", tid);
81         else
82             printf("(%d) consumer consumed %d\n", tid, item);
83
84         sem_post(&lock);
85         sem_post(&empty);
86     }
87 }
88
89 int main(int argc, char *argv[]) {

```

```

88
89 int main(int argc, char *argv[]) {
90     unsigned int i;
91
92     if (argc != 4) {
93         printf("USAGE: ./Lab8_1 <mainSleepTime> <numProd> <numCons>\n");
94         return 1;
95     }
96
97     int mainSleepTime = atoi(argv[1]);
98     int numProd = atoi(argv[2]);
99     int numCons = atoi(argv[3]);
100
101     pthread_t producerThreads[numProd];
102     pthread_t consumerThreads[numCons];
103     initializeData();
104
105     for (i = 0; i < numProd; i++)
106         pthread_create(&producerThreads[i], NULL, producer, NULL);
107     for (i = 0; i < numCons; i++)
108         pthread_create(&consumerThreads[i], NULL, consumer, NULL);
109
110     usleep(mainSleepTime);
111
112     for (i = 0; i < numProd; i++)
113         pthread_cancel(producerThreads[i]);
114     for (i = 0; i < numCons; i++)
115         pthread_cancel(consumerThreads[i]);
116
117     printf("Exit the program\n");
118     return 0;
119 }
120

```

```

D:\MyPython3\Programming-Labs\Computer systems Software\Sem4\Lab8>Lab8_1.exe 2000 4 2
(1) producer produced 19467
(2) producer produced 19467
(3) producer produced 19467
(1) producer produced 27500
(4) producer produced 19467
(5) consumer consumed 19467
(6) consumer consumed 27500
(5) consumer consumed 19467
(2) producer produced 27500
(6) consumer consumed 27500
(3) producer produced 27500
(5) consumer consumed 27500
(1) producer produced 16724
(6) consumer consumed 16724
(4) producer produced 27500
(5) consumer consumed 27500
(2) producer produced 16724
(6) consumer consumed 16724
(3) producer produced 16724
(5) consumer consumed 16724
(1) producer produced 30358
(6) consumer consumed 30358
(4) producer produced 16724
(5) consumer consumed 16724
(2) producer produced 30358
(6) consumer consumed 30358
(3) producer produced 30358
(5) consumer consumed 30358
(1) producer produced 25464
(6) consumer consumed 25464
(4) producer produced 30358
(5) consumer consumed 30358
(2) producer produced 25464
(6) consumer consumed 25464
(3) producer produced 25464
(5) consumer consumed 25464
(1) producer produced 29145
(6) consumer consumed 29145
(4) producer produced 25464
Exit the program

```

2. Продовження розминки. Теж саме, але не на семафорах, а на м'ютексі і умовних змінних. Модифікуйте програму п. 1 так, щоби використовувати м'ютекс і умовну змінну.

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <time.h>
4  #include <pthread.h>
5  #include <semaphore.h>
6  #include <unistd.h>
7
8  typedef unsigned long int buffer_item;
9  #define BUFFER_SIZE 5
10 #define RAND_DIVISOR 100000000
11
12 pthread_mutex_t lock;
13 pthread_cond_t full, empty;
14 buffer_item buffer[BUFFER_SIZE];
15 int counter;
16
17 void *producer(void *param);
18 void *consumer(void *param);
19
20 void initializeData() {
21     pthread_mutex_init(&lock, NULL);
22     pthread_cond_init(&full, NULL);
23     pthread_cond_init(&empty, NULL);
24     counter = 0;
25     srand(time(NULL));
26 }
27
28 int insert_item(buffer_item item) {
29     if (counter < BUFFER_SIZE) {
30         buffer[counter] = item;
31         counter++;
32         return 0;
33     }
34     return -1;
35 }
36
37 int remove_item(buffer_item *item) {
38     if (counter > 0) {
39         *item = buffer[(counter-1)];
40         counter--;
41         return 0;
42     }
43     return -1;
44 }
45
46 void *producer(void *param) {
```

```

46 void *producer(void *param) {
47     buffer_item item;
48     pthread_t tid = pthread_self();
49
50     while (1) {
51         int rNum = rand() / RAND_DIVISOR;
52         usleep(rNum);
53
54         item = rand() % 100000 + 1000;
55
56         pthread_mutex_lock(&lock);
57         while (counter == BUFFER_SIZE)
58             pthread_cond_wait(&empty, &lock);
59
60         if (insert_item(item))
61             printf("(%d) Producer report error condition\n", tid);
62         else
63             printf("(%d) producer produced %d\n", tid, item);
64
65         pthread_mutex_unlock(&lock);
66         pthread_cond_signal(&full);
67     }
68 }
69
70 void *consumer(void *param) {
71     buffer_item item;
72     pthread_t tid = pthread_self();
73
74     while (1) {
75         int rNum = rand() / RAND_DIVISOR;
76         usleep(rNum);
77
78         pthread_mutex_lock(&lock);
79         while (counter == 0)
80             pthread_cond_wait(&full, &lock);
81
82         if (remove_item(&item))
83             printf("(%d) Consumer report error condition\n", tid);
84         else
85             printf("(%d) consumer consumed %d\n", tid, item);
86
87         pthread_mutex_unlock(&lock);
88         pthread_cond_signal(&empty);
89     }
90 }
91

```



```

92 int main(int argc, char *argv[]) {
93     unsigned int i;
94
95     if (argc != 4) {
96         printf("USAGE: ./Lab8_2 <mainSleepTime> <numProd> <numCons>\n");
97         return 1;
98     }
99
100    int mainSleepTime = atoi(argv[1]);
101    int numProd = atoi(argv[2]);
102    int numCons = atoi(argv[3]);
103
104    pthread_t producerThreads[numProd];
105    pthread_t consumerThreads[numCons];
106    initializeData();
107
108    for (i = 0; i < numProd; i++) {
109        pthread_create(&producerThreads[i], NULL, producer, NULL);
110        pthread_detach(producerThreads[i]);
111    }
112
113    for (i = 0; i < numCons; i++) {
114        pthread_create(&consumerThreads[i], NULL, consumer, NULL);
115        pthread_detach(consumerThreads[i]);
116    }
117
118    usleep(mainSleepTime);
119
120    for (i = 0; i < numProd; i++)
121        pthread_cancel(producerThreads[i]);
122    for (i = 0; i < numCons; i++)
123        pthread_cancel(consumerThreads[i]);
124
125    printf("Exit the program\n");
126    return 0;
127 }
128

```

```
D:\MyPython3\Programming-Labs\Computer systems Software\Sem4\Lab8>Lab8_2.exe 2000 4 2
(2) producer produced 19467
(1) producer produced 19467
(2) producer produced 27500
(3) producer produced 19467
(1) producer produced 27500
(5) consumer consumed 27500
(6) consumer consumed 19467
(3) producer produced 27500
(1) producer produced 16724
(5) consumer consumed 16724
(2) producer produced 16724
(6) consumer consumed 16724
(4) producer produced 19467
(5) consumer consumed 19467
(2) producer produced 30358
(6) consumer consumed 30358
(4) producer produced 27500
(5) consumer consumed 27500
(3) producer produced 16724
(6) consumer consumed 16724
(1) producer produced 30358
(5) consumer consumed 30358
(3) producer produced 30358
(6) consumer consumed 30358
(2) producer produced 25464
(5) consumer consumed 25464
(4) producer produced 16724
(6) consumer consumed 16724
(2) producer produced 29145
(5) consumer consumed 29145
(1) producer produced 25464
(6) consumer consumed 25464
(3) producer produced 25464
(5) consumer consumed 25464
(1) producer produced 29145
(6) consumer consumed 29145
(4) producer produced 30358
(5) consumer consumed 30358
(2) producer produced 17827
(6) consumer consumed 17827
(4) producer produced 25464
(5) consumer consumed 25464
(3) producer produced 29145
(6) consumer consumed 29145
(1) producer produced 17827
(5) consumer consumed 17827
(3) producer produced 17827
(6) consumer consumed 17827
(2) producer produced 1491
(5) consumer consumed 1491
Exit the program
```

3. Продовження розминки для тих, хто шукає пригод. Взаємне блокування
Модифікуйте програму п. 1 так, щоби викликати взаємне блокування. Для цього поміняйте місцями семафори. Переконайтесь у факті взаємного блокування і отримайте задоволення.

```

45 void *producer(void *param) {
46     buffer_item item;
47     pthread_t tid = pthread_self();
48
49     while (1) {
50         int rNum = rand() / RAND_DIVISOR;
51         usleep(rNum);
52
53         item = rand() % 100000 + 1000;
54
55         sem_wait(&lock);
56         sem_wait(&empty);
57
58         if (insert_item(item))
59             printf("(%d) Producer report error condition\n", tid);
60         else
61             printf("(%d) producer produced %d\n", tid, item);
62
63         sem_post(&lock);
64         sem_post(&full);
65     }
66 }
67
68 void *consumer(void *param) {
69     buffer_item item;
70     pthread_t tid = pthread_self();
71
72     while (1) {
73         int rNum = rand() / RAND_DIVISOR;
74         usleep(rNum);
75
76         sem_wait(&lock);
77         sem_wait(&full);
78
79         if (remove_item(&item))
80             printf("(%d) Consumer report error condition\n", tid);
81         else
82             printf("(%d) consumer consumed %d\n", tid, item);
83
84         sem_post(&lock);
85         sem_post(&empty);
86     }
87 }
88

```

```

D:\MyPython3\Programming-Labs\Computer systems Software\Sem4\Lab8>Lab8_3.exe 2000 4 2
(1) producer produced 19467
(2) producer produced 19467
(3) producer produced 19467
(1) producer produced 27500
(2) producer produced 27500
Exit the program

```

4. Індивідуальне завдання А тепер напишіть програму згідно індивідуального завдання (варіант вказує викладач).

Варіант 11 (2 а). Філософи, що обідають. Розробіть симулятор класичної задачі про філософів, що обідають. П'ять філософів сидять за круглим столом і їдять спагетті. Спагетті їдять за допомогою двох виделок. Всього виделок п'ять. Кожні двоє філософів, що сидять поруч, користуються однією спільною виделкою. Кожний філософ незалежно від інших може знаходитись в одному з двох станів — їсть або думає. Філософ думає деякий час (передбачте можливість рандомізувати цей час у певному

інтервалі, а також можливість задавати цей інтервал для дослідження), потім він намагається взяти виделки. У цьому варіанті завдання усі філософи спочатку намагаються взяти ліву виделку, а потім праву. Якщо йому вдалося захопити обидві виделки, він починає їсти. Ість він також деякий час (як і думає — але співвідношення часів варто змінювати для дослідження), після чого він звільняє обидві виделки і знову починає думати. І так далі, поки у нього не закінчатся спагетті. Якщо одну з виделок взяти неможливо, філософ чекає, поки вона звільниться. Якщо йому протягом певного часу (помітно більшого, ніж час їжи і час роздумів) так і не вдається ухопити дві виделки, він падає в обморок (потік завершується). Природно моделювати філософів за допомогою потоків, а виделки — за допомогою м'ютексів. Програма повинна синхронно (тобто, у тому ж порядку, як воно і відбувалося) друкувати усі події з мітками часу. Наприклад:

10:31:11.253 Філософ 1 узяв виделку 5 (ліву). Стан виделок ХОООХ

10:31:11.255 Філософ 2 узяв виделку 2 (праву). Стан виделок ХХООХ

10:31:11.541 Філософ 1 не зміг узяти виделку 1 (праву). Стан виделок ХХООХ

10:31:11.883 Філософ 4 узяв виделку 3 (ліву). Стан виделок ХХХОХ

10:31:11.253 Філософ 2 почав їсти.

10:31:12.117 Філософ 3 не зміг узяти виделку 2 (ліву). Стан виделок ХХХОХ

10:31:12.733 Філософ 4 узяв виделку 4 (праву). Стан виделок ХХХХХ ...

10:31:14.125 Філософ 2 закінчив їсти.

...

10:31:11.255 Філософ 2 поклав виделку 1 (ліву). Стан виделок ОХХХХ

...

```

1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <time.h>
4  #include <sys/time.h>
5  #include <pthread.h>
6  #include <unistd.h>
7  #include <string>
8
9  #define RAND_DIVISOR 100000000
10
11  using namespace std;
12
13  int rNum;
14  string forkStatus = "00000";
15  struct timeval nowTime;
16  double msNowTime;
17  pthread_t philosopher[5];
18  pthread_mutex_t forks[5];
19
20  void* func(void *arg) {
21      int n = *((int *) arg);
22      while (1) {
23          gettimeofday(&nowTime, NULL);
24          msNowTime = (nowTime.tv_sec) * 1000 + (nowTime.tv_usec) / 1000;
25          printf("[%f] Philosopher %d is thinking\n", msNowTime, n);
26
27          rNum = rand() / RAND_DIVISOR;
28          usleep(5000);
29
30          pthread_mutex_lock(&forks[n]);
31          forkStatus[n] = 'X';
32          gettimeofday(&nowTime, NULL);
33          msNowTime = (nowTime.tv_sec) * 1000 + (nowTime.tv_usec) / 1000;
34          printf("[%f] Philosopher %d takes fork %d (left). Fork status %s\n", msNowTime, n, n, forkStatus.c_str());
35
36          pthread_mutex_lock(&forks[(n + 1) % 5]);
37          forkStatus[(n + 1) % 5] = 'X';
38          gettimeofday(&nowTime, NULL);
39          msNowTime = (nowTime.tv_sec) * 1000 + (nowTime.tv_usec) / 1000;
40          printf("[%f] Philosopher %d takes fork %d (right). Fork status %s\n", msNowTime, n, (n + 1) % 5, forkStatus.c_str());
41
42          gettimeofday(&nowTime, NULL);
43          msNowTime = (nowTime.tv_sec) * 1000 + (nowTime.tv_usec) / 1000;
44          printf("[%f] Philosopher %d is eating\n", msNowTime, n);
45
46          rNum = rand() / RAND_DIVISOR;

```

```

45     printf("[%f] Philosopher %d is eating\n", msNowTime, n);
46     rNum = rand() / RAND_DIVISOR;
47     usleep(10000);
48
49     pthread_mutex_unlock(&forks[n]);
50     forkStatus[n] = '0';
51     gettimeofday(&nowTime, NULL);
52     msNowTime = (nowTime.tv_sec) * 1000 + (nowTime.tv_usec) / 1000;
53     printf("[%f] Philosopher %d leave fork %d (left). Fork status %s\n", msNowTime, n, n, forkStatus.c_str());
54
55     pthread_mutex_unlock(&forks[(n + 1) % 5]);
56     forkStatus[(n + 1) % 5] = '0';
57     gettimeofday(&nowTime, NULL);
58     msNowTime = (nowTime.tv_sec) * 1000 + (nowTime.tv_usec) / 1000;
59     printf("[%f] Philosopher %d leave fork %d (right). Fork status %s\n", msNowTime, n, (n + 1) % 5, forkStatus.c_str());
60
61     printf("Philosopher %d finished eating\n", n);
62 }
63 }
64
65 int main(int argc, char *argv[]) {
66     if (argc != 2) {
67         printf("USAGE: ./Lab8_4 start\n");
68         return 1;
69     }
70
71     unsigned int i;
72     srand(time(NULL));
73     int ids[] = {0, 1, 2, 3, 4};
74
75     for (i = 0; i < 5; i++)
76         pthread_mutex_init(&forks[i], NULL);
77     for (i = 0; i < 5; i++) {
78         pthread_create(&philosopher[i], NULL, func, &ids[i]);
79     }
80
81     for (i = 0; i < 5; i++)
82         pthread_join(philosopher[i], NULL);
83     for (i = 0; i < 5; i++)
84         pthread_mutex_destroy(&forks[i]);
85
86     printf("Exit the program\n");
87     return 0;
88 }
89

```

```

D:\MyPython3\Programming-Labs\Computer systems Software\Sem4\Lab8>Lab8_4.exe start
[-702208722.000000] Philosopher 0 is thinking
[-702208722.000000] Philosopher 1 is thinking
[-702208722.000000] Philosopher 2 is thinking
[-702208721.000000] Philosopher 3 is thinking
[-702208721.000000] Philosopher 4 is thinking
[-702208716.000000] Philosopher 2 takes fork 2 (left). Fork status 1xxx00
[-702208716.000000] Philosopher 3 takes fork 3 (left). Fork status XXXX00
[-702208713.000000] Philosopher 3 takes fork 4 (right). Fork status XXXXX0
[-702208713.000000] Philosopher 3 is eating
[-702208716.000000] Philosopher 1 takes fork 1 (left). Fork status XXXXX0
[-702208716.000000] Philosopher 0 takes fork 0 (left). Fork status XXXXX0
[-702208701.000000] Philosopher 2 takes fork 3 (right). Fork status XXXOX0
[-702208701.000000] Philosopher 2 is eating
[-702208701.000000] Philosopher 3 leave fork 3 (left). Fork status XXXOX0
[-702208700.000000] Philosopher 3 leave fork 4 (right). Fork status XXXO00
Philosopher 3 finished eating
[-702208700.000000] Philosopher 3 is thinking
[-702208700.000000] Philosopher 4 takes fork 4 (left). Fork status XXXOX0
[-702208685.000000] Philosopher 2 leave fork 2 (left). Fork status XXXOX0
[-702208685.000000] Philosopher 2 leave fork 3 (right). Fork status XXXOX0
Philosopher 2 finished eating
[-702208685.000000] Philosopher 2 is thinking
[-702208685.000000] Philosopher 3 takes fork 3 (left). Fork status XXXXX0
[-702208685.000000] Philosopher 1 takes fork 2 (right). Fork status XXXXX0
[-702208684.000000] Philosopher 1 is eating
[-702208670.000000] Philosopher 1 leave fork 1 (left). Fork status X0XXX0
[-702208669.000000] Philosopher 1 leave fork 2 (right). Fork status XX0XX0
Philosopher 1 finished eating

```

Висновок

ОС Linux має багато можливостей для управління потоками, але без методів синхронізації все це немає сенсу. На щастя, в ОС Linux також є всі відомі методи синхронізації, такі як умовна змінна, мютекси та семафори. На Windows я ще знаю є Event, Critical Section та Timer, але вони як правило додаткові. Загалом мені сподобався механізм роботи з синхронізацією. Весь процес доволі інтуїтивний та ефективно синхронізує потоки, щоб вони не заважали один одному при використанні загальних ресурсів.