Міністерство освіти і науки України

Харківський національний університет імені В. Н. Каразіна

Факультет комп'ютерних наук

Лабораторна робота №10

з навчальної дисципліни

«Операційні системи»

Виконав:
студент групи КУ-31
Нечипоренко Д.І.

Харків – 2022

Завдання

Задание №1 (Семафоры)

Существует поток-производитель и поток-потребитель данных. Производитель генерирует целое псевдослучайное число из заданного диапазона. Потребитель забирает это число. С помощью семафоров синхронизируйте их работу так, чтобы потребитель не мог пытаться получить еще не созданные числа, а производитель не мог сделать больше чисел, чем может получить потребитель. Через заданный промежуток времени потоки отменяются главным потоком и программа завершает свою работу.

Для синхронизации потоков разместим между потоком-производителем и потоком-потребителем буфер заданного при старте программы размера. Производитель может помещать сгенерированные числа в буфер, потребитель может забирать числа из буфера. Если потребитель забирает число, то его исключают из буфера. Необходимо обеспечить несколько требований:

1. когда потребитель или производитель работает с буфером остальные потоки должны ждать, пока он завершит свою работу;
2. когда производитель пытается поместить объект в буфер, а буфер полный, он должен дождаться, пока в буфере появится место;
3. когда потребитель пытается забрать объект из буфера, а буфер пустой, он должен дождаться, пока в нем появится объект.

Дополнительно: решите задачу для случая, когда работают несколько потоков-потребителей и несколько потоков-производителей (особое внимание уделите завершению приложения).

Задание №2 (Условные переменные)

Существует поток-производитель и поток-потребитель данных. Поток-производитель с достаточно большим периодом генерирует псевдослучайное число, присваивает его глобальной переменной и оповещает поток-потребитель. Поток-потребитель получает оповещение, забирает данные (исключает их из переменной) и сразу выводит их в стандартный поток вывода. Пока данных нет, поток-потребитель ожидает уведомления. Синхронизируйте работу потоков с помощью условных переменных. Через заданный промежуток времени потоки отменяются главным потоком и программа завершает свою работу.

Дополнительно: решите задачу для случая, когда дан буфер заданного (необязательно единичного) размера и работают несколько потоков-потребителей и несколько потоков-производителей (особое внимание уделите завершению приложения).

Код:

Task 1

Main

```
#include <stdlib.h>


#include "buffer.h"

#include "task.h"


int main() {
```

```
    TPARAM var;
    BUFFER buf;

    init(&buf, 7);

    var.buf = &buf;
    var.num_readers = 2;
    var.num_writers = 2;
    var.work_time = 5;
    task_solution(&var);

    destroy(&buf);

    return EXIT_SUCCESS;
}
```

Task

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

#include "task.h"
#include "threads.h"

void task_solution(TPARAM *init) {
    pthread_t *producers;
```

```
pthread_t *consumers;
pthread_attr_t attr;
int i;

pthread_attr_init(&attr);
pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);

producers = (pthread_t *) calloc(init->num_writers, sizeof(pthread_t));
consumers = (pthread_t *) calloc(init->num_readers, sizeof(pthread_t));

for (i = 0; i < init->num_writers; i++) {
    if (pthread_create(&producers[i], &attr, producer, init->buf)) {
        fprintf(stderr, "Writer Thread %d Creation Error\n", i);
        exit(EXIT_FAILURE);
    }
}

for (i = 0; i < init->num_readers; i++) {
    if (pthread_create(&consumers[i], &attr, consumer, init->buf)) {
        fprintf(stderr, "Reader Thread %d Creation Error\n", i);
        exit(EXIT_FAILURE);
    }
}
pthread_attr_destroy(&attr);
sleep(init->work_time);

for (i = 0; i < init->num_readers; i++) {
```

```
    if (pthread_cancel(consumers[i]) != 0) {
        fprintf(stderr, "Reader Thread %d Canceling Error\n", i);
        exit(EXIT_FAILURE);
    }
}
for (i = 0; i < init->num_readers; i++) {
    if (pthread_join(consumers[i], NULL) != 0) {
        fprintf(stderr, "Reader Thread %d Waiting Error\n", i);
        exit(EXIT_FAILURE);
    }
}
printf("\t\tThe Reader Thread has been Stopped!!\n");
for (i = 0; i < init->num_writers; i++) {
    if (pthread_cancel(producers[i]) != 0) {
        fprintf(stderr, "Writer Thread %d Canceling Error\n", i);
        exit(EXIT_FAILURE);
    }
}
for (i = 0; i < init->num_writers; i++) {
    if (pthread_join(producers[i], NULL) != 0) {
        fprintf(stderr, "Writer Thread %d Waiting Error\n", i);
        exit(EXIT_FAILURE);
    }
}
printf("\t\tThe Writer Thread has been Stopped!!\n");

free(producers);
```

```
    free(consumers);
}
```

Task 2

Main

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

#include "buffer.h"
#include "task.h"

int main() {
    TPARAM init;
    BUFFER buf;
    initializer(&buf, 7);

    init.buf = &buf;
    init.num_readers = 5;
    init.num_writers = 6;
    init.num_reps = 100;
    init.work_time = 5;
    task_solution(&init);

    destructor(&buf);

    printf("\nFinish\n");
```

```
    return EXIT_SUCCESS;
}
```

Task

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

#include "task.h"
#include "threads.h"

typedef struct {
    pthread_t * thread;
    TARG * arg;
} MAKE_RES;

static pthread_t make_detached(BUFFER * buf);
static MAKE_RES * make_producers(TARG * targ);
static MAKE_RES * make_consumers(TARG * targ);

void task_solution(TPARAM * init) {
    pthread_t thread;
    TARG arg_producers, arg_consumers;
    MAKE_RES * producers;
    MAKE_RES * consumers;
    int i;
```

```
thread = make_detached(init->buf);
arg_producers.buf = init->buf;
arg_producers.num_reps = init->num_reps;
arg_producers.num_thread = init->num_writers;
producers = make_producers(&arg_producers);

arg_consumers.buf = init->buf;
arg_consumers.num_reps = init->num_reps;
arg_consumers.num_thread = init->num_readers;
consumers = make_consumers(&arg_consumers);

sleep(init->work_time);
for (i = 0; i < init->num_writers; i++) {
    if (pthread_cancel(producers->thread[i]) != 0) {
        fprintf(stderr, "Writer Thread %d Canceling Error\n", i);
        exit(EXIT_FAILURE);
    }
}
for (i = 0; i < init->num_readers; i++) {
    if (pthread_cancel(consumers->thread[i]) != 0) {
        fprintf(stderr, "Reader Thread %d Canceling Error\n", i);
        exit(EXIT_FAILURE);
    }
}

for (i = 0; i < init->num_writers; i++) {
    if (pthread_join(producers->thread[i], NULL) != 0) {
```

```
            fprintf(stderr, "Writer Thread %d Waiting Error\n", i);
            exit(EXIT_FAILURE);
        }
    }
    for (i = 0; i < init->num_readers; i++) {
        if (pthread_join(consumers->thread[i], NULL) != 0) {
            fprintf(stderr, "Reader Thread %d Waiting Error\n", i);
            exit(EXIT_FAILURE);
        }
    }

    free(producers->thread);  free(producers->arg);  free(producers);
    free(consumers->thread);  free(consumers->arg);  free(consumers);

    if(pthread_cancel(thread)) {
        fprintf(stderr, "ERROR! Cannot stop detached thread!\n");
        exit(EXIT_FAILURE);
    }
}

static pthread_t make_detached(BUFFER * buf) {
    pthread_attr_t attr;
    pthread_t thread;

    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
    if (pthread_create(&thread, &attr, &detach_thread, (void*)buf)) {
```

```
        fprintf(stderr, "Error while Creation Detached Thread\n");
        exit(EXIT_FAILURE);
    }
    pthread_attr_destroy(&attr);

    return thread;
}


static MAKE_RES * make_producers(TARG * targ) {
    MAKE_RES * res;
    TARG * arg;
    int i;

    res = (MAKE_RES *) malloc(sizeof(MAKE_RES));
    if (!res) {
        fprintf(stderr, "Allocation memory error\n");
        exit(EXIT_FAILURE);
    }
    res->thread = (pthread_t *) calloc(targ->num_thread, sizeof(pthread_t));
    res->arg = (TARG *) calloc(targ->num_thread, sizeof(TARG));
    if ((res->thread == NULL) || (res->arg == NULL)) {
        fprintf(stderr, "Allocation memory error\n");
        exit(EXIT_FAILURE);
    }

    for (i = 0; i < targ->num_thread; i++) {
```

```
        res->arg[i].num_reps = targ->num_reps;
        res->arg[i].num_thread = i;
        res->arg[i].buf = targ->buf;
        if (pthread_create(&(res->thread[i]), NULL, &producer_thread, &(res->arg[i]))) {
            fprintf(stderr, "Writer Thread %d Creation Error\n", i);
            exit(EXIT_FAILURE);
        }
    }

    return res;
}


static MAKE_RES * make_consumers(TARG * targ) {
    MAKE_RES * res;
    TARG * arg;
    int i;

    res = (MAKE_RES *) malloc(sizeof(MAKE_RES));
    if (!res) {
        fprintf(stderr, "Allocation memory error\n");
        exit(EXIT_FAILURE);
    }
    res->thread = (pthread_t *) calloc(targ->num_thread, sizeof(pthread_t));
    res->arg = (TARG *) calloc(targ->num_thread, sizeof(TARG));
    if ((res->thread == NULL) || (res->arg == NULL)) {
        fprintf(stderr, "Allocation memory error\n");
```

```
        exit(EXIT_FAILURE);
    }

    for (i = 0; i < targ->num_thread; i++) {
        res->arg[i].num_reps = targ->num_reps;
        res->arg[i].num_thread = i;
        res->arg[i].buf = targ->buf;
        if (pthread_create(&(res->thread[i]), NULL, &consumer_thread, &(res->arg[i]))) {
            fprintf(stderr, "Reader Thread %d Creation Error\n", i);
            exit(EXIT_FAILURE);
        }
    }

    return res;
}
```

Результат:

Task 1

```
gcc main.c task.c threads.c buffer.c -lpthread -lrt -o
./main.out
Producer(1375377152): = 0
        Consumer(1366984448): = 0
Producer(1383769856): = 0
        Consumer(1358591744): = 0
Producer(1375377152): = 4
        Consumer(1366984448): = 4
Producer(1383769856): = 8
        Consumer(1358591744): = 8
Producer(1375377152): = 2
        Consumer(1358591744): = 2
Producer(1383769856): = 0
        Consumer(1366984448): = 0
Producer(1375377152): = 9
        Consumer(1366984448): = 9
Producer(1383769856): = 8
        Consumer(1358591744): = 8
Producer(1375377152): = 0
        Consumer(1358591744): = 0
Producer(1383769856): = 6
        Consumer(1366984448): = 6
                The Reader Thread has been Stopped!!
                The Writer Thread has been Stopped!!
```

Task 2

```
gcc main.c task.c threads.c buffer.c -lpthread -lrt -
./main.out
        Writer Thread 1 is started
        Writer Thread 2 is started
        Writer Thread 3 is started
        Writer Thread 4 is started
        Writer Thread 5 is started
                Reader Thread 0 is started
                Reader Thread 1 is started
                Reader Thread 2 is started
                Reader Thread 3 is started
                Reader Thread 4 is started
        Writer Thread 0 is started
  0.0000   0.0000   0.0000   0.0000   0.0000   0.000
        Writer Thread No 0, res = 0.626561
                Reader Thread No 0, res = 0.626561
        Writer Thread No 0, res = 0.526604
        Writer Thread No 0, res = 0.923369
        Writer Thread No 1, res = 0.984432
```

```
                Reader Thread 3 is started
                Reader Thread 4 is started
        Writer Thread 0 is started
   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
        Writer Thread No 0, res = 0.626561
                Reader Thread No 0, res = 0.626561
        Writer Thread No 0, res = 0.526604
        Writer Thread No 0, res = 0.923369
        Writer Thread No 1, res = 0.984432
                Reader Thread No 1, res = 0.984432
   0.5266   0.9234   0.0000   0.0000   0.0000   0.0000   0.0000
        Writer Thread No 2, res = 0.346078
                Reader Thread No 2, res = 0.346078
   0.5266   0.9234   0.0000   0.0000   0.0000   0.0000   0.0000
        Writer Thread No 3, res = 0.697711
                Reader Thread No 1, res = 0.697711
                Reader Thread No 3, res = 0.923369
   0.5266   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
                Reader Thread No 0, res = 0.526604
        Writer Thread No 0, res = 0.76816
        Writer Thread No 4, res = 0.0573885
                Reader Thread No 4, res = 0.0573885
   0.7682   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000

Finish
```