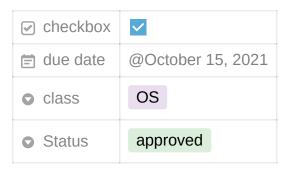


# OS2 Lab14



## Task

```
Решите задачу 11 с использованием двух семафоров-счетчиков
```

## **Notes**

```
#include <stdio.h>
#include <pthread.h>
#include <string.h>
#include <stdbool.h>
#include <semaphore.h>
sem_t sem1;
sem_t sem2;
void* threadRoutine(void* args){
    for(int i = 0; i < 10; i++){
        sem_wait(&sem2);
        printf("pupu\n");
        sem_post(&sem1);
    }
}
int main(){
    sem_init(&sem1, 0, 1);
    sem_init(&sem2, 0, 0);
    pthread_t childThread;
    if (pthread_create(&childThread, NULL, threadRoutine, NUL
L)){
        perror("pthread_create:");
    }
    for(int i = 0; i < 10; i++){
        sem_wait(&sem1);
        printf("pipi\n");
        sem_post(&sem2);
    }
    sem_destroy(&sem1);
    sem_destroy(&sem2);
    pthread_exit(0);
}
```

### Семафоры-счетчики

Исторически, семафоры-счетчики были одним из первых примитивов синхронизации. В старых учебниках они известны под названием "семафоры Дийкстры". Семафор представляет собой целочисленную переменную, над которой определены две операции, post и wait.

Операция wait пытается вычесть единицу из флаговой переменной семафора. Если значение флаговой переменной больше 0, то происходит обычное вычитание и нить, выполнившая операцию wait, продолжает исполнение. Если значение флаговой переменной было равно 0, то wait блокируется до тех пор, пока кто-то (скорее всего, другая нить) не увеличит значение этой переменной.

Операция post добавляет единицу к флаговой переменной семафора. Если при этом на семафоре ждала нить, операция post пробуждает эту нить.

Семафоры можно использовать в качестве примитивов взаимоисключения, при этом операция wait аналогична захвату мутекса, а операция post - операции его освобождения. Однако, в отличие от мутексов, операции post и wait не обязаны выполняться одной и той же нитью и даже не обязаны быть парными. Это позволяет использовать семафоры в различных ситуациях, которые сложно или невозможно разрешить при помощи мутексов. Иногда семафоры используют в качестве разделяемых целочисленных переменных, например в качестве счетчиков записей в очереди.

B Solaris функции работы с семафорами-счетчиками включены в библиотеку librt.so. Их использование требует сборки программы с

98

д.В. Иртегов Многопоточное программирование с использованием POSIX Threads ключом -lrt. В отличие от остальных функций POSIX Thread API, функции работы с семафорами и сам тип семафора не имеют префикса pthread\_.

Как и остальные примитивы взаимодействия, рассматривавшиеся ранее, семафор POSIX представляет собой непрозрачный тип данных, операции над которым должны осуществляться специальными функциями. Этот тип называется  $sem_t$  и определен в файле  $sem_t$ .

OS2 Lab14 1

### SYNOPSIS

#include <semaphore.h>

int sem\_init(sem\_t \*sem, int pshared, unsigned int value);

### **DESCRIPTION**

The <code>sem\_init()</code> function is used to initialize the unnamed semaphore referred to by  $\underline{sem}$ . The value of the initialized semaphore is  $\underline{value}$ . Following a successful call to  $\underline{sem_init()}$ , the semaphore may be used in subsequent calls to  $\underline{sem_wait(3C)}$ ,  $\underline{sem_trywait(3C)}$ ,  $\underline{sem_post(3C)}$ , and  $\underline{sem_destroy(3C)}$ . This semaphore remains usable until the semaphore is destroyed.

If the <u>pshared</u> argument has a non-zero value, then the semaphore is shared between processes; in this case, any process that can access the semaphore  $\underline{sem}$  can use  $\underline{sem}$  for performing  $\underline{sem}$  <u>wait(3C)</u>,  $\underline{sem}$  <u>trywait(3C)</u>,  $\underline{sem}$  <u>post(3C)</u>, and  $\underline{sem}$  <u>destroy(3C)</u> operations.

Only  $\underline{sem}$  itself may be used for performing synchronization. The result of referring to copies of  $\underline{sem}$  in calls to  $\underline{sem}$   $\underline{wait(3C)}$ ,  $\underline{sem}$   $\underline{trywait(3C)}$ ,  $\underline{sem}$   $\underline{post(3C)}$ , and  $\underline{sem}$   $\underline{destroy(3C)}$ , is undefined.

If the <u>pshared</u> argument is zero, then the semaphore is shared between threads of the process; any thread in this process can use  $\underline{sem}$  for performing  $\underline{sem}$  wait(3C),  $\underline{sem}$  trywait(3C),  $\underline{sem}$  post(3C), and  $\underline{sem}$  destroy(3C) operations. The use of the semaphore by threads other than those created in the same process is undefined.

Attempting to initialize an already initialized semaphore results in undefined behavior.

# **Reading list**