Лабораторная работа №7
«Семафоры»

Теоретические сведения

Семафоры в ОС Линукс реализованы двух типов: согласно стандарту **System V IPC** и удовлетворяющие стандарту **POSIX-1.2001.**

Семафоры первого типа имеют более широкий функционал по сравнению с классическими семафорами Дейкстры (даже примитивов не два, а больше), однако считаются устаревшими (как и сама System V). Поэтому изучение данного вида семафоров ограничим только вашим знакомством с ними через **man** (например, посмотрите команды semop, semget и т.д.).

Второй тип семафоров, т.н. POSIX-семафоры, намного проще и намного лучше проработаны. Они реализуют классические семафоры Дейкстры. Именно эти семафоры настоятельно рекомендуется использовать в данной работе. Недостаток у данных семафоров один, но весьма существенный: до версии 2.6 ядра Linux они были реализованы только с поддержкой нескольких нитей одного процесса (были только неименованные семафоры).

Начиная с ядра 2.6 появились именованные семафоры, которые могут использоваться несколькими процессами. Начиная с ядра 2.6, POSIX-семафоры полноценно могут быть вами использованы.

POSIX-семафоры позволяют синхронизировать работу процессам и потокам.

Семафоры

Семафор - целое число, которое не может быть меньше нуля. Две операции могут производиться над семафорами: увеличение семафора на 1 (**sem_post**) и уменьшение на 1 (**sem_wait**). Если значение семафора в данный момент равно 0, то вызов sem_wait переводит процесс (поток) в состояние блокировки до тех пор, пока семафор не станет больше 0.

Начиная с ядра Linux 2.6 POSIX-семафоры существуют двух видов: именованные и неименованные.

Именованные семафоры идентифицируются именем формы somename. Два процесса могут оперировать одним и тем же семафором, передавая его имя в sem_open. Данная функция создает новый именованный семафор либо открывает существующий. В дальнейшем к нему можно применять операции sem_post и sem_wait. Когда процесс завершает использование семафора, тот может быть удален из системы с использованием **sem_unlink** (иначе он будет существовать до завершения работы ОС).

Неименованные семафоры располагаются в области памяти, которая является разделяемой между несколькими потоками одного процесса (a thread-shared semaphore) или несколькими процессами (a process-shared semaphore). Например, в глобальной переменной. Разделяемый между процессами семафор должен располагаться в разделяемой памяти (например, построенной с использованием shm_open). Перед использованием неименованный семафор должен быть проинициализирован с использованием **sem_init**. После с ним также можно оперировать sem_post и sem_wait. Когда он больше не нужен, уничтожается с помощью **sem_destroy**.

Внимание! Программы, использующие POSIX семафоры, должны компилироваться с ключом **-lrt** или **-pthread** , чтобы подключать библиотеку librt.

В файловой системе Линукс именованные семафоры обычно монтируются в /dev/shm, с именем типа sem.name.

Основные системные вызовы по работе с семафорами

```
#include <semaphore.h>
sem_t *sem_open(const char *name, int oflag);
sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value);
```

Описание системного вызова

Открывает и инициализирует именованный семафор с именем ***name***, правами доступа ***mode***, начальным значением ***value***.

Пример:

```
char sem_name1[]="mysemaphore1";
char sem_name2[]="mysemaphore2";
sem_t *s1 = sem_open(sem_name1, O_CREAT);
sem_t s2 = sem_open(sem_name2, O_CREAT, 0644, 10);
```

Открывается семафор с именем mysemaphore1, уже имеющийся в системе, значение и права уже установлены. Открывается семафор с именем mysemaphore2 , правами доступа 0644(rw-r--r--) и начальным значением 10.

```
#include <semaphore.h>
int sem_close(sem_t *sem);
```

Описание системного вызова

Закрывает семафор, но семафор все еще существует в системе в /dev/shm.

Пример:

```
sem_close(s1);
sem_close(&s2);
```

```
#include <semaphore.h>
int sem_post(sem_t *sem);
```

Описание системного вызова

Увеличивает значение семафора на 1, тем самым разблокирует другой процесс, который блокировался на этом семафоре.

Пример:

```
sem_post(s1);
sem_post(&s2);
```

```
#include <semaphore.h>
int sem_wait(sem_t *sem);
int sem_trywait(sem_t *sem);
int sem_timedwait(sem_t *sem, const struct timespec *abs_timeout);
```

Описание системного вызова

Блокирует процесс на семафоре, если тот равен 0, иначе уменьшает его на 1. sem_trywait вместо блокирования процесса при невозможности уменьшения семафора вызывает ошибку. sem_timedwait устанавливает предельное время блокировки.

Пример:
sem_wait(s1);
sem_wait(&s2);

Прототип системного вызова

```
#include <semaphore.h>
int sem_getvalue(sem_t *sem, int *sval);
```

Описание системного вызова

Функция sem_getvalue возвращает текущее значение семафора, помещая его в целочисленную переменную, на которую указывает sval. Если семафор заблокирован, возвращается либо 0, либо отрицательное число, модуль которого соответствует количеству потоков, ожидающих разблокирования семафора.

Пример:

```
int val;
sem_getvalue(sem, &val);
printf("Semaphore value = %d\n", val);
```

Больше информации в man sem_overview и man по соответствующим системным вызовам

Мьютэксы

Мьютекс – примитив синхронизации, обеспечивающий взаимное исключение исполнения критических участков кода. Классический мьютекс отличается от двоичного семафора наличием эксклюзивного владельца, который и должен его освобождать (т.е. переводить в незаблокированное состояние)

Условно классический мьютекс можно представить в виде переменной, которая может находиться в двух состояниях: в заблокированном и в незаблокированном. При входе в свою критическую секцию поток вызывает функцию перевода мьютекса в заблокированное состояние, при этом поток блокируется до освобождения мьютекса, если другой поток уже владеет им. При выходе из критической секции поток вызывает функцию перевода мьютекса в незаблокированное состояние. В случае наличия нескольких заблокированных по мьютексу потоков во время разблокировки планировщик выбирает поток для возобновления выполнения (в зависимости от реализации это может быть, как случайный, так и детерминированный по некоторым критериям поток).

Основные системные вызовы по работе с мьютексами

Прототип системного вызова

```
#include <pthread.h>
int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr)
```

Создание и инициализация мьютекса.

Пример:

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

или

pthread_mutex_t mutex;
pthread_mutex_init(&mutex, NULL);

Прототип системного вызова

```
#include <pthread.h>
pthread_mutex_destroy(pthread_mutex_t *mutex)
```

Уничтожение мьютекса.

Пример:
pthread_mutex_destroy(&mutex);

Далее системные вызовы имеют идентичный пример использования.

Прототип системного вызова

```
#include <pthread.h>
pthread_mutex_lock(pthread_mutex_t *mutex)
```

Перевод мьютекса в заблокированное состояние (захват мьютекса).

Прототип системного вызова

```
#include <pthread.h>
pthread_mutex_trylock(pthread_mutex_t *mutex)
```

Попытка перевода мьютекса в заблокированное состояние, и возврат ошибки в случае, если должна произойти блокировка потока из-за того, что у мьютекса уже есть владелец.

Прототип системного вызова

```
#include <pthread.h>
pthread_mutex_timedlock(pthread_mutex_t *mutex)
```

Попытка перевода мьютекса в заблокированное состояние, и возврат ошибки в случае, если попытка не удалась до наступления указанного момента времени.

Прототип системного вызова

```
#include <pthread.h>
pthread_mutex_unlock(pthread_mutex_t *mutex)
```

Перевод мьютекса в незаблокированное состояние (освобождение мьютекса).

Пример решения задачи производителя-потребителя с использованием семафоров

Producer.c

```
#include <semaphore.h>
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
```

```
#include <stdio.h>
#include <sys/msg.h>
#include <sys/sem.h>
#include <unistd.h>
#define buf 5

int main()
{
 printf("Producer\n");
 sem_t *full;
 sem_t *empty;
 sem_t *mutex;
 const char *sem_full="full";
 const char *sem_empty="empty";
 const char *sem_mutex="mutex";
 full= sem_open(sem_full,O_CREAT,0777,0);
 empty= sem_open(sem_empty,O_CREAT,0777,buf);
 mutex=sem_open(sem_mutex,O_CREAT,0777,1);
 while(1){
       sem_wait(empty);
       sem_wait(mutex);
       printf("produce_item\n");
       sleep(1);
       sem_post(mutex);
       sem_post(full);
 }
 sem_close(full);
 sem_close(empty);
 sem_close(mutex);
 sem_unlink(sem_full);
 sem_unlink(sem_empty);
 sem_unlink(sem_mutex);
return 0;
}
```

Consumer.c

```
#include <semaphore.h>
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/msg.h>
#include <sys/sem.h>
#include <unistd.h>
#define buf 5

int main()
{
 printf("Consumer\n");
 sem_t *full;
 sem_t *empty;
 sem_t *mutex;
 const char *sem_full="full";
 const char *sem_empty="empty";
 const char *sem_mutex="mutex";
 full= sem_open(sem_full,O_CREAT,0777,0);
 empty= sem_open(sem_empty,O_CREAT,0777,buf);
 mutex=sem_open(sem_mutex,O_CREAT,0777,1);
 while(1){
       sem_wait(full);
       sem_wait(mutex);
       printf("consume_item\n");
       sleep(1);
       sem_post(mutex);
       sem_post(empty);
```

```
    }
    sem_close(full);
    sem_close(empty);
    sem_close(mutex);
    sem_unlink(sem_full);
    sem_unlink(sem_empty);
    sem_unlink(sem_mutex);
  return 0;
  }
```

В примере вместо настоящего мьютекса был использован бинарный семафор, в классическом его варианте. Для более верного решения использовать следует pthread_mutex системные вызовы и переменные.

Задание для выполнения

Ознакомиться с руководством, теоретическими сведениями и лекционным материалом по использованию и функционированию средств синхронизации - **семафоров Дейкстры**, и их реализацией в Linux - System V IPC семафоры и POSIX-семафоры.

Написать две (или более) программы, которые, работая параллельно зацикленно, обмениваются информацией согласно варианту. **Передачу и получение информации** каждым из процессов сопровождать выводом на экран информации типа "процесс такой-то передал/получил такую-то информацию". Синхронизацию работы процессов реализовать с помощью семафоров. Учтите, что **при организации совместного доступа к разделяемому ресурсу (например, файлу)** вам понадобится применять мьютексы.

Для наглядности запускайте свои процессы в разных окнах терминала. Запустите программы в нескольких экземплярах (одну первую и две/три вторых, две первых и две вторых...).

Варианты индивидуальных заданий

Вариант 1. Первый процесс в цикле ожидает ввода символа с потока stdin, после чего пишет в файл случайное число, каждый раз открывая и закрывая за собой файл. Второй эти числа из файла забирает и выводит на экран.

Вариант 2. Первый процесс семафорами передаёт второму число, а второй умножает его на свой pid и выводит значение на экран. *Внимание: никаких средств взаимодействия (файлов, pipe, ...), кроме семафоров и мьютексов, не использовать.*

Вариант 3. Первый процесс в цикле ожидает ввода символа с потока stdin, после чего передает второму процессу соответствующий символ. Второй получает и выводит его на экран случайное количество раз. *Внимание: никаких средств взаимодействия (файлов, pipe, ...), кроме семафоров и мьютексов, не использовать!*

Вариант 4. Первый процесс в цикле ожидает ввода символа с потока stdin, после чего пишет в файл соответствующий символ, если он является согласной буквой, каждый раз открывая и закрывая за собой файл. Второй процесс забирает из файла символы и выводит на экран.

Вариант 5. Первый процесс в цикле ожидает ввода символа с потока stdin, после чего пишет в файл случайное число, каждый раз открывая и закрывая за собой файл. Второй процесс забирает из файла числа и выводит на экран соответствующее числу количество любых символов.

Вариант 6. Первый процесс в цикле ожидает ввода символа с потока stdin, после чего пишет в файл соответствующий символ случайное количество раз, каждый раз открывая и закрывая за собой файл. Второй процесс забирает из файла символы и выводит на экран их количество.

Вариант 7. Первый процесс пишет в файл строки вида «pid - текущее время», каждый раз открывая и закрывая файл, а второй процесс эти строки читает и выводит, дополняя своим pid.

Вариант 8. Первый процесс в цикле ожидает ввода символа с потока stdin, после чего пишет в файл символ, если он не является ни цифрой, ни буквой, каждый раз открывая и закрывая за собой файл. Второй процесс забирает из файла символы и выводит на экран.

Вариант 9. Первый процесс семафорами передаёт второму текущее время, а второй выводит его в форматированном для человека виде. *Внимание: никаких средств взаимодействия (файлов, pipe, ...), кроме семафоров и мьютексов, не использовать!*

Вариант 10. Первый процесс в цикле ожидает ввода символа с потока stdin, после чего пишет в файл соответствующий символ, каждый раз открывая и закрывая за собой файл. Второй процесс считывает символ из файла и передает его ASCII через семафор первому процессу. *Внимание: никаких средств взаимодействия (файлов, pipe, ...), кроме семафоров и мьютексов, не использовать!*

Вариант 11. Первый процесс семафорами передаёт второму величины сторон прямоугольника, а второй вычисляет значение периметра и возвращает первому семафорами. *Внимание: никаких средств взаимодействия (файлов, pipe, ...), кроме семафоров и мьютексов, не использовать!*

Вариант 12. Первый процесс в цикле ожидает ввода символа с потока stdin, после чего пишет в файл соответствующий символ, каждый раз открывая и закрывая за собой файл. Второй процесс забирает из файла символ и выводит его на экран несколько раз подряд.

Вариант 13. Первый процесс в цикле ожидает ввода символа с потока stdin, после чего пишет в файл соответствующий символ, каждый раз открывая и закрывая за собой файл. Второй процесс забирает из файла символы и выводит на экран только гласные буквы.

Вариант 14. Первый процесс в цикле ожидает ввода символа с потока stdin и пишет в файл строки вида «символ - текущее время», каждый раз открывая и закрывая файл, а второй процесс эти строки читает и выводит, дополняя своим pid.

Вариант 15. Первый процесс в цикле ожидает ввода строки с потока stdin и пишет в файл, каждый раз открывая и закрывая файл, а второй процесс эти строки читает и выводит на экран только гласные буквы.

Вариант 16. Первый процесс в цикле ожидает ввода символа с потока stdin, после чего передает второму процессу соответствующий символ. Второй получает и выводит его на экран. *Внимание: никаких средств взаимодействия (файлов, pipe, ...), кроме семафоров и мьютексов, не использовать!*

Вариант 17. Первый процесс в цикле ожидает ввода строки с потока stdin и пишет в файл, каждый раз открывая и закрывая файл, а второй процесс эти строки читает и выводит на экран только цифре, если цифр нет, то выводит сообщение в поток ошибок сообщение «error».

Вариант 18. Первый процесс семафорами передаёт второму величины катетов, а второй вычисляет значение гипотенузы и возвращает первому семафорами. *Внимание: никаких средств взаимодействия (файлов, pipe, ...), кроме семафоров и мьютексов, не использовать!*

Вариант 19. Первый процесс в цикле ожидает ввода символа с потока stdin, после чего пишет в файл соответствующий символ, каждый раз открывая и закрывая за собой файл. Второй процесс забирает из файла символ и выводит на экран номер этого символа в ASCII.

Вариант 20. Процессы строят числа Фибоначчи, поочередно вычисляя очередное число, выводя его на экран и передавая его другому процессу, чтобы тот вычислил следующее. *Внимание: никаких средств взаимодействия (файлов, pipe, ...), кроме семафоров и мьютексов, не использовать!*

Отчет

Как и в других работах, отчет по проделанной работе представляется преподавателю в стандартной форме: на листах формата А4, с титульным листом (включающим тему, фио, номер зачетки и пр.), целью, ходом работы и выводами по выполненной работе. Каждое задание должно быть отражено в отчете следующим образом: 1) что надо было сделать, 2) как это сделали, 3) что получилось, и, в зависимости от задания – 4) почему получилось именно так, а не иначе. При наличии заданий с вариантами необходимо указать свой вариант.