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...).

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. Ubuntu 7.10 использует ядро 2.6, потому POSIX-семафоры полноценно могут быть вами использованы.

POSIX-семафоры позволяют синхронизировать работу процессам и потокам.

Семафор - целое чило, которе 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 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 може 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(3). После с ним также можно оперировать sem_post и sem_wait(3). Когда он больше не нужен, уничтожается с помощью sem_destroy.

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

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

Больше информации в man sem_overview и man по соответствующим системным вызовам.

sem_open Открывает и инициализирует именованный семафор

```
#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);
```

Пример:

```
char sem_name1[]="mysemaphore1";
char sem_name2[]="mysemaphore2";
sem_t *s1 = sem_open(sem_name1, O_CREAT, 0777, 0);
sem_t s2 = sem_open(sem_name2, O_CREAT, 0644, 10);
```

Открывается семафор с именем mysemaphore1 , правами доступа 0777(rwxrwxrwx) и начальным значением 0. Открывается семафор с именем mysemaphore2 , правами доступа 0644(rw-r--r--) и начальным значением 10.

sem_close – Закрывает семафор

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

Пример:

```
sem_close(s1);
```

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

sem_post – Увеличивает значение семафора на 1, тем самым разблокируя другой процесс, который блокировался на этом семафоре.

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

Пример:

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

sem_wait – 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.

```
#include <semaphore.h>
int sem_wait(sem_t *sem);
int sem_trywait(sem_t *sem);
#define _XOPEN_SOURCE 600
#include <semaphore.h>
int sem_timedwait(sem_t *sem, const struct timespec *abs_timeout);
```

Пример:

```
sem_wait(s1);
sem_wait(&s2);
```

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, например, 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, 11. 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, 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 3, 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 4, 14. 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, 15. 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, 16. Первый процесс в цикле ожидает ввода символа с потока stdin, после чего передает второму процессу соответствующий символ. Второй получает и выводит на экран. *Внимание: никаких средств взаимодействия (файлов, pipe, ...), кроме семафоров и мьютексов, не использовать!*

Вариант 7, 17. 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, 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 9, 19. 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, 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, и расчет его номера, а также всѐ вышеуказанное для данного варианта задания.