Лабораторная работа №2. Синхронизация потоков.

Цель работы: получение практических навыков по организации синхронизации потоков и процессов.

Теоретические сведения

Создание потоков в ОС Linux

Создание потока в ОС Linux означает создание дочернего процесса, разделяющего определенные ресурсы со своим родителем. Изначально это осуществлялось с помощью системного вызова clone(CLONE_VM | CLONE_FS | CLONE_FILES | CLONE_SIGHAND, 0), где переданные флаги указывают вид разделяемого ресурса:

- CLONE_VM - родительский и дочерний процесс разделяют адресное пространство, то есть любые модификации памяти и создание отображений в память дочерним процессом будут видны родительскому и наоборот;
- CLONE_FS - родительский и дочерний процесс разделяют информацию о файловой системе (корневой каталог файловой системы, текущий рабочий каталог и т. д.);
- CLONE_FILES - родительский и дочерний процесс разделяют таблицу дескрипторов открытых файлов;
- CLONE_SIGHAND - родительский и дочерний процесс разделяют таблицу обработчиков сигналов.

Однако, после внедрения в Linux стандарта POSIX, создание дополнительного потока исполнения можно произвести при помощи функции:

```
int pthread_create(
    pthread_t *thread,
    const pthread_attr_t *attr,
    void* (*start_routine)(void*),
    void *arg);
```

Данная функция запускает новый поток исполнения, вызывая функцию *start_routine* с аргументом *arg*. Эта функция предоставляется библиотекой *libpthread.so*, которую в свою очередь необходимо прилинковать к компилируемой программе указав компилятору флаг *-pthread*.

Механизмы синхронизации в ОС Linux (POSIX)

1. **Мьютекс**. Библиотека pthread также предоставляет возможность организации блокировки типа мьютекс. Для этого сперва должна быть создана переменная типа *pthread_mutex_t* в области памяти доступной всем потокам данного процесса. Затем созданную переменную нужно проинициализировать с помощью функции

```
int pthread_mutex_init(pthread_mutex_t *mutex,
                       const pthread_mutexattr_t *attr);
```

Параметр *attr* может быть установлен в *NULL*. По окончанию использования мьютекса он должен быть удален с помощью функции

```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

Основные операции над мьютексом выполняются посредством следующих функций:

- *int pthread_mutex_lock(pthread_mutex_t *mutex)* - производит блокировку мьютекса данным потоком исполнения.
- *int pthread_mutex_trylock(pthread_mutex_t *mutex)* - неблокирующая версия функции *pthread_mutex_lock*, то есть данная функция не блокирует выполнение текущего потока, в случае если мьютекс уже был заблокирован другим потоком, а только возвращает ненулевое значение.
- *int pthread_mutex_unlock(pthread_mutex_t *mutex)* - разблокирует ранее заблокированный мьютекс.

2. **Семафор,** определенный стандартом POSIX, позволяет организовывать синхронизацию как между потоками, так и между процессами. Для этого в стандарте определен неименованный и именованный семафор. Переменная семафора имеет тип *sem_t*. Переменная соответствующая неименованному семафору должна быть помещена в память, разделяемую потоками или процессами, а затем инициализирована с помощь функции

```
int sem_init(sem_t *sem, int pshared, unsigned int value);
```

В результате будет создан семафор инициализированный числом *value*. Также, если значение pshared отлично от 0, семафор будет разделяться процессами имеющими доступ к участку памяти, где храниться его переменная. Именованный семафор создаётся или открывается с помощью функции

```
sem_t *sem_open(const char *name, int oflag,
                mode_t mode, unsigned int value);
```

Имя семафора должно начинаться с символа /, за которым следует непосредственно имя семафора.

Операции над семафором могут быть выполнены с помощью функций:

- *int sym_post(sem_t*)* - инкрементирует значение семафора.
- *int sym_wait(sem_t*)* - декрементирует значение семафора. В случае, если исходное значение семафора равно нулю, то выполняющийся поток блокируется ожидая увеличения значения семафора.

- *int sym_trywait(sem_t*)* - неблокирующий поток исполнения аналог функции *syn_wait*.

Организация синхронизации между процессами

Предоставляемые библиотекой pthread средства синхронизации хранящие свое состояние в переменных, созданных в памяти, разделяемой потоками исполнения данного процесса, не позволяют произвести синхронизацию некоторого ресурса между процессами. Однако этого можно добиться выполнив ряд дополнительных действий при создании переменных объекта синхронизации.

Во-первых, переменная объекта синхронизации должна быть создана в области памяти разделяемой несколькими процессами (не находящимися в отношении отец и сын в дереве процессов). Разделение памяти между процессами можно обеспечить отобразив в память вспомогательный файл с одним и тем же именем с помощью системного вызова *mmap*. Отображаемый файл должен быть открыт с помощью системного вызова open с режимом 0666 и следующим набором флагов:

- O_RDWR - файл открывается для чтения и записи
- O_CREAT - в случае, если файл не существует, то он создаётся.
- O_EXCL - заставляет системный вызов *open* вернуть значение ошибки (-1), в случае, если файл уже существует.

Последний флаг позволяет выделить процесс создавший файл, который впоследствии должен выполнить инициализацию общего для нескольких процессов объекта синхронизации. Если данный процесс создал файл, то он должен подогнать размер файла под размер объекта синхронизации с помощью функции *ftruncate(fd, sizeof(объект_синхронизации))*. Те процессы для которых создание файла завершилось ошибкой, должны повторно выполнить открытие файла используя только флаг O_RDWR.

Открытый файл затем необходимо отобразить в память процесса с помощью системного вызова *mmap(NULL, sizeof(объект_синхронизации), PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0)*. Здесь комбинация *PROT_READ | PROT_WRITE* , *MAP_SHARED* - обеспечивает разделение страниц памяти, в которые был отображен файл, всеми процессами, открывшими этот файл.

```
int fd = open("/путь/к/служебному/файлу", O_RDWR | O_CREAT | O_EXCL,
0666);
if (fd < 0) {
  fd = open("/путь/к/служебному/файлу", O_RDWR, 0666);
} else {
  ftruncate(fd, sizeof(объект_синхронизации));
}
void *shared_ptr = mmap(NULL, sizeof(объект_синхронизации),
                        PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
close(fd);
```

Теперь в разделяемом участке памяти необходимо создать и инициализировать объект синхронизации. Однако, по умолчанию эти объекты создаются в режиме

оптимизированном для использования потоками одного процесса. Для создания их в режиме разделения необходимо установить соответствующий атрибут с помощью функции *pthread_mutexattr_setpshared*, куда в качестве значения атрибута передать *PTHREAD_PROCESS_SHARED*.

```
pthread_mutexattr_t psharedm;
pthread_mutex_t *mutex_ptr = shared_ptr;

pthread_mutexattr_init(&psharedm);
pthread_mutexattr_setpshared(&psharedm, PTHREAD_PROCESS_SHARED);
pthread_mutex_init(mutex_ptr, &psharedm);
```

Задания

1. Разработать программу Счётчик, подсчитывающую количество секунд прошедших с момента запуска данного экземпляра программы и выводящую это значение в терминал при каждом его увеличении. Однако, в случае запуска дополнительных экземпляров программы, обновление счетчика должно производиться только одним из экземпляров в текущий момент времени, а через каждые 5 секунд подсчет должен продолжиться на одном из остальных экземпляров программы.
   При реализации программы следует использовать механизм синхронизации, выбранный из списка ниже в соответствии с номером варианта по модулю 2:
   1. Мьютекс
   2. Семафор
2. По выполненной работе нужно оформить отчет, содержащий описание ключевых аспектов реализации и демонстрацию работы написанных программ.

Ссылки

1. POSIX Threads Programming: https://hpc-tutorials.llnl.gov/posix/
2. pthread_mutexattr_init(3) - Linux man page: https://linux.die.net/man/3/pthread_mutexattr_init
3. Basics of Futexes: https://eli.thegreenplace.net/2018/basics-of-futexes/