Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Факультет комп’ютерних наук

# ЛАБОРАТОРНА РОБОТА № 10

з дисципліни «Операційні системи»

Тема: «Процеси. Спільна пам’ять»

Виконав:

студент 3 курсу

групи КС-32

Безрук Юрій Русланович

Перевірив: Споров О.Є.

Харків – 2020

Целью данной работы продолжение изучения основных средств, позволяющих осуществлять взаимодействие между процессами – IPC. В данной работе будет рассмотрено средство для обмена информацией – разделяемая память.

# ХОД РАБОТЫ

**Задание 1**

В ходе выполнения работы требуется написать две программы, обменивающиеся информацией при помощи разделяемой памяти и синхронизировать их работу при помощи сигналов.

При этом одна из программ создает сегмент разделяемой памяти требуемого размера, после чего создаёт процесс-потомок и загружает в него бинарный образ второй программы и начинает ввод данных в общую память, запрашивая их у пользователя. После этого она должна проинформировать вторую программу при помощи сигналов о том, что та должна начать вычисление, и ожидает ответа от нее. Второй же процесс после получения сигнала вычисляет сумму помещенных в общую память чисел и помещает результат в общую память, после чего посылает сигнал первому процессу, который выводит результат в консоль. Таким образом процессы продолжают взаимодействие, пока пользователь не остановит их признаком остановки с клавиатуры. При этом в качестве варианта программы был выбран вариант б) – идентификатор общей области памяти передается второму процессу с помощью дополнительной информации сигнала.

Реализовано это было следующим образом: Первая программа находится в файле main.c и скомпилирована в исполняемый файл shared. Текст второй программы находится в файле second.c и скомпилирован в файл second.

Синхронизация с помощью сигналов была выполнена следующим образом: обе программы имеют целочисленную переменную continue\_flag, которая по умолчания равна 0, и функцию wait\_for\_signal(), в которой находится цикл ожидания положительного значения флага над функцией ожидания сигнала (pause()). Сам же флаг становится равным 1 только в случае, если обработчик сигналов поймал нужный нам сигнал. Таким образом, при вызове функции wait\_for\_signal(), процесс переходит в ожидание приема конкретного сигнала. Сразу же после выхода из цикла значение флага снова обнуляется. Данное решение было использовано, т.к. процессы общаются друг с другом поочередно, отправляя сигналы в обе стороны и состояние гонки носит слабовыраженный характер.

Первым делом программа получает максимальное количество чисел которое может быть передано (по умолчанию 10). После этого создается приватный сегмент общей памяти (что означает, что его не может быть получен никакими другими комбинациями вызовов функции ftok() на имени файла и небольшом целом числе. Делается это при помощи константы IPC\_PRIVATE.) Сегмент выделяется на количество байт, нужное для размещения в памяти переданного количества чисел +1 (для передачи размера массива). Сегмент создаётся с флагами IPC\_CREAT (если сегмента для данного ключа нет, то его нужно создать) и IPC\_EXCL (сегмент должен быть эксклюзивным, т.е., если он уже существует, то это приведен к ошибке), однако, так как область памяти создается с ключом IPC\_PRIVATE, эти флаги не играют роли. Также через побитовое или с ключами передаются права доступа – 0600 (разрешение на чтение и запись).

Далее общая память присоединяется к адресному пространству процесса при помощи функции shmat, после чего проверяем что размер выделенной памяти соответствует тому, который нам нужен при помощи вызова функции shctl() с ключом IPC\_STAT и передаваемой ей структурой shmid\_ds. Ее параметр shm\_segsz указывает на это.

Далее при помощи функции fork() процесс создаёт потомка, который сразу же меняет бинарный образ на другую программу (./second) при помощи функции execlp. Основной же процесс вызывает функцию wait\_for\_signal(), ожидая от потомка сигнала о том, что тот совершил все приготовления и готов к работе.

После того, как сигнал получен, основной процесс создаёт переменную типа union sigval, присваивает ее полю sival\_int значение идентификатора общей памяти, и, при помощи функции sigqueue(), отправляет дочернему процессу сигнал с этими данными, что бы он нашёл общую область памяти.

После этого процесс начинает работу с пользователем. Вначале запрашивается количество чисел, которые будут передаваться. Если передано значение меньшее нуля, процесс закончит свою работу освободив память и завершив дочерний процесс, т.е. это является точкой выхода из программы. Если значение больше размера буфера, то будет установлено значение размера буфера с выводом соответствующего сообщения.

Далее в нулевой элемент массива общих данных записывается количество чисел которое будет передано, а в остальные – считанные из консоли числа. После этого потомку отправляется другой сигнал, означающий, что он может приступить к их обработке, и первая программа ожидает ответа от него о завершении обработки опять же при помощи функции wai\_for\_signal(). Когда сигнал получен, резулльтат выводится в консоль, после чего процесс повторяется заново, пока пользователь не введет отрицательное число в качестве размера массива.

По завершении работы программы процесс отсоединяет общую память, отправляет потомку сигнал SIGTERM, принудительно завершая его, при помощи функции waitpid() дожидается его завершения и удаляет общий сегмент памяти. После этого он завершается сам.

Дочерний процесс устроен так: он также имеет фукцию wait\_for\_signal(). Но обработчик сигналов у него расширенный, более того, он принимает от родителя два вида сигналов. При помощи сигнала SIGRTMIN ему передается идентификатор общей памяти, а SIGRTMIN+1 выполняет обычные функции ожидания. Так же у дочерней программы имеется обработчик завершения, который при любом исходе отсоединяет общую память от процесса.

Первым делом дочерняя программа получает идентификатор родительского процесса функцией getppid() для того что бы ему можно было отправлять сигналы. Далее на всякий случай создается и инициализируется сигнальная маска.

Обработчик сигналов регистрируется при помощи функции sigaction, чтобы иметь возможность получать данные вместе с сигналом. Ей же передается и маска.

На этом подготовления процесса закончены и он отсылает родителю сигнал о том что готов получить идентификатор, после чего ждёт ответного сигнала. Приём ответного сигнала подразумевает, что идентификатор получен, поэтому после выхода из функции wait\_for\_signal(), процесс подключает общую память по идентификатору функцией shmat.

Далее процесс уже заходит в цикл постоянной работы и ждёт сигнала о том, что в память были помещены данные. Когда он его получает, он вычисляет сумму чисел, размещённых в памяти, и результат помещает в нулевой элемент массива, после чего отправляет сигнал о том что вычисления окончены. Далее цикл повторяется и процесс снова ждёт сигнала о приходе данных.

Самостоятельного завершения у процесса нет – его заставляет завершиться родитель, поэтому и предусмотрен обработчик завершения.

После завершения работы проверяем, что нашего сегмента общей памяти больше нет в системе вызовом команды ipcs.

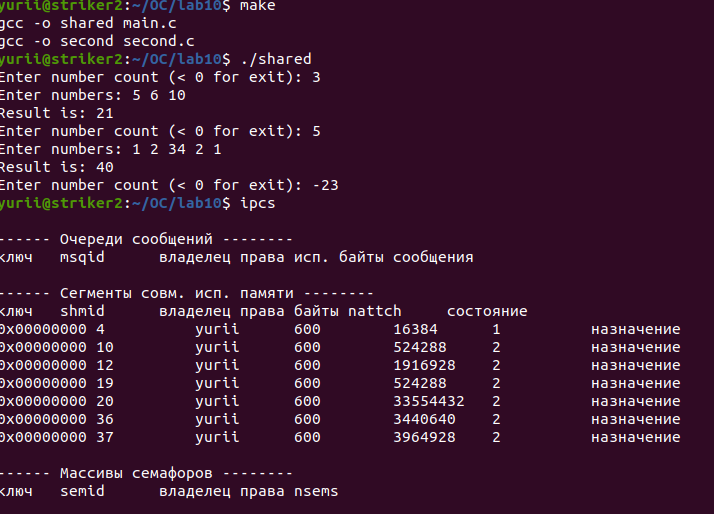


Рисунок 1 – пример работы программы

# ВЫВОДЫ

Таким образом, в ходе выполнения данной работы были рассмотрены сегменты общей памяти и управление ими.Исходные, исполняемые и файлы сборки помещены в архив.