**Лабораторная работа № 7 – Семафоры и синхронизация**

**Цель работы**

Освоение семафоров (semaphores) как эффективных средств синхронизации доступа процессов к разделяемым ресурсам операционной системы, а также синхронизации доступа потоков (в части 2) к разделяемым ресурсам процесса.

**Пункт 1**

Скомпилируйте и выполните программу gener\_sem.cpp , иллюстрирующую создание наборов с семафорами или получение доступа к ним. Запустите программу несколько раз и после каждого ее завершения выполните команду ipcs -s. Поясните зависимость процедуры создания семафоров от используемых в вызове semget() флагов.

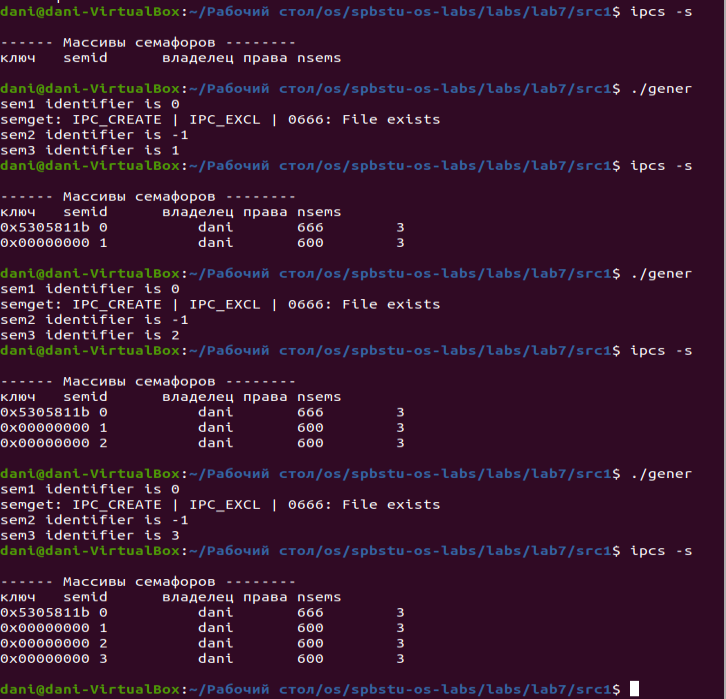


Рис. 7-1 Работа программы gener\_sem.cpp.

Первый набор создается с флагом **IPC\_CREATE**, если набора с таким ключом нет — создается новый, а если такой набор существует — **semget()** вернет его идентификатор. Это мы видим из вывода программы — набор было создан лишь один раз (один идентификатор).При создании второго набора был дополнительно указан флаг **IPC\_EXCL**, что означает, что мы всегда хотим создавать новый набор. Так как мы уже создали набор (sem1) с таким же ключом, то мы получим ошибку.

Флаг **IPC\_PRIVATE** — Набор sem3 будет создаваться при каждом новом

запуске программы. Причем каждый раз с новым уникальным идентификатором.

**Пункт 2**

Удалите созданные на предыдущем шаге семафоры с помощью команды ipcrm с соответствующей опцией и значением id семафора или ключа.

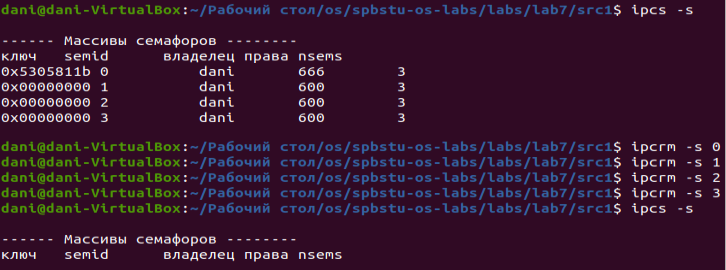
****

Рис. 7-2 Удаление семафоров.

**Пункт 3**

Скомпилируйте semdemo.cpp, демонстрирующую организацию разделения доступа к общему ресурсу между несколькими процессами с помощью технологии семафоров. Запустите сразу несколько процессов на разных терминалах и проанализируйте их взаимодействие и соблюдение очередности в попытках получения общего ресурса.

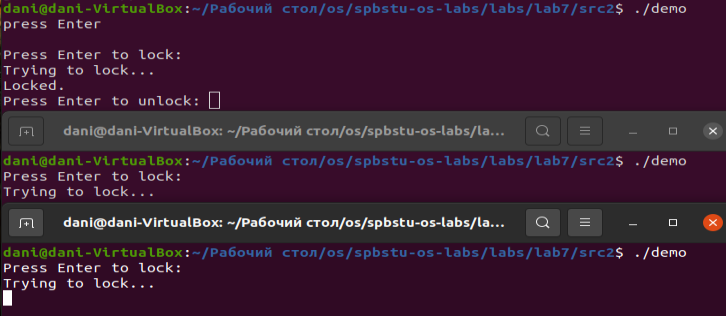


Рис. 7-3 Демонстрация работы программы semdemo.cpp.

Мы заблокировали наш семафор в первом процессе, второй и третий процесс ожидают разблокировки.

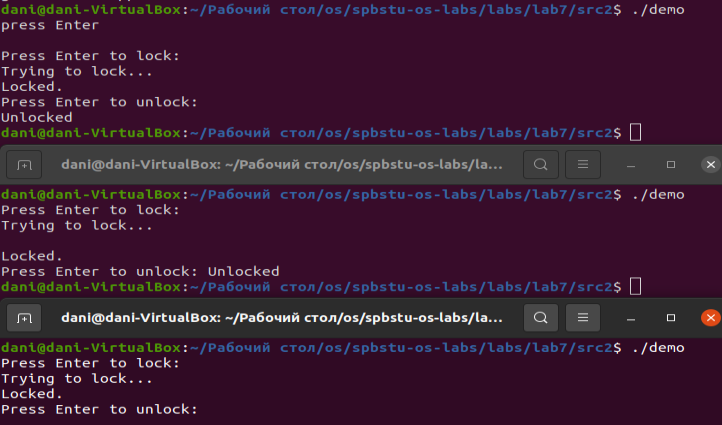


Рис. 7-4 Демонстрация работы программы semdemo.cpp.

Разблокируем семафор в первом процессе, второй процесс займет семафор. Третий продолжит ожидание. Таким образом каждый процесс будет по очереди занимать семафор.

**Пункт 4**

Скомпилируйте программу semrm.cpp и произведите с ее помощью удаление созданного на предыдущем шаге семафора. Поясните, почему данная программа удаляет только те семафоры, которые были созданы при выполнении программы semdemo.cpp.

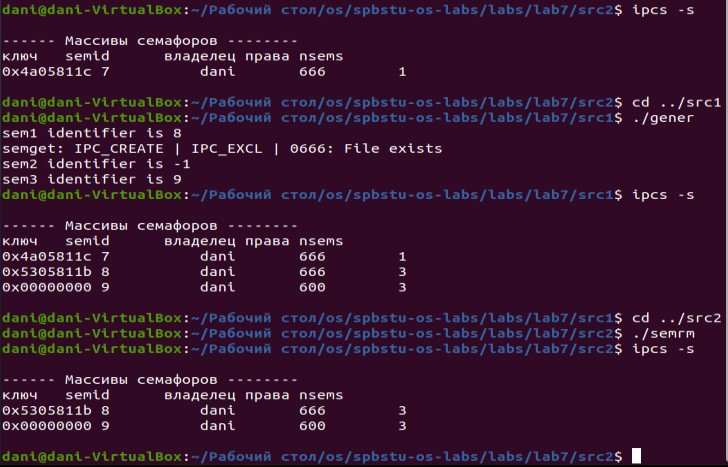


Рис. 7-5 Удаление семафора.

Как видно программа **semrm** удаляет только семафор оставшийся от

semdemo, потому что они используют один и тот же ключ.

**Пункт 5**

Попробуйте удалить семафор с помощью запуска semrm.cpp во время исполнения semdemo.cpp и проанализируйте ситуацию.

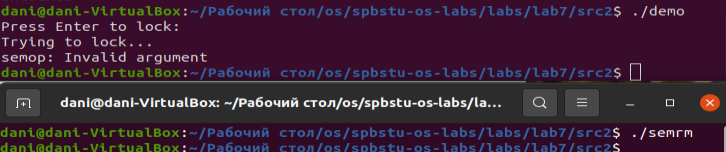


Рис. 7-5 Удаление семафора во время работы.

Получили ошибку, так как семафор был удалён посреди выполнения программы.

**Пункт 6**

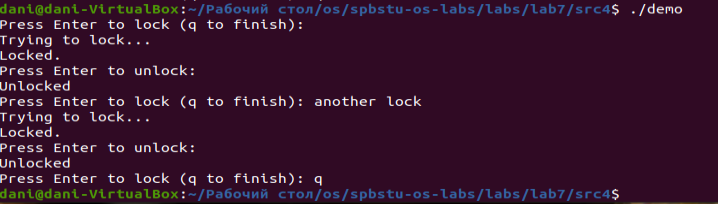
Попытайтесь улучшить программу semdemo.cpp , например, предоставив процессу возможность после освобождения ресурса становиться снова в очередь на повторное его занятие (а не завершаться), организовав при этом завершение процесса по вводу какого-либо символа.



Рис. 7-6 Модифицированная часть программы semdemo.cpp.

Программа модифицированна таким образом, чтобы процесс мог циклически занимать семафор, а для завершения процесса нужно прописать ‘q’ на этапе занятия семафора.

Теперь программа работает следующим образом:

 Рис. 7-7 Демонстрация работы модифицированной semdemo.cpp.

**Пункт 7**

Составьте программу, позволяющую мониторить количество процессов (типа semdemo), находящихся в состоянии ожидания освобождения ресурса (Trying to lock...) в каждый момент времени. Программа строится на основе вызова semctl() с соответствующими параметрами и запускается на отдельном терминале.

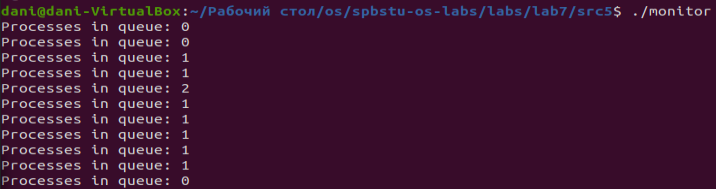
|  |
| --- |
| Листинг 7-1 monitorWaiting.cpp |
| #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <sys/types.h>  #include <sys/ipc.h>  #include <sys/sem.h>  #define MAX\_RETRIES 10  int main(void)  {  char u\_char = 'J';  key\_t key = ftok(".", u\_char);  int semid;  if ((semid = semget(key, 1, 0)) < 0) {  perror("semget");  exit(1);  }  int waitingCount;  while(1) {  waitingCount = semctl(semid, 0, GETNCNT);  if (waitingCount < 0) {  perror("semctl");  exit(1);  }  printf("Processes in queue: %d\n", waitingCount);  sleep(10);  }  return 0;  } |

Для начала получим ключ, который будет точно таким же, как и в программе semdemo.cpp.

Затем, при помощи функции semget(), получим идентификатор множества семафоров, ассоциированного с ключом key.

После чего, в цикле мониторим число процессов, ожидающих разблокировки ресурса при помощи функции semctl() и значения GETNCNT.

Получаем следующее поведение:



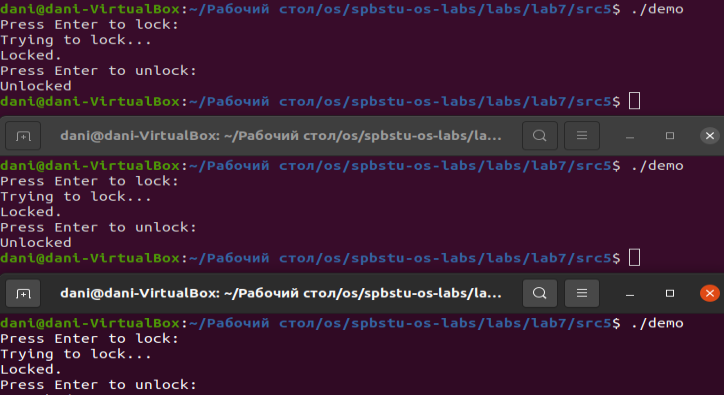


Рис. 7-8 Демонстрация работы программы мониторинга ожидания процессов.

Здесь у нас идет следующая цепочка:

Мы на процессе 1 блокируем семафор – в очереди 0;

На процессе 2 встаем в ожидание – в очереди 1;

На процессе 3 встаем в ожидание – в очереди 2;

Процесс 1 освобождает семафор, второй процесс занимает его – в очереди 1;

Процесс 2 освобождает семафор, третий процесс занимает его – в очереди 0;

**Вывод**

В данной лабораторной работе мы познакомились с созданием семафоров и осуществлением с их помощью синхронизации между процессами в операционной системе Linux.