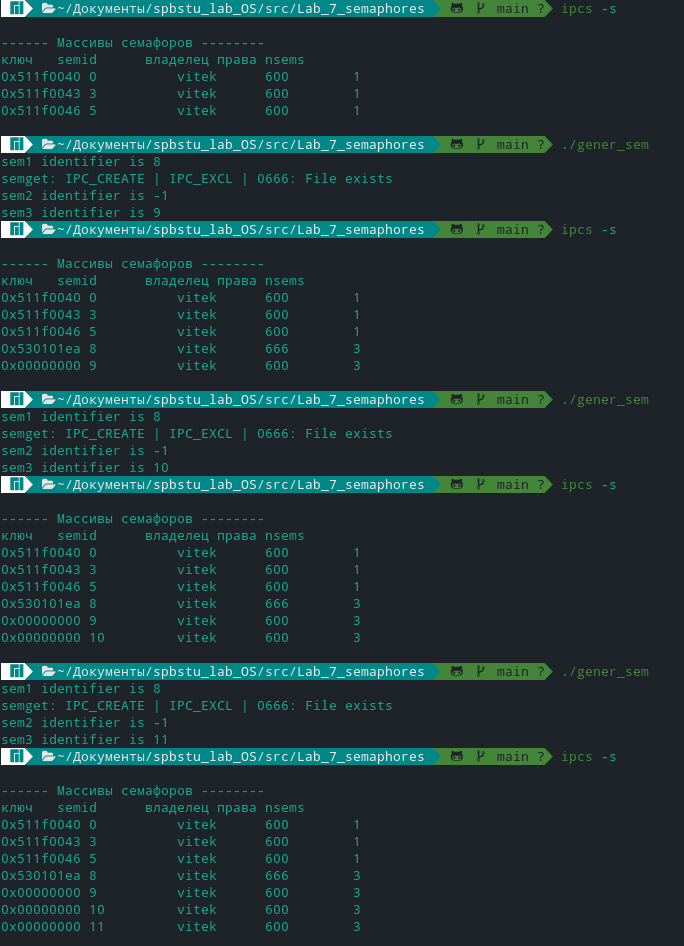
# **Лабораторная работа №7 «СЕМАФОРЫ И СИНХРОНИЗАЦИЯ»**

**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.

**Последовательность выполнения работы:**

1. Войдите в систему и скопируйте с разделяемого ресурса в свой HOME каталог набор исходных файлов для седьмого занятия.
2. 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.



Первый аргумент semget() - это ключ (в нашем случае возвращается ftok()-ом). Он сравнивается с ключами остальных множеств семафоров, присутствующих в системе. Вместе с этим решается вопрос о выборе между созданием и подключением к множеству семафоров в зависимости от аргумента msgflg.

ftok - преобразовывает имя файла и идентификатор проекта в ключ для системных вызовов. Системный вызов semget возвращает идентификатор множества семафоров, ассоциированный с ключом key.

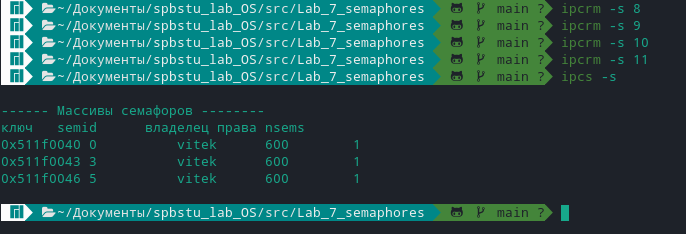
При запуске данной программы несколько раз видим, что:

• набор sem1 будет создан единожды, а затем каждая новая попытка будет всего лишь открывать доступ к существующему ресурсу;

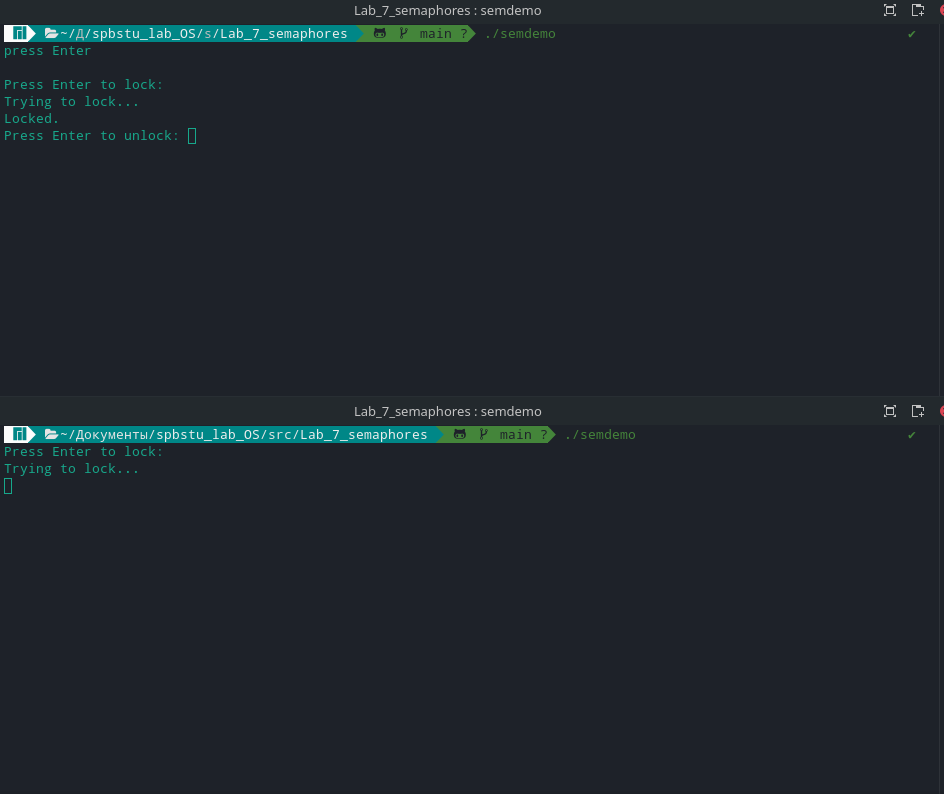
• попытки создания набора sem2 на том же ключе всегда будут приводить к ошибке из-за наличия флагов IPC\_CREATE | IPC\_EXCL, не допускающих открытия ресурса вместо его создания;

• набор sem3 будет создаваться при каждом новом запуске программы. Причем каждый раз с новым уникальным идентификатором.

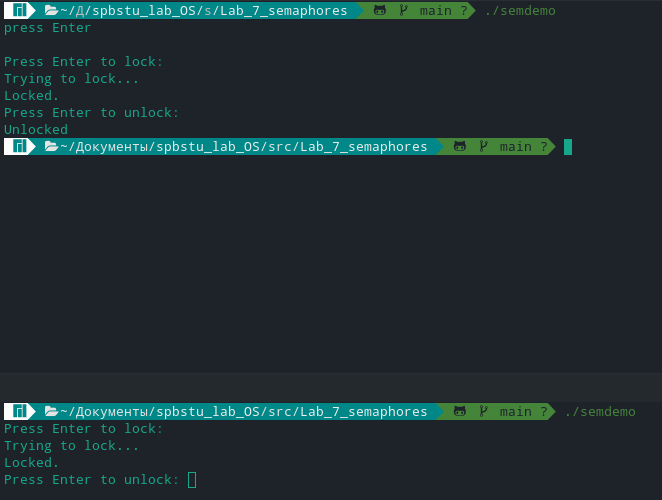
1. Удалите созданные на предыдущем шаге семафоры с помощью команды ipcrm с соответствующей опцией и значением id семафора или ключа.



1. 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.

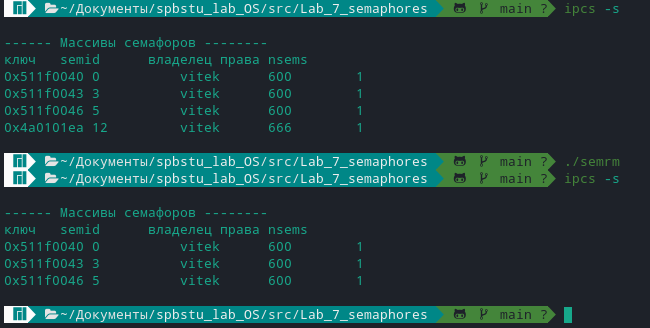


В верхнем терминале мы заблокировали ресурс, а в нижнем пытаемся заблокировать этот же ресурс, но вынуждены ждать, пока он освободится другим процессом (верхний терминал).

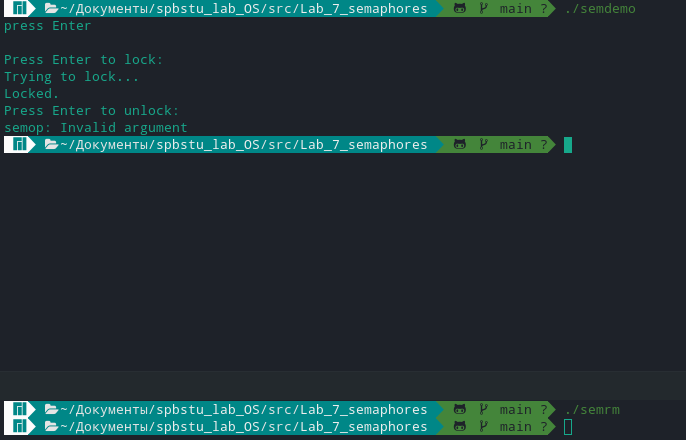


Как только мы освободили ресурс в верхнем терминале он тут же был занят ожидающим его нижним терминалом.

1. 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 .

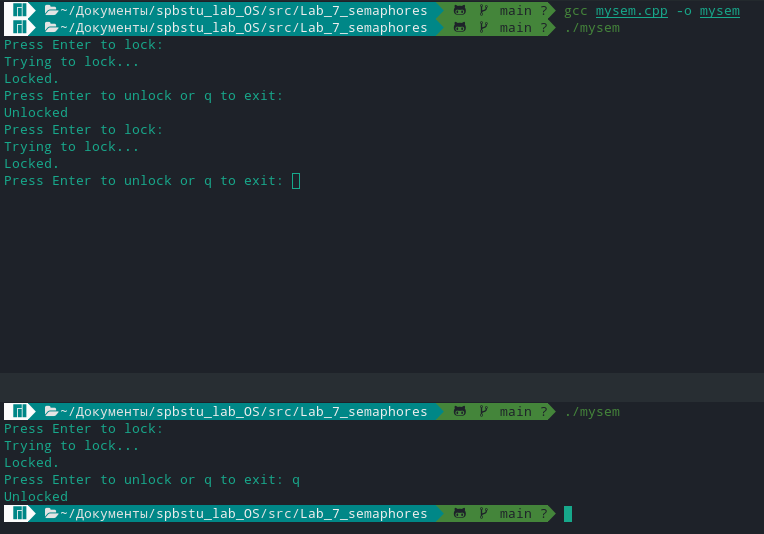


Программа может удалить только те семафоры, что были созданы при выполнении предыдущего пункта т. к. в semrn.cpp задается тот же ключ key = ftok(".", u\_char) (где u\_char = `J`; ключи семафоров уникальны)

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

Т.к. семафор с данным ключом посреди работы программы был удален, вызывается ошибка.

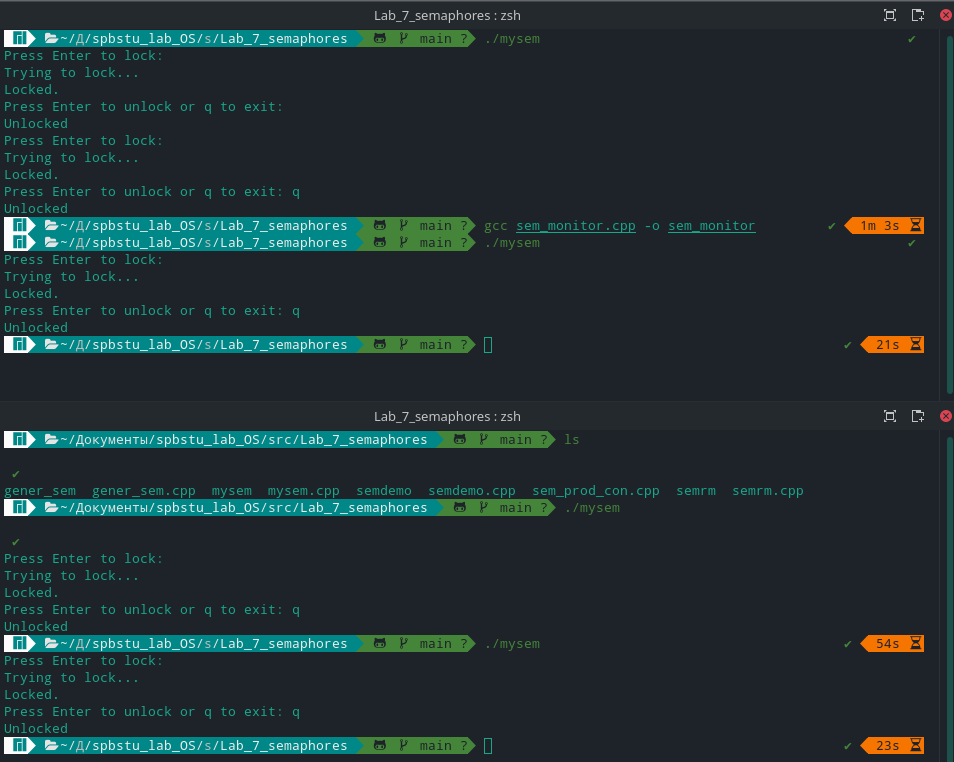
1. Попытайтесь улучшить программу semdemo.cpp , например, предоставив процессу возможность после освобождения ресурса становиться снова в очередь на повторное его занятие (а не завершаться), организовав при этом завершение процесса по вводу какого-либо символа.

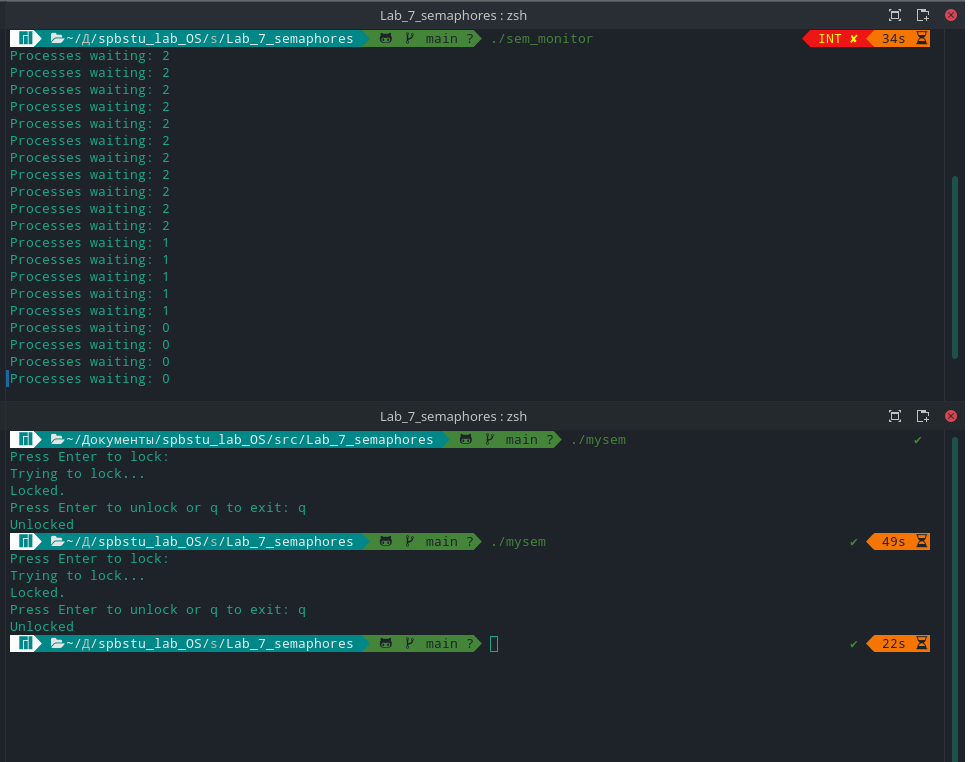


|  |
| --- |
| int main(void)  {  int isWorking = 1;  key\_t key;  int semid;  char u\_char = 'J';  struct sembuf sb;  while (isWorking) {  sb.sem\_num = 0;  sb.sem\_op = -1; /\* set to allocate resource \*/  sb.sem\_flg = SEM\_UNDO;  if ((key = ftok(".", u\_char)) == -1) {  perror("ftok");  exit(1);  }  /\* grab the semaphore set created by initsem: \*/  if ((semid = initsem(key, 1)) == -1) {  perror("initsem");  exit(1);  }  printf("Press Enter to lock: ");  getchar();  printf("Trying to lock...\n");  if (semop(semid, &sb, 1) == -1) {  perror("semop");  exit(1);  }  printf("Locked.\n");  printf("Press Enter to unlock or q to exit: ");  if (getchar() == 'q') isWorking = 0;  sb.sem\_op = 1; /\* free resource \*/  if (semop(semid, &sb, 1) == -1) {  perror("semop");  exit(1);  }  printf("Unlocked\n");  }  return 0;  } |

1. 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.





|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <errno.h>  #include <sys/types.h>  #include <sys/ipc.h>  #include <sys/sem.h>  #define MAX\_RETRIES 10  union semun {  int val;  struct semid\_ds \*buf;  ushort \*array;  };  int initsem(key\_t key, int nsems) /\* key from ftok() \*/  {  int i;  union semun arg;  struct semid\_ds buf;  struct sembuf sb;  int semid;  semid = semget(key, nsems, IPC\_CREAT | IPC\_EXCL | 0666);  if (semid >= 0) { /\* we got it first \*/  sb.sem\_op = 1; sb.sem\_flg = 0;  arg.val = 1;  printf("press Enter\n"); getchar();  for(sb.sem\_num = 0; sb.sem\_num < nsems; sb.sem\_num++) {  /\* do a semop() to "free" the semaphores. \*/  /\* this sets the sem\_otime field, as needed below. \*/  if (semop(semid, &sb, 1) == -1) {  int e = errno;  semctl(semid, 0, IPC\_RMID); /\* clean up \*/  errno = e;  return -1; /\* error, check errno \*/  }  }  } else if (errno == EEXIST) { /\* someone else got it first \*/  int ready = 0;  semid = semget(key, nsems, 0); /\* get the id \*/  if (semid < 0) return semid; /\* error, check errno \*/  /\* wait for other process to initialize the semaphore: \*/  arg.buf = &buf;  for(i = 0; i < MAX\_RETRIES && !ready; i++) {  semctl(semid, nsems-1, IPC\_STAT, arg);  if (arg.buf->sem\_otime != 0) {  ready = 1;  } else {  sleep(1);  }  }  if (!ready) {  errno = ETIME;  return -1;  }  } else {  return semid; /\* error, check errno \*/  }  return semid;  }  int main(void)  {  key\_t key;  int semid;  char u\_char = 'J';  if ((key = ftok(".", u\_char)) == -1) {  perror("ftok");  exit(1);  }  /\* grab the semaphore set created by initsem: \*/  if ((semid = initsem(key, 1)) == -1) {  perror("initsem");  exit(1);  }  while (true){  int output = semctl(semid, 0, GETNCNT);  printf("Processes waiting: %d\n", output);  sleep(1);  }  return 0;  } |

Вывод

Для синхронизации доступа к разделяемым ресурсам удобно использовать семафоры. Мы научились методам их создания, мониторинга, удаления, а также работы с ними.