Потоки

- Создание потока
  int **pthread_create**(pthread_t *restrict thread, const pthread_attr_t *restrict attr,
  void *(*start_routine)(void*), void *restrict arg);
- Ожидание завершения потока
  int **pthread_join**(pthread_t thread, void **retval);
- Запрос на завершение потока
  int **pthread_cancel**(pthread_t thread);
- Завершение потока
  void **pthread_exit**(void *value_ptr);
- Получение идентификатора потока
  pthread_t **pthread_self**(void);
- Compile and link with -pthread.

Потоки

```
#include<pthread.h>
#include<stdio.h>

#define THREADS_COUNT 5
int a = 0;

void* func(void* arg) {
  int tmp;
  for(int i = 0; i < 1e6; ++i) {
    tmp = a;
    tmp++;
    a = tmp;
  }
  pthread_exit(0);
}
```

```
int main() {
  pthread_t tid[THREADS_COUNT];
  void* status[THREADS_COUNT];
  int i, i1;
  printf("Initial value, a = %d\n", a);
  for(i = 0; i < THREADS_COUNT; ++i) {
    pthread_create(&tid[i], NULL, func, NULL);
  }
  for(i1 = 0; i1 < THREADS_COUNT; ++i1) {
    pthread_join(tid[i1], &status[i1]);
  }
  printf("Final value, a = %d\n", a);
  return 0;
}
```

Мьютексы

- Мьютекс (mutex) — примитив синхронизации, обеспечивающий взаимное исключение исполнения критических участков кода.
- Инициализация мьютекса
  int **pthread_mutex_init**(pthread_mutex_t *restrict mutex,
  const pthread_mutexattr_t *restrict attr);
  pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
- Захват
  int **pthread_mutex_lock**(pthread_mutex_t *mutex);
  int **pthread_mutex_trylock**(pthread_mutex_t *mutex);
- Освобождение
  int **pthread_mutex_unlock**(pthread_mutex_t *mutex);
- Уничтожение
  int **pthread_mutex_destroy**(pthread_mutex_t *mutex);

Мьютексы

1

```
lock(m);
tmp = a;
tmp++;
a = tmp;
unlock(m);
```

2

```
lock(m);
tmp = a;
tmp++;
a = tmp;
unlock(m);
```

3

```
tmp = a;
tmp++;
a = tmp;
```

Мьютексы

1	2
`lock(m1);`	`lock(m2);`
`lock(m2);`	`lock(m1);`

Семафоры

- Семафор – способ синхронизации работы процессов (потоков), в основе которого лежит счётчик, над которым можно производить две атомарные операции: увеличение (V) и уменьшение (P) значения на единицу. При этом операция уменьшения для нулевого значения счётчика является блокирующейся.
- По умолчанию при создании семафора значение счетчика равно нулю.
- Число в семафоре означает количество процессов, которые могут получить доступ к данным. При обращении процесса к данным число должно быть уменьшено на единицу и увеличено, когда работа с данными завершена.
- Число в семафоре можно использовать и для других целей. Например, оно может означать количество свободных ресурсов или количество клиентов, подключенных к серверу.

Семафоры

- Producer (производитель):

```
while(1) {
 produce_item;
 put_item;
}
```

- Consumer (потребитель):

```
while(1) {
 get_item;
 consume_item;
}
```

```
Semaphore mutex = 1;
Semaphore empty = N;
Semaphore full = 0;
```

- Producer:

```
while(1) {
 produce_item;
 P(empty);
 P(mutex);
 put_item;
 V(mutex);
 V(full);
}
```

- **mutex** – управление доступом к буферу
- **empty** – ожидание опустошения буфера
- **full** – ожидание наполнения буфера

- Consumer:

```
while(1) {
 P(full);
 P(mutex);
 get_item;
 V(mutex);
 V(empty);
 consume_item;
}
```

Семафоры (System V)

- Создание группы семафоров:
  `int semget(key_t key, int nsems, int semflg);`
  - **key** – IPC ключ
    `key_t ftok(const char *pathname, int proj_id);`
  - **nsems** – число семафоров, которое необходимо создать
  - **semflg** – права доступа (12 бит: первые три бита – режим создания, остальные девять – права на запись и чтение для пользователя, группы и остальных).

Семафоры (System V)

- Операции над семафором:
  int **semop**(int **semid**, struct sembuf ***sops**, unsigned **nsops**);
  int **semtimedop**(int **semid**, struct sembuf ***sops**, unsigned **nsops**,
  struct timespec ***timeout**);

- В **sops** описываются операции, которые необходимо сделать с каждым семафором. Все эти операции выполняются атомарно при вызове semop.

- struct **sembuf**:
  - `short sem_num; /* semaphore number: 0 = first */`
  - `short sem_op;  /* semaphore operation (>0, <0, ==0) */`
  - `short sem_flg;  /* operation flags (IPC_NOWAIT, SEM_UNDO) */`

Семафоры (System V)

- struct **sembuf**:
  - `short sem_num; /* semaphore number: 0 = first */`
  - `short sem_op;  /* semaphore operation (>0, <0, ==0) */`
  - `short sem_flg;  /* operation flags */`

- **sem_op < 0**

Если модуль значения в семафоре больше или равен модулю sem_op, то sem_op добавляется к значению в семафоре (т.е. значение в семафоре уменьшается). Иначе процесс переходит в спящий режим, пока не будет достаточно ресурсов.

- **sem_op = 0**

Процесс спит, пока значение в семафоре не достигнет нуля.

- **sem_op > 0**

Значение sem_op добавляется к значению в семафоре.

Семафоры (System V)

- struct **sembuf**:
  - `short sem_num; /* semaphore number: 0 = first */`
  - `short sem_op;  /* semaphore operation (>0, <0, ==0) */`
  - `short sem_flg;  /* operation flags */`

0 – enable, 1 – disable:

```
struct sembuf lock[2] = {{0, 0, 0},
                         {0, 1, 0}};
struct sembuf unlock = {0, -1, 0};
semop(semid, &lock, 2);
```

0 – disable, 1 – enable:

```
struct sembuf lock = {0, -1, 0};
struct sembuf unlock[2] = {{0, 0, 0},
                           {0, 1, 0}};
semop(semid, &lock, 1);
```

Семафоры (System V)

- Управление семафором:
  `int` **`semctl`**`(int semid, int semnum, int cmd, ...);`
  выполняет действие **cmd** на наборе семафоров **semid** или (если требуется командой) на одном семафоре с номером **semnum**.
- 3 или 4 параметра.
- **`semctl`**`(semid, semnum, cmd, arg);` // arg имеет тип union semun
- cmd: удаление объекта, просмотр и изменение значений

Семафоры (System V)

- **`semctl`**`(semid, semnum, cmd, `**`arg`**`);` // arg имеет тип union semun

```
union semun {
    int val; /* value for SETVAL */
    struct semid_ds *buf; /* buffer for IPC_STAT, IPC_SET */
    unsigned short *array; /* array for GETALL, SETALL */
                           /* Linux specific part: */
    struct seminfo *__buf; /* buffer for IPC_INFO */
};
```

Семафоры (System V)

Значения cmd:

- IPC_STAT - скопировать информацию из структуры данных набора семафоров в структуру, указанную в arg.buf.
- IPC_SET - внести значения некоторых членов структуры semid_ds, на которую указывает arg.buf , в структуру данных набора семафоров и обновить sem_ctime. Изменяются значения полей структуры semid_ds.
- IPC_RMID - немедленно удалить из системы набор семафоров и структуры его данных, запускающие все процессы, находящиеся в режиме ожидания.
- GETALL - возвращает значение semval всем семафорам в массиве arg.array.
- GETNCNT - возвращает значение semncnt семафору semnum-th (например, число процессов, ожидающих увеличения значения semval семафора semnum-th).
- GETPID - возвращает значение sempid семафору semnum-th (например, идентификатор процесса, который последним делал вызов semop семафору semnum-th).
- GETVAL - возвращает значение semval семафору semnum-th.
- GETZCNT - возвращает значение semzcnt семафору semnum-th (например, количество процессов, ожидающих, чтобы значение semval семафора semnum-th стало равным нулю).
- SETALL - установить значение semval всех семафоров равным значениям элементов массива, на который указывает arg.array, изменяя также sem_ctime, являющееся членом структуры semid_ds ; а эта структура ассоциируется с набором семафоров.
- SETVAL - установить значение semval на указанное в arg.val для всех семафоров semnum-th, изменяя также sem_ctime в структуре semid_ds, соотносимой с набором семафоров.

Семафоры (System V)

- Создать только один семафор

```
semid = semget(key, 1, 0666 | IPC_CREAT);
```

- В семафоре 0 установить значение 1

```
arg.val = 1;
semctl(semid, 0, SETVAL, arg);
```

- Удалить семафор

```
semctl(semid, 0, IPC_RMID);
```

Семафоры (System V)

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

union semun {
 int val;                  /* значение для SETVAL */
 struct semid_ds *buf;     /* буферы для  IPC_STAT, IPC_SET */
 unsigned short *array;    /* массивы для GETALL, SETALL */
                           /* часть, особенная для Linux: */
 struct seminfo *__buf;    /* буфер для IPC_INFO */
};
```

Семафоры (System V)

```
int main(int argc, char *argv[])
{
  /* IPC */
  pid_t pid;
  key_t key;
  int semid;
  union semun arg;
  struct sembuf lock_res = {0, -1, 0};
  struct sembuf rel_res = {0, 1, 0};
  struct sembuf push[2] = {{1, -1, IPC_NOWAIT}, {2, 1, IPC_NOWAIT}};
  struct sembuf pop[2] = {{1, 1, IPC_NOWAIT}, {2, -1, IPC_NOWAIT}};

  /* Остальное */
  int i;
  if(argc < 2) {
    printf("Usage: bufdemo [dimensione]\n");
    exit(0);
  }
```

Семафоры (System V)

```
/* Семафоры */
  key = ftok("/etc/fstab", getpid());

/* Создать набор из трёх семафоров */
  semid = semget(key, 3, 0666 | IPC_CREAT);

/*Установить в семафоре № 0 (Контроллер ресурсов) значение "1"*/
  arg.val = 1;
  semctl(semid, 0, SETVAL, arg);

/* Установить в семафоре номер 1 (Контроллер свободного места)
     значение длины буфера */
  arg.val = atol(argv[1]);
  semctl(semid, 1, SETVAL, arg);

/* Установить в семафоре № 2 (Контроллер элементов в буфере) значение "0" */
  arg.val = 0;
  semctl(semid, 2, SETVAL, arg);
```

```
/* Fork */
  for (i = 0; i < 5; i++){
    pid = fork();
    if (!pid){
      for (i = 0; i < 20; i++){
         sleep(rand()%6);

         /* Попытаться заблокировать ресурс (семафор номер 0) */
         if (semop(semid, &lock_res, 1) == -1){ perror("semop:lock_res"); }

         /* Уменьшить свободное место (семафор номер 1)
            Добавить элемент (семафор номер 2) */
         if (semop(semid, push, 2) != -1){
           printf("---> Process:%d\n", getpid());
         }
         else{
           printf("---> Process:%d  BUFFER FULL\n", getpid());
         }

         /* Разблокировать ресурс */
         semop(semid, &rel_res, 1);
      }
      exit(0);
    }
  }
```

```
for (i = 0; i < 10; i++){
    sleep(rand()%3);

    /* Попытаться заблокировать ресурс (семафор номер 0)*/
    if (semop(semid, &lock_res, 1) == -1){ perror("semop:lock_res"); }

    /* Увеличить свободное место (семафор номер 1)
       Взять элемент (семафор номер 2) */
    if (semop(semid, pop, 2) != -1){
      printf("<--- Process:%d\n", getpid());
    }
    else printf("<--- Process:%d  BUFFER EMPTY\n", getpid());

    /* Разблокировать ресурс */
    semop(semid, &rel_res, 1);
  }

  /* Удалить семафоры */
  semctl(semid, 0, IPC_RMID);
  return 0;
}
```

Задания

- **4.1** (2 балла). Скорректировать программу 2.10 так, чтобы доступ к каналу регулировался семафором.

- **4.2** (2 балла). Скорректировать программу 3.6 так, чтобы доступ к файлу регулировался семафором.

- **4.3** (2 балла). Скорректировать программу 4.2 так, чтобы несколько процессов (ограниченное количество) могли читать из файла. Запись в файл возможна, когда он никем не читается.

Семафоры (POSIX)

- Создание семафора (один семафор):
  sem_t ***sem_open**(const char *name, int oflag);
  sem_t ***sem_open**(const char *name, int oflag, mode_t mode, unsigned int value);
- Увеличить значение семафора на 1:
  int **sem_post**(sem_t *sem);
- Уменьшить значение семафора на 1:
  int **sem_wait**(sem_t *sem);
  int **sem_trywait**(sem_t *sem);
- Получить текущее значение семафора:
  int **sem_getvalue**(sem_t *restrict sem, int *restrict sval);
- Закрыть семафор: int **sem_close**(sem_t *sem);
- Удалить семафор: int **sem_unlink**(const char *name);

Link with **–lrt** or **-pthread**.

Задания

- **4.4 – 4.6**: решить задачи 4.1 – 4.3 с использованием семафоров POSIX.

Разделяемая память

- Разделяемая память позволяет осуществлять обмен информацией через общий для процессов сегмент памяти без использования системных вызовов ядра.
- Использование разделяемой памяти:
  - Процесс 1 получает доступ к разделяемой памяти, используя семафор.
  - Процесс 1 производит запись данных в разделяемую память.
  - После завершения записи данных Процесс 1 освобождает доступ к разделяемой памяти с помощью семафора.
  - Процесс 2 получает доступ к разделяемой памяти, запирая доступ к этой памяти для других процессов с помощью семафора.
  - Процесс 2 производит чтение данных из разделяемой памяти, а затем освобождает доступ к памяти с помощью семафора.

Разделяемая память (System V)

- Создание сегмента разделяемой памяти:
  int **shmget**(key_t key, int size, int shmflg);
  - key – ключ IPC
    key_t **ftok**(const char *pathname, int proj_id);
  - size – размер сегмента (округляется до размера, кратного PAGE_SIZE)
  - shmflg – права доступа
- Подключение сегмента к адресному пространству процесса:
  void ***shmat**(int shmid, const void *shmaddr, int shmflg);
  - shmaddr - адрес присоединяемого сегмента (рекомендуется указывать NULL)
  - shmflg: SHM_RND, SHM_RDONLY, SHM_REMAP (для Linux)

Разделяемая память (System V)

- Отключение сегмента от адресного пространства процесса:
  int **shmdt**(const void *shmaddr);
- Управление сегментом:
  int **shmctl**(int shmid, int cmd, struct shmid_ds *buf);

Команды для работы с ресурсами System V IPC:

- Получение информации о средствах, к которым пользователь имеет право доступа на чтение:
  ipcs [-asmq] [-tclup]
  ipcs [-smq] -i id
  ipcs -h;
- Удаление ресурса:
  ipcrm [shm | msg | sem] id

Задания

- **4.7** (3 балла). Родительский процесс генерирует наборы из случайного количества случайных чисел и помещает в разделяемую память. Дочерний процесс находит максимальное и минимальное число и также помещает их в разделяемую память, после чего родительский процесс выводит найденные значения на экран. Процесс повторяется до получения сигнала SIGINT, после чего выводится количество обработанных наборов данных.

- **4.8** (3 балла). Скорректировать решение задачи 4.7 так, чтобы порождались дополнительные дочерние процессы, находящие минимум, сумму и среднее значение элементов.

Разделяемая память (POSIX)

- Создание или подключение объекта разделяемой памяти:
  int **shm_open**(const char *name, int oflag, mode_t mode);
- Задать или изменить размер разделяемой памяти:
  int **truncate**(const char *path, off_t length);
- Подключить/отключить сегмент разделяемой памяти к адресному пространству процесса:
  void * **mmap**(void *start, size_t length, int prot , int flags, int fd, off_t offset);
  int **munmap**(void *start, size_t length);
- Удаление объекта разделяемой памяти (сегмент существует, пока не отключен от всех процессов):
  int **shm_unlink**(const char *name);

Задания

- **4.9 – 4.10**: решить задачи 4.7 – 4.8 с использованием разделяемой памяти POSIX.