**Практическая работа**

**«Семафоры»**

**Цели и задачи**

Познакомиться с общими принципами работы семафоров. Научиться использовать семафоры для синхронизации процессов и потоков, и для защиты критических секций.

**Общие сведения**

**Семафоры**

Для синхронизации работы процессов и для синхронизации доступа нескольких процессов к общим ресурсам используются семафоры. Общими ресурсами процессов являются файлы, сегменты разделяемой памяти. Возможность одновременного изменения несколькими процессами общих данных называют критической секцией, так как такая совместная работа процессов может привести к возникновению ошибок. Например, если несколько процессов осуществляют запись данных в один и тот же файл, эти данные могут оказаться перемешанными. Наиболее простой механизм защиты критической секции состоит в расстановке «замков», пропускающих только один процесс для выполнения критической секции, и останавливающий все остальные процессы, пытающиеся выполнить критическую секцию, до тех пор, пока эту критическую секцию не выполнит пропущенный процесс. Семафоры позволяют выполнять такую операцию, как и многие другие.

Под семафором может пониматься как единичный семафор, так и несколько семафоров, объединенных в группу. Общий механизм действия семафора таков: семафор обладает внутренним значением – числом, из множества целых чисел с нижней границей (например – с нулем), процессы могут изменять значение семафора – увеличивать или уменьшать. Если процесс изменяет значение семафора и выходит за граничное значение, такой процесс приостанавливается, пока какой-нибудь другой процесс не изменит значение семафора так, чтобы заблокированный процесс смог выполнить изменение значения семафора.

Использование общих данных несколькими процессами может привезти к ошибкам и конфликтам. Но при этом семафоры и сами являются общими данными. Такое положение не является противоречивым, в силу того, что:

1.Значение семафора расположено не в адресном пространстве некоторого процесса, а в адресном пространстве ядра.

2.Операция проверки и изменения значения семафора, вызываемая процессом является атомарной, т.е. непрерываемой другими процессами. Эта операция выполняется в режиме ядра.

Общими данными процессов также являются каналы и сообщения. Но операции с каналами и сообщениями защищаются системой, и, как правило, программист не должен использовать семафор для защиты записи сообщения в очередь сообщений, либо записи данных в канал. Однако это не всегда правильно. Например, система обеспечивает атомарную запись в канал, только для данных не больше определенного объема.

Семафоры используются и для синхронизации потоков. В многопоточном приложении критической секцией является возможность изменения глобальной переменной несколькими потоками.

**Семафоры для синхронизации процессов**

Семафор обладает внутренним значением - целым числом, принадлежащим типу unsigned short. Семафоры могут быть объединены в единую группу.

Рассмотрим функции для работы с семафорами (заголовочные файлы – **sys/ipc.h** и **sys/sem.h**).

**int semget ( key\_t key, int nsems, int semflag)**

Функция создает группу семафоров с ключом key (или дает доступ к уже существующей группе с заданным ключом), состоящую из nsems семафоров. Параметр key может быть равен IPC\_PRIVATE, в этом случае, система сама определяет незанятый ключ для группы семафоров. Параметр semflag может быть равен IPC\_CREAT – создать группу семафоров, или IPC\_EXCL – получить доступ к существующей группе. Если же shmflag равен IPC\_CREAT | IPC\_EXCL, функция создаст новую группу семафоров с ключом key, только когда не существует другой группы семафоров с тем же ключом. Параметр shmflag также может включать флаги доступа. Функция возвращает дескриптор группы семафоров в случае успеха, или -1 в случае неудачи. Все семафоры в созданной группе имеют внутреннее значение ноль.

**int semop(int semid, sembuf \*semop, size\_t nops)**

Функция выполняет над группой семафоров с дескриптором semid набор операций semop, nops – количество операций, выполняемых из набора semop.

Для задания операции над группой семафоров используется структура sembuf.

Первый параметр структуры sembuf определяет порядковый номер семафора в группе. Семафоры в группе индексируются с нуля.

Второй параметр структуры sembuf представляет собой целое число = S, и определяет действие, которое необходимо произвести над семафором, с индексом, записанным в первом параметре.

Если S>0 к внутреннему значению семафора добавляется число S. Эта операция не блокирует процесс.

Если S=0 процесс приостанавливается, пока внутреннее значение семафора не станет равно нулю.

Если S<0 процесс должен отнять от внутреннего значения семафора модуль S. Если значение семафора - |S| ≥ 0 производится вычитание и процесс продолжает свою работу. Если значение семафора - |S| < 0 процесс останавливается до тех пор, пока другой процесс не увеличит значение семафора на достаточную величину, чтобы операция вычитания выдала неотрицательный результат. Тогда производиться операция вычитания и процесс продолжает свою работу. Например, если значение семафора равно трем, а процесс пытает выполнить над ним операцию -4, этот процесс будет заблокированным, пока значение семафора не увеличиться хотя бы на единицу.

Третий параметр структуры sembuf может быть равен 0 – тогда операция S≤0 будет предполагать блокировку процесса, т.е. выполняться так, как описано выше. Также sembuf может быть равен IPC\_NOWAIT, в этом случае, работа процесса не будет останавливаться. Если процесс будет пытаться выполнить вычитание от значения семафора дающее отрицательный результат, эта операция просто игнорируется и процесс продолжает выполнение.

**Пример использования семафора для синхронизации процессов**

Процесс-родитель создает четыре процесса-потомка и ожидает их завершения, используя для этого семафор.

#include <stdio.h>

#include <unistd.h>

#include <sys/sem.h>

#include <sys/ipc.h>

int main()

{int semid ; *//для хранения дескриптора группы семафоров*

*//создать группу семафоров, состоящую из одного семафора*

semid = semget (IPC\_PRIVATE, 1, IPC\_CREAT|0666) ;

if ( semid < 0 ) *//если не удалось создать группу семафоров, завершить выполнение*

{ fprintf(stdout, “\nОшибка”) ; return 0 ; }

sembuf Plus1 = {0, 1, 0} ; *//операция прибавляет единицу к семафору с индексом 0*

sembuf Minus4 = {0, -4, 0} ; //*операция вычитает 4 от семафора с индексом 0*

*//создать четыре процесса-потомка*

for (int i=0 ; i<4 ; i++)

{if ( fork() == 0 ) *//истинно для дочернего процесса*

{ //здесь должен быть код выполняемый процессом-потомком

//добавить к семафору единицу, по окончанию работы

semop( semid, &Plus1, 1) ; return 1 ;

}

}

semop( semid, &Minus4, 1) ; return 1 ;

}

В описанном примере, созданный семафор при создании обладает нулевым значением. Каждый из четырех порожденный процессов, после выполнения своих вычислений, увеличивает значение семафора на единицу. Родительский процесс пытается уменьшить значение семафора на четыре. Таким образом, процесс-родитель останется заблокированным, до тех пор, пока не отработают все его потомки.

Последний параметр в функции semop определяет количество операций, берущихся для выполнения из второго параметра функции. Т.е. следующий вызов

sembuf Minus4 = {0, -4, 0} ;

semop( semid, &Minus4, 1) ;

Нельзя заменить таким вызовом.

sembuf Minus1 = {0, -1, 0} ;

semop( semid, &Minus1, 4) ;

Корректной заменой может являться.

sembuf Minus1[4] = {0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0} ;

semop( semid, Minus1, 4) ;

**Пример использования семафора для защиты критической секции**

Воспользуемся семафором для синхронизации доступа нескольких процессов к общему ресурсу, т.е. для защиты критической секции. Общим ресурсом будет являться разделяемая память.

Процесс – родитель создает сегмент разделяемой памяти и порождает три дочерних процесса, процесс-родитель и его потомки вычисляют сумму элементов определенной части массива и записывает вычисленную сумму в разделяемую память. Родительский процесс дожидается окончания работы потомков, и выводит окончательный результат – сумму всех элементов массива.

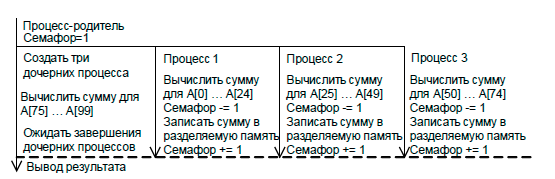


Рис.1

Так как четыре процесса могут изменять данные разделяемой памяти, необходимо сделать это изменение атомарным для каждого процесса. Для этого создадим семафор, который будет принимать два значения – ноль и единицу.

#include <stdio.h>

#include <unistd.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <sys/wait.h>

int shmid ; *//для хранения дескриптора разделяемой памяти*

int semid ; *//для хранения дескриптора группы семафоров*

sembuf Plus1 = {0, 1, 0} ; *//операция прибавляет единицу к семафору с индексом 0*

sembuf Minus1 = {0, -1, 0} ; //*операция вычитает единицу от семафора с индексом 0*

int A[100] ; *//массив, сумма элементов которого вычисляется процессами*

struct mymem *//структура, под которую будет выделена разделяемая память*

{ int sum ; *//для записи суммы*

} \*mem\_sum ;

void summa (int p) *//для вычисления суммы части элементов массива*

{ int i, sum = 0 ; *//для суммирования элементов*

int begin =25\*p ; *//индекс массива, с которого начинается суммирование*

int end = begin+25 ; *//индекс массива, на котором завершается суммирование*

for(i=begin ; i<end ; i++) sum+=A[i] ; *//вычисление суммы части элементов массива*

semop( semid, &Minus1, 1) ; *//отнять единицу от семафора*

mem\_sum->sum+=sum; *//добавление вычисленного результата в общую переменную*

semop( semid, &Plus1, 1) ; *//добавить единицу к семафору*

}

int main()

{//запрос на создание разделяемой памяти объемом 2 байта

shmid = shmget(IPC\_PRIVATE, 2, IPC\_CREAT|0666) ;

*//если запрос оказался неудачным, завершить выполнение*

if (shmid < 0 ) { fprintf(stdout,"\nОшибка") ; return 0 ; }

*//создать группу семафоров, состоящую из одного семафора*

semid = semget (IPC\_PRIVATE, 1, IPC\_CREAT|0666) ;

*//если не удалось создать группу семафоров, завершить выполнение*

if ( semid < 0 ) { fprintf(stdout, “\nОшибка”) ; return 0 ; }

semop( semid, &Plus1, 1) ; *//теперь семафор равен единице*

*//теперь mem\_sum указывает на выделенную разделяемую память*

mem\_sum = (mymem \*)shmat(shmid,NULL,0) ;

mem\_sum->sum = 0 ;

//тут должна быть инициализация элементов массива А

*//создать три процесса-потомка*

for (int i=0 ; i<3 ; i++)

{if ( fork() == 0 ) *//истинно для дочернего процесса*

{ summa(i) ; return 1 ; }

}

summa(3) ; *//родительский процесс вычисляет последнюю четверть массива*

for (int i=0 ; i<3 ; i++)

wait(NULL) ; *//дождаться завершения процессов-потомков*

*//вывести на экран сумму всех элементов массива*

fprintf(stdout,"\nРезультат = %d",mem\_sum->sum) ;

return 1;

}

Семафор получает при создании значение равное нулю, которое сразу же устанавливается в единицу. Первый процесс, вызвавший функцию semop( semid, &Minus1, 1), уменьшает значение семафора до нуля, переходит к записи в разделяемую память, и по завершении операции записи, устанавливает значение семафора в единицу, вызвав semop( semid, &Plus1, 1). Если управление перейдет к другому процессу, во время записи в разделяемую память первым процессом, вызов функции «отнять от семафора единицу» остановит работу другого процесса, до того момента, когда значение семафора не станет положительным. Что может произойти только тогда, когда первый процесс завершит запись в общую память, и выполнит операцию – добавить к семафору единицу.

Если процессы обладают несколькими общими ресурсами, то необходимо для каждого общего ресурса создавать свой семафор.

**Семафоры для синхронизации потоков**

Для многопоточного приложения, также как и для многопроцессного, критической секцией является изменение несколькими потоками общего ресурса, например файла или глобальной переменной. Для синхронизации работы потоков и для синхронизации доступа нескольких потоков к общим ресурсам используются семафоры. Но при этом семафоры, используемые для синхронизации потоков, принадлежат к другому стандарту, чем семафоры, используемые для синхронизации процессов. Рассмотрим семафоры, блокирующие потоки.

Каждый семафор содержит неотрицательное целое значение. Любой поток может изменять значение семафора. Когда поток пытается уменьшить значение семафора, происходит следующее: если значение больше нуля, то оно уменьшается, если же значение равно нулю, поток приостанавливается до того момента, когда значение семафора станет положительным, тогда значение уменьшается и поток продолжает работу. Увеличение значения семафора, возможно всегда, эта операция не предполагает блокировки. Однако значение семафора не должно выходить за границы типа unsigned int.

Рассмотрим функции для работы с семафорами (заголовочный файл–**semaphore.h**).

**int sem\_init (sem\_t \*sem, int pshared, unsigned int value)**

Функция инициализирует семафор sem и присваивает ему значение value. Если параметр pshared больше нуля, семафор может быть доступен нескольким процессам, если pshared равен нулю, семафор создается для использования внутри одного процесса. Функция возвращает ноль в случае успеха.

**int sem\_destroy (sem\_t \*sem)**

Функция разрушает семафор sem и возвращает ноль в случае успеха.

**int sem\_getvalue (sem\_t \*sem, int \*sval)**

Функция записывает значение семафора sem в \*sval. Необходимо понимать, что если несколько потоков используют семафор, полученное значение семафора может быть устаревшим.

**int sem\_post (sem\_t \*sem)**

Функция увеличивает значение семафора sem на единицу.

**int sem\_wait (sem\_t \*sem)**

Если текущее значение семафора больше нуля, функция уменьшает значение семафора sem на единицу. Если текущее значение семафора равно нулю, выполнение потока, вызвавшего функцию sem\_wait, приостанавливается до тех пор, когда значение семафора станет положительным, тогда значение семафора уменьшается на единицу и поток продолжает работу.

**int sem\_trywait (sem\_t \*sem)**

Если текущее значение семафора больше нуля, функция уменьшает значение семафора sem на единицу и возвращает ноль. Если текущее значение семафора равно нулю, функция возвращает не нулевое значение. Функция sem\_trywait не останавливает работу потока.

**Обратите внимание!**

Для корректной работы описанных семафоров, необходимо при компиляции программы использовать команду:

**g++ <исходный файл> -o <исполняемый файл> -lpthread**.

**Пример использования семафора для синхронизации потоков**

Перепишем пример использования потоков из предыдущей работы, введя семафор для защиты глобальной переменной SUM и семафор, блокирующий процесс-родитель, пока не выполняться все его дочерние потоки. Изменения выделены жирным шрифтом.

#include <sched.h>

#include <stdio.h>

#include <unistd.h>

**#include <semaphore.h>**

#define NUMSTACK 5000 *//объем стека для отдельного потока*

**sem\_t sem ;** *//семафор для защиты критической секции*

**sem\_t sem4 ;** *//семафор для синхронизации родителя и потомков*

int A[100] ; *//массив, сумма элементов которого вычисляется потоками*

int SUM=0 ; *//для записи общей суммы*

char stack[4][ NUMSTACK] ; //*//для хранения стека четырех потоков*

int func(void \*param) *//стартовая функция потоков*

{int i, sum = 0 ; *//для суммирования элементов*

*//индекс массива, с которого начинается суммирование*

int p =\*(int \*)param ; p=р\*25 ;

for(i=p ; i<p+25 ; i++) sum+=A[i] ; *//вычисление суммы части элементов массива*

**sem\_wait(&sem) ;** *//отнять единицу от значения семафора sem*

SUM+=sum ; *//добавление вычисленного результата в общую переменную*

**sem\_post(&sem) ;** *//добавить единицу к значения семафора sem*

**sem\_post(&sem4) ;** *//добавить единицу к значения семафора sem4*

return 1 ;

}

int main()

{//тут должна быть инициализация элементов массива А

**sem\_init (&sem, 1, 1) ;** *//инициализируем семафор sem со значением 1*

**sem\_init (&sem4, 1, 0) ;** *//инициализируем семафор sem4 со значением 0*

int param[4] ; *//для хранения параметров потоков*

**for (int i=0 ; i<4 ; i++)** *//создание четырех потоков*

**{ param[i]=i ; char \*tostack=stack[i] ;**

**clone(func,(void\*)( tostack+ NUMSTACK -1),CLONE\_VM, (void \*)(param+i)) ;**

**}**

*//отнять четыре единицы от значения семафора sem4*

**for (int i=0 ; i<4 ; i++) sem\_wait(&sem4) ;**

fprintf (stdout,"\nРезультат = %d", SUM) ;

return 1 ;

}

Принцип работы семафоров, использованных в данном примере, полностью аналогичен семафорам из двух предыдущих примеров - пример использования семафора для синхронизации процессов и пример использования семафора для защиты критической секции.

**Порядок выполнения практической работы**

1.В задании а лабораторной работы 4 ввести защиту критических секций с помощью семафора. Если у вашего алгоритма отсутствуют критические секции, объяснить, почему их нет.

2.В задании б лабораторной работы 4 ввести защиту критических секций и обеспечить синхронизацию между процессом-родителем и дочерними потоками с помощью семафоров. Если у вашего алгоритма отсутствуют критические секции, объяснить, почему их нет.

3.Разработать алгоритм решения задания практической работы 5, с учетом разделения вычислений между несколькими процессами. Для обмена информацией между процессами использовать разделяемую память. Для защиты операций с разделяемой памятью и синхронизации процессов использовать семафоры. Реализовать алгоритм и протестировать его на нескольких примерах.

4.Разработать алгоритм решения задания лабораторной работы 5, с учетом разделения вычислений между несколькими потоками. Для синхронизации потоков и защиты критических секций использовать семафоры. Реализовать алгоритм и протестировать его на нескольких примерах.

5.Посмотреть в динамике работу семафоров для созданных приложений, используя команду ipcs -s.

**Варианты заданий**

1.Найти максимальный элемент из минимальных элементов в каждой строке матрицы А. Входные данные: целое положительное число n, целое положительное число к, массив чисел от А размерности nxк. Использовать n или n+1 процессов (потоков) для решения задачи.

2.Найти наибольший элемент на главной диагонали, и наименьший элемент на побочной диагонали, заменить элемент, стоящий на пересечении диагоналей на сумму двух найденных значений. Входные данные: целое положительное нечетное число n, массив чисел от А размерности nxn.

3.Определить все целые числа из интервала [А, В], имеющие наибольшее количество делителей. Найти среднее арифметическое найденных чисел. Входные данные: число А, число В, целое положительное число к > 1 и < (B-A)/2. Использовать к процессов (потоков) для решения задачи.

4.Найти все простые натуральные числа из интервала [А, В], двоичная запись которых является палиндромом – одинаково читается как слева направо, так и справа налево. Вычислить сумму найденных чисел. Входные данные: число А, число В.

5.Задана строка S, содержащая не менее двух слов, и символ с. Составит новую строку S1 из слов строки S, в которых есть символ с, и новую строку S2 из слов строки S, в которых нет символа с. Учитывать порядок вхождения слов в строку. Входные данные: строка S произвольной длины и символ с. Для решения задачи использовать столько процессов (потоков), сколько слов в строке.

6.Задана строка S, имеющая следующий вид «число о число о число … о число», где о может быть равно +, -, /, \*. Вычислить выражение записанное в строке. Входные данные: строка S.

7.Задана строка S, содержащая не менее двух предложений. Найти слово с максимальной длиной, встречающееся во всех предложениях или сообщить, что такого слова нет. Входные данные: строка S.

8.Задана строка S, содержащая не менее двух слов, и символ к. Найти слово с минимальной длиной, начинающееся с символа к, или сообщить, что такого слова нет. Входные данные: строка S, символ к. Для решения задачи использовать столько процессов (потоков), сколько слов в строке.

9.Определить является ли строка симметричной относительно указанного индекса r. Входные данные: строка S произвольной длины, целое число r > 0 и < длины строки. Для решения задачи использовать два процесса (потока).

10.Задана строка S, и множество пар символов (ai, bi) i = 1, 2 … n, получить новую строку, заменив в строке S каждое вхождение ai символа на bi. Входные данные: строка S произвольной длины, целое положительное число n, множество пар символов (ai, bi) i = 1, 2 … n. Для решения задачи использовать четыре процесса (потока), разделив между ними строку S.

11.Задана строка S, и множество пар символов (ai, bi) i = 1, 2 … n, получить новую строку, заменив в строке S каждое вхождение ai символа на bi. Входные данные: строка S произвольной длины, целое положительное число n, множество пар символов (ai, bi) i = 1, 2 … n. Для решения задачи использовать n процессов (потоков), работающих параллельно, причем каждый процесс (поток) находит и заменяет только свою пару символов.

12.Задана строка S1, содержащая не менее двух слов и строка S2, содержащая такое же количество слов, что и S1. Слово из строки S2 является синонимом соответствующего слова из строки S1, если оно записано с префиксом ‘!’. Заменить в строке S1 слова на их синонимы, удалив префиксы. Входные данные: строки S1 и S2, содержащие одинаковое количество слов.

13.Найти среднее арифметическое всех факториалов в интервале [А, В]. Входные данные: положительное число А, положительное число В, целое число к ≥ 2 и ≤ 6. Использовать к процессов (потоков) для решения задачи. Предусмотреть возможность автоматического уменьшения числа процессов (потоков), если это целесообразно.

14.Сформировать одномерный массив В, элементами которого являются наибольшие из четырех рядом стоящих элементов, образующих квадрат 2х2, в двоичном массиве А. Порядок занесения элементов в массив В не важен. Входные данные: целое положительное четное число n, массив чисел от А размерности nxn.

15.Задана строка S, содержащая не менее двух слов, и символ к. Составит новую строку из тех слов строки S, которые начинаются и заканчиваются символом к. Учитывать порядок расположения слов в строке. Входные данные: строка S.

16.Сформировать одномерный массив В, элементами которого являются средние арифметические положительных элементов столбцов двумерного массива А. Найти сумму минимального и максимального элемента массива В. Порядок занесения элементов в массив В должен соответствовать их индексации в массиве А. Входные данные: целое положительное число n, целое положительное число к, массив чисел от А размерности nxк. Использовать к или к+1 процессов (потоков) для решения задачи.

17.Сформировать массив В заменяя элементы массива А их наибольшими делителями. Найти среднее арифметическое элементов массива В, сумма индексов которых является нечетным числом. Входные данные: целое положительное число n, массив чисел от А размерности nxn, целое число к ≥ 2 и ≤ n/2. Использовать к процессов (потоков) для решения задачи.

18.Определить в каком числе - А или В больше вхождений цифры ‘0’, вхождений цифры ‘1’, вхождений цифры ‘2’ … вхождений цифры ‘9’. Определить число, сумма вхождений четных цифр которого больше. Входные данные: целое положительное число А, целое положительное число В. Для решения задачи использовать 10 процессов (потоков),каждый из которых должен определять количество вхождений своей цифры.

19.Задана строка S, содержащая не менее двух целых чисел, разделителем является запятая. Составить новую строку, записав в нее все нечетные числа в обратном порядке, пример: было 2961, стало 1692. Порядок занесения чисел в новую строку не важен. Входные данные: строка S, целое число к ≥ 2 и ≤ 8. Использовать к процессов (потоков) для решения задачи. Предусмотреть возможность автоматического уменьшения числа процессов (потоков), если это целесообразно.

20.Задан массив чисел, записанных в двоичной системе счисления, вычислить сумму всех чисел, не переводя их в десятичную систему счисления. Входные данные: целое положительное четное число n > 2, двоичный массив А размерности nxn, заполненный нулями и единицами, элементы строк составляют числа, целое число к ≥ 2 и ≤ n/2. Использовать к процессов (потоков) для решения задачи.