Выполнила: Белоусова Е., ИП-911

Цель: познакомиться с принципами синхронизации потоков с помощью объектов ядра.

Задание:

• протестировать программы Лекции 10.

• Изменить программу deadlock.c так, чтобы избежать взаимоблокировки.

Описание работы программы

Протестируем программы из лекции 10.

- Event представляет объект синхронизации потоков, сигнализирующим о завершении операции. По сути, это просто флаг, который может находиться либо в нейтральном состоянии, либо в сигнальном. Создать событие можно с помощью функции CreateEvent. В программе создается один поток и два события, которые будут сменять друг друга в цикле. Одно создано с начальным сигнальным состоянием, другое с занятым.

С точки зрения операционной системы объекты ядра, поддерживающие интерфейс синхронизируемых объектов, могут находиться в одном из двух состояний: свободном (signaled) и занятом (nonsignaled). Функции проверяют состояние ожидаемого объекта или ожидаемых объектов и продолжают выполнение, только если объекты свободны. В зависимости от типа ожидаемого объекта, система может предпринять специальные действия (например, как только поток дожидается освобождения объекта исключительного владения, он сразу должен захватить его).

Обычно событие - некоторый объект, который может находиться в одном из двух состояний: занятом или свободном. Переключение состояний осуществляется явным вызовом соответствующих функций; при этом любой процесс/поток, имеющий необходимые права доступа к объекту "событие", может изменить его состояние.

В Windows различают события с ручным и с автоматическим сбросом (тип и начальное состояние задаются при создании события функцией `CreateEvent`). События с ручным сбросом ведут себя обычным образом: функция `SetEvent` переводит событие в свободное (сигнальное) состояние, а функция `ResetEvent` - в занятое (не сигнальное). События с автоматическим сбросом переводятся в занятое состояние либо явным вызовом функции `ResetEvent`, либо ожидающей функцией `WaitFor...`

```
HANDLE CreateEvent(
LPSECURITY_ATTRIBUTES lpEventAttributes,      // атрибут защиты
BOOL bManualReset,              // тип сброса TRUE - ручной
BOOL bInitialState,             // начальное состояние TRUE - сигнальное
LPCTSTR lpName                  // имя обьекта
);
```

- Mutex представляет объект синхронизации потоков, который может использоваться несколькими процессами. Объект Mutex можно использовать для защиты общего ресурса от одновременного доступа несколькими потоками или процессами. Каждый поток должен

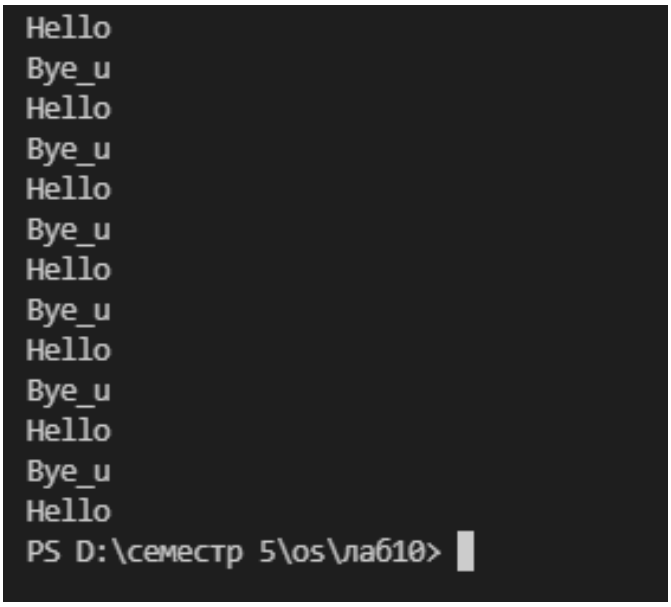
ожидать владения мьютексом, прежде чем он сможет выполнить код, обращающийся к общему ресурсу. Например, чтобы предотвратить одновременную запись двух потоков в общую память, каждый поток ждет владения объектом Mutex перед выполнением кода, который обращается к памяти. После записи в общую память поток освобождает объект Mutex. В программе по очереди в потоке и в цикле в main захватывается владение мьютексом и производятся действия с переменной sh.

```
HANDLE CreateMutex(
LPSECURITY_ATTRIBUTES lpMutexAttributes,      // атрибут безопастности
BOOL bInitialOwner,                    // флаг начального владельца
LPCTSTR lpName                         // имя объекта
);
```

- Semaphore используется для учета ресурсов. Сигнализирует потоку о доступности ресурса на данный момент.

Данный объект синхронизации позволяет ограничить доступ потоков к объекту синхронизации на основании их количества. Например, мы хотим, чтобы к какому-нибудь объекту могли обратиться максимум 3 потока. Не больше. Тогда нам нужен семафор. Сначала семафор инициализируется и ему передается количество потоков, которые к нему могут обратиться. Дальше при каждом обращении к ресурсу его счетчик уменьшается. Когда счетчик уменьшиться до 0 к ресурсу обратиться больше нельзя. При отсоединении потока от семафора его счетчик увеличивается, что позволяет другим потокам обратиться к нему. Сигнальному состоянию соответствует значение счетчика больше нуля. Когда счетчик равен нулю, семафор считается не установленным (сброшенным).

```
HANDLE CreateSemaphore(
LPSECURITY_ATTRIBUTES lpSemaphoreAttributes,  // атрибут доступа
LONG lInitialCount,    // инициализированное начальное состояние счетчика
LONG lMaximumCount,           // максимальное количество обращений
LPCTSTR lpName                // имя объекта
);
```



Изменим программу deadlock так, чтобы избежать взаимоблокировки. Проблема возникает в порядке захвата мьютексов.

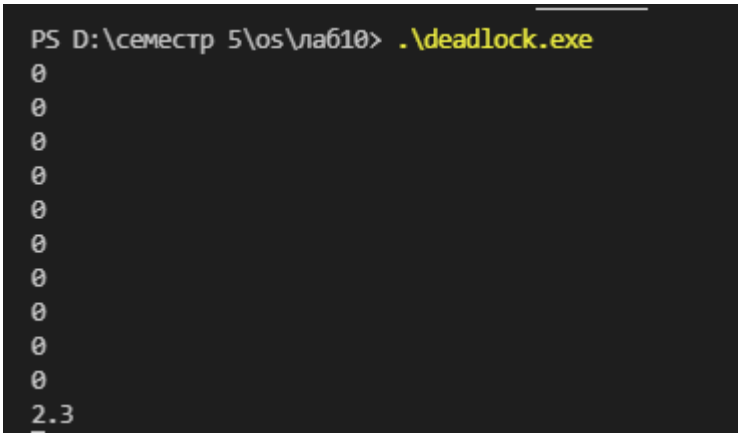


Рисунок 1 Момент блокировки

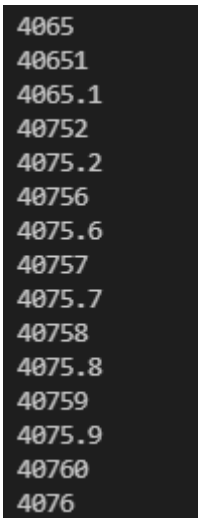


Рисунок 2 После исправления порядка захвата мьютексов

Листинг

```
#include <windows.h>
#include <process.h>
#include <stdio.h>

HANDLE hMutex1, hMutex2;
double sh1 = 0.0;
int sh2 = 0;
void Thread(void* pParams);

int main(void)
```

```
{
   hMutex1 = CreateMutex(NULL, FALSE, NULL);
   hMutex2 = CreateMutex(NULL, FALSE, NULL);

   _beginthread(Thread, 0, NULL);
   while(1)
   {
      WaitForSingleObject(hMutex2, INFINITE);
      printf("%d\n", sh2);
      WaitForSingleObject(hMutex1, INFINITE);
      printf("%g\n", sh1);
      ReleaseMutex(hMutex1);
      ReleaseMutex(hMutex2);
   }
   return 0;
}

void Thread(void* pParams)
{
   while(1)
   {
      WaitForSingleObject(hMutex2, INFINITE);
      sh2++;
      WaitForSingleObject(hMutex1, INFINITE);
      sh1 += 0.1;
      ReleaseMutex(hMutex1);
      ReleaseMutex(hMutex2);
   }
}
```