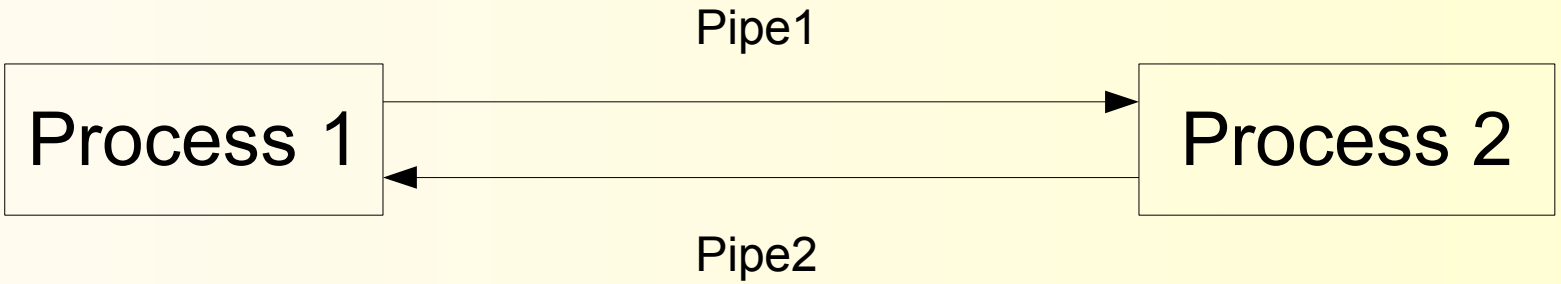
Параллельное выполнение процессов и потоков. Синхронизация.

- Проблемы параллельного выполнения
- Критические секции. Детерминированность.
- Атомарность операций
- Объекты синхронизации
- Классические проблемы синхронизации
- Реализация объектов синхронизации в ОС Windows
- Особенности реализации объектов синхронизации в ОС Unix

Задание

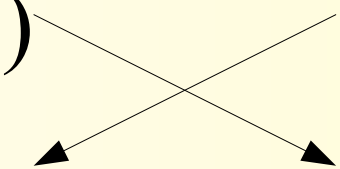


Два процесса обмениваются информацией через пару симплексных каналов. Существуют операции Create(PIPE), которая создает канал с указанными именем и ожидает открытия второго его конца, и Open (PIPE), которая открывает второй конец канала.

Напишите код для создания и открытия каналов обоими процессами.

Тупик (deadlock)

- Process1
  - Create(Pipe1)
  - Open(Pipe2)
- Process2
  - Create(Pipe2)
  - Open(Pipe1)



Операция Create(Pipe1) в первом процессе ожидает выполнения Open(Pipe1) во втором.

Однако Open(Pipe1) не может выполнится потому, что Create(Pipe2) не завершилась, ожидая Open(Pipe1)...

Задание

- Записи хранятся в массиве:

```
int Count;//Количество записей
struct ... Data[100];//Массив записей
```

- Процедура добавления записи:

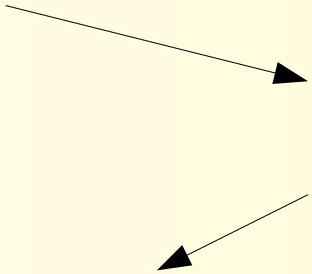
```
void Add( struct ... NewRecord) {
   Count++;
   Data[Count]=NewRecord;
}
```

- Безопасна ли процедура при одновременном выполении? Если нет, приведите контрпример.

Состояние гонки (race condition)

Count=7

- Thread1
  - `Count++//Count=8`
  - (ожидание)
  - (ожидание)
  - `Data[Count]=NewRecord //Data[9]`
- Thread2
  - (ожидание)
  - `Count++ //Count=9`
  - `Data[Count]=NewRecord //Data[9]`
  - (ожидание)



Детерминированность

- Набор программ, работающих с общими данными, детерминирован если при псевдопараллельном его выполнении для одинаковых входных данных он будет обеспечивать одинаковые результаты
- Параллельное выполнение недетерминированных программ приводит к редким, трудно обнаруживаемым и отлаживаемым ошибкам.

Критическая секция

- Часть программы, в которой есть обращение к общим данным, называется критической секцией.
- Условия детерминированности и работоспособности параллельных программ:
  - два процесса не должны одновременно находится в критических секциях;
  - в программе не должно быть предположений о скорости или количестве процессоров;
  - процесс, находящийся вне критической секции, не может блокировать другие процессы;
  - невозможна ситуация, в которой процесс вечно ждет попадания в критическую секцию.

Атомарность. Виды ожидания.

- Операция называется атомарной, если во время ее выполнения не может происходить переключения процессора на другой процесс (поток).
- Ожидание блокировки называют:
  - активным, если программа в цикле проверяет выполнение условия окончания блокировки;
  - пассивным, если операционная система переводит процесс (поток) в состояние блокированного до окончания блокировки.

Объекты синхронизации

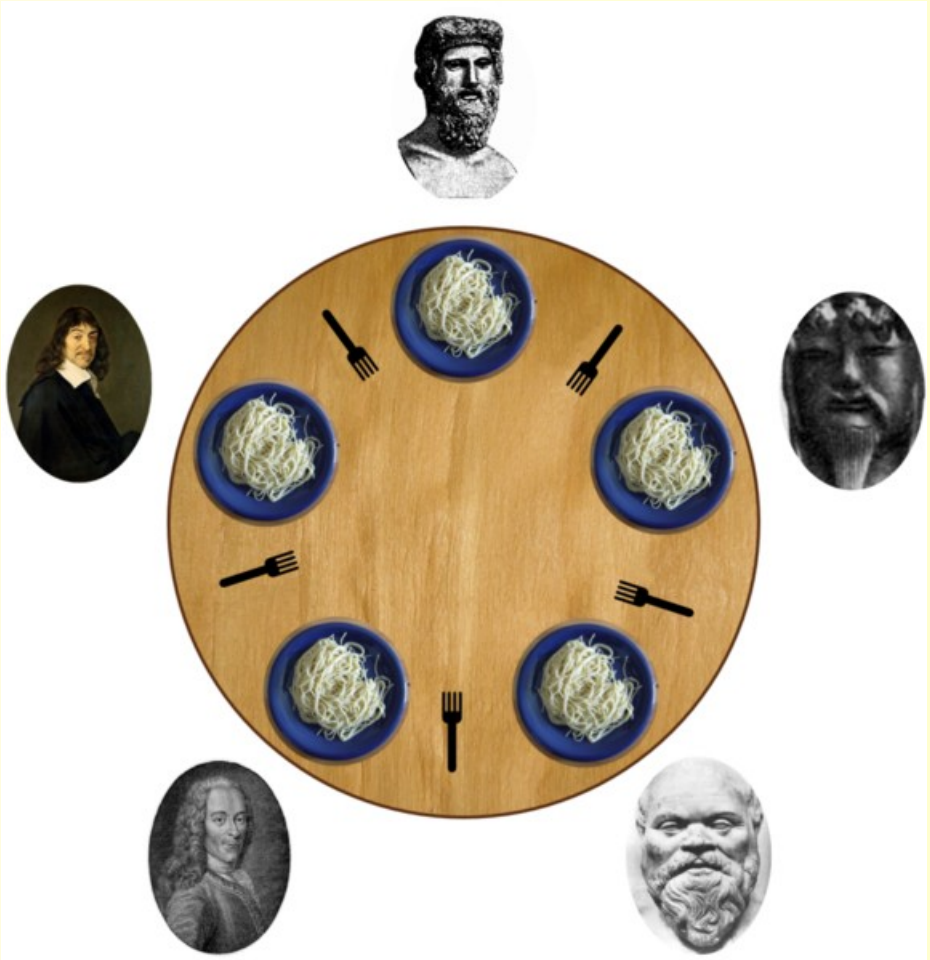
- Мьютекс (*mut*ual *ex*clusive) — двоичная переменная, находящаяся в двух состояниях — свободном и занятом. Операция «*проверить*» ждет освобождения мьютекса и переводит его в занятое состояние, операция «*освободить*» переводит в свободное состояние.
- Семафор — целая переменная с неотрицательным значением. Операция «*проверить*» вычитает значение семафора, если он положителен, и ожидает увеличения если равен нулю. Операция «*освободить*» увеличивает значение семафора на 1.

Классические проблемы синхронизации

- Проблема обедающих философов — контроль доступа к ограниченному количеству ресурсов
- Проблема производителя и потребителя — синхронизация доступа к каналу обмена информацией
- Проблема читателей и писателей — синхронизация доступа к общим данным
- Проблема спящего брадобрея — синхронизация в системе массового обслуживания

Проблема обедающих философов

- Философы сидят за круглым столом, перед каждым тарелка, между каждой тарелкой по вилке.
- Для еды нужна пара вилок.
- Философ периодически то думает, то ест.



Проблема производителя и потребителя

- Имеется буфер ограниченного размера (склад) и два процесса: производитель и потребитель.
- Производитель генерирует новые объекты и помощает их в буфер. Если буфер полон, то производитель переходит в состояние блокировки.
- Потребитель извлекает объекты из буфера. Если буфер пуст, то потребитель переходит в состояние блокировки.

Проблема читателей и писателей

- Существует общая база данных.
- В любое время из нее может читать любое количество процессов.
- Одновременная запись в базу запрещена.
- Если в базу данных производится запись, то данные могут находится в промежуточном состоянии, поэтому чтение во время записи также запрещено.

Проблема спящего брадобрея

- В кабинете у брадобрея стоит его стул, а также несколько стульев для ожидания.
- Если клиентов нет, то брадобрей спит.
- Приходящий клиент будит брадобрея, если он спит, и садится на обслуживание.
- Если в момент прихода клиента брадобрей занят, то клиент садится на стул ожидания.
- Если свободных стульев для ожидания нет, то клиент уходит.

Синхронизация в ОС Windows

- Функция `Sleep`
- Объекты синхронизации
- Функции ожидания
- Критические секции

Функция `Sleep`

- `VOID Sleep(DWORD dwMilliseconds);`
- Переводит поток в состояние блокировки на указанное количество милисекунд.
- В качестве значения параметра можно использовать макрос `INFINITE`.
- Поток не продолжит работу в течении *не менее* чем указанное количество времени.

Объекты синхронизации ОС Windows

- Объекты синхронизации ОС Windows — это объекты, которые имеют дескрипторы и могут находится в двух состояниях: *сигналированом* и *несигналированом*.
- Объекты синхронизации используются в функциях ожидания.
- Функция ожидания позволяет процессу дождаться *сигналирования* одного или нескольких объектов.
- Все операции над объектами синхронизации (в т.ч. производимые функциями ожидания) атомарны.

Объекты синхронизации

- Системные объекты
  - ввод с консоли (стандартный поток ввода)
  - процесс
  - поток
- Синхронизационные объекты
  - Событие
  - Мьютекс
  - Семафор
  - Таймер

Системные объекты синхронизации

- Ввод с консоли *сигналирован*, когда буфер не пуст. Ожидание на вводе с консоли позволяет дождаться появления новых данных от пользователя.
- Процесс и поток *сигналированы*, когда они завершили свою работу. Ожидание на дескрипторе процесса или потока позволяет дождаться окончания их работы.

События

- Событие — бинарный объект синхронизации, состоянием которого управляет пользователь.
- События могут использоваться для того, чтобы известить один поток (процесс) о наступлении какого-то события во втором.
- Типы событий:
  - сбрасываемые вручную — переходят из сигнального в несигнальное состояние по вызову специальной функции;
  - с автоматическим сбросом — переходят из сигнального в несигнальное состоение по завершению ожидания одного из потоков.

Операции над событиями

- Создание события:
  - `HANDLE CreateEvent (LPSECURITY_ATTRIBUTES lpEventAttributes, BOOL bManualReset, BOOL bInitialState, LPCTSTR lpName );`
- Сигналирование события
  - `BOOL SetEvent(HANDLE hEvent);`
- Сброс ручного события
  - `BOOL ResetEvent( HANDLE hEvent);`

Мьютекс

- Мьютекс — бинарный объект синхронизации. В каждый момент времени он свободен (сигналирован) или принадлежит какому-либо потоку (несигналирован).
- Успешно завершенная *функция ожидания* захватывает мьютекс. Для его освобождения поток использует функцию `ReleaseMutex`
- Если поток начал ожидание на мьютексе, которым уже владеет, то оно немедленно заканчивается. Функцию `ReleaseMutex` для освобождения необходимо вызвать столько раз, сколько была вызвана функция ожидания.

Операции над мьютексами

- Создание мьютекса
  - `HANDLE CreateMutex ( LPSECURITY_ATTRIBUTES lpMutexAttributes, BOOL bInitialOwner, LPCTSTR lpName);`
- Освобождение мьютекса
  - `BOOL ReleaseMutex(HANDLE hMutex);`

Семафор

- Семафор — объект, состояние которого описывается неотрицательным целым числом (с пределом значения).
- Семафор *сигналирован*, когда его значение *положительно*, *несигналирован* — когда оно *равно нулю*.
- *Функция ожидания* (после завершения ожидания при необходимости) *уменьшает* значение семафора на 1.
- Функция `ReleaseSemaphore` увеличивает значение семафора.
- Если же поток организует несколько ожиданий на одном семафоре не отпуская его, то каждое ожидание уменьшает семафор.

Операции над семафорами

- Создание семафора
  - HANDLE CreateSemaphore (LPSECURITY_ATTRIBUTES lpSemaphoreAttributes, LONG lInitialCount, LONG lMaximumCount, LPCTSTR lpName);
- Увеличение значения семафора
  - BOOL ReleaseSemaphore(HANDLE hSemaphore, LONG lReleaseCount, LPLONG lpPreviousCount);

Задание

Создайте безымянный семафор `hSem` и с максимальным значением 5 и начальным состоянием 1

Создание семафора

```
hSem=CreateSemaphore(NULL,1,5,NULL);
```

Таймер

- Таймер сигналируется по истечении определенного времени.
- Таймер может быть
  - сбрасываемым вручную — остается сигналированным до установки нового значения времени (перезапуска);
  - с автоматическим сбросом — переходит в несигналированное состояние при завершении операции ожидания.
- Периодический таймер автоматически перезапускается через определенный период времени.

Операции над таймерами

- Создание таймера
  - HANDLE CreateWaitableTimer
    (LPSECURITY_ATTRIBUTES lpTimerAttributes,
    BOOL bManualReset, LPCTSTR lpTimerName);
- Запуск таймера
  - BOOL SetWaitableTimer(HANDLE hTimer,
    const LARGE_INTEGER* pDueTime,/* 100нс,
    положительные — абсолютное время,
    отрицательные — относительное */
    LONG lPeriod,//период, мс
    PTIMERAPCROUTINE pfnCompletionRoutine,
    LPVOID lpArgToCompletionRoutine,
    BOOL fResume//обычно 0
  );

Функции ожидания

- Функции ожидания позволяют организовывать *пассивное* ожидание, пока не наступит одно из следующих условий:
  - один либо все (в зависимости от параметров) объекты сигналированы;
  - истек отведенный на ожидание интервал времени (может быть бесконечным).

WaitForSingleObject

- Ожидает сигналирования одного объекта

```
DWORD WaitForSingleObject( HANDLE
  hHandle, DWORD dwMilliseconds);
```

- Возвращаемые значения:
  - `WAIT_OBJECT_0` — ожидание завершилось успешно (объект сигналирован);
  - `WAIT_TIMEOUT` — время, отпущенное на ожидание истекло, а объект не сигналирован.

WaitForMultipleObjects

```
DWORD WaitForMultipleObjects
  ( DWORD nCount, const HANDLE*
  lpHandles, BOOL bWaitAll, DWORD
  dwMilliseconds);
```

- Возвращаемые значения:
  - `WAIT_OBJECT_0+i` — ожидание завершилось успешно (i-й объект сигналирован, если ожидали всех то i=0);
  - `WAIT_TIMEOUT` — время, отпущенное на ожидание истекло, а объекты не сигналированы.

Задание

Напишите фрагмент программы, который ожидает сигналирования одного из двух объектов: процесса `hChild` или семафора `hSem` в течении 1 мин, после чего выводит сообщение о том, что вышло ли время ожидания или какой-то (указать, какой) объект сигналирован.

Ожидание двух объектов

```
HANDLE hWait[2];
hWait[0]=hChild;
hWait[1]=hSem;
int result=WaitForMultipleObjects( 2,
  hWait, 0, 60000);
if(result==WAIT_TIMEOUT)
   printf("Time out");
else
   if(result==WAIT_OBJECT_0)
      printf("Process signaled");
   else
      printf("Semaphor signaled");
```

Задание

- Напишите фрагмент программы, которая:
  - дожидается освобождения мьютекса `h1`, но не захватывает его;
  - устанавливает событие `h2` в несигналируемое состояние, дабы предотвратить состояние гонки с другими программами;
  - выполняет некие действия (поставить многоточие)
  - освобождает событие `h2`.

Пример блокировки с предварительной проверкой условия

```
WaitForSingleObject(h1,INFINITE);
ReleaseMutex(h1);/* Освобождаем
  мьютекс, поскольку при ожидании он
  захватывается*/
ResetEvent(h2);
//.........
SetEvent(h2);
```

Критические секции

- Критические секции — способ межпоточной синхронизации.
- Критические секции не имеют имен и дескрипторов (представлены структурами), поэтому не могут использоваться между процессами.
- Критические секции подобны *мьютексам*, но работают быстрее последних за счет использования локальной памяти.

Операции над критическими секциями

- Инициализировать критическую секцию
  - `void InitializeCriticalSection (LPCRITICAL_SECTION lpCriticalSection);`
- Войти в критическую секцию
  - `void EnterCriticalSection ( LPCRITICAL_SECTION lpCriticalSection);`
- Покинуть критическую секцию
  - `void LeaveCriticalSection ( LPCRITICAL_SECTION lpCriticalSection);`

Особенности работы с семафорами в ОС Unix

- Семафоры создаются и уничтожаются наборами (массивами).
- Операции над семафорами выполняются группами, группа операций выполняется атомарно (полностью и без разрывов, либо не выполняется).
- Операции над семафорами:
  - увеличить значение семафора на определенное значение;
  - уменьшить значение семафора на определенное значение (с возможным ожиданием);
  - дождаться, пока значение семафора не станет равным нулю.