Лекция 5. Синхронизация потоков

Операционные системы

14 октября 2016 г.

Задержка

Windows API `Sleep()`

```
VOID WINAPI Sleep(
  _In_  DWORD dwMilliseconds          // 0, INFINITE, и т. д.
);
```

POSIX `sleep()` (<unistd.h>)

```
unsigned sleep(unsigned seconds);
```

Задержка (окончание)

Пример (Windows API `Sleep()`)

```
volatile bool g_vbWait = true;

DWORD WINAPI MyThreadProc(LPVOID pvData)
{
  // ...
  while (g_vbWait)
    Sleep(0);
  // ...
  return 0;
}
```

Событие

Windows API CreateEvent()

```
HANDLE WINAPI CreateEvent(
  _In_opt_ LPSECURITY_ATTRIBUTES lpEventAttributes,
  _In_     BOOL                  bManualReset,
  _In_     BOOL                  bInitialState,
  _In_opt_ LPCTSTR               lpctszName
);

BOOL WINAPI SetEvent(
  _In_     HANDLE                hEvent
);
```

Событие (продолжение)

Пример

```
HANDLE g_hEventGo =
  INVALID_HANDLE_VALUE;

DWORD WINAPI MyThreadProc(
  LPVOID pvData)
{
  // ...
  WaitForSingleObject(
    g_hEventGo, INFINITE);
  // ...
  return 0;
}
```

Пример (окончание)

```
int main()
{
  HANDLE hThread;
  g_hEventGo = CreateEvent(
    NULL, FALSE, FALSE, NULL);
  hThread = CreateThread(
    NULL, 0, MyThreadProc,
    NULL, 0, NULL);
  // ...
  SetEvent(g_hEventGo);
  // ...
  CloseHandle(g_hEventGo);
```

Событие (продолжение)

Пример

```
DWORD WINAPI MyThreadProc(LPVOID)
{
  int n = 0;
  while (g_vbRun)
  {
    WaitForSingleObject(
      g_hEventGo, INFINITE);
    cout
      << "Ev. " << ++ n << endl;
  }
  return 0;
}
```

Пример (окончание)

```
int main()
{
  // ...
  for (int i = 0; i < 3; ++ i)
  {
    Sleep(500);
    SetEvent(g_hEventGo);
  }
  g_vbRun = false;
  // ...
}
```

Событие (продолжение)

Пример

```
DWORD WINAPI MyThreadProc(LPVOID)
{
  int n = 0;
  while (g_vbRun)
  {
    WaitForSingleObject(
      g_hEventGo, INFINITE);
    cout
      << "Ev. " << ++ n << endl;
  }
  return 0;
}
```

Пример (окончание)

```
int main()
{
  // ...
  for (int i = 0; i < 3; ++ i)
  {
    Sleep(0);
    SetEvent(g_hEventGo);
  }
  Sleep(500);
  g_vbRun = false;
  // ...
```

Событие (окончание)

Windows API ResetEvent(), PulseEvent()

```
BOOL WINAPI ResetEvent(
  _In_     HANDLE                  hEvent
);

BOOL WINAPI PulseEvent(
  _In_     HANDLE                  hEvent
);
```

Использование установки и сброса события

Пример (Windows API SetEvent())

```
DWORD WINAPI MyThreadProc(LPVOID pvData)
{
  // ...
  SetEvent(g_hEventAsk);
  WaitForSingleObject(g_hEventAnswer, INFINITE);
  // ...
}
```

Пример ситуации гонки

Пример (пополнение счёта)

входные данные: d

начало

```
1  Считать(x)
2  x ← x + d
3  Записать(x)
```

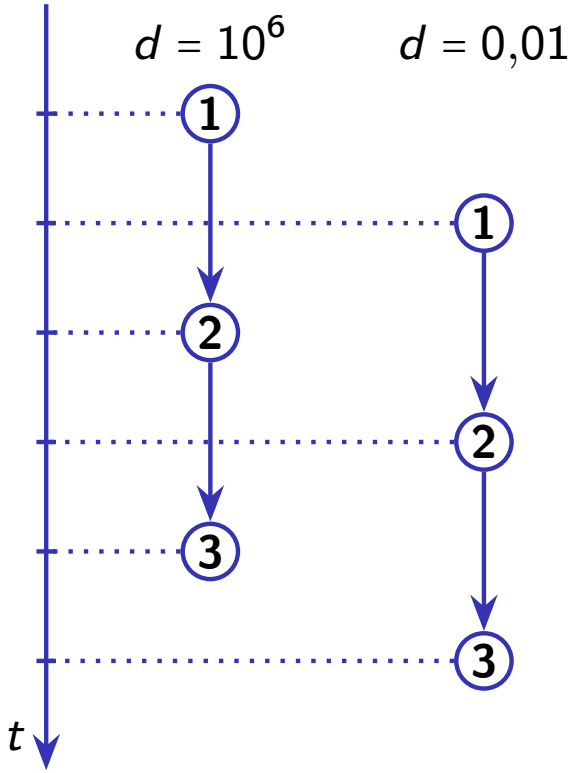

Рис. 1: последовательность исполнения алгоритма двумя процессами

Атомарность

Определения

Условие гонки: (*race condition*) — особенность функционирования системы, при которой её выходной сигнал непредсказуемо зависит от последовательности и/или временны́х задержек происходящих в ней событий.

Атомарность: (*atomicity*) — свойство операции в параллельной системе по отношению к набору ресурсов, определяющее невозможность одновременного доступа к ним до завершения операции (+ закрепление или откат).

Критическая секция: (*critical section*) — участок кода, исполняемый не более чем одним потоком из числа имеющих доступ к общему *ресурсу синхронизации*.

Взаимная блокировка: (*mutual exclusion, mutex*) — алгоритм обеспечения неодновременности использования общего ресурса разными потоками.

Атомарность

Определения

Условие гонки: (*race condition*) — особенность функционирования системы, при которой её выходной сигнал непредсказуемо зависит от последовательности и/или временны́х задержек происходящих в ней событий.

Атомарность: (*atomicity*) — свойство операции в параллельной системе по отношению к набору ресурсов, определяющее невозможность одновременного доступа к ним до завершения операции (+ закрепление или откат).

Критическая секция: (*critical section*) — участок кода, исполняемый не более чем одним потоком из числа имеющих доступ к общему *ресурсу синхронизации*.

Взаимная блокировка: (*mutual exclusion, mutex*) — алгоритм обеспечения неодновременности использования общего ресурса разными потоками.

Атомарность

Определения

Условие гонки: (*race condition*) — особенность функционирования системы, при которой её выходной сигнал непредсказуемо зависит от последовательности и/или временны́х задержек происходящих в ней событий.

Атомарность: (*atomicity*) — свойство операции в параллельной системе по отношению к набору ресурсов, определяющее невозможность одновременного доступа к ним до завершения операции (+ закрепление или откат).

Критическая секция: (*critical section*) — участок кода, исполняемый не более чем одним потоком из числа имеющих доступ к общему *ресурсу синхронизации*.

Взаимная блокировка: (*mutual exclusion, mutex*) — алгоритм обеспечения неодновременности использования общего ресурса разными потоками.

Атомарность

Определения

Условие гонки: (*race condition*) — особенность функционирования системы, при которой её выходной сигнал непредсказуемо зависит от последовательности и/или временны́х задержек происходящих в ней событий.

Атомарность: (*atomicity*) — свойство операции в параллельной системе по отношению к набору ресурсов, определяющее невозможность одновременного доступа к ним до завершения операции (+ закрепление или откат).

Критическая секция: (*critical section*) — участок кода, исполняемый не более чем одним потоком из числа имеющих доступ к общему *ресурсу синхронизации*.

Взаимная блокировка: (*mutual exclusion, mutex*) — алгоритм обеспечения неодновременности использования общего ресурса разными потоками.

Класс непрерывно-стохастических моделей

Определения

Заявка: (требование) — запрос на выполнение некоторой работы.

Система массового обслуживания: (*Queueing System*) — модель процесса функционирования системы, включающей:

- заявки (требования) на обслуживание — появляются в случайные моменты времени;
- каналы (устройства) обслуживания — включают очередь ожидающих заявок; время обслуживания представляет случайную величину.

Поток событий: последовательность событий, происходящих в случайные моменты времени.

Входящий поток: поток требований, поступающих в систему.

Выходящий поток: поток требований, покидающих систему.

Математический аппарат

Теория массового обслуживания

Раздел исследования операций, изучающий модели массового обслуживания. Позволяет вычислять оценки эффективности, включающие:

- среднее время ожидания в очереди;
- ожидаемое количество заявок, ожидающих обслуживания в очереди;
- вероятность того, что в данный момент система пуста (переполнена, имеет свободное устройство обслуживания);
- вероятность ожидания обслуживания заявки в течение заданного времени;
- ...

Классификация систем массового обслуживания

По количеству единиц обслуживания

- одноканальные;
- многоканальные.

По количеству этапов обслуживания

- однофазные;
- многофазные.

Классификация каналов (устройств) обслуживания

По дисциплине обслуживания

- в порядке поступления (FIFO);
- в случайном порядке (закон распределения).
- в соответствии с приоритетами.
- ...

По действию при занятом канале

- с отказами;
- с ожиданием;
- с ограничением ожидания (по времени/длине очереди).

Критическая секция

Windows API `InitializeCriticalSection()` и т. д.

```
void WINAPI InitializeCriticalSection(
  _Out_   LPCRITICAL_SECTION lpCriticalSection
);

void WINAPI DeleteCriticalSection(
  _Inout_ LPCRITICAL_SECTION lpCriticalSection
);
```

Критическая секция (продолжение)

Windows API EnterCriticalSection() и т. д.

```
void WINAPI EnterCriticalSection(
  _Inout_ LPCRITICAL_SECTION lpCriticalSection
);

void WINAPI LeaveCriticalSection(
  _Inout_ LPCRITICAL_SECTION lpCriticalSection
);

BOOL WINAPI TryEnterCriticalSection(
  _Inout_ LPCRITICAL_SECTION lpCriticalSection
);
```

Критическая секция (продолжение)

Пример

```
CRITICAL_SECTION g_CriticalS;
std::list <int> g_Numbers;

DWORD WINAPI MyThreadProc(LPVOID pvData)
{
  int n = 0;
  // ...
  EnterCriticalSection(&g_CriticalS);
  g_Numbers.push_front(n);
  LeaveCriticalSection(&g_CriticalS);
  // ...
  return 0;
}
```

Критическая секция (окончание)

Пример (окончание)

```
int main()
{
  HANDLE hThread;
  InitializeCriticalSection(&g_CriticalS);
  hThread = CreateThread(
    NULL, 0, MyThreadProc, NULL, 0, NULL);
  // ...
  EnterCriticalSection(&g_CriticalS);
  print(g_Numbers);
  LeaveCriticalSection(&g_CriticalS);
  // ...
  DeleteCriticalSection(&g_CriticalS);
}
```

Мьютекс

Windows API CreateMutex(), ReleaseMutex()

```
HANDLE WINAPI CreateMutex(
  _In_opt_  LPSECURITY_ATTRIBUTES lpMutexAttributes,
  _In_      BOOL                  bInitialOwner,
  _In_opt_  LPCTSTR               lpctszName
);

BOOL WINAPI ReleaseMutex(
  _In_      HANDLE                hMutex
);
```

Мьютекс (продолжение)

Пример

```
HANDLE g_hMutex = INVALID_HANDLE_VALUE;

DWORD WINAPI MyThreadProc(LPVOID pvData)
{
  int n = 1;
  // ...
  WaitForSingleObject(g_hMutex, INFINITE);
  g_Numbers.push_front(n);
  ReleaseMutex(g_hMutex);
  // ...
  return 0;
}
```

Мьютекс (продолжение)

Пример (окончание)

```
int main()
{
  HANDLE hThread;
  g_hMutex = CreateMutex(NULL, FALSE, NULL);
  hThread = CreateThread(
    NULL, 0, MyThreadProc, NULL, 0, NULL);
  // ...
  WaitForSingleObject(g_hMutex, INFINITE);
  print(g_Numbers);
  ReleaseMutex(g_hMutex);
  // ...
  CloseHandle(g_hMutex);
}
```

Мьютекс (продолжение)

POSIX pthread_mutex_init(), pthread_mutex_destroy() и т. д. (<pthread.h>)

```
int pthread_mutex_init(
  pthread_mutex_t *restrict           pMutex,
  const pthread_mutexattr_t *restrict pAttr);

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

int pthread_mutex_destroy(
  pthread_mutex_t *                   pMutex);
```

Мьютекс (продолжение)

POSIX pthread_mutex_lock(), pthread_mutex_unlock() и т. д.

```
int pthread_mutex_lock(
  pthread_mutex_t *                pMutex);

int pthread_mutex_trylock(
  pthread_mutex_t *                pMutex);

int pthread_mutex_unlock(
  pthread_mutex_t *                pMutex);
```

Мьютекс (продолжение)

Пример

```
pthread_mutex_t g_Mutex = PTHREAD_MUTEX_INITIALIZER;

void *MyThreadProc(void *pvData)
{
  int n = 2;
  // ...
  pthread_mutex_lock(&g_Mutex);
  g_Numbers.push_front(n);
  pthread_mutex_unlock(&g_Mutex);
  // ...
  return 0;
}
```

Мьютекс (окончание)

Пример (окончание)

```
int main()
{
  pthread_t hThread;
  int nRet = pthread_create(
    &hThread, NULL, MyThreadProc, NULL);
  // ...
  pthread_mutex_lock(&g_Mutex);
  print(g_Numbers);
  pthread_mutex_unlock(&g_Mutex);
  // ...
  pthread_join(hThread, NULL);
  pthread_mutex_destroy(&g_Mutex);
}
```

Блокировка с двойной проверкой

Пример (блокировка)

```
Data *get_data()
{
  // ...
  pthread_mutex_lock(&g_Mutex);
  if (!g_List.empty())
  {
    pData = g_List.back();
    g_List.pop_back();
  }
  pthread_mutex_unlock(&g_Mutex);
  // ...
}
```

Блокировка с двойной проверкой

Пример (блокировка)

```
Data *get_data()
{
  // ...
  if (!g_List.empty())
  {
    pthread_mutex_lock(&g_Mutex);
    pData = g_List.back();
    g_List.pop_back();
    pthread_mutex_unlock(&g_Mutex);
  }
  // ...
}
```

Блокировка с двойной проверкой

Пример (блокировка с двойной проверкой)

```
Data *get_data()
{
  // ...
  if (!g_List.empty())
  {
    pthread_mutex_lock(&g_Mutex);
    if (!g_List.empty())
    {
      pData = g_List.back();
      g_List.pop_back();
    }
    pthread_mutex_unlock(&g_Mutex);
  }
```

Барьер по памяти

Определение

Барьер по памяти: (*memory barrier, memory fence*) — точка в коде программы, для которой гарантируется, что все операции доступа к памяти, выполненные до неё, завершатся до начала операций с памятью после неё.

Пример (поток 0)

пока $b =$„ложь“, **выполнять**
  ; // *пусто*
// *барьер*
`Напечатать(`x`);`

Пример (поток 1)

$x \leftarrow 42$;
// *барьер*
$b \leftarrow$ „истина“;

Семафор

Windows API CreateSemaphore(), ReleaseSemaphore()

```
HANDLE WINAPI CreateSemaphore(
  _In_opt_  LPSECURITY_ATTRIBUTES lpSemaphoreAttributes,
  _In_      LONG                  lInitialCount,
  _In_      LONG                  lMaximumCount,
  _In_opt_  LPCTSTR               lpctszName
);

BOOL WINAPI ReleaseSemaphore(
  _In_      HANDLE                hSemaphore,
  _In_      LONG                  lReleaseCount,
  _Out_opt_  LPLONG               lplPreviousCount
);
```

Семафор (продолжение)

Пример

```
HANDLE g_hSemaphore = INVALID_HANDLE_VALUE;
/* volatile */ LONG g_vlNum = 0;

DWORD WINAPI MyThreadProc(LPVOID pvData)
{
  WaitForSingleObject(g_hSemaphore, INFINITE);
  InterlockedIncrement(&g_vlNum);
  cout << g_vlNum << " " << flush;
  ReleaseSemaphore(g_hSemaphore, 1, NULL);
  InterlockedDecrement(&g_vlNum);
  return 0;
}
```

Семафор (продолжение)

Пример (окончание)

```
int main()
{
  int i;
  HANDLE ahThreads[MY_NUM_THREADS];
  // ...
  g_hSemaphore = CreateSemaphore(
    NULL, MY_MAX_THREADS, MY_MAX_THREADS, NULL);
  for (i = 0; i < MY_NUM_THREADS; ++ i)
    ahThreads[i] = CreateThread(
      NULL, 0, MyThreadProc, NULL, CREATE_SUSPENDED, NULL);
  // ...
  CloseHandle(g_hSemaphore);
}
```

Семафор (продолжение)

POSIX sem_init() (<semaphore.h>)

```
int sem_init(
  sem_t *  pSem,
  int      bShared,
  unsigned uValue);

int sem_destroy(
  sem_t *  pSem);
```

POSIX sem_wait() и т. д.

```
int sem_wait(sem_t *pSem);

int sem_trywait(sem_t *pSem);

int sem_post(sem_t *pSem);
```

Семафор (продолжение)

Пример

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

sem_t g_Semaphore;
pthread_mutex_t g_Mutex =
  PTHREAD_MUTEX_INITIALIZER;
volatile LONG g_vlNum = 0;

void *MyThreadProc(void *)
{
```

Пример (продолжение)

```
  sem_wait(&g_Semaphore);
  pthread_mutex_lock(&g_Mutex);
  ++ g_vlNum;
  pthread_mutex_unlock(&g_Mutex);
  cout << g_vlNum << " " << flush;
  sem_post(&g_Semaphore);
  pthread_mutex_lock(&g_Mutex);
  -- g_vlNum;
  pthread_mutex_unlock(&g_Mutex);
  return 0;
}    // MyThreadProc()
```

Семафор (окончание)

Пример (окончание)

```
int main()
{
  int i;
  pthread_t ahThreads[MY_NUM_THREADS];
  sem_init(&g_Semaphore, 0, MY_MAX_THREADS);
  for (i = 0; i < MY_NUM_THREADS; ++ i)
    pthread_create(ahThreads[i], NULL, MyThreadProc, NULL);
  // ...
  sem_destroy(&g_Semaphore);
  pthread_mutex_destroy(&g_Mutex);
}
```

Кольцевой буфер

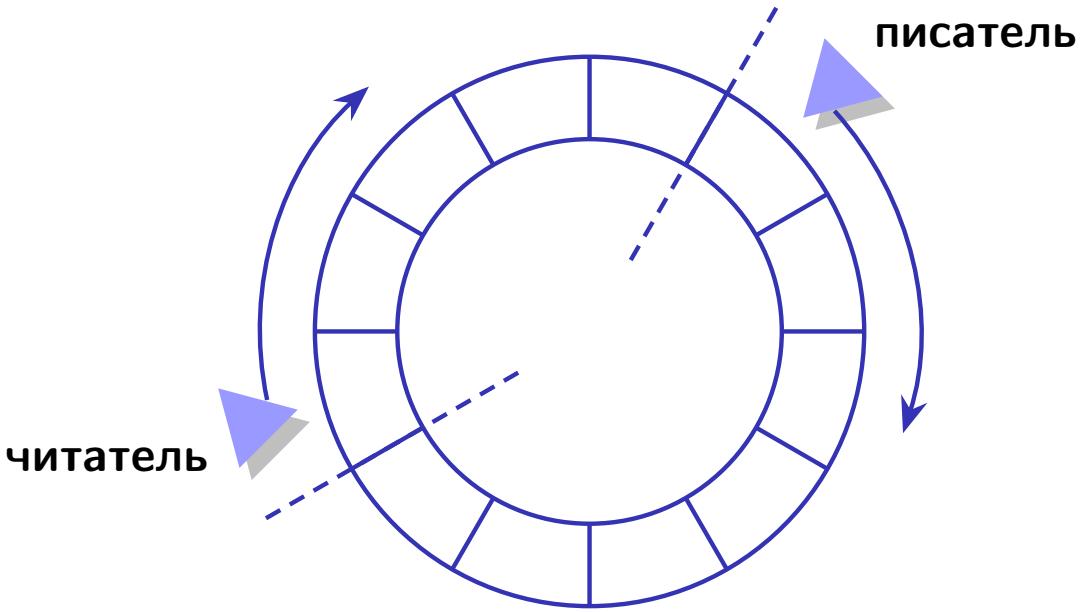

Рис. 2: концепция кольцевого буфера

Пример: кольцевой буфер POSIX

Пример

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

#include <stdio.h>
#include <stddef.h>
#include <math.h>

#define NUM_BLOCKS  10
#define BLOCK_SIZE   8
#define AMPLITUDE  100
#define PERIOD      30
```

Пример (продолжение)

```
#define INTERVAL (3 * PERIOD)
#define BUFFER_SIZE \
  (NUM_BLOCKS * BLOCK_SIZE)

int g_anBuffer[BUFFER_SIZE];
sem_t g_FreeSpace, g_UsedSpace;

int divides(int n1, int n2)
{
  return (n1 % n2 == 0);
}
```

Пример: кольцевой буфер POSIX (продолжение)

Пример (продолжение)

```
void *thread_proc(void *pvData)
{
  int t;
  for (t = 0; t <= INTERVAL; ++ t)
  {
    int i = t % BUFFER_SIZE;
    if (divides(i, BLOCK_SIZE))
      sem_wait(&g_FreeSpace);
    //
```

Пример (продолжение)

```
    g_anBuffer[i] =
      t == INTERVAL ?
      AMPLITUDE + 1 :
      AMPLITUDE *
      sin(1.0 * t / PERIOD);
    //
    if (t == INTERVAL ||
      divides(i + 1, BLOCK_SIZE))
      sem_post(&g_UsedSpace);
  }    // for (t = 0; ...)
}    // thread_proc()
```

Пример: кольцевой буфер POSIX (продолжение)

Пример (продолжение)

```
int main()
{
  int i = 0, nData;
  pthread_t hThread;
  //
  sem_init(&g_FreeSpace, 0, NUM_BLOCKS);
  sem_init(&g_UsedSpace, 0, 0);
  //
  pthread_create(&hThread, NULL, &thread_proc, NULL);
  //
```

Пример: кольцевой буфер POSIX (продолжение)

Пример (продолжение)

```
do
{
  if (divides(i, BLOCK_SIZE))
    sem_wait(&g_UsedSpace);
  //
  nData = g_anBuffer[i];
  printf(" (%d, %d)", i, nData);
  fflush(stdout);
  //
  i = (i + 1) % BUFFER_SIZE;
```

Пример (продолжение)

```
  //
  if (nData > AMPLITUDE ||
    divides(i, BLOCK_SIZE))
  {
    sem_post(&g_FreeSpace);
    printf(" |");
    fflush(stdout);
  }
}
while (nData <= AMPLITUDE);
```

Пример: кольцевой буфер POSIX (окончание)

Пример (окончание)

```
  //
  pthread_join(hThread, NULL);
  //
  printf("\n");
  //
  sem_destroy(&g_FreeSpace);
  sem_destroy(&g_UsedSpace);
}    // main()
```

Пример (сборка программы)

```
$ gcc -pthread sem_posix.c -lm
```

Барьер

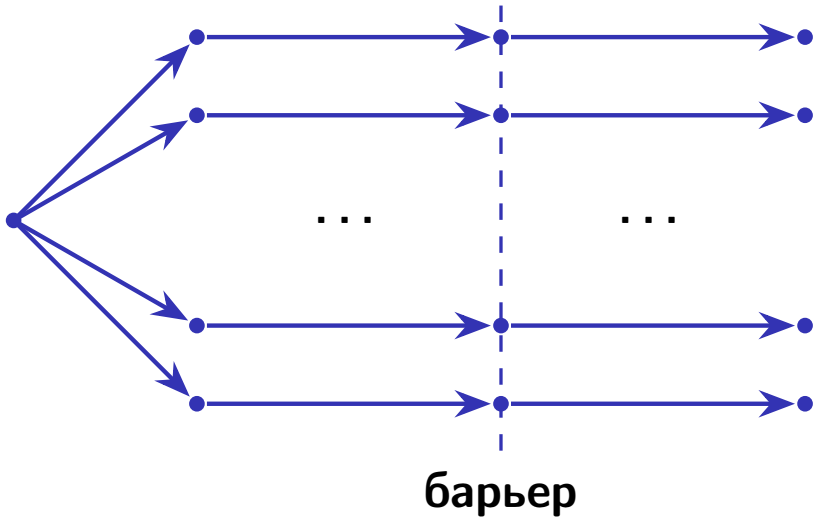

Рис. 3: концепция барьера

Барьер (продолжение)

Windows API `InitializeSynchronizationBarrier()` и т. д.

```
BOOL WINAPI InitializeSynchronizationBarrier(
  _Out_      LPSYNCHRONIZATION_BARRIER lpBarrier,
  _In_       LONG                      lTotalThreads,
  _In_       LONG                      lSpinCount
);

BOOL WINAPI DeleteSynchronizationBarrier(
  _Inout_    LPSYNCHRONIZATION_BARRIER lpBarrier
);
```

Барьер (продолжение)

Windows API EnterSynchronizationBarrier()

```
BOOL WINAPI EnterSynchronizationBarrier(
  _Inout_   LPSYNCHRONIZATION_BARRIER lpBarrier,
  _In_      DWORD                     dwFlags
);
```

Барьер (продолжение)

POSIX pthread_barrier_init() и т. д. (<pthread.h>)

```
int pthread_barrier_init(
  pthread_barrier_t *restrict           pBarrier,
  const pthread_barrierattr_t *restrict pAttr,
  unsigned                              uCount);

int pthread_barrier_destroy(
  pthread_barrier_t *                   pBarrier);

int pthread_barrier_wait(
  pthread_barrier_t *                   pBarrier);
```

Барьер (продолжение)

Пример

```
pthread_barrier_t g_Barrier;

void *MyThreadProc(void *)
{
  // ...
  pthread_barrier_wait(&g_Barrier);
  // ...
  return 0;
}
```

Барьер (окончание)

Пример (окончание)

```
int main()
{
  int i;
  pthread_t ahThreads[MY_NUM_THREADS];
  pthread_barrier_init(&g_Barrier, NULL, MY_NUM_THREADS);
  for (i = 0; i < MY_NUM_THREADS; ++ i)
    pthread_create(
      &ahThreads[i], NULL, MyThreadProc, NULL);
  // ...
  pthread_barrier_destroy(&g_Barrier);
}
```

Задача о читателях и писателях

Постановка задачи

- Чтение данных возможно одновременно любым количеством читателей.
- Запись данных возможна одновременно только одним писателем.
- Во время записи ни один читатель не имеет доступа к данным.

Дополнение к условию

- Чтение данных невозможно, если хотя бы один писатель изъявил о намерении записи.

Задача о читателях и писателях

Постановка задачи

- Чтение данных возможно одновременно любым количеством читателей.
- Запись данных возможна одновременно только одним писателем.
- Во время записи ни один читатель не имеет доступа к данным.

Дополнение к условию

- Чтение данных невозможно, если хотя бы один писатель изъявил о намерении записи.

Блокировка чтения/записи

Windows API InitializeSRWLock()

```
VOID WINAPI InitializeSRWLock(
  _Out_   PSRWLOCK pSRWLock
);

VOID WINAPI AcquireSRWLockShared(
  _Inout_ PSRWLOCK pSRWLock
);

VOID WINAPI ReleaseSRWLockShared(
  _Inout_ PSRWLOCK pSRWLock
);
```

Windows API SRWLOCK_INIT и т. д.

```
SRWLOCK rwlock = SRWLOCK_INIT;



VOID WINAPI AcquireSRWLockExclusive(
  _Inout_ PSRWLOCK pSRWLock
);

VOID WINAPI ReleaseSRWLockExclusive(
  _Inout_ PSRWLOCK pSRWLock
)
```

Блокировка чтения/записи (продолжение)

POSIX pthread_rwlock_init() и т. д. (<pthread.h>)

```
int pthread_rwlock_init(
  pthread_rwlock_t *restrict           pRWLock,
  const pthread_rwlockattr_t *restrict pAttr);

pthread_rwlock_t rwlock = PTHREAD_RWLOCK_INITIALIZER;

int pthread_rwlock_destroy(
  pthread_rwlock_t *                   pRWLock);
```

Блокировка чтения/записи (продолжение)

POSIX pthread_rwlock_rdlock() и т. д.

```
int pthread_rwlock_rdlock(
  pthread_rwlock_t *                    pRWLock);

int pthread_rwlock_wrlock(
  pthread_rwlock_t *                    pRWLock);

int pthread_rwlock_unlock(
  pthread_rwlock_t *                    pRWLock);
```

Пример: поиск простых чисел POSIX

Пример

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

#include <iostream>
#include <set>
#include <cstdlib>

typedef std::set <int> IntSet;
```

Пример (продолжение)

```
volatile bool g_vbRun = true;
IntSet g_Bitcoins;
pthread_rwlock_t g_Lock =
  PTHREAD_RWLOCK_INITIALIZER;

inline bool divides(
  int n1, int n2)
{
  return (n1 % n2 == 0);
}
```

Пример: поиск простых чисел POSIX (продолжение)

Пример (продолжение)

```
void *thread_proc(void *pvData)
{
  int n = 2;
  while (g_vbRun)
  {
    ++ n;
    bool bFound = true;
    //
    pthread_rwlock_rdlock(
      &g_Lock);
    //
```

Пример (продолжение)

```
    IntSet::const_iterator
      i = g_Bitcoins.begin(),
      e = g_Bitcoins.end();
    for (; i != e; ++ i)
      if (divides(n, *i))
      {
        bFound = false;
        break;
      }
    //
    pthread_rwlock_unlock(
      &g_Lock);
    //
```

Пример: поиск простых чисел POSIX (продолжение)

Пример (продолжение)

```
if (bFound)
{
  pthread_rwlock_wrlock(
    &g_Lock);
  //
  IntSet::const_iterator
    i = g_Bitcoins.begin(),
    e = g_Bitcoins.end();
  while (i != e)
    if (divides(*i, n))
      g_Bitcoins.erase(i ++);
    else
      ++ i;
```

Пример (продолжение)

```
      //
      g_Bitcoins.insert(n);
      //
      pthread_rwlock_unlock(
        &g_Lock);
    }    // if (bFound)
  }    // while (g_vbRun)
}    // thread_proc()
```

Пример: поиск простых чисел POSIX (продолжение)

Пример (продолжение)

```
int main()
{
  const int cnThreads = 8;
  pthread_t ahThreads[cnThreads];
  //
  g_Bitcoins.insert(2);
  //
  for (int i = 0; i < cnThreads; ++ i)
    pthread_create(&ahThreads[i], NULL, thread_proc, NULL);
  //
  sleep(10);
  //
  g_vbRun = false;
```

Пример: поиск простых чисел POSIX (окончание)

Пример (окончание)

```
  //
  for (int i = 0; i < cnThreads; ++ i)
    pthread_join(ahThreads[i], NULL);
  //
  IntSet::const_iterator
    i = g_Bitcoins.begin(),
    e = g_Bitcoins.end();
  for (; i != e; ++ i)
    std::cout << ' ' << *i;
  //
  std::cout << std::endl;
}    // main()
```

Инвариант и предикат

Определения

Инвариант: (*invariant*) — предположения об условиях, накладываемых на данные, обеспечивающие корректную работу программы.

Предикат: (*predicate*) — набор условий, накладываемых на данные, определяющий их некоторое состояние.

Условная переменная

Windows API InitializeConditionVariable() и т. д.

```
VOID WINAPI InitializeConditionVariable(
  _Out_   PCONDITION_VARIABLE pConditionVariable
);

VOID WINAPI WakeConditionVariable(
  _Inout_ PCONDITION_VARIABLE pConditionVariable
);

VOID WINAPI WakeAllConditionVariable(
  _Inout_ PCONDITION_VARIABLE pConditionVariable
);
```

Условная переменная (продолжение)

Windows API SleepConditionVariableCS() и т. д.

```
BOOL WINAPI SleepConditionVariableCS(
  _Inout_ PCONDITION_VARIABLE pConditionVariable,
  _Inout_ PCRITICAL_SECTION   pCriticalSection,
  _In_    DWORD               dwMilliseconds         // INFINITE
);

BOOL WINAPI SleepConditionVariableSRW(
  _Inout_ PCONDITION_VARIABLE pConditionVariable,
  _Inout_ PSRWLOCK            pSRWLock,
  _In_    DWORD               dwMilliseconds,
  _In_    ULONG               ulFlags
);                          // CONDITION_VARIABLE_LOCKMODE_SHARED
```

Условная переменная (продолжение)

POSIX pthread_cond_init(), pthread_cond_destroy() и т. д. (<pthread.h>)

```
int pthread_cond_init(
  pthread_cond_t *restrict           pCond,
  const pthread_condattr_t *restrict pAttr);

pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

int pthread_cond_destroy(
  pthread_cond_t *                   pCond);
```

Условная переменная (окончание)

POSIX pthread_cond_wait(), и т. д.

```
int pthread_cond_wait(
  pthread_cond_t *restrict          pCond,
  pthread_mutex_t *restrict         pMutex);

int pthread_cond_signal(
  pthread_cond_t *                  pCond);

int pthread_cond_broadcast(
  pthread_cond_t *                  pCond);
```

Ложные и украденные пробуждения

Определения

Ложное пробуждение: (*Spurious Wakeup*) — пробуждение, вызванное случайными причинами вместо операции сигнализирования.

Украденное пробуждение: (*Stolen Wakeup*) — пробуждение, после которого текущий поток был вытеснен, и сработавший до его продолжения другой поток изменил условие.

Пример: задача производителей и потребителей POSIX

Пример

```
int g_nItem = 0;
const int MY_MAX_QUEUE = 50;
std::queue <int> g_Queue;
pthread_mutex_t g_MutexQueue = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t g_CondNotEmpty = PTHREAD_COND_INITIALIZER;
pthread_cond_t g_CondNotFull = PTHREAD_COND_INITIALIZER;
volatile bool g_vbRun = true;
```

Пример: производители и потребители (продолжение)

Пример (продолжение)

```
void *thread_proc_get(void *pvData)
{
  while (true)
  {
    // ...
    pthread_mutex_lock(&g_MutexQueue);
    while (g_Queue.empty() && g_vbRun)
      pthread_cond_wait(&g_CondNotEmpty, &g_MutexQueue);
    if (!g_vbRun && g_Queue.empty())
    {
      pthread_mutex_unlock(&g_MutexQueue);
      break;
    }
```

Пример: производители и потребители (продолжение)

Пример (продолжение)

```
    // ... g_Queue.front()
    g_Queue.pop();
    //
    pthread_mutex_unlock(&g_MutexQueue);
    //
    pthread_cond_signal(&g_CondNotFull);
    //
  }   // while (true)
  //
  return 0;
}   // thread_proc_get()
```

Пример: производители и потребители (продолжение)

Пример (продолжение)

```
void *thread_proc_put(void *)
{
  while (true)
  {
    g_nItem = g_nItem % MY_MAX_QUEUE + 1;
    pthread_mutex_lock(&g_MutexQueue);
    while (g_Queue.size() == MY_MAX_QUEUE && g_vbRun)
      pthread_cond_wait(&g_CondNotFull, &g_MutexQueue);
    if (!g_vbRun)
    {
      pthread_mutex_unlock(&g_MutexQueue);
      break;
    }
```

Пример: производители и потребители (продолжение)

Пример (продолжение)

```
    g_Queue.push(g_nItem);
    // ... g_Queue.front()
    //
    pthread_mutex_unlock(&g_MutexQueue);
    //
    pthread_cond_signal(&g_CondNotEmpty);
    // ...
    //
  }   // while (true)
  //
  return 0;
}    // thread_proc_put()
```

Пример: производители и потребители (продолжение)

Пример (продолжение)

```
int main()
{
  pthread_t thread_get_1, thread_get_2, thread_put;
  pthread_create(
    &thread_get_1, NULL, thread_proc_get, (void *) 1);
  // ...
  pthread_mutex_lock(&g_MutexQueue);
  g_vbRun = false;
  pthread_mutex_unlock(&g_MutexQueue);
  //
  pthread_cond_broadcast(&g_CondNotEmpty);
  pthread_cond_broadcast(&g_CondNotFull);
```

Пример: производители и потребители (окончание)

Пример (окончание)

```
  //
  pthread_join(thread_get_1, NULL);
  // ...
  pthread_mutex_destroy(&g_MutexQueue);
  pthread_cond_destroy(&g_CondNotEmpty);
  pthread_cond_destroy(&g_CondNotFull);
}    // main()
```