Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные системы и системное программирование

Отчёт

к лабораторной работе №6

на тему

Средства синхронизации и взаимного исключения (Windows). Изучение и использование средств синхронизации и взаимного исключения.

Студент: гр.153502

Кирзнер А. П.

Проверил: Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

1 Теоретические сведения 3

2 Результат выполнения программы 5

Список использованных источников 6

Приложение А 7

# 1 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Функции ожидания позволяют потоку блокировать собственное выполнение. Функции ожидания не возвращаются до тех пор, пока не будут выполнены указанные условия. Тип функции ожидания определяет набор используемых критериев. При вызове функции ожидания она проверяет, выполнены ли критерии ожидания. Если условия не были выполнены, вызывающий поток переходит в состояние ожидания до тех пор, пока не будут выполнены условия условий ожидания или истекает указанный интервал времени ожидания.

Функции *SignalObjectAndWait*, *WaitForSingleObject* и *WaitForSingleObjectEx* требуют дескриптора для одного объекта синхронизации. Эти функции возвращаются при выполнении одного из следующих действий:

Указанный объект находится в состоянии сигнала.

Истекает интервал времени ожидания. Для интервала ожидания можно задать значение *INFINITE*, чтобы указать, что время ожидания не будет истекать.

Функция *SignalObjectAndWait* позволяет вызывающему потоку атомарно задавать состояние объекта как сигнальное и ожидать, пока состояние другого объекта будет задано как сигнальное [1].

Объект синхронизации – это объект, дескриптор которого можно указать в одной из функций ожидания для координации выполнения нескольких потоков. Несколько процессов могут иметь дескриптор одного и того же объекта синхронизации, что делает возможной синхронизацию между процессами.

Следующие типы объектов предоставляются исключительно для синхронизации:

– событие (уведомляет один или более ожидающих потоков о том, что произошло событие);

– мьютекс (может принадлежать только одному потоку за раз, что позволяет потокам координировать взаимоисключающий доступ к общему ресурсу);

– семафор (поддерживает число от нуля до некоторого максимального значения, ограничивая количество потоков, которые одновременно обращаются к общему ресурсу);

– таймер ожидания (уведомляет один или несколько ожидающих потоков о получении указанного времени) [2].

Объект критического раздела обеспечивает синхронизацию, аналогичную той, которая предоставляется объектом мьютекса, за исключением того, что критический раздел может использоваться только потоками одного процесса. Объекты критических разделов нельзя совместно использовать в процессах.

Объекты событий, мьютексов и семафоров также можно использовать в однопроцессорном приложении, но объекты критически важных разделов обеспечивают немного более быстрый и эффективный механизм синхронизации взаимоисключения (зависящий от процессора тест и инструкция установки). Как и объект мьютекса, объект критического раздела может принадлежать только одному потоку за раз, что делает его полезным для защиты общего ресурса от одновременного доступа.

Поток использует функцию *EnterCriticalSection* или *TryEnterCriticalSection* для запроса владения критически важным разделом. Функция *LeaveCriticalSection* используется для освобождения владения критически важным разделом. Если объект критического раздела в настоящее время принадлежит другому потоку, *EnterCriticalSection* бесконечно ожидает права владения.

Любой поток процесса может использовать функцию *DeleteCriticalSection* для освобождения системных ресурсов, выделенных при инициализации объекта критического раздела. После вызова этой функции объект критического раздела нельзя использовать для синхронизации [3].

Переменные условия – это примитивы синхронизации, которые позволяют потокам ожидать выполнения определенного условия. Переменные условия – это объекты пользовательского режима, которые нельзя совместно использовать в процессах.

Переменные условия позволяют потокам атомарно освобождать блокировку и переходить в спящее состояние. Переменные условия поддерживают операции, которые "пробуждают один" или "пробуждают все" ожидающие потоки. После пробуждения поток повторно получает блокировку, освобожденную при входе потока в спящее состояние.

Вызывающий объект должен выделить *CONDITION\_VARIABLE* структуру и инициализировать ее путем вызова метода *InitializeConditionVariable* (для динамической инициализации структуры) или назначения константы *CONDITION\_VARIABLE\_INIT* переменной структуры (для статической инициализации структуры) [4].

# 2 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ

Задача "Производители-Потребители" заключается в обеспечении согласованного доступа нескольких потоков к разделяемому циклическому буферу. Корректное решение должно удовлетворять следующим условиям:

– потоки выполняются параллельно;

– одновременно в критической секции, связанной с каждым критическим ресурсом, должно находиться не более одного потока;

– потоки должны корректно использовать операции с циклическим буфером.

При запуске создаётся два потока «производителей», два потока «потребителей» и буфер. Производитель генерирует число и записывает его в буфер, в то время как потребитель извлекает генерируемое число из буфера. Для более удобного и безопасного управления доступом к ресурсу (буферу) было выбрано использование мониторов. В *WinAPI* нет встроенной поддержки мониторов, поэтому было решено «эмулирование» монитора с использованием условных переменных и критических секций.

Реализация циклического буфера заключалась в хранении индексов ячеек буфера для предстоящего чтения и записи.

Каждый поток выполняет своё действие, а затем ждёт до трёх секунд для выполнения операций с разной скоростью.

В консоль выводится, какой именно поток выполняет действие, кем он является (производитель или потребитель), какой элемент он записывает или извлекает. Также выводится сообщение, если поток ожидает завершения действия предыдущего потока (если это необходимо). Следует отметить, что именно использование монитора помогает избежать ошибки при изъятии объекта (например, если буфер пустой, а потребитель хочет изъять элемент). Данная ситуация отображена на рисунке 1.

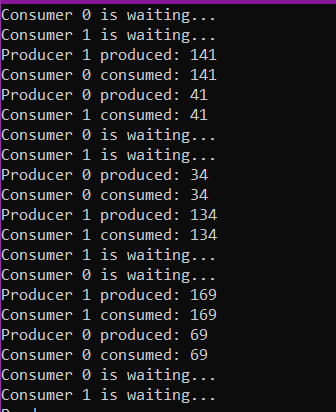


Рисунок 1 – Вывод программы

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Функции ожидания [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/sync/wait-functions>.

[2] Объекты синхронизации [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/sync/synchronization-objects>.

[3] Объекты критического раздела [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/sync/critical-section-objects>.

[4] Переменные условия [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/sync/condition-variables>.

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Исходный код программы**

#include <windows.h>

#include <iostream>

using namespace std;

const int BUFFER\_SIZE = 2;

int buffer[BUFFER\_SIZE];

int in = 0;

int out = 0;

CONDITION\_VARIABLE notFull;

CONDITION\_VARIABLE notEmpty;

CRITICAL\_SECTION criticalSection;

void produceItem(int item) {

buffer[in] = item;

in = (in + 1) % BUFFER\_SIZE;

}

int consumeItem() {

int item = buffer[out];

out = (out + 1) % BUFFER\_SIZE;

return item;

}

void producer(int id) {

while (true) {

int newItem = (id \* 100) + (rand() % 100);

EnterCriticalSection(&criticalSection);

while ((in + 1) % BUFFER\_SIZE == out) {

cout << "Producer " << id << " is waiting..." << endl;

SleepConditionVariableCS(&notFull, &criticalSection, INFINITE);

}

produceItem(newItem);

cout << "Producer " << id << " produced: " << newItem << endl;

WakeConditionVariable(&notEmpty);

LeaveCriticalSection(&criticalSection);

Sleep(rand() % 3000);

}

}

void consumer(int id) {

while (true) {

EnterCriticalSection(&criticalSection);

while (in == out) {

cout << "Consumer " << id << " is waiting..." << endl;

SleepConditionVariableCS(&notEmpty, &criticalSection, INFINITE);

}

int item = consumeItem();

cout << "Consumer " << id << " consumed: " << item << endl;

WakeConditionVariable(&notFull);

LeaveCriticalSection(&criticalSection);

Sleep(rand() % 3000);

}

}

int main() {

InitializeConditionVariable(&notFull);

InitializeConditionVariable(&notEmpty);

InitializeCriticalSection(&criticalSection);

const int NUM\_PRODUCERS = 2;

const int NUM\_CONSUMERS = 2;

HANDLE producerThreadHandles[NUM\_PRODUCERS];

HANDLE consumerThreadHandles[NUM\_CONSUMERS];

for (int i = 0; i < NUM\_PRODUCERS; i++) {

producerThreadHandles[i] = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)producer, (LPVOID)i, 0, NULL);

}

for (int i = 0; i < NUM\_CONSUMERS; i++) {

consumerThreadHandles[i] = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)consumer, (LPVOID)i, 0, NULL);

}

WaitForMultipleObjects(NUM\_PRODUCERS, producerThreadHandles, TRUE, INFINITE);

WaitForMultipleObjects(NUM\_CONSUMERS, consumerThreadHandles, TRUE, INFINITE);

for (int i = 0; i < NUM\_PRODUCERS; i++) {

CloseHandle(producerThreadHandles[i]);

}

for (int i = 0; i < NUM\_CONSUMERS; i++) {

CloseHandle(consumerThreadHandles[i]);

}

DeleteCriticalSection(&criticalSection);

return 0;

}