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

на тему

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

Студент: гр.153504

Пригожий К. А.

Проверил: Гриценко Н. Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc146635545)

[2 Теоретические сведения 4](#_Toc146635546)

[3 Результат выполнения программы 5](#_Toc146635547)

[Список использованных источников 7](#_Toc146635548)

[Приложение А 8](#_Toc146635549)

# 1 ЦЕЛЬ РАБОТЫ

Реализовать алгоритм философов обедающих, где философы (потоки) соревнуются за доступ к вилкам (ресурсам) с использованием мьютексов.

# 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Задача об обедающих философах — классический пример, используемый в информатике для иллюстрации проблем синхронизации при разработке параллельных алгоритмов и техник решения этих проблем.

Задача была сформулирована в 1965 году Эдсгером Дейкстрой как экзаменационное упражнение для студентов. В качестве примера был взят конкурирующий доступ к ленточному накопителю. Вскоре задача была сформулирована Энтони Хоаром в том виде, в каком она известна сегодня.

*Deadlock* — ситуация в многозадачной среде или СУБД, при которой несколько процессов находятся в состоянии бесконечного ожидания ресурсов, захваченных самими этими процессами.

Мьютекс – это синхронизационный объект, используемый для обеспечения взаимного исключения между потоками в многозадачных приложениях. Мьютексы позволяют только одному потоку одновременно получать доступ к критическому участку кода или ресурсам.

Для создания и управления мьютексами в *Win32* *API* используются функции, такие как *CreateMutex*, *WaitForSingleObject*, *ReleaseMutex*. Мьютексы могут использоваться как внутри одного процесса, так и между разными процессами [1].

Семафор - это синхронизационный объект, который может использоваться для управления доступом нескольких потоков к общим ресурсам. В отличие от мьютекса, семафор может разрешать доступ нескольким потокам одновременно, в зависимости от значения счетчика семафора.

В *Win32* *API* семафоры могут быть созданы с помощью функции *CreateSemaphore*, и потоки могут увеличивать или уменьшать значение счетчика семафора с помощью функций *ReleaseSemaphore* и *WaitForSingleObject.*[2]

*CreateThread* – это функция в *Win32* *API*, используемая для создания нового потока выполнения в текущем процессе. Она принимает указатель на функцию, которая будет выполняться в новом потоке, и аргументы, передаваемые этой функции.

После создания потока его выполнение начинается с указанной функции. *CreateThread* возвращает дескриптор нового потока, который может быть использован для управления и синхронизации потоками, например, с помощью мьютексов или семафоров.[3]

# 3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ

Реализованная программа является консольным приложением имитирующим действия пяти философов сидящих за одним столом. Каждый философ может думать и есть. Философы могут есть только двумя вилками. В рамках задачи вилки – общий ресурс (мьютекс). Если каждый философ одновременно возьмет вилку, например левую от себя, возникнет такая ситуация, при которой невозможно будет взять вторую вилку.

Одно из решений данной проблемы звучит следующим образом. Философов тянущихся за вилкой можно изобразить в виде циклического графа, где каждый философ ждет, пока сосед освободит нужный ресурс (рисунок 1).

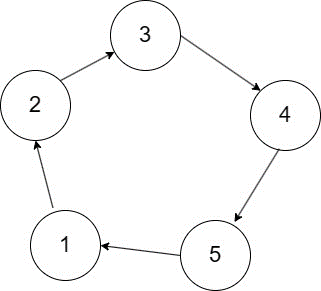


Рисунок 1 – Граф ожидающих друг друга философов

Решение заключается в изменении направления ожидания последнего философа в другую сторону (рисунок 2). Это значит, что последний философ будет брать не левую вилку первой, а правую.

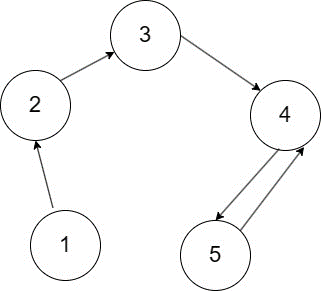


Рисунок 2 – Граф, где последний философ берет правую вилку

В результате программа выводит сообщения, показывающие, что делает философ с уникальным идентификатором в текущее время (рисунок 3).

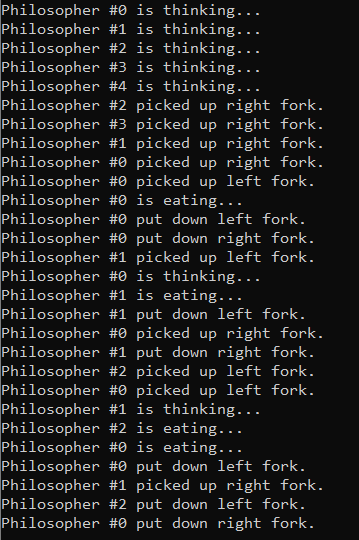


Рисунок 3 – Вывод программы

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Using Mutex Objects - Win32 apps | Microsoft Learn [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/sync/using-mutex-objects>

[2] Using Semaphore Objects - Win32 apps | Microsoft Learn [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/sync/using-semaphore-objects>

[3] Creating Threads - Win32 apps | Microsoft Learn [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/procthread/creating-threads>

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Исходный код программы**

**Файл main.h**

#pragma once

#include <windows.h>

#include <iostream>

#include <string>

const int NUM\_PHILOSOPHERS = 5;

HANDLE forks[NUM\_PHILOSOPHERS];

HANDLE philosophers[NUM\_PHILOSOPHERS];

HANDLE waiter;

HANDLE outputMutex;

int eatingTimes[NUM\_PHILOSOPHERS];

void print(std::string message);

void eat(int philosopherId);

void think(int philosopherId);

void pickUpFork(HANDLE fork, int philosopherId, std::string side);

void putDownFork(HANDLE fork, int philosopherId, std::string side);

void printEatingTimes();

**Файл main.cpp**

#include "main.h"

DWORD WINAPI PhilosopherThread(LPVOID param) {

int philosopherId = (int)param;

int leftFork = philosopherId;

int rightFork = (philosopherId + 1) % NUM\_PHILOSOPHERS;

if (philosopherId + 1 == NUM\_PHILOSOPHERS) {

int tmp = leftFork;

leftFork = rightFork;

rightFork = tmp;

}

while (true) {

think(philosopherId);

pickUpFork(forks[rightFork], philosopherId, "right");

pickUpFork(forks[leftFork], philosopherId, "left");

eat(philosopherId);

putDownFork(forks[leftFork], philosopherId, "left");

putDownFork(forks[rightFork], philosopherId, "right");

}

return 0;

}

int main() {

outputMutex = CreateMutex(NULL, FALSE, NULL);

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

forks[i] = CreateMutex(NULL, FALSE, NULL);

philosophers[i] = CreateThread(NULL, 0, PhilosopherThread, (LPVOID)i, 0, NULL);

}

WaitForMultipleObjects(NUM\_PHILOSOPHERS, philosophers, TRUE, 3000);

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

CloseHandle(forks[i]);

CloseHandle(philosophers[i]);

}

printEatingTimes();

return 0;

}

void print(std::string message)

{

WaitForSingleObject(outputMutex, INFINITE);

std::cout << message << "\n";

ReleaseMutex(outputMutex);

}

void eat(int philosopherId)

{

print("Philosopher #" + std::to\_string(philosopherId) + " is eating...");

Sleep(300);

eatingTimes[philosopherId]++;

}

void think(int philosopherId)

{

print("Philosopher #" + std::to\_string(philosopherId) + " is thinking...");

Sleep(300);

}

void pickUpFork(HANDLE fork, int philosopherId, std::string side)

{

WaitForSingleObject(fork, INFINITE);

print("Philosopher #" + std::to\_string(philosopherId) + " picked up " + side + " fork.");

}

void putDownFork(HANDLE fork, int philosopherId, std::string side)

{

ReleaseMutex(fork);

print("Philosopher #" + std::to\_string(philosopherId) + " put down " + side + " fork.");

}

void printEatingTimes()

{

std::string message = "Eating times: ";

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

message += std::to\_string(eatingTimes[i]) + " ";

}

print(message);

}