САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

КАФЕДРА ИНФОРМАТИКИ И ПРИКЛАДНОЙ МАТЕМАТИКИ

Отчет по лабораторной работе №3
по курсу «Операционные системы»

Вариант 7

Выполнил:
студент гр. P3318
Петкевич Константин Вячеславович
Преподаватель:
Лаздин А.В.

2017

Задание

Написать программу для консольного процесса, который состоит из трёх потоков: main , work, и третьего (см. варианты).
Поток main должен выполнить следующие действия:

1.создать массив, размерность и элементы которого вводятся пользователем с консоли; вывести размерность и элементы исходного массива на консоль;
2.запустить поток work;
3.запустить поток SumElement;
4.освобождение выходной поток stdout после вывода на консоль каждого нового элемента массива.
5.выводить на экран поэлементно элементы массива (итогового) параллельно с работой потока work;

Поток work должен выполнить следующие действия (Для синхронизации с потоком main – использовать семафор. Проверить работу используя бинарный семафор для синхронизации с потоком main, объяснить отличия, если есть!): запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве; Поиск в массиве лексем, (разделители – цифры). Полученные лексемы поместить в массиве слева, разделитель - пробел, остальные элементы - заполнить символом ‘0’. Элементы массива - символы.
Извещать поток main о новом элементе; после каждого готового элемента отдыхать в течение заданного интервала времени; известить поток SumElement о начале суммирования (момент запуска произойдёт после того, будет сформирован итоговый массив.

Поток SumElement должен выполнить следующие действия (Для синхронизации с потоком work, использовать критическую секцию!):

1.ждёт от потока work сигнал о начале суммирования;
2.выполнить суммирование элементов (кодов)
3.итогового массива; вывести итоговую сумму.

Листинг программы

```
#include <windows.h>
#include <iostream>
#include <string>

using namespace std;

struct workParameters
{
	char* a;
	int n;
	HANDLE hSemaphore;
	CRITICAL_SECTION cs;
};
struct sumElementsParameters
{
	char* a;
	int n;
	HANDLE hThreadWork;
	CRITICAL_SECTION cs;
```

```
};



void swap(char* a, char* b)
{
	char buf = *a;
	*a = *b;
	*b = buf;
}
DWORD WINAPI work(LPVOID workParams)
{
	struct workParameters* params = (struct workParameters*)workParams;

	char* a = (*params).a;
	int n = (*params).n;
	HANDLE hSemaphore = (*params).hSemaphore;


	TryEnterCriticalSection(&((*params).cs));
	int interval;
	cout << "Enter time interval: ";
	cin >> interval;
	ReleaseSemaphore(hSemaphore, 1, NULL);

	int j = 0;
	for (int i = 0; i < n; i++)
	{
		if (a[i] < '0' || a[i] > '9')
		{
			swap(&(a[j]), &(a[i]));
			j++;
			// отмечаем, что один элемент готов
			ReleaseSemaphore(hSemaphore, 1, NULL);
			Sleep(interval);
		}
		else if (a[j - 1] != ' ')
		{
			a[j] = ' ';
			j++;
			// отмечаем, что один элемент готов
			ReleaseSemaphore(hSemaphore, 1, NULL);
			Sleep(interval);
		}
	}

	for (int i = j; i < n; i++)
	{
		a[i] = '0';
		// отмечаем, что один элемент готов
		ReleaseSemaphore(hSemaphore, 1, NULL);
		Sleep(interval);
	}

	LeaveCriticalSection(&((*params).cs));
```

```
	return 0;
}


DWORD WINAPI SumElement(LPVOID sumElementsParams)
{
	struct sumElementsParameters* params = (struct
sumElementsParameters*)sumElementsParams;
	char* a = (*params).a;
	int n = (*params).n;

	WaitForSingleObject((*params).hThreadWork, INFINITE);

	TryEnterCriticalSection(&((*params).cs));

	int sum = 0;
	for (int i = 0; i < n; i++)
		sum += a[i];

	cout << "Sum: " << sum << endl;
	cout.flush();

	LeaveCriticalSection(&((*params).cs));

	return 0;
}

int main()
{
	char* a;
	int n;
	bool isWorkFinished;

	HANDLE hSemaphore;
	CRITICAL_SECTION cs;


	string str;
	cout << "Type any string: ";
	getline(cin, str);
	a = (char*)str.c_str();
	n = str.length();
	cout << "Number of elements in array: " << n << endl;

	InitializeCriticalSection(&cs);

	//Work
	HANDLE hThreadWork;
	DWORD IDThreadWork;

	hSemaphore = CreateSemaphore(NULL, 0, n, NULL);
	if (hSemaphore == NULL)
		return GetLastError();

	struct workParameters workParams;
	workParams.a = a;
	workParams.n = n;
```

```
	workParams.hSemaphore = hSemaphore;
	workParams.cs = cs;

	hThreadWork = CreateThread(NULL, 0, work,(LPVOID) (&workParams), 0,
&IDThreadWork);
	if (hThreadWork == NULL)
		return GetLastError();

	//SumElement
	HANDLE hThreadSumElement;
	DWORD IDThreadSumElement;

	struct sumElementsParameters sumElementsParams;
	sumElementsParams.a = a;
	sumElementsParams.n = n;
	sumElementsParams.hThreadWork = hThreadWork;
	sumElementsParams.cs = cs;

	hThreadSumElement = CreateThread(NULL, 0, SumElement, (LPVOID)
(&sumElementsParams), 0, &IDThreadSumElement);
	if (hThreadSumElement == NULL)
		return GetLastError();

	//Output
	WaitForSingleObject(hSemaphore, INFINITE);
	cout << endl << "Result array: ";

	for (int i = 0; i < n; i++)
	{
		WaitForSingleObject(hSemaphore, INFINITE);
		cout << a[i];
		cout.flush();
	}
	cout << endl;

	WaitForSingleObject(hThreadSumElement, INFINITE);

	CloseHandle(hSemaphore);
	CloseHandle(hThreadWork);

	cin >> n;

	return 0;
}
```