Министерство образования и науки Российской Федерации
Санкт-Петербургский Политехнический Университет Петра Великого
—
Институт кибербезопасности и защиты информации

Л А Б О Р А Т О Р Н А Я Р А Б О Т А № 3

«МЕХАНИЗМЫ МНОГОПОТОЧНОСТИ»

по дисциплине «Операционные системы»

Выполнил
студент гр. 4851003/10002 Галкин К. К.

Руководитель
К. н. т Крундышев В. М.

Санкт-Петербург
2023

1. ЦЕЛЬ РАБОТЫ

Цель работы — изучить принципы разработки многопоточных программ, изучить программный интерфейс операционных систем для организации многопоточности, получить навыки организации взаимодействия потоков в многопоточных программах.

Задачи:

- Написать программу expr.cpp, которая находит все возможные разложения числа на сумму натуральных чисел
- Написать программу qsort.cpp, которая производит быструю сортировку многопоточно.
- Написать программу msort.cpp, которая производит сортировки слиянием многопоточно
- Написать программу phill.cpp, решающую задачу об обедающих философах через управляющий процесс (официанта).

2. ХОД РАБОТЫ

2.1. expr.cpp

Идея кода: сделать для каждого потока из файла очередь задач, где задача – начальное число, с которого происходит декомпозиция числа

Таким образом, каждый поток берет задачу из своей очереди, выполняет рекурсивное разложение, захватывает примитив синхронизации и инкрементирует переменную – счетчик.

Блок – схема алгоритма работы программы.

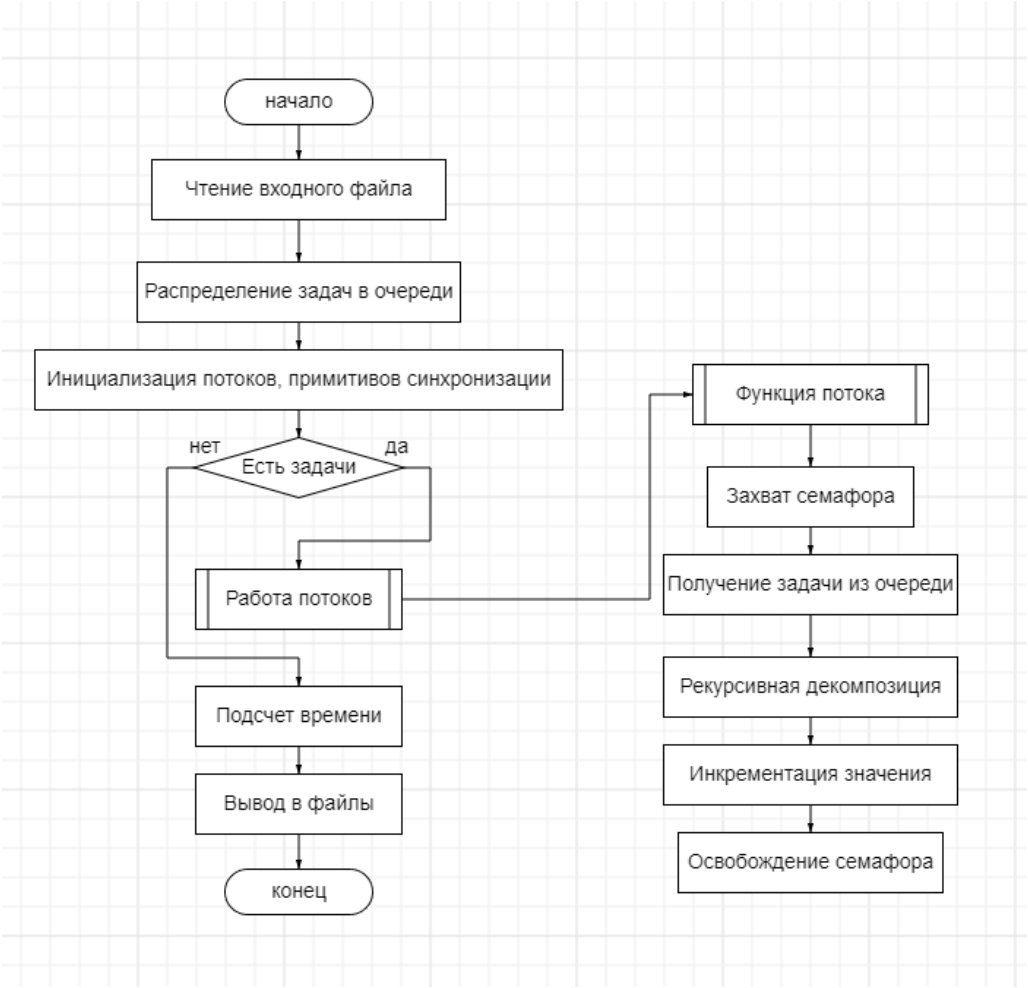


Диаграмма взаимодействия потоков:

Результаты работы программы с разными входными данными

N	10	10	10	30	30	50	50	80	80

Количество потоков	1	4	6	2	8	4	8	12	16
Результат, мс	0.127143	0.071514	0.049134	0.402379	0.300975	15.6196	15.6372	1370.71	1343.32

Список и описание используемых функций:

Функция	Назначение
void task_schedule()	Распределение задач по потокам
int num_decomposition(int n, int k)	Декомпиозиция n, начиная с k
void *thread_entry(void *param)	Функция метка для потока. В цикле берутся задачи.
void prepare_output(void)	Подсчет времени и запись результатов в файл

2.2. qsort.cpp

Идея: разделять массив на два подмассива до тех пор, пока размер подмассива меньше 1000.

Присутствует отдельная очередь задач, которая пополняется прямо в рекурсивной функции, если разность между индексами больше границы.

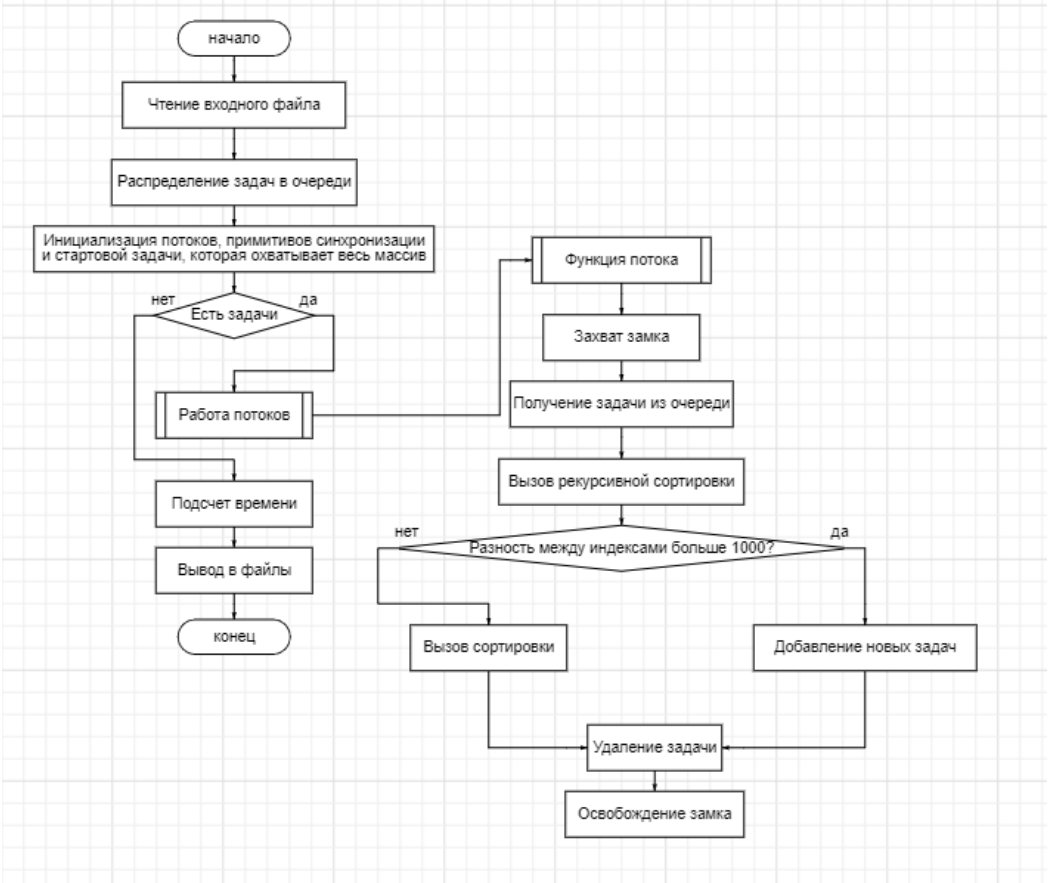


Диаграмма работы потоков.

Количество элементов	50K	50K	1M	1M	1M	5M	5M	5M	10M
Количество потоков	4	12	4	12	16	8	16	24	32
Время	150	61	673	216	170	1603	2172	2235	7120

Список используемых функций:

Функции	Назначение

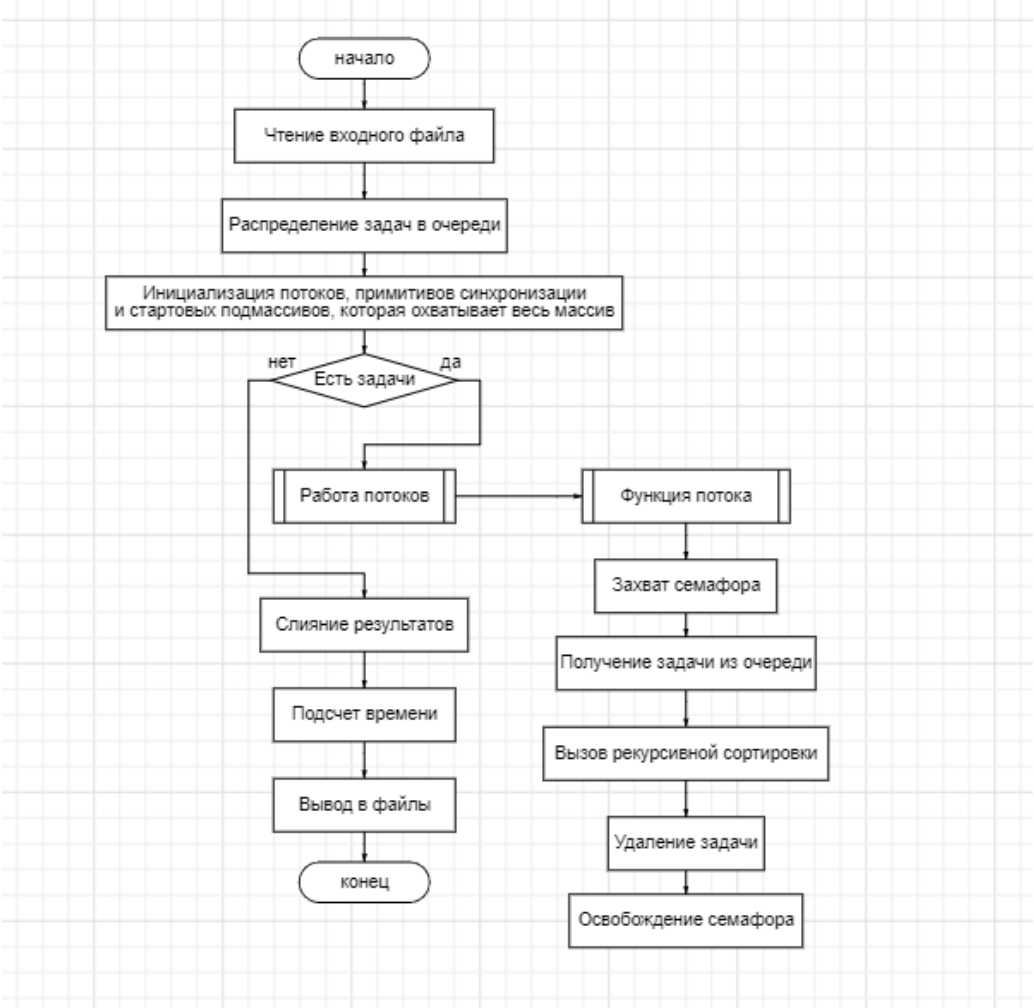
void addNewTask(int64_t left, int64_t right, int64_t lindex, int64_t rindex)	Добавление двух новых задач. Деление подмассива
void quickSort(vector<int> &arr, int64_t left, int64_t right)	Рекурсивная функция сортировки с делением подмассива на задачи
void *threadFunc(void *args)	Функция-метка потока, которая берет задач
void prepareOutput()	Вывод результатов

2.3. msort.cpp

Идея кода: разделить массив на n частей, где n – количество потоков.

Каждую часть отсортировать обычным слиянием.

После, все части, слить в один результирующий массив.

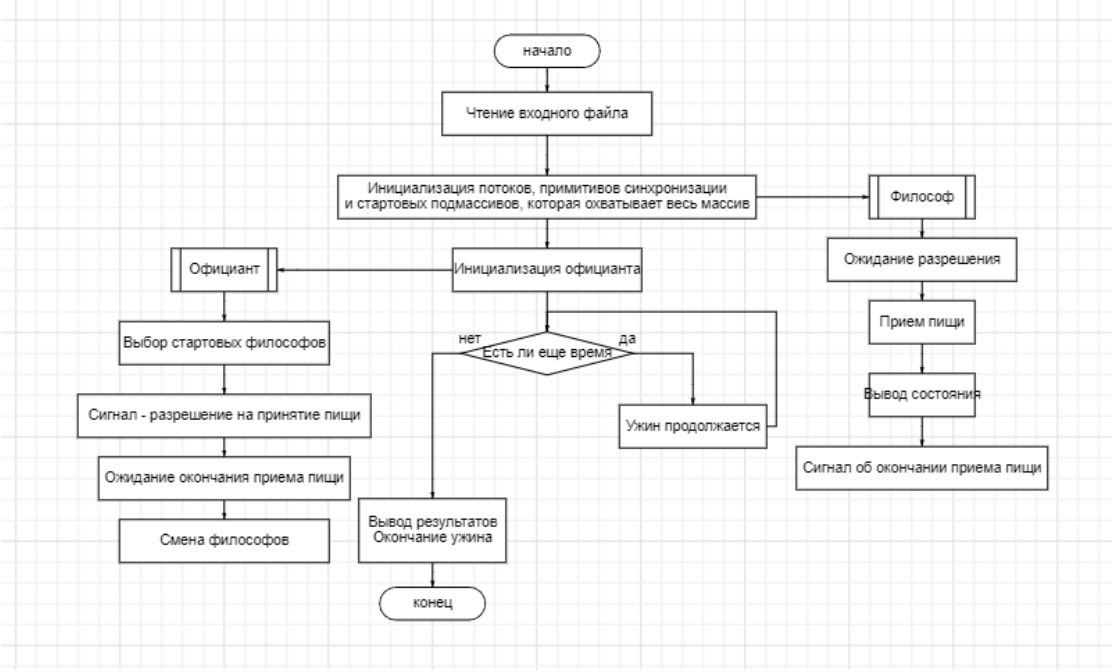


Количество элементов	50K	50K	1M	1M	1M	5M	5M	5M	10M
Количество потоков	4	12	4	12	16	8	16	24	32
Время	215	190	377	306	285	454	496	503	1238

Функции	Назначение
void addNewTask(int64_t left, int64_t right, int64_t lindex, int64_t rindex)	Добавление двух новых задач. Деление подмассива
void mergeSort(vector<int> &arr, int l, int r, TASK_HANDLE task)	Стандартная сортировка слиянием
DWORD WINAPI mergeThread(LPVOID lpParam)	Функция-метка потока для выполнения задачи
void finalMerge()	Слияние результирующих подмассивов.
void prepareOutput()	Вывод результатов

2.4. Задача об обедающих философах

Идея: есть официант, у которого запрашивают разрешение на принятие пищи. Разрешения контролируются через 3 события: событие – запрос, событие – ответ, событие – окончание.



В процессе работы каждый раунд ест два философа, начиная с 1 и 3 философов.

Функции	Назначение
void print_results(int phill, string step)	Вывод состояния философа
DWORD WINAPI control_thread(LPVOID param)	Функция официанта.
DWORD WINAPI phill_eating(LPVOID param)	Функция-метка философа
typedef struct phill	Структура каждого философа, которая содержит в себе 3 события.

3. ВЫВОД

В ходе выполнения лабораторной работы были изучены примитивы синхронизаций ОС Linux и Windows. Были разработаны программы, использующие разные наборы примитивов синхронизаций и тип используемой ОС.

Конечно, некоторые решения можно оптимизировать, например, сократив количество используемых примитивов синхронизации или препроцессинга данных для потоков.