Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Робототехники и комплексной автоматизации

КАФЕДРА Системы автоматизированного проектирования (РК-6)

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

Студент	Никифорова Ирина Андреевна
Группа	РК6-61б
Тип задания	лабораторная работа
Тема лабораторной работы	Многопроцессорное программирование

Студент _____ **Никифорова И. А.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ **Федорук В.Г.**
подпись, дата *фамилия, и.о.*

Оценка _____

Москва, 2019 г.

Оглавление

Задание на лабораторную работу

Составить программу, решающую задачу разрешимости логического выражения, содержащего 3 переменные (т.е. задачу отыскания значений всех комбинаций 3 логических переменных, делающих выражение истинным).

Замечание: Логическое выражение реализовать в виде функции языка СИ с тремя аргументами.

Для проверки истинности логического выражения на каждой комбинации переменных организовать 8 параллельно выполняемых процессов, при этом родственные связи процессов имеют вид дерева с тремя ярусами. В качестве значений логических переменных целесообразно использовать значения, возвращаемые системным вызовом fork. В каждом процессе осуществить печать значений переменных и истинности выражения.

Описание структуры программы и реализованных способов взаимодействия процессов

В ходе работы была написана программа, решающая задачу разрешимости логического выражения для трёх переменных, на языке Си.

Для чистоты кода, т.к. в языке Си отсутствует логический тип, был объявлен дополнительный алиас на тип char - BOOL(листинг 1).

Листинг 1. Дополнительный тип

```
1. typedef char BOOL;
```

Разработанная программа состоит из трёх функций: *logic*, *print_result* и *main.*

Функция *logic* (листинг 2) вычисляет значение логической функции при полученных аргументах. Она принимает указатель на массив аргументов типа BOOL *a*, а также размер массива *size*. Далее, используя эти аргументы, она вычисляет значение конъюнкции для всех элементов. Вместо конъюнкции может быть задана любая логическая функция.

Листинг 2. Функция *logic*

```
1. int logic(BOOL *a, int size) {
2.     BOOL logic = 1;
3.     for (int i = 0; i < size; i++) {
4.         logic = logic && a[i];
5.     }
6.     return logic;
7. }
```

Функция *print_result* (листинг 3) принимает те же значения, что и функция *logic*. Она выводит в консоль значения, записанные в переданном массиве, а также вычисляет функцию *logic* для полученных аргументов и печатает его.

Листинг 3. Функция *print_result*

```
1. void print_result(BOOL *a, int size) {
2.     for (int i = 0; i < size; i++) {
3.         a[i] = a[i] ? 1 : 0;
4.         printf("%d ", a[i]);
5.     }
6.     printf("%s\n", logic(a, size)? "True" : "False");
7. }
```

Функция *main* (листинг 4) является основной функцией программы. Она создает массив переменных типа BOOL. Он заполняется посредством записи в каждый из элементов результата выполнения системного вызова *fork*. В результате данных вызовов *fork*, в каждом из процессов образуется собственный уникальный набор аргументов для функции *logic*. При этом, такой способ получения аргументов учитывает абсолютно все возможные комбинации. На рисунке 1 представлено дерево вариантов массива аргументов для размерности 3.

Листинг 4. Функция *main*

```
1. int main() {
2.     BOOL a[N];
3.     for (int i = 0; i < N; i++) {
4.         a[i] = fork();
5.     }
6.     print_result(a, N);
7.     while(wait(NULL) > 0);
8. }
```

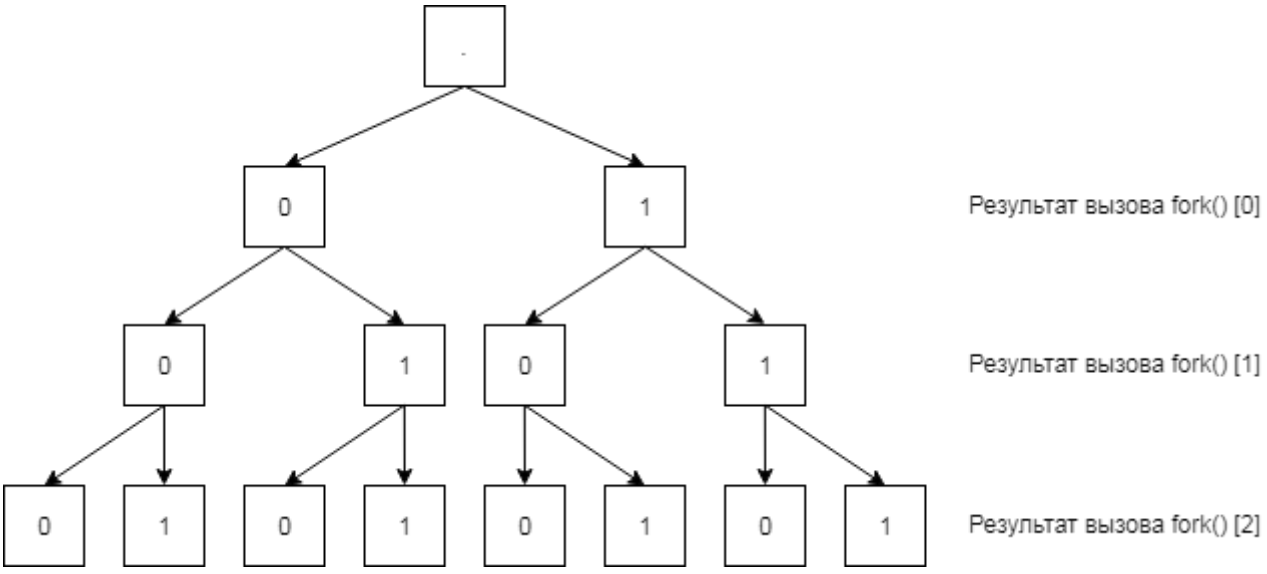

Рис. 1. Дерево результатов вызовов *fork* для трёх элементов массива аргументов функции *logic,* где 1 - любое значение, интерпретируемое, как истина, 0 - как ложь. Путь от вершины до листа представляет один из вариантов массива аргументов.

Это же дерево (рис. 1) наглядно демонстрирует порождение процессов в функции *main*. Процесс-родитель верхнего уровня (обозначен на рис. 1 точкой) производит первый вызов *fork*. В результате этого, получается два процесса (уровень “Результат вызова fork()[0]” на рис. 1), в каждом из которых будет свой результат вызова *fork*. Тот процесс, которому вернулся ноль - родительский, а где вернулось значение, интерпретируемое, как логическая единица -

порожденный. Аналогично, в результате второго вызова *fork* имеется четыре процесса, а в результате третьего - восемь.

Описание основных используемых структур данных

В программе использовалась структура данных массив. Она представляет из себя множество пронумерованных и упорядоченных по номеру элементов.

Массив был использован для хранения аргументов логической функции, описанной в функции на языке Си *logic*.

Блок-схема программы

Блок-схемы функций *logic*, *print_result* и *main* представлены на рисунках 2, 3 и 4 соответственно.

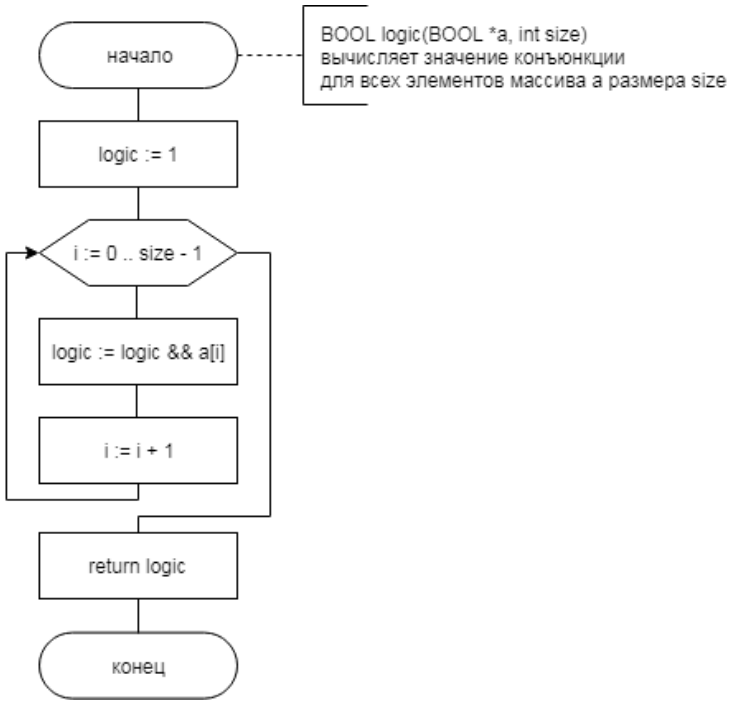

Рис. 2. Блок-схема функции *logic*

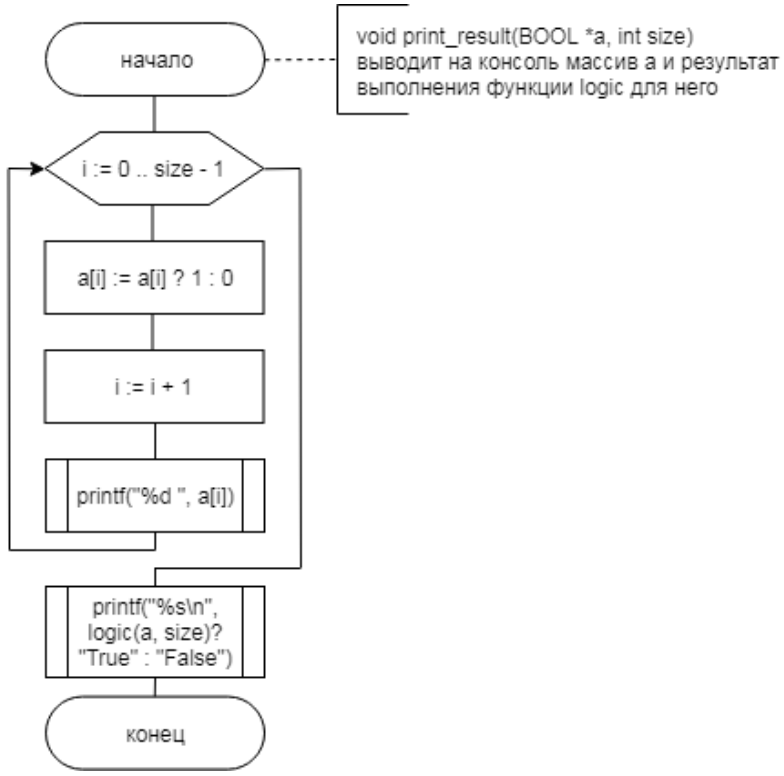

Рис. 3. Блок-схема функции *print_result*

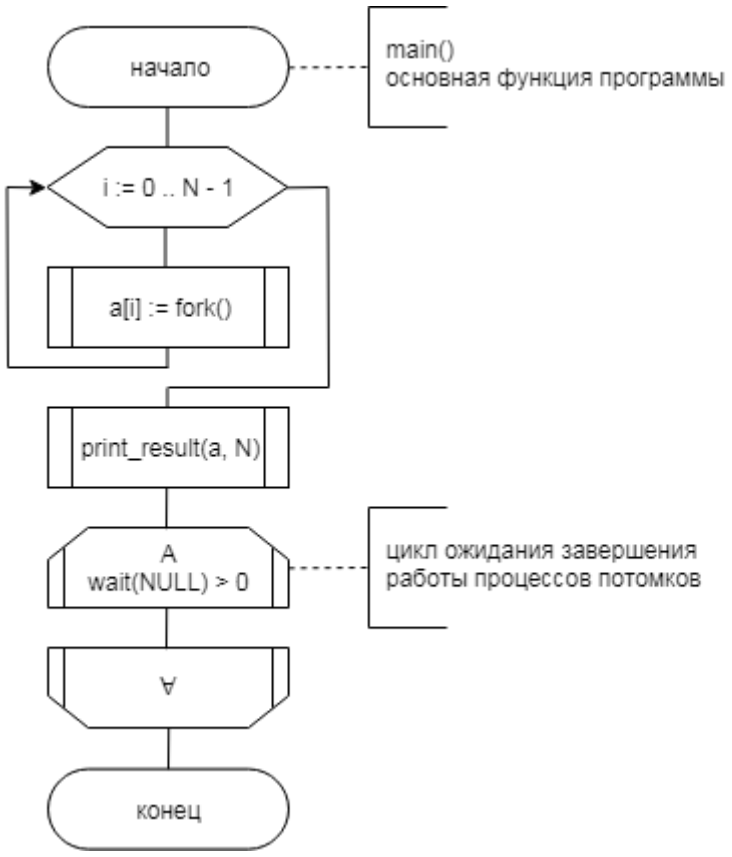

Рис. 4. Блок-схема функции *main*

Примеры результатов работы программы

В результате выполнения исходной программы был получен результат, приведенный в листинге 5. Далее были произведены следующие модификации программы и получены их результаты:

- логическая функция была изменена на дизъюнкцию для всех переменных (листинг 6)
- количество аргументов логической функции было изменено на пять и к ним применена конъюнкция (листинг 7) и дизъюнкция (листинг 8)

Листинг 5. Конъюнкция, три переменных

```
1. 1 1 0 False
2. 1 0 0 False
3. 0 0 0 False
4. 0 1 0 False
5. 0 0 1 False
6. 0 1 1 False
7. 1 0 1 False
8. 1 1 1 True
```

Листинг 6. Дизъюнкция, три переменных

```
1. 0 1 0 True
2. 0 0 0 False
3. 0 0 1 True
4. 0 1 1 True
5. 1 1 0 True
6. 1 0 0 True
7. 1 0 1 True
8. 1 1 1 True
```

Листинг 7. Конъюнкция, пять переменных

```
1.   0 1 1 1 0 False
2.   1 0 1 1 0 False
3.   1 1 1 1 0 False
4.   1 1 0 1 0 False
5.   0 1 1 0 0 False
6.   1 1 0 0 0 False
7.   0 1 1 0 1 False
8.   1 1 0 0 1 False
9.   1 1 1 0 0 False
10.  1 1 0 1 1 False
11.  1 0 1 0 0 False
12.  1 0 1 0 1 False
13.  0 0 0 0 0 False
14.  0 0 0 1 0 False
15.  0 0 0 0 1 False
16.  0 0 0 1 1 False
17.  1 0 0 0 0 False
18.  1 0 0 1 0 False
19.  1 0 0 0 1 False
20.  1 0 0 1 1 False
21.  0 0 1 1 0 False
22.  0 0 1 0 0 False
23.  1 1 1 0 1 False
```

Листинг 8. Дизъюнкция, пять переменных

```
1.   1 0 1 1 0 True
2.   1 0 1 0 0 True
3.   0 0 1 1 0 True
4.   1 0 1 0 1 True
5.   1 0 0 0 0 True
6.   1 0 0 0 1 True
7.   0 1 1 1 0 True
8.   1 1 1 1 0 True
9.   0 1 0 1 0 True
10.  0 0 0 1 0 True
11.  0 1 0 0 0 True
12.  0 1 0 0 1 True
13.  1 1 0 1 0 True
14.  0 1 1 0 0 True
15.  1 0 0 1 0 True
16.  1 0 0 1 1 True
17.  1 0 1 1 1 True
18.  0 1 0 1 1 True
19.  0 1 1 0 1 True
20.  0 0 1 0 0 True
21.  0 0 0 0 0 False
22.  0 0 0 0 1 True
23.  1 1 1 0 0 True
```

24.	0	0	1	0	1	False
25.	0	0	1	1	1	False
26.	1	0	1	1	1	False
27.	0	1	0	0	0	False
28.	0	1	0	0	1	False
29.	0	1	0	1	0	False
30.	0	1	0	1	0	False
31.	0	1	1	1	1	False
32.	1	1	1	1	1	True

24.	1	1	1	0	1	True
25.	0	0	0	1	1	True
26.	1	1	0	0	0	True
27.	1	1	0	0	1	True
28.	0	0	1	0	1	True
29.	1	1	0	1	1	True
30.	0	0	1	1	1	True
31.	0	1	1	1	1	True
32.	1	1	1	1	1	True

Текст программы

В листинге 9 приведен полный текст исходной программы (без модификаций).

Листинг 9. Полный текст программы

```
1.  #include <unistd.h>
2.  #include <string.h>
3.  #include <stdlib.h>
4.  #include <sys/types.h>
5.  #include <sys/wait.h>
6.  #include <stdio.h>
7.
8.  #define N 3
9.  typedef char BOOL;
10.
11. // вычисляет логическую функцию
12. int logic(BOOL *a, int size) {
13.     BOOL logic = 1;
14.     for (int i = 0; i < size; i++) {
15.         logic = logic && a[i];
16.     }
17.     return logic;
18. }
19.
20. // вычисляет и распечатывает результаты
21. void print_result(BOOL *a, int size) {
22.     for (int i = 0; i < size; i++) {
23.         a[i] = a[i] ? 1 : 0;
24.         printf("%d ", a[i]);
25.     }
26.     printf("%s\n", logic(a, size)? "True" : "False");
27. }
28.
29. int main() {
30.     BOOL a[N];
31.
32.     for (int i = 0; i < N; i++) {
33.         a[i] = fork();
34.     }
35.
36.     print_result(a, N);
37.
38.     // ожидает завершения всех потомков текущего процесса
39.     while(wait(NULL) > 0);
40. }
```