МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №6

по дисциплине: Алгоритмы и структуры данных
тема: «Структуры данных «стек» и «очередь» (С)»

Выполнил: ст. группы ПВ-202
Буйвало Анастасия Андреевна

Проверил:
Кабалянц Петр Степанович
Маньшин Илья Михайлович

Белгород 2021 г.

Лабораторная работа №6

«Структуры данных «стек» и «очередь» (С)»

Цель работы:

Изучить СД типа «стек» и «очередь», научиться их программно реализовывать и использовать.

Задание:

1. Для СД типа «стек» и «очередь» определить:
   1.1. Абстрактный уровень представления СД:
      1.1.1. Характер организованности и изменчивости.
      1.1.2. Набор допустимых операций.
   1.2. Физический уровень представления СД:
      1.2.1. Схему хранения.
      1.2.2. Объем памяти, занимаемый экземпляром СД.
      1.2.3. Формат внутреннего представления СД и способ его интерпретации.
      1.2.4. Характеристику допустимых значений.
      1.2.5. Тип доступа к элементам.
   1.3. Логический уровень представления СД.
      a. Способ описания СД и экземпляра СД на языке программирования.
2. Реализовать СД типа «стек» и «очередь» в соответствии с вариантом индивидуального задания в виде модуля.
3. Разработать программу, моделирующую вычислительную систему с постоянным шагом по времени (дискретное время) в соответствии с вариантом индивидуального задания (табл. 16) с использованием модуля, полученного в результате выполнения пункта 2. Результаты работы программы представить в виде таблицы 15. В первом столбце указывается время моделирования 0, 1, 2, …, N. Во втором – для каждого момента времени указываются имена объектов (очереди – F_1, F_2, …, F_N, стеки – S_1, S_2, …, S_M, процессоры $P_1, P_2, \dots, P_K$, а в третьем – задачи (имя, время) находящиеся в объектах

Таблица 15

Результаты работы программы

Время	Объекты	Задачи
0	F_1	(имя,время),(имя,время),. . .,(имя,время)
	:	: : :
	F_N	(имя,время),(имя,время),. . .,(имя,время)
	S_1	(имя,время),(имя,время),. . .,(имя,время)
	:	: : :
	S_M	(имя,время),(имя,время),. . .,(имя,время)
	P_1	(имя,время)
	:	:
	P_K	(имя,время)
1	F_1	(имя,время),(имя,время),. . .,(имя,время)
	:	: : :
	F_N	(имя,время),(имя,время),. . .,(имя,время)
	S_1	(имя,время),(имя,время),. . .,(имя,время)
	:	: : :
	S_M	(имя,время),(имя,время),. . ., (имя,время)
	P_1	(имя,время)
	:	:
	P_K	(имя,время)
:	:	: : :

7	7	7	7

Выполнение работы:

Задание 1:

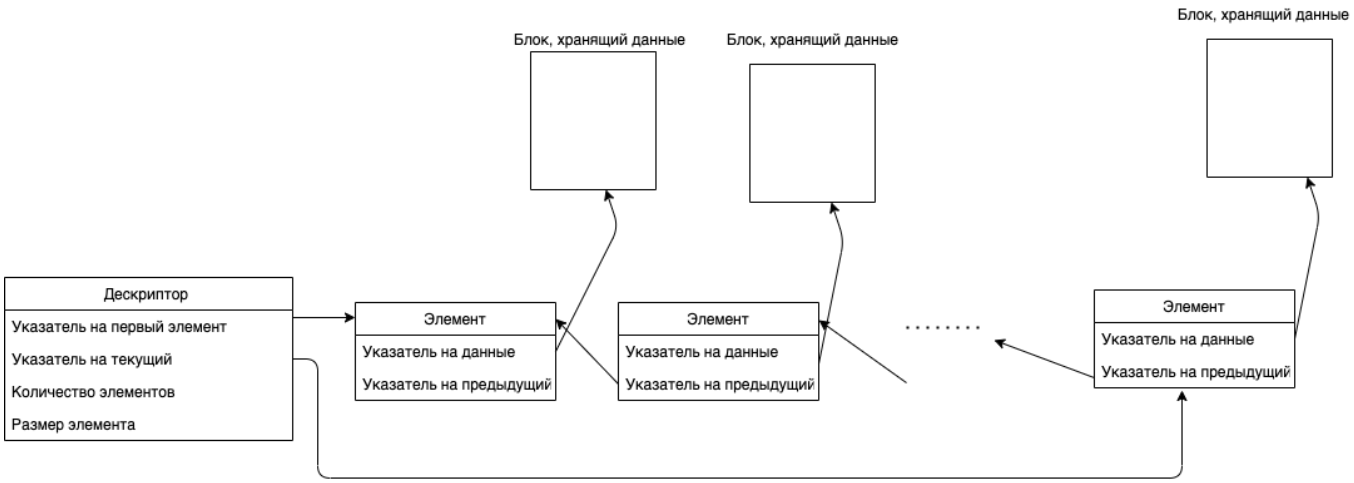
СД «Стек»:

Абстрактный уровень представления СД:

1. Характер организованности – последовательность;
2. Изменчивость – динамическая СД;
3. Набор допустимых операций: инициализация, включение элемента, исключение элемента, чтение элемента, проверка пустоты стека, уничтожение стека.

Физический уровень представления СД:

1. Схема хранения: последовательная или связная.
2. Объем памяти, занимаемый экземпляром СД: в зависимости от реализации.
3. Формат внутреннего представления СД(ОЛС) и способ его интерпретации:



Объем занимаемой памяти: $N * (8 + M) + 16$, где N – количество узлов списка, M – размер базового типа, 8 – размер двух указателей, 16 – размер дескриптора (2 числа + 2 указателя).

4. Характеристика допустимых значений: в зависимости от реализации
5. Тип доступа к элементам: в зависимости от реализации

Логический уровень представления СД:

Способ описания СД (ОЛС 4 поля) и экземпляра СД на языке С:
Дескриптор ОЛС состоит из 4-х полей:
1 - Количество элементов списка;
2 - Указатель на первый элемент;
3 - Указатель на текущий элемент;
4 – Размер элемента

Способ описания СД:

```
typedef struct List {
ptrel start; //Указатель на первый элемент
ptrel ptr; // Указатель на текущий элемент
unsigned N; // Кол-во элементов списка
unsigned size //Размер элемента
} Stack;
```

Способ описания экземпляра: `Stack S;`

СД «Очередь»:

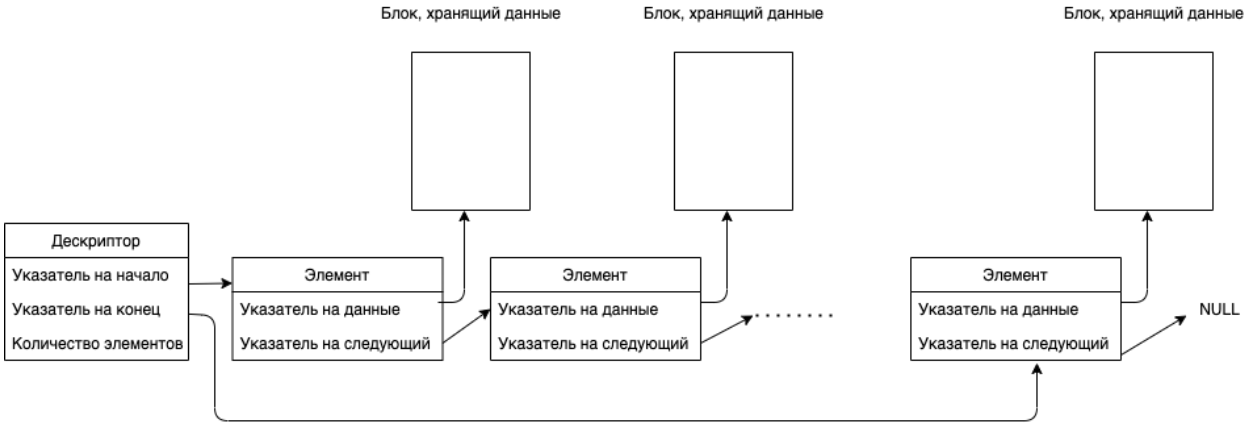
Абстрактный уровень представления СД:

1. Характер организованности – последовательность;

2. Изменчивость – динамическая СД;
3. Набор допустимых операций: инициализация, включение элемента, исключение элемента, чтение элемента, проверка пустоты очереди, уничтожение очереди.

Физический уровень представления СД:

1. Схема хранения: последовательная или связная.
2. Объем памяти, занимаемый экземпляром СД: в зависимости от реализации.
3. Формат внутреннего представления СД(ОЛС) и способ его интерпретации:



Объем занимаемой памяти: $N * M + 16$, где N – размер внутреннего массива, M – размер базового типа, 16 – размер дескриптора (3 числа + 1 указатель).

4. Характеристика допустимых значений: в зависимости от реализации
5. Тип доступа к элементам: в зависимости от реализации

Логический уровень представления СД:

Способ описания СД (На массиве, 4 поля) и экземпляра СД на языке С:
Дескриптор очереди состоит из 4-х полей:
1 – Внутренний массива для элементов;
2 – Индекс головы очереди;
3 – Индекс хвоста очереди;
4 – Количество элементов в очереди.

Способ описания СД:

```
typedef struct Fifo {
basetype_queue buf[FifoSize]; //внутренний массив
элементов
unsigned Uk1; // Голова очереди
unsigned Uk2; // Хвост очереди
unsigned N // Размер очереди
```

```
} Fifo;
```

Способ описания экземпляра: `Fifo f;`

Задание 2:

Файл format.h

```
//
// Created by настя буйвало on 24/10/2021.
//

#ifndef AISD6_FORMAT_H
#define AISD6_FORMAT_H

typedef struct task{
    char Name[10]; // имя запроса
    unsigned TimeStart; // время начала обслуживания
    unsigned Time; //время обслуживания
    char T; // тип задачи: 1 – T1, 2 – T2, 3 – T3 };
};

#endif //AISD6_FORMAT_H
```

Файл List2.h

```
//
// Created by настя буйвало on 19/10/2021.
//

#ifndef AISD6___LIST2_H_H
#define AISD6___LIST2_H_H

#include "format.h"
//const int ListOk = 0;
//const int ListNotMem = 1;
//const int ListUnder = 2;
//const int ListEnd = 3;

typedef struct task BaseType;

struct element {
    BaseType data;
    struct element *next;
};
typedef struct element *ptrel;

typedef struct List {
    ptrel Start;
    ptrel ptr;
    unsigned int N;
};
```

```
extern int ListError;
void InitList(struct List *L);
void PutList(struct List *L, BaseType E);
void DelFirst(struct List *L);
void GetList(struct List *L, BaseType* E);
void ReadList(struct List *L,BaseType *E);
int FullList(struct List *L);
unsigned int Count(struct List *L);
void BeginPtr(struct List *L);

#endif //AISD6___LIST2_H_H
```

Файл List2.c

```
//
// Created by настя буйвало on 19/10/2021.
//
#include "__LIST2_H.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>


//ListError = ListOk;

//инициализация
void InitList(struct List *L)
{
    ptrel fict = (struct element*)malloc(sizeof(struct element));
    L->Start = fict;
    L->ptr = fict;
    L->N = 0;
}

//включение элемента
void PutList(struct List *L, BaseType E)
{
    struct element* a = (struct element*)malloc(sizeof(struct element));
    a->data.T = E.T;
    a->data.Time = E.Time;
    a->data.TimeStart = E.TimeStart;
    strcpy(a->data.Name, E.Name);

    a->next = NULL;
    L->ptr->next = a;
    L->ptr = a;
    L->N++;

}

void Copy(struct List *L,BaseType *E)
{
    E->TimeStart = L->Start->next->data.TimeStart;
    E->Time = L->Start->next->data.Time;
    E->T = L->Start->next->data.T;
}

//чтение первого элемента
void ReadList(struct List *L,BaseType *E)
{
```

```
    if(L->N) {
        Copy(L, E);
        strcpy(E->Name, L->Start->next->data.Name);
    }
    else
        ;//ListError = ListUnder;
}

//удаление первого
void DelFirst(struct List *L)
{
    if(L->N) {
        ptrel t = L->Start->next;
        ptrel t2 = L->Start->next->next;
        free(t);
        L->Start->next = t2;
        L->N--;
        if(L->N == 0) {
            L->ptr = L->Start;
        }
    }
    else
        ;//ListError = ListUnder;
}

void GetList(struct List *L, BaseType* E) {
    if (L->N) {
        ReadList(L, E);
        DelFirst(L);
    } else
        ;//ListError = ListUnder;
}

//проверка: свободен ли список.
int FullList(struct List *L)
{
    return(L->N == 0);
}

//возвращает количество элементов в списке.
unsigned Count(struct List *L)
{
    return(L->N);
}

//устанановка в начало списка.
void BeginPtr(struct List *L)
{
    L->ptr = L->Start;
}
```

Файл FIFO7.h

```
//
// Created by настя буйвало on 19/10/2021.
//

#ifndef AISD6___FIFO7_H_H
#define AISD6___FIFO7_H_H
#include "__LIST2_H.h"
```

```
//const int FifoOk = ListOk;
//const int FifoUnder = ListUnder;
//const int FifoOver = ListNotMem;

extern int FifoError; // Переменная ошибок

typedef struct List Fifo;

void InitFifo(Fifo *f); // Инициализация очереди
void PutFifo(Fifo *f, BaseType E); /* Поместить элемент в очередь */
void GetFifo(Fifo *f, BaseType *E); /* Извлечь элемент из очереди */
void ReadFifo(Fifo *f, BaseType *E); // Прочитать элемент
int EmptyFifo(Fifo *f); // Проверка, пуста ли очередь?
void DoneFifo(Fifo *f); // Разрушить очередь

#endif //AISD6___FIFO7_H_H
```

Файл FIFO7.c

```
#include "__FIFO7_H.h"
#include <stdlib.h>

//FifoError = ListOk;

void PutFifo(Fifo *f, BaseType E) /* Поместить элемент в очередь */
{
    PutList(f, E);
}

void InitFifo(Fifo *f)// Инициализация очереди
{
    InitList(f);
}

void GetFifo(Fifo *f, BaseType *E) /* Извлечь элемент из очереди */
{
    GetList(f, E);
}

void ReadFifo(Fifo *f, BaseType *E) // Прочитать элемент
{
    ReadList(f, E);
}

int EmptyFifo(Fifo *f) // Проверка, пуста ли очередь?
{
    return FullList(f);
}

void DoneFifo(Fifo *f) // Разрушить очередь
{
    BaseType t;
    while(EmptyFifo(f) != 0){
        GetFifo(f, &t);
    }
    free(f->Start->next);
    free(f->Start);
    free(f->ptr);
}
```

Файл List3.h

```
////
//// Created by настя буйвало on 19/10/2021.
////
//
#ifndef AISD6___LIST3_H_H
#define AISD6___LIST3_H_H
#include <stdlib.h>
#include <string.h>
#include "format.h"

//
//const ListOk = 0;
//const  ListNotMem = 1;
//const  ListUnder = 2;
//const ListEnd = 3;

typedef void* BaseType2;
typedef struct element2 *ptrel2;
typedef struct element2 {
    BaseType2 data;
    ptrel2 next;
};

typedef struct List2 {
    ptrel2 Start;
    ptrel2 ptr;
    unsigned int N;//размер списка
    unsigned int size;//размер информационной части элемента
};

//extern int ListError;
void InitList2(struct List2 *L);
void PutList2(struct List2 *L, BaseType2 E);
void GetList2(struct List2 *L, BaseType2 E);
void ReadList2(struct List2 *L,BaseType2 E);
int FullList2(struct List2 *L);
//int EndList2(struct List2 *L);
//unsigned int Count2(struct List2 *L);
void BeginPtr2(struct List2 *L);
//void EndPtr2(struct List2 *L);
//void MovePtr2(struct List2 *L);
//void MoveTo2(struct List2 *L, unsigned int n);
//void DoneList2(struct List2 *L);
//void CopyList2(struct List2 *L1,struct List2 *L2);
void Copy2(void *a, void *b, size_t size);


#endif //AISD6___LIST3_H_H
```

Файл List3.c

```
//
// Created by настя буйвало on 19/10/2021.
//

#include "__LIST3_H.h"
```

```
#include <stdio.h>

//инициализация
void InitList2(struct List2 *L)
{
    ptrel2 fict = (struct element*)malloc(sizeof(struct element2));
    L->Start = fict;
    L->ptr = fict;
    L->N = 0;
    L->size = sizeof(struct task);
}

void Copy2(void *a, void *b, size_t size)
{
    char *r = a;
    char *w = b;
    for(int i = 0; i < size; i++)
        *w++ = *r++;
}

//включение элемента
void PutList2(struct List2 *L, BaseType2 E) {

    ptrel2 a = (struct element *) malloc(sizeof(struct element2));
    a->data = (struct task *) malloc(sizeof(struct task));
    Copy2(E, a->data, L->size);

    a->next = L->ptr;
    L->ptr = a;
    L->N++;
}

//чтение первого элемента
void ReadList2(struct List2 *L,BaseType2 E)
{
    if(L->N) {
        Copy2(L->ptr->data, E, L->size);
    }
    else;
       // ListError = ListUnder;
}

//удаление последнего
void DelLast2(struct List2 *L)
{
    if(L->N) {
        ptrel2 t = L->ptr;
        L->ptr = L->ptr->next;
        free(t);
        L->N--;
    }
    else;
        //ListError = ListUnder;
}

void GetList2(struct List2 *L, BaseType2 E) {
    if (L->N) {
        ReadList2(L, E);
        DelLast2(L);
    } else;
       // ListError = ListUnder;
}
```

```
//проверка: свободен ли список.
int FullList2(struct List2 *L)
{
    return(L->N == 0);
}

//возвращает количество элементов в списке.
unsigned Count2(struct List2 *L)
{
    return(L->N);
}

//устанановка в начало списка.
void BeginPtr2(struct List2 *L)
{
    L->ptr = L->Start;
}
```

Файл Stack7.h

```
////
//// Created by настя буйвало on 19/10/2021.
////

#ifndef AISD6___STACK7_H_H
#define AISD6___STACK7_H_H

#include "__LIST3_H.h"

//const StackOk = ListOk;
//const StackUnder = ListUnder;
//const StackOver = ListNotMem;
//int StackError; // Переменная ошибок
typedef struct List2 Stack;
void InitStack(Stack *s); /* Инициализация стека */
void PutStack(Stack *s, void *E); // Поместить элемент в стек
void GetStack(Stack *s, void *E); // Извлечь элемент из стека
int EmptyStack(Stack s); // Проверка: стек пуст?

#endif //AISD6___STACK7_H_H
```

Файл Stack7.c

```
//
// Created by настя буйвало on 19/10/2021.
//

#include "__STACK7_H.h"

void InitStack(Stack *s) /* Инициализация стека */
{
    InitList2(s);
}
```

```
void PutStack(Stack *s, void *E) // Поместить элемент в стек
{
    PutList2(s, E);
}

void GetStack(Stack *s, void *E) // Извлечь элемент из стека
{
    GetList2(s, E);
}

int EmptyStack(Stack s) // Проверка: стек пуст?
{
    return (FullList2(&s));
}
```

Файл main.c

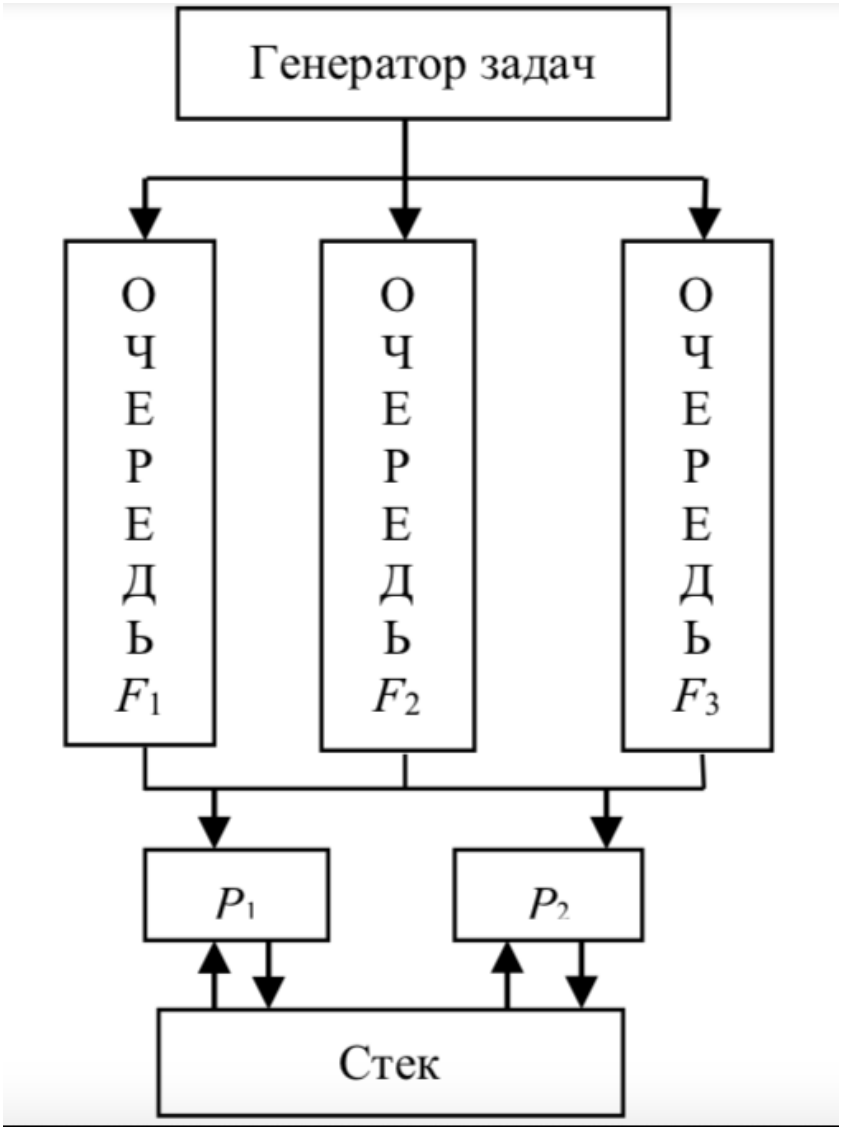
```
#include <stdio.h>
#include "__FIFO7_H.h"
#include "__STACK7_H.h"

/*
 typedef struct task{
        char Name[10]; // имя запроса
        unsigned TimeStart; // время начала обслуживания
        unsigned Time; //время обслуживания
        char T; // тип задачи: 1 – T1, 2 – T2, 3 – T3 };
};
 */

int max(int a, int b)
{
    if(a>b)
        return a;
    else
        return b;
}

int summ_time(struct task tasks[], int n)
{
    int summ = 0;
    for(int i = 0; i < n; i++)
        summ+= tasks[i].Time;
    return summ;
}


int main() {
    struct task tasks[] = {
            {"Task1", 0, 5, 1},
            {"Task2", 0, 2, 1},
            {"Task3", 1, 3, 3},
            {"Task4", 2, 5, 2},
            {"Task5", 3, 2, 1}
    };
    int nTasks = sizeof(tasks) / sizeof(struct task);

    Fifo * F1;
```

```
    F1 = (struct List*)malloc(sizeof(struct List));
    InitFifo(F1);
    Fifo * F2;
    F2 = (struct List*)malloc(sizeof(struct List));
    InitFifo(F2);
    Fifo * F3;
    F3 = (struct List*)malloc(sizeof(struct List));
    InitFifo(F3);

    Stack s;
    InitStack(&s);

    struct task proc1 = {"Processor1", 0, 0, 0};
    struct task proc2 = {"Processor2", 0, 0, 0};

    int i = 0;
    int j = 0;
    while(!EmptyStack(s) || i < nTasks || !EmptyFifo(F1) || !EmptyFifo(F2) ||
!EmptyFifo(F3) || proc1.Time != 0 || proc2.Time != 0){
        printf("Время %d :\n", i);
        while(tasks[j].TimeStart <= i && j < nTasks){ //распределение по очередям
            if(tasks[j].T == 1) {
                printf("Задача %s попала в очередь %d\n", tasks[j].Name, tasks[j].T);
                PutFifo(F1, tasks[j]);
            }
            if(tasks[j].T == 2) {
                printf("Задача %s попала в очередь %d\n", tasks[j].Name, tasks[j].T);
                PutFifo(F2, tasks[j]);
            }
            if (tasks[j].T == 3) {
                printf("Задача %s попала в очередь %d\n", tasks[j].Name, tasks[j].T);
                PutFifo(F3, tasks[j]);
            }
            j++;
        }

        if((proc1.Time == 0 || proc1.T == 3 && proc2.Time == 0 && EmptyFifo(F2)) &&
!EmptyFifo(F1)) {//задача типа 1 в 1 процессор
            if(proc1.Time == 0) {
                GetFifo(F1, &proc1);
                printf("Задача %s попала в 1 процессор\n", proc1.Name);
            }
            else{
//перенос из 1 процессора во второй задачи 3 типа
                Copy2(&proc1, &proc2, sizeof(struct task));
//задача 1 типа в 1 процессор
                printf("Задача из 1 процессора перенесена во 2 процессор\n");
                GetFifo(F1, &proc1);
                printf("Задача %s попала в 1 процессор\n", proc1.Name);
            }
        }
        else if(proc1.T == 3 && (proc2.Time != 0 || !EmptyFifo(F1)) &&
!EmptyFifo(F1)) { //перенос задачи типа 3 из 1 процессора в стек
            PutStack(&s, &proc1);
            printf("Задача из 1 процессора перенесена в стек\n");
            GetFifo(F1, &proc1);
            printf("Задача %s попала в 1 процессор\n", proc1.Name);
        }


        if((proc2.Time == 0 || proc2.T == 3 && proc1.Time == 0 && EmptyFifo(F1)) &&
!EmptyFifo(F2)) {//задача типа 2 во 2 процессор
```

```
            if(proc2.Time == 0) {
                GetFifo(F2, &proc2);
                printf("Задача %s попала во 2 процессор\n", proc2.Name);
            }
            else{
                Copy2(&proc2, &proc1, sizeof(struct task));    //перенос из 2
процессора в 1 задачи 3 типа
                printf("Задача из 2 процессора перенесена в 1 процессор\n"); //задача
типа 2 во 2 процессор
                GetFifo(F2, &proc2);
                printf("Задача %s попала во 2 процессор\n", proc2.Name);
            }
        }
        else if(proc2.T == 3 && (proc1.Time != 0 || !EmptyFifo(F2)) &&
!EmptyFifo(F2)) {//перенос задачи типа 3 из 2 процессора в стек
            PutStack(&s, &proc2);
            printf("Задача из 2 процессора перенесена в стек\n");
            GetFifo(F2, &proc2);
            printf("Задача %s попала во 2 процессор\n", proc2.Name);
        }

        if(!EmptyFifo(F3) && (proc1.Time == 0 && EmptyFifo(F1) || proc2.Time == 0 &&
EmptyFifo(F2))){//перенос задачи 3 типа из очереди 3 в процессоры
            if(proc1.Time == 0) {
                GetFifo(F3, &proc1);
                printf("Задача %s типа 3 попала в 1 процессор\n", proc1.Name);
            }
            else {
                GetFifo(F3, &proc2);
                printf("Задача %s типа 3 попала во 2 процессор\n", proc2.Name);
            }
        }

        if(!EmptyStack(s) && (proc1.Time == 0 && EmptyFifo(F1) || proc2.Time == 0 &&
EmptyFifo(F2))){//перенос задачи 3 типа из стека в процессоры
            if(proc1.Time == 0) {
                GetStack(&s, &proc1);
                printf("Задача %s пернесена из стека в 1 процессор\n", proc1.Name);
            }
            else {
                GetStack(&s, &proc2);
                printf("Задача %s пернесена из стека во 2 процессор\n", proc2.Name);
            }
        }

        if(proc1.Time == 1)
            printf("законечно выполнение задачи %s\n", proc1.Name);
        if(proc2.Time == 1)
            printf("законечно выполнение задачи %s\n", proc2.Name);

        proc1.Time = max(proc1.Time - 1, 0);
        proc2.Time = max(proc2.Time - 1, 0);
        i++;
    }
    return 0;
}
```

Результаты работы программы:

Как выглядит система:



Результаты работы программы:

```
struct task tasks[] = {
        { .Name: "Task1", .TimeStart: 0, .Time: 5, .T: 1},
        { .Name: "Task2", .TimeStart: 0, .Time: 2, .T: 1},
        { .Name: "Task3", .TimeStart: 1, .Time: 3, .T: 3},
        { .Name: "Task4", .TimeStart: 2, .Time: 5, .T: 2},
        { .Name: "Task5", .TimeStart: 3, .Time: 2, .T: 1}
};
```

```
Время 0 :
Задача Task1 попала в очередь 1
Задача Task2 попала в очередь 1
Задача Task1 попала в 1 процессор
Время 1 :
Задача Task3 попала в очередь 3
Задача Task3 типа 3 попала во 2 процессор
Время 2 :
Задача Task4 попала в очередь 2
Задача из 2 процессора перенесена в стек
Задача Task4 попала во 2 процессор
Время 3 :
Задача Task5 попала в очередь 1
Время 4 :
законечно выполнение задачи Task1
Время 5 :
Задача Task2 попала в 1 процессор
Время 6 :
законечно выполнение задачи Task2
законечно выполнение задачи Task4
Время 7 :
Задача Task5 попала в 1 процессор
Задача Task3 пернесена из стека во 2 процессор
Время 8 :
законечно выполнение задачи Task5
законечно выполнение задачи Task3
```

```
struct task tasks[] = {
        { .Name: "Task1",  .TimeStart: 0,  .Time: 5,  .T: 1},
        { .Name: "Task2",  .TimeStart: 0,  .Time: 2,  .T: 1},
        { .Name: "Task3",  .TimeStart: 1,  .Time: 3,  .T: 2},
        { .Name: "Task4",  .TimeStart: 2,  .Time: 5,  .T: 2},
        { .Name: "Task5",  .TimeStart: 3,  .Time: 2,  .T: 1}
};
```

```
/Users/nastabujvalo/CLionProjects/AiSD
Время 0 :
Задача Task1 попала в очередь 1
Задача Task2 попала в очередь 1
Задача Task1 попала в 1 процессор
Время 1 :
Задача Task3 попала в очередь 2
Задача Task3 попала во 2 процессор
Время 2 :
Задача Task4 попала в очередь 2
Время 3 :
Задача Task5 попала в очередь 1
законечно выполнение задачи Task3
Время 4 :
Задача Task4 попала во 2 процессор
законечно выполнение задачи Task1
Время 5 :
Задача Task2 попала в 1 процессор
Время 6 :
законечно выполнение задачи Task2
Время 7 :
Задача Task5 попала в 1 процессор
Время 8 :
законечно выполнение задачи Task5
законечно выполнение задачи Task4

Process finished with exit code 0
```

Вывод:

Во время выполнения лабораторной работы были реализованы различные представления ОЛС, написаны функции для работы с ними, эти ОЛС использованы при реализации СД стек и очередь, а также смоделирована система по выполнению задач согласно условию варианта.