УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

**“ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО”**

Кафедра «Информационные технологии»

Лабораторная работа №6

**«Программирование планировщика процессов»**

Выполнил: студент гр. ИТП-11

Козлов О.В.

Принял: преподаватель

Карась О. В.

Гомель 2022

**Цель**: разработать программу, осуществляющую моделирование режима работы с разделение времени.

**Ход работы**

В соответствии с вариантом, разработать программу на языке С# (или другом), осуществляющую моделирование работы планировщика процессов. Моделирование режима разделения времени выполняется в соответствии с заданием (таблица 2.1). На экран выводится следующая информация:

1.Номер текущего кванта времени процессора;

2. Таблица процессов с указанием имени процессов, продолжительности, приоритета (в зависимости от задания), оставшегося времени выполнения, время появления;

3. Таблица планирования процессов с отображением текущего состояния процессов.

После запуска, программа должна диалоговом режиме, ввести информацию процессах – имя, длительность, приоритет, время появления. Для алгоритмов RR число квантов времени. Выполнение должно производиться в пошаговом режиме (по нажатию на кнопку). По окончанию работы процесса на экране должно выводится сообщение о его завершении «Процесс такой-то завершен».

В соответствии с вариантом надо сделать SJF не вытесняющий приоритетный, таблица процессов приведена ниже:

|  |  |  |  |
| --- | --- | --- | --- |
| Процесс | Продолжительность | Время появления в очереди | Приоритет |
| P0 | 9 | 3 | 3 |
| P1 | 3 | 0 | 2 |
| P2 | 4 | 0 | 2 |
| P3 | 4 | 4 | 3 |

Текст программы, написанной на языке *JavaScript*, представлен в листинге **А**. Если у процессов одинаковый приоритет, то первым выполнится тот процесс, время выполнения которого меньше. Если процесс выполняется, его состояние — “E”; ожидает своей очереди — “W”. В результате мы увидим следующее. Согласно программе, нам необходимо отображать, когда процесс начался и когда закончился, и при выполнении мы увидим следующее:

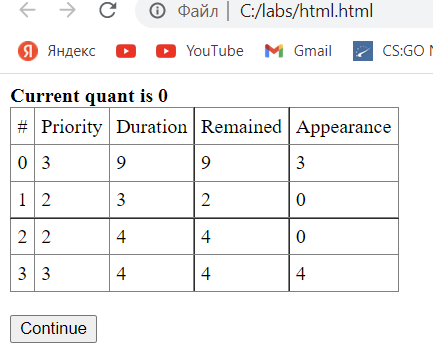


Рисунок 1 — Пример состояния на

текущий квант времени

После выполнения программы окончательно увидим следующее:

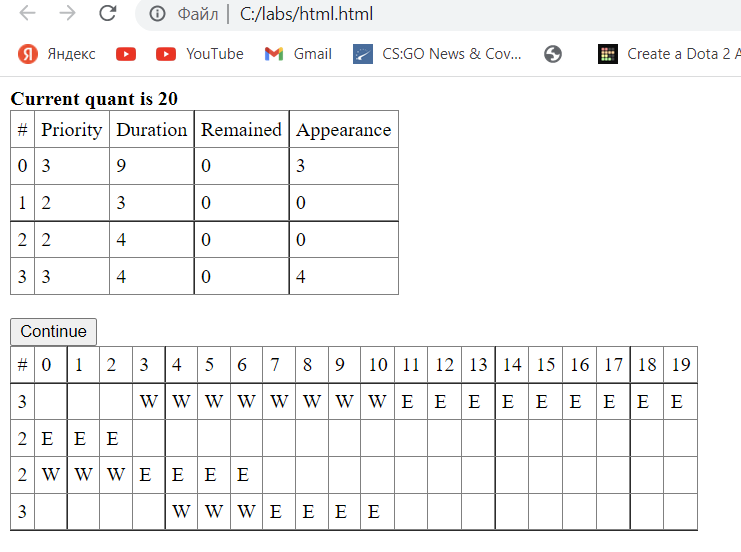


Рисунок 2 — Результат выполнения программы

**Вывод**: в ходе работы была разработана программа, осуществляющая моделирование режима работы с разделение времени.

**ЛИСТИНГ А**

**КОД ПРОГРАММЫ, МОДЕЛИРУЮЩЕЙ**

**РАБОТУ ПЛАНИРОВЩИК ЗАДАЧ**

"use strict";

let div = document.getElementById("tableDiv");

let btn = document.getElementById("btn");

let process = [];

let processN = +prompt("Enter amount of processess: ");

for (let i = 0; i < processN; i++)

{

    let priority = +prompt("Enter priority of process: ");

    let duration = +prompt("Enter duration of process: ");

    let appearance = +prompt("Enter appearance of process: ");

    process.push({priority, duration, remained: duration, appearance, isCompleted: false, states: []});

}

let table = [];

let statesN = [];

let wAmount = 0, eAmount = 0;

let isPrEx = false, f = false;

let numOfExPr = 0;

let q = 0;

btn.addEventListener("click", (e) =>

{

    if (process.some(el => !el.isCompleted))

    {

        f = false;

        if (!isPrEx)

        {

            let processQ = findQprocess(process);

            if (processQ.length)

            {

                numOfExPr = findMaxProcess(processQ);

                isPrEx = true;

                process = putStates(process, processQ, numOfExPr);

                process[numOfExPr].remained--;

            }

            else

                process = putStates(process, processQ, numOfExPr)

        }

        else if (isPrEx && process[numOfExPr].remained)

        {

            let processQ = findQprocess(process);

            process = putStates(process, processQ, numOfExPr);

            process[numOfExPr].remained--;

        }

        else if (isPrEx && !process[numOfExPr].remained)

        {

            process[numOfExPr].isCompleted = true;

            console.log(`Process with priority ${process[numOfExPr].priority} is over!\n`);

            isPrEx = false;

            f = true;

        }

        div.innerHTML = "";

        showQuantTable(div, process, q);

        if (!f) {

            if (isPrEx)

                console.log(`Process with priority ${process[numOfExPr].priority} is executing!`);

            statesN.push(q++);

        }

    } else

        showTable();

});

btn.click();

function findMaxProcess(process) {

    return process.sort((el1, el2) =>

            {

                if (el1.pr.priority == el2.pr.priority)

                    if (el1.pr.duration < el2.pr.duration)

                        return -1;

                    else

                        return 1;

                else if (el1.pr.priority < el2.pr.priority)

                    return -1;

                else

                    return 1;

            })[0].i;

}

function findQprocess(process) {

    let arr = [];

    process.forEach((pr, i) => pr.appearance <= q && !pr.isCompleted && arr.push({pr, i}));

    return arr;

}

function putStates(process, processQ, numOfExPr)

{

    process.forEach((el, i) =>

    {

        if (i != numOfExPr && !processQ.some(e => e.i == i))

            el.states.push(" ");

        else if (i != numOfExPr && processQ.some(e => e.i != i))

            el.states.push("W");

        else if (i == numOfExPr && !el.isCompleted)

            el.states.push("E");

    });

    return process;

}

function showQuantTable(el, process, q)

{

    let str1 = ["#", 'Priority', 'Duration', 'Remained', 'Appearance'];

    let b = document.createElement("b");

    b.innerHTML = `Current quant is ${q}`;

    el.append(b);

    el.append(document.createElement("br"));

    let table = document.createElement("table");

    let thead = document.createElement("thead");

    let tbody = document.createElement("tbody");

    table.style.borderCollapse = "collapse";

    table.setAttribute("border", "1");

    str1.forEach((el) => {

        let td = document.createElement("td");

        td.innerHTML = el;

        thead.append(td);

    })

    process.forEach((el, ind) => {

        let tr = document.createElement("tr");

        let td = document.createElement("td");

        td.innerHTML = ind;

        tr.append(td);

        Object.values(el).slice(0, 4).forEach((propName) =>

        {

            let td = document.createElement("td");

            td.innerHTML = propName;

            tr.appendChild(td);

        });

        tbody.append(tr);

    })

    table.append(tbody);

    table.append(thead);

    el.append(table);

}

function showTable()

{

    table.push(["#", ...statesN]);

    let t = document.createElement("table");

    let thead = document.createElement("thead");

    let tbody = document.createElement("tbody");

    t.style.borderCollapse = "collapse";

    t.setAttribute("border", "1");

    let tr = document.createElement("tr");

    table[0].forEach((el) => {

        let td = document.createElement("td");

        td.innerHTML = el;

        tr.append(td);

    })

    thead.append(tr);

    t.append(thead);

    for (let i = 0; i < process.length; i++)

    {

        let tr = document.createElement("tr");

        let td = document.createElement("td");

        td.innerHTML = process[i].priority;

        tr.append(td);

        process[i].states.forEach((state) => {

            td = document.createElement("td");

            td.innerHTML = state;

            tr.append(td);

        });

        tbody.append(tr);

    }

    t.append(tbody);

    document.body.appendChild(t);

}