НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ
Кафедра системного програмування та
спеціалізованих комп‘ютерних систем

Лабораторна робота №1

З дисципліни: «Інженерія програмного забезпечення»

«Планування процесів»

Варіант 14

Студент КВ-42: Савицький Т.П.
Перевірив(ла):
_____
_____

Київ-2017

Завдання

1. Написати програму, що моделює процедуру обслуговування процесів, які знаходяться у черзі готових до виконання, за певним алгоритмом планування

Програма повинна обчислювати:

- час початку виконання для кожного процесу;
- час завершення для кожного процесу;
- час затримки (очікування) в черзі для кожного процесу;
- повний час виконання (з урахуванням затримки) для кожного процесу;
- середній час очікування процесів у черзі;
- середній повний час виконання процесів (з урахуванням затримки);
- інші результати, необхідні для оцінки алгоритму планування.

2. Зробити візуалізацію роботи програми у часі (на різних наборах вхідних даних, що ілюструють особливості алгоритму), а також представити кінцеві результати роботи на екрані у вигляді таблиці.

3. Проаналізувати та пояснити отримані результати. Надати висновки щодо алгоритму планування за результатами роботи.

Завдання

Приорітетне планування з підвищенням приорітетів (t кванту та t порогове задати самостійно):

1) черга 1 – алгоритм **RR**
2) черга 2 – алгоритм **RR**
3) черга 3 – алгоритм **FCFS**

Висновки

Із двох заданих алгоритмів краще працює FCFS тому що, він одразу повністю завершує задачу і переходить до іншої. Таким чином, час на затримку досить маленький, що дає, в свою чергу, швидкодію, вівдмінно від алгоритму RR. Алгоритм RR виконує задачі за певною довжиною такту і переходить до наступної задачі, тобто залишає іншу задачу або виконаною, або - ні. Багато часу забирає цей алгоритм на переключення однієї задачі на іншу. І навіть коли задача виконана, то вона все одно перевіряється на зробленість.

Час затримки - невеликий, бо швидкодія комп'ютера - дуже висока. Перед початком виконання задач, вони сортуються по пріоритетності, що дозволяє краще зберегти ресурси комп'ютера.

Текст програми

```
//
// appController.js
//

import Observer from './common/observer.js';
import Variables from './common/variables.js'

import ModelController from './modelController.js';
import ViewController from './viewController.js';

export class AppController {

        constructor() {
  this.model = new ModelController();
```

```
    this.view = new ViewController(this.model);

                        Observer.addListener(Variables.isFreeTaskFIFO, this.isFreeProcessFIFO.bind(this));
                        Observer.addListener(Variables.isFreeTaskRR1, this.isFreeProcessRR1.bind(this));
                        Observer.addListener(Variables.isFreeTaskRR2, this.isFreeProcessRR2.bind(this));

                        this.model.startFIFO();
                        this.model.startRR1();
                        this.model.startRR2();
  }

            isFreeProcessFIFO(task) {
                        this.view.taskIsDone(task, 'body-table-fifo');
            }

            isFreeProcessRR1(task) {
                        this.view.taskIsDone(task, 'body-table-rr-1');
            }

            isFreeProcessRR2(task) {
                        this.view.taskIsDone(task, 'body-table-rr-2');
            }
}

//
// modelController.js
//


import Variables from './common/variables.js';
import Observer from './common/observer.js';

import Process from './Model/process.js';
import Task from './Model/task.js';

export default class ModelController {

  constructor() {
    this.time = 1000;
    this.processor = new Process('Single Process');
  }

  startFIFO() {
                        this.carriageFIFO([
                                    new Task('1', 200, 3),
                                    new Task('2', 150, 1),
                                    new Task('3', 300, 9),
                                    new Task('4', 100, 1),
                                    new Task('5', 50,  2),
                                    new Task('6', 250, 5),
                                    new Task('7', 100, 4),
                                    new Task('8', 200, 8),
                                    new Task('9', 500, 6),
                        ], this.processor);
            }

            startRR1() {
                        this.carriageRR([
                                    new Task('1', 200, 3),
                                    new Task('2', 150, 1),
                                    new Task('3', 300, 9),
                                    new Task('4', 100, 1),
                                    new Task('5', 50,  2),
                                    new Task('6', 250, 5),
                                    new Task('7', 100, 4),
                                    new Task('8', 200, 8),
                                    new Task('9', 500, 6),
                        ], this.processor, Variables.isFreeTaskRR1);
            }

            startRR2() {
                        this.carriageRR([
                                    new Task('1', 100, 8),
                                    new Task('2', 50,  2),
                                    new Task('3', 400, 7),
                                    new Task('4', 250, 9),
                                    new Task('5', 500, 3),
                                    new Task('6', 350, 1),
                                    new Task('7', 50,  5),
                                    new Task('8', 25,  4),
                                    new Task('9', 525, 6),
```

```
                ], this.processor, Variables.isFreeTaskRR2);
        }


        carriageFIFO(tasks, process) {
  tasks.sort(ModelController.comparePriory);

                for (let task of tasks) {
                        task.beginTime = performance.now();
                        process.makingFIFO(task);
                        tasks.done = true;
                }

                process.endTime = performance.now() - process.beginTime;
                Observer.emit(Variables.isFreeTaskFIFO, tasks);
 }


 carriageRR(tasks, process, nameTask, time = 25) {
                tasks.sort(ModelController.comparePriory);

                let counterIsDone = 0;

                while (counterIsDone < tasks.length) {
                        const task = tasks.shift();

                        if (!task.done) {
                                if (!task.beginTime) {
                                        task.beginTime = performance.now();
                                }

                                process.makingRR(task, time);

                                if (task.time <= task.durationTime) {
                                        task.done = true;
                                        counterIsDone++;
                                }
                        }

                        tasks.push(task);
                }

                process.endTime = performance.now() - process.beginTime;
                Observer.emit(nameTask, tasks);
        }

  static comparePriory(a, b) {
    if (a.priority < b.priority)
      return -1;
    if (a.priority > b.priority)
      return 1;
    return 0;
  }
}


//
// viewController.js
//

export default class ViewController {

  constructor(model) {
        this.model = model;
  }

  taskIsDone(tasks, idName) {
                this.counter = 1;

                for (let task of tasks) {
                        const dom = document.getElementById(idName);
                        dom.append(this._createRow(task));
                        this.counter++;
                }
        }

  _createRow(task) {
        const dom = document.createElement('tr');
                dom.innerHTML = `<td>${this.counter}</td>
                                        <td>${task.beginTime.toFixed(3)}</td>
                                        <td>${task.time}</td>
```

```
                                        <td>${task.startTime.toFixed(3)}</td>
                                        <td>${task.finishTime.toFixed(3)}</td>
                                        <td>${task.delayTime.toFixed(3)}</td>
                                        <td>${task.totalTime.toFixed(3)}</td>
                                        <td>${task.priority}</td>
                                        <td>${task.title}</td>`;
                        return dom;
  }

}

//
// process.js
//

export default class Process {

  constructor(title, performance = 100) {
    this.title = title;
    this.performance = performance;
    this._isBusy = false;
  }

  get isBusy() {
    return this._isBusy;
  }

  makingFIFO(task) {
    this._isBusy = true;
                        this._delay(task);
                        this._isBusy = false;
  }

  makingRR(task, time) {
                        this._isBusy = true;
                        this._delay(task, time);
                        task.durationTime += time;
                        this._isBusy = false;
            }

            _delay(task, time) {
            task.runningTime = time || task.time;
                        task.startTime = performance.now();
            const finishTime = performance.now() + task.runningTime;

                        while (performance.now() <= finishTime) {}

                        task.finishTime = performance.now();
                        task.delayTime = Math.abs(task.runningTime - (task.finishTime - task.beginTime));
                        task.totalTime = task.finishTime - task.beginTime;
            }

}


//
// task.js
//

export default class Task {
  constructor(title, time, priority) {
    this.title = title;
    this.time = time;
    this.priority = priority;
    this.done = false;
    this.durationTime = 0;
  }
}
```

Тести

FIFO

#	t begin	t running	t start	t finish	t delay	t total	priory	title
1	1554.530	150	1554.540	1704.555	0.025	150.025	1	2
2	1704.590	100	1706.615	1806.625	2.035	102.035	1	4
3	1806.645	50	1806.650	1856.655	0.010	50.010	2	5
4	1856.660	200	1856.665	2056.675	0.015	200.015	3	1
5	2056.685	100	2056.685	2156.690	0.005	100.005	4	7
6	2156.700	250	2156.705	2406.710	0.010	250.010	5	6
7	2406.720	500	2406.725	2906.735	0.015	500.015	6	9
8	2906.750	200	2906.750	3106.750	0.000	200.000	8	8
9	3106.760	300	3106.760	3406.765	0.005	300.005	9	3

RR

#	t begin	t running	t start	t finish	t delay	t total	priory	title
1	3600.970	200	4782.120	4807.125	1181.155	1206.155	8	8
2	3625.990	300	5057.290	5082.290	1431.300	1456.300	9	3
3	3407.415	150	4431.810	4456.815	1024.400	1049.400	1	2
4	3432.445	100	4106.595	4131.600	674.155	699.155	1	4
5	3457.500	50	3701.085	3726.090	243.590	268.590	2	5
6	3482.515	200	4707.065	4732.075	1224.560	1249.560	3	1
7	3525.915	100	4156.620	4181.630	630.715	655.715	4	7
8	3550.935	250	4907.200	4932.205	1356.270	1381.270	5	6
9	3575.950	500	5257.400	5282.405	1681.455	1706.455	6	9

RR

#	t begin	t running	t start	t finish	t delay	t total	priory	title
1	5433.010	400	7284.290	7309.290	1851.280	1876.280	7	3
2	5458.020	100	5958.425	5983.435	500.415	525.415	8	1
3	5483.035	250	6733.895	6758.900	1250.865	1275.865	9	4
4	5282.875	350	7059.125	7084.130	1776.255	1801.255	1	6
5	5307.890	50	5533.060	5558.160	225.270	250.270	2	2
6	5332.900	500	7459.350	7484.350	2126.450	2151.450	3	5
7	5357.930	25	5357.930	5382.940	0.010	25.010	4	8
8	5382.975	50	5583.180	5608.180	200.205	225.205	5	7
9	5407.995	525	7509.390	7534.390	2101.395	2126.395	6	9